[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, [Angyal Ádám Szilárd (CDEFO8), Mérnökinformatikus szak, alapképzés] szakos hallgató kijelentem, hogy a [Állatok detektálása felvételről mesterséges intelligencia segítségével] 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

[1 oldalas, magyar nyelvű tartalmi kivonat]

# Abstract

[Title in English]

[1 oldalas, angol nyelvű kivonat]

# Tartalomjegyzék

Tartalom

[Nyilatkozat 5](#_Toc134386809)

[Kivonat 6](#_Toc134386810)

[Abstract 7](#_Toc134386811)

[Tartalomjegyzék 8](#_Toc134386812)

[1.Bevezetés 10](#_Toc134386813)

[2.Neurális hálózat 11](#_Toc134386814)

[**2.1.Mesterséges Neurális hálózat** 12](#_Toc134386815)

[**2.1.1.Története** 13](#_Toc134386816)

[2.1.2.Neuronrétegek 14](#_Toc134386817)

[2.1.3.A neurális hálózatok felépítése 15](#_Toc134386818)

[2.1.4.Mélytanulás 19](#_Toc134386819)

[2.1.5.Tanítás 19](#_Toc134386820)

[**2.1.6.Implementációk** 22](#_Toc134386821)

[**3.Keretrendszerek** 23](#_Toc134386822)

[**3.1.OpenCV** 23](#_Toc134386823)

[**3.2.PyTorch** 25](#_Toc134386824)

[**3.3.OpenCV vagy PyTorch** 25](#_Toc134386825)

[**4.Mélytanulási modellek** 26](#_Toc134386826)

[**4.1.YOLO** 26](#_Toc134386827)

[**4.1.1.YOLO v2** 27](#_Toc134386828)

[**4.1.2.YOLO v3** 27](#_Toc134386829)

[**4.1.3.YOLO v4** 28](#_Toc134386830)

[**4.1.4.YOLO v5** 29](#_Toc134386831)

[**4.1.5.YOLO v6** 30](#_Toc134386832)

[**4.1.6.YOLO v7** 30](#_Toc134386833)

[**4.1.7.YOLO v8** 32](#_Toc134386834)

[**4.2.EfficientDet** 32](#_Toc134386835)

[**4.2.1.EfficientDet-Lite0** 33](#_Toc134386836)

[**4.2.2. EfficientDet-Lite2** 33](#_Toc134386837)

[**4.3.RetinaNet** 34](#_Toc134386838)

[4.4.Tensorflow 34](#_Toc134386839)

[4.5.Inception V3 model 36](#_Toc134386840)

[4.6.ImageAI 36](#_Toc134386841)

[**5.Adatkészlet** 37](#_Toc134386842)

[**6.Gyakorlati alkalmazás** 39](#_Toc134386843)

[6.1. MediaPipe Object Detector 39](#_Toc134386844)

[6.1.1.Jellemzők 39](#_Toc134386845)

[6.1.2.Modell 39](#_Toc134386846)

[6.1.3.Modell referenciaértékek 40](#_Toc134386847)

[6.1.4.Modellkövetelmények és metaadatok 40](#_Toc134386848)

[6.1.5.Metadata követelmények 41](#_Toc134386849)

[6.1.6.Project létrehozása 41](#_Toc134386850)

[6.1.7.Konfigurációs lehetőségek 42](#_Toc134386851)

[6.1.8.Adatok előkészítése 43](#_Toc134386852)

[6.1.9.A projekt futtatása 43](#_Toc134386853)

[6.1.10.Az eredmények kezelése és megjelenítése 44](#_Toc134386854)

[**6.2.ImageAI** 44](#_Toc134386855)

[6.2.1.Projekt létrehozása 45](#_Toc134386856)

[6.2.2.Modell betöltése 45](#_Toc134386857)

[6.2.3.Egyéni videoobjektum-érzékelés 46](#_Toc134386858)

[6.2.4.Videó elemzés 47](#_Toc134386859)

[6.2.5.Képkocka észlelési intervallumok 50](#_Toc134386860)

[**6.3.Implementing Real** 51](#_Toc134386861)

[6.3.1. A modell betöltése 51](#_Toc134386862)

[6.3.2.Egyetlen képkocka pontozása 52](#_Toc134386863)

[6.3.3.Dobozok kirajzolása a keret fölé 52](#_Toc134386864)

[**6.3.4.Összegzés** 53](#_Toc134386865)

[**7.Összehasonlítás** 55](#_Toc134386866)

[8.Irodalomjegyzék 56](#_Toc134386867)

[9.Mellékletek 57](#_Toc134386868)

# 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.

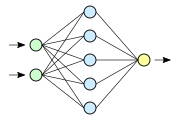
A szakdolgozatom témája a fentebb említettekre próbál néhány elterjedtebb példával (neurális hálózatot használó, python programozási nyelven íródott programok) bemutatni ezen problémakör megoldását. Ezen módszereket részletesen bemutatom és számszerűsítve is összehasonlítom (pl. hatékonyság, pontosság, erőforrás felhasználás, stb alapján), majd a kapott eredmények alapján egy levonok egy végső konklúziót.

….

# 2.Neurális hálózat

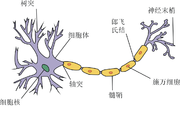
A neurális hálózat biológiai [neuronok](https://hu.wikipedia.org/wiki/Neuron) összekapcsolt csoportja. Modern használatban a szó alatt a mesterséges neurális hálót értjük, amelyek mesterséges neuronokból állnak. Így a neurális háló kifejezés két különböző koncepciót is jelent:

1. A [biológiai neurális hálózat](https://hu.wikipedia.org/w/index.php?title=Biol%C3%B3giai_neur%C3%A1lis_h%C3%A1l%C3%B3zat&action=edit&redlink=1) a gócok csatlakozása, vagy funkcionálisan összefüggő neuronok, a periférikus idegrendszerben, vagy a központi idegrendszerben. A neurális tudományok területén a leggyakrabban az idegrendszer egy csoportjának azonosítják, mely laboratóriumi analizálásra alkalmas.
2. A [mesterséges neurális hálózat](https://hu.wikipedia.org/wiki/Mesters%C3%A9ges_neur%C3%A1lis_h%C3%A1l%C3%B3zat), egy biológiai indíttatású gép/program, ami a biológiai neurális hálózat néhány tulajdonságát modellezi. Az alkalmazások többsége technikai jellegű, és nem kognitív modell. A mesterséges neurális hálók nem csak a biológia, hanem más tudományterületek (matematika, fizika, pszichológia) eredményeit is felhasználják.



1.ábra:Egy neurális Hálózat egyszerűsített nézete

A [természetes neuron-hálózatok](https://hu.wikipedia.org/w/index.php?title=Term%C3%A9szetes_neuron-h%C3%A1l%C3%B3zatok&action=edit&redlink=1) vagy [mesterséges neuron-hálózatok](https://hu.wikipedia.org/w/index.php?title=Mesters%C3%A9ges_neuron-h%C3%A1l%C3%B3zatok&action=edit&redlink=1) N-hálók származástól függetlenül a hálók működésének [mechanizmusa](https://hu.wikipedia.org/wiki/Mechanizmus) többé-kevésbé megegyezik: ezen hálózatok alapelve, hogy a számolásokat egymással összekapcsolt kis feldolgozóegységek, neuronok végzik. A számítások során fontos szerepet játszik a neuronok közötti kapcsolatrendszer, ezért a neurális hálókat konnekciós hálózatoknak, a velük foglalkozó szakembereket pedig konnekcionistáknak is nevezik.



2.ábra:Egy biológiai neuron vázlata

A neurális hálózat egyszerű egységekből áll, abban az értelemben, hogy belső állapotai leírhatók számokkal, ezek az aktivációs [értékek](https://hu.wikipedia.org/w/index.php?title=%C3%89rt%C3%A9k&action=edit&redlink=1). Mindegyik egység generál egy aktiválási [értéktől](https://hu.wikipedia.org/w/index.php?title=%C3%89rt%C3%A9k&action=edit&redlink=1) függő kimeneti értéket ([jelet](https://hu.wikipedia.org/wiki/Jel)). Az egységek csatlakoznak egymáshoz, mindegyik csatlakozás tartalmaz egy egyéni súlyt (szintén számokkal leírva, lásd [súlyozás](https://hu.wikipedia.org/w/index.php?title=S%C3%BAlyoz%C3%A1s&action=edit&redlink=1)). Minden egység kiküldi a kimeneti értékét az összes többi egységnek, amelyekkel kimenő kapcsolatban vannak. A "[rendszer](https://hu.wikipedia.org/wiki/Rendszer)" bemenetei lehetnek érzékszervek vagy mesterséges szenzorok, érzékelők adatai, míg kimenetei lehet a [viselkedés](https://hu.wikipedia.org/wiki/Viselked%C3%A9s), jel egy kimeneti [neuronon](https://hu.wikipedia.org/wiki/Neuron), esetleg bármilyen mesterségesen megjelenített *válasz* egy *kérdésre* (amik neuron-hálózatok esetében persze [mintázatok](https://hu.wikipedia.org/w/index.php?title=Mint%C3%A1zat&action=edit&redlink=1)). Ezen kapcsolatok miatt az egység kimenete hatással van a másik egység aktivációjára. A kapcsolat bemeneti oldalán álló egység fogadja az értékéket, és azok súlyozásával kiszámolja az aktivációs értékét (összeszorozza a bemeneti jelet a hozzá tartozó bemenet súlyával, és veszi ezek összegét) A kimenetet az aktivációtól függően az aktivációs függvény határozza meg (pl az egység kimenetet generál –„tüzel”- ha az aktivizáció egy határérték felett van). Hogy ezek a [jelek](https://hu.wikipedia.org/wiki/Jel) ([értékek](https://hu.wikipedia.org/w/index.php?title=%C3%89rt%C3%A9k&action=edit&redlink=1)) [elektrokémiai](https://hu.wikipedia.org/wiki/Elektrok%C3%A9mia), [elektromos](https://hu.wikipedia.org/wiki/Elektromos), netán [szimbolikus](https://hu.wikipedia.org/w/index.php?title=Szimbolikus&action=edit&redlink=1), ez a "megvalósítás" mikéntjétől ([biológiai](https://hu.wikipedia.org/wiki/Biol%C3%B3gia" \o "Biológia), [hardver](https://hu.wikipedia.org/wiki/Hardver), [szoftver](https://hu.wikipedia.org/wiki/Szoftver)) függ, de ez a működés alapelveit nem befolyásolja.

Fontos megjegyezni, hogy a neuronok bár számításokat végeznek ugyan, de mégsem processzorok. A fő különbség a kettő között az, hogy amíg a processzorokat programozzák (szekvenciális utasítássorozatot adnak meg neki), addig a neuronokat tanítják (a súlymátrix értékeinek beállításával). A hálózat tanulási technikája lehet ellenőrzött, ill. nem ellenőrzött típusú [tanulás](https://hu.wikipedia.org/w/index.php?title=Tanul%C3%A1s&action=edit&redlink=1). Az ellenőrzött tanulású N-hálók esetében a rendszer nagyszámú, előre megadott példa alapján tanul: speciális [algoritmusokkal](https://hu.wikipedia.org/wiki/Algoritmus) addig változtatja a [neuronok](https://hu.wikipedia.org/wiki/Neuron) közötti kapcsolatokat, míg a megadott [bemenetek](https://hu.wikipedia.org/w/index.php?title=Bemenetek&action=edit&redlink=1" \o "Bemenetek (a lap nem létezik)) mindig a megadott [kimeneteket](https://hu.wikipedia.org/w/index.php?title=Kimenet&action=edit&redlink=1) "okozzák". Ilyenkor a hálózat a legtöbb esetben a csatlakozások súlyának módosításával tanul. A súlymódosítás során az ún. [hibafüggvény](https://hu.wikipedia.org/wiki/Hibaf%C3%BCggv%C3%A9ny) eredményét veszi figyelembe. A [hibafüggvény](https://hu.wikipedia.org/wiki/Hibaf%C3%BCggv%C3%A9ny) értékét sokféle módon lehet kiszámítani, a legegyszerűbb eset, amikor a kimeneti értékből kivonja a helyes kimeneti értéket.

A nem ellenőrzött hálóknál leginkább a [Kohonen önszervező térképet](https://hu.wikipedia.org/w/index.php?title=Kohonen_%C3%B6nszervez%C5%91_t%C3%A9rk%C3%A9p&action=edit&redlink=1) használják, amely hálózat azon feltételezések alapján működik, hogy a [hálózat](https://hu.wikipedia.org/wiki/H%C3%A1l%C3%B3zat) képes a teljes bemeneti [mintakészlet](https://hu.wikipedia.org/w/index.php?title=Mintak%C3%A9szlet&action=edit&redlink=1) közös jellemzőinek azonosítására.

**2.1.Mesterséges Neurális hálózat**

A mesterséges neurális hálózat, mesterséges neuronháló vagy ANN (artificial neural network) biológiai ihletésű szimuláció. Fő alkalmazási területe a [gépi tanulás](https://hu.wikipedia.org/wiki/G%C3%A9pi_tanul%C3%A1s), melynek célja ezeknek a hálóknak a tanuló rendszerként történő gyakorlati alkalmazása. [Gráf](https://hu.wikipedia.org/wiki/Gr%C3%A1f) alapú modell, melyben rétegekbe rendezett mesterséges neuronok kommunikálnak egymással nemlineáris aktivációs függvényeken keresztül. A legalább három rétegbe rendezett, nemlineáris aktivációs függvényt használó neurális hálózat univerzális függvényapproximátor, a visszacsatolásokat, [rekurrenciát](https://hu.wikipedia.org/w/index.php?title=Rekurrencia&action=edit&redlink=1) tartalmazó architektúrák pedig [Turing-teljességgel](https://hu.wikipedia.org/w/index.php?title=Turing-teljes&action=edit&redlink=1) rendelkeznek, így képesek megoldani bármilyen problémát vagy futtatni bármilyen algoritmust, amit számítógép segítségével meg lehet oldani vagy futtatni lehet.

Tanításuk általában a [hiba-visszaterjesztéssel](https://hu.wikipedia.org/w/index.php?title=Hiba-visszaterjeszt%C3%A9s&action=edit&redlink=1) *(backpropagation of errors)* kombinált [gradiensereszkedéssel](https://hu.wikipedia.org/w/index.php?title=Gradiensereszked%C3%A9s&action=edit&redlink=1) történik, de számos alternatív algoritmus is elérhető ezeken kívül.

**2.1.1.Története**

A mesterséges neuron ma is használt modellje a [Hebb tanulásban](https://hu.wikipedia.org/w/index.php?title=Hebb_tanul%C3%A1s&action=edit&redlink=1) és a küszöblogikában gyökerezik, előbbi leírta, hogy a tanulás nem passzív folyamat, hanem az ideghálózatban ideiglenesen vagy véglegesen bekövetkező biokémiai és fiziológiai változások összessége, az úgynevezett neuroplaszticitás, mely szerint az együtt tüzelő neuronok egymás iránt fogékonyabbak, egymás jelére érzékenyebbek lesznek.A küszöblogika a neuron egyik első modellje, a neuront számítási egységként kezeli, melynek több bemenete és egy kimenete van (a biológiai [neuron](https://hu.wikipedia.org/wiki/Idegsejt) számos [dendritjének](https://hu.wikipedia.org/w/index.php?title=Dendrit&action=edit&redlink=1) és egyetlen [axonjának](https://hu.wikipedia.org/wiki/Axon_(sz%C3%B6vettan)) analógiájára). A bemenetekhez egyedi súlyok tartoznak, melyekből [lineáris kombinációval](https://hu.wikipedia.org/wiki/Line%C3%A1ris_kombin%C3%A1ci%C3%B3" \o "Lineáris kombináció) előállítható a neuron izgatottsága. Ha az izgatottság egy megadott küszöbértéket átlép, a neuron "tüzel", kimenete 1, különben 0.Ezt a neuronmodellt használta a [Rosenblatt-perceptron](https://hu.wikipedia.org/w/index.php?title=Perceptron&action=edit&redlink=1), mely a 20. század közepén hatékony képfelismerő algoritmus volt.

A perceptron hátránya, hogy kettőnél több réteg esetén a tanítása nehezen kivitelezhető, ugyanis azok a gradiensereszkedések, melyek egy veszteségfüggvényt próbálnak iteratív módon minimalizálni, és ehhez a függvény gradiensével számolnak, alkalmatlanak a nem differenciálható küszöblogika tanítására. Más tanító algoritmusok (pl. [differenciál-evolúció](https://hu.wikipedia.org/w/index.php?title=Differenci%C3%A1l-evol%C3%BAci%C3%B3&action=edit&redlink=1), [hegymászó algoritmus](https://hu.wikipedia.org/wiki/Hegym%C3%A1sz%C3%B3_algoritmus)) pedig a gradiensereszkedéshez képest sokkal lassabban konvergálnak. A kettőnél több rétegű (rejtett rétegeket tartalmazó) perceptron esetében a rejtett réteg tanítása szintén egy nehéz probléma, akkor is, ha a küszöblogikát a modernebb [szigmoid](https://hu.wikipedia.org/wiki/Szigmoid_f%C3%BCggv%C3%A9nyek) jellegű aktivációs függvényeket alkalmazó szigmoidneuronokra cseréljük. Egy évtizeddel később bebizonyították, hogy a többrétegű, küszöblogikán alapuló perceptron csupán lineáris döntési határ képzésére képes, függetlenül attól, hogy hány réteggel látják el azt. A leírt nehézségekből fakadóan a mesterséges intelligencia kutatásban beköszöntött egy kevéssé termékeny időszak.

A mesterséges intelligencia telét a [hiba-visszaterjesztés algoritmus](https://hu.wikipedia.org/w/index.php?title=Hiba-visszaterjeszt%C3%A9s&action=edit&redlink=1) *(backpropagation of errors)* leírása törte meg, mely lehetővé tette a rejtett rétegeket tartalmazó, differenciálható aktivációs függvénnyel ellátott neurális hálózatok tanítását [gradiensereszkedéssel](https://hu.wikipedia.org/w/index.php?title=Gradiensereszked%C3%A9s&action=edit&redlink=1). Az új felfedezés, a számítási kapacitás exponenciális növekedése és a videókártyák lineáris algebra gyorsítóinak általános célú felhasználhatósága ([GPGPU](https://hu.wikipedia.org/w/index.php?title=GPGPU&action=edit&redlink=1)) lehetővé tette új neurális architektúrák feltalálását és azok gyakorlati alkalmazását.

További fontosabb közlemények, architektúrák a teljesség igénye nélkül:

* [LSTM](https://hu.wikipedia.org/w/index.php?title=Hossz%C3%BA-r%C3%B6vid_t%C3%A1v%C3%BA_mem%C3%B3ria&action=edit&redlink=1) (Hosszú-rövid távú memória): forradalmasította a visszacsatolásos (rekurrens) architekturákat, csökkentve a gradiens robbanás jelenség visszatartó hatását visszacsatolás alkalmazása esetén. Idősorok (pl. hang, szöveg, kézírás) analízisében használatos architektúra.
* [CNN](https://hu.wikipedia.org/w/index.php?title=Konvol%C3%BAci%C3%B3s_neur%C3%A1lis_h%C3%A1l%C3%B3zat&action=edit&redlink=1) (Konvolúciós neurális hálózat): a képanalitikában használatos, filtereket képes megtanulni.
* ReLU (aktivációs függvény): először tette lehetővé, hogy felügyeletlen előtanítás alkalmazása nélkül tanítsanak mély neurális hálózatokat, kiküszöbölve a szigmoid aktivációs függvények okozta gradiens-robbanás és gradiens-elhalás jelenségeket.
* [Adam](https://hu.wikipedia.org/w/index.php?title=Adapt%C3%ADv_lend%C3%BClet_becsl%C3%A9s&action=edit&redlink=1) (Adaptív lendület becslés): a gradiensereszkedést kiterjesztő tanító algoritmus, mely napjainkra lényegében felváltotta az eredeti optimalizálót.
* [GAN](https://hu.wikipedia.org/w/index.php?title=Generat%C3%ADv_p%C3%A1rharc_h%C3%A1l%C3%B3zat&action=edit&redlink=1) (Generatív párharc hálózat): két szembeállított neurális hálózat, generátor és diszkriminátor, az előbbi mintákat generál (pl. képeket), az utóbbi eldönti, hogy egy bemenetként átadott adat (kép) eredeti vagy a generátor által készített. A valóságoshoz megtévesztésig hasonló, de mégis egyedi minták (képek) létrehozására képes.

### 2.1.2.Neuronrétegek

A mesterséges neuron a neurális hálózat elemi számítási egysége, a biológiai neuron erősen leegyszerűsített modellje. A neuronokat általában nem önmagukban modellezzük, hanem sok ugyanolyan neuront egy rétegként értelmezünk, ekkor a legtöbb neurális számítás felírható mátrixműveletekként. A pontos művelet, amelyet egy adott réteg végez adja a réteg típusát, a hálózatban alkalmazott rétegek száma, típusa és sorrendje az adott neurális hálózat architektúrája.

-**Teljesen kapcsolt réteg** (Fully Connected, Dense, Linear Combination): előállítja a bemenetek és egy tárolt súlymátrix lineáris kombinációját:

H=XW+b

, ahol X a bemeneti mátrix, W a súlymátrix, b egy opcionális eltolósúly-vektor.

-**Egyszerű rekurrens réteg** (Recurrent, Simple recurrent): bemenetként visszakapja a saját kimenetét (az adatból képzett bemenet mellett):

Ht = XWx + bx + H(t-1)Wh + bh,

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

-**Konvolúciós réteg** (ConvXD, ahol X a bemenet dimenzióját jelöli, pl. Conv2D képek esetében): nincs teljes kapcsolás, a neuronok csak egy részét kapják a bemenetnek. Lényegében kereszt-korrelációt végez a neuronok súlymátrixával a bemeneti mátrixon:

H = X \* W + b,

ahol \* a kereszt-korrelációt jelöli.

-**Aktivációs réteg** (Activation): a bemenetként kapott mátrixra vagy tenzorra elemenként hív egy aktivációs függvényt:

H=g(X).

### 2.1.3.A neurális hálózatok felépítése

A neuronrétegek anatómiája

A neurális hálózatok rendszerint legalább három funkcionálisan és strukturálisan jól elkülöníthető részből állnak:

* **Bementi réteg**: módosítatlanul továbbítja a bemenetként átadott adatot a hálózat többi részének. Egy neurális hálózatnak több bemeneti rétege is lehet, ha elágazásokat is tartalmaz. A neuronok számát a bemeneti adat dimenzionalitása, alakja határozza meg.
* **Rejtett rétegek**: a bemenet és a kimenet között helyezkednek el, feladatuk az információ transzformációja, kódolása, illetve absztrakciók, köztes reprezentációk létrehozása. Számuk, típusuk, egymáshoz való kapcsolódásuk sorrendje és a bennük lévő neuronok száma változtatható paraméterei a hálózatnak.
* **Kimeneti réteg**: A kimeneti függvényt és a kimeneti neuronok számát az adott probléma jellege határozza meg. *Osztályozás* esetében jellemzően annyi kimeneti neuron van, ahány kategória áll a rendelkezésre, a kimeneti függvény pedig az adott osztályba tartozás valószínűségét hivatott reprezentálni a kategóriák között. Ilyen jellegű kimenet képzésére alkalmas a *szigmoid* és a *SoftMax* függvény. *Regressziós probléma* esetén kimeneti függvényt nem alkalmazunk, az utolsó rejtett réteg kimenetének lineáris kombinációját számítjuk ki. Egy neurális hálózatnak elméletileg több kimenete is lehet, ebben az esetben minden kimenethez társítani kell egy veszteségfüggvényt.

A [kernel trükköt](https://hu.wikipedia.org/w/index.php?title=Kernel_tr%C3%BCkk&action=edit&redlink=1) alkalmazó tanuló algoritmusoktól vett analógiával kijelenthetjük, hogy neurális hálózatok esetében a rejtett rétegek egy konfigurálható, tanulható kernelként működnek, ahol a kimeneti réteg az osztályozó.

Aktivációs függvények

A neuronrétegek között sokféle aktivációs függvényt alkalmazhatunk. Ezeket jellemzően elemenként értékeljük ki a bemeneti mátrixra, egyes különleges esetekben a bemenet többi elemével is számolunk.

* [**Szigmoid**](https://hu.wikipedia.org/wiki/Szigmoid_f%C3%BCggv%C3%A9nyek) vagy **logisztikus** **függvény**: �=11+�−�

H = 1/(1 + e^-X)

* . rejtett rétegek aktivációs függvényeként háttérbe szorult, mert szélsőségesen negatív vagy pozitív bemenet esetén a gradiense nagyon kis szám, ami csökkenti a tanítás hatékonyságát (gradiens elhalást idéz elő). 0 és 1 közé szorítja a bemenetet, így kimeneti rétegekben még használatos kétkategóriás osztályozás esetén és többkategóriás, többcímkés kategorizálásnál, ahol az egyes kategóriába való tartozás valószínűségét fejezhetjük ki vele.
* [**Hiperbolikus tangens**](https://hu.wikipedia.org/w/index.php?title=Hiperbolikus_tangens&action=edit&redlink=1): �=tanh⁡(�)=21+�−2�−1

H = tanh(X) = 2/(1 + e^-2X) - 1

, a szigmoidhoz hasonló aktivációs függvény, melyet jellemzően rejtett rétegekben alkalmaznak. Mára modernebb függvények jórészt felváltották. A bemenetet -1 és +1 közé szorítja.

* **ReLU** (rektifikált lineáris egység):

H = max(0, X)

 �=max(0,�). A rejtett rétegek között talán leggyakrabban használt aktivációs függvény. A negatív bemeneteket nullára állítja, a pozitívakat változatlanul hagyja. Bár 0-nál nincs deriváltja, de lebegőpontos számítás esetén rejtett rétegek között 0 bemeneti érték nagyon valószínűtlen és a gyakorlatban nem okoz problémát. Nem számításigényes és nem okoz gradiens-elhalást.

* **Leaky ReLU** (szivárgó ReLU): �=max(��,�)

H = max(*λX, X*)

. A *ReLU* esetében fellépő "Halott ReLU" jelenség kiküszöbölésére találták ki. Ha egy *ReLU* valamilyen okból akkora eltolósúlyt tanul meg, ami minden bemenetre 0 kimenetet képez, az a *ReLU* onnantól működésképtelenné válik, mert a gradiense is mindig 0 lesz. A szivárgó *ReLU*-ba épített szivárgási együttható (*λ*) egy tanulható paraméter, ideálisan 0 és 1 közötti szám.

* **Softmax**: �(��)=���∑����



. Többkategóriás osztályozási problémák esetén használt kimeneti aktivációs függvény. A *Softmax* egy vektor bemenetű függvény, melyet a [logisztikus regresszióban](https://hu.wikipedia.org/w/index.php?title=Logisztikus_regresszi%C3%B3&action=edit&redlink=1) is használnak. Először kiszámítjuk az nevező exponenciális szummáját, majd az egyes elemeket exponenciálisát elosztjuk ezzel a szummával. Kimenetként egy olyan vektort kapunk vissza, amelynek elemei 0 és 1 közötti értékek és a vektor szummája 1, így a kimenet valószínűségeloszlásként értelmezzük, mely az egyes kategóriákba való tartozás valószínűségét adja meg, a maximum érték indexe pedig a legmagasabb valószínűségű kategória.

Veszteségfüggvények

A gradiensereszkedés kivitelezése végett választanunk kell egy olyan függvényt, mely deriválható és egy objektív számértékként összefoglalja a hálózat hibáját a kimenet és a várt kimenet ismeretében. Számos veszteségfüggvény létezik a gépi tanulás témakörében, neurális hálózatok esetében a leggyakrabban alkalmazott veszteségfüggvények a következők:

* **Átlagos négyzetes eltérés** (mean squared error): �=12�‖�−�‖2



. *Szigmoid* és *softmax* kimeneti aktivációs függvénnyel együtt lassan konvergál, ezért főleg *regressziós problémák* esetében használatos, *lineáris* aktivációs függvény mellett.

* **Bináris kereszt-entrópia** (binary cross-entropy): �=−1�∑�[�log⁡(�)+(1−�)log⁡(1−�)]



 kétkategóriás osztályozási probléma esetén szigmoid kimeneti függvény mellett használják.

* **Többkategóriás kereszt-entrópia** (multiclass cross-entropy): �=−∑��log⁡(�)



. Gyakorlatilag a negatív log-likelihood értékét számítjuk ki vele. Információelméletben az információveszteség mértékét fejezi ki két bináris vektor között.

De a fentiek mellett alkalmazható a támasztóvektor-gépeknél használt **Hinge-veszteségfüggvény** vagy a **Kullback-Leibler** divergencia is.

Regularizációs technikák

A neurális hálózatok, különösen a mély architektúrák rendkívül ki vannak téve a statisztikai [túlillesztés](https://hu.wikipedia.org/w/index.php?title=T%C3%BAlilleszt%C3%A9s&action=edit&redlink=1) problémájának. Ennek kiküszöbölése végett regularizációs technikákat vethetünk be. Az alapelv az, hogy regularizált tanulás esetén a megtanult modell általánosabb lesz a regularizálatlan modellhez képest.

* [**Lp**](https://hu.wikipedia.org/wiki/Norma_(matematika))**regularizáció**: a súlyok P-normáját adjuk a veszteségfüggvényhez, általában L1 vagy L2 normát alkalmazunk. Ez a regularizáció arra készteti a hálózatot, hogy kis súlyokat tanuljon meg vagy ritkítsa a súlymátrixot (növelje a nulla súlyok számát). Ezeket a regularizációs tagokat egyszerűen hozzáadjuk a veszteségfüggvényhez és ellátjuk egy együtthatóval, amely az osztályozási pontosságból és a regularizációs tagból származó hibaérték egymáshoz képesti súlyozását képviseli ([Lagrange-szorzó](https://hu.wikipedia.org/w/index.php?title=Lagrange-szorz%C3%B3&action=edit&redlink=1)).
* **Kiejtéses (Dropout) regularizáció**: a neuronok egy véletlenszerűen kiválasztott halmazát kiejtjük a tanulási körből.
* **Lecke-normalizálás (Batch normalization)**: a hálózat nyilvántart egy futó átlagot és egy futó szórást a bemenetekről, mellyel normalizálja az újonnan kapott bemeneteket. Drasztikusan csökkenthető ezzel a konvergencia sebessége és csökken a túlillesztés esélye is.

Optimalizálók

A tanuló fázisban meghatározzuk egy neurális hálózat kimenetének hibáját egy differenciálható hibafüggvénnyel, majd megbecsüljük az egyes súlyok hozzájárulását ehhez a hibához a hibafüggvény súlyok tekintetében vett gradiensének meghatározásával. Egyszerű esetben ezt a gradienst kivonjuk a súlyokból, ezzel olyan irányba módosítjuk a súlyokat, ami garantáltan csökkenti a veszteségfüggvényt. Azonban egyes esetekben a gradiens önmagában csak lassan képes biztosítani a konvergenciát. Ilyen esetekben a konvergencia meggyorsítására a gradiensereszkedés algoritmust különböző kiterjesztésekkel láthatjuk el, ezeket a technikákat, illetve a gradienssel nem is számoló egyéb technikákat nevezzük **optimalizálóknak**.

Tipikus gradiens alapú optimalizálók:

* **Gradiensereszkedés**: a súlyokból kivonjuk a veszteségfüggvény tekintetében vett gradiensüket.
* **Lendület módszer**: nyilvántartunk egy mozgó átlagot a gradiensekből (egy "súrlódással" csökkentett sebességet) és ezt vonjuk ki a súlyokból, mintha egy labdát gurítanánk le egy domboldalon.
* **Nesterov lendület**: hasonló a lendülethez, de először megtesszük a lépést a tárolt lendület mentén, utána számolunk gradienst és visszalépünk, ha romlott a pontosság.
* **Adagrad**:adaptív gradiens, minden súly részére tárol egy egyedi tanulási rátát, mely az adott súlyra számított összes eddigi gradiens négyzetösszege. Az aktuális tanulókörben számított gradienst ennek az értéknek a gyökével elosztja, így a sokáig alacsony gradienst kapó súlyok nagyobb lépést képesek tenni.
* **RMSprop**: gyökös átlagos négyzetes terjedés: [*Geoffrey Hinton*](https://hu.wikipedia.org/w/index.php?title=Geoffrey_Hinton&action=edit&redlink=1) (a mély tanulás keresztapja) adattudós nem publikált módszere, mely nyilvántart egy mozgó átlagot a gradiensek négyzetéből és ezzel módosítja súlyonként a tanulási rátát. A reziliens terjedés (Rprop) tanítási mód adaptációja részmintás tanulásra.
* **Adam**: adaptív lendület módszer: nyilvántartja a gradiensek mozgó átlagát (lendület) és a gradiensek négyzetének mozgó átlagát (memória), kombinálva az *Adagrad* és az *RMSprop* technikákat, és a memóriával módosított egyedi tanulási rátával megszorzott lendület alapján csökkenti a súlyok értékét. Általában ezt az optimalizálót ajánlják először alkalmazni egy új problémán.

### 2.1.4.Mélytanulás

A mesterséges neurális hálózatok tanítása minden esetben egy felügyelt regressziós problémára kerül visszavezetésre, de kivitelezhető osztályozás és felügyeletlen tanítás is. A hálózatok működésében két fázist különíthetünk el: **tanítási fázisban** a ismert bemeneti paraméterek és várt kimenetek ismeretében a súlyokat változtatjuk úgy, hogy egy veszteségfüggvény értékét (például az átlagos négyzetes hibát) minimalizáljuk ezzel. A feltanított neurális hálózat a **predikciós fázisban** ezután ismeretlen bemenet átadásakor kimenetet képez, mely lehet például egy kategóriába való tartozás valószínűsége.

A jól leírható eset a három rétegű, összesítőként lineáris kombinációt, aktivációs függvényként valamilyen folytonos függvényt alkalmazó, előreterjesztéses (visszacsatolást nem tartalmazó) hálózat, amelynek esetében levezetjük a tanítás folyamatát.

### 2.1.5.Tanítás

A mesterséges neurális hálózatok tanítása minden esetben egy felügyelt regressziós problémára kerül visszavezetésre, de kivitelezhető osztályozás és felügyeletlen tanítás is. A hálózatok működésében két fázist különíthetünk el: **tanítási fázisban** a ismert bemeneti paraméterek és várt kimenetek ismeretében a súlyokat változtatjuk úgy, hogy egy veszteségfüggvény értékét (például az átlagos négyzetes hibát) minimalizáljuk ezzel. A feltanított neurális hálózat a **predikciós fázisban** ezután ismeretlen bemenet átadásakor kimenetet képez, mely lehet például egy kategóriába való tartozás valószínűsége.

A jól leírható eset a három rétegű, összesítőként lineáris kombinációt, aktivációs függvényként valamilyen folytonos függvényt alkalmazó, előreterjesztéses (visszacsatolást nem tartalmazó) hálózat, amelynek esetében levezetjük a tanítás folyamatát.

Kimenetképzés

Tanításkor jellemzően nem egyetlen bemeneti vektort adunk át, hanem egy részmintát képzünk az adatainkból és azokra egyszerre határozzuk meg a veszteségfüggvényünk gradiensét, majd a kapott gradienseket átlagoljuk az egyes súlyokra. *m* részmintaméret és *d* dimenziós bemenet esetében tehát a bemenetünk a következő mátrix lesz: �∈��×�

.

A hálózat rejtett rétegének súlyait és eltolósúlyait a következőképpen definiálhatjuk: �ℎ∈��×�,�ℎ∈��



, ahol ***W****h* a súlymátrix, ***b****h* pedig az úgynevezett eltolósúly-vektor.

A rejtett réteg által végzett művelet a következő: �=�ℎ(��ℎ+�ℎ)



, ahol *gh* a rejtett réteg aktivációs függvényét jelöli. A bemeneti mátrixot megszorozva a súlymátrixszal, az eredeti bemenetet neurontérbe transzformáljuk. Az eltolósúly a bemenetekben fellelhető esetleges torzítás kiküszöbölése végett van jelen. A transzformáció eredményére elemenként a logisztikus függvényt hívjuk, mely 0 és 1 közé szorítja a kimenetet. A rejtett réteg kimenete, �∈��×�



.

Az utolsó réteg a kimeneti réteg, melyet szintén egy súlymátrix és egy eltolósúly-vektor definiál: ��∈��×�,��∈��



, ahol *v* a kimeneti neuronok száma.

A kimeneti réteg a következő műveletet végzi: �=��(���+��)



, ahol *go* a kimeneti réteg aktivációs függvényét jelöli.

Hiba meghatározása és visszaterjesztése

A hálózat kimenetének hibáját a várt kimenet ismeretében egy folytonos függvény, az úgynevezett veszteségfüggvény segítségével számszerűsítjük. A hálózat egyes súlyainak hozzájárulása a hibához a veszteségfüggvény súlyok tekintetében vett gradiensével egyenlő: ∇��={∂�∂��}



, ahol *w* a hálózat összes súlyát tartalmazó képzeletbeli vektor, *C* pedig a veszteségfüggvény (például az átlagos négyzetes eltérés).

A kimeneti réteg súlyai tekintetében vett gradiens a láncszabállyal határozható meg: ∂�∂�=∂�∂�∂�∂�

.

A rejtett rétegek hibájának meghatározásához alkalmaznunk kell a ***hiba-visszaterjesztést***. Ehhez észre kell vennünk, hogy a lineáris kombinációig (tehát az aktivációs függvény hívása előttig) visszaszámított gradiens kétszer is felhasználásra kerül: egyrészt meghatározható általa a súlyok gradiense, másrészt az előző réteg kimenetének tekintetében vett gradiens, mely az előző réteg tanításához kell. A lineáris kombinációig visszaszámított gradiensre bevezetjük a delta-jelölést: ��=∇�∗�



, ahol O\* a kimeneti réteg lineáris kombinációját jelöli, ��pedig jelen definíció szerint a kimeneti réteg hibája vagy deltája és a következőképpen határozzuk meg: ��=∂�∂�⊙�′(�)



, ahol **g** az aktivációs függvény deriváltja, melyet elemenként kiértékelünk az *O* mátrixra, ⊙ pedig mátrixok elemenkénti szorzását jelöli. A delta jelöléssel felírjuk a kimeneti súlyok gradiensét:∇���=��∂�∂��=����



, ahol *T* felső indexben a transzponálást jelöli.

A rejtett réteg hibájának meghatározásához vissza kell terjeszteni a gradienst a rejtett réteg kimenetéig, majd kiszámítani a rejtett réteg deltáját, végül a deltából a súlyok gradiensét:

∂�∂�=��∂�∂�=����� eljuttat minket a rejtett réteg kimenetének gradienséig.

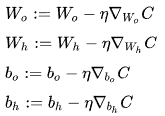
��=∂�∂�⊙�′(�) a rejtett réteg deltája (gradiens a lineáris kombinációnál).

∇�ℎ�=��∂�∂�ℎ=���� a rejtett réteg súlyainak gradiense.

Az eltolósúlyok gradiense ∇���=∂�∂�∂�∂�=��×1 és ∇�ℎ�=∂�∂�∂�∂�=��×1.

Grádiensereszkedés

Ha neurális hálózat minden rétegének és minden súlyának meghatároztuk a gradiensét, a tanulási rátával (*η*) modulált gradiens kivonható a súlyok aktuális értékéből, így minden súlyt olyan irányba térítünk el, amely a veszteségfüggvény értékének csökkenéséhez vezet:��:=��−�∇���

�ℎ:=�ℎ−�∇�ℎ���:=��−�∇���

Ezzel zárul egy tanulási iteráció. A következő iterációban újra részmintát képzünk a tanuló adatsorból, kimenetet képzünk, hibát számolunk és visszaterjesztünk, majd gradiensereszkedést hajtunk végre. A tanuló adatsor egyszer teljes végigjárását nevezzük egy tanulókorszaknak. Az iteráció során a veszteségfüggvény kimenetét nyilvántartjuk és addig folytatjuk a tanítást, míg a veszteséget lehetőleg minimálisra csökkentjük. Ekkor elmondható, hogy az algoritmus egy minimumra konvergált.

**2.1.6.Implementációk**

* [Tensorflow](https://www.tensorflow.org/): szimbolikus számítási gráfok definíciójára és automatikus differenciálására használható, [Google](https://hu.wikipedia.org/wiki/Google_Inc.) által fejlesztett könyvtár, mely neurális operációkat is tartalmaz. [Python](https://hu.wikipedia.org/wiki/Python_(programoz%C3%A1si_nyelv)), [C++](https://hu.wikipedia.org/wiki/C%2B%2B) és más nyelveken elérhető könyvtárként.
* [Theano](http://deeplearning.net/software/theano/) [Archiválva](https://web.archive.org/web/20201108233358/http:/deeplearning.net/software/theano/) 2020. november 8-i dátummal a [Wayback Machine](https://hu.wikipedia.org/wiki/Internet_Archive)-ben: a Tensorflow-hoz hasonló könyvtár, a [Montreáli Egyetem](https://hu.wikipedia.org/w/index.php?title=University_of_Montr%C3%A9al&action=edit&redlink=1) fejlesztésében. Pythonon elérhető könyvtárként.
* [CNTK](https://www.microsoft.com/en-us/cognitive-toolkit/): Microsoft Cognitive Toolkit, a Microsoft által fejlesztett szimbolikus könyvtár. Python, C++ és más programnyelveken elérhető könyvtárként.
* [Keras](https://keras.io/): Tensorflow-ra, Theano-ra vagy CNTK-ra épülő, kifejezetten mély tanuláshoz és neurális hálózatok gyors definíciójához, [CPU](https://hu.wikipedia.org/wiki/Central_processing_unit)-n és [GPU](https://hu.wikipedia.org/wiki/Graphics_processing_unit)-n történő futtatásához használható, Python nyelvhez elérhető könyvtár.
* [Torch](http://torch.ch/): [Lua](https://hu.wikipedia.org/wiki/Lua_(programoz%C3%A1si_nyelv)) nyelvre elérhető neurális hálózat és gépi tanulás könyvtár.
* [Caffe](http://caffe.berkeleyvision.org/): Pythonon és [MATLAB](https://hu.wikipedia.org/wiki/MATLAB)-on is futni képes, neurális hálózatok és számítások definiálhatóak vele [JSON](https://hu.wikipedia.org/wiki/JSON)-szerű szintaxissal.
* [Brainforge](https://github.com/csxeba/brainforge): szimbolikus gráfokat nem alkalmazó, csupán mátrix-műveletekként definiált neurális hálózat könyvtár Python programnyelvhez.

**3.Keretrendszerek**

A [számítógép-programozásban](https://hu.wikipedia.org/wiki/Sz%C3%A1m%C3%ADt%C3%B3g%C3%A9p-programoz%C3%A1s) a szoftverkörnyezet egy absztrakció, ami a szoftver által nyújtott általános funkcionalitást képes szelektíven megváltoztatni a felhasználói kód alapján, így alkalmazásspecifikus szoftvert biztosítanak. A keretrendszer szabványosítja az alkalmazások felépítésére és telepítésére. Általános, újrafelhasználható szoftverkörnyezet, ami egy nagyobb platform részeként megkönnyíti alkalmazások, termékek és megoldások fejlesztését. Tartalmazhatnak programokat, fordítókat, könyvtárakat, eszközkészleteket, [alkalmazásprogramozási interfészek](https://hu.wikipedia.org/wiki/API) (API) [programkönyvtárakat](https://hu.wikipedia.org/wiki/Programk%C3%B6nyvt%C3%A1r), amelyek komponenseket raknak össze, hogy támogassák a projekt vagy a rendszer fejlesztését.

A témához kapcsolódó, a fejlesztésben gyakran használt keretrendszerek: OpenCV, PyTorch, NumPy, CMake, TensorFlow.

Az általam letesztelt programok közül az OpenCV és a PyTorch volt alkalmazva.

**3.1.OpenCV**

Az OpenCV (Open Source Computer Vision Library: http://opencv.org) egy nyílt forráskódú könyvtár, amely több száz számítógépes látási algoritmust tartalmaz. A dokumentum az úgynevezett OpenCV 2.x API-t írja le, amely lényegében egy C++ API, szemben a C-alapú OpenCV 1.x API-val (a C API elavult és az OpenCV 2.4-es kiadása óta nem tesztelik "C" fordítóval).

Az OpenCV moduláris felépítésű, ami azt jelenti, hogy a csomag több megosztott vagy statikus könyvtárat tartalmaz. A következő modulok állnak rendelkezésre:

Alapfunkciók (core) - egy kompakt modul, amely meghatározza az alapvető adatszerkezeteket, beleértve a sűrű többdimenziós tömböt Mat és az összes többi modul által használt alapvető funkciókat.

Képfeldolgozás (imgproc) - egy képfeldolgozó modul, amely lineáris és nemlineáris képszűrést, geometriai képtranszformációkat (átméretezés, affin és perspektivikus torzítás, általános táblázat alapú remapping), színtérkonverziót, hisztogramokat stb. tartalmaz.

Videoelemzés (video) - egy videóelemző modul, amely mozgásbecslési, háttérkivonási és tárgykövető algoritmusokat tartalmaz.

Kamerakalibrálás és 3D rekonstrukció (calib3d) - alapvető többnézetes geometria algoritmusok, egy- és sztereó kamerakalibrálás, objektum pózbecslés, sztereó megfeleltetési algoritmusok és a 3D rekonstrukció elemei.

2D Features Framework (features2d) - kiemelkedő jellemzők detektorai, deszkriptorok és deszkriptor-illesztők.

Objektumdetektálás (objdetect) - objektumok és az előre meghatározott osztályok példányainak (például arcok, szemek, bögrék, emberek, autók stb.) detektálása.

Magas szintű GUI (highgui) - egyszerű felhasználói felület egyszerű UI-képességekhez.

Video I/O (videoio) - könnyen használható felület a videófelvételekhez és a videokodecekhez.

... néhány egyéb segédmodul, például FLANN és Google teszt wrapperek, Python kötések és egyéb..

Az OpenCV automatikusan felszabadítja a memóriát, valamint a legtöbbször automatikusan kiosztja a memóriát a kimeneti függvény paraméterei számára. Az OpenCV automatikusan kezeli az összes memóriát. Számítógépes látás könyvtárként az OpenCV sokat foglalkozik olyan képpontokkal, amelyek gyakran kompakt, csatornánként 8 vagy 16 bites formában vannak kódolva, és így korlátozott értéktartományuk van. Továbbá a képeken végzett bizonyos műveletek, mint például a színtérkonverziók, a fényerő/kontraszt beállítások, az élességállítás, az összetett interpoláció (bi-kubikus, Lanczos) a rendelkezésre álló tartományon kívüli értékeket eredményezhetnek. Ha csak az eredmény legalsó 8 (16) bitjét tárolja, az vizuális leleteket eredményez, és befolyásolhatja a további képelemzést. Ennek a problémának a megoldására az úgynevezett telítési aritmetikát használják.

A sablonok a C++ egyik nagyszerű tulajdonsága, amely lehetővé teszi nagyon hatékony, hatékony és biztonságos adatszerkezetek és algoritmusok megvalósítását. A sablonok széleskörű használata azonban drámaian megnövelheti a fordítási időt és a kód méretét. Emellett a sablonok kizárólagos használata esetén nehéz szétválasztani az interfészt és az implementációt. Ez jó lehet az alapvető algoritmusok esetében, de nem jó a számítógépes látás könyvtárak számára, ahol egyetlen algoritmus több ezer sornyi kódot is tartalmazhat. Emiatt, valamint azért is, hogy egyszerűsítsük a kötések fejlesztését más nyelvekhez, például Pythonhoz, Java-hoz, Matlabhoz, amelyek egyáltalán nem rendelkeznek sablonokkal vagy korlátozott sablonképességekkel, a jelenlegi OpenCV implementáció a polimorfizmuson és a sablonok feletti futásidejű diszpécselésen alapul. Az OpenCV számos függvénye sűrű kétdimenziós vagy többdimenziós numerikus tömböket dolgoz fel.

Az OpenCV a kritikus hibák jelzésére kivételeket használ. Ha a bemeneti adatok megfelelő formátumúak és a megadott értéktartományba tartoznak, de az algoritmus valamilyen okból nem jár sikerrel (például az optimalizáló algoritmus nem konvergált), akkor egy speciális hibakódot ad vissza (jellemzően csak egy bólusváltozót). A jelenlegi OpenCV implementáció teljesen újra beírható. Ez azt jelenti, hogy ugyanaz a függvény vagy ugyanazok a metódusok különböző osztálypéldányokban különböző szálakból hívhatók. Továbbá ugyanaz a Mat különböző szálakban is használható, mivel a hivatkozási számítási műveletek az architektúra-specifikus atomi utasításokat használják.

**3.2.PyTorch**

A PyTorch egy optimalizált tenzorkönyvtár a GPU-kat és CPU-kat használó mélytanuláshoz. A TorchScript segítségével a PyTorch egyszerű használatot és rugalmasságot biztosít eager módban, miközben a sebesség, optimalizálás és funkcionalitás érdekében a C++ futásidejű környezetekben zökkenőmentesen átvált grafikus módba. A TorchServe egy könnyen használható eszköz a PyTorch modellek méretarányos telepítéséhez. Felhő- és környezetfüggetlen, és olyan funkciókat támogat, mint a több modell kiszolgálása, a naplózás, a mérőszámok és a RESTful végpontok létrehozása az alkalmazásintegrációhoz. Optimalizálja a teljesítményt mind a kutatásban, mind a termelésben a kollektív műveletek aszinkron végrehajtásának és a Pythonból és C++-ból elérhető peer-to-peer kommunikációnak a natív támogatásának kihasználásával. A PyTorch támogatja a Pythontól az iOS és Android rendszereken történő telepítésig tartó, végponttól végpontig tartó munkafolyamatot. Kiterjeszti a PyTorch API-t, hogy lefedje az ML mobilalkalmazásokba való beépítéséhez szükséges általános előfeldolgozási és integrációs feladatokat. A kutatók és fejlesztők aktív közössége a PyTorch bővítésére és a számítógépes látástól a megerősített tanulásig terjedő területeken történő fejlesztés támogatására szolgáló eszközök és könyvtárak gazdag ökoszisztémáját építette ki. Exportálja a modelleket a szabványos ONNX (Open Neural Network Exchange) formátumban az ONNX-kompatibilis platformok, futtatók, vizualizátorok stb. közvetlen eléréséhez. A C++ frontend egy tiszta C++ interfész a PyTorch-hoz, amely követi a Python frontend kialakítását és architektúráját. Célja, hogy lehetővé tegye a nagy teljesítményű, alacsony késleltetésű és csupasz fém C++ alkalmazások kutatását. A PyTorch jól támogatott a főbb felhőplatformokon, súrlódásmentes fejlesztést és egyszerű skálázást biztosít az előre elkészített képek, a GPU-kon történő nagyméretű képzés, a modellek termelési környezetben történő futtatásának képessége és még sok más.

**3.3.OpenCV vagy PyTorch**

OpenCV előnyei: Computer Vision, nyílt forráskódú, képalkotó tulajdonság, gépi tanulás, arcfelismerés, nagyobb közösség, valós idejű képfeldolgozás, CV problémák közel teljes megoldása, képnövelés.

PyTorch előnyei: könnyebb használat, fejlesztőbarát, könnyebb hibakeresés, akár a TensorFlow-nál is gyorsabb lehet.

**4.Mélytanulási modellek**

A mélytanulási modellek hasonlóak a szuperintelligens gépekhez, amelyek nagyszámú képet képesek megvizsgálni, és meghatározni, mi a közös bennük. Ezt úgy érik el, hogy szétszedik a képeket, és mindegyiket külön-külön tanulmányozzák. Ezután alkalmazzák a tanultakat, hogy azonosítsák a mintákat, és jóslatokat készítsenek olyan friss képekről, amelyeket korábban soha nem láttak. A mélytanulási modellek olyan mesterséges neurális hálózatok, amelyek képesek megtanulni és kinyerni a bonyolult mintákat és jellemzőket hatalmas adatkészletekből. Ezek a modellek összekapcsolt csomópontok vagy neuronok több rétegéből állnak, amelyek elemzik és módosítják a bejövő adatokat a kimenet létrehozása érdekében.

A mély tanulási modellek különösen jól illeszkednek a nagy pontosságot és precizitást igénylő munkákhoz, mint például a képazonosítás, a beszédfelismerés, a természetes nyelv feldolgozása és a robotika. Íme a vizualizáció egyszerűsített változata az adatáramlás szemléltetésére egy mély tanulási modellben. A bemeneti adatok a modell bemeneti rétegébe áramlanak, amely aztán számos rejtett rétegen továbbítja az adatokat, mielőtt kimeneti előrejelzést adna. Minden rejtett réteg matematikai műveletek sorozatát hajtja végre a bemeneti adatokon, mielőtt átadná azokat a következő rétegnek, amely a végső előrejelzést adja.

**4.1.YOLO**

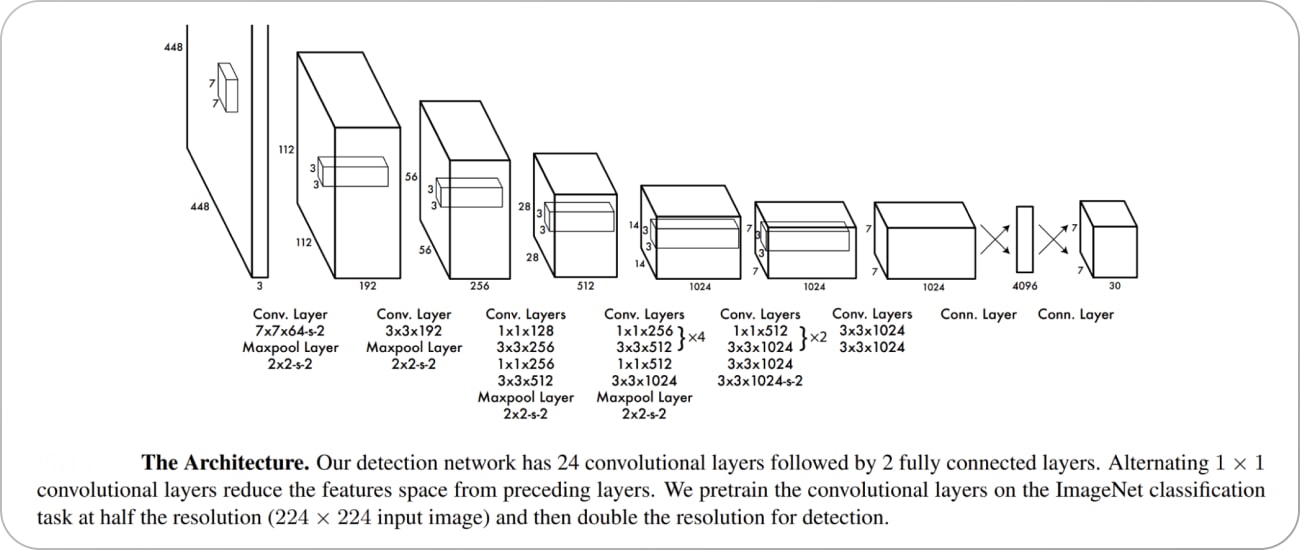
A You Only Look Once (YOLO) egy végponttól végpontig működő neurális hálózatot javasol, amely egyszerre készít előrejelzéseket a határoló dobozokról és az osztályok valószínűségeiről. Ez eltér a korábbi objektumdetektáló algoritmusok megközelítési módjától, amelyek az osztályozókat használták fel a detektáláshoz.

A YOLO a tárgyak észlelésének alapvetően eltérő megközelítését követve a legkorszerűbb eredményeket érte el, és nagy különbséggel verte meg a többi valós idejű tárgyfelismerő algoritmust.

Míg az olyan algoritmusok, mint a Faster RCNN úgy működnek, hogy a Region Proposal Network segítségével detektálják a lehetséges érdekes régiókat, majd külön-külön elvégzik a felismerést ezeken a régiókon, addig a YOLO egyetlen teljesen összekapcsolt réteg segítségével végzi el az összes előrejelzést.

A Region Proposal Networks-et használó módszerek több ismétlést végeznek ugyanazon a képen, míg a YOLO egyetlen ismétléssel megússza.

A YOLO 2015-ös első kiadása óta ugyanannak a modellnek több új verzióját javasolták, amelyek mindegyike az elődjére épül és azt javítja. A YOLO algoritmus bemenetként egy képet vesz, majd egy egyszerű mély konvolúciós neurális hálózatot használ a képen lévő objektumok felismerésére. A YOLO gerincét alkotó CNN-modell felépítése az alábbiakban látható.



**4.1.1.YOLO v2**

A YOLO v2-t úgy tervezték, hogy gyorsabb és pontosabb legyen, mint a YOLO, és az objektumosztályok szélesebb körét képes legyen felismerni. Ez a frissített változat egy másik CNN-hátteret is használ, amelyet Darknet-19-nek neveznek, és amely a VGGNet architektúra egy változata egyszerű progresszív konvolúciós és pooling rétegekkel.

A YOLO v2 egyik fő újítása a horgonydobozok használata. A horgonydobozok különböző képarányú és méretarányú, előre meghatározott határoló dobozok halmaza. A határoló dobozok előrejelzésekor a YOLO v2 a horgonydobozok és a megjósolt eltolódások kombinációját használja a végső határoló doboz meghatározásához. Ez lehetővé teszi, hogy az algoritmus az objektumméretek és képarányok szélesebb skáláját kezelje.

A YOLO v2 másik fejlesztése a tételes normalizálás használata, amely segít a modell pontosságának és stabilitásának javításában. A YOLO v2 több léptékű képzési stratégiát is alkalmaz, amely a modell több léptékű képeken történő képzését, majd a jóslatok átlagolását jelenti. Ez segít javítani a kisméretű objektumok észlelési teljesítményét.

A YOLO v2 egy új veszteségfüggvényt is bevezet, amely jobban megfelel a tárgyfelismerési feladatokhoz. A veszteségfüggvény az előre jelzett és a földi igazság határoló dobozok és az osztály valószínűségei közötti négyzetes hibák összegén alapul.

**4.1.2.YOLO v3**

A YOLO v3 egyik fő újítása a Darknet-53 nevű új CNN-architektúra használata. A Darknet-53 a ResNet architektúra egy változata, és kifejezetten tárgyfelismerési feladatokra tervezték. Az architektúra 53 konvolúciós réteggel rendelkezik, és képes a legkorszerűbb eredményeket elérni különböző objektumdetektálási benchmarkokon.

A YOLO v3 másik fejlesztése a különböző méretarányú és képarányú horgonyzódobozok. A YOLO v2-ben a horgonydobozok mind egyforma méretűek voltak, ami korlátozta az algoritmus képességét a különböző méretű és alakú objektumok észlelésére. A YOLO v3-ban a horgonyzódobozok méretaránya és a képarányok változnak, hogy jobban illeszkedjenek az észlelt objektumok méretéhez és alakjához.

A YOLO v3 bevezeti a "feature pyramid networks" (FPN) fogalmát is. Az FPN-ek egy olyan CNN-architektúra, amelyet több léptékű objektumok észlelésére használnak. Ezek egy jellemzőtérképekből álló piramist építenek fel, amelynek minden egyes szintjét különböző léptékű objektumok észlelésére használják. Ez segít javítani a kis méretű objektumok észlelési teljesítményét, mivel a modell több léptékben is képes látni az objektumokat.

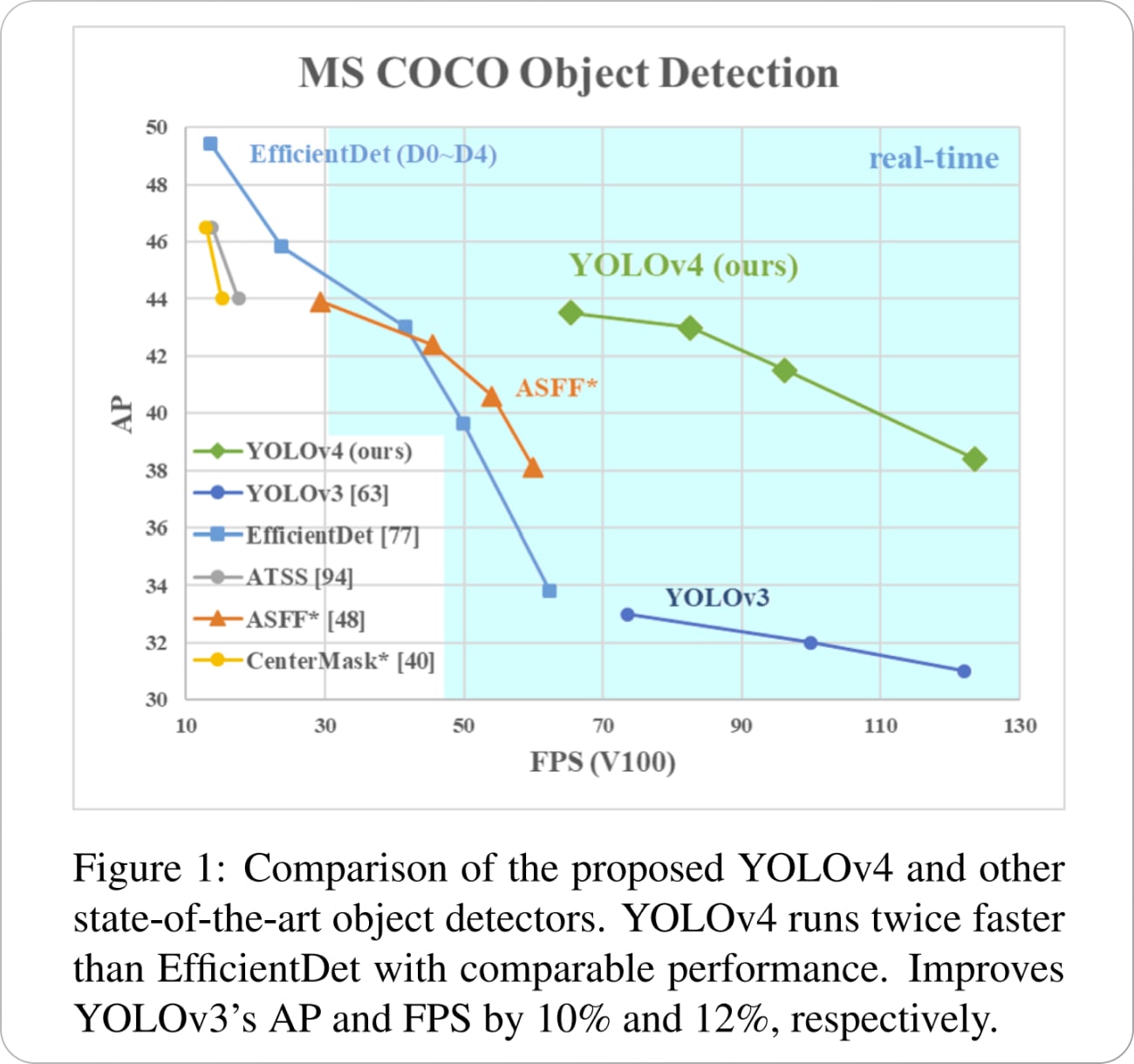
Ezen fejlesztések mellett a YOLO v3 a tárgyméretek és képarányok szélesebb skáláját tudja kezelni. Emellett pontosabb és stabilabb, mint a YOLO korábbi verziói.

**4.1.3.YOLO v4**

A YOLO v4 elsődleges fejlesztése a YOLO v3-hoz képest a CSPNet nevű új CNN-architektúra használata (lásd alább). A CSPNet a "Cross Stage Partial Network" rövidítése, és a ResNet architektúra egy olyan változata, amelyet kifejezetten tárgyfelismerési feladatokra terveztek. Viszonylag sekély struktúrával rendelkezik, mindössze 54 konvolúciós réteggel. Ennek ellenére a legkorszerűbb eredményeket képes elérni különböző objektumdetektálási benchmarkokon.

Mind a YOLO v3, mind a YOLO v4 különböző méretarányú és képarányú horgonyzódobozokat használ, hogy jobban illeszkedjen az észlelt objektumok méretéhez és alakjához. A YOLO v4 új módszert vezet be a horgonyzódobozok létrehozására, az úgynevezett "k-means klaszterezést". Ennek lényege, hogy egy klaszterező algoritmus segítségével az alaphelyzetben használt határoló dobozokat klaszterekbe csoportosítja, majd a klaszterek középpontjait használja horgonyzódobozként. Ez lehetővé teszi, hogy a horgonyzódobozok jobban igazodjanak az észlelt objektumok méretéhez és alakjához.

Bár a YOLO v3 és a YOLO v4 is hasonló veszteségfüggvényt használ a modell képzéséhez, a YOLO v4 bevezet egy új kifejezést, a "GHM veszteséget". Ez a fókuszos veszteségfüggvény egy változata, és arra szolgál, hogy javítsa a modell teljesítményét kiegyensúlyozatlan adathalmazokon. A YOLO v4 javítja a YOLO v3-ban használt FPN-ek architektúráját is.



**4.1.4.YOLO v5**

A YOLO v5 a YOLO-tól eltérően egy összetettebb, EfficientDet nevű architektúrát használ (az alábbiakban látható architektúra), amely az EfficientNet hálózati architektúrán alapul. A YOLO v5 komplexebb architektúra használata lehetővé teszi a nagyobb pontosság elérését és a jobb általánosítást a tárgykategóriák szélesebb körére.

Egy másik különbség a YOLO és a YOLO v5 között a tárgyfelismerő modell megtanulásához használt képzési adatokban rejlik. A YOLO-t a PASCAL VOC adathalmazon képezték ki, amely 20 tárgykategóriából áll. A YOLO v5-öt viszont egy nagyobb és változatosabb, D5 nevű adathalmazon képezték ki, amely összesen 600 tárgykategóriát tartalmaz.

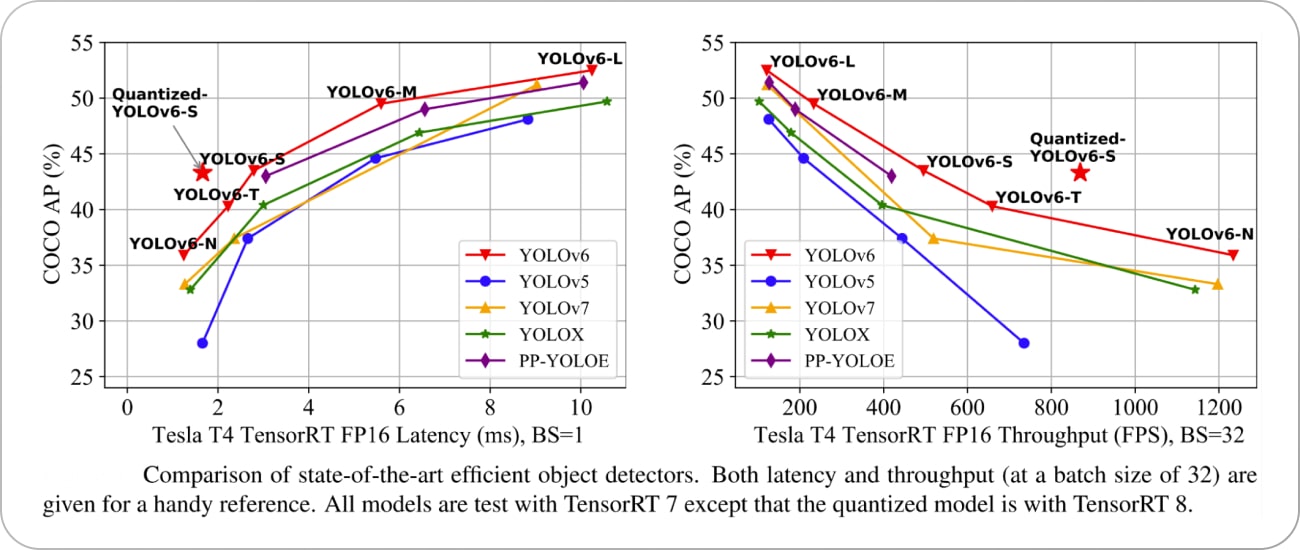
A YOLO v5 egy új módszert használ a horgonydobozok generálására, amelyet "dinamikus horgonydobozoknak" nevezünk. Ennek lényege, hogy egy klaszterező algoritmus segítségével a ground truth bounding boxokat klaszterekbe csoportosítja, majd a klaszterek középpontjait használja horgonydobozként. Ez lehetővé teszi, hogy a horgonyzódobozok jobban igazodjanak az észlelt objektumok méretéhez és alakjához.

A YOLO v5 bevezeti a "spatial pyramid pooling" (SPP) fogalmát is, amely egyfajta pooling réteg, amelyet a jellemzőtérképek térbeli felbontásának csökkentésére használnak. Az SPP-t a kis objektumok észlelési teljesítményének javítására használják, mivel lehetővé teszi, hogy a modell több léptékben lássa az objektumokat. A YOLO v4 szintén SPP-t használ, de a YOLO v5 számos olyan fejlesztést tartalmaz az SPP architektúrájában, amelyek jobb eredmények elérését teszik lehetővé.

A YOLO v4 és a YOLO v5 hasonló veszteségfüggvényt használ a modell betanításához. A YOLO v5 azonban bevezet egy új kifejezést, a "CIoU veszteséget", amely az IoU veszteségfüggvény egy olyan változata, amelynek célja a modell teljesítményének javítása kiegyensúlyozatlan adathalmazokon.

**4.1.5.YOLO v6**

A YOLO v5 és a YOLO v6 közötti egyik fő különbség az alkalmazott CNN-architektúra. A YOLO v6 az EfficientNet architektúra EfficientNet-L2 nevű változatát használja. Ez egy hatékonyabb architektúra, mint a YOLO v5-ben használt EfficientDet, kevesebb paraméterrel és nagyobb számítási hatékonysággal. A legkorszerűbb eredményeket képes elérni különböző objektumfelismerési benchmarkokon. A YOLO v6 modell keretrendszere az alábbiakban látható. A YOLO v6 egy új módszert is bevezet a horgonyzó dobozok létrehozására, az úgynevezett "sűrű horgonyzó dobozokat".



**4.1.6.YOLO v7**

Az egyik legfontosabb fejlesztés a horgonyzó dobozok használata.

A horgonydobozok különböző képarányú, előre definiált dobozok halmaza, amelyek különböző alakú objektumok felismerésére szolgálnak. A YOLO v7 kilenc horgonyzódobozt használ, ami lehetővé teszi, hogy a korábbi verziókhoz képest a tárgyak formáinak és méreteinek szélesebb skáláját észlelje, ezáltal csökkentve a téves pozitív eredmények számát.

A YOLO v7 egyik legfontosabb fejlesztése egy új veszteségfüggvény, a "fókuszveszteség" használata. A YOLO korábbi verziói egy standard kereszt-entrópia veszteségfüggvényt használtak, amelyről köztudott, hogy kevésbé hatékony a kis objektumok észlelésében. A fókuszveszteség ezt a problémát úgy küszöböli ki, hogy a jól osztályozott példák veszteségét lefelé súlyozza, és a nehéz példákra - a nehezen észlelhető objektumokra - összpontosít.

A YOLO v7 nagyobb felbontással is rendelkezik, mint a korábbi verziók. A képeket 608 x 608 pixeles felbontással dolgozza fel, ami magasabb, mint a YOLO v3-ban használt 416 x 416 pixeles felbontás. Ez a nagyobb felbontás lehetővé teszi a YOLO v7 számára, hogy kisebb objektumokat is felismerjen, és összességében nagyobb pontosságot érjen el.

A YOLO v7 egyik fő előnye a gyorsasága. A képeket másodpercenként 155 képkocka sebességgel képes feldolgozni, ami sokkal gyorsabb, mint más korszerű objektumfelismerő algoritmusok. Még az eredeti YOLO alapmodell is legfeljebb 45 képkocka/másodperc sebességgel volt képes feldolgozni a képeket. Ez alkalmassá teszi az eszközt olyan érzékeny, valós idejű alkalmazásokhoz, mint a megfigyelés és az önvezető autók, ahol a nagyobb feldolgozási sebesség kulcsfontosságú.

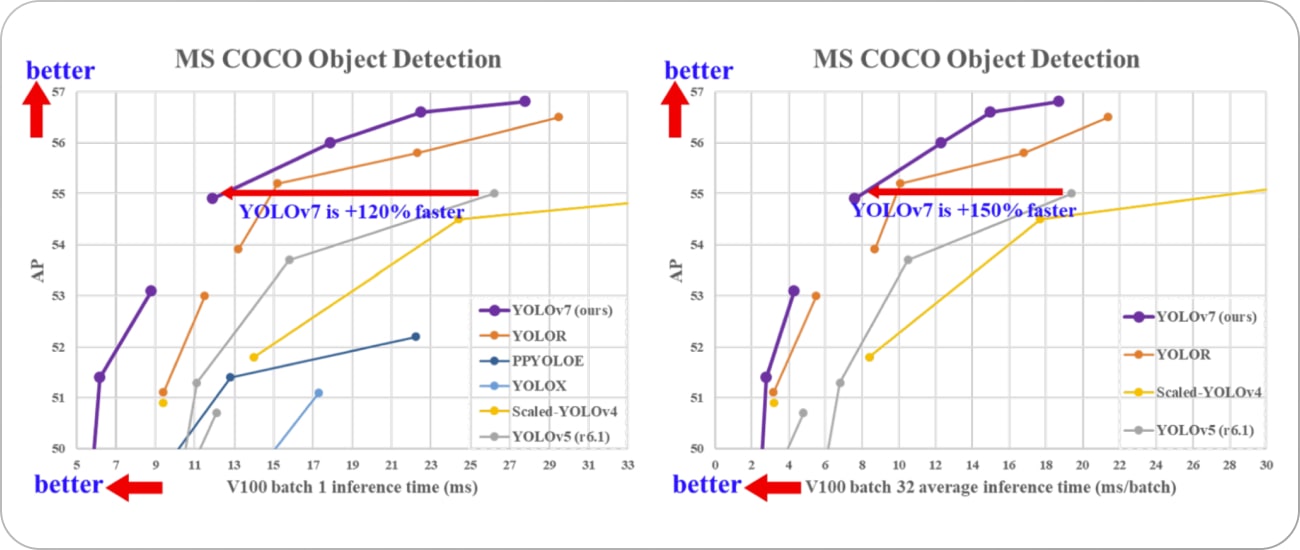
A YOLO v7 egy erős és hatékony objektumfelismerő algoritmus, de van néhány korlátja.

A YOLO v7, mint sok más objektumfelismerő algoritmus, nehezen érzékeli a kis objektumokat. A zsúfolt jelenetekben vagy a kamerától távol lévő tárgyak esetén előfordulhat, hogy nem érzékeli pontosan a tárgyakat.

A YOLO v7 nem tökéletes a különböző léptékű objektumok észlelésében sem. Ez megnehezítheti a jelenet többi objektumához képest nagyon nagy vagy nagyon kicsi objektumok észlelését.

A YOLO v7 érzékeny lehet a fényviszonyok vagy más környezeti feltételek változásaira, ezért nem biztos, hogy kényelmetlen lehet a valós alkalmazásokban használni, ahol a fényviszonyok változhatnak.

A YOLO v7 számításigényes lehet, ami megnehezítheti a valós idejű futtatást erőforrás-korlátozott eszközökön, például okostelefonokon vagy más peremeszközökön.

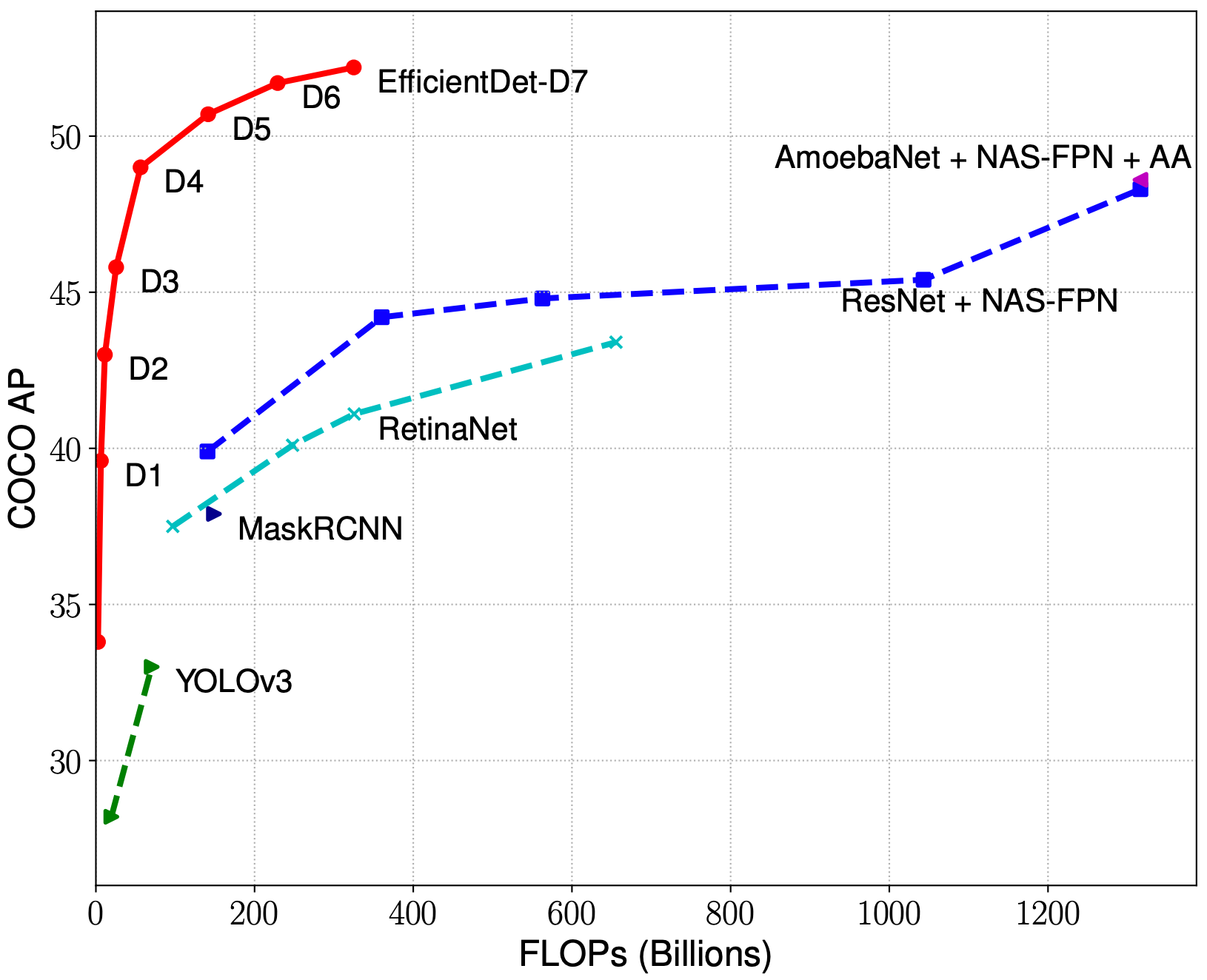


**4.1.7.YOLO v8**

A YOLO v8 egy új API-val büszkélkedhet, amely sokkal könnyebbé teszi a képzést és a következtetést mind CPU-, mind GPU-eszközökön, és a keretrendszer támogatja a korábbi YOLO-verziókat. A fejlesztők még dolgoznak egy tudományos cikk kiadásán, amely a modell architektúrájának és teljesítményének részletes leírását tartalmazza majd.

**4.2.EfficientDet**

Az EfficientDet egy korszerű objektumdetektálási modell valós idejű objektumdetektálásra, amely eredetileg Tensorflow és Keras nyelven íródott, de ma már PyTorch implementációval rendelkezik - ez a notebook az EfficientDet PyTorch implementációját használja. EfficientNet gerinccel és egy egyedi detektáló és osztályozó hálózattal rendelkezik. Ennek a gerinchálózatnak köszönhetően az EffcientDet úgy van kialakítva, hogy a legkisebb modellmérettől kezdve hatékonyan skálázódjon. A legkisebb EfficientDet, az EfficientDet-D0 4 millió súlyparaméterrel rendelkezik - ez valóban apró. Az EfficientDet ebben az eloszlásban 30 ms alatt következtet, és mindössze 17 megabájtnyi tárhelyet tekintve és tárolva - így egyszerre kicsi és gyors modell.



**4.2.1.EfficientDet-Lite0**

Az EfficientDet-Lite0 modell egy EfficientNet-Lite0 gerinchálózatot használ 320x320 bemeneti mérettel és BiFPN funkcióhálózattal. A modellt a COCO-adatkészlettel képeztük ki, amely egy nagyméretű tárgyfelismerő adathalmaz, amely 1,5 millió tárgypéldányt és 80 tárgycímkét tartalmaz. Tekintse meg a támogatott címkék teljes listáját. Az EfficientDet-Lite0 int8 modellként, float16 modellként vagy float32 modellként érhető el. Ez a modell ajánlott, mert egyensúlyt teremt a késleltetés és a pontosság között. Egyszerre pontos és elég könnyű számos felhasználási esethez.

**4.2.2. EfficientDet-Lite2**

Az EfficientDet-Lite2 modell egy EfficientNet-Lite2 gerinchálózatot használ 448x448 bemeneti mérettel és BiFPN funkcióhálózattal. A modellt a COCO-adatkészlettel, egy nagyméretű tárgyfelismerési adatkészlettel képeztük, amely 1,5 millió tárgypéldányt és 80 tárgycímkét tartalmaz. Tekintse meg a támogatott címkék teljes listáját. Az EfficientDet-Lite2 int8 modellként, float16 modellként vagy float32 modellként érhető el. Ez a modell általában pontosabb, mint az EfficientDet-Lite0, de lassabb és memóriaigényesebb is. Ez a modell olyan felhasználási esetekben megfelelő, ahol a pontosság nagyobb prioritást élvez a sebességgel és a mérettel szemben.

**4.3.RetinaNet**

A RetinaNet az egyik legjobb egylépcsős objektumfelismerő modell, amely bizonyítottan jól működik sűrű és kis méretű objektumok esetén. Emiatt vált népszerű objektumfelismerő modellé, amelyet légi és műholdas képeken használnak. A RetinaNet a meglévő egylépcsős objektumdetektálási modellek - Feature Pyramid Networks (FPN) és Focal Loss - két továbbfejlesztésével jött létre.

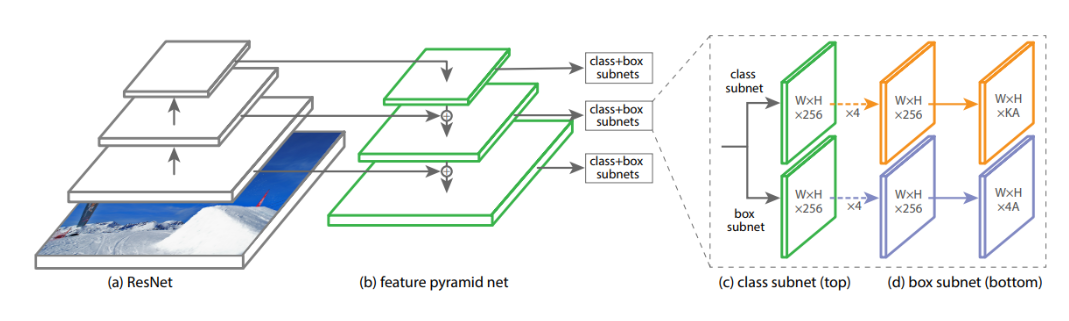
A RetinaNet modell architektúrájának négy fő összetevője van.

a) Bottom-up Pathway - A gerinchálózat (pl. ResNet), amely a bemeneti kép méretétől vagy a gerinchálózattól függetlenül különböző léptékben számítja ki a jellemzőtérképeket.

b) Top-down pathway és laterális kapcsolatok - A top-down pathway felfelé mintavételezi a magasabb piramisszintekről származó, térben durvább jellemzőtérképeket, a laterális kapcsolatok pedig egyesítik a top-down rétegeket és az azonos térbeli méretű bottom-up rétegeket.

c) Osztályozási alhálózat - Megjósolja annak valószínűségét, hogy egy objektum jelen van-e az egyes térbeli helyeken minden egyes horgonydoboz és objektumosztály esetében.

d) Regressziós alhálózat - Ez regresszi a határoló dobozok eltolódását a horgonydobozoktól minden egyes alapigazság objektumhoz.



## 4.4.Tensorflow

A TensorFlow egy ingyenes és nyílt forráskódú szoftverkönyvtár a gépi tanuláshoz és a mesterséges intelligenciához. Számos feladatban használható, de különös hangsúlyt fektet a mély neurális hálózatok képzésére és következtetésére.

A TensorFlow-t a Google Brain csapata fejlesztette ki a Google belső kutatási és termelési célú felhasználására. 2015-ben az Apache License 2.0 alatt adták ki a kezdeti változatot. 2019 szeptemberében a Google kiadta a TensorFlow frissített változatát TensorFlow 2.0 néven. A TensorFlow frissített változatát TensorFlow 2.0 néven adta ki.

A TensorFlow számos programozási nyelven használható, többek között Python, JavaScript, C++ és Java nyelven. Ez a rugalmasság számos különböző ágazatban alkalmazható.

Jellemzők:

AutoDifferenciálás

Az AutoDifferenciálás egy modell gradiens vektorának automatikus kiszámítása az egyes paraméterek tekintetében. Ezzel a funkcióval a TensorFlow képes automatikusan kiszámítani a modell paramétereinek gradienseit, ami hasznos az olyan algoritmusok számára, mint például a backpropagation, amelyeknek a teljesítmény optimalizálásához gradiensekre van szükségük. Ehhez a keretrendszernek nyomon kell követnie a modell bemeneti Tenzorjaival végzett műveletek sorrendjét, majd ki kell számolnia a megfelelő paraméterek tekintetében a gradienseket. A TensorFlow-nak a modell paramétereihez tartozó grádienseket kell kiszámítania.

Buzgó végrehajtás

A TensorFlow tartalmaz egy "buzgó végrehajtási" módot, ami azt jelenti, hogy a műveletek azonnal kiértékelésre kerülnek, szemben a később végrehajtott számítási gráfhoz való hozzáadással. A buzgón végrehajtott kód lépésről lépésre vizsgálható egy hibakeresőn keresztül, mivel az adatok a kód minden egyes sorában bővülnek, nem pedig később a számítási gráfban. Ezt a végrehajtási paradigmát könnyebbnek tartják a hibakeresés szempontjából, mivel lépésről lépésre átláthatóvá teszi. A hibakeresés könnyebbé válik.

Elosztás

A TensorFlow mind az eager, mind a gráfos végrehajtásokban biztosít egy API-t a számítások több eszközre történő elosztására különböző elosztási stratégiákkal. Ez az elosztott számítás gyakran felgyorsíthatja a TensorFlow modellek képzésének és kiértékelésének végrehajtását, és ez egy gyakori gyakorlat a mesterséges intelligencia területén.

Veszteségek

A modellek képzéséhez és értékeléséhez a TensorFlow egy sor veszteségfüggvényt (más néven költségfüggvényt) biztosít. Néhány népszerű példa: átlagos négyzetes hiba (MSE) és bináris keresztentrópia (BCE). Ezek a veszteségfüggvények a modell kimenete és a várt kimenet (tágabb értelemben két tenzor közötti különbség) "hibáját" vagy "különbségét" számítják ki. A különböző adatkészletek és modellek esetében különböző veszteségeket használnak a teljesítmény bizonyos szempontjainak előtérbe helyezésére.

Mérőszámok

A gépi tanulási modellek teljesítményének értékeléséhez a TensorFlow API hozzáférést biztosít az általánosan használt metrikákhoz. Ilyen például a különböző pontossági metrikák (bináris, kategorikus, ritka kategorikus), valamint más metrikák, mint például a Precision, Recall és Intersection-over-Union (IoU).

TF.nn

A TensorFlow.nn egy olyan modul, amely primitív neurális hálózati műveleteket hajt végre a modelleken. Néhány ilyen művelet a konvolúciók (1/2/3D, Atrous, depthwise), aktiválási függvények (Softmax, RELU, GELU, Sigmoid, stb.) és ezek variációi, valamint egyéb Tensor műveletek (max-pooling, bias-add, stb.).

Optimalizátorok

A TensorFlow optimalizátorok sorát kínálja a neurális hálózatok képzéséhez, köztük az ADAM, ADAGRAD és a sztochasztikus gradiens leszállás (SGD). A modell képzésénél a különböző optimalizátorok különböző paraméterhangolási módokat kínálnak, amelyek gyakran befolyásolják a modell konvergenciáját és teljesítményét.

## 4.5.Inception V3 model

Az Inception v3 egy konvolúciós neurális hálózat, amely képelemzésben és objektumfelismerésben nyújt segítséget, és a Googlenet egyik moduljaként indult. Ez a Google Inception konvolúciós neurális hálózatának harmadik kiadása, amelyet eredetileg az ImageNet felismerési verseny során mutattak be. Az Inceptionv3 tervezése arra irányult, hogy mélyebb hálózatokat tegyen lehetővé, ugyanakkor a paraméterek száma ne növekedjen túlságosan nagyra: "25 millió paraméter alatt van", szemben az AlexNet 60 milliójával.

Ahogyan az ImageNet az osztályozott vizuális objektumok adatbázisaként is felfogható, az Inception az objektumok osztályozását segíti a számítógépes látás világában. Az Inceptionv3 architektúrát számos különböző alkalmazásban újra felhasználták, gyakran az ImageNetből "előre betanított" rendszereket használva. Az egyik ilyen felhasználási terület az élettudományok, ahol a leukémia kutatását segíti.

## 4.6.ImageAI

A Python egy olyan könyvtárat kínál, amely a programozók és fejlesztők számára lehetővé teszi, hogy néhány sor egyszerű kódolási szkript segítségével önálló mélytanulási és számítógépes látás képességekkel rendelkező alkalmazásokat és rendszereket építsenek. Az ImageAI szinte az összes legkorszerűbb mélytanulási algoritmus, például a RetinaNet, a YOLOv3 és a TinyYOLOv3 Python implementációjából áll.

Az ImageAI számos offline működő API-t használ - rendelkezik objektumfelismerő, videófelismerő és objektumkövető API-kkal, amelyek az internet elérése nélkül is meghívhatók. Az ImageAI egy előre betanított modellt használ, és könnyen testre szabható.

Az ImageAI könyvtár ObjectDetection osztálya olyan metódusokból áll, amelyekkel bármilyen képen vagy képhalmazon objektumdetektálást lehet végezni az előre betanított modellek segítségével. Az ImageAI segítségével nyolcvan különböző típusú, hétköznapi, hétköznapi tárgyat tudunk felismerni és detektálni.

**5.Adatkészlet**

Az adathalmaz (vagy adatkészlet) adatok gyűjteménye. Táblázatos adatok esetében az adathalmaz egy vagy több adatbázis-táblának felel meg, ahol a táblázat minden oszlopa egy adott változót képvisel, és minden sor a kérdéses adathalmaz egy adott rekordjának felel meg. Az adathalmaz az adathalmaz minden egyes tagjára vonatkozóan felsorolja az egyes változók értékeit, mint például egy tárgy magassága és súlya. Az adathalmazok dokumentumok vagy fájlok gyűjteményéből is állhatnak.

A nyílt adatok tudományágában az adathalmaz a nyilvános nyílt adattárban közzétett információk mértékegysége. Az európai data.europa.eu portál több mint egymillió adathalmazt aggregál. Néhány más kérdés (valós idejű adatforrások, nem relációs adathalmazok stb.) növeli a konszenzus elérésének nehézségeit.

Számos jellemző határozza meg egy adathalmaz szerkezetét és tulajdonságait. Ezek közé tartozik az attribútumok vagy változók száma és típusa, valamint a rájuk vonatkozó különböző statisztikai mérőszámok, mint például a szórás és a kurtózis.

Az értékek lehetnek számok, például valós számok vagy egész számok, például egy személy magasságát centiméterben kifejezve, de lehetnek nominális (azaz nem számértékekből álló) adatok is, például egy személy etnikai hovatartozását ábrázolva. Általánosabban fogalmazva, az értékek lehetnek a mérési szintként leírtak bármelyik fajtája. Az egyes változók esetében az értékek általában mind azonos típusúak. Lehetnek azonban hiányzó értékek is, amelyeket valamilyen módon jelezni kell.

A statisztikában az adathalmazok általában egy statisztikai sokaság mintavételezésével nyert tényleges megfigyelésekből származnak, és minden sor a sokaság egy elemére vonatkozó megfigyeléseknek felel meg. Az adathalmazokat továbbá algoritmusok is létrehozhatják bizonyos típusú szoftverek tesztelése céljából. Egyes modern statisztikai elemző szoftverek, mint például az SPSS, még mindig a klasszikus adathalmazok formájában mutatják be az adataikat. Hiányzó vagy gyanús adatok esetén imputációs módszer használható az adathalmaz kiegészítésére.

Többféle előre tanított (pre-trained) és egyéni adatkészlet lett felhasználva a programok tesztelésére.

[Animal Kingdom Dataset](https://sutdcv.github.io/Animal-Kingdom)

Az Animal Kingdom egy nagy és változatos adathalmaz, amely több kommentált feladatot kínál a természetes állati viselkedés alaposabb megértéséhez. Az adathalmazban használt vadon élő állatokról készült felvételek a nap különböző időpontjait rögzítik a környezetek széles skáláján, amelyek a háttér, a nézőpontok, a megvilágítás és az időjárási viszonyok változatosságát tartalmazzák. Pontosabban, az adatkészlet 50 órányi annotált videót tartalmaz a releváns állati viselkedési szegmensek lokalizálására hosszú videókban a videó-alapozási feladathoz, 30K videósorozatot a finom szemcseméretű, több címkés cselekvésfelismerési feladathoz, és 33K képkockát a pózbecslési feladathoz, amelyek az állatok változatos skálájának felelnek meg, 850 fajjal, 6 fő állatosztályban.

**6.Gyakorlati alkalmazás**

A korábban tárgyaltak működését néhány gyakorlati projekttel szeretném szemléltetni.

6.1. MediaPipe Object Detector

A MediaPipe Object Detector feladat lehetővé teszi több objektumosztály jelenlétének és helyének észlelését. Ezek az utasítások megmutatják, hogyan használhatja az Object Detector feladatot Pythonban.

### 6.1.1.Jellemzők

Bemeneti képfeldolgozás - A feldolgozás magában foglalja a kép elforgatását, átméretezését, normalizálását és színtérkonvertálását.

Címketérkép helymeghatározás - A megjelenített nevek nyelvének beállítása.

Pontszámküszöb - Az eredmények szűrése a predikciós pontszámok alapján.

Top-k felismerés - A számfelismerési eredmények szűrése.

Címkeengedélyezési lista és címkeelutasítási lista - Az észlelt kategóriák megadása.

Csomagok

Az Object Detector feladathoz a mediapipe pip csomag szükséges. A szükséges csomagokat a következő parancsokkal telepítheti:

$ python -m pip install mediapipe

| **Task inputs** | **Task outputs** |
| --- | --- |
| * Az objektumdetektor API a következő adattípusok egyikét fogadja el: * - Állóképek * - dekódolt videoképkockák * - Élő videótovábbítás | * Az objektumdetektor API a következőket adja ki * eredményeket az észlelt objektumokra vonatkozóan: * - Az objektum kategóriája * - Valószínűségi pontszám * - Határoló doboz koordinátái |

### 6.1.2.Modell

Válasszon ki és töltsön le egy modellt, majd tárolja azt egy helyi könyvtárban:

model\_path = '/absolute/path/to/lite-model\_efficientdet\_lite0\_detection\_metadata\_1.tflite'

A BaseOptions objektum model\_asset\_path paraméterével adhatja meg a modell elérési útvonalát.

### 6.1.3.Modell referenciaértékek

Itt vannak a fenti, előre betanított modellek referenciaértékei. A késleltetési eredmény a Pixel 6 és a CPU 1 szálon készült.

| **Model Name** | **Model size** | **MAP (mean average precision)** | **CPU Latency** | **GPU Latency** |
| --- | --- | --- | --- | --- |
| EfficientDet-Lite0 float32 model | 14M | 24.38% | 51.11ms | 33.01ms |
|  |  |  |  |  |
| EfficientDet-Lite0 float16 model | 6.6M | 24.37% | 48.99ms | 33.13ms |
|  |  |  |  |  |
| EfficientDet-Lite0 int8 model | 4.4M | 24.08% | 27.8ms | - |
|  |  |  |  |  |
| EfficientDet-Lite2 float32 model | 21.7M | 31.70% | 158.51ms | 56.62ms |
|  |  |  |  |  |
| EfficientDet-Lite2 float16 model | 11M | 31.67% | 162.91ms | 57.00ms |
|  |  |  |  |  |
| EfficientDet-Lite2 int8 model | 7.2M | 31.21% | 82.45ms | - |
|  |  |  |  |  |
| SSD MobileNetV2 float model | 11M | 21.20% | 31.26ms | 24.55ms |
|  |  |  |  |  |
| SSD MobileNetV2 int8 model | 3.3M | 20.10% | 16.72ms | - |

### 6.1.4.Modellkövetelmények és metaadatok

Ez a szakasz az egyéni modellekkel szemben támasztott követelményeket ismerteti, ha úgy dönt, hogy modellt készít a feladathoz. Az egyéni modelleknek TensorFlow Lite formátumúaknak kell lenniük, és tartalmazniuk kell a modell működési paramétereit leíró metaadatokat.

#### Tervezési követelmények

| **Input** | **Shape** | **Description** |
| --- | --- | --- |
| Input image | Float32 tensor of shape[1, height, width, 3] | A normalizált bemeneti kép. |
| **Output** | **Shape** | **Description** |
| detection\_boxes | Float32 tensor of shape [1, num\_boxes, 4] | Az egyes észlelt objektumok dobozhelyzete. |
| detection\_classes | Float32 tensor of shape [1, num\_boxes] | Az egyes észlelt objektumok osztályneveinek indexei. |
| detection\_scores | float32 tensor of shape [1, num\_boxes] | Előrejelzési pontszámok minden egyes észlelt  objektumhoz. |
| num\_boxes | Float32 tensor of size 1 | Az észlelt dobozok száma. |

### 6.1.5.Metadata követelmények

| **Parameter** | **Description** | **Description** |
| --- | --- | --- |
| input\_norm\_mean | A bemeneti tenzor normalizálásához használt középérték. | A normalizált bemeneti kép. |
| input\_norm\_std | A bemeneti tenzor normalizálásához használt mező normája. | Az egyes észlelt objektumok dobozhelyzete. |
| label\_file\_paths | A kategória tenzorcímkefájlok elérési útvonalai. Ha a modellnek nincsenek címkefájljai, adjon meg egy üres listát. | Az egyes észlelt osztályok neveinek indexei  objektumhoz. |
| score\_calibration\_md | Az osztályozási tenzorban végzett pontszám-kalibrálási műveletre vonatkozó információk. Ez a paraméter nem szükséges, ha a modell nem használja a pontszámok  kalibrálást. | Előrejelzési pontszámok minden egyes észlelt objektumhoz. |
| num\_boxes | Float32 1 méretű tenzor | Az észlelt dobozok száma. |

### 6.1.6.Project létrehozása

A feladat létrehozásához használja a create\_from\_options függvényt. A create\_from\_options függvény olyan konfigurációs beállításokat fogad el, mint a futtatási mód, a megjelenített nevek helyi nyelve, az eredmények maximális száma, a megbízhatósági küszöb, a kategória engedélyezési listája és a tiltási lista. Ha nem állít be konfigurációs opciót, a feladat az alapértelmezett értéket használja.

Az Objektumdetektor feladat többféle bemeneti adattípust támogat: állóképek, videofájlok és élő videofolyamok.

import mediapipe as mp  
  
BaseOptions = mp.tasks.BaseOptions  
ObjectDetector = mp.tasks.vision.ObjectDetector  
ObjectDetectorOptions = mp.tasks.vision.ObjectDetectorOptions  
VisionRunningMode = mp.tasks.vision.RunningMode  
  
options = ObjectDetectorOptions(  
    base\_options=BaseOptions(model\_asset\_path='/path/to/model.tflite'),  
    max\_results=5,  
    running\_mode=VisionRunningMode.VIDEO)  
  
with ObjectDetector.create\_from\_options(options) as detector:  
  # The detector is initialized. Use it here.  
  # ...

### 6.1.7.Konfigurációs lehetőségek

Ez a feladat a következő konfigurációs lehetőségekkel rendelkezik Python alkalmazásokhoz:

| **Option Name** | **Description** | **Value Range** | **Default Value** |
| --- | --- | --- | --- |
| running\_mode | A feladat futási módjának beállítása. Az objektumdetektornak három üzemmódja van:  KÉP: Az objektumok felismerésének üzemmódja egyetlen kép bemenetén.  VIDEÓ: A videó dekódolt képkockáin lévő objektumok felismerésének üzemmódja.  LIVE\_STREAM: A bemeneti adatok élő adatfolyamán, például egy kamerából származó objektumok észlelésére szolgáló üzemmód. Ebben az üzemmódban az resultListener-t meg kell hívni egy olyan figyelő beállításához, amely aszinkron módon fogadja az észlelési eredményeket. | {IMAGE, VIDEO, LIVE\_STREAM} | IMAGE |
| display\_names | Beállítja a feladat modelljének metaadataiban megadott megjelenítési nevekhez használandó címkék nyelvét, ha rendelkezésre áll. Az alapértelmezett érték en az angol nyelvre. A TensorFlow Lite Metadata Writer API segítségével helyi címkéket adhat hozzá egy egyéni modell metaadataihoz. | Lokális kód | en |
| max\_results | Beállítja a legmagasabb pontszámú észlelési eredmények opcionális maximális számát. | Bármilyen pozitív szám | -1 (all results are returned) |
| score\_threshold | Beállítja az előrejelzési pontszám küszöbértékét, amely felülírja a modell metaadataiban megadott küszöbértéket (ha van ilyen). Az ezen érték alatti eredmények elutasításra kerülnek. | Bármilyen float | Not set |
| category\_allow\_list | Beállítja a megengedett kategórianevek opcionális listáját. Ha nem üres, akkor a rendszer kiszűri azokat az észlelési eredményeket, amelyek kategórianeve nem szerepel ebben a listában. A duplikált vagy ismeretlen kategórianevek figyelmen kívül maradnak. Ez az opció és a category\_deny\_list kölcsönösen kizárja egymást, és mindkettő használata hibát eredményez. | Bármilyen string | Not set |
| category\_deny\_list | Beállítja a nem engedélyezett kategórianevek opcionális listáját. Ha nem üres, akkor a rendszer kiszűri azokat az észlelési eredményeket, amelyek kategórianeve szerepel ebben a halmazban. A duplikált vagy ismeretlen kategórianevek figyelmen kívül maradnak. Ez az opció és a category\_allow\_list kölcsönösen kizárja egymást, és mindkettő használata hibát eredményez. | Bármilyen string | Not set |

### 6.1.8.Adatok előkészítése

Készítse elő a bemenetet képfájlként vagy numpy tömbként, majd konvertálja azt mediapipe.Image objektummá. Ha a bemenet egy videófájl vagy egy webkameráról származó élő stream, akkor egy külső könyvtárat, például az OpenCV-t használhatja a bemeneti képkockák numpy tömbként történő betöltéséhez.

A következő példák elmagyarázzák és bemutatják, hogyan kell előkészíteni az adatokat a feldolgozáshoz az egyes rendelkezésre álló adattípusok esetében:

import mediapipe as mp  
  
# Use OpenCV’s VideoCapture to load the input video.  
  
# Load the frame rate of the video using OpenCV’s CV\_CAP\_PROP\_FPS  
# You’ll need it to calculate the timestamp for each frame.  
  
# Loop through each frame in the video using VideoCapture#read()  
  
# Convert the frame received from OpenCV to a MediaPipe’s Image object.  
mp\_image = mp.Image(image\_format=mp.ImageFormat.SRGB, data=numpy\_frame\_from\_opencv)

### 6.1.9.A projekt futtatása

A következtetések kiváltásához meghívhatja a detektáló függvények egyikét. Az Object Detector feladat visszaadja a bemeneti képen vagy képkockán belül észlelt objektumokat.

# Calculate the timestamp of the current frame  
frame\_timestamp\_ms = 1000 \* frame\_index / video\_file\_fps  
  
# Perform object detection on the video frame.  
detection\_result = detector.detect\_for\_video(mp\_image, frame\_timestamp\_ms)

Vegye figyelembe a következőket:

Ha videó vagy élő stream üzemmódban fut, akkor a bemeneti képkocka időbélyegzőjét is meg kell adnia az Objektumdetektor feladatnak.

A kép- vagy videomodellben történő futtatáskor az Object Detector task blokkolja az aktuális szálat, amíg az be nem fejezi a bemeneti kép vagy képkocka feldolgozását.

Amikor az objektumdetektor feladat élő stream üzemmódban fut, az objektumdetektor feladat nem blokkolja az aktuális szálat, hanem azonnal visszatér. Minden alkalommal, amikor befejezte egy bemeneti képkocka feldolgozását, meghívja az eredményhallgatóját az észlelés eredményével. Ha a detektáló függvényt akkor hívja meg, amikor az Object Detector task éppen egy másik képkocka feldolgozásával van elfoglalva, az új bemeneti képkockát figyelmen kívül hagyja.

### 6.1.10.Az eredmények kezelése és megjelenítése

A következtetés lefutása után az Object Detector feladat egy ObjectDetectionResult objektumot ad vissza, amely leírja a bemeneti képen talált objektumokat.

Az alábbiakban egy példa látható a feladat kimeneti adataira:

ObjectDetectionResult:  
 Detection #0:  
  Box: (x: 355, y: 133, w: 190, h: 206)  
  Categories:  
   index       : 17  
   score       : 0.73828  
   class name  : dog  
 Detection #1:  
  Box: (x: 103, y: 15, w: 138, h: 369)  
  Categories:  
   index       : 17  
   score       : 0.73047  
   class name  : dog

**6.2.ImageAI**

Az ImageAI kényelmes, rugalmas és hatékony módszereket kínál a videókon történő objektumfelismeréshez. A rendelkezésre bocsátott videóobjektum-felismerő osztály csak a RetinaNet, a YOLOv3 és a TinyYOLOv3 modelleket támogatja. Az ImageAI ezen verziója kereskedelmi szintű videoobjektum-felismerési funkciókat biztosít, amelyek többek között, de nem kizárólagosan eszköz/IP-kamera bemenetek, képkockánként, másodpercenként, percenként és teljes videóelemzésre terjednek ki az adatbázisokban és/vagy valós idejű vizualizációkban való tároláshoz és a jövőbeli meglátásokhoz.

### 6.2.1.Projekt létrehozása

Miután letöltötte az objektumfelismerő modellfájlt, másolja a modellfájlt a projekt mappájába, ahol a .py fájlok lesznek. Ezután hozzon létre egy python fájlt, és adjon neki egy nevet; egy példa erre a FirstVideoObjectDetection.py. Ezután írja be az alábbi kódot a python fájlba:

**FirstVideoObjectDetection.py**

from imageai.Detection import VideoObjectDetection

import os

execution\_path = os.getcwd()

detector = VideoObjectDetection()

detector.setModelTypeAsRetinaNet()

detector.setModelPath( os.path.join(execution\_path , "retinanet\_resnet50\_fpn\_coco-eeacb38b.pth"))

detector.loadModel()

video\_path = detector.detectObjectsFromVideo(input\_file\_path=os.path.join(execution\_path, "traffic.mp4"),

output\_file\_path=os.path.join(execution\_path, "traffic\_detected")

, frames\_per\_second=20, log\_progress=True)

print(video\_path)

Végezzük el a fent használt objektumfelismerő kód lebontását.

from imageai.Detection import VideoObjectDetection

import os

execution\_path = os.getcwd()

A fenti 3 sorban , az első sorban importáljuk az \*\*ImageAI video object detection\*\* osztályt, a második sorban importáljuk az os-t, és megkaptuk a mappa elérési útját, ahol a python fájlunk fut.

### 6.2.2.Modell betöltése

detector = VideoObjectDetection()

detector.setModelTypeAsRetinaNet()

detector.setModelPath( os.path.join(execution\_path , "retinanet\_resnet50\_fpn\_coco-eeacb38b.pth"))

detector.loadModel()

A fenti 4 sorban az első sorban létrehoztuk a VideoObjectDetection osztály új példányát, a második sorban a modell típusát RetinaNet-re állítottuk, a harmadik sorban a modell elérési útját a letöltött és a python fájlmappába másolt RetinaNet modellfájlra állítottuk, a negyedik sorban pedig betöltöttük a modellt.

video\_path = detector.detectObjectsFromVideo(input\_file\_path=os.path.join(execution\_path, "traffic.mp4"),

output\_file\_path=os.path.join(execution\_path, "traffic\_detected"),

frames\_per\_second=20, log\_progress=True)

print(video\_path)

A fenti 2 sorban futtattuk a detectObjectsFromVideo() függvényt, és elemezzük a videó elérési útvonalát,az új videó elérési útvonalát (kiterjesztés nélkül, alapértelmezés szerint .avi videót ment), amelyet a függvény el fog menteni, a másodpercenkénti képkockák számát (fps), amelyet a kimeneti videónak kell lennie, és a konzolban a felismerés előrehaladásának naplózását. Ezután a függvény visszaadja a mentett videó elérési útvonalát, amely tartalmazza a videóban észlelt objektumok dobozait és százalékos valószínűségét.

### 6.2.3.Egyéni videoobjektum-érzékelés

Az ImageAI által támogatott videoobjektum-felismerő modell (RetinaNet) 80 különböző típusú objektumot képes felismerni. Ezek közé tartoznak:

person, bicycle, car, motorcycle, airplane, bus, train, truck, boat, traffic light, fire hydrant, stop\_sign,

parking meter, bench, bird, cat, dog, horse, sheep, cow, elephant, bear, zebra,

giraffe, backpack, umbrella, handbag, tie, suitcase, frisbee, skis, snowboard,

sports ball, kite, baseball bat, baseball glove, skateboard, surfboard, tennis racket,

bottle, wine glass, cup, fork, knife, spoon, bowl, banana, apple, sandwich, orange,

broccoli, carrot, hot dog, pizza, donot, cake, chair, couch, potted plant, bed,

dining table, toilet, tv, laptop, mouse, remote, keyboard, cell phone, microwave,

oven, toaster, sink, refrigerator, book, clock, vase, scissors, teddy bear, hair dryer,

toothbrush.

Érdekes, hogy az ImageAI lehetővé teszi, hogy a fenti elemek közül egy vagy több felismerést végezzen. Ez azt jelenti, hogy testre szabhatja, hogy milyen típusú tárgy(ak)at szeretne észlelni a videóban. Nézzük meg az alábbi kódot:

from imageai.Detection import VideoObjectDetection

import os

execution\_path = os.getcwd()

detector = VideoObjectDetection()

detector.setModelTypeAsRetinaNet()

detector.setModelPath( os.path.join(execution\_path , "retinanet\_resnet50\_fpn\_coco-eeacb38b.pth"))

detector.loadModel()

custom\_objects = detector.CustomObjects(person=True, bicycle=True, motorcycle=True)

video\_path = detector.detectCustomObjectsFromVideo(

custom\_objects=custom\_objects,

input\_file\_path=os.path.join(execution\_path, "traffic.mp4"),

output\_file\_path=os.path.join(execution\_path, "traffic\_custom\_detected"),

frames\_per\_second=20, log\_progress=True)

print(video\_path)

Nézzük meg a kódnak azt a részét, amely ezt lehetővé tette.

custom\_objects = detector.CustomObjects(person=True, bicycle=True, motorcycle=True)

video\_path = detector.detectCustomObjectsFromVideo(

custom\_objects=custom\_objects,

input\_file\_path=os.path.join(execution\_path, "traffic.mp4"),

output\_file\_path=os.path.join(execution\_path, "traffic\_custom\_detected"),

frames\_per\_second=20, log\_progress=True)

A fenti kódban a modell betöltése után (a modell betöltése előtt is megtehetjük) definiáltunk egy új változót custom\_objects = detector.CustomObjects(), amelyben a person, car és motorcycle tulajdonságait True értékre állítjuk. Ezzel azt mondjuk a modellnek, hogy csak az általunk True értékre beállított objektumokat érzékelje. Ezután meghívjuk a detector.detectCustomObjectsFromVideo() függvényt, amely lehetővé teszi számunkra az egyéni objektumok felismerését. Ezután a custom\_objects értéket az általunk definiált custom objects változóra állítjuk.

### 6.2.4.Videó elemzés

Az ImageAI mostantól kereskedelmi szintű videóelemzést kínál a Video Object Detection osztályban, mind a videófájlok, mind a kamerák bemeneteihez. Ez a funkció lehetővé teszi a fejlesztők számára, hogy mély betekintést nyerjenek az ImageAI-val feldolgozott videókba. Ezek a meglátások valós időben megjeleníthetők, és egy NoSQL adatbázisban tárolhatók a későbbi felülvizsgálat vagy elemzés céljából.

A videóelemzéshez a detectObjectsFromVideo() és detectCustomObjectsFromVideo() mostantól lehetővé teszi, hogy megadhassa a saját definiált függvényeit, amelyek az észlelt videó minden egyes képkockájára, másodpercére és/vagy percére végrehajtásra kerülnek, valamint egy olyan függvényt, amely a videó észlelés végén kerül végrehajtásra. Miután ezeket a függvényeket megadták, nyers, de átfogó elemzési adatokat kapnak a képkocka/másodperc/perc indexéről, az észlelt objektumokról (név, percentage\_probability és box\_points), az egyes észlelt egyedi objektumok példányainak számáról és az egyes észlelt egyedi objektumok átlagos előfordulási számáról egy másodperc/perc és a teljes videó során.

A videóelemzéshez mindössze annyit kell tennie, hogy megadja a függvényt, megadja a megfelelő paramétereket, amelyeket kapni fog, és a függvény nevét a per\_frame\_function, per\_second\_function, per\_minute\_function és video\_complete\_function paraméterekbe elemzi az észlelési függvényben. Az alábbiakban példákat talál a videóelemző függvényekre.

def forFrame(frame\_number, output\_array, output\_count):

print("FOR FRAME " , frame\_number)

print("Output for each object : ", output\_array)

print("Output count for unique objects : ", output\_count)

print("------------END OF A FRAME --------------")

def forSeconds(second\_number, output\_arrays, count\_arrays, average\_output\_count):

print("SECOND : ", second\_number)

print("Array for the outputs of each frame ", output\_arrays)

print("Array for output count for unique objects in each frame : ", count\_arrays)

print("Output average count for unique objects in the last second: ", average\_output\_count)

print("------------END OF A SECOND --------------")

def forMinute(minute\_number, output\_arrays, count\_arrays, average\_output\_count):

print("MINUTE : ", minute\_number)

print("Array for the outputs of each frame ", output\_arrays)

print("Array for output count for unique objects in each frame : ", count\_arrays)

print("Output average count for unique objects in the last minute: ", average\_output\_count)

print("------------END OF A MINUTE --------------")

video\_detector = VideoObjectDetection()

video\_detector.setModelTypeAsYOLOv3()

video\_detector.setModelPath(os.path.join(execution\_path, "yolov3.pt"))

video\_detector.loadModel()

video\_detector.detectObjectsFromVideo(

input\_file\_path=os.path.join(execution\_path, "traffic.mp4"),

output\_file\_path=os.path.join(execution\_path, "traffic\_detected"),

frames\_per\_second=10,

per\_second\_function=forSeconds,

per\_frame\_function=forFrame,

per\_minute\_function=forMinute,

minimum\_percentage\_probability=30

)

Minden olyan függvény esetében, amelyet a per\_frame\_functionba elemezel, a függvény minden egyes videokép feldolgozása után végrehajtásra és elemzésre kerül.

Frame Index: Ez a képkocka pozíciójának száma a videón belül (pl. 1 az első képkocka és 20 a huszadik képkocka).

Kimeneti tömb: Ez egy szótárakból álló tömb. Minden szótár megfelel a képen található minden egyes felismert objektumnak, és tartalmazza az objektum "name", "percentage\_probabaility" és "box\_points"(x1,y1,x2,y2) értékeit.

Kimeneti szám: Ez egy szótár, amelynek kulcsa minden egyes felismert objektum neve, értékei pedig a felismert objektumok példányainak száma.

A fenti eredményben a videó feldolgozása és mentése 10 képkocka/másodperc (FPS) sebességgel történt. A per\_second\_functionba elemzett bármelyik függvény esetében a függvény a feldolgozott videó minden egyes másodperce után végrehajtásra és elemzésre kerül.

Második index: Ez a másodperc pozíciójának száma a videón belül (pl. 1 az első másodperchez és 20 a huszadik másodperchez).

Kimeneti tömb: A fenti példában 10 tömb van, amely megfelel az egy másodpercben található 10 képkockának). Minden egyes tartalmazott tömb szótárakat tartalmaz. Minden szótár megfelel a képen található minden egyes felismert objektumnak, és tartalmazza az objektum "neve", "százalékos valószínűsége" és "box\_points" (x1,y1,x2,y2) értékeit.

Számtáblák: Ez egy szótárakból álló tömb. Minden szótár és annak pozíciója (tömbindex + 1) megfelel a videó utolsó másodpercének megfelelő képkockának. Minden egyes szótár kulcsként tartalmazza minden egyes észlelt egyedi objektum nevét, értékként pedig az észlelt objektumok példányainak számát.

Átlagos kimeneti szám: Ez egy olyan szótár, amelynek kulcsai az utolsó másodpercben észlelt minden egyes egyedi objektum neve, értékei pedig az észlelt objektumok példányainak átlagos száma a képkockák számában.

A perc funkció eredményei A fenti 4 paraméter, amelyek a feldolgozott videó minden másodpercére visszakerülnek, ugyanazok a paraméterek, amelyek a feldolgozott videó minden percére visszakerülnek. A különbség az, hogy a visszaadott index megfelel a percindexnek, az output\_arrays egy tömb, amely az FPS \* 60 tömb számát tartalmazza (a fenti kódpéldában 10 képkocka másodpercenként(fps) \* 60 másodperc = 600 képkocka = 600 tömb), a count\_arrays pedig egy olyan tömb, amely az FPS \* 60 szótárak számát tartalmazza (a fenti kódpéldában 10 képkocka másodpercenként(fps) \* 60 másodperc = 600 képkocka = 600 szótár), az average\_output\_count pedig egy olyan szótár, amely az elmúlt percben található összes képkockában észlelt összes objektumot tartalmazza.

A Video Complete funkció eredményei Az ImageAI lehetővé teszi, hogy a teljes feldolgozott videó teljes elemzését megkapja. Mindössze egy olyan függvényt kell definiálnia, mint a forSecond vagy forMinute függvény, és a video\_complete\_function paramétert kell beállítania a .detectObjectsFromVideo() vagy .detectCustomObjectsFromVideo() függvénybe. A per\_second-funkció és a per\_minute\_funkció ugyanazokat az értékeket fogja visszaadni. A különbség az, hogy a funkció nem ad vissza indexet, hanem a másik 3 értéket adja vissza, és a 3 érték a videó összes képkockájára kiterjed. Az alábbiakban egy mintafunkciót mutatunk be:

def forFull(output\_arrays, count\_arrays, average\_output\_count):

#Perform action on the 3 parameters returned into the function

video\_detector.detectObjectsFromVideo(

input\_file\_path=os.path.join(execution\_path, "traffic.mp4"),

output\_file\_path=os.path.join(execution\_path, "traffic\_detected"),

frames\_per\_second=10,

video\_complete\_function=forFull,

minimum\_percentage\_probability=30

)

VÉGREHAJTÁS A VIDEÓANALÍZISRÓL : Az ImageAI lehetővé teszi, hogy az észlelt videóképkockát Numpy tömbként kapja meg minden egyes képkocka, másodperc és perc funkcióban. Mindössze egy további paramétert kell megadni a függvényben, és a return\_detected\_frame=True értéket kell beállítani a detectObjectsFromVideo() vagy detectCustomObjectsFrom() függvényben. Ha ez be van állítva, akkor a függvényedben külön megadott extra paraméter lesz a detektált képkocka Numpy tömbje. Lásd az alábbi mintát:

def forFrame(frame\_number, output\_array, output\_count, detected\_frame):

print("FOR FRAME " , frame\_number)

print("Output for each object : ", output\_array)

print("Output count for unique objects : ", output\_count)

print("Returned Objects is : ", type(detected\_frame))

print("------------END OF A FRAME --------------")

video\_detector.detectObjectsFromVideo(

input\_file\_path=os.path.join(execution\_path, "traffic.mp4"),

output\_file\_path=os.path.join(execution\_path, "traffic\_detected"),

frames\_per\_second=10,

per\_frame\_function=forFrame,

minimum\_percentage\_probability=30,

return\_detected\_frame=True

)

### 6.2.5.Képkocka észlelési intervallumok

A fenti videoobjektum-felismerési feladatot a képkocka valós idejű objektumfelismerésre optimalizálták, ami biztosítja, hogy a videó minden egyes képkockájában felismerjük az objektumokat. Az ImageAI lehetőséget biztosít a videóképkocka észlelés beállítására, ami felgyorsíthatja a videó észlelési folyamatot. A .detectObjectsFromVideo() vagy .detectCustomObjectsFromVideo() meghívásakor megadhatja, hogy a detektálást milyen képkocka intervallumban végezze el. A frame\_detection\_interval paraméter 5 vagy 20 értékűre állításával ez azt jelenti, hogy a videóban lévő objektumdetektálások 5 vagy 20 képkocka után frissülnek. Ha a kimeneti videó frames\_per\_second értéke 20, ez azt jelenti, hogy a videóban lévő objektumdetektálások negyed másodpercenként vagy másodpercenként egyszer frissülnek. Ez olyan esetekben hasznos, amikor a rendelkezésre álló számítási kapacitás kisebb, és a mozgó objektumok sebessége alacsony. Ez biztosítja, hogy az objektumok észlelése másodperces valós idejű , fél másodperces valós idejű vagy az Ön igényeinek megfelelő módon történjen. A videoobjektum-érzékelést ugyanazon a bemeneti videón végeztük el, amelyet mindvégig használtunk, 5 értékű frame\_detection\_interval értéket alkalmazva.

###Videó észlelési időkorlát

Az ImageAI mostantól lehetővé teszi, hogy másodpercben megadjon egy időkorlátot a videókban vagy a kamera élő adásában lévő objektumok észlelésére. A videóérzékelési kódodban az időkorlát beállításához csak annyit kell tenned, hogy a detectObjectsFromVideo() függvényben a detection\_timeout paramétert a kívánt másodpercek számára állítod be. Az alábbi példakódban a detection\_timeout paramétert 120 másodpercre (2 perc) állítjuk be.

from imageai.Detection import VideoObjectDetection

import os

import cv2

execution\_path = os.getcwd()

camera = cv2.VideoCapture(0)

detector = VideoObjectDetection()

detector.setModelTypeAsRetinaNet()

detector.setModelPath(os.path.join(execution\_path , "retinanet\_resnet50\_fpn\_coco-eeacb38b.pth"))

detector.loadModel()

video\_path = detector.detectObjectsFromVideo(camera\_input=camera,

output\_file\_path=os.path.join(execution\_path, "camera\_detected\_video"),

frames\_per\_second=20,

log\_progress=True,

minimum\_percentage\_probability=40,

detection\_timeout=120)

**6.3.Implementing Real**

Valós idejű objektum detektálást végrehajtó program PyTorch alapon és YOLO v5 modellt használva.

### 6.3.1. A modell betöltése

A modellt közvetlenül a PyTorch hubból tölthetjük be, a kód első futtatásakor néhány percig tarthat, mivel a modell letöltődik az internetről, de a következő alkalommal már közvetlenül a lemezről töltődik be.

from torch import hub # Hub contains other models like FasterRCNNmodel = torch.hub.load( \  
 'ultralytics/yolov5', \  
 'yolov5s', \  
 pretrained=True)

### 6.3.2.Egyetlen képkocka pontozása

A következtetés elvégzéséhez használt eszköz óriási különbséget jelent a következtetés sebességében, a modern mélytanulási modellek akkor működnek a legjobban, ha GPU-kkal dolgozunk, így ha van egy CUDA kernellel rendelkező GPU-nk, az hatalmas mértékben javítja a teljesítményt. Tapasztalataim szerint a rendszer akár egyetlen GPU-val is képes 45-60 képkockát elérni másodpercenként, míg egy CPU legfeljebb 25-30 képkockát adhat.

"""  
The function below identifies the device which is availabe to make the prediction and uses it to load and infer the frame. Once it has results it will extract the labels and cordinates(Along with scores) for each object detected in the frame.  
"""def score\_frame(frame, model):  
 device = 'cuda' if torch.cuda.is\_available() else 'cpu'  
 model.to(device)  
 frame = [torch.tensor(frame)]  
 results = self.model(frame)  
 labels = results.xyxyn[0][:, -1].numpy()  
 cord = results.xyxyn[0][:, :-1].numpy()  
 return labels, cord

### 6.3.3.Dobozok kirajzolása a keret fölé

Miután kiértékeltük a keretet, az azonosított objektumokat és dobozaikat a keret fölé kell rajzolnunk, mielőtt a keretet a kimeneti folyamba írnánk. Ehhez használhatjuk az OpenCV képfeldolgozó eszközkészletét.

"""  
The function below takes the results and the frame as input and plots boxes over all the objects which have a score higer than our threshold.  
"""  
def plot\_boxes(self, results, frame):  
labels, cord = results  
n = len(labels)  
x\_shape, y\_shape = frame.shape[1], frame.shape[0]  
for i in range(n):  
row = cord[i]  
# If score is less than 0.2 we avoid making a prediction.  
if row[4] < 0.2:   
continue  
x1 = int(row[0]\*x\_shape)  
y1 = int(row[1]\*y\_shape)  
x2 = int(row[2]\*x\_shape)  
y2 = int(row[3]\*y\_shape)  
bgr = (0, 255, 0) # color of the box  
classes = self.model.names # Get the name of label index  
label\_font = cv2.FONT\_HERSHEY\_SIMPLEX #Font for the label.  
cv2.rectangle(frame, \  
(x1, y1), (x2, y2), \  
bgr, 2) #Plot the boxes  
cv2.putText(frame,\  
classes[labels[i]], \  
(x1, y1), \  
label\_font, 0.9, bgr, 2) #Put a label over box.  
return frame

**6.3.4.Összegzés**

Hozzunk létre egy videó streaming bemenetet.

Töltsük be a modellt.

Amíg a bemenet elérhető, olvassuk be a következő képkockát.

Pontozzuk a képkockát, hogy megkapjuk a címkéket és a koordinátákat.

Rajzolja a dobozokat az észlelt objektumok fölé.

Írja a feldolgozott képkockát a kimeneti videófolyamra.

Hat egyszerű lépés a működéshez, bár hozzáadunk néhány infrastrukturális kódot, hogy az alkalmazást robusztusabbá tegyük, az alapok ugyanazok. Tehát csináljuk meg.

"""  
The Function below oracestrates the entire operation and performs the real-time parsing for video stream.  
"""  
def \_\_call\_\_(self):  
player = self.get\_video\_stream() #Get your video stream.  
assert player.isOpened() # Make sure that their is a stream.   
#Below code creates a new video writer object to write our  
#output stream.  
x\_shape = int(player.get(cv2.CAP\_PROP\_FRAME\_WIDTH))  
y\_shape = int(player.get(cv2.CAP\_PROP\_FRAME\_HEIGHT))  
four\_cc = cv2.VideoWriter\_fourcc(\*"MJPG") #Using MJPEG codex  
out = cv2.VideoWriter(out\_file, four\_cc, 20, \  
(x\_shape, y\_shape))   
ret, frame = player.read() # Read the first frame.  
while rect: # Run until stream is out of frames  
start\_time = time() # We would like to measure the FPS.  
results = self.score\_frame(frame) # Score the Frame  
frame = self.plot\_boxes(results, frame) # Plot the boxes.  
end\_time = time()  
fps = 1/np.round(end\_time - start\_time, 3) #Measure the FPS.  
print(f"Frames Per Second : {fps}")  
out.write(frame) # Write the frame onto the output.  
ret, frame = player.read() # Read next frame.

Vegyük ezeket az összetevőket, és csomagoljuk be őket egy szép osztályba, amely meghívható az URL-lel és a kimeneti fájllal együtt, amelyre a kimeneti adatfolyamot írni kívánja.

**7.Összehasonlítás**

A különböző módszerek összehasonlításához szükségünk van a sikeres objektum detektálások számszerűsített eredményeire. Ehhez az kell, hogy a programok futásakor a detektált értékeket folyamatosan rögzítse és mentse le egy külön excel fájlba, ahonnan a kapott értékeket átlagolhatjuk.

# 8.Irodalomjegyzék

1. Balázs Csanád Csáji (2001) Approximation with Artificial Neural Networks; Faculty of Sciences; Eötvös Loránd University, Hungary
2. <http://binds.cs.umass.edu/papers/1992_Siegelmann_COLT.pdf>
3. Hebb, D. (1949) The Organization of Behavior. New York: Wiley & Sons.
4. McCulloch, & Pitts. (1943). A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY. Bulletin of Mathematical Biophysics, Vol. 5. , 115-133.
5. Rosenblatt, F. (1958). The Perceptron: A Probabilistic Model For Information Storage And Organization In The Brain. Psychological Review Vol. 65 (6), 386–408.
6. Minsky, M., & Papert, S. (1969). Perceptrons: An Introduction to Computational Geometry. Cambridge: The MIT Press.
7. 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).
8. 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
9. Xavier Glorot, Antoine Bordes and Yoshua Bengio (2011). Deep sparse rectifier neural networks (PDF). AISTATS.
10. 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>
11. 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).
12. 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).
13. <https://arxiv.org/abs/1502.03167>
14. 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.
15. <http://jmlr.org/papers/v12/duchi11a.html>
16. Bengio, Y. (2009). Learning Deep Architectures for AI. Foundations and Trends in Machine Vol. 2. (1) , 1-127.
17. Gaiar Baimuratov(2020): Detecting animals in the backyard — practical application of deep learning. https://towardsdatascience.com/detecting-animals-in-the-backyard-practical-application-of-deep-learning-c030d3263ba8
18. Mengyu Tan ,†,Wentao Chao ,†ORCID,Jo-Ku Cheng ,Mo Zhou ,Yiwen Ma ,Xinyi Jiang , Jianping Ge 1,Lian Yu , andLimin Feng: Animal Detection and Classification from Camera Trap Images Using Different Mainstream Object Detection Architectures
19. <https://wellsr.com/python/object-detection-from-videos-with-yolo/>
20. Akash Agnihotri(2021): Implementing Real-time Object Detection System using PyTorch and OpenCV<https://towardsdatascience.com/implementing-real-time-object-detection-system-using-pytorch-and-opencv-70bac41148f7>
21. <https://developers.google.com/mediapipe/solutions/vision/object_detector/python#video>
22. <https://docs.opencv.org/4.x/d1/dfb/intro.html>
23. <https://pytorch.org/features/>
24. <https://stackshare.io/stackups/opencv-vs-pytorch>
25. Lu Tan, Tianran Huangfu, Liyao Wu & Wenying Chen(2021): Comparison of RetinaNet, SSD, and YOLO v3 for real-time pill identification <https://bmcmedinformdecismak.biomedcentral.com/articles/10.1186/s12911-021-01691-8>
26. Rohit Kundu(2023): YOLO: Algorithm for Object Detection Explained [+Examples] <https://www.v7labs.com/blog/yolo-object-detection>
27. <https://developers.arcgis.com/python/guide/how-retinanet-works/>
28. https://www.javatpoint.com/object-recognition-using-python
29. İlke Candan Bengi(2023): A 10 legjobb mélytanulási modell

# 9.Mellékletek

1. [A dolgozat mellékletei, ha vannak]