[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

[Címsorokból generált tartalomjegyzék!]

# 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.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.

2.1.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.

A dokumentum további fejezetei az egyes modulok funkcionalitását ismertetik. De előbb mindenképpen ismerkedjünk meg alaposan a könyvtárban használt általános API fogalmakkal.

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.

2.2.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.

2.3.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.

**3.Modellek**

Olyan algoritmusok, amelyek valós időben képesek detektálni kép alapú (kép,videó) forrásfájlokról tárgyakat.

3.1.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.

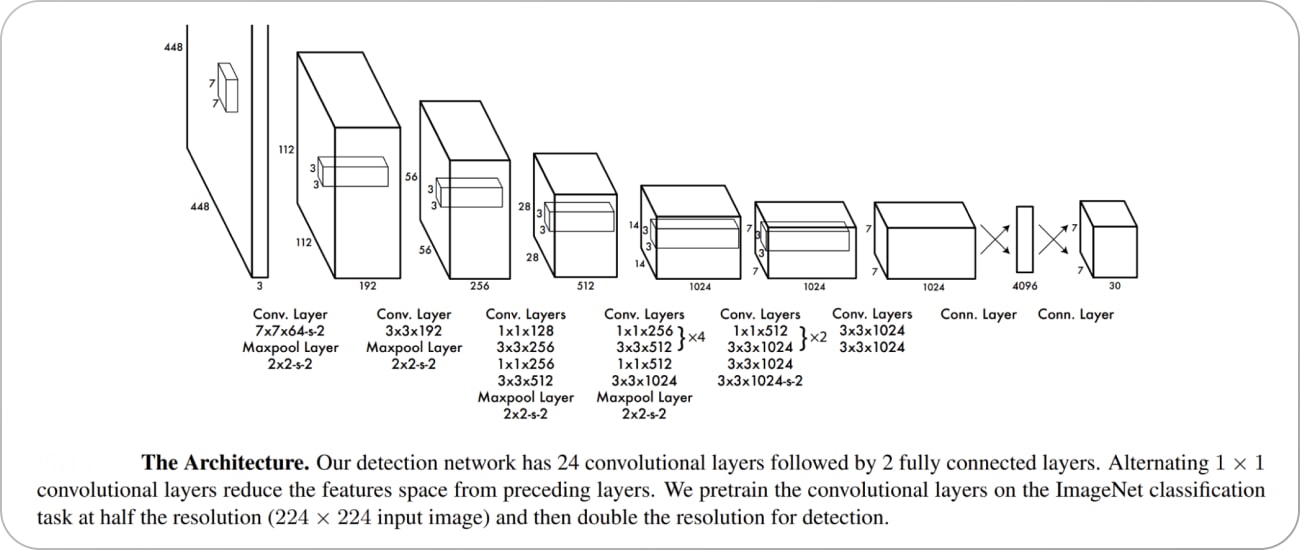
A YOLO a tárgyak észlelésének alapvetően eltérő megközelítését követve a legkorszerűbb eredményeket érte el, és nagy különbséggel verte meg a többi valós idejű tárgyfelismerő algoritmust.

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. Íme egy idővonal, amely a YOLO elmúlt évekbeli fejlődését mutatja be.

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.



3.1.1.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.

3.1.2.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.

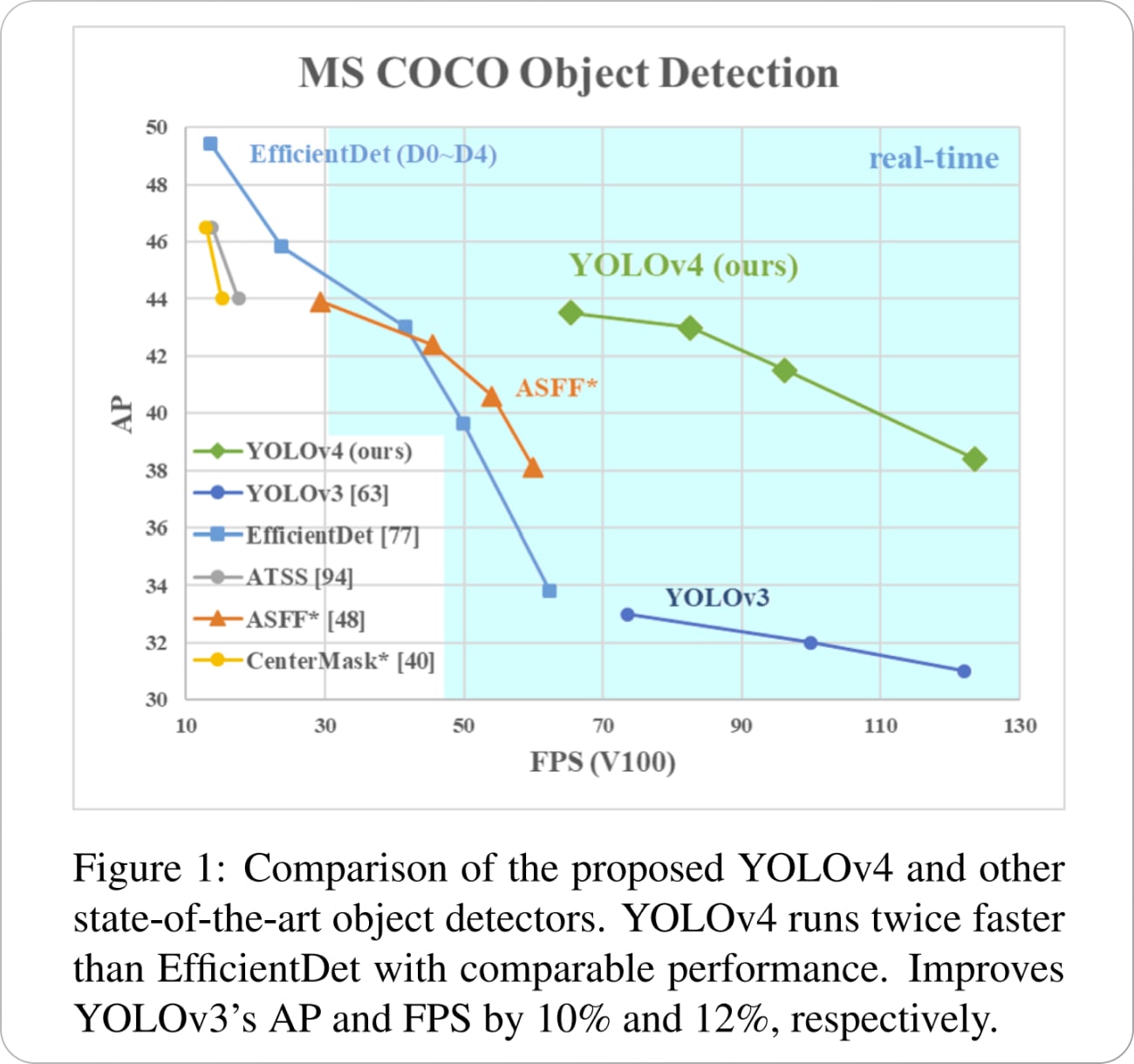
Ezen fejlesztések mellett a YOLO v3 a tárgyméretek és képarányok szélesebb skáláját tudja kezelni. Emellett pontosabb és stabilabb, mint a YOLO korábbi verziói.

3.1.3.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.



3.1.4.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.

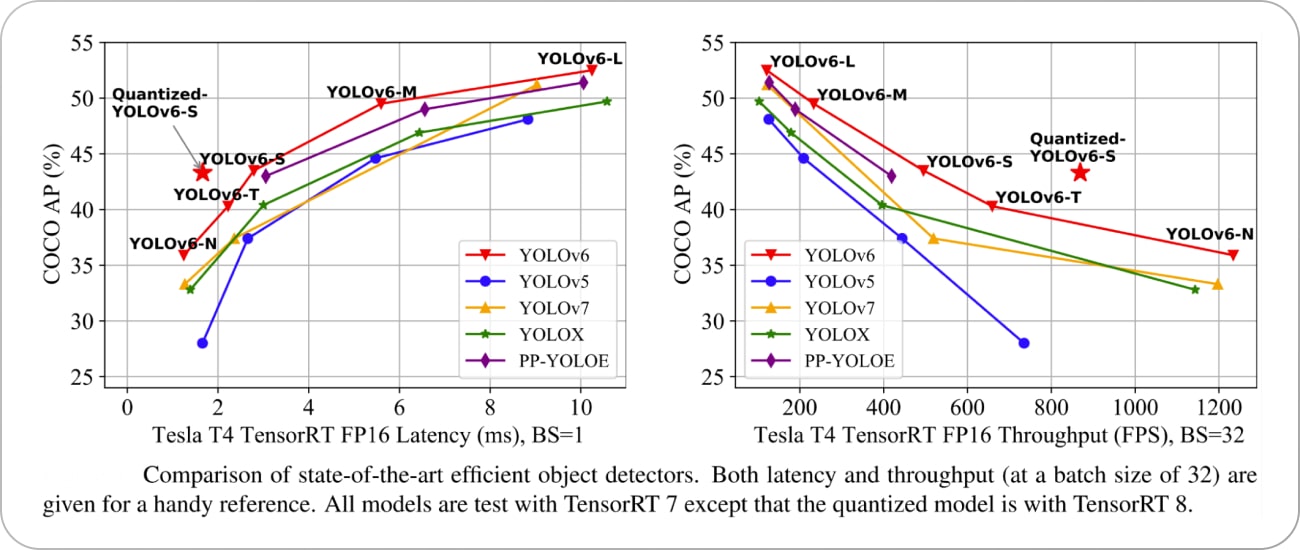
A YOLO v5 egy új módszert használ a horgonydobozok generálására, amelyet "dinamikus horgonydobozoknak" nevezünk. Ennek lényege, hogy egy klaszterező algoritmus segítségével a ground truth bounding boxokat klaszterekbe csoportosítja, majd a klaszterek középpontjait használja horgonydobozként. Ez lehetővé 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.

3.1.5.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".



3.1.6.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.

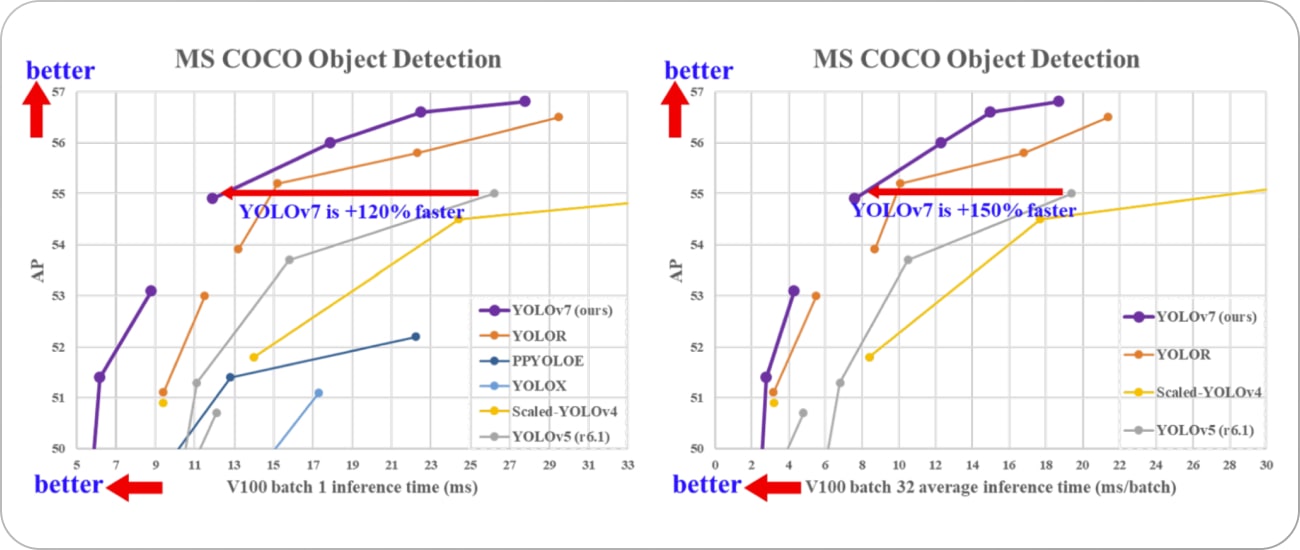
A YOLO v7 egy erős és hatékony objektumfelismerő algoritmus, de van néhány korlátja.

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.

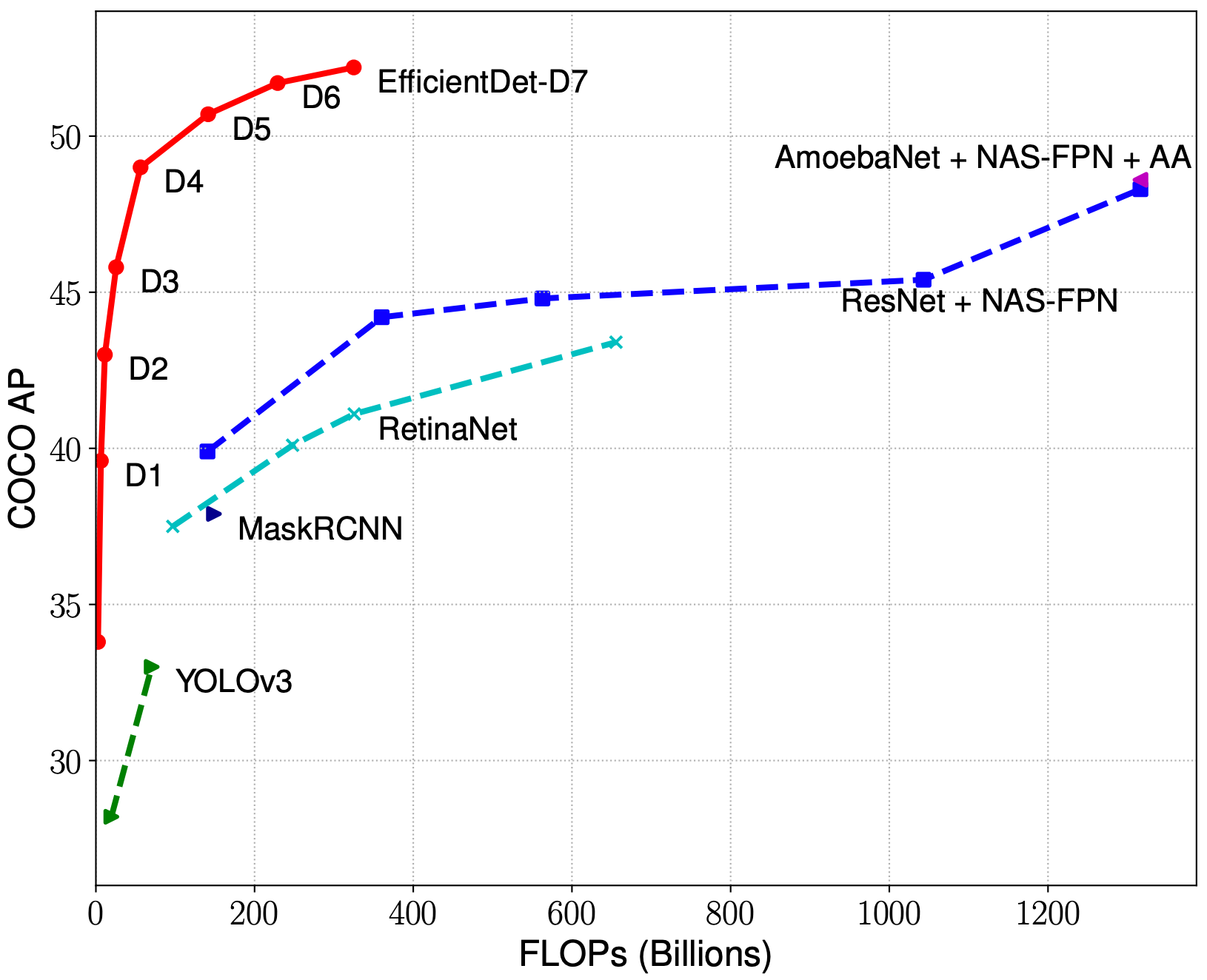


3.1.7.YOLO v8

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.

3.2.EfficientDet

z 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.



3.2.1.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.

3.2.2. 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.

3.3.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.

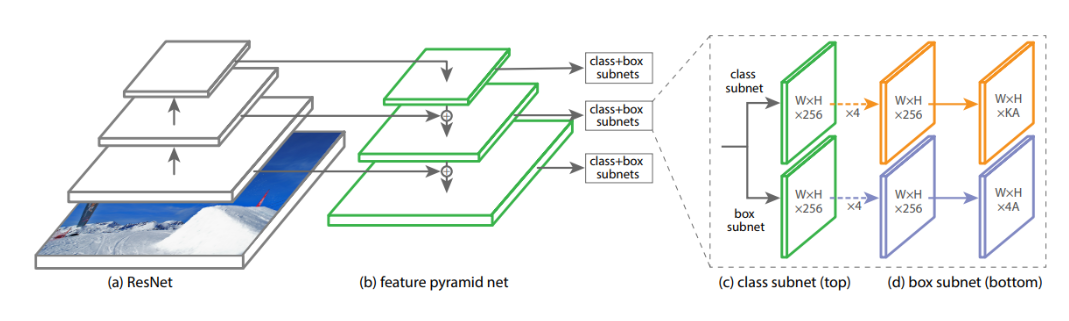
A RetinaNet modell architektúrájának négy fő összetevője van.

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.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.

**5.Gyakorlati alkalmazás**

A korábban tárgyalt keretrendszerek és modellek működését néhány működő gyakorlati projekttel szeretném szemléltetni.

5.1. 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.

5.1.1.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 |

5.1.2.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.

5.1.3.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 | - |

5.1.4.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. |

5.1.5.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. |

5.1.6.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 |

5.1.7.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)

5.1.8.A feladat 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.

5.1.9.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

5.2.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.

5.2.1.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.

5.2.2.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.

5.2.3.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

)

5.2.4.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)

5.3.Implementing Real

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

5.3.1. 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)

5.3.2.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

5.3.3.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

5.3.4.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.

**6.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.

# Irodalomjegyzék

[A felhasznált szakirodalom megadása]

[A szerzők nevét mindenütt “Családnév, X.” formában kell megadni, ahol X. a szerző keresztnevének (keresztneveinek) kezdőbetűje. Magyar cikk esetén a vessző a családnév és a keresztnév kezdőbetűje közt elhagyható. Ha az egyértelműség megkívánja, a keresztnév kiírható teljesen is.

A dolgozat szerzője szabadon választhat, az A vagy a B típust használja.]

**[A-típus:**

A cikkekre való hivatkozás egy []-be írt sorszámmal történik. A sorszámozást folytonosan kell megtenni, a sorba rendezés alapja az első szerző családneve.]

[1] Szerző1 (, Szerző2 ...): *Cikk címe*

Folyóirat neve, sorszám, kezdőoldal-végoldal, év.

[2] Szerző1 (, Szerző2 ...): *Konferencia-kiadvány-beli cikk címe*

„Konferenciakiadvány:” Konferencia neve, hely, kezdőoldal-végoldal, év.

[3] Szerző1 (, Szerző2 ...): *Könyvcím*

„Könyv:” Kiadó, hely, oldalszám, év.

[4] Szerző1 (, Szerző2 ...): *Kutatási jelentés címe (csak publikus elérhető jelentés!)*

„Kutatási jelentés”: Kutatási projekt neve, intézet, oldal, év.

[5] Szerző: *Disszertáció címe*

„PhD/kandidátusi/stb. disszertáció”: Egyetem, kar neve, év.

[6] *Internetes oldal elnevezése:*

*URL, letöltés ideje (csak konkrét dokumentumra mutató URL adható meg!)*

**[B-típus:**

A hivatkozás a szerzők családi nevéből és a kiadás évéből képezett azonosítóval történik. Kettőnél több szerző esetén az „et.al.” rövidítés használható. Pl. „[Vijayasundaram, 1986.]”, „[Meister and Sonar, 1998.]”, „[Felcman et.al., 1994.]”

Abban a ritka esetben, amikor több cikknek is azonos azonosító jutna (megegyeznek a szerzők és a kiadás éve) az év után az „a”, „b”, „c”, stb. betűk csatolandók, pl. „[Stone and Norman, 1993a.]”.

A sorba rendezés alapja a szerzők családneve, végül az év.

Ezt leszámítva a formátum ugyanaz, mint az A-típus esetén, de ekkor a hivatkozási sorszám lehagyható.]

# Mellékletek

1. [A dolgozat mellékletei, ha vannak]