ÓBUDAI EGYETEM

Kandó Kálmán Villamosmérnöki Kar
Elektronikai és Kommunikációs Rendszerek Intézet
Mikroelektronikai és Technológia Tanszék

SZAKDOLGOZAT

OE-KVK
2023

Hallgató neve: **Vasas Bence Barnabás**

Hallgató törzskönyvi száma: T010276/FI12904/K

Budapest, 2023

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Óbudai Egyetem
Kandó Kálmán Villamosmérnöki Kar
Műszertechnikai és Automatizálási Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: Vasas Bence Barnabás
Szakdolgozat száma: SZD2309290841228023[redacted]
Törzskönyvi száma: T010276/FI12904/K **Neptun kódja:** [redacted]

Szak: kórház- és orvostechnikai szakmérnök
Specializáció:

A dolgozat címe: Adott fajra optimalizálható pollinátor-felismerő szoftver tervezése
A dolgozat címe angolul: Design of pollinator recognition software optimised for a given species
A feladat részletezése: Kutassa fel a különböző képfelismerő algoritmusokat és válassza ki az adott feladatnak megfelelőt! Alkalmazza a kiválasztott algoritmusokat Python programozási nyelvben és tesztelje megoldását videomintán! A szoftvert implementálja a kiválasztott hardverbe!

Intézményi konzulens neve: Molnár József Zsolt

A kiadott téma elévülési határideje:2025. 12. 31.
Beadási határidő: 2023. 12. 15.

A szakdolgozat: Nem titkos.

Kiadva: Budapest, 2023. 11. 14. P.H.

Óbudai Egyetem Kandó Kálmán Villamosmérnöki Kar Elektronikai és Kommunikációs Rendszerek Intézet

..... Intézetigazgató

A dolgozatot beadásra alkalmasnak találom:

..... belső konzulens

..... külső konzulens

ÓBUDAI EGYETEM

Kandó Kálmán Villamosmérnöki Kar
Elektronikai és Kommunikációs Rendszerek Intézet
Mikroelektronikai és Technológia Tanszék

HALLGATÓI NYILATKOZAT

Alulírott *Vasas Bence Barnabás* hallgató kijelentem, hogy a szakdolgozat saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült szakdolgozatban található eredményeket az egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja, a titkosításra vonatkozó esetleges megkötések mellett.

Budapest, 2023. 12. 14.

.....
hallgató aláírása

Tartalomjegyzék

Előszó

1. Bevezetés **1**

2. Mesterséges intelligencia **3**

2.1. Az automatizálás 3

2.2. A mesterséges intelligencia 4

2.3. Kép- és objektumfelismerés 5

2.4. Gyógyászati alkalmazások 7

3. Neurális hálózatok **10**

3.1. A neuronok felépítése 10

3.2. Haar kaszkád 14

3.3. Irányított gradiensek hisztogramja (HOG) 16

3.4. Elforgatott címke használata 17

4. Konvolúciós neurális hálózat (CNN) **19**

4.1. A konvolúciós művelet 19

4.2. Összevonás 21

5. A YOLO algoritmus **23**

5.1. A YOLO algoritmus működése 24

5.2. Az algoritmus későbbi fejlesztései 25

6. Neurális hálózat betanítása **28**

6.1. Adatgyűjtés 28

6.2. Címkézés 30
6.3. File-ok és környezet előkészítése 31
6.4. Neurális hálózat betanítása 32

7. Objektum detektálás és az adatok további feldolgozása **34**
7.1. Modell tesztelése valós minta videón 34
7.2. Mentett adatok feldolgozása 35

8. Videorögzítő hardver **37**
8.1. A hardverrel szemben támasztott követelmények 37
8.2. Kamera modul 38
8.3. Adatok mentése 39
8.4. Áramellátás 40
8.5. Rögzítés 41
8.6. A tesztrendszer költsége 41

9. Összegzés **42**

10. Summary **43**

Irodalomjegyzék **44**

Mellékletek **46**
M.1. Fő program 46

Köszönetnyilvánítás

Elsősorban köszönetet szeretnék mondani Molnár Zsolt Tanár Úrnak, akitől nem csak a képzésem során kaptam megfelelő oktatást az általa tartott tantárgyak keretein belül, hanem egy kihívással teli probléma elé állított, melynek megoldása során sokat fejlődtem és a megszerzett tudást számos területen tudom alkalmazni.

Illetve köszönöm Tóth Zoltánnak, aki kontextusba helyezte a munkámat, és fokozta érdeklődésemet azzal, hogy az általam készített képfelismerő rendszernek milyen célt szán.

Budapest, 2023. december 15.

Vasas Bence Barnabás

1. fejezet

Bevezetés

Jelen dolgozatom egy remek lehetőség apropóján jött létre, melyet Molnár Zsolt Tanár Úr vetett fel számomra. Dr. Tóth Zoltán, az Agrártudományi Kutatóközpont Növényvédelmi Intézet munkatársa, a különböző pollinátorokat vizsgálja és a jövőben szüksége lesz egy rendszerre, mely egy kiválasztott növényről készít felvételt, melyet idő közben rovarok látogatnak és beporoznak. A szoftver szerepe ezen rovarok automatikus regisztrálása és a statisztikai paraméterek meghatározása, melyekből később remélhetőleg eredmények fognak származni. Ez a feladat remek kihívás és számos lehetőséget kínál, hogy ismereteimet bővítsem a műszaki területeken.

Erre a célra a megfelelő és talán egyetlen megoldás egy objektumfelismerő rendszer. Ha a szoftver megfelelően van megtervezve, akkor esetleges bővítéseken kívül nem igényel beavatkozást és célját mégis megfelelően ellátja. Azonban az egyszerű módosíthatóság is szempont.

Jelen korban az ilyen megoldások egyre szélesebb körben terjednek el, a gyorsabb és olcsóbb hardvereknek köszönhetően. Életünk majdnem minden területét behálózza az automatikus és okos eszközök hada, beleértve a közlekedést, személyazonosítást, a gyártási folyamatokat és az orvostudományt. Ezért is tesz kedvemre a lehetőség, hogy ezt a feladatot oldhatom meg. A dolgozat megírása közben nagymértékű tájékozódásra van szükség és a megszerzett tudás, a jövőben több területen is hasznomra lehet.

A fentebb említett okokból kifolyólag, ezen dolgozatnak több célt is szánok. Természetesen elsősorban számot szeretnék adni tájékozottságomról a témában és a feladat sikeres és minőségi megoldásáról. Továbbá, mivel az eredményeim a jövőben bővítésre és felhasználásra kerülnek, dolgozatomnak egy dokumentációul is kell szolgálnia, mely követhető és jól strukturált a későbbi olvasóknak.

Dolgozatom elején, áttekintem a forrásokat, melyek alátámasztják a téma fontosságát. Több, különböző iparági alkalmazást is bemutatok. Természetesen, mivel kórházés orvostechnikai szakmérnök tanulmányom záródolgozata a jelenlegi, köteles vagyok a témát ezen az ágon belül is kontextusba helyezni és a képfelismerés relevanciáját bemutatni. Számos orvostechnikai képalkotó berendezés által szolgáltatott adathalmaz, jelen dolgozatban szereplő technikákkal kerül feldolgozásra, amelyek a példákban is szerepelnek. Ezen technikák implementálása különböző iparágakba jelenleg is zajlik, aktívan kutatott terület.

A mesterséges intelligencia szakirodalma hatalmas mennyiségű, tekintve, hogy számtalan különböző metódus és algoritmus létezik. Csak a felszínt kapargatva, bemutatom azokat az alap matematikai módszereket és alapfogalmakat, melyek megértése szükséges volt ahhoz, hogy a dolgozatot el tudjam készíteni. Megindoklom az algoritmus kiválasztásának okát és példát is mutatok arra, hogy milyen másik módszer vallott kudarcot a feladat megoldásában.

Mindezek után bemutatom a megvalósítást is. Teljes áttekintést nyújtok az adatgyűjtéstől kezdve, az adatok előkészítésén és feldolgozáson át a modell betanításáig. Az ezekhez használt kódokat is a kellő részletességig tárgyalom. A betanítás eredményeinek áttekintése után, a valós környezetben készült videón tesztelem az algoritmus sikerességét, ezzel értékelni kívánom a további felhasználását. Dolgozatomat a projekt minimális anyagi vonzataival zárom, melyek elengedhetetlen részei a mérnöki feladatnak. Egy műszaki terv, alkotás célja, hogy egy problémára megoldást nyújtson, azonban azt a megfelelő árért cserébe kell tennie. A megvalósításnak anyagilag is helyt kell állnia.

2. fejezet

Mesterséges intelligencia

2.1. Az automatizálás

Az automatizálás a technológia alkalmazása olyan feladatok elvégzésére, amelyeket korábban emberek végeztek [1]. A hatékonyság, a pontosság és a következetesség javítására, valamint a költségek csökkentésére használható. Az automatizálást számos iparágban alkalmazzák, többek között a gyártásban, a mezőgazdaságban, az egészségügyben és a közlekedésben.

Az automatizálásnak sokféle típusa létezik, de nagyjából két kategóriába sorolható: fix automatizálás és rugalmas automatizálás. A fix automatizálást olyan feladatok elvégzésére használják, amelyek ismétlődőek és kiszámíthatóak. Gyakran használják a gyártásban termékek összeszerelésére vagy más, nagyfokú pontosságot igénylő feladatok elvégzésére. A rugalmas automatizálás olyan feladatok elvégzésére szolgál, amelyek összetettebbek és változékonyabbak. Gyakran alkalmazzák az olyan iparágakban, mint az egészségügy és a szállítás, hogy reagáljanak a változó körülményekre.

Az automatizálás jelentős hatással volt a munkahelyekre. Új munkahelyek létrejöttéhez vezetett, de néhány munkahelyet meg is szüntetett. Az automatizálásnak a foglalkoztatásra gyakorolt általános hatása pozitív volt, de aggodalomra ad okot a munkahelyek kiszorulása és az egyenlőtlenségek miatt is.

A mesterséges intelligencia (AI) az automatizálás egy olyan típusa, amely számítógépek segítségével szimulálja az emberi intelligenciát. A mesterséges intelligencia rendszerek képesek adatokból tanulni és alkalmazkodni az új helyzetekhez. Sokféle feladat elvégzésére használhatók, beleértve az adatelemzést, a döntéshozatalt és a problémamegoldást [22]. Az AI egyre fontosabbá válik az automatizálásban. Az AI-alapú automatizálási rendszerek olyan feladatokat is el tudnak végezni, amelyek korábban túl bonyolultak vagy időigényesek voltak a hagyományos automatizálási rendszerek

számára. A mesterséges intelligencia például felhasználható önvezető autók fejlesztésére és a betegségek diagnosztizálásának és kezelésének automatizálására. [17] [18]

Íme néhány példa arra, hogyan használják a mesterséges intelligenciát feladatok automatizálására: A gyártásban a mesterséges intelligenciát olyan feladatok automatizálására használják, mint a minőségellenőrzés, a termékellenőrzés és a megelőző karbantartás. A mezőgazdaságban a mesterséges intelligenciát olyan feladatok automatizálására használják, mint a termésfigyelés, a kártevők elleni védekezés és a betakarítás. Az egészségügyben a mesterséges intelligenciát olyan feladatok automatizálására használják, mint az orvosi képalkotás elemzése, a betegségek diagnosztizálása és a kezelés tervezése. A közlekedésben a mesterséges intelligenciát olyan feladatok automatizálására használják, mint az önvezető autók, a forgalomirányítás és a fuvarozási logisztika. A mesterséges intelligencia még mindig a fejlődés korai szakaszában van, de meg van benne a lehetőség, hogy forradalmasítsa az automatizálást. Az AI-alapú automatizálási rendszerek hatékonyabbak, pontosabbak és rugalmasabbak lehetnek, mint a hagyományos automatizálási rendszerek. Olyan feladatok elvégzésére is alkalmasak lehetnek, amelyeket korábban lehetetlen volt automatizálni [3].

2.2. A mesterséges intelligencia

A mesterséges intelligencia története egészen az ókorig vezethető vissza, amikor olyan filozófusok, mint például Arisztotelész és később Alan Turing matematikus elkezdtek találgatni a gondolkodni képes gépek létrehozásának lehetőségéről. A mesterséges intelligencia kutatása azonban csak a 20. században kezdett jelentős előrelépést tenni. A mesterséges intelligencia egyik legfontosabb korai fejleménye a Turing-teszt megalkotása volt 1950-ben. A Turing-teszt annak vizsgálata, hogy egy gép képes-e az emberi viselkedéssel egyenértékű vagy attól megkülönböztethetetlen intelligens viselkedést tanúsítani. A teszt során egy emberi értékelő természetes nyelvű beszélgetést folytat két másik féllel, akik közül az egyik ember, a másik pedig egy emberhez hasonló válaszok generálására tervezett gép. Minden résztvevő el van választva egymástól. Ha az értékelő nem tudja megbízhatóan megkülönböztetni a gépet az embertől, akkor a gép átment a teszten [16]. A mesterséges intelligencia egy másik fontos korai fejleménye az első mesterséges neurális hálózat létrehozása volt 1951-ben. A mesterséges neurális hálózatokat az emberi agy szerkezete és működése ihlette. Összekapcsolt csomópontokból állnak, amelyek információkat dolgoznak fel és adatokból tanulnak. Az 1960-as és 1970-es években a mesterséges intelligencia kutatása jelentős eredményeket ért el a természetes nyelvfeldolgozás, a gépi tanulás és a robotika területén. Az 1980-as években azonban a mesterséges intelligencia kutatása hanyatlásnak indult, amit „mesterséges intelligencia télként” emlegetnek. Ennek számos oka volt, többek között a mesterséges

intelligencia képességeinek túlzott ígérete és a mesterséges intelligencia kutatásának alulfinanszírozása.

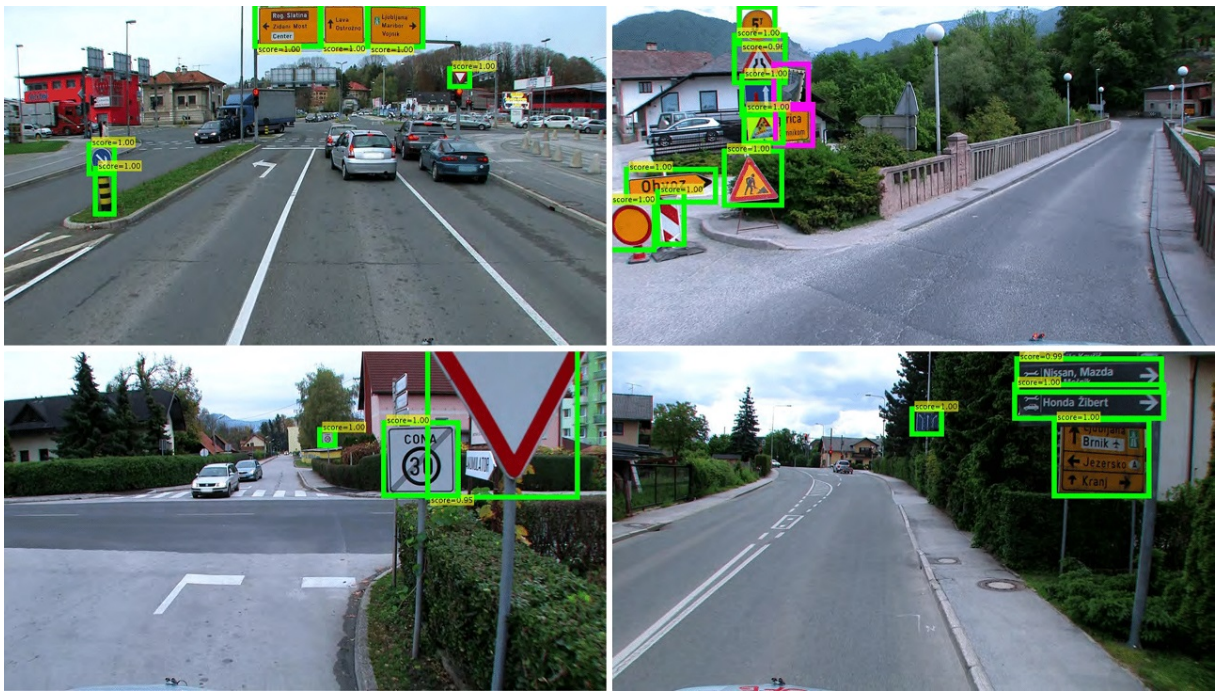
A mesterséges intelligencia tél az 1990-es években kezdett olvadni, köszönhetően a gépi tanulásban elért eredményeknek és a világháló fejlődésének. A világháló hatalmas új adatforrást biztosított a mesterséges intelligencia kutatói számára, amelyen algoritmusaikat betaníthatták.

A 2000-es években az AI kutatás jelentős előrelépést ért el a mélytanulás, a számítógépes látás és a természetes nyelvi feldolgozás területén. A mélytanulási algoritmusok a gépi tanulási algoritmusok egy olyan típusa, amely mesterséges neurális hálózatokat használ. A mélytanulási algoritmusok az elmúlt években rendkívül sikeresek voltak, és a legkorszerűbb eredményeket érték el a feladatok széles skáláján, többek között a képfelismerés, a természetes nyelvfeldolgozás és a gépi fordítás területén.

2.3. Kép- és objektumfelismerés

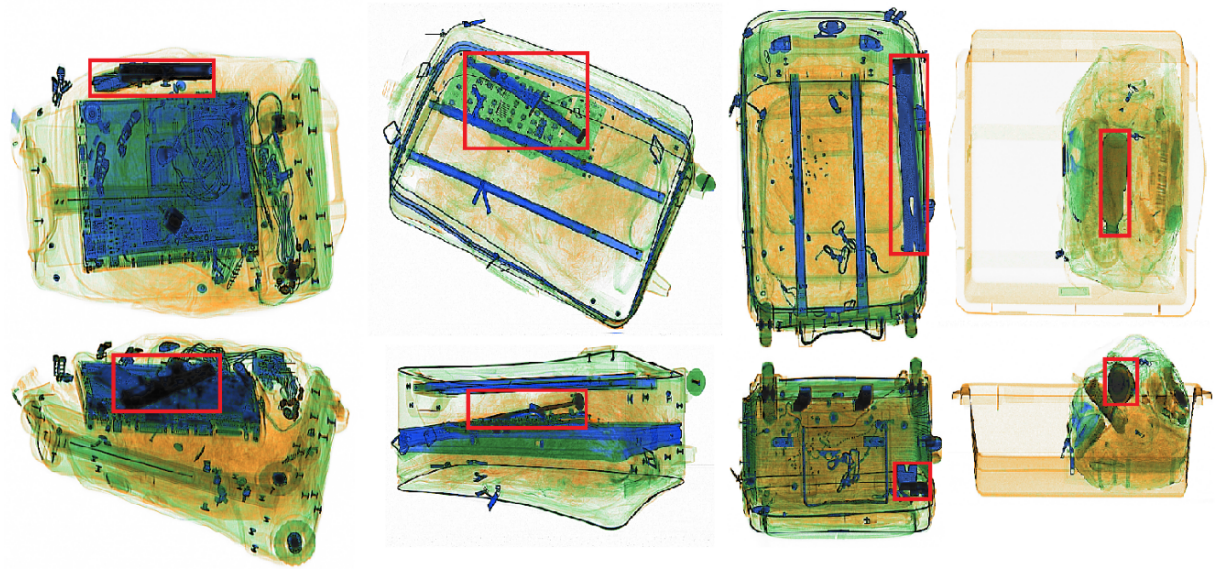
A kép- és objektumfelismerés térhódítása számos területen jelenleg is folyamatban van. Az emberek hétköznapjaiban is szerepet játszik, évről évre jelennek meg termékek és szolgáltatások, melyek használnak ilyen algoritmusokat, módszereket.

Az egyik legkézzelfoghatóbb ilyen terület a közlekedés. A gépjárművek, vezetőt segítő alkatrészeinek száma évtizedek óta növekszik, ezek többnyire elektronikus és mechanikus eszközök. A 2000-es években, az alkalmazott a szoftveres eszközök tárháza jelentősen megnőtt. Az elmúlt 10 évben pedig jelentős forradalmasítás kezdte meg első fázisát. Míg a jövő energiaforrás tekintetében nincs sem politikai, sem műszaki egyetértés, addig az önvezetés tekintetében véleményem szerint, más a helyzet. A teljesen önálló, önvezető járművek megalkotása minden tekintetben egy előrehaladást ígér. A közlekedési dugók, az egymással kommunikáló és környezetük változására gyorsan reagáló okos járművek segítségével megszűnhet. Ezen fejlesztések a gépjárművek biztonságát is növeli, kiaknázza az emberi figyelmetlenségben, reakcióidőben rejlő kockázatokat. Az önvezető járművek alapvető építőeleme az objektumfelismerés. Az autóra szerelt kamerák és érzékelők segítségével detektálhatóak más járművek, gyalogosok és akadályok, képesek a forgalmi rendet irányító táblák és jelzőlámpák felismerésére. Ehhez gyorsan reagáló hardver és szoftver párosításra van szükség, a hatékony működés és a változó körülményre való azonnali reagálás érdekében [2].



2.1. ábra. Autonóm jármű környezeti detektálása[20]

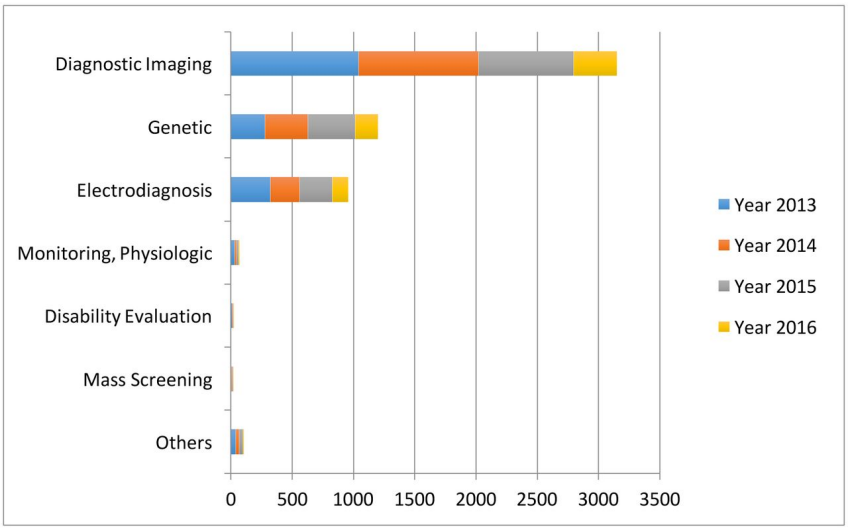
Az utazóközönség biztonsága érdekében a Közlekedésbiztonsági Hivatal (TSA) biztonsági ellenőrző pontokat működtet az Egyesült Államok repülőterein, hogy a veszélyes tárgyakat távol tartsa a repülőgépektől. Ezeken az ellenőrző pontokon a TSA röntgenszkennereket alkalmaz, hogy a közlekedésbiztonsági tisztviselők megvizsgálhassák a kézipoggyászok tartalmát. Az összes potenciális fenyegetés azonosítása és felkutatása azonban kihívást jelentő feladat lehet. Ennek eredményeképpen a TSA az utóbbi időben érdeklődést mutat a mélytanuláson alapuló automatikus felismerő algoritmusok iránt, amelyek segíthetik a biztonsági szakembereket. [10]



2.2. ábra. Reptéri röntgenkamera és a képfelismerés kombinációja [10]

2.4. Gyógyászati alkalmazások

A fentebb említett alkalmazások mellett, az egyik legjelentősebb és engem talán legjobban foglalkoztató terület, az orvostechnológiai implementálás. A képfelismerés kritikus szerepet játszik az orvosi alkalmazásokban, forradalmasítva a diagnosztikát, a kezeléstervezést és a betegellátást. Ez magában foglalja az orvosi képek, például röntgenfelvételek, MRI-felvételek, CT-felvételek és szövettani preparátumok elemzését és értelmezését, hogy segítse az egészségügyi szakembereket a pontos diagnózisok és kezelési döntések meghozatalában.

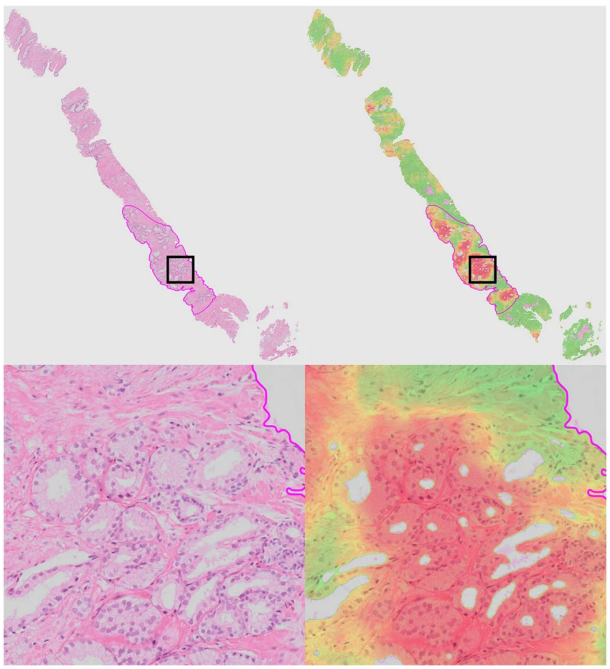


2.3. ábra. A PubMed adatbázisban megjelenő mesterséges intelligenciával foglalkozó irodalmak, különböző tevékenységekre bontva. Látható, hogy a mesterséges intelligencia jelenléte a képalkotó diagnosztikában a legjelentősebb. [5]

Az orvosi képfelismerés során fejlett algoritmusokat és gépi tanulási technikákat alkalmaznak a releváns információk kinyerésére és a különböző betegségekre, rendellenességekre vagy állapotokra utaló minták azonosítására. Ezek az algoritmusok hatalmas mennyiségű címkézett orvosi képadatokból tanulnak, hogy felismerjék a különböző orvosi állapotokhoz kapcsolódó sajátos jellemzőket és mintákat. A képfelismerés egyik legjelentősebb alkalmazása az orvostudományban a számítógépes diagnosztika (computer-aided diagnosis, CAD), ahol az algoritmusok segítik a radiológusokat és a klinikusokat az orvosi képek értelmezésében. [6] A mammográfiában például a CAD-rendszerek képesek elemezni az emlőkről alkotott képeket, hogy azonosítsák a potenciálisan aggályos területeket, segítve ezzel az emlőrák korai felismerését. [15] A képfelismerés megkönnyíti az orvosi képek automatizált szegmentálását is, ami az érdekes struktúrák körvonalazását és körülhatárolását jelenti. Ez a szegmentálás kulcsfontosságú a sebészeti tervezés, a sugárterápia és a betegség előrehaladásának számszerűsítése szempontjából. Az agyi képalkotás során az algoritmusok képesek az agydaganatok szegmentálására, hogy pontos méréseket végezzenek, és segítsék a kezelés tervezését. [21] A képfelisme-

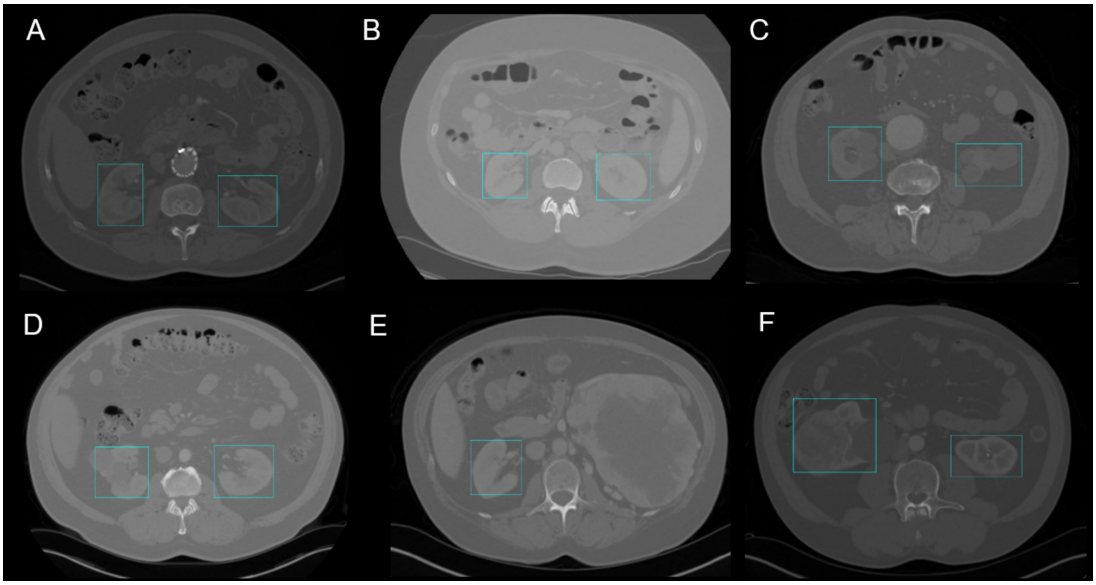
rés emellett segít az orvosi képeken található konkrét anatómiai struktúrák vagy szervek azonosításában és osztályozásában. Ez a képesség különösen értékes az olyan eljárásokban, mint a szervátültetés, ahol a donor és a recipiens szervek pontos azonosítása és egyeztetése kritikus fontosságú. Emellett a képfelismerés a patológiában is alkalmazásra került, ahol az algoritmusok digitalizált szövettani preparátumokat elemeznek, hogy segítsék a patológusokat a betegségek diagnosztizálásában, a daganatok osztályozásában és a sejtes rendellenességek azonosításában. Ez a technológia ígéretes a diagnosztikai pontosság, a hatékonyság és a megfigyelők közötti variabilitás javítására a patológiában. [19]

A tudósok egy kutatásban a mélytanulást (Deep Learning) a szövettani preparátumelemzés hatékonyságának és pártatlanságának növelésére szolgáló módszerként alkalmazzák. Két példán keresztül mutatják be, hogy ez az innovatív technológia csökkentheti a patológusok munkaterhelését, miközben egyidejűleg javítja a diagnózisok objektivitását: a prosztatarák azonosítása biopsziás mintákban és az emlőrák metasztázisainak kimutatása őrszem nyirokcsomókban (2.4. ábra). További immunhisztokémiai markerek vagy emberi beavatkozás nélkül felfedezték, hogy a prosztatarákot és az emlőrák mikro- és makrometasztázisait tartalmazó valamennyi preparátum automatikusan azonosítható, míg a jóindulatú és normális szöveteket tartalmazó preparátumok 30-40%-a kizárható. Arra a következtetésre jutottak, hogy a mélytanulásban nagy potenciál rejlik az emlőrák stádiumbeosztásának és a prosztatarák diagnózisának pontosságának növelésében. [11]



2.4. ábra. Reprezentatív példa egy 30%-ban rákos prosztata biopsziás teljes preparátumra. A felső sorban a teljes látómező, az alsó sorban egy közeli felvétel. A piros szín a rák nagy valószínűségét, míg az átlátszó/zöld szín az alacsony valószínűséget jelzi.[11]

Az általam használt algoritmus (YOLO), aktívan használt az orvostudományban. CT-felvételeken, 2 és 3 dimenzióban is képesek voltak ennek segítségével szerveket azonosítani. A Dice score egy átfedést jellemző mérőszám, melynek értéke átfedés nélkül 0, teljes átfedéssel pedig 1. A YOLO algoritmussal ez a pontszám 2 dimenziós vizsgálatnál elérte a 0,851 értéket, bizonyítva hatékonyságát. [9]



2.5. ábra. CT-felvételeken vese detektálása YOLO algoritmussal[9]

Összességében a képfelismerés az orvosi felhasználásban nagy lehetőségeket rejt, a diagnosztikai pontosság fokozására, a kezelés tervezésének segítésére és a betegek eredményeinek javítására. A fejlett algoritmusok és a gépi tanulás erejét kihasználva az orvosi képfelismerés átalakítja az egészségügyi ellátást, mivel gyorsabb, pontosabb diagnózisokat és személyre szabott kezelési stratégiákat tesz lehetővé. Ezen eszközökkel tehermentesíthetőek az egészségügyi szakemberek azon intézményekben és régiókban, ahol a túlterhelés a jellemző, illetve a világ számos területén, ahol az orvossal való kapcsolatfelvétel nehézségeket okoz, életmentő lehet.

3. fejezet

Neurális hálózatok

Az első mesterséges neurális hálózatot McCulloch és Pitts találta ki egy biológiai neuron modellezésére tett kísérletként [12]. Ezek a struktúrák az emberi agy működését hivatottak utánozni, ahhoz hasonlóan a beérkezett adatokat feldolgozzák és ezek alapján megtanulnak döntést hozni.

A neurális hálózat működésének első szakasza a tanulás. Ezen folyamat során maga a hálózat kerül kialakításra és ebbe betápláljuk a tanulási adatokban rejlő mintázatokat. A második szakaszban, az előhívási fázisban használjuk ezt az információ-feldolgozó rendszert. A tanulási fázis lényegesen több időt vesz igénybe, mint az előhívási fázis. Ez a két szakasz általában időben elválik, azonban vannak olyan rendszerek is, melyek az alkalmazás során adaptálódnak, tovább tanulnak. Erre abban az esetben van szükség, amikor a vizsgálati körülmény időben nem állandó.

3.1. A neuronok felépítése

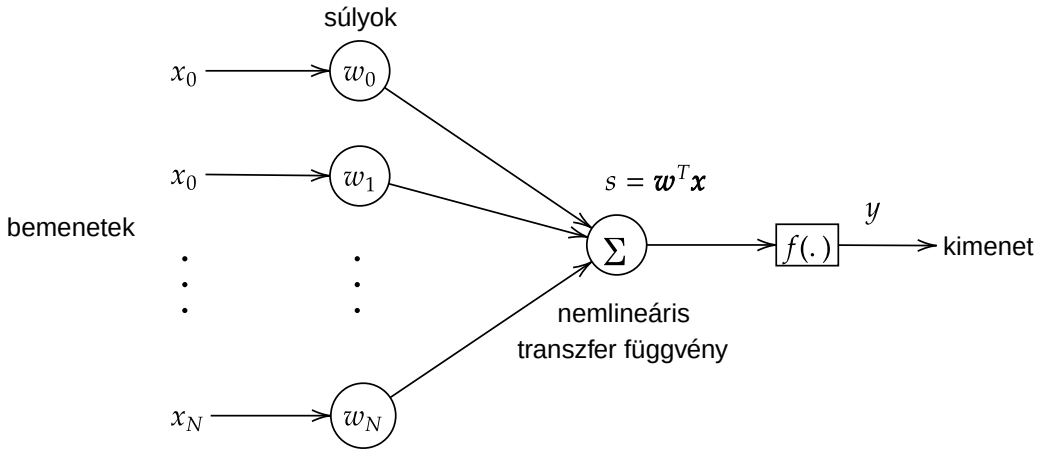
A neuron, vagy más nevén csomópont (node), egy több-bemenetű és egy-kimenetű elem. A bemenet és kimenet között valamilyen, általában nemlineáris leképezést valósít meg. A neuronok működését a következő matematikai formában adhatjuk meg:

$$y(k) = f(x(k), x(k-1), ..., x(k-M), y(k-1), ..., y(k-L)) \tag{3.1}$$

ahol

$$x(k) = [x_0(k), x_1(k), ..., x_N(k)]^T. \tag{3.2}$$

Ebben az esetben, a neuron egy $f : \mathbb{R}^{(M+1)(N+1)+L} \rightarrow \mathbb{R}$ leképezést valósít meg. A 3.1. ábrán látható az egyenrangú bemenetekkel rendelkező neuron tipikus felépítése. Az x_i skalár bemenetek $w_i (i = 0, 1, ..., N)$ súlyozásra majd összegzésre kerülnek. Ez után a súlyozott összeg (s) egy $f(.)$ nemlineáris elemre kerül. A bemeneti jelet szokás ingernek,



3.1. ábra. Egyenrangú bemenetekkel rendelkező neuron

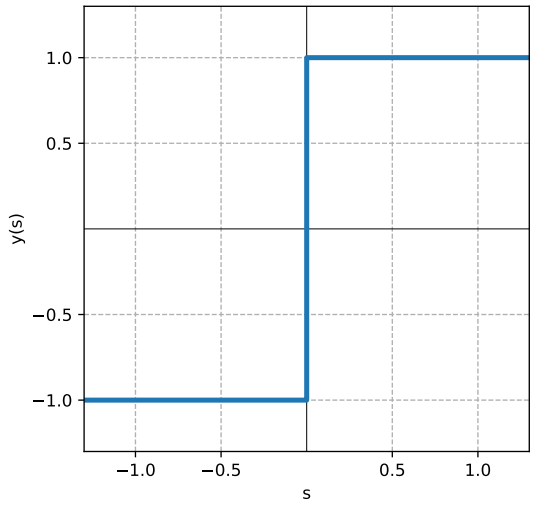
a kimenetit pedig válasznak nevezni, a biológiai analógia szerint [13]. Az összegző pont, az abba bemenő értékek lineáris kombinációját adja:

$$s = \sum_{i=0}^{N} w_i x_i = \boldsymbol{w}^T \boldsymbol{x} \tag{3.3}$$

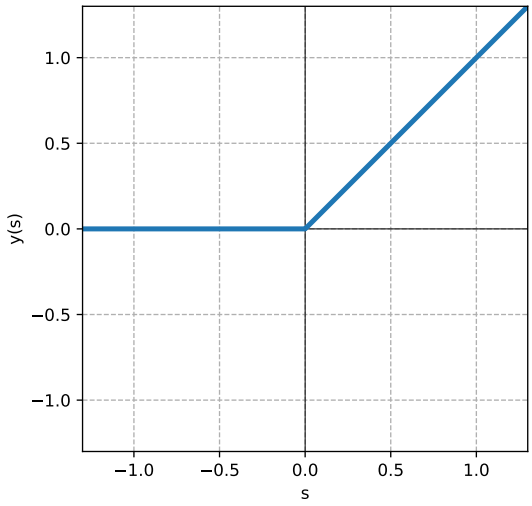
A leképezést a következőképpen kapjuk:

$$y = f(s) = f(w^T x). \tag{3.4}$$

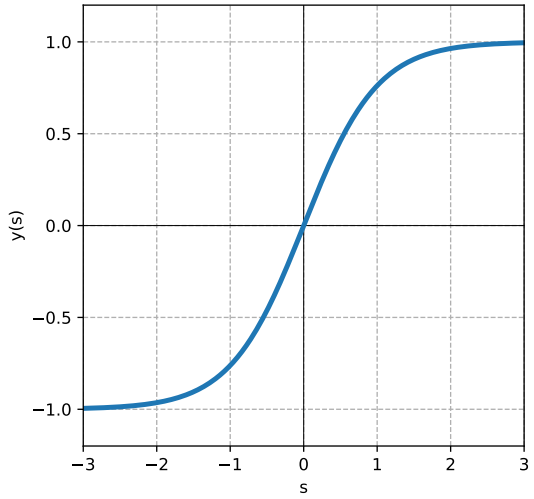
Az aktivációs függvény jellemzően nemlineáris. Jellegre küszöbfüggvény, melynek értelmezési tartománya a valós számok halmaza, értékkészlete pedig alulról és felülről is korlátos a valós számok halmazán. A 3.2. ábrán különböző, ilyen tipikus nemlineáris függvényeket láthatunk. A 3.2c és 3.2d monoton növekvő, folytonos függvények telítődő jelleggel, melyeket szigmoid függvényeknek is nevezünk. Ezeknek a függvényeknek kitüntetett szerepük van a neuronok működésében.



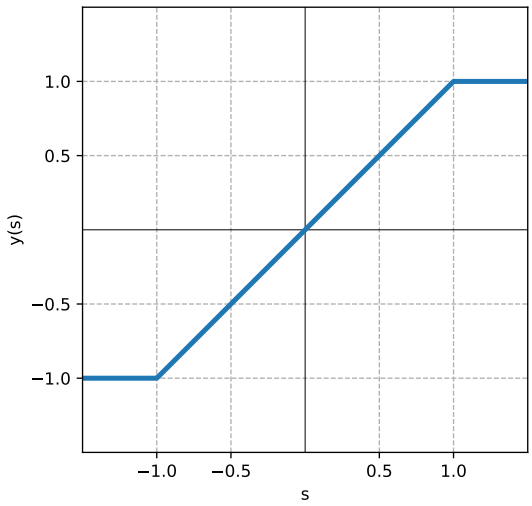
(*a*) *lépcsőfüggvény*



(*b*) *ReLU függvény*



(*c*) *tangens hiperbolikusz függvény*



(*d*) *telítéses lineáris függvény*

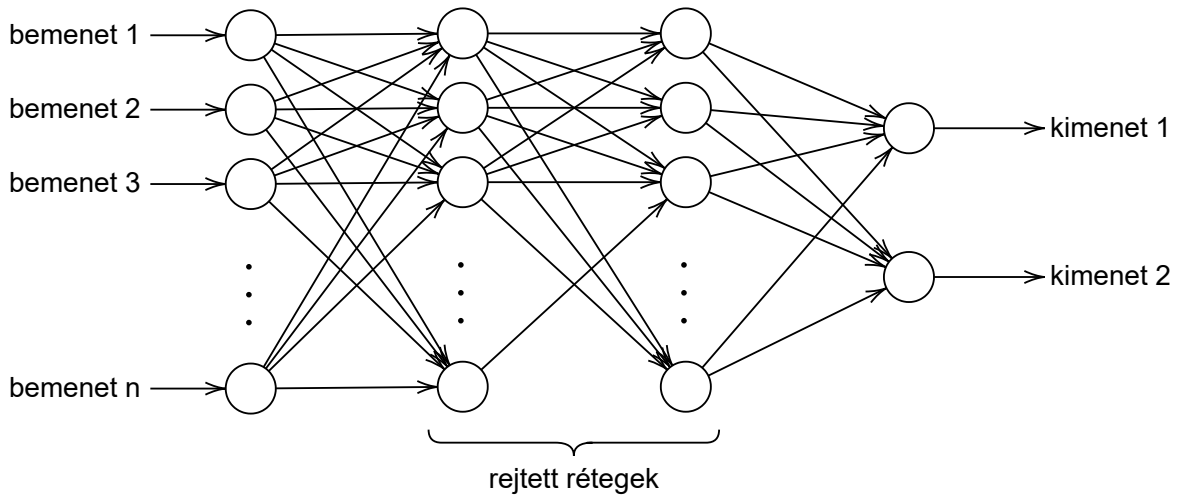
3.2. ábra. nemlineáris aktivációs függvények

$$(a) \qquad y(s) = \begin{cases} -1 & \text{ha } s < 0 \\ 1 & \text{ha } s \geq 0 \end{cases} \tag{3.5}$$

$$(b) \qquad y(s) = \begin{cases} s & \text{ha } s > 0 \\ 0 & \text{egyébként} \end{cases} \tag{3.6}$$

$$(c) \qquad y(s) = \frac{1 - e^{-Ks}}{1 + e^{-Ks}}; \quad K > 0 \tag{3.7}$$

$$(d) \qquad y(s) = \begin{cases} +1 & s > 1 \\ s & -1 \leq s \leq 1 \\ -1 & s < -1 \end{cases} \tag{3.8}$$



3.3. ábra. Neurális hálózat rejtett rétegekkel

A rejtett réteg a neuronok olyan rétegét jelenti, amely nem a bemeneti és nem a kimeneti réteg (3.3. ábra). A rejtett rétegek teszik a neurális hálózatokat „mélyrehatóvá", és teszik lehetővé számukra az összetett adatreprezentációk megtanulását. Ezek a mély tanulási modellek számítási munkaereje, amelyek lehetővé teszik a neurális hálózatok számára a függvények közelítését és a bemeneti adatokból származó minták megragadását.

A rejtett rétegek elsődleges szerepe az, hogy a bemeneteket olyanná alakítsák, amit a kimeneti réteg fel tud használni. Ezt úgy érik el, hogy a bemenetekre súlyokat alkalmaznak, és azokat egy aktiválási függvényen keresztül vezetik. Ez a folyamat lehetővé teszi, hogy a hálózat megtanuljon nemlineáris kapcsolatokat a bemeneti és kimeneti adatok között. A rejtett réteg minden egyes neuronja bemenetet kap az előző réteg összes neuronjától, megszorozza ezeket a bemeneteket a saját súlyaival, hozzáad egy torzítótermet, majd az eredményt egy aktiválási függvényen továbbítja. Az egyes neuronok kimenete ezután a következő réteg bemeneteként szolgál.

A rejtett rétegek száma és az egyes rejtett rétegekben lévő neuronok száma határozza meg a neurális hálózat felépítését. A hálózat mélysége a rejtett rétegek számát, míg a szélessége az egyes rejtett rétegekben lévő neuronok számát jelenti. A több rejtett réteggel rendelkező mélyebb hálózatok összetettebb reprezentációkat képesek megtanulni, de a betanításukhoz több adatra és számítási teljesítményre is szükség van. Ezzel szemben a szélesebb, több neuronnal rendelkező hálózatok több információt rögzíthetnek a bemeneti adatokról, de megfelelő kezelés hiányában túlillesztéshez is vezethetnek.

A túlillesztés egy gyakori probléma, amikor a neurális hálózat túl jól megtanulja a képzési adatokat, beleértve a zajt és a kiugró értékeket is, ami rossz teljesítményt eredményez az új, nem látott adatokon. Az én esetemben is történt ilyen túlillesztés vagy túltanítás, ezt a problémát később tárgyalom.

A rejtett rétegek elengedhetetlenek a neurális hálózatok számára az összetett problémák megoldásához. Lehetővé teszik a hálózat számára a jellemző-kivonást, azaz a bemeneti adatokból a releváns információk azonosítását és elkülönítését, amelyek szükségesek a jóslatok vagy döntések meghozatalához.

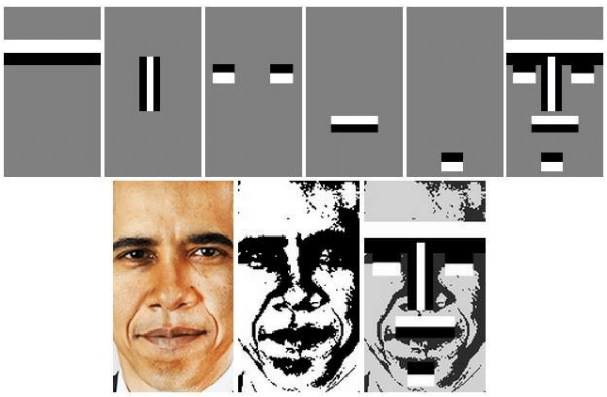
Minden egyes alkalommal, amikor egy adathalmaz áthalad az algoritmuson, azt mondjuk, hogy egy epoch-ot teljesített. Ezért az epoch a gépi tanulásban a tréningadatoknak az algoritmuson való teljes áthaladására utal. Ez egy hiperparaméter, amely meghatározza a gépi tanulási modell képzésének folyamatát.

A képzési adatokat mindig kis tételekre bontjuk, hogy leküzdjük a számítógépes rendszer tárolóhely-korlátozásából adódó problémát. Ezek a kisebb tételek könnyen betáplálhatók a gépi tanulási modellbe, hogy azt betanítsák. Ezt a kisebb darabokra bontási folyamatot a gépi tanulásban batch-nek nevezik.

Ezen paraméterek meghatározása nem egyértelmű, a folyamat többszöri lefuttatása, különböző epoch és batch szám megadásával szükséges lehet. Ha az epoch-ok száma túl alacsony, akkor alultanulást eredményezhet, illetve ha túl magas akkor túltanulást. A magas batch szám olyan problémához vezet, mely során a rendelkezésre álló memória elfogy.

3.2. Haar kaszkád

Az első modell, aminek a betanításával és használatával próbálkoztam, az úgynevezett Haar-kaszkád volt. A modell széles körben alkalmazott. Ehhez pozitív és negatív adatokra van szükség. A pozitív képek ábrázolják azt az objektumot, amit fel akarunk ismerni. A negatív képek olyan képek, amit a modellnek ignorálnia kell, vagyis leginkább a háttérrel és az objektummal összetéveszthető képek. A modell a Haar-jellemzőket tanulja meg a pozitív képekről és azokat keresi az input képen, illetve videón is.



3.4. ábra. Haar-jellemző előállítása [7]

A Haar jellemzők leképezése az eredeti kép „lebutításával” történik. Ahogy a 3.4. ábrán is jól látszik, ezek a jellemzők jól definiáltak és egymáshoz viszonyított helyzetük is fontos szerepet játszik. Ha a bemeneti kép valamilyen szöggel el van forgatva, akkor a megtanult jellemző már nem használható a felismeréshez. Ezt forgatási invariancia tulajdonságnak nevezünk. Azonban ez kiküszöbölhető, ha a pozitív képek között minden forgatási pozícióban szerepel az objektum. Ezen modell betanítását nagyméretű adathalmazzal (kb. 350 pozitív kép) elvégeztem, azonban az eredmény meglepően rossz volt (3.5. ábra). 900-1000 darab negatív kép esetén is rengetek fals eredményt adott.

3.5. ábra. Haar kaszkád használatával a képen több fals pozitív eredmény is látható, azonban a valóban megjelölni kívánt objektumot nem találja az algoritmus.

Több interneten elérhető fórum is azt javasolja, hogy ebben az esetben növelni kell a betanításhoz szükséges képek pozitív képek mennyiségét. Ez azonban túlillesztéshez vezetett. A gépi tanulásban a modell túlillesztés" néven ismert forgatókönyv akkor fordul elő, amikor egy modell túlságosan specializálódik vagy túlságosan szorosan illeszkedik a képzési adatokhoz. Ez akkor következik be, amikor a modell a mögöttes minták mellett a képzési adathalmazban szereplő zajt és véletlenszerű ingadozásokat is megtanulja. Ennek eredményeképpen a túlillesztett modell kiválóan teljesít a képzési adatokon, de nehezen tud jól általánosítani a friss, nem vizsgált adatokra.

A túlilleszkedés egyszerűen úgy magyarázható, mint amikor a tanuló megjegyzi bizonyos kérdésekre adott válaszokat anélkül, hogy ténylegesen megértené a mögöttes gondolatokat. A tanuló nehezen tud megfelelő válaszokat adni, amikor olyan új kérdésekkel szembesül, amelyek hasonlóak, de eltérnek a korábban megjegyzettektől. Ehhez hasonlóan egy túlilleszkedő modell "megjegyzi" a képzési adatokat, és elveszíti

az általánosításra és az új adatpontokra vonatkozó pontos előrejelzések készítésére való képességét.

Mivel a gépi tanulás fő célja olyan modellek létrehozása, amelyek hatékonyak ismeretlen adatokkal, a túlillesztés problémát jelent. Egy modell rosszul teljesíthet, pontatlan előrejelzéseket készíthet, és csökkent megbízhatósággal rendelkezhet valós körülmények között, ha túlilleszkedik. Ahhoz, hogy a gépi tanulási modellek erősek és hatékonyak legyenek, el kell kerülni a túlillesztést.

A paraméterek variálása mellett megoldandó probléma marad a forgatási invariancia. Erre egy megoldást jelenthet az, ha az egyes képkockákat több különböző forgatással is megvizsgáljuk. Ez, az amúgy sem túl gyors algoritmussal ötvözve egy hihetetlenül lassú folyamatot eredményezett.

A fentebb összegyűjtött problémák miatt másik algoritmus kellett kipróbálnom.

3.3. Irányított gradiensek hisztogramja (HOG)

A HOG, azaz Histogram of Oriented Gradients, egy olyan jellemzőleíró, amelyet széles körben használnak a számítógépes látásban és a képfeldolgozásban az objektumok felismerésére. A HOG különösen hatékony az emberfelismerési feladatokban, de számos objektumfelismerési problémára alkalmazható.

Az első lépés a kép gradiensének kiszámítása a helyi intenzitásváltozásokra vonatkozó információk rögzítése érdekében. Ezt jellemzően derivált szűrőkkel végzik. A gradiens kiszámítása mind az x, mind az y irányban történik, ami két mátrixot eredményez, amelyek a gradiens nagyságát és irányát reprezentálják minden egyes pixelnél. A képet kis cellákra osztjuk (pl. 8x8 pixel). Az egyes cellákon belüli gradiens információkat ezután a helyi textúrainformációk rögzítésére használjuk. Minden egyes cellára kiszámítjuk a gradiens orientációk hisztogramját. A hisztogram bin-ek különböző orientációs tartományokat képviselnek, és a bin-értékeket a cellán belüli pixelek gradiens orientációi határozzák meg. A hisztogram segít megragadni a cellán belüli domináns gradiens irányokat.

A megvilágítás és a kontraszt eltéréseinek figyelembevétele érdekében a szomszédos cellákat blokkokba (pl. 2x2 cellák) csoportosítjuk. Az egyes blokkokon belüli hisztogramokra normalizálást alkalmazunk. A normalizálás különböző módszerekkel végezhető, hogy a leíró kevésbé érzékeny legyen a megvilágítás változásaira.

A végső HOG leíró a kép összes cellájából származó normalizált blokkhisztogramok összekapcsolásával jön létre. Ez a leíró kódolja a helyi gradiens orientációjára és nagyságára vonatkozó információkat, így reprezentálja a kép textúráját és szerkezetét. A tesztelés során a betanított osztályozót a bemeneti kép csúszóablakaira alkalmazzák, és az osztályozó válaszai alapján detektálják a tárgy jelenlétét. A HOG-ot általában csúszóablakos megközelítéssel és gépi tanulási osztályozóval kombinálva használják a tárgyfelismerési feladatokhoz.

3.6. ábra. HOG által felismert gradiensekre példa [4]

A HOG hasonlóan rossz eredménnyel zárt a feladat megoldásában, mint a Haar. Kevesebb fals pozitív eredmény volt, de a képfelismerés folyamata jelentősen több időt vett igénybe.

3.4. Elforgatott címke használata

Számomra egyértelmű volt, hogy ha az objektumokat a betanításhoz szükséges képeken minél pontosabban jelölöm meg, az pontosabb eredményhez is vezet. Ennek érdekében pár száz képet a 3.7. ábrán szereplő módon, elforgatott kerettel jelöltem meg.

Ez azonban a már említett problémákat tovább növelte. A modell túl jól megtanulta a mintát és szigorúan csak a betanult sémát kereste. A teszt során egy objektumot sem ismert fel, illetve még jobban ragaszkodott az orientációhoz.

3.7. ábra. Elforgatott téglalap használata címkeként

Mind a HOG, mind a Haar algoritmust az OpenCV által nyújtott, terminálban kezelendő betanításon és grafikus betanításon is teszteltem. A továbbiakban fejlettebb algoritmus kipróbálása szükséges.

4. fejezet

Konvolúciós neurális hálózat (CNN)

A konvolúciós hálózatok (LeCun, 1989), más néven konvolúciós neurális hálózatok vagy CNN-ek, az adatok feldolgozására szolgáló neurális hálózatok egy speciális fajtája, amely ismert rácsszerű topológiával rendelkezik. A "konvolúciós neurális hálózat" elnevezés arra utal, hogy a hálózat egy konvolúciónak nevezett matematikai műveletet alkalmaz. A konvolúció a lineáris műveletek egy speciális fajtája, mely a CNN legalább egyik rétegében végbemegy.

4.1. A konvolúciós művelet

Legáltalánosabb formájában a konvolúció két valós értékű argumentummal rendelkező függvényen végzett művelet. Tegyük fel, hogy egy időtől függő x(t) mennyiséget valamilyen zajjal mérünk. A kevésbé zajos méréshez átlagolásra van szükség, és tudjuk, hogy a frissebb mérések relevánsabbak, ezért azt szeretnénk, hogy ez egy súlyozott átlag lenne, mely nagyobb súlyt ad a frissebb mérésekre. Ezt egy $w(a)$ súlyozási függvénnyel tudjuk elérni, ahol a a mérés kora. Ha minden pillanatban ilyen súlyozott átlagolási műveletet alkalmazunk, akkor egy új s függvényt kapunk, amely a mért mennyiség simított becslését adja:

$$s(t) = \int x(a)\, w(t-a)\, da. \tag{4.1}$$

Ezt a műveletet nevezik konvolúciónak. A konvolúciós műveletet jellemzően csillaggal jelöljük:

$$s(t) = (x * w)(t). \tag{4.2}$$

A konvolúciós hálózat terminológiájában a konvolúció első argumentumát (ebben a példában az x függvényt) gyakran bemenetnek, a második argumentumot (ebben a példában a w függvényt) pedig kernelnek nevezik. Általában, amikor adatokkal dolgozunk egy számítógépen, az idő diszkretizált lesz, és az érzékelőnk rendszeres időközönként

szolgáltat adatokat. Ha most feltételezzük, hogy x és w csak egész t-re definált, akkor definálni tudjuk a diszkrét konvolúciót:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)\, w(t-a). \tag{4.3}$$

A gépi tanulási alkalmazásokban a bemenet általában adatok többdimenziós tömbje, a kernel pedig általában paraméterek többdimenziós tömbje, amelyeket a tanulási algoritmus adaptál. Ezeket a többdimenziós tömböket tenzoroknak nevezzük. Mivel a bemenet és a kernel minden egyes elemét külön-külön kell explicit módon tárolni, általában feltételezzük, hogy ezek a függvények mindenhol nulla, kivéve azon pontok véges halmazában, amelyekre az értékeket tároljuk. Ez a gyakorlatban azt jelenti, hogy az végtelen összegzést véges számú tömbelem fölötti összegzésként valósíthatjuk meg. Végül, gyakran használunk egyszerre több tengelyen átívelő konvolúciót. Ha például egy kétdimenziós I képet használunk bemenetként, akkor valószínűleg egy kétdimenziós K kernelt is használni akarunk:

$$S(i,j) = (I * K)(i,j) = \sum_{m} \sum_{n} I(m,n)\, K(i-m, j-n). \tag{4.4}$$

A konvolúció kommutatív művelet, ezért az előzővel egyenértékűen írhatjuk:

$$S(i,j) = (I * K)(i,j) = \sum_{m} \sum_{n} I(i-m, j-n)\, K(m,n). \tag{4.5}$$

A neurális hálózat képzésében a konvolúció a modellnek megtanulandó súlyok számának csökkentését jelenti a kép kevesebb pixelre történő összegzésével. A konvolúció a helyi pixelekről szóló információkat úgy kombinálja, hogy a képen egymáshoz közeli pixeleket egy kisebb pixelhalmaz "foglalja össze". Ez az "összegzés" úgy történik, hogy egy kernelt "csúsztatunk" a képen, hogy egy végeredményt adjunk ki minden egyes pozíciójához.

A konvolúcióban a kernel egy n x n számmátrix. A kernel mátrixon belül minden egyes számot megszorzunk minden egyes pixel értékével, amelyhez illeszkedik, majd a kernel összes elemét összegezzük, hogy egyetlen értéket kapjunk. Ezután a kernel átcsúszik a kép egy új pozíciójára, és a folyamat megismétlődik. A kernelmátrix optimális számait a modell megtanulja. A legnagyobb sikeresség érdekében a modell olyan értékeket tanul meg a kernelhez, amelyek valamilyen kulcsfontosságú információt rögzítenek a képen belül, bár ez az információ az ember számára közvetlenül nem értelmezhető. Néhány példa arra, hogy a CNN fejlesztői felfedezték, hogy a kernelek képesek a képen belüli konkrét objektumok, strukturális minták vagy a kép domináns körvonalai

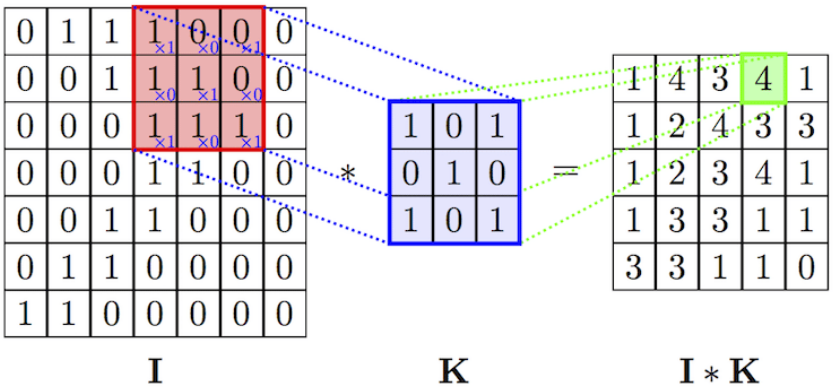
kinyerésére. Egy példa a konvolúciós számításra:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = (a \cdot 1) + (b \cdot 2) + (c \cdot 3) + (d \cdot 4) \tag{4.6}$$

Mátrix formában a következőképpen írható:

$$\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix} \cdot \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1n} \\ y_{21} & y_{22} & \cdots & y_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ y_{m1} & y_{m2} & \cdots & y_{mn} \end{bmatrix} = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} x_{(m-i)(n-j)} y_{(1+i)(1+j)} \tag{4.7}$$

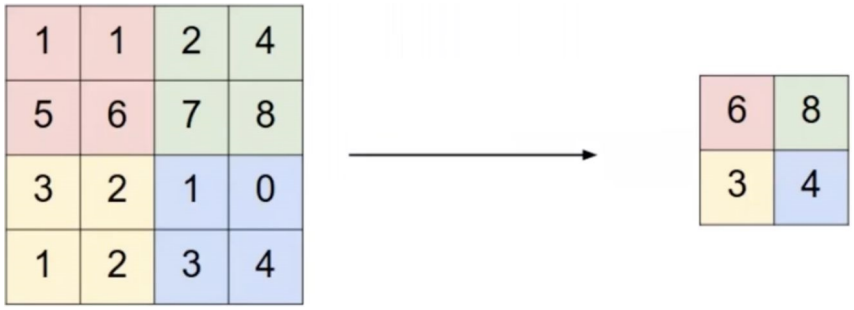
A 4.1. ábrán pedig egy konkrét példán láthatjuk a konvolúciós számítást, melyben Az I egy bemenő kép pixeleiből képzett mátrix, a K pedig a már előbb definiált kernel.



4.1. ábra. Konvolóciós számítás

4.2. Összevonás

Az összevonás (pooling) egy másik, a képen végzett konvolúcióhoz hasonló művelet. Néha a képre alkalmazott konvolúció nem csökkenti eléggé a képet, ezért tovább használhatjuk a pooling műveleteket. A pooling sokkal egyszerűbb, mint a konvolúció alkalmazása, mivel nincs egy elemekkel teli kernelünk. Ehelyett van egy szűrőnk, amelyet végigcsúsztatunk a képen, és a szűrő minden egyes pozíciójában egyszerűen a szűrő alá eső képpontok halmazának minimumát, maximumát, átlagát vagy mediánját vesszük. A legjellemzőbb, hogy egy max pool-t használunk 2x2 szűrővel és 2x2 lépcsővel, úgy, hogy a szűrők között nincs átfedés. A max pool használata jobbnak tűnt, mint az átlagos pool használata.



4.2. ábra. Egy 2x2-es szűrő, 2 egység lépéssel, mely maximra összegez [14]

Minden egyes művelet után némi dimenziócsökkentést kapunk. Ez a dimenziócsökkentés segít abban, hogy kevesebb súlyt kelljen megtanulni, mivel most kevesebb adatunk van, és az adatok az eredeti bemenet összefoglalása. A konvolúciókkal és a poolokkal olyan kerneleket és szűrőket alkalmazunk, amelyek a teljes képen átcsúsznak. Tehát annak ellenére, hogy kevesebb adatunk van, az adatokat a kép szétszórt darabjaiból nyerjük ki, így ezek a csökkentett méretű adatok, elég jól reprezentálják a képet, még akkor is, ha nem tudjuk pontosan, hogyan választotta a reprezentációt.

Ez a módszer nem csak a mélytanulási algoritmust optimalizálja, hanem a modellnek alapvető csoportosításokat is megtanít egy kétdimenziós képen belül (eltolási-invariancia). A képen belüli körvonalakat ezeken a rétegeken keresztül tanulja meg. Megtanulhatja, hogyan társítson egy fényes sárga gömböt az égen azzal a viselkedéssel, hogy árnyékot vet más tárgyakra. Ezek mind olyan kapcsolatok egy kétdimenziós téren belül, amelyeket a konvolúciókkal meg lehet ragadni, de ugyanezek a kapcsolatok nem biztos, hogy detektálhatók, ha a képet mint mátrixot, vektorba rendezzük.

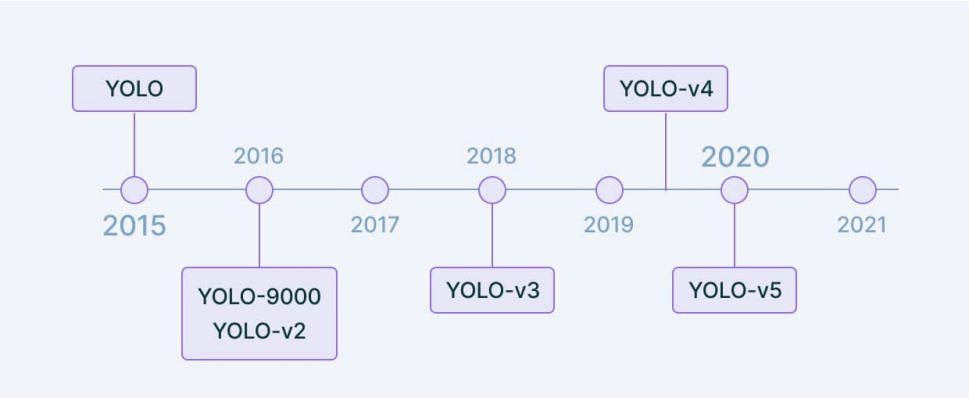
A CNN-ben a konvolúciókat alternatív módon úgy is elképzelhetjük, hogy minden egyes kernelt neuronként kezelünk. Úgy gondolhatjuk, hogy minden egyes kernel vagy neuron felelős a képről való tanulásért. A feladatunktól függően meghatározott számú kimeneti neuronunk lesz. A végső célunk a kimeneti neuronok súlyainak optimalizálása. Tehát minden egyes kimeneti neuron a kép egy adott tulajdonságához (éles élek, csáp, mintázat) tartozó, meghatározott súlykészletet fog megtanulni a teljes kép letapogatásával.

5. fejezet

A YOLO algoritmus

A You Only Look Once (YOLO) egy végponttól végpontig működő neurális hálózatot használ, 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. Az end-to-end neurális hálózat a gépi tanulás vagy a mélytanulás olyan típusára utal, amelyet úgy terveztek, hogy egy teljes feladatot az elejétől a végéig kezeljen anélkül, hogy különálló, kézzel készített komponensekre vagy funkciótervezésre támaszkodna. Más szóval a hálózat nyers bemeneti adatokat vesz fel, és közvetlenül, köztes feldolgozási lépések nélkül állítja elő a kívánt kimenetet.

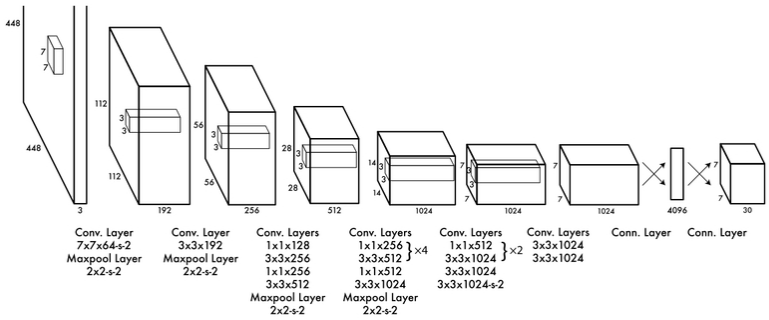
A YOLO a tárgyak észlelésének alapvetően eltérő megközelítését követve a legkorszerűbb eredményeket érte el, és nagy különbséggel verte meg a többi valós idejű tárgyfelismerő algoritmust. Míg az olyan algoritmusok, mint az RCNN (Region Based Convolutional Neural Networks) úgy működnek, hogy a Region 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-öt használó módszerek több ismétlést végeznek ugyanazon a képen, míg a YOLO egyetlen ismétléssel boldogul. 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 (5.1), amely a YOLO elmúlt évekbeli fejlődését mutatja be.



5.1. ábra. A YOLO algoritmus verziói az évek során [8]

5.1. A YOLO algoritmus működése

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 (5.2. ábra).



5.2. ábra. YOLO algoritmus működési ábrája

Az architektúra a következőképpen működik:

1. Bemeneti átméretezés: A bemeneti kép mérete kezdetben 448x448 képpontra van átméretezve. Ez a szabványosítás segít biztosítani a neurális hálózat egységes bemeneti méretét.

2. 1x1 konvolúció: Az átméretezett képre 1x1 konvolúciós műveletet alkalmazunk. Ez a művelet egy olyan matematikai művelet, amelyben egy 1x1 szűrő konvolválódik a bemeneti képpel. A bemenet csatornáinak (vagy mélységének) számának csökkentésére szolgál, hatékonyan tömörítve az információt.

3. 3x3 konvolúció: Az 1x1-es konvolúciót követően 3x3-as konvolúciót alkalmazunk. Ez a művelet az előző lépésben kapott csökkentett csatornareprezentációból vonja

ki a jellemzőket. A kimenetet kocka alakúnak írják le, ami azt jelzi, hogy három dimenzióval rendelkezik, amelyek megfelelnek a magasságnak, szélességnek és mélységnek.

4. Aktiválási függvény (ReLU): A Rectified Linear Unit (ReLU) az egész hálózatban használt aktiválási függvény, kivéve az utolsó réteget. A ReLU nemlinearitást vezet be a modellbe azáltal, hogy pozitív értékek esetén a bemenetet adja ki, negatív értékek esetén pedig nullát. Ez segít a hálózatnak megtanulni az adatokban lévő összetett mintákat és összefüggéseket.

5. Lineáris aktiválás az utolsó rétegben: A hálózat utolsó rétege lineáris aktiválási függvényt használ. A ReLU-val ellentétben a lineáris aktiválási függvény a bemenet súlyozott összegét adja ki nemlineáris transzformáció alkalmazása nélkül. Ez gyakori a regressziós feladatokban, vagy amikor a hálózatnak korlátlan kimeneti értékeket kell produkálnia.

6. Szabályozási technikák: Tételes normalizálás: A tételes normalizálás a neurális hálózatok képzésének stabilizálására és felgyorsítására használt technika. Az egyes rétegek bemenetét normalizálja az aktivációk beállításával és skálázásával.

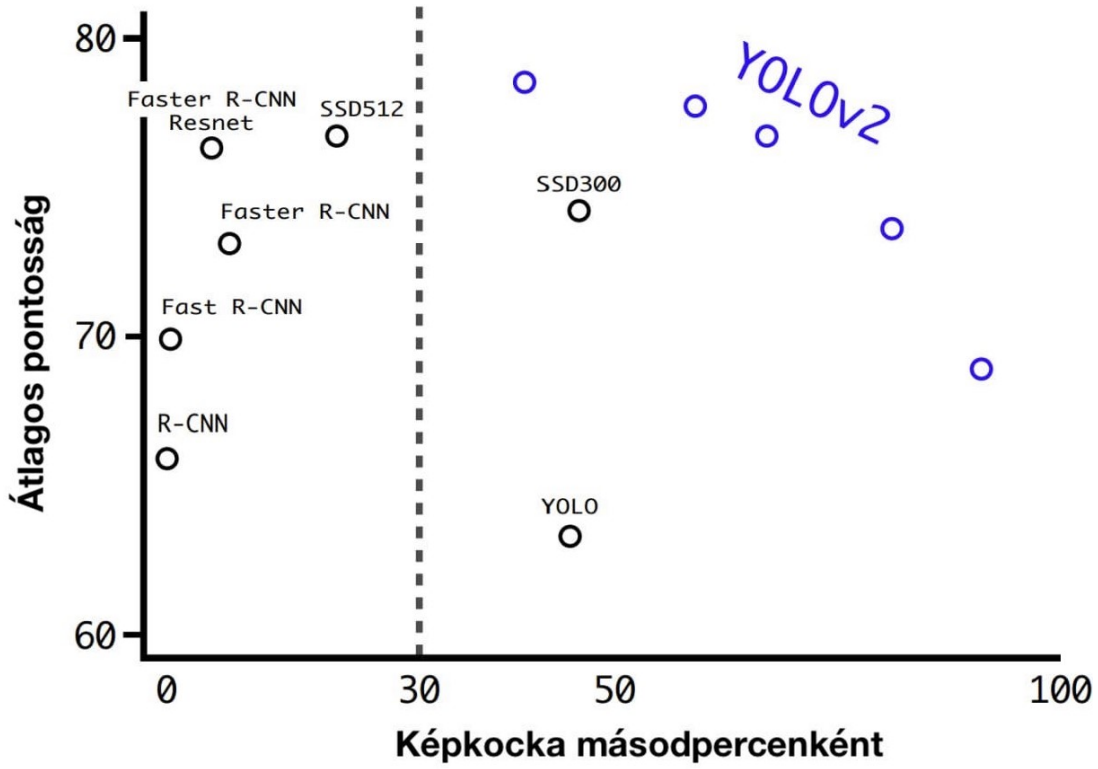
7. Kiesés: A kiesés egy olyan regularizációs technika, amely segít megelőzni a túlillesztést. A neuronok egy részét véletlenszerűen ejti ki (nullára állítja) a képzés során, így kényszerítve a hálózatot, hogy robusztusabb és általánosíthatóbb jellemzőket tanuljon.

A YOLO-modellekben alkalmazott egyik kulcsfontosságú technika a nem maximális elnyomás (NMS). Az NMS egy utófeldolgozási lépés, amely az objektumfelismerés pontosságának és hatékonyságának javítására szolgál. Az objektumfelismerés során gyakran előfordul, hogy egy képen lévő egyetlen objektumhoz több határoló doboz generálódik. Ezek a határoló dobozok átfedhetik egymást, vagy különböző pozíciókban helyezkedhetnek el, de mind ugyanazt az objektumot ábrázolják. Az NMS a felesleges vagy hibás határoló dobozok azonosítására és eltávolítására, valamint a képen lévő minden egyes objektumhoz egyetlen határoló doboz kimenetére szolgál.

5.2. Az algoritmus későbbi fejlesztései

A YOLO v2 vagy YOLO9000 2016-ban került bevezetésre. A legfontosabb fejlesztések közé tartozik a határoló dobozok előrejelzésére szolgáló horgonydobozok bevezetése, ami lehetővé teszi a különböző méretű és oldalarányú objektumok felismerését. A horgonydobozok előre meghatározott méretű és képarányú határoló dobozok. Ezeket a

dobozokat a képzés előtt határozzuk meg, és referenciasablonként szolgálnak a modell számára, hogy a képzés és a következtetés során megjósolja a határoló dobozokat. Az ötlet lényege, hogy a horgonydobozok használatával a modell jobban tud alkalmazkodni a különböző formájú és méretű objektumokhoz. Az eredmények (5.3. ábra) bizonyítják a YOLO v2 fölényét az eredeti verzióhoz és más kortárs modellekhez képest.



5.3. ábra. YOLOv2 teljesítményének összehasonlítása más algoritmusokkal

Az egyik fejlesztés a YOLO v3-ban a különböző méretarányú és oldalarányú horgonyzó dobozok. A YOLO v2-ben a horgonyzódobozok mind egyforma méretűek voltak, ami korlátozta az algoritmus képességét a különböző méretű és alakú objektumok felismerésére. A YOLO v3-ban a horgonydobozok méretaránya és a képarányok változnak, hogy jobban illeszkedjenek az észlelt objektumok méretéhez és alakjához. Ezen fejlesztések mellett a YOLO v3 a tárgyak méretének és képarányának szélesebb skáláját tudja kezelni. Emellett pontosabb és stabilabb, mint a YOLO korábbi verziói.

A YOLO v4 egy új módszert vezet be a horgonyzó dobozok létrehozására, az úgynevezett "k-means klaszterezést". Ennek lényege, hogy egy klaszterező algoritmus segítségével az alaphelyzetben lévő határoló dobozokat klaszterekbe csoportosítjuk, majd a klaszterek középpontjait használjuk horgonyzódobozként. Ez lehetővé teszi, hogy a horgonyzódobozok jobban igazodjanak az észlelt objektumok méretéhez és alakjához.

A YOLO v5 egy új módszert használ a horgonyzó dobozok létrehozására, az úgynevezett "dinamikus horgonyzó dobozokat". Ennek lényege, hogy egy klaszterező algo-

ritmussal klaszterekbe csoportosítja az alaphelyzetben létező határoló dobozokat, majd a klaszterek középpontjait használja horgonyzódobozként. 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 a kis objektumok észlelési teljesítményének javítására szolgál, mivel lehetővé teszi, hogy a modell több léptékben is 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.

6. fejezet

Neurális hálózat betanítása

A betanított neurális hálózat elkészítése több szakaszból áll, úgy mint az adatgyűjtés, tanítás és validáció. Ebben a fejezetben ezen lépések megvalósítását részletezem.

6.1. Adatgyűjtés

A betanításhoz nagy adathalmazra van szükség. Ebben az esetben egy kiválasztott faj, az ékfoltos zengőlégy (Episyrphus balteatus), felismerését szeretném megvalósítani. A Global Biodiversity Information Facility weboldalán az egyes fajokról sok információ érhető el úgy, mint a megjelenés gyakorisága és helye és számos kép letölthető, melyeket a felhasználók töltenek fel. Egy fajról minden információ elérhető egyetlen szöveges dokumentumban (6.1. ábra), melyben piros aláhúzással jelöltem azt a linket, mely egy képet tartalmaz.

```
2014-10-13                          © all rights reserved   Tuula Jacobsson
1038641284                          https://naturgucker.net/?sprache=en&bild=633553690
1038641423                          https://naturgucker.net/?sprache=en&bild=-1945366816
1038641518                          https://naturgucker.net/?sprache=en&bild=-819084555
1038641612                          https://naturgucker.net/?sprache=en&bild=1383512443
1038641612                          https://naturgucker.net/?sprache=en&bild=1508687308
1038642053                          https://naturgucker.net/?sprache=en&bild=-1756682434
1038642206                          https://naturgucker.net/?sprache=en&bild=1129582237
1038642206                          https://naturgucker.net/?sprache=en&bild=1493298475
1038642206                          https://naturgucker.net/?sprache=en&bild=-84900695
1038642355                          https://naturgucker.net/?sprache=en&bild=1840664071
1090401943      StillImage      image/jpeg      https://www.artportalen.se/MediaLibrary/2015/5/844cd292-80f4-4705-9532-5a68b820a0dd_image.jpg   https://www.artportalen.se/Image/1389654
2015-05-28                          © all rights reserved   Anders Wånge Kjellsson
1123738184      StillImage      image/jpeg      https://www.artportalen.se/MediaLibrary/2015/7/dca9244c-5dce-49ac-ae82-061e2bd9d524_image.jpg   https://www.artportalen.se/Image/1421961
2015-07-19                          © all rights reserved   Fredrik Andersson
1132403058      StillImage      image/jpeg      https://inaturalist-open-data.s3.amazonaws.com/photos/2091043/original.JPG      https://www.inaturalist.org/photos/2091043
05T14:38:55-07:00       Jakob Fahr              iNaturalist     http://creativecommons.org/licenses/by-nc/4.0/  Jakob Fahr
1132406483      StillImage      image/jpeg      https://inaturalist-open-data.s3.amazonaws.com/photos/2149388/original.JPG      https://www.inaturalist.org/photos/2149388
18T12:50:12-07:00       Jakob Fahr              iNaturalist     http://creativecommons.org/licenses/by-nc/4.0/  Jakob Fahr
1132406483      StillImage      image/jpeg      https://inaturalist-open-data.s3.amazonaws.com/photos/2149387/original.JPG      https://www.inaturalist.org/photos/2149387
18T12:49:52-07:00       Jakob Fahr              iNaturalist     http://creativecommons.org/licenses/by-nc/4.0/  Jakob Fahr
1132409854      StillImage      image/jpeg      https://inaturalist-open-data.s3.amazonaws.com/photos/2159816/original.jpg      https://www.flickr.com/photos/dhobern/19678899418/
2015-07-18T16:59:44-07:00       Donald Hobern           iNaturalist     http://creativecommons.org/licenses/by/4.0/     Donald Hobern
1132460591      StillImage      image/jpeg      https://www.artportalen.se/MediaLibrary/2015/7/49cfd1c8-0af5-411b-9703-a483b5dc8853_image.jpg   https://www.artportalen.se/Image/1427575
2015-07-27                          © all rights reserved   Bjarne Andersen
1132460591      StillImage      image/jpeg      https://www.artportalen.se/MediaLibrary/2015/7/0ec5f802-44d3-43a1-8b0b-b1cbf307ef71_image.jpg   https://www.artportalen.se/Image/1427577
2015-07-27                          © all rights reserved   Bjarne Andersen
1132466946      StillImage      image/jpeg      https://www.artportalen.se/MediaLibrary/2015/7/892393a3-e617-4ef6-a0f8-c4314fcf610c_image.jpg   https://www.artportalen.se/Image/1427847
2015-07-27                          © all rights reserved   Veronica Budin
1132467184      StillImage      image/jpeg      https://www.artportalen.se/MediaLibrary/2015/7/50462b6b-72b3-4150-8cbd-774035806e50_image.jpg   https://www.artportalen.se/Image/1427791
2015-07-27                          © all rights reserved   Joakim Lindblom
```

6.1. ábra. Adathalmaz, mely a képek linkjét is tartalmazza

Ezen képek egyesével való letöltése rengeteg időt venne igénybe, ezért ezt a feladatot egy egyszerű kóddal automatizáltam. Az első kódrészlet a fentebb említett dokumentumból kiszűri a képek linkjeit, ennek feltétele, hogy az adott szövegtöredék .jpg végződésű legyen. A linkeket egy „links” nevű változóba elmentem.

```
import numpy as np
from google.colab import drive
drive.mount('/content/drive')

picnumber=0
tx = np.genfromtxt('/content/drive/MyDrive/linkCleaning/multimedia.csv', encoding='utf8', delimiter='\t',
    dtype=str, usecols=np.arange(0,4))
r = np.char.endswith(tx[:,3], '.jpg')
index = np.where(r==True)
links=np.array(tx[index, 3])
print(links)
print(links[0,1])
```

Az előbb létrehozott „links” nevű tömb elemein az alábbi kód végighalad és minden linket található képet letölt a google colab ideiglenes mappájába.

```
import requests
i=300
while i < picnumber:
url = links[0,i]
response = requests.get(url)
with open("ims/image" + str(i+1) + '.jpg', "wb") as f:
f.write(response.content)
i = i+1
```

A képeket az alábbi program becsomagolja egy fájlba és letölti.

```
from google.colab import files
!zip -r /content/ims.zip /content/ims
files.download("/content/ims.zip")
```

Ezzel a képek rendelkezésre állnak. A 300 darab kép közül azonban nem mindegyik használható. Egy gyors szortírozással azokat, melyeken homályosan, részlegesen, vagy nem megfelelően szerepel a légy, kitöröltem.

6.2. Címkézés

A képeket a következő lépésben fel kell címkézni, vagyis megjelölni, hogy hol van rajtuk az objektum, amiket fel szeretnénk ismerni. Ezt automatizálni nem lehet, egyesével kell a mintákon a jelöléseket elvégezni. Ehhez a Makesense ingyenes webalkalmazást használtam .

6.2. ábra. Egy tanításhoz szükséges kép címkézve

A képek címkézése után, az adathalmazt az alkalmazott modellel kompatibilis formátumban kell elmenteni. Ebben az esetben ez a formátum a következő:

```
0 0.553516 0.543679 0.216598 0.456321
```

Az első szám (0), a képen megjelölt objektum osztálya. Ennek akkor van jelentősége, ha különböző objektumokhoz tartozó címkéket is alkalmazunk. Az ezt követő négy szám, a legyet befoglaló négyzet koordinátáit jelöli.

Továbbá, a validációhoz szükséges képeken is elvégeztem a címkézést. Ezen képek száma kevesebb, 40 db.

6.3. File-ok és környezet előkészítése

A YOLO algoritmus, az Ultralytics által készített Google Colaboratory jegyzetfüzettel könnyen használható, azonban nincs a teljes kódra szükség. Az első szükséges lépés, hogy a futtatókörnyezetet, a könyvtárakat és a külső kódokat importáljuk:

```
!git clone https://github.com/ultralytics/yolov5  # clone
%cd yolov5
%pip install -qr requirements.txt comet_ml  # install

import torch
import utils
display = utils.notebook_init()  # checks
```

A képeket és a címkéket, a következő mappa struktúrába kell rendezni:

```
train_data
├── labels
│   ├── train
│   └── val
└── ims
    ├── train
    └── val
```

A legutolsó mappák (*train*, *val*), tartalmazzák a képeket és a szöveges dokumentumokat. A *train_data* fő mappát kell a Colaboratory mappához hozzáadni, melyet legegyszerűbben úgy tudunk, hogy .zip file-ba tömörítjük és feltöltjük. Ezt azonban a feltöltés után ki kell csomagolni, hogy kezelni tudjuk, amit a következő sor hajt végre:

```
!unzip -q /train_data.zip -d ../
```

A tanítás megkezdése előtt, a *yolov5/data* mappában lévő *coco.yaml* file-t kell módosítani. Meg kell adni a képek útját a betanuláshoz és validációhoz, illetve definiálni kell a a felismerni kívánt objektumot egy osztályként:

```
path: ../train_data  # dataset root dir
train: images/train/  # train images (relative to 'path') 128 images
val: images/val/  # val images (relative to 'path') 128 images

# Classes
names:
0: Episyrphus balteatus
```

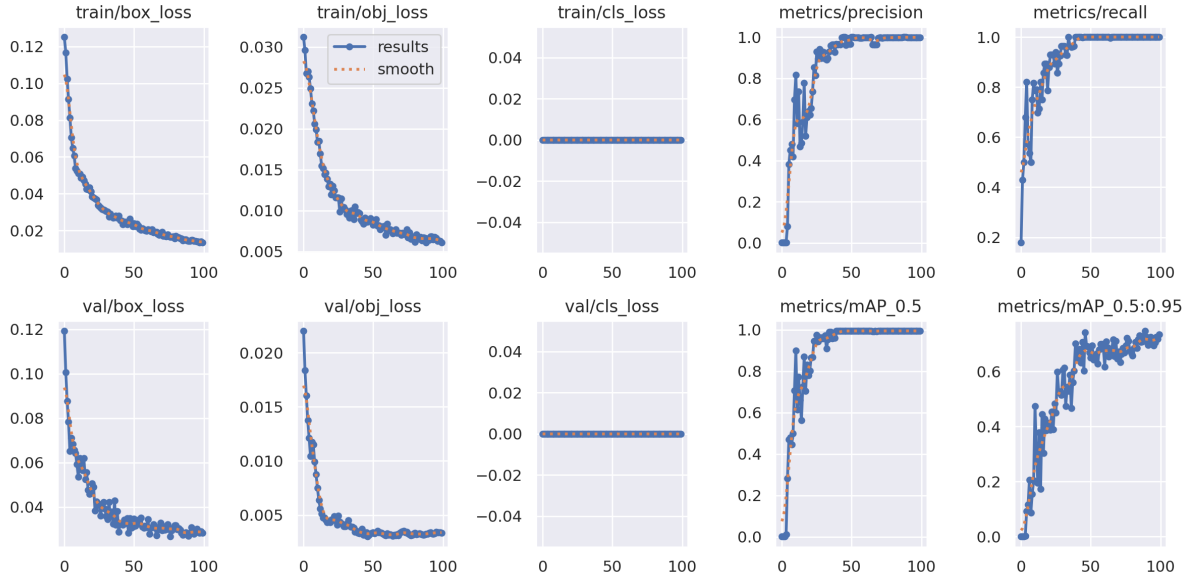
Ezt a dokumentumot *custom_data* néven visszatöltöm eredeti helyére. Ezzel minden készen áll a tanítás megkezdéséhez.

6.4. Neurális hálózat betanítása

A betanítás megkezdéséhez futtatni kell a következő sort:

```
!python train.py --img 640 --batch 64 --epochs 100 --data custom_data.yaml --weights yolov5s.pt --cache
```

A folyamat a paraméterektől függően, több időt is igénybe vehet. A fenti paraméterekkel, az említett képek mennyiségével ez 15-20 percig tartott. Előfordulhat, hogy a sor futtatása nem sikerül és hibát dob a program: *RuntimError: CUDA out of memory*. Ennek leggyakoribb oka, hogy a rendelkezésre álló memória elfogy. Ez úgy orvosolható, hogy a *batch* méretét csökkentjük.

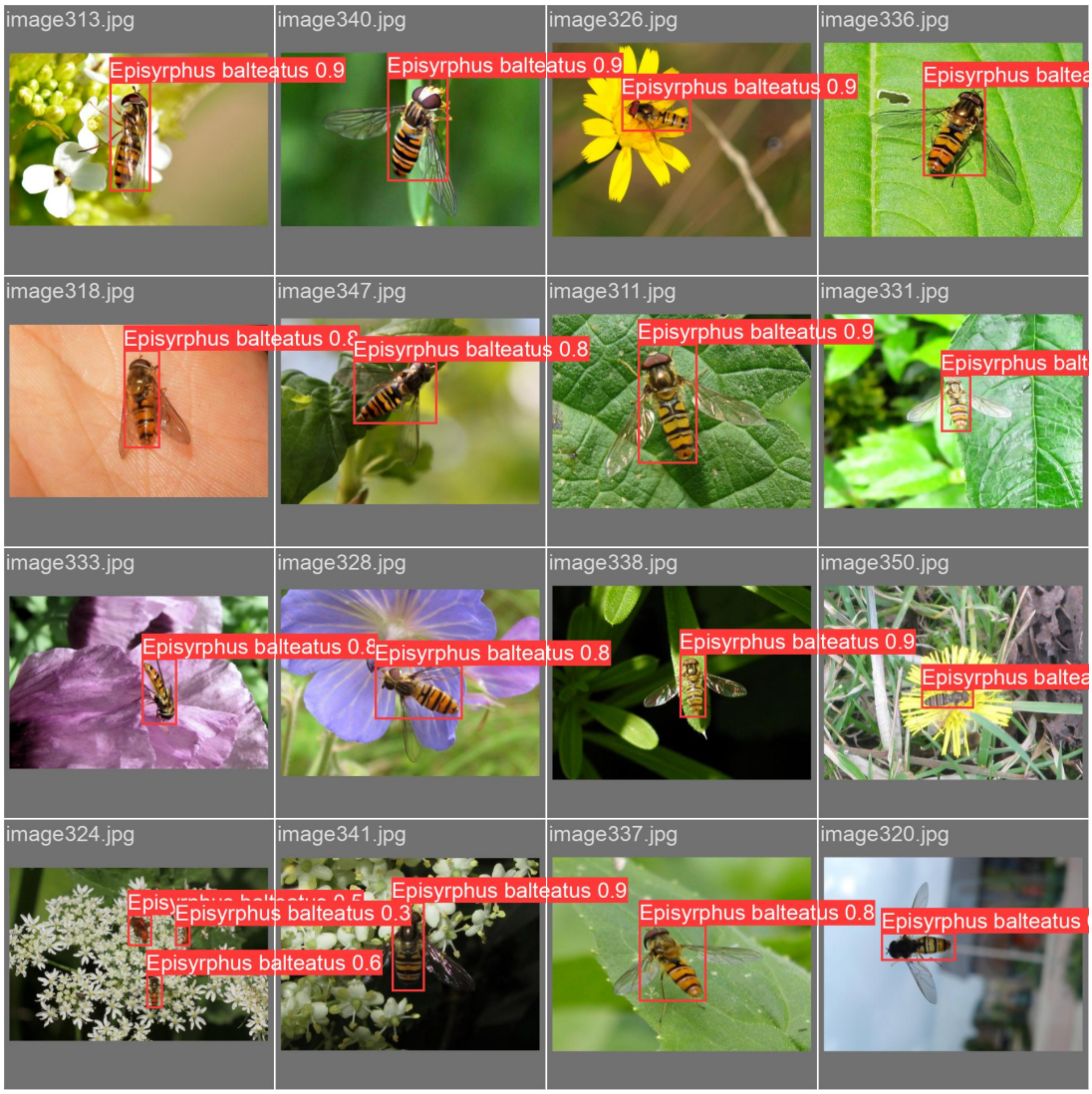


6.3. ábra. Neurális hálózat tanítási eredményei

A tanítási folyamatból több adat is kinyerhető, ezeket a 6.3. ábra foglalja össze. Minden ábra adatokat szolgáltat a 100. *batch*-ig. A dobozvesztés (*box_loss*) egy speciális veszteségfüggvényen alapuló veszteségmetrika, amely azt méri, hogy a megjósolt határoló dobozok mennyire „szorosan” illeszkednek az eredeti objektumokhoz, az adatkészlet képein lévő címkékhez. Egy alacsonyabb érték azt jelzi, hogy a modell javul az általánosítás tekintetében, és jobb határoló dobozokat hoz létre az adatkészletben azonosított objektumok körül. Az objektivitásveszteség (*obj_loss*) a modell előre jelzett kimenete és a tényleges értékek közötti különbséget méri, ez kifejezetten annak meghatározására összpontosít, hogy egy objektum jelen van-e egy adott határoló doboz javaslatban. Az osztályozási veszteség (*cls_loss*) a megjósolt határoló doboz osztályozásának helyességét méri. Jelen esetben egy osztály van, így ennek nincs jelentősége. A precízió (*precision*) érték úgy számolható, hogy az igaz pozitív eredményeket elosztjuk az összes jósolt pozitív eredménnyel. Értelemszerűen a hibásan megjelölések számának csökkentésével ez

az érték jó esetben 1-hez tart. A visszahívás (*recall*) annak értékelésére szolgál, hogy egy előrejelző rendszer eléggé találékony-e. Az igaz pozitív találatokat elosztjuk az összes lehetséges igaz pozitív találattal. Az mAP azt mutatja meg, hogy a modell mennyire helyesen lokalizálja az objektumokat az általa feldolgozott képeken, különböző átütési értékeknél. Ez az eredmény, az előre meghatározott és a jósolt objektumokat befoglaló dobozok átfedéséből adódik.

A validációhoz elkészített címkékből és képekből álló adatok alapján ellenőrizhető az eredmény. A 6.4. képen látható, hogy egész magas, 80-90%-os valószínűséggel jelölte meg a legyet a képeken. Ezen adatok szerint, a tanítás sikeres volt, minden hiba a 100. *batch*-re eléri a kellően alacsony szintet.



6.4. ábra. Validáció során megjósolt objektumok

7. fejezet

Objektum detektálás és az adatok további feldolgozása

7.1. Modell tesztelése valós minta videón

A betanított modell készen áll a tesztelésre, melyet az alábbi kóddal tudunk megtenni:

```
!python detect.py --weights yolov5/runs/train/exp2/weights/last.pt --img 640 --conf 0.25 --source ../
  video.mp4 --save-csv
```

A *detect.py* kódsor felelős a teljes program lefuttatásáért, beleértve a videó betöltését, kiértékelését, az eredmények dokumentálását. A *--conf* parancs után álló szám egy küszöbértéket jelent, mely alatt a megjósolt objektumot a program elveti és nem veszi figyelembe. Ez esetben, ha csak egy osztályunk van és nincs számottevő téves megjelölés, maradhat alacsony szinten. Azonban ha több osztály is van, melyek hasonlítanak egymásra, tévesen összekeveredhetnek a jelölések. A *--source* után álló argumentum lehet akár kép, akár videó formátum. Az eredmények a *--save-scv* parancs segítségével egy könnyen kezelhető *csv* file-ba menthető, később ezzel dolgozom tovább.

7.1. ábra. Két különböző képkockán való tesztelés eredménye. Látható, hogy mindkét esetben magas valószínűséggel sikerült a kívánt pollinátort detektálni

7.2. Mentett adatok feldolgozása

A program által mentett *csv* file a következő formátumú:

```
video.mp4,Episyrphus balteatus,0.51
video.mp4,Episyrphus balteatus,0.92
video.mp4,Episyrphus balteatus,0.74
video.mp4,Episyrphus balteatus,0.94
video.mp4,Episyrphus balteatus,0.95
video.mp4,Episyrphus balteatus,0.93
...
```

Minden egyes sor tartalmazza a file-t, amit a program beolvasott, a detektált objektum osztályát, illetve annak valószínűségét, hogy az adott osztály szerepel rajta. Ebből a file-ból különböző statisztikai adatokat tudunk kinyerni. Amennyiben tudjuk, hogy az input file-két szolgáló videó mekkora képkocka/másodperc-cel rendelkezik, egyszerűen számolható az az idő, amit az objektum a látómezőben tartózkodott.

```
from google.colab import drive
drive.mount('/content/drive')
import csv
import numpy as np

source = []
species = []
conf = []

csv_file_path = '/content/drive/MyDrive/OE_final_data/predictions.csv'

with open(csv_file_path, 'r') as file:
csv_reader = csv.reader(file)
for row in csv_reader:
source.append(row[0])
species.append(row[1])
conf.append(row[2])
conf = (np.asarray(conf)).astype(np.float)
print('Összes detektált képkocka: ',conf.size)
print('Átlagos konfidencia: ',np.mean(conf))
print('Konfidencia minimum: ',np.amax(conf))
print('Konfidencia maximum: ',np.amin(conf))
```

A fentebb látható kód beolvassa a *prediction.csv* file-t, amely az adatokat tartalmazza. Ezt oszlopokra bonja és ezeknek megfelelő három változóban tárolja. Kiírja azon képkockák számát, melyeken objektumot észlelt, a valószínűségek átlagát, maximumát és minimumát. Az output az alább látható:

```
Output:
Összes detektált képkocka:  244
Átlagos konfidencia:  0.9086475409836064
Konfidencia minimum:  0.51
Konfidencia maximum:  0.95
```

Ezek alapján meghatározható az összes eltöltött másodperc:

$$t = \frac{\text{Összes detektált képkocka}}{\text{A videó FPS értéke}} = \frac{244}{30} = 8,13\ [sec] \tag{7.1}$$

A kiértékelt file Excel táblázatba is könnyen importálható és kezelhető, tetszőleges statisztikai értékek kinyerésére alkalmas.

8. fejezet

Videorögzítő hardver

Az elkészített felismerő szoftver egy viszonylag nagy számítási kapacitást igénylő rendszer, melyhez szükség van teljes számítógépre. Azonban a videóanyag rögzítéséhez nincs szükség professzionális felvevő rendszerre, egy nyomtatott áramkörre szerelt kamera teljesen elegendő erre a célra. Ebben a fejezetben szeretném bemutatni, hogy milyen kihívásokkal kell a kamerának szembenéznie, illetve ezekre milyen megoldásokat javaslok.

8.1. A hardverrel szemben támasztott követelmények

A kamera a tervek szerint szabadban, azonban az ATK Növényvédelmi Intézet telephelyén lenne üzembe helyezve. Ezek alapján a következő környezeti hatásokra lehet számítani, ha a legszélsőségesebb adatokat vesszük alapul:

- magas hőmérséklet: 30-35 [°C]
- szél
- napsütés
- eső: 20-40 [mm/nap]

(*forrás: https://www.met.hu/*)

A kamerának rendelkeznie kell megfelelő felbontással, energiaellátással, illetve adatátviteli kapcsolattal, a videófelvételek továbbítása érdekében.

8.2. Kamera modul

Az OpenMv kínálatából választható Cam H7 megfelelő lehet a feladat ellátására. Ez a cég több, kis méretű, nyomtatott áramköre szerelt egységet forgalmaz. Ezek a termékek megfizethető áron kínálnak megoldást computer vision feladatokra. Programozásuk Python nyelvben is lehetséges, erre és további kezelésükre rengetek forrás áll rendelkezése.

8.1. ábra. Cam H7 lapra szerelt kamera modul

Termék főbb katalógus adatai:

Tárolási hőmérséklet	-40–125 °C
Működési hőmérséklet	-20–70 °C
Videofelvétel minősége	Grayscale: 640x480, RGB565: 320x240
Energiafogyasztás	110mA @ 3.3V

Az adatok szerint a működési hőmérséklet és a felbontás is megfelelő az adott feltételeknek.

8.3. Adatok mentése

Az adatok mentése történhet memóriakártyára, illetve közvetlenül továbbítható is egy szerverre vagy közvetlenül a számítógépre, amely feldolgozza azokat. A Wfifi egység azonban nem tartozéka a fő kamera modulnak. Az OpenMV katalógusában található, a kamera modullal kompatibilis csatlakozású Fifi modul, mely megoldás lehet erre a feladatra.

8.2. ábra. OpenMV Wifi Shield

Tárolási hőmérséklet	-40–85 °C
Működési hőmérséklet	-20–85 °C
Adatátviteli sebesség	IEEE 802.11n (akár 48Mbps)
Energiafogyasztás	220 mA @ 3.3V kamera modullal együtt

A fentebb említett adatátviteli sebesség bőven elegendő a videófelvétel, vagy akár az élő kép továbbítására is.

8.4. Áramellátás

A végleges alkalmazás során a videórögzítés szünetmentes lesz és ehhez szünetmentes áramellátás is szükséges. Azonban a teszt fázisban elegendő egy ideiglenes áramellátás, amit egy akkumulátor biztosítana. A választás a lítiomion típusú akkumulátorra esett, a lítium polimerrel szemben, mivel az előbbi nagyobb energiasűrűséggel és biztonságosabb működéssel rendelkezik.

8.3. ábra

Feszültség	3,7 V
Kapacitás	1000 mAh
Kisülési áramerősség	550 mA

Ez az akkumulátor az adatok szerint a fentebb említett kamera modult képes ellátni energiával. 1000 mAh kapacitással és 220 mA fogyasztással számolva, a működés névlegesen 4 és fél órán át biztosított. Természetesen ez változhat a felhasználástól és a környezeti hatásoktól függően, de a rendszer tesztelésére megfelel.

8.5. Rögzítés

A kamera modul számára a gyártó oldalán elérhető egy tok. Azonban ez nem megfelelő, ha a WiFi modul és az akkumulátor is a rendszer része. Ezen okokból kifolyólag egy egyedi tartót javaslok, mely megfelelő tömítéssel is rendelkezik a vízkizárás érdekében. Ennek gyártása additív technológiával történhet, melynek költsége elenyésző. Ennek tervezését dolgozatomban nem taglalom.

8.6. A tesztrendszer költsége

Ebben a fejezetben összefoglaltam a rendszerhez szükséges elemeket, melyekkel a szoftver valós körülmények között tesztelhető. A mérnöki gyakorlatban nem elhanyagolható, hogy a megvalósítandó terveknek milyen anyagi vonzatai vannak. Ezért dolgozatomat a hardver költségeinek összegzésével zárom.

Termék	Ár (USD)
OpenMV Cam H7	85
OpenMV WiFi Shield	30
3.7V Li Battery	10.16
Összesen	125.16

Ez a végösszeg 2023. november 29-én, 344,61 Ft-os árfolyamon 43 135 Ft. Azonban ehhez még hozzájön a szállítási, illetve a tok gyártási költsége, 50-55 ezer Forintos végösszeggel érdemes számolni.

9. fejezet

Összegzés

Dolgozatom témája a mesterséges intelligenciával és mélytanulással segített képfelismerés, mely aktívan kutatott terület és implementálása számos különböző ipari alkalmazásba, mindennapi kihívást jelent. Ezen okokból kifolyólag örülök, hogy ezen a témán dolgozhattam és oldhattam meg egy valós problémát. Dolgozatom elején, számos irodalmi forrással támasztottam alá a téma relevanciáját a 21. században. Több, különböző iparági alkalmazást bemutattam, kiemelve az orvostechnológiai felhasználásokat.

Az objektumdetektálás alapfogalmai és matematikai alapjainak megértése elengedhetetlen a probléma megoldásához. Ezeket a különböző algoritmusokra sajátosságaival együtt összefoglaltam és segítségükkel feltártam az egyes algoritmusok hibáit, kiválasztottam az aktuális feladatra megfelelőt.

A nagy méretű adathalmaz elengedhetetlen része a mélytanulásos algoritmusoknak. Az adatok gyűjtését, a megfelelő források felhasználásával automatizáltam. A begyűjtött, felismerni kívánt objektumot tartalmazó képeket a megfelelő módon feldolgoztam és betanítottam az algoritmust. Folyamatos, 80% fölötti megbízhatósági eredménnyel zárt a YOLO algoritmus. A videó a https://youtu.be/ngKnL8DkLu8 linkre kattintva megtekinthető. A felismerő rendszerhez szükséges alapvető hardverekkel és ezek anyagi vonzataival zártam dolgozatomat.

10. fejezet

Summary

The topic of my thesis is artificial intelligence and deep learning assisted image recognition, which is an actively researched field and its implementation in many different industrial applications is a daily challenge. For these reasons, I am pleased to have been able to work on this topic and solve a real-world problem. At the beginning of my thesis, I used several literature sources to support the relevance of this topic in the 21st century. I presented several applications in different industries, highlighting medical technology applications.

An understanding of the basic concepts and mathematical foundations of object detection is essential to solve the problem. I have summarized these, along with the specificities of the different algorithms, and used them to identify the flaws in each algorithm, selecting the one that is appropriate for the task at hand.

Large data sets are an essential part of deep learning algorithms. I have automated the data collection, using the appropriate resources. I processed the collected images containing the object to be recognized in the appropriate way and trained the algorithm. The YOLO algorithm closed with a continuous reliability result above 80%. The video can be viewed by clicking on the link: https://youtu.be/ngKnL8DkLu8. I concluded my thesis with the basic hardware and its financial implications for the recognition system.

Irodalomjegyzék

[1] Ajay Agrawal – Joshua Gans – Avi Goldfarb: *Prediction Machines, Updated and Expanded: The Simple Economics of Artificial Intelligence*. 2021.

[2] Abhishek Balasubramaniam – Sudeep Pasricha: Object detection in autonomous vehicles: Status and open challenges. *arXiv preprint arXiv:2201.07706*, 2022.

[3] Erik Brynjolfsson – Andrew McAfee: *The second machine age: Work, progress, and prosperity in a time of brilliant technologies.* 2014, W W Norton & Co, 306. p. ISBN 978-0-393-23935-5 (Hardcover).

[4] Haar cascade classifier vs histogram of oriented gradients(hog). `https://medium.com/@goutam0157/haar-cascade-classifier-vs-histogram-of-oriented-gradients-hog-6f4373ca239b`. Utolsó elérés: 2023. 12. 2.

[5] Fei Jiang – Yong Jiang – Hui Zhi – Yi Dong – Hao Li – Sufeng Ma – Yilong Wang – Qiang Dong – Haipeng Shen – Yongjun Wang: Artificial intelligence in healthcare: past, present and future. *Stroke and Vascular Neurology*, 2. évf. (2017) 4. sz., 230–243. p. ISSN 2059-8688. URL `https://svn.bmj.com/content/2/4/230`.

[6] Yoon Hyun Jin – Jeong Young Jin – Kang Hyun – Jeong Ji Eun – Kang Do-Young: Medical image analysis using artificial intelligence. *pmp*, 30. évf. (2019) 2. sz., 49–58. p.
URL `http://www.e-sciencecentral.org/articles/?scid=1129102`.

[7] Kushsairy Kadir – Mohd Kamaruddin – Haidawati Nasir – Sairul Safie – Zulkifli Bakti: A comparative study between lbp and haar-like features for face detection using opencv. 2014. 08, 335–339. p.

[8] Rohit Kundu: Yolo: Algorithm for object detection explained. `https://www.v7labs.com/blog/yolo-object-detection#:~:text=YOLO%20v5%20uses%20a%20new,clusters%20as%20the%20anchor%20boxes.` Utolsó elérés: 2023. 11. 22.

[9] Andréanne Lemay: Kidney recognition in ct using yolov3. 2019.

[10] Kevin J Liang – John Brevard Sigman – Gregory P. Spell – Dan A. Strellis – William Chang – Felix Liu – Tejas Mehta – Lawrence Carin: Toward automatic threat recognition for airport x-ray baggage screening with deep convolutional object detection. *ArXiv*, abs/1912.06329. évf. (2019).
URL `https://api.semanticscholar.org/CorpusID:209370487`.

[11] Geert Litjens – Clara I. Sánchez – Nadya Timofeeva – Meyke Hermsen – Iris Nagtegaal – Iringo Kovacs – Christina Hulsbergen van de Kaa – Peter Bult – Bram van Ginneken – Jeroen van der Laak: Deep learning as a tool for increased accuracy and efficiency of histopathological diagnosis. *Scientific Reports*, 6. évf. (2016. 5), 26286. p. ISSN 2045-2322.

[12] Warren S. McCulloch – Walter Pitts: A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5. évf. (1943. 12), 115–133. p. ISSN 0007-4985.

[13] Altrichter Márta – Horváth Gábor – Pataki Béla: Mesterséges intelligencia elektronikus almanach. `http://project.mit.bme.hu/mi_almanach/node/1`. Utolsó elérés: 2023. 09. 23.

[14] Amal Nanavati: Chapter 18 convolutional neural networks. `https://courses.cs.washington.edu/courses/cse416/22su/lectures/10/lecture_10.pdf`. Utolsó elérés: 2023. 11. 2.

[15] Robert M. Nishikawa: Current status and future directions of computer-aided diagnosis in mammography. *Computerized Medical Imaging and Graphics*, 31. évf. (2007) 4. sz., 224–235. p. ISSN 0895-6111. URL `https://www.sciencedirect.com/science/article/pii/S0895611107000171`. Computer-aided Diagnosis (CAD) and Image-guided Decision Support.

[16] Graham Oppy – David Dowe: The Turing Test. In Edward N. Zalta (szerk.): *The Stanford Encyclopedia of Philosophy*. Winter 2021. kiad. 2021, Metaphysics Research Lab, Stanford University.

[17] Alison O'Mara-Eves – James Thomas – John McNaught – Makoto Miwa – Sophia Ananiadou: Erratum to: Using text mining for study identification in systematic reviews: a systematic review of current approaches. *Systematic Reviews*, 4. évf. (2015. 12), 59. p. ISSN 2046-4053.

[18] Diana Papaioannou – Anthea Sutton – Christopher Carroll – Andrew Booth – Ruth Wong: Literature searching for social science systematic reviews: consideration of a range of search techniques. *Health Information & Libraries Journal*, 27. évf. (2009. 10), 114–122. p. ISSN 14711834.

[19] Saba Shafi – Anil V. Parwani: Artificial intelligence in diagnostic pathology. *Diagnostic Pathology*, 18. évf. (2023. 10), 109. p. ISSN 1746-1596.

[20] Domen Tabernik: Traffic-sign detection. `https://www.vicos.si/research/traffic-sign-detection/`. Utolsó elérés: 2023. 12. 01.

[21] Sandra Vieira – Walter H.L. Pinaya – Andrea Mechelli: Using deep learning to investigate the neuroimaging correlates of psychiatric and neurological disorders: Methods and applications. *Neuroscience & Biobehavioral Reviews*, 74. évf. (2017), 58–75. p. ISSN 0149-7634.
URL `https://www.sciencedirect.com/science/article/pii/S0149763416305176`.

[22] Gerit Wagner – Roman Lukyanenko – Guy Paré: Artificial intelligence and the conduct of literature reviews. *Journal of Information Technology*, 37. évf. (2022. 6), 209–226. p. ISSN 0268-3962.

Mellékletek

M.1. Fő program

A képfelismerés során alkalmazott *detect.py* python program, mely bemenetként számos lehetőséget kínál. Az argumentumok között több hasznos funkció is megtalálható a további fejlesztés érdekében.

```
# YOLOv5 by Ultralytics, AGPL-3.0 license
"""
Run YOLOv5 detection inference on images, videos, directories, globs, YouTube, webcam, streams, etc.

Usage - sources:
$ python detect.py --weights yolov5s.pt --source 0                               # webcam
img.jpg                         # image
vid.mp4                         # video
screen                          # screenshot
path/                           # directory
list.txt                        # list of images
list.streams                    # list of streams
'path/*.jpg'                    # glob
'https://youtu.be/LNwODJXcvt4'  # YouTube
'rtsp://example.com/media.mp4'  # RTSP, RTMP, HTTP stream

Usage - formats:
$ python detect.py --weights yolov5s.pt                 # PyTorch
yolov5s.torchscript        # TorchScript
yolov5s.onnx               # ONNX Runtime or OpenCV DNN with --dnn
yolov5s_openvino_model     # OpenVINO
yolov5s.engine             # TensorRT
yolov5s.mlmodel            # CoreML (macOS-only)
yolov5s_saved_model        # TensorFlow SavedModel
yolov5s.pb                 # TensorFlow GraphDef
yolov5s.tflite             # TensorFlow Lite
yolov5s_edgetpu.tflite     # TensorFlow Edge TPU
yolov5s_paddle_model       # PaddlePaddle
"""

import argparse
import csv
import os
import platform
import sys
from pathlib import Path

import torch
```

```
FILE = Path(__file__).resolve()
ROOT = FILE.parents[0]  # YOLOv5 root directory
if str(ROOT) not in sys.path:
sys.path.append(str(ROOT))  # add ROOT to PATH
ROOT = Path(os.path.relpath(ROOT, Path.cwd()))  # relative

from ultralytics.utils.plotting import Annotator, colors, save_one_box

from models.common import DetectMultiBackend
from utils.dataloaders import IMG_FORMATS, VID_FORMATS, LoadImages, LoadScreenshots, LoadStreams
from utils.general import (LOGGER, Profile, check_file, check_img_size, check_imshow, check_requirements,
    colorstr, cv2,
increment_path, non_max_suppression, print_args, scale_boxes, strip_optimizer, xyxy2xywh)
from utils.torch_utils import select_device, smart_inference_mode


@smart_inference_mode()
def run(
weights=ROOT / 'yolov5s.pt',  # model path or triton URL
source=ROOT / 'data/images',  # file/dir/URL/glob/screen/0(webcam)
data=ROOT / 'data/coco128.yaml',  # dataset.yaml path
imgsz=(640, 640),  # inference size (height, width)
conf_thres=0.25,  # confidence threshold
iou_thres=0.45,  # NMS IOU threshold
max_det=1000,  # maximum detections per image
device='',  # cuda device, i.e. 0 or 0,1,2,3 or cpu
view_img=False,  # show results
save_txt=False,  # save results to *.txt
save_csv=False,  # save results in CSV format
save_conf=False,  # save confidences in --save-txt labels
save_crop=False,  # save cropped prediction boxes
nosave=False,  # do not save images/videos
classes=None,  # filter by class: --class 0, or --class 0 2 3
agnostic_nms=False,  # class-agnostic NMS
augment=False,  # augmented inference
visualize=False,  # visualize features
update=False,  # update all models
project=ROOT / 'runs/detect',  # save results to project/name
name='exp',  # save results to project/name
exist_ok=False,  # existing project/name ok, do not increment
line_thickness=3,  # bounding box thickness (pixels)
hide_labels=False,  # hide labels
hide_conf=False,  # hide confidences
half=False,  # use FP16 half-precision inference
dnn=False,  # use OpenCV DNN for ONNX inference
vid_stride=1,  # video frame-rate stride
):
source = str(source)
save_img = not nosave and not source.endswith('.txt')  # save inference images
is_file = Path(source).suffix[1:] in (IMG_FORMATS + VID_FORMATS)
is_url = source.lower().startswith(('rtsp://', 'rtmp://', 'http://', 'https://'))
webcam = source.isnumeric() or source.endswith('.streams') or (is_url and not is_file)
screenshot = source.lower().startswith('screen')
if is_url and is_file:
source = check_file(source)  # download

# Directories
save_dir = increment_path(Path(project) / name, exist_ok=exist_ok)  # increment run
```

```
(save_dir / 'labels' if save_txt else save_dir).mkdir(parents=True, exist_ok=True)  # make dir

# Load model
device = select_device(device)
model = DetectMultiBackend(weights, device=device, dnn=dnn, data=data, fp16=half)
stride, names, pt = model.stride, model.names, model.pt
imgsz = check_img_size(imgsz, s=stride)  # check image size

# Dataloader
bs = 1  # batch_size
if webcam:
view_img = check_imshow(warn=True)
dataset = LoadStreams(source, img_size=imgsz, stride=stride, auto=pt, vid_stride=vid_stride)
bs = len(dataset)
elif screenshot:
dataset = LoadScreenshots(source, img_size=imgsz, stride=stride, auto=pt)
else:
dataset = LoadImages(source, img_size=imgsz, stride=stride, auto=pt, vid_stride=vid_stride)
vid_path, vid_writer = [None] * bs, [None] * bs

# Run inference
model.warmup(imgsz=(1 if pt or model.triton else bs, 3, *imgsz))  # warmup
seen, windows, dt = 0, [], (Profile(), Profile(), Profile())
for path, im, im0s, vid_cap, s in dataset:
with dt[0]:
im = torch.from_numpy(im).to(model.device)
im = im.half() if model.fp16 else im.float()  # uint8 to fp16/32
im /= 255  # 0 - 255 to 0.0 - 1.0
if len(im.shape) == 3:
im = im[None]  # expand for batch dim

# Inference
with dt[1]:
visualize = increment_path(save_dir / Path(path).stem, mkdir=True) if visualize else False
pred = model(im, augment=augment, visualize=visualize)

# NMS
with dt[2]:
pred = non_max_suppression(pred, conf_thres, iou_thres, classes, agnostic_nms, max_det=max_det)

# Second-stage classifier (optional)
# pred = utils.general.apply_classifier(pred, classifier_model, im, im0s)

# Define the path for the CSV file
csv_path = save_dir / 'predictions.csv'

# Create or append to the CSV file
def write_to_csv(image_name, prediction, confidence):
data = {'Image Name': image_name, 'Prediction': prediction, 'Confidence': confidence}
with open(csv_path, mode='a', newline='') as f:
writer = csv.DictWriter(f, fieldnames=data.keys())
if not csv_path.is_file():
writer.writeheader()
writer.writerow(data)

# Process predictions
for i, det in enumerate(pred):  # per image
seen += 1
if webcam:  # batch_size >= 1
```

```
p, im0, frame = path[i], im0s[i].copy(), dataset.count
s += f'{i}: '
else:
p, im0, frame = path, im0s.copy(), getattr(dataset, 'frame', 0)

p = Path(p)  # to Path
save_path = str(save_dir / p.name)  # im.jpg
txt_path = str(save_dir / 'labels' / p.stem) + ('' if dataset.mode == 'image' else f'_{frame}')  # im.txt
s += '%gx%g ' % im.shape[2:]  # print string
gn = torch.tensor(im0.shape)[[1, 0, 1, 0]]  # normalization gain whwh
imc = im0.copy() if save_crop else im0  # for save_crop
annotator = Annotator(im0, line_width=line_thickness, example=str(names))
if len(det):
# Rescale boxes from img_size to im0 size
det[:, :4] = scale_boxes(im.shape[2:], det[:, :4], im0.shape).round()

# Print results
for c in det[:, 5].unique():
n = (det[:, 5] == c).sum()  # detections per class
s += f"{n} {names[int(c)]}{'s' * (n > 1)}, "  # add to string

# Write results
for *xyxy, conf, cls in reversed(det):
c = int(cls)  # integer class
label = names[c] if hide_conf else f'{names[c]}'
confidence = float(conf)
confidence_str = f'{confidence:.2f}'

if save_csv:
write_to_csv(p.name, label, confidence_str)

if save_txt:  # Write to file
xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn).view(-1).tolist()  # normalized xywh
line = (cls, *xywh, conf) if save_conf else (cls, *xywh)  # label format
with open(f'{txt_path}.txt', 'a') as f:
f.write(('%g ' * len(line)).rstrip() % line + '\n')

if save_img or save_crop or view_img:  # Add bbox to image
c = int(cls)  # integer class
label = None if hide_labels else (names[c] if hide_conf else f'{names[c]} {conf:.2f}')
annotator.box_label(xyxy, label, color=colors(c, True))
if save_crop:
save_one_box(xyxy, imc, file=save_dir / 'crops' / names[c] / f'{p.stem}.jpg', BGR=True)

# Stream results
im0 = annotator.result()
if view_img:
if platform.system() == 'Linux' and p not in windows:
windows.append(p)
cv2.namedWindow(str(p), cv2.WINDOW_NORMAL | cv2.WINDOW_KEEPRATIO)  # allow window resize (Linux)
cv2.resizeWindow(str(p), im0.shape[1], im0.shape[0])
cv2.imshow(str(p), im0)
cv2.waitKey(1)  # 1 millisecond

# Save results (image with detections)
if save_img:
if dataset.mode == 'image':
cv2.imwrite(save_path, im0)
else:  # 'video' or 'stream'
```

```
if vid_path[i] != save_path:  # new video
vid_path[i] = save_path
if isinstance(vid_writer[i], cv2.VideoWriter):
vid_writer[i].release()  # release previous video writer
if vid_cap:  # video
fps = vid_cap.get(cv2.CAP_PROP_FPS)
w = int(vid_cap.get(cv2.CAP_PROP_FRAME_WIDTH))
h = int(vid_cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
else:  # stream
fps, w, h = 30, im0.shape[1], im0.shape[0]
save_path = str(Path(save_path).with_suffix('.mp4'))  # force *.mp4 suffix on results videos
vid_writer[i] = cv2.VideoWriter(save_path, cv2.VideoWriter_fourcc(*'mp4v'), fps, (w, h))
vid_writer[i].write(im0)

# Print time (inference-only)
LOGGER.info(f"{s}{'' if len(det) else '(no detections), '}{dt[1].dt * 1E3:.1f}ms")

# Print results
t = tuple(x.t / seen * 1E3 for x in dt)  # speeds per image
LOGGER.info(f'Speed: %.1fms pre-process, %.1fms inference, %.1fms NMS per image at shape {(1, 3, *imgsz)}'
    % t)
if save_txt or save_img:
s = f"\n{len(list(save_dir.glob('labels/*.txt')))} labels saved to {save_dir / 'labels'}" if save_txt else
    ''
LOGGER.info(f"Results saved to {colorstr('bold', save_dir)}{s}")
if update:
strip_optimizer(weights[0])  # update model (to fix SourceChangeWarning)


def parse_opt():
parser = argparse.ArgumentParser()
parser.add_argument('--weights', nargs='+', type=str, default=ROOT / 'yolov5s.pt', help='model path or
    triton URL')
parser.add_argument('--source', type=str, default=ROOT / 'data/images', help='file/dir/URL/glob/screen/0(
    webcam)')
parser.add_argument('--data', type=str, default=ROOT / 'data/coco128.yaml', help='(optional) dataset.yaml
    path')
parser.add_argument('--imgsz', '--img', '--img-size', nargs='+', type=int, default=[640], help='inference
    size h,w')
parser.add_argument('--conf-thres', type=float, default=0.25, help='confidence threshold')
parser.add_argument('--iou-thres', type=float, default=0.45, help='NMS IoU threshold')
parser.add_argument('--max-det', type=int, default=1000, help='maximum detections per image')
parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
parser.add_argument('--view-img', action='store_true', help='show results')
parser.add_argument('--save-txt', action='store_true', help='save results to *.txt')
parser.add_argument('--save-csv', action='store_true', help='save results in CSV format')
parser.add_argument('--save-conf', action='store_true', help='save confidences in --save-txt labels')
parser.add_argument('--save-crop', action='store_true', help='save cropped prediction boxes')
parser.add_argument('--nosave', action='store_true', help='do not save images/videos')
parser.add_argument('--classes', nargs='+', type=int, help='filter by class: --classes 0, or --classes 0 2
    3')
parser.add_argument('--agnostic-nms', action='store_true', help='class-agnostic NMS')
parser.add_argument('--augment', action='store_true', help='augmented inference')
parser.add_argument('--visualize', action='store_true', help='visualize features')
parser.add_argument('--update', action='store_true', help='update all models')
parser.add_argument('--project', default=ROOT / 'runs/detect', help='save results to project/name')
parser.add_argument('--name', default='exp', help='save results to project/name')
parser.add_argument('--exist-ok', action='store_true', help='existing project/name ok, do not increment')
parser.add_argument('--line-thickness', default=3, type=int, help='bounding box thickness (pixels)')
```

```
parser.add_argument('--hide-labels', default=False, action='store_true', help='hide labels')
parser.add_argument('--hide-conf', default=False, action='store_true', help='hide confidences')
parser.add_argument('--half', action='store_true', help='use FP16 half-precision inference')
parser.add_argument('--dnn', action='store_true', help='use OpenCV DNN for ONNX inference')
parser.add_argument('--vid-stride', type=int, default=1, help='video frame-rate stride')
opt = parser.parse_args()
opt.imgsz *= 2 if len(opt.imgsz) == 1 else 1  # expand
print_args(vars(opt))
return opt


def main(opt):
check_requirements(ROOT / 'requirements.txt', exclude=('tensorboard', 'thop'))
run(**vars(opt))


if __name__ == '__main__':
opt = parse_opt()
main(opt)
```