Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Távközlési és Médiainformatikai Tanszék

Mészáros Jakab

Önvezetésben alkalmazott felhő alapú képfeldolgozás hatékonysága

Tudományos Diákköri Konferencia

Konzulens

Dr. Fehér Gábor

BUDAPEST, 2023

Tartalom

Kivonat 3
Abstract 4
1. Bevezetés 5
2. Feldolgozott szakirodalom 6
2.1. Az objektumfelismerés fejlődése 6
2.2. YOLO 6
2.3. Adatfeldolgozás a felhőben 8
3. Elméleti háttér 8
3.1. Objektum felismerés 8
4. A probléma vizsgálata 11
5. A megoldás menete 11
5.1. Adatbázis készítés 11
5.2. Adatbázis tanítása 12
5.3. Tesztelő környezet 15
5.4. Jármű-felhő kommunikáció 16
5.5. Tesztelés 17
6. Eredmények 19
6.1. Átvitel 19
6.2. Modellek hatékonysága 21
6.2.1. Nano modell 21
6.2.2. Small modell 22
6.2.3. Medium modell 23
6.2.4. Large modell 24
6.3. Összesített elemzés 24
7. Összegzés 27
Köszönetnyílvánítás: 28
Irodalomjegyzék 29

Kivonat

Manapság egyre növekvő jelentőséggel bírnak az önvezető járművek a közlekedésben, és ebben kiemelt szerepet játszik a képfelismerés technológiája. Egy lehetséges módszer a megvalósításra, hogy a járművön elhelyezünk szenzorokat, például radarokat és kamerákat, amelyek felelősek az adatok begyűjtéséért. A képfeldolgozást egy szoftver végzi, amely a mesterséges intelligencia és a neurális hálózatok segítségével határozza meg a környezeti viszonyokat és közlekedési helyzeteket. A folyamat lezajlása után az eredmény figyelembevételével avatkozik be a vezető. A képfeldolgozó szoftver futhat lokálisan és a felhőben is, TDK dolgozatomban utóbbit vizsgáltam.

Kutatásomban azzal foglalkoztam, hogy az adatok feldolgozása a felhőben, különböző paraméterek mellett milyen hatékonysággal kivitelezhető. Lokálisan is végezhetnénk az adatelemzést, ha olyan dolgokat vizsgálnánk az utakon, amelyek ritkán változnak (ilyenek lehetnek például a közlekedési táblák, jelzőlámpák, útfestések), így kisebb méretű neurális hálózat is hatékonyan tudna működni. A közlekedésben azonban sokkal jellemzőbb, hogy folyamatosan változó objektumokat kell felismerni (például városkép, közlekedésben részt vevő járművek, ugyan itt közlekedési táblák különböző időjárási helyzetekben), amihez már egy dinamikusan növekvő neurális hálózatra van szükségünk - ez lokálisan futtatva viszont már nem praktikus. A folyamatosan bővülő háló több szempontból előnyös. A felismerés hatékonysága nem konstans, hanem javuló tendenciát mutatott, ennek következtében kiterjeszthető a feladatkör több közlekedési helyzet elemzésére, amely növelheti az utasok biztonságát.

A felhő alapú adatfeldolgozás során vizsgáltam, hogy mekkora adatátvitel szükséges a videó- és képanyag továbbításához, különböző videó sávszélességek és frissítési ráta értékek mellett elemeztem a működést, mértem a késleltetést, és kerestem az optimális paraméter értékeket, amelyek mellett még hatékonyan értelmezni lehet az eredményt.

Abstract

Today, self-driving vehicles are becoming increasingly important in transport, and image recognition technology plays a key role in this. One possible way of implementing this is to place sensors on the vehicle, such as radars and cameras, which are responsible for collecting data. Image processing is performed by software that uses artificial intelligence and neural networks to determine environmental conditions and traffic situations. Once the process has been completed, the driver intervenes by taking the result into account. The image processing software can run locally or in the cloud, in my TDK thesis I studied the latter.

In my research, I looked at the efficiency of processing data in the cloud under different parameters. We could also do data analysis locally by looking at things on the roads that rarely change (such as traffic signs, traffic lights, road paintings), so a small-scale neural network could be efficient. In transport, however, it is more typical to have to detect constantly changing objects (e.g. cityscape, vehicles in traffic, traffic signs in different weather conditions), which requires a dynamically growing neural network - which is impractical when run locally. A continuously expanding network has several advantages. The efficiency of the detection is not constant but improving, and as a result the task can be extended to analyse more traffic situations, which can increase passenger safety.

In cloud computing, I investigated the data throughput required to transmit video and image data, analyzed the operation at different video bandwidths and FPS, measured the latency, and searched for optimal parameter values at which the results can still be interpreted efficiently.

1. Bevezetés

Manapság a képfeldolgozás megkerülhetetlen része az életünknek. Az automatizálási folyamatok jelentős részére megoldást kínál, hiszen lehetővé teszi a számítógépek számára, hogy vizuális információt elemezzenek. Nincs ez máshogy a közlekedésben sem. Az önvezető és vezetést segítő rendszerekkel rendelkező járművek különféle szenzorokkal, köztük kamerákkal is fel vannak szerelve, amelyek képét elemezni szükséges. A módszer adott, a megvalósításban azonban – amellett, hogy egy rohamos ütemben fejlődő területről beszélünk – még mindig sok a megválaszolásra váró kérdés.

TDK dolgozatomban arra kívánok választ találni, hogy milyen feltételei vannak annak, ha a képen lévő információ elemzésére a felhőben kerül sor, milyen kihívásokkal, milyen előnyökkel jár együtt ez a fajta megvalósítás. Egy elterjedt kézenfekvő megvalósítás, hogy a képfeldolgozás lokálisan a járműben történik [1]. Ennek is megvannak a maga előnyei, mint például az azonnali reakció. Nincs szükség hálózatra, nincs szükség kommunikációra, minden a jármű szoftveres rendszerén belül történik. Ilyenkor azonban limitálva van a feldolgozó neurális hálózat, és a súlyok mérete, valamint a lokális modell frissítési gyakoriságával is foglalkozni kell. Egy ilyen rendszer hasznos lehet, hogyha olyan objektumokat szeretnénk észlelni az úton, amik nem, vagy csak nagyon ritkán változnak, ilyenek például a közlekedési táblák, lámpák és sávok.

Abban az esetben, ha a vizsgált objektum gyakran változik, mint például a különböző járművek, vagy a környezet, akkor hasznosabb lehet egy dinamikusan növekvő konvolúciós neurális hálózat. Ez azonban már olyan számítási igényekkel és idővel rendelkezik, amellyel nem érdemes az autó lokális szoftverét terhelni. Ilyenkor jelenthet előnyt a felhő alapú adatfeldolgozás [2].

Munkám során ilyen objektumokon keresztül próbáltam azokat a paramétereket keresni, és optimálisan megválasztani, amelyek mellett hatékony működést lehet elérni a felhő alapú képfeldolgozás során. A megvalósított elemek, amelyek a dolgozatomban később bemutatásra kerülnek a következők:

- A YOLOv8 [3] objektum detektáló modell több típusának betanítása saját adathalmazzal.
- Videó küldésére és fogadására alkalmas szerver-kliens páros létrehozása, amely reprezentálja a jármű és a felhő közötti kapcsolatot.

- A különböző felismerő modellek paramétereinek mérése, valamint az átvitel tulajdonságainak vizsgálata, végül az eredmények értelmezése.

2. Feldolgozott szakirodalom

2.1. Az objektumfelismerés fejlődése

Az objektumfelismerés a huszadik század végén kezdte el érdekelni az embereket, az első nagyobb áttörésig azonban egészen 2001-ig kellett várni [4][5]. Ekkor alkották meg a Viola Jones detektorokat. Ezek a korhoz képest nagyságrendekkel gyorsabb működést értek el, ezért voltak kiemelkedőek a többi modell közül. Főként arcfelismerésre használták. Működését tekintve egy csúszó ablakot használtak, amelyet a vizsgált képen végig mozgatva (több méretű ablakokat is), próbálta meg felismerni az objektumot. Ha az arc pont beleesett a keretbe akkor egyezést talált, és megtörtént a felismerés. Ez a módszer, még úgy is, hogy korabeli társaihoz képest gyorsnak számított, nagyon időigényes volt.

A következő ugrás 2005-ben volt, ekkor megalkották a HOG (Histogram of Oriented Gradients) detektort. Ennél a modellnél a képfelismerés a következőképpen zajlott. A képet azonos méretű cellákra bontották, ezzel létrehozva egy rácsot. Az egyes cellákban gradienst számolnak, amellyel kiszűrték a kontrasztokat. Ezeket blokkokba rendezték, majd normalizálták. A blokkokat összefűzték, amely már egy cellákat átfedő alakzatot alkotott. A modellt elsősorban gyalogosok felismerésére fejlesztették ki.

A HOG detektor egy továbbfejlesztett verziója jelent meg 2008-ban, ez volt a DPM (Deformable Part-Based Model). Ez annyiban változtatott az előző verzióhoz képest, hogy a teljes objektumot részekből tette össze. Egy járművet az alapján azonosított, hogy felismerte a kerekeit, ablakait, lámpáit, és egyéb részletét.

2.2. YOLO

Az igazán nagy áttörést a konvolúciós neurális hálózatokon alapuló modellek hozták. A gépi tanulás segítségével a kép minden egyes paramétere betanítható, ez rengeteg lehetőséget és egy új világot nyitott meg az objektum detektálás területében. 2015-ben jelent meg a YOLO [6][7]][8] modell, ami abban volt különleges társaihoz képest, hogy ez volt az első egylépcsős detektor, ami azt jelenti, hogy egyetlen neurális hálózattal elemezték a teljes képet. Az eredeti verzió 24 konvolúciós rétegből, és 2 teljesen összekapcsolt rétegből állt, utóbbiak feleltek az objektum osztályának, méretének és pozíciójának észleléséért. A YOLO régiókra osztja a képet, és megjósolja annak határait. Ezáltal nem a teljes képet vizsgálja, csak azokat a részeket, ahol

nagy eséllyel található az objektum [1][7]. Miután megtörtént a lokalizálás és a klasszifikáció, az objektumok köré úgynevezett keretdobozok kerülnek. Ezek a keretdobozok 6 paraméterrel jellemezhetőek. Az első négy a doboz lokációjára vonatkozik a képen belül, ezt általában egy x, y, h, w paraméter-együttessel írható le [7]. Ezek rendre az x és y koordináták, valamint a keret magassága és szélessége. Utóbbi két paraméter helyett a YOLO modell biztosítja, hogy használhatunk szintén x és y koordinátákat, amelyek egy másik pontot határoznak meg. Így két pont fogja megadni a keretet. Az 5. paraméter a felismert osztály konfidenciájára vonatkozik, ez százalékos értékben adják meg, a 6. paraméter meg maga a felismert osztály.

Model	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

1. ábra A YOLOv8 modell specifikációja előre betanított adathalmazzal tesztelve [3]

Dolgozatomban a legfrissebb YOLOv8 [3] modellt használtam. Az 1. ábrán ennek a specifikációját láthatjuk. Az modell egy beépített COCO nevezetű 80 elemű adathalmazt használ [9]. Ez az adatbázis olyan hétköznapi tárgyakat tartalmaz, amelyekkel a leggyakrabban találkozunk. A fenti táblázatban bal oldalt helyezkednek el a különböző méretű modellek egymás alatt. A „size” paraméter a modell bementi nagyságára utal. A következő a hatékonyságot jellemzi, majd a sebességet. Itt érdemes megjegyezni, hogy mivel egy nagy teljesítményű számítási folyamatról van szó, egyáltalán nem mindegy, hogy min futtatjuk a felismerést. Egy jobb minőségű grafikus feldolgozó egységgel átlagosan két nagyságrenddel jobb eredményt tudunk elérni, mint egy processzorral, ami a feldolgozási sebességet illeti. A maradék két jellemző azt mutatja, hogy mekkora paraméterszámmal dolgozott a modell, és mennyi számítást végzett a tanítás során.

2.3. Adatfeldolgozás a felhőben

Jangwon Lee és munkatársai azt kutatták, hogy miért van relevanciája a felhő alapú kép és adatfeldolgozásnak, ennek milyen előnyei és hátulütői vannak, mi lenne az optimális megoldás [2]. A kutatócsoport nem földi járművekkel, hanem drónokkal végezte a tesztelést, mivel ott ez a probléma hatványozottan értendő a magasabb sebesség és a korlátozott súly miatt [10]. Ezek a jellemzők egyébként az autókban is megvannak, ha nem is a feltétlenül a súlykorlát, de a sebesség mindenképpen. Egy másik közös pont, hogy a környezet folyamatosan változik, és mozgó, változó méretű, egyszerre több, akár mozgó objektumot kell felismerniük. Itt szintén megemlítik, hogy a tanítás és az elemzés egy magasabb kategóriájú grafikus feldolgozó egységet igényel.

A felhőben szinte korlátlan mennyiségű adatforrás van, így a képfelismerés hatékonysága nagymértékben növekedhet, egyszerre akár több száz objektumot is észlelni lehetne [11][12]. Egy másik előny, hogy az autó plusz képi és térképes forráshoz is hozzáférhet. Egy ilyen konstrukció lehetővé teszi egy olyan rendszer kialakulását és fejlesztését, amelyben az autonóm- és részben autonóm járművek egy azon adatbázist tápláljanak. Ez több szempontból előnyös. Egyrészt a felismerés hatékonysága arányosan növekedne a közlekedésben résztvevők számával, másrészt a gyártónak nem kéne foglalkoznia azzal a problémával, hogy folyamatosan minden forgalomban résztvevő járművén frissítse a tanító hálókat.

A módszer hátránya ugyanakkor az, hogy a kommunikáció, és a küldendő adat mennyisége az autó és a felhő között kiszámíthatatlan mértékű késést okoz, amely sem a földi, sem a légi közlekedésben nem megengedhető.

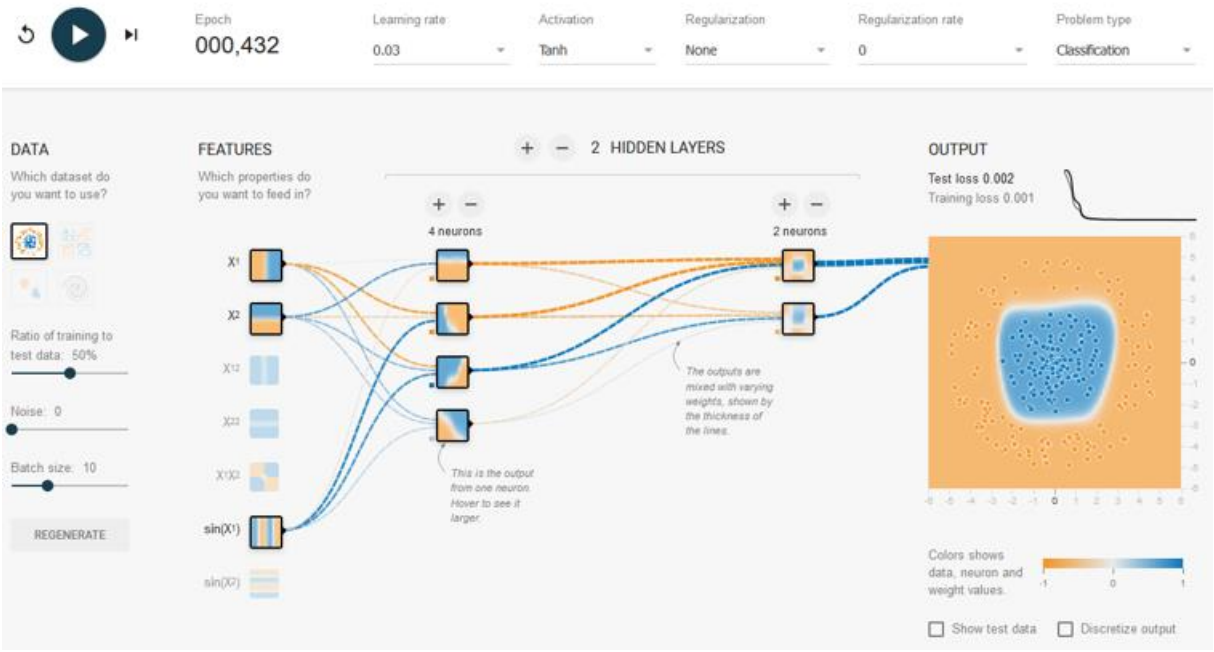
A cikkben feltüntetett javaslat egy olyan rendszerre utal, amely egy kisebb méretű hálózatot futtat lokálisan, amely gyorsan csak azt nézi, hogy az adott képen hol található érdemi információ, és csak ezt az információt továbbítja a felhőbe [13][14].

3. Elméleti háttér

3.1. Objektum felismerés

Az objektumok, jelen esetben közlekedésben résztvevő tárgyak felismerése többféle módon is történhet [15]. Az egyik lehetséges mód, hogy geometriai alakzat felismerőt használunk, ez azonban nem hatékony. Egy másik, hogy szín szerint szűrünk, ezzel meghatározzuk, hogy hol helyezkedik el a képen a lényegi információ, ezt kivágva, a többi rész

számunkra nem érdekes, majd ezen egy szám/szöveg felismerést futtatunk. Nem ezek a modellek a legelterjedtebbek. A leggyakrabban használt, legismertebb módszer az objektumfelismerésre a konvolúciós neurális hálózatok alkalmazása (CNN) [16][17]. Ezt a modellt az emberi működés alapján fejlesztették ki, innen is jön a neuron elnevezés, hasonló feladatot lát el a hálózatban is, mint egy élő organizmusban. Ez a hálózat képes rengeteg bemeneti adat alapján megtanulni, hogy mi található a képen. Az egyes neuronoknak vannak bemenetei, amelyeket súlyozunk. A súlyozás azt jelenti, hogy az egyes bemenetek milyen és mekkora hatással legyenek a neuronra. A bemeneteket összegezzük, majd egy aktivációs függvény eldönti, hogy mi jelenjen meg a kimeneten. Néhány fontosabb aktivációs függvény: *sigmoid, relu, leaky relu, tanh, maxout*, [7]. A neurális hálózat úgy néz ki, hogy a neuronok kimenetét egy másik neuron bemenetére kapcsoljuk.



2. ábra Példa egy konvolúciós neurális hálózatra [18]

Az *2. ábrán* látható egy ilyen példahálózat, amelyet a *tensorflow* [18] oldal használatával hoztam létre. Ez előre csatolt struktúrában épült, ami azt jelenti, hogy az egyes rétegek kimenetei csak egy következő réteg bemeneteire csatlakoznak. Léteznek olyan hálózatok is, amelyeknél a neuron kimenetei azonos rétegben található neuron bemenetekre csatlakoznak, de akár visszafelé is küldhetnek információkat. Ezek a rekurrens (visszacsatolt) hálók.

A *2. ábra* alapján bemutatom egy ilyen hálózat működését. Ezeknek a hálózatoknak az eszköze a tanítás. Ezt a súlyozással oldjuk meg, ami a következőképpen működik. A bemenetre

adunk egy képet, ez úgynevezett konvolúciós blokkonként végigfut a neurális hálózaton, majd megpróbálja megjósolni, hogy mi látható a képen. Amennyiben nem megfelelő választ adott, úgy visszafele elkezdem állítani a súlyokat annak megfelelően, hogy mi volt a valódi kimenet. Ezt nevezzük hiba visszaterjesztéses tanulásnak. Az ábrán a kék és a sárga pöttyök alkotják a bemenetet, erre kell, hogy rátanuljon a rendszer. A Data résznél kiválaszthatjuk, hogy milyen bonyolultságú mintára akarjuk rátanítani. Az előzőekben már említett *tanh* aktivációs függvényt használható a kimenetek megválasztására. Ez a hálózat hat neuront tartalmaz, amit két rejtett rétegbe osztottam szét. A bemeneti tanító függvények az *X1, X2* és *sin(X1).* A feladat bonyolultságától függően állíthatjuk be a neuronok, valamint a rejtett rétegek számát, a tanító függvények típusát, valamint az aktivációs függvényt. A vastagabb vonalak a „hasznosabbak", ott a leghatékonyabb a tanítás. Ha egy vékonyabb halványabb vonalon lévő neuront kiveszünk a rendszerből, akkor szinte semmilyen szinten nem lesz gyengébb a hálózatunk. A rendszerünk akkor optimális, ha minél kisebb a hálózat, de már el tudja látni azt a nehézségű feladatot, amit adtunk neki.

A tanítás folyamatát két részre tudjuk bontani. Az egyik maga a betanítás, a másik a tesztelés. Mindkét esetnél nézzük a folyamat hatékonyságát, ez jellemezhető az ábrán is látható *test loss* és *training loss* változókkal. A karakterisztikát az *epoch* függvényében ábrázoltuk, ami azt jelenti, hogy hányszor futott végig az összes bemeneti adat a hálózaton. Az eredmény nem lineárisan, hanem csökkenő, laposodó ütemben javul az *epoch* számának növekedésével, ez függ a feladat bonyolultságától, valamint attól, hogy milyen „bátran" állítjuk a súlyokat. A *batch* értékkel azt tudjuk állítani, hogy ha beküldök egy adatot, akkor utána állítsa a súlyokat, vagy csak több adat bevitele után tegye ezt meg. Minél nagyobb a *batch* értéke, annál gyorsabban tanul a rendszer.

Egy lehetséges veszélyforrás, amire figyelni kell, az a túlilleszkedés. Ez azt jelenti, hogy túl sokszor küldöm be az adatokat a hálózatnak,vagyis túl nagy az *epoch* szám, és a rendszerem nagyon jól rátanul a tanító mintákra. Ez csak azért probléma, mert így azokat a bemeneteket, amikkel tanítottunk, azokat akár hibátlanul fogja tudni, de ez egyáltalán nem jelenti azt, hogy majd a tesztelésnél egy random bementre jól meg fogja jósolni, hogy mit tartalmaz az adott kép. Ezt különböző módszerekkel ki tudjuk küszöbölni.

Fontos, hogy a bementet, hogyan adjuk be hálózatnak, ettől is függ a hatékonyság. Ahelyett, hogy a pixeleket adnánk be adatként, sokkal célravezetőbb, ha a pixelek környezetét vesszük konvolúcióval. Ez tulajdonképpen egy mátrixot illeszt minden pixelre, amely mátrix elemei meghatározzák, hogy mennyivel kell szorozni a szomszédos pixel értékeket. A kép szélén figyelni kell, hogy a mátrix, ha kilóg, akkor a széleket majd utólag vágjuk le (nullával

szorozzuk be), vagy ne engedélyezzük a mátrixnak, hogy a kép szélénél tovább menjen. Ezzel a művelettel éleket keresünk. A metódus alapja, hogy a szomszédos pixelek hasonlóak, ezeket egyszerűsítjük kevesebb pixellé. Egy hálózat többfajta műveletsorozatok összessége, ilyenek a már említett konvolúció, *max pooling*, aminek megadható, hogy mennyi elem közül válasszon egyet, így csökkenteni tudja a dimenziószámot, létezik még *dropout*, amelynek segítségével gátolni lehet a túltanulást. Ezeket a műveleteket akár többször is elvégezhetjük egymás után, a kimenetnek egyeznie kell a neurális hálózat bemeneti elemszámával. A neurális hálózat kimenete annyi neuront kell, hogy tartalmazzon, ahány fajta lehetséges kimenet van, például, ha számjegyeket akarunk felismerni, akkor 10 féle neuron lesz a kimeneten, mindegyikhez tartozik egy számjegy 0-ától 9-ig.

4. A probléma vizsgálata

A probléma forrása az a korlátolt kapacitás, amely a járműre jellemző, mind szoftveresen, mind hardveresen. Az objektum detektálás egy nagy számításokat igénylő folyamat, amely hosszabb időt vesz igénybe. A komplexitás fokozódik, ha dinamikusan változó objektumokat szeretnénk meghatározni. Amennyiben például közlekedési táblákat ismerünk fel, nincs szükség olyan nagyméretű hálóra, mert a táblák mindig ugyanúgy néznek ki, és ami még fontosabb, hogy sok van belőlük. Lehetséges, hogy egy-egy tábla elé belóg egy növény, vagy el van ferdülve, forgatva, esetleg sérült, de az esetek legnagyobb részében a modell alkalmazható, és az ilyen eseteket is könnyen felismeri. A probléma ott kezdődik, amikor gyakran változó objektumokat akarunk felismerni, amelyekből ráadásul kevés számú is van, ilyen lehet például egy konkrét épület, vagy természeti látványosság. Felmerülhet a kérdés, hogy mi szükség van ezeknek a felismerésére. Elsősorban az, hogy fejlettebb navigációs rendszereket tudjunk létrehozni.

5. A megoldás menete

5.1. Adatbázis készítés

Mint már említettem, a kutatásomhoz a YOLOv8 [3] modellt használtam, ez azonban csak azt a 80 objektumot képes detektálni, amelyeket a COCO [9] adathalmaz tartalmaz. Ahhoz, hogy a modell olyan dolgot ismerjen fel, amit csak szeretnénk, az első lépés az adatok gyűjtése. Erre két lehetőség van, léteznek adatbázisok, amelyeket le lehet tölteni, vagy saját magunknak kell azt elkészíteni, például fotók alapján. Vizsgálataim során a teszteléshez két, eltérő

karakterisztikájú épületet használtam, az egyik egy lakóház, a másik egy templom, és ezeken keresztül vizsgáltam a fent említett problémát (*3. ábra*).

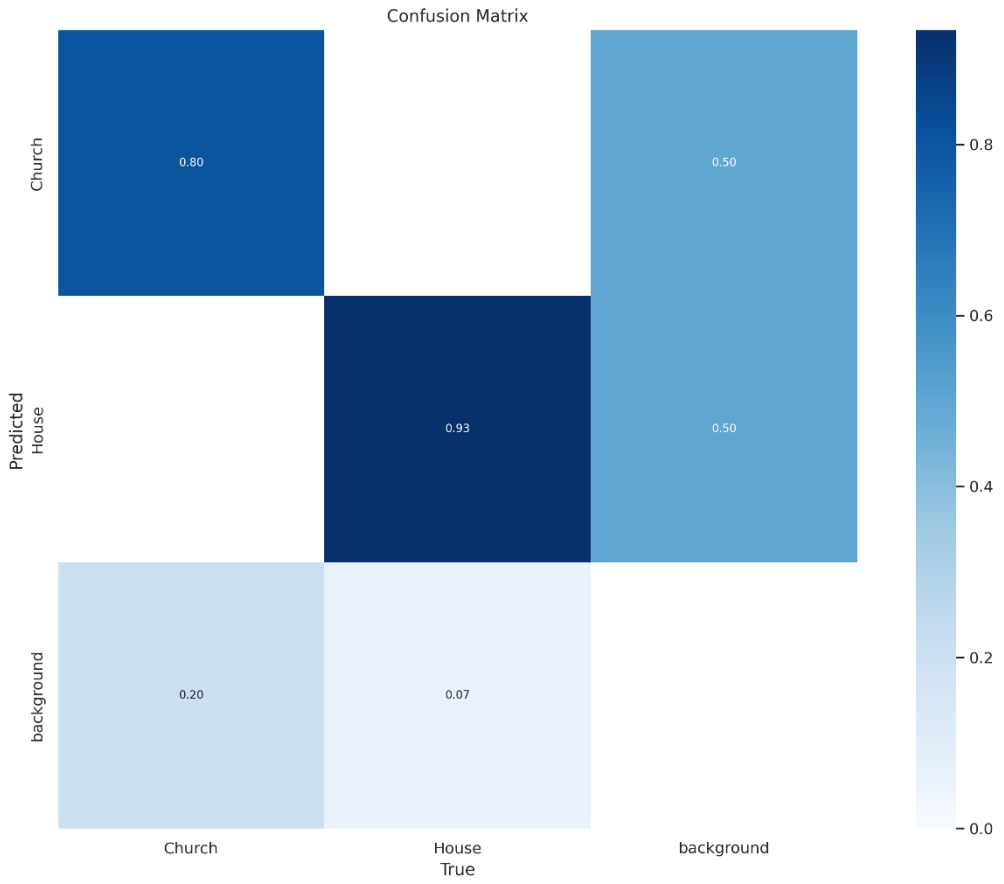
3.ábra Felismerendő objektumok – lakóház (balra) és templom (jobbra)

A képeken már látszik, hogy gyakorlati szempontból miben tér el ez a fajta felismerés, mint egy táblafelismerés: mindkét épület takarásban van, nincs olyan pozíció, ahonnan az egészet látni lehetne, a hiányzó részeket a többi készített képből kell összerakni.

A képeket olyan formátumba kell hozni, hogy azokat a YOLOv8 modellekkel tanítani lehessen. Ehhez a „*Roboflow*” [19] nevezetű online szoftvert használtam. Az alkalmas formátumhoz a képeket annotálni kell, amely során egy szöveges fájlban megadjuk mindazt, ami a tanításhoz szükséges, így a felismerendő objektum koordinátáit a már ismertetett módon, valamint az osztály számát és nevét. Jelen esetben 2 osztályt tudunk definiálni: „House” és „Church”. Az adathalmazom 239 elemes lett, ami egy kisebb mennyiségű adatnak felel meg tanítási szempontból. A képek mellett létrehoztam tesztelésre szánt videókat is, amelyeket úgy készítettem, hogy gépjárművel különböző irányokból elhaladtam a felismerendő objektumok mellett.

5.2. Adatbázis tanítása

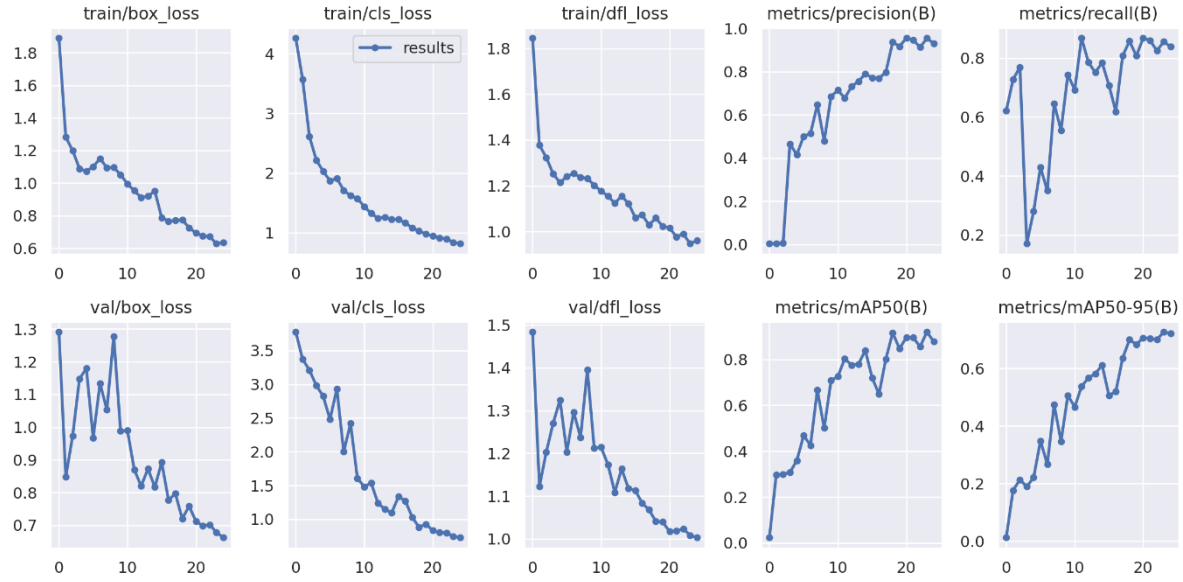
A dolgozatomban 4 típusú modellt tanítottam be *nano*, *small*, *medium*, *large*. Mindegyik 25 *epoch* hosszan tanult, és az eredmény a következő lett:



4. ábra Konfúziós mátrix

A *4. ábrán* látható eredmény, a *nano* modell tanítása után keletkezett. Egy ilyen konfúziós mátrixot a következőképpen kell értelmezni. Egy változós esetben 4 blokkra osztható a mátrix. Az első az igaz pozitív, ez azt mutatja meg, hogy az esetek hány százalékában azonosította a modell az objektumot helyesen. A 4. ábrán két ilyen cella is van, ott, ahol az azok a sorok és oszlopok metszik egymást, amiknek azonos a neve. Ezek alapján leolvashatjuk, hogy a legkisebb modellel való tanítás után a templomot 80, a házat 93 százalékban ismerte fel helyesen, az adatbázisban található képek alapján (teszteléshez is ezt használta). A következő cella az igaz negatív, ezek azok az események, amikor a képen nem volt az adott objektumból, és helyesen, nem is ismerte fel. A következő eset a hamis pozitív, azaz, ha egy objektum nincs ott, mégis azonosítja valamelyik osztályként. Végül az utolsó eset a hamis negatív, ahol az objektum ott van, mégsem ismeri fel a program. Ezt a mátrixban a *background* sor és oszlop, és az osztályok metszetei jelölik.

A tanítás másik fontos eredménye maga, hogy hányadik *epoch* esetén milyen volt a hatékonyság. Ezt az *5. ábra* szemlélteti.



5. ábra Tanítási statisztikák

Amit az 5. ábráról fontos kiemelni, és jól reprezentálja a tanulás milyenségét, az az első két paraméter, a *train/box_loss,* valamint a *train/cls_loss.* Az ábrán látható többi paraméter is a tanítás folyamatáról ad információt, de ezeket jelen kutatásomban nem vizsgáltam. A *train/box_loss* a keretdoboz jellemzésére szolgál, tehát a lokációra, a *train/cls_loss* pedig az osztály predikciójának pontosságára. Az látható mindkét grafikonon, hogy egy laposodó, lassuló tendenciájú hatékonyság növekedést (veszteség csökkenést) fedezhetünk fel. Az itt látható adatok nagyban függnek az adathalmaz méretétől, valamint az osztályok számától. Itt levonható az a konzekvencia, hogy nagyjából 20 *epoch* elegendő lett volna, mert utána jelentős mértékben nem javult a kimenet. Ilyenkor azért érdemes csökkenteni a tanítási ciklusok számát, hogy időt, és számítási kapacitást nyerjünk.

A *small* modell tanítása hasonló eredményekkel zárult, itt az igaz pozitív értékek száma a lakóháznál 95, a templomnál 80 volt. Az már 2 tanítás után is feltűnő, hogy a módszer a házat sokkal jobb hatékonysággal felismeri, mint a templomot, ez annak köszönhető, hogy a lakóházat „tisztább” körülmények között sikerült lefotózni.

Medium és *large* modellek esetében csak minimális változás volt az előző esethez képest. Ez a kis változás az adathalmaz kicsiny jellegének tudható be.

5.3. Tesztelő környezet

A modellek teszteléséhez *python* programozási nyelven hoztam létre egy arra alkalmas környezetet, ahogyan az a 6. ábrán is látszik.

```python
from ultralytics import YOLO
import cv2

model = YOLO("modell_neve")
cap = cv2.VideoCapture("Video")

while True:
    ret, frame = cap.read()
    results = model(frame, show=True, conf=0.75)


    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

6. ábra A modell tesztelésére alkalmas python kód

A teszteléshez szükséges a YOLO modell, valamint a *cv2*, ami egy nyílt forráskódú képfeldolgozó könyvtár, ez utóbbit csak a videó megjelenítésére használtam. A konfidencia szintet 75 százalékra állítottam, ezáltal csak azok az objektumok jelennek meg, amelyeket legalább ilyen pontossággal fel tud ismerni. Ahogyan azt az *1. ábrán* már bemutattam, nagyságrendbeli különbség van a processzor és a grafikus feldolgozó egység sebességében.

7. ábra Eredmény – a vizsgált lakóház felismerése a modell által. A számérték a konfidenciát jelenti.

A teszteléshez egy 681 képkocka (frame) hosszú videót használtam, ez nagyjából 22 másodperc. Ha processzorról futtattam, akkor ez a program több mint 9 perc alatt fordult le. Dolgozatomban később kitérek arra, hogy hogyan lehet GPU alapú tesztelést végezni.

8. ábra Eredmény – a vizsgált templom felismerése a módszer által. A számértékek a konfidenciákat jelentik.

A *7. és 8. ábrán* látható a 2 osztály sikeres felismerése. Mindkét képen megjelenik a körül határoló doboz, valamint a konfidencia szint.

5.4. Jármű-felhő kommunikáció

A kommunikációra a jármű és felhő között szintén *python* programozási nyelven létrehoztam egy szerver-kliens párost (*9. ábra*).

```python
def receive_video():
    frame_count = 0
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.connect((HOST, PORT))
        print("Connected to client.")

        while True:
            data = b""
            while b"\xff\xd9" not in data:
                packet = s.recv(4096)
                if not packet:
                    break
                data += packet

            if len(data) == 0:
                break
            frame_count += 1
        end_time = time.time()
        elapsed_time = end_time - start_time
```

```python
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    print("Waiting for connection...")
    s.bind((HOST, PORT))
    s.listen()
    conn, addr = s.accept()
    with conn:
        print(f"Connected by {addr}")
        start_time = time.time()
        start_time_byte = time.time()

        while True:
            ret, frame = cap.read()
            if not ret:
                break
            resize = cv2.resize(frame, (width, height))
            frame_count += 1
            data = cv2.imencode('.jpg', resize)[1].tobytes()
```

9. ábra Szerver (balra) és Kliens(jobbra) fontosabb részletei

A kapcsolatot Wifi hálózaton keresztül hozom létre, ehhez szükséges a hálózat ip-címe, valamint egy port, amin a kapcsolatot meg tudjuk teremteni. A kapcsolatot a *9. ábrán* jelen esetben a kliens hozza létre, ezt kell először elindítani. Ehhez kapcsolódik a szerver. A videót képekre bontja a program, majd ezeket bájtként továbbítja a hálózaton. A fogadó oldalon 4096-os bájtcsomagonként fogadja az adatot, a gyorsabb beolvasás érdekében. Ha ennél több adat jön egyszerre akkor a *recv()* függvény többször is lefut. Ebből visszaállítható az eredeti videó, amin már lehet futtatnunk a betanított modellünket.

A valóságban ezt úgy kell elképzelni, hogy amint az autó elindul, és automatikusan vagy manuálisan bekapcsoljuk az objektum felismerő rendszert, úgy a járművünk felveszi a kapcsolatot a felhővel, és amint ez létrejött megindulhat az adatküldés.

5.5. Tesztelés

A teszteléshez *docker* környezetet használtam, mivel a saját gépemben nincs megfelelő erősségű grafikus feldolgozó egység. Dockeren belül létrehoztam egy *ubuntu* disztribúciós virtuális környezetet, ahova feltöltöttem a szükséges *python*, videó és súlyfájlokat. Ahhoz, hogy mindent futtatni tudjak, ide is le kellett töltenem a szükséges követelményeket. A vizuális megjelenéshez *noVNC* ablakot használtam.

A tesztelésnél először meg kellett vizsgálnom, hogy milyen paramétereim vannak, és hogyan szeretném őket módosítani. Egy fontos szempontnak tartottam a bitráta vizsgálatát. Ebből is 2 fajtát néztem. Az első a CBR (Constant Bit Rate), azaz konstans bitráta, ami azt jelenti, hogy a videó folyamán egységnyi idő alatt (általában egy másodperc), mennyi adat áramoljon át. Ha ez az érték konstans, akkor hasznos lehet akkor, ha az átviteli sebességnek stabilnak kell lennie. Ennek viszont az az ára, hogy a videó egyes pontjain az eredetinél rosszabb képminőséget kapunk. Ellenkező esetben VBR-ről (Variable Bit Rate), azaz változó bitrátáról beszélhetünk [20][21].

Ezen felül vizsgáltam még a frissítési ráta értékét. Ehhez meg kellett mérnem, hogy a videó, amin tesztelek, mekkora FPS (Frame Per Second) értékkel rendelkezik. A mérést szintén *python* programban végeztem, az eredmény 27nek adódott. Így 3 limitet állítottam be. 30, azaz nincs korlát, mert ezt az értéket sosem lépte túl, 15 és 5.

A feldolgozást befolyásoló további tényező még a videó vagy élőkép minősége. Jelen esetben a felbontás 1920×1080 pixel volt, azaz *Full HD*. Ezt vettem 100 százaléknak, majd

vizsgáltam 80, 60, 40, 20 és 10 százalékon. A különböző videókat *ffmpeg* szoftver segítségével állítottam elő.

Ha minden eshetőséget vizsgálok, akkor ez 6x3x2x4, azaz 144 eshetőség. A tesztelést 2 részre bontottam. Az elsőben csak azt néztem, hogy a modelljeim milyen hatékonysággal elemzik a videóimat, a második esetben pedig az átvitel paramétereit vizsgáltam. Itt az átlagos sávszélességet, valamint az átvitel idejét mértem.

Az eredményeket minden alkalommal egy .csv kiterjesztésű fájlba mentettem. Ezeket a .csv fájlokat aztán a *10. ábrán* látható módon elemeztem.

```
for index, row in data.iterrows():
    frame = row['Frame']
    conf = row['Conf']

    if frame < 310:
        accuracy_neg += 1
    elif 310 <= frame <= 557:
        accuracy_pos += 1
        conf_values.append(conf)
    elif frame > 557:
        accuracy_neg += 1

average_conf = sum(conf_values) / len(conf_values) if conf_values else 0

normalized_accuracy = ((434-accuracy_neg) + accuracy_pos)/maximum_frame
```

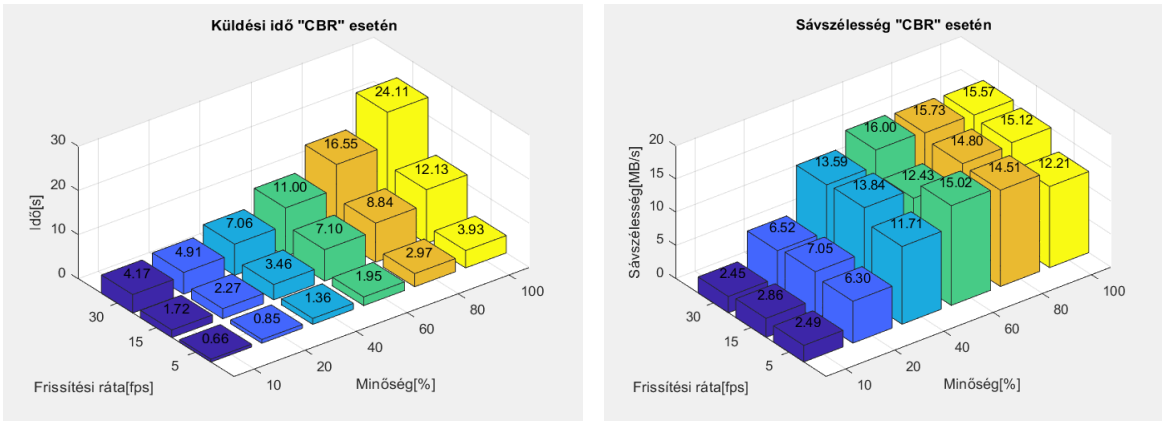
10. ábra Az adatelemzés folyamata – python kódrészlet

A videó, amit feldolgoztam 22 másodperc hosszú volt, 681 képkockából állt. Ezen a 310. *frame*-től az 557.-ig látszódott az 1. objektum, azaz a ház. A kiírt .csv fájlokból 2 adatot vizsgáltam. Az átlagos konfidenciát, amit igaz pozitív esetben mértem. Itt fontos megjegyeznem, hogy még mindig csak azokat az objektumokat listáztam, amelyek konfidenciája 75 százalék fölött volt. A másik, vizsgált paraméter a teljes hatékonyság volt. Itt szét kellett választanom 2 esetet. Az első a hamis pozitív, tehát amikor úgy ismerte fel az objektumot, hogy nem is volt ott, a másik az igaz pozitív, amikor pedig ott volt és fel is ismerte. Ez alapján a programkód 2 változót tart fent, az egyiket akkor növeli, ha a hamis pozitívban van érték, a másikat akkor, ha az igaz pozitívban. A végén ebből úgy számoltam hatékonyságot, hogy az összes hamis képkockaszámból kivontam az első változót, majd hozzáadtam a második változót és elosztottam a teljes képkocka számmal.

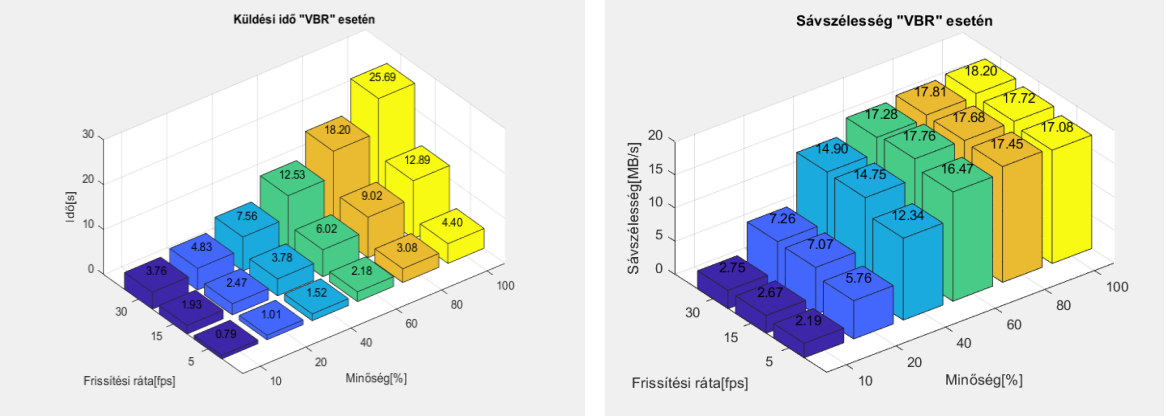
6. Eredmények

6.1. Átvitel

a) b)



c) d)



11. ábra CBR Videó küldési ideje (a), és az átlagos sávszélessége (b), VBR videó küldési ideje (c), és az átlagos sávszélessége (d)

Ahhoz, hogy a mért adatokat megfelelően tudjuk értelmezni, először tudnunk kell, hogy mennyi idő, és milyen sávszélesség szükséges az egyes paraméterű videók továbbításához a hálózaton. A méréseket Wifi, és 4G hálózaton is elvégeztem, de mivel mindkét esetben azonos eredményt kaptam (a legnagyobb eltérés 0,01 másodperc volt a teljes küldési időben), ezért most csak a Wifi hálózaton való kommunikáció eredményeit mutatom be, lásd *11.ábra*.

A küldési idők esetében megállapítható, hogy a frissítési rátától mind CBR, mind VBR esetben arányosan függenek. Ez az előzetes becslések alapján várható is volt, hiszen, ha fele annyi adatot kell elküldeni, akkor fele akkora lesz a küldési idő nagyságrendileg azonos küldési sebesség mellett. A minőséggel már korántsem ez a helyzet, itt a növekedés inkább exponenciális jellegű. A vizsgált videó hossza 22 másodperc, így kapásból kizárhatjuk az 1920×1080 pixeles felbontást. Ennek a képkockának küldésével folyamatosan csúszna a

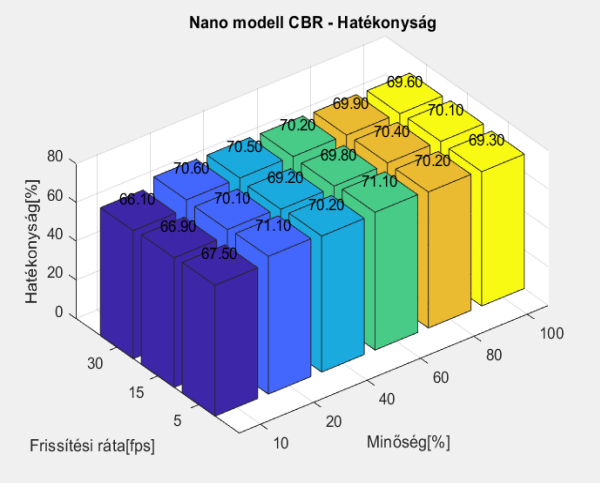
késleltetés a valósághoz képest. Ha a valóságot nézzük, ahol élőben jön a kép, akkor ez azt jelentené, hogy 1 képkocka konstans bitráta esetén 354, változó bitráta esetén 368 ezredmásodperc alatt menne át, tehát ennyit minimum csúszna az adat, amihez még hozzáadódik a képfeldolgozási idő, valamint a minimális adatvisszaküldési idő. Legjobb esetben ez 1, valamint 1,1 ezredmásodperc. A különbség 2 nagyságrend, a kérdés, hogy a feldolgozás milyen hatékonyan működik ezekben az esetekben.

Ahogyan azt előzetesen vártuk, konstans bitráta esetében a sávszélesség alacsonyabb, mint változó bitráta esetén. Az átlagos sávszélességek szinte függetlenek a frissítési rátától. A minőségben pedig egy olyan tendencia látszik, hogy csökkenthetjük a felbontást nagyjából 50 százalékig, azaz konkrét értékekkel számolva 960×540 pixelig, az átlagos sávszélesség eddig a pontig hozzávetőlegesen konstans. Innen viszont egy lineárishoz közelíthető, attól kicsit erősebben csökkenő értékesést láthatunk. Ez magyarázható azzal, hogy 50 százalékos felbontásnál valamilyen hálózati maximumba ütközik a küldés. Nagyobb méretű videóknál több adatod viszünk át egységnyi idő alatt, viszont ez azt is jelenti, hogy ilyen küldési idők mellett, legalább ezt a sávszélességet kell biztosítani. A modellhatékonyságokkal összevetve a dolgozatomban később megállapítást teszek arra, hogy melyik változat hatékonyabb.

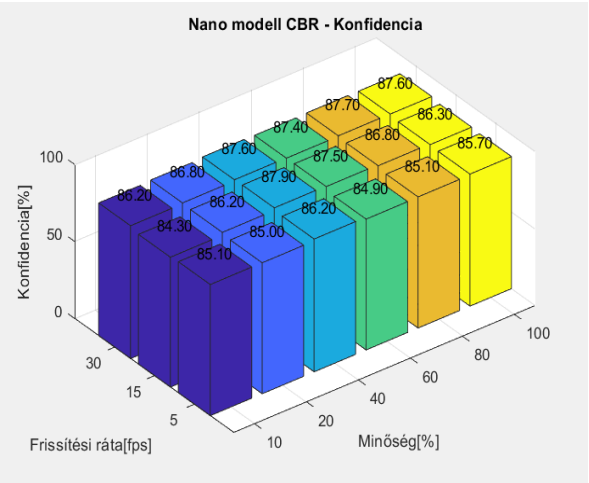
6.2. Modellek hatékonysága

6.2.1. Nano modell

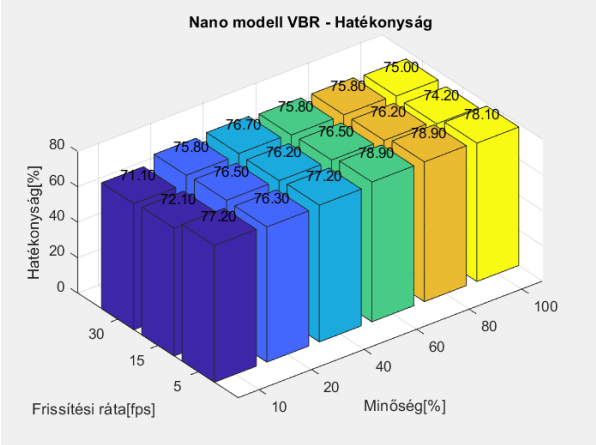
a)



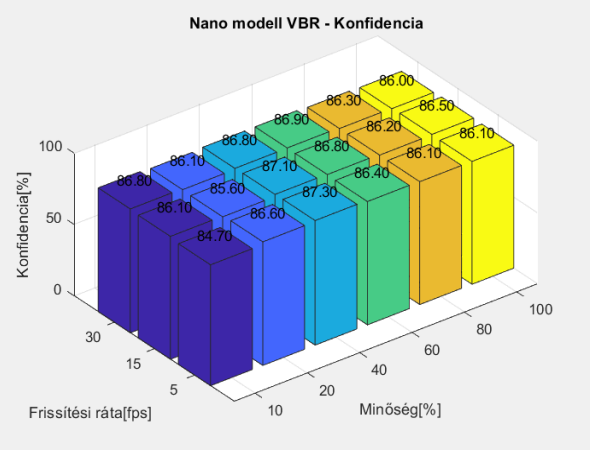
b)



c)



d)

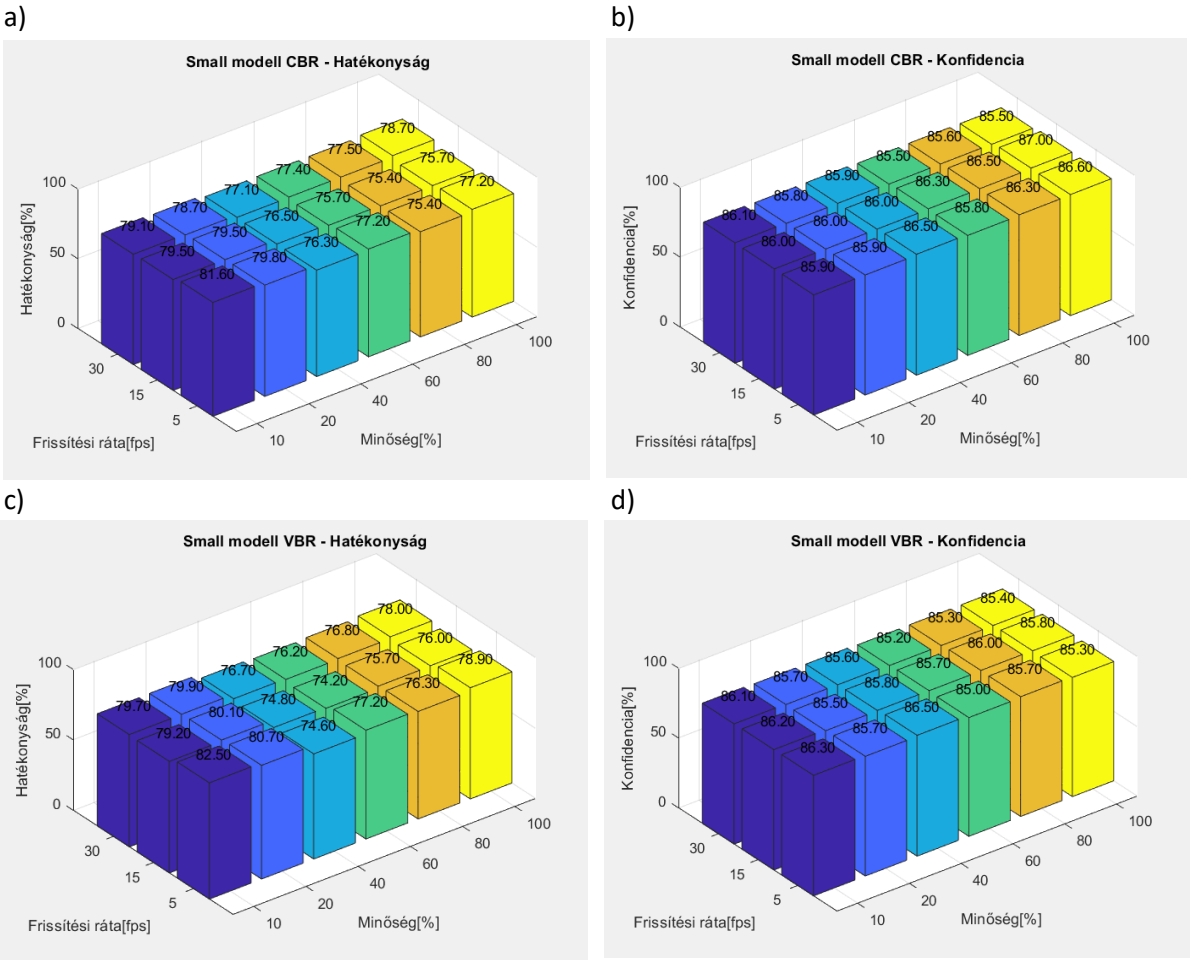


12. ábra Nano modell CBR hatékonyság (a), konfidencia (b), VBR hatékonyság (c), konfidencia (d)

Az első modell, amit teszteltem az a *nano* méretű volt, ahol konstan bitrátát alkalmaztam. Ennek eredményeit láthatjuk a 12. ábrán. A vizsgálat során minden esetben a hatékonyságot, valamit a konfidencia szintet mértem, majd ezeket ábrázoltam a frissítési ráta és a minőség függvényében. Azt már az első modell alapján levonhatjuk, hogy az frissítési ráta értékek nem befolyásolják, sem a konfidenciát, sem a hatékonyságot. Ezt vártuk, hiszen attól, hogy csak feleannyi, vagy hatod annyi képkockát vizsgálok, attól még nem lesz más a felismerés, ha arányosan hagyok ki a pozitívból és a negatívból. Szembetűnő még, hogy a hatékonyságra nagyobb hatással van a kép minősége, mint a konfidenciára. Ez alapján, kisebb képminőségnél kevesebb objektumot ismer fel a modell, vagy rosszkor, viszont, ha felismeri, akkor ugyan olyan magabiztosan, mint nagyobb képminőségnél.

Változó bitrátájú esetben lényegesen nagyobb értékeket kaptunk, mint konstans bitráta esetén. A hatékonyság átlagosan 5-10 százalékértékkel volt magasabb, a felismerés hatékonysága pedig közel változatlan maradt. A *nano* modell alkalmazása esetén a képfeldolgozási idő 40 és 50 ms között ingadozott.

6.2.2. Small modell



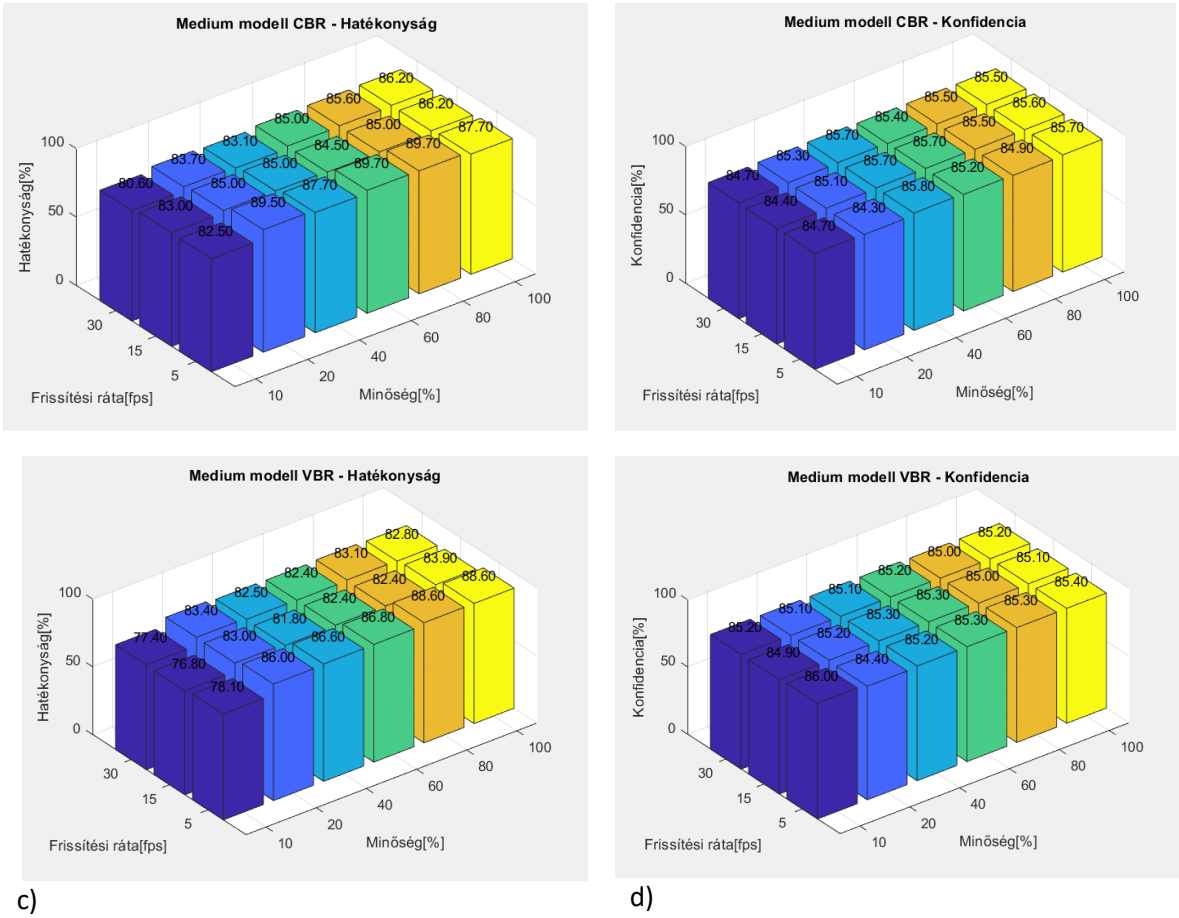
13. ábra Small modell CBR hatékonyság (a), konfidencia (b), VBR hatékonyság (c), konfidencia (d)

A small modell vizsgálatánál, ha a hatékonyságot nézzük, akkor azt látjuk, hogy a kisebb minőségű videók felzárkóztak a nagyobb felbontású videók mellé, sőt, konstans bitrátájú esetben a 30 és az 5, változó bitrátájú esetben az 5 frissítési ráta értékeknél a legalacsonyabb felbontásnál találkozunk a legnagyobb értékkel (*13. ábra*). Ennek köszönhetően *nano* modellhez képest itt tapasztaltuk a legnagyobb változást. CBR tesztelés során a hatékonyság nagyot ugrott, helyenként 5-10 százalékérték, viszont a VBR tesztelésnél alig változott valami az előző, kisebb modellhez képest, egyedül a gyengébb minőségnél, magasabbnál még romlott

is. Valószínűleg azért nőhetett a hatékonyság a kisebb felbontásnál, mert a hamis pozitív ismerések száma csökkent. Ezzel a modellel egy kép felismerése 80-110 ms időt vett igénybe.

6.2.3. Medium modell

a) b)



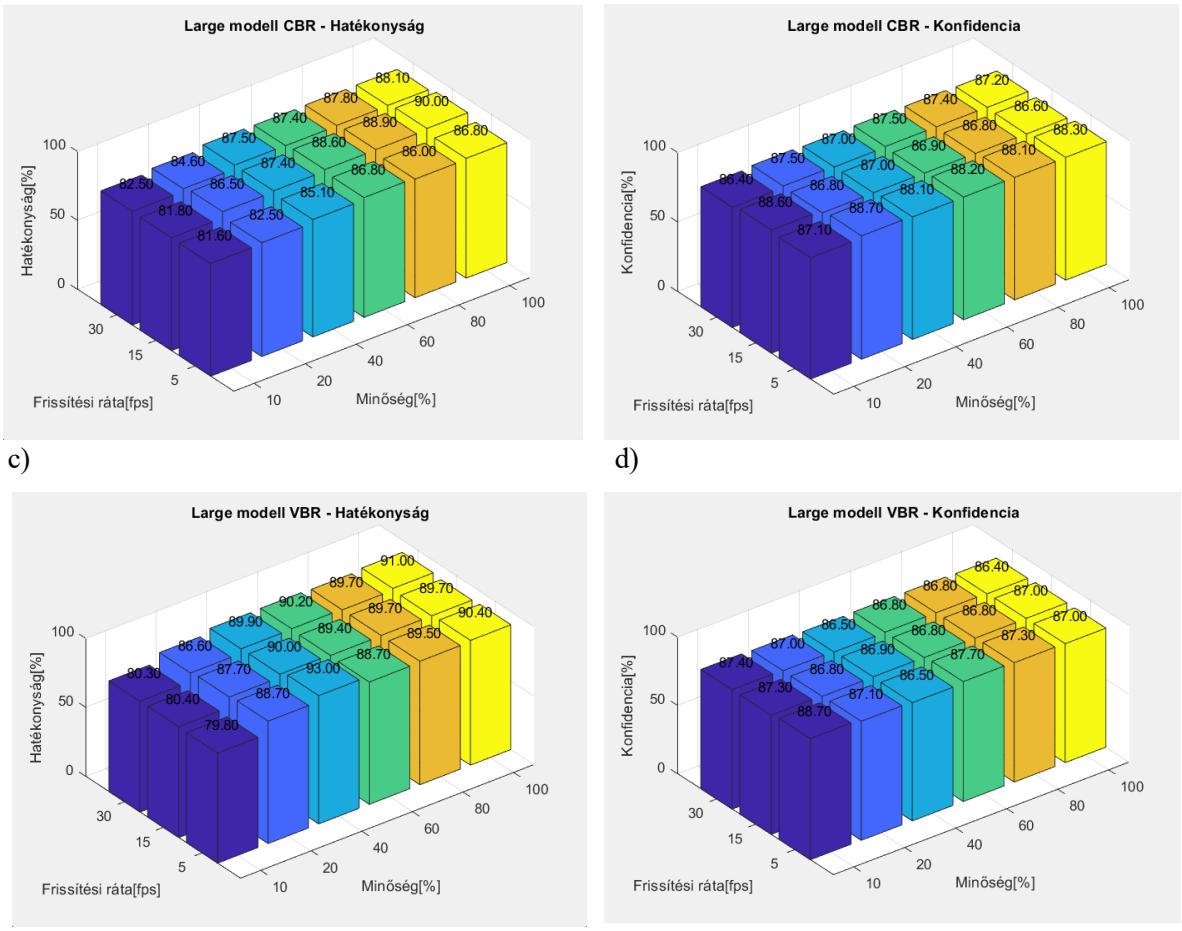
c) d)

14. ábra Medium modell CBR hatékonyság (a), konfidencia (b), VBR hatékonyság (c), konfidencia (d)

A m*edium* modell hatékonyságánál általánosságban megállapíthatjuk, hogy a 10 százalékos felbontásnál a *nano* modellhez hasonlóan visszaesett az érték, de ahhoz, és az előző modellhez képest is, itt már egyértelmű a növekedés. A konfidencia szint is növekedett, most VBR esetben látjuk, hogy a legkisebb minőségi beállítás mellett a legmagasabb a felismerési biztosság. Változó bitrátánál, az összes minőségen, és az összes frissítési ráta érték mellett közel azonosnak adódott a konfidencia. A képfeldolgozási idő itt 160-240 ms időt vett igénybe.

6.2.4. Large modell

a) b)



c) d)

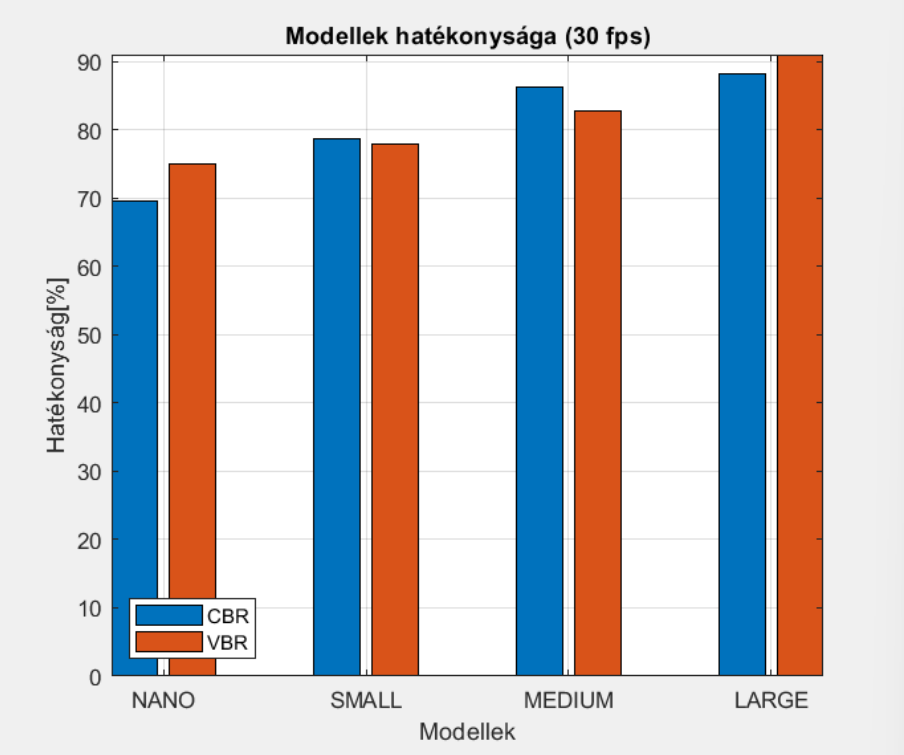
15. ábra Large modell CBR hatékonyság (a), konfidencia (b), VBR hatékonyság (c), konfidencia (d)

A *large* modell messze a legjobb teljesítményt nyújtja, ennek viszont az az ára, hogy egy kép feldolgozásához 280-440 ms időre van szüksége. Ha a hatékonyságot vizsgáljuk, akkor azt tapasztaljuk, hogy megint kialakult egy lépcsőzetes eloszlás a minőség függvényében konstans és változó bitráta esetén is. A modellek végéhez érve ki lehet jelenteni, hogy a vizsgált paraméterek vizsgált tartományán a konfidencia szintje független volt mind a frissítési rátától, mind a kép minőségétől.

6.3. Összesített elemzés

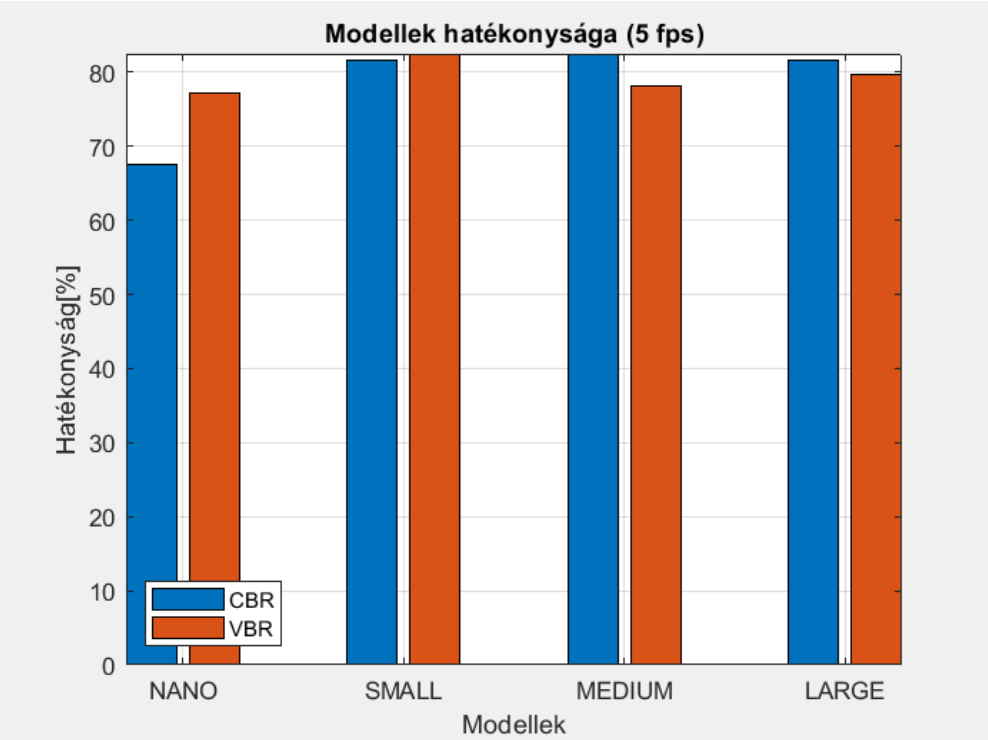
Az összesített elemzés célja, hogy az ismeretek tükrében meghatározzuk a legoptimálisabb paramétereket, amivel a kitűzött cél elérhető és megvalósítható a gyakorlatban. A döntéshez fontos ismernünk néhány információt az emberek, és a gépek reakcióidejéről. A vezetésnél beszélhetünk reakcióidőről, valamint arról az időről, amíg észlelem az objektumot,

a veszélyt és megadom rá a megfelelő reakciót. Ez az embereknél átlagosan 390 és 600 ms között mozog, a fiatalabbak esetében, akár elérheti a 220 ms-os értéket. A gépeknél az elfogadott átlagos cselekvési idő 500 ms. Ha ezeket a számokat figyelembe vesszük, akkor nagyjából olyan értéket kell választanunk, ahol a feldolgozási, valamint átviteli idő ez alatt az érték alatt van [22][23][24]. Az optimális paraméterek kiválasztását vizsgáltam saját mért adatokkal is, valamint erősebb grafikus feldolgozó egységet feltételezve is.



16. ábra Modellek hatékonysága 30 frissítési rátán

A konstans bitráta, és változó bitráta összehasonlítására vegyük alapul a *16. ábrát*. Itt a modellek függvényében ábrázoltam a hatékonyságot, valamint különválasztottam a VBR és CBR értékeket. Az összehasonlítást 30 frissítési ráta érték és 100 százalékos felbontás mellett végeztem. Azt tapasztaltam, hogy a hatékonyság a vártnak megfelelően a modellek méretének növekedésével növekszik. A bitráta összehasonlításban viszont nincs ilyen eredmény, nem tudjuk általánosan meghatározni, hogy melyik a hatékonyabb, ez a különböző modellek esetében eltér.



17. ábra Modellek hatékonysága 5 frissítési rátán

Ahhoz, hogy tisztábban lássunk megvizsgáltam még egy konkrét esetet (*17. ábra*). A frissítési ráta itt 5, a felbontás pedig 10 százaléka a teljes felbontásnak. Ezzel a legnagyobb és legkisebb méretű videót is megvizsgálom. Itt már azt sem tudjuk egyértelműen megállapítani, hogy a modellek méretének növelésével növekszik a hatékonyság is. A legnagyobb értékek a *small* és *medium* modelleknél található.

Az esettanulmányok alapján a következő megállapítások tehetők a feldolgozáshoz választott paraméterekről.

Az 5 FPS érték ugyan az átvitelnél jelentősen csökkenti a küldési időt, mégsem célszerű alkalmazni, mert ezt annak az árán éri el, hogy rengeteg adatot veszít, egészen pontosan a teszt során alkalmazott videón az adatok 80 százaléka elveszett. Ilyenkor minden 0,2. másodpercben érkezik egy adat. Ez 50 km/h sebességnél még csak 2,8 méter kiesést, de 130 km/h sebességnél már 7,2 méter kiesést jelent. Ezek az értékek nem tűnnek soknak, de balesetmegelőzési szempontból megengedhetetlen. 15 FPS esetén ezek az értékek már viszonylag elfogadhatóbbak: rendre 0,925 m és 2,4 m. A közlekedés során azonban minden pillanat számít, ezért itt a tartomány lehető legmagasabb értékét célszerű választani, jelen esetben a 30 frissítési ráta értéket.

Ha megvizgsáljuk a kükldési idő, az átlagos sávszélesség, és a hatékonyság arányát, akkor arra juthatunk, hogy sokkal nagyobb mértékben csökken a küldési idő és a sávszélesség, mint a hatékonyság, ami az összes vizsgált modellre igaz.

Mivel a konstans bitrátájú videó küldése kisebb sávszélességet igényel, ezért célszerű olyan modellt választani, ahol a hatékonyság CBR esetében nagyobb mint VBR alkalmazásakor. Ez leginkább a *small* és *medium* modellekre igaz.

A felbontásnál érdemesebb a küldési paramétereket vizsgálni, mivel az objektum detektálás során ami leginkább számított az a modell mérete volt. Ha a legkisebb felbontással megvizsgáljuk a *small* és *medium* modelleket, akkor a következő értékeket kapjuk. *Small* modellt alapul véve 79,1 százalékos, míg *medium* modellnél 80,6 százalékos volt a hatékonyság. Ez kisebb különbség, mint amekkora a modellek közötti feldolgozási idő eltérése.

A vizsgált paraméterek mellett a végső választásom tehát a *small* modellre esett. A modell konstans bitrátájú, 30 frissítési rátájú és 192×108 pixel felbontású videója az alábbiak szerint alakult: az észleléstől a reakcióig eltelt idő: 37 ms (a legrosszabb esetben ennyit kell várni a következő képkockára [22]) + 6 ms (egy képkocka küldési ideje) + 95 ms (a modell egy képkockára számított feldolgozási ideje), tehát összesen 138 ms volt. Ez bőven az autók által elfogadott cselekvési idő alatt van.

7. Összegzés

TDK munkám során az volt a célom, hogy vizsgáljam, milyen paraméterek és milyen beállítással szükségesek ahhoz, hogy a felhő alapú képfeldolgozást hatékonyan lehessen végezni. Azért van erre szükség, mert ha gyakran változó objektumokat észlelünk, akkor hatékonyabb működést tudunk elérni akkor, ha a tanító hálózat folyamatosan bővül. A másik szempont, hogy egy ilyen bővülő, nagyméretű neurális hálózatot nem célszerű lokálisan működtetni, valamint folyamatosan frissíteni.

A kutatásom első része adatgyűjtésből állt, képeket készítettem azokról az objektumokról, amelyeket fel szerettem volna ismerni. Ezek után elkészítettem az adathalmazomat, amit a YOLOv8 modell segítségével betanítottam. Ennek segítségével 4 különböző méretű súlyfájlt is létre tudtam hozni. A projektem másik része a kommunikáció, így ehhez is létrehoztam egy tesztelő környezetet *python* programozási nyelven. A jármű felhő kapcsolatot egy szerver-kliens páros valósította meg, melyek Wifi hálózaton keresztül kommunikáltak. Az átvitelt és a tanítást a következő paraméterek módosítgatásával teszteltem.

Videó minősége, frissítési ráta, bitráta milyensége (konstans vagy változó) és a különböző méretű modellek.

Az eredményeket értékelve kijelenthető, hogy az idő legnagyobb részét nem az adatok küldése, hanem a képfeldolgozás teszi ki. A hatékonyság és a konfidencia szint alig függ a frissítési ráta értékétől, és csak kis mértékben függ a videó minőségétől. Ezek a kijelentések abban a tartományban értendők, amelyben a vizsgálatot végeztem. Ezen paraméterek helyes megválasztása főként a videó átvitelének helyes megválasztása szempontjából fontosak.

A kutatásom alapján kijelentem, hogy összességében van relevanciája a felhő alapú feldolgozásnak, a szinte korlátlan mértékben elérhető erőforrások mellett már csak az átvitel paramétereit kell helyesen megválasztani.

Köszönetnyílvánítás:

Köszönöm témavezetőmnek, Dr. Fehér Gábornak, hogy motivált a TDK dolgozat elkészítésében, és hasznos tanácsaival segítette munkámat.

Irodalomjegyzék

[1] Gróf A.B. (2018): Vezetést segítő funkciók fejlesztése okostelefonra mély tanulás alapon. Tudományos Diákköri Dolgozat, p. 64. Budapesti Műszaki és Gazdaságtudományi Egyetem.

[2] Lee, J., Wang, J., Crandall, D., Šabanović S. and Fox, G. (2017): Real-Time, Cloud-Based Object Detection for Unmanned Aerial Vehicles. First IEEE International Conference on Robotic Computing (IRC), Taichung, Taiwan, 2017, pp. 36–43. https://doi.org/doi: 10.1109/IRC.2017.77

[3] https://github.com/ultralytics/ultralytics.git

[4] Zou, Z., Chen, K., Shi, Z., Guo, Y. and Ye, J. (2023): Object Detection in 20 Years: A Survey. in: Proceedings of the IEEE, vol. 111, no. 3, pp. 257–276. https://doi.org/10.1109/JPROC.2023.3238524

[5] https://medium.com/analytics-vidhya/evolution-of-object-detection-582259d2aa9b

[6] Jiang, P., Ergu, D., Liu, F., Cai, Y. and Ma B. (2022): A Review of Yolo Algorithm Developments. Procedia Computer Science 199 (2022) 1066–1073. https://doi.org/10.1016/j.procs.2022.01.135

[7] Révy G. (2018): Object detection with YOLO deep learning algorithm. Tudományos Diákköri Dolgozat, p. 34. Budapesti Műszaki és Gazdaságtudományi Egyetem.

[8] Bereczki M. (2018): Objektum lokalizás képeken mély neurális hálóval. Tudományos Diákköri Dolgozat, p. 31. Budapesti Műszaki és Gazdaságtudományi Egyetem.

[9] Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár P. and Lawrence Zitnick C. (2014): Microsoft COCO: Common Objects in Context. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds) Computer Vision – ECCV 2014. Lecture Notes in Computer Science, vol 8693. Springer, Cham. https://doi.org/10.1007/978-3-319-10602-1_48

[10] Gavrila D.M. and Philomin, V. (1999): Real-time object detection for "smart" vehicles. Proceedings of the Seventh IEEE International Conference on Computer Vision, Kerkyra, Greece, 1999, pp. 87–93 vol.1. https://doi.org/10.1109/ICCV.1999.791202

[11] Mollah, M.B., Islam K.R. and Islam, S.S. (2012): Next generation of computing through cloud computing technology. 25th IEEE Canadian Conference on Electrical and Computer Engineering (CCECE), Montreal, QC, Canada, 2012, pp. 1–6. https://doi.org/10.1109/CCECE.2012.6334973

[12] Ahmed, M. and Hossain, M.A. (2014): Cloud computing and security issues int he cloud. International Journal of Network Security & Its Applications (IJNSA), Vol.6, No.1. 25–36. https://doi.org/10.5121/ijnsa.2014.6103

[13] Köhler, M:, Eisenbach, M. and Gross, H-M. (2022): Few-Shot Object Detection: A omprehensive Survey. arXiv:2112.11699v2. https://doi.org/10.48550/arXiv.2112.11699

[14] Long, L.N., Hanford, S.D:, Janrathitikarn, O., Sinsley G.L. and Miller, J.A. (2007): A Review of Intelligent Systems Software for Autonomous Vehicles," 2007 IEEE Symposium on Computational Intelligence in Security and Defense Applications, Honolulu, HI, USA, 2007, pp. 69–76. https://doi.org/10.1109/CISDA.2007.368137

[15] Yao, J., Fidler, S. and Urtasun, R. (2012): Describing the scene as a whole: Joint object detection, scene classification and semantic segmentation. IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, 2012, pp. 702–709. https://doi.org/10.1109/CVPR.2012.6247739

[16] Hoeser, T. and Kuenzer, C. (2020): Object Detection and Image Segmentation with Deep Learning on Earth Observation Data: A Review-Part I: Evolution and Recent Trends. Remote Sens. 12, 1667. https://doi.org/10.3390/rs12101667

[17] O'Shea, K. and Nash, R. (2015): An Introduction to Convolutional Neural Networks. arXiv:1511.08458. https://doi.org/10.48550/arXiv.1511.08458

[18] https://www.tensorflow.org/

[19] https://roboflow.com/

[20] Assuncao P.A.A. and Ghanbari, M. (2000): Buffer analysis and control in CBR video transcoding. in IEEE Transactions on Circuits and Systems for Video Technology, vol. 10, no. 1, pp. 83–92, Feb. 2000. https://doi.org/10.1109/76.825863

[21] Assuncao, P.A.A. and Ghanbari, M. (2000): Buffer analysis and control in CBR video transcoding. in IEEE Transactions on Circuits and Systems for Video Technology, vol. 10, no. 1, pp. 83–92, Feb. 2000, https://doi.org/10.1109/76.825863

[22] Rydzewski, A. and Czarnul, P. (2021): Human awareness versus Autonomous Vehicles view: comparison of reaction times during emergencies. 2021 IEEE Intelligent Vehicles Symposium (IV), 732-739. https://doi.org/10.1109/iv48863.2021.9575602

[23] https://medium.com/predict/making-roads-safer-with-self-driving-cars-and-5g-c1e28526362c

[24] Melegh G. (1995): Reakcióidő a közúti közlekedésben. Közlekedéstudományi Szemle 45. 319–329.