



TUDOMÁNYOS DIÁKKÖRI DOLGOZAT

VIRÁG NÁNDOR GÁBOR

2024.11.15.

DUNAÚJVÁROSI EGYETEM



**TUDOMÁNYOS DIÁKKÖRI
DOLGOZAT**

*Kézzel felismerő szoftver mély tanulási
módszer alkalmazásával*

Témavezető:

DR. HABIL KATONA JÓZSEF

KIRSCH NORBERT

Hallgató:

VIRÁG NÁNDOR GÁBOR

MÉRNÖKINFORMATIKUS

2024.11.15.

Rezümé

A mai világban véleményem szerint nehezen lehet elmenni a gyorsan fejlődő mesterséges intelligencia mellett, illetve szerintem fejlesztőként akár érdemes is lehet kicsit jobban beleásni magunkat. Manapság életünk egyre több területén jelenik meg amik akár hasznosak is lehetnek. Így jutottam én is arra a döntésre, hogy megpróbálok valami olyat alkotni, ami segíthet az emberek mindennapi életében.

Szakdolgozatom célja egy olyan szoftver implementálása, ami képes a siketek és némák által használt angol abc betűinek megfelelő kézjelek valós időben történő felismerésére. Amelynek alapját egy mély tanulási módszerrel betanított objektumfelismerő modell fogja adni.

A modellem tanításához már egy meglévő objektum detektáló alapmodellt vettem alapul, amit transfer learning segítségével tanítottam át saját céloknak megfelelő objektumfelismerésre. Annak érdekében, hogy a modell minél jobb eredményekre legyen képes szükségem volt egy nagy adathalmazra, ahol az objektumok változatos háttérben, pozícióban és fényviszonyok mellett jelennek meg. A munkámat Python programozási nyelven végeztem el, ahol segítségemre voltak a már előre megírt mesterséges intelligenciához használatos könyvtárak.

A dolgozatomban arra törekszem, hogy minél érthetőbben és részletesebben mutassam be egy mély tanulási modellt alkalmazó szoftver implementációját és annak nehézségeit.

Abstract

In today's world, in my opinion, it is difficult to ignore the rapidly developing artificial intelligence, and as developers, it might even be worth delving deeper into it. Nowadays, artificial intelligence appears in more and more areas of our life and it can be quite useful. This is what led me to the decision to try and create something that could help people in their everyday life.

The goal of my thesis is to implement a software capable of recognizing the hand signs corresponding to the letters of the English alphabet used by the deaf and mute in real-time. This will be based on an object detection model trained using a deep learning method.

For training my model, I used an existing object detection base model, which I retrained using transfer learning to suit my specific object recognition needs. To ensure the model achieves the best possible results, I needed a large dataset where the objects appear in diverse backgrounds, positions, and lighting conditions. I did my work in the Python programming language, where I was helped by the already pre-written libraries used for artificial intelligence.

In my thesis, I aim to present the implementation of a software using a deep learning model and the challenges associated with it as clearly and comprehensively as possible.

Tartalomjegyzék

1.	Bevezetés	1
1.1.	A dolgozat célkitűzései	2
1.2.	A dolgozat felépítése	3
2.	Felhasznált technológiák	4
2.1.	Mesterséges intelligencia	4
2.1.1.	Gépi tanulás	4
2.1.2.	Mély tanulás	4
2.2.	Neurális hálózatok.....	5
2.2.1.	Tanítási módok	7
2.3.	Konvolúciós neurális hálózat (CNN)	7
2.4.	Keretrendszerek.....	8
2.4.1.	TensorFlow	9
2.4.2.	Pytorch.....	9
3.	Adathalmaz készítése.....	11
3.1.	Kézjelek.....	11
3.2.	Adatok gyűjtése.....	12
3.3.	Annotáció	13
3.3.1.	LabelImg.....	13
3.3.2.	Roboflow	14
4.	Kézel felismerő szoftver implementációja	16
4.1.	Modell tanítása	16
4.2.	Vizualizációs teszt.....	20
4.3.	Valós időben történő felismerés	21
5.	Fejlesztési során felmerült problémák	26
5.1.	GPU használata	26
5.1.1.	Verzió váltás	26

5.1.2. WSL.....	26
5.2. Implementáció alatt felmerült hibák	27
6. Modell tesztelése	30
6.1. Teljes modell tesztelése.....	30
6.2. Modellek összehasonlítása	37
6.3. Valós időben történő felismerés tesztelése.....	40
6.3.1. A tesztelők	41
6.3.2. A vizsgálat	41
6.3.3. Tesztelés	41
6.4. További tesztelési lehetőségek	43
7. Továbbfejlesztési lehetőségek	44
7.1. Adathalmaz bővítése	44
7.1.1. Gesztusokkal való bővítés	45
7.2. Szoftverszintű fejlesztések	46
7.3. Oda és vissza kapcsolat	47
8. Összegzés.....	48
Irodalomjegyzék	49
Ábrajegyzék	53
Táblázatjegyzék	54

1. Bevezetés

Manapság a rohamosan fejlődő technológiák mellett nehéz elmenni, különösen a mesterséges intelligencia mellett, ami az utóbbi pár évben hatalmas léptekben halad előre és tör utat magának egyre több területen ezzel akár könnyítve az emberek életet vagy munkáját. Például ma már léteznek olyan algoritmusok, amik képesek a jól megfogalmazott mondatainkból egészen jó weboldal dizájnokat, képeket generálni, illetve akár magát a fejlesztését is képesek megkönnyíteni számunkra. De ide sorolhatjuk az olyan mesterséges intelligenciákat, amelyek képesek képeken objektumokat felismerni, detektálni. A szakdolgozatom témáját pont az ilyen fajta szoftverek motiválták, amit akár igencsak hasznos dolgokra is fel lehet használni.

A dolgozatom célja egy olyan szoftver megalkotása, ami képes a siket és némák által használt kézjelek felismerésére valós időben. Minden betűnek meg van a saját kézjele az angol ábécében így ezeket a kézjeleket vettem alapul. A magyar kézjel ábécében vannak olyan betűk, amiknek a jelentéséhez egy mozgásrészlet megtanítása lett volna a szükséges, de tekintve azt, hogy számomra ez egy teljesen új terület így maradtam az angolnál, ahol ezek a betűk képről felismerhetők. De úgy gondoltam ez magában annyira még nem mondható látványosnak és használhatónak így jött a valós időben történő felismerés ötlete. Ennek a funkciónak köszönhetően a szoftverem már annál érdekesebb és használhatóbb hiszen képes valós időben visszajelzést adni a betűkről kézjelek alapján, illetve, ha ezt tovább fejlesztem, ahol már képes szavakat is felismerni jelentősen megkönnyítheti a kommunikációt a siketek és némákkal rengeteg területen. Gondolok itt akár egy állásinterjúra is. Véleményem szerint haladni kell a korról és nyitni kell akár minimálisan is a mesterséges intelligencia felé, mert ide tart a jövő és nem szabad ettől lemaradni. Szakdolgozatommal próbáltam erre a területre is kisebb betekintést is nyerni, amit úgy gondolok egészen jól sikerült.

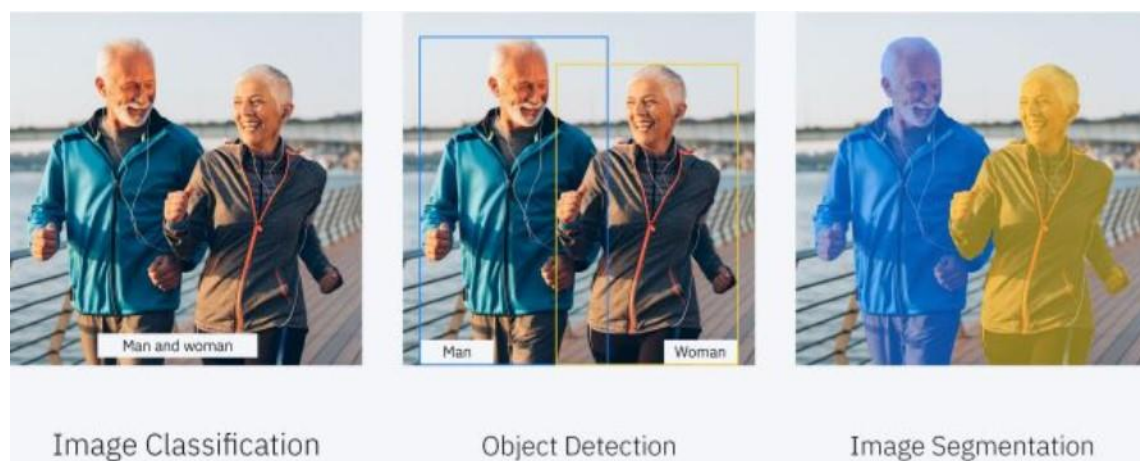
Az ilyen objektumfelismerő algoritmusok mögött úgynevezett neurális hálózatok vannak, amik próbálják utánozni az emberi agyban lévő neuronok működését. Van egy külön neurális hálózat fajta, ami kifejezetten erre a célra a legalkalmasabb ez az úgynevezett konvolúciós neurális hálózat. Ezeknek a tanításához szükségünk van robosztus képi adathalmazra, ami lehetőleg minél változatosabb képeket tartalmaz az objektumról. A jó eredmények eléréséhez szükséges azonban az adatok precíz felcímkézése és az objektumok megjelölése.

Az objektumfelismerés átfedésben van más számítógépes látási technikákkal, melyeket három csoportra bonthatunk [1].

Ezek egyik csoportja a kép osztályozás (image classification), ahol a képeket előre meghatározott osztályokba soroljuk kategóriák szerint [28]. Mondhatjuk azt, hogy ez volt az alapkőve a számítógépes látási technológiáknak [1]. Itt nem jelöli meg az objektumokat az algoritmus szimplán eldönti egy képről, hogy az osztályok alapján mit tartalmaz [1].

Az objektumfelismerés (object detection) ezzel szemben meghatározza az egyes objektumokat egy képen a megadott kategóriák szerint [1]. Az objektumfelismerés lokalizálja és minden objektumpéldányhoz hozzárendel egy megfelelő osztálycímket az előre meghatározott osztályok lehetőségei közül [27], [42]. Az 1. ábrán látható külön választva, bekeretezve határozza meg, hogy pontosan hol találhatók az adott objektumok [27].

A számítógépes látásban jelenleg az egyik legnépszerűbb technológia a képszegmentálás (image segmentation), amely a mintafelismerés és képmegértés alapját képezi [26], [41]. Ez szintén képes meghatározni és elkülöníteni az objektumokat, de itt nem egy dobozzal szegmentáljuk őket, hanem pixel szinten határozzuk meg, hogy ezek az objektumok hol találhatók a képen [1].



1. ábra: Különböző számítógépes látási lehetőségek [1]

1.1. A dolgozat célkitűzései

A szakdolgozat célkitűzései 5 fő pontba sorolhatók:

- A témához kapcsolódó szakirodalmak gyűjtése, áttekintése és megértése.
- Szoftverrel szembe támasztott elvárások, célok bemutatása.
- Egy kézzel felismerő szoftver megtervezése, ami képes a jelek valós időben történő felismerésére.
- Implementáció megvalósítása. A fejlesztés során felmerült problémák, hibák ismertetése, illetve javítása.
- Szoftver tesztelése és az eredmények kiértékelése. Továbbfejlesztési lehetőségek bemutatása.

1.2. A dolgozat felépítése

1. Az első részben, a 2. fejezetben ismertetem a témához kapcsolódó fogalmakat, bemutatom a választott keretrendszereket.
2. A 3. fejezetben az adathalmaz gyűjtésének folyamatát, feldolgozását, illetve annak mesterséges bővítési lehetőségét mutatom be.
3. A 4. fejezetben a szoftver implementációjáról fogok részletesebb bemutatást tartani, illetve bemutatom a kész szoftvert.
4. Az 5. fejezetben bemutatom a nehézségeket és a felmerült problémákat tanítás során és ezeknek a megoldásait.
5. A 6. fejezetben tesztelem, hogy a kész szoftver, hogyan teljesít olyan képi adathalmazon, amit még nem láttot, illetve összehasonlítom egy másik modellel.
6. Az utolsó részben, a 7. és 8. fejezetben ismertetem milyen továbbfejlesztési opciókkal lehet a szoftvert bővíteni. Összegzem a dolgozatom eredményeit és a személyes fejlődésemet.

2. Felhasznált technológiák

Dolgozatom megkezdéséhez elengedhetetlennek éreztem azt, hogy bővítsem a tudásom elméleti szinten a mesterséges intelligenciában, hiszen hiába csinálunk valamit, ha nem értjük, hogy ez hogyan is működik és mi van a háttérben. Ebben a fejezetben ezeket az ismereteket szeretném bemutatni.

2.1. Mesterséges intelligencia

Ez a téma abszolút nem új keleti dolog hiszen már az 1943-as években történtek kezdeményezések Warren McCulloch és Walter Pitts által. Viszont napjainkban indult igencsak rohamos fejlődésnek. A mesterséges intelligencia olyan tudomány terület, ahol olyan szoftvereket és eszközöket próbálnak készíteni, amelyek képesek érvelni, tanulni és olyan módon cselekedni, ami emberi intelligenciát igényelne. Emellett segítségünkre is vannak hiszen képesek olyan monoton munkákat elvégezni az ember helyett, ami alapvetően rengeteg időt venne igénybe, de ezek az algoritmusok jelentősen rövidebb idő alatt elvégzik ezeket. Ez a terület számos tudományágat foglal magába, mint például a számítástechnikát, adatelemzést, előrejelzéseket, természetes nyelv feldolgozást és még nem tudhatjuk, hogy a jövőben mennyi új területen fog megjelenni. Szerintem ettől nem szabad félni, hanem haladni kell a korral. [4-5], [29]

2.1.1. Gépi tanulás

A gépi tanulást arra alkalmazzák, hogy számítógépeket tanítsanak adatokból és a hibázásokból, tapasztalataiból képes legyen önmagát javítani külső programozási behatás nélkül. Mesterséges neurális hálózatokat tanítanak arra, hogy képesek legyenek elemzési, előrejelzési és hasonló feladatok elvégzésére. A gépi tanulás, azaz machine learning a mesterséges intelligencia egy részhalmazát képezi. Előnyei a gépi tanulásnak, hogy kisebb erőforrásra van szüksége, képes a CPU-t használva elvégezni a tanítást, kisebb adathalmazra van szüksége a tanításhoz. A mély tanulással szemben gyorsabb a tanulása viszont kisebb a pontossága is.

2.1.2. Mély tanulás

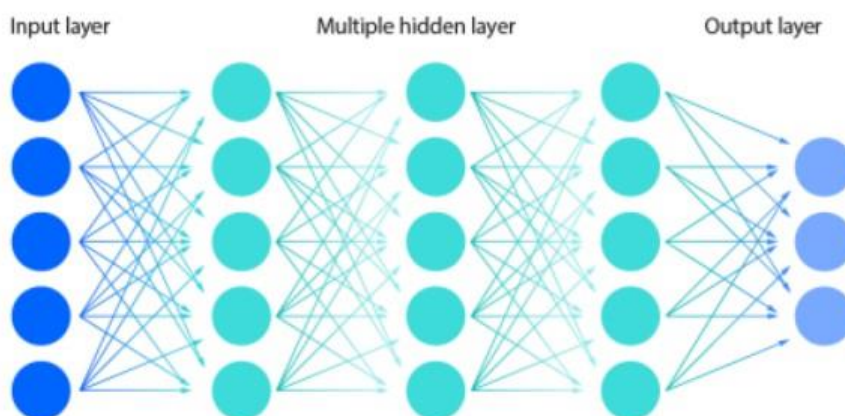
A mély tanulás pedig a gépi tanulás részhalmazát képezi [21]. Ez a tanulási mód is mesterséges neurális hálózatokat használ az információk feldolgozására. Viszont, ha a neurális hálózatok meghaladják a három vagy több réteget akkor ezeket mélynek nevezik. Innen ered a mély tanulás vagy is a deep learning elnevezés. A konvolúciós neurális

hálózatok főként erre a tanulási módra jellemzőek. Számos olyan feladatban alkalmazzák manapság, amelyet a mesterséges intelligenciához kapcsolunk, mint a képfelismerés, objektumfelismerés vagy akár a beszédfelismerés [5]. Ebből kifolyólag az én szoftverem is mély tanulást fog alkalmazni. Igaz, hogy a tanítás tovább tarthat, mint a gépi tanulás esetében viszont pontosabb végeredménnyel lesz képes dolgozni, illetve sokkal kevesebb emberi beavatkozást igényel. Viszont hátránya, hogy a tanításhoz szükséges egy nagy teljesítményű GPU.

2.2. Neurális hálózatok

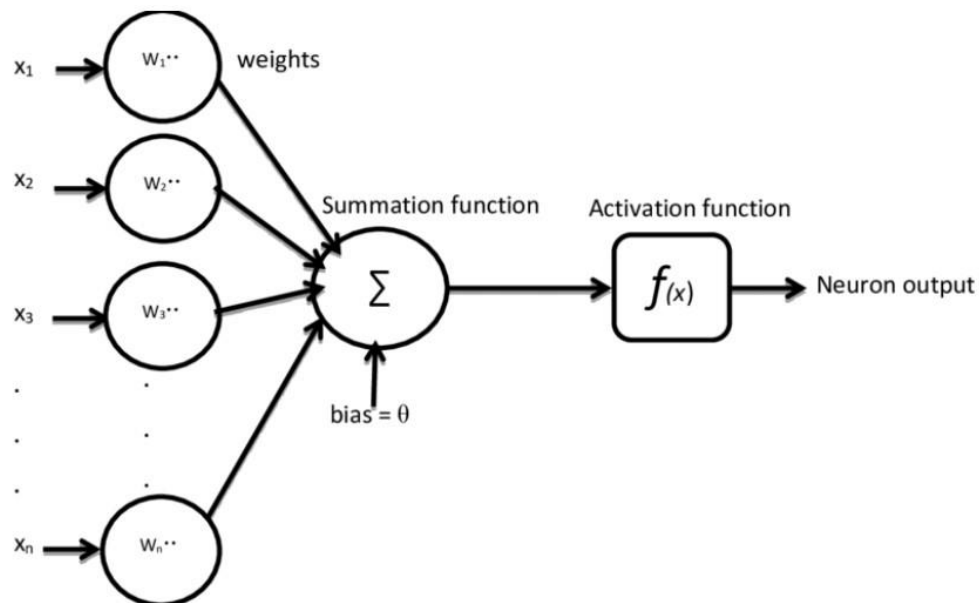
Az én gondolkodásom szerint az első lépés elméleti szinten a neurális hálózatok megismerése volt. A mesterséges neurális hálózat egy gépi tanulási program vagy modell, amely az emberi agyban található neuronok működését próbálja utánozni, azaz döntéseket, következtetéseket levonni [2].

Egy ilyen rendszerben rengeteg mesterséges neuron található, melyek rendezett topológiájú, nagymértékben összekapcsolt rendszerekből állnak, illetve rendelkeznek tanulási algoritmussal, amihez tanulási adata van szükségük és a tanulást követően képesek a megtanult információt felhasználni. Ezek rétegekre bontva találhatók meg a hálózatokban, amelyek a 2. ábrán láthatóak. A bemeneti és kimeneti rétegek között egy vagy több rejtett réteg található ezek szám szerinti megállapítására nincsen külön szabály. Szabadon megszabhatóak a feladat teljesítéséhez mérten, de ha egy neurális hálózatban a rejtett rétegek száma átlépi a hármas számot akkor a fent említett definíció alapján már mély tanulásról beszélünk. [6], [31-32]



2. ábra: Neurális hálózatok [2]

Ahogy a mi agyunkban is, ha valami inger ér minket a neuronok aktiválódnak és továbbítják az adatot egymást között. A mesterséges neuronok is ilyen elven működnek ezek a neuronok több-bemenetűek és egy-kimenetű eszközök, melyek között egyfajta nemlineáris kapcsolat van. A bementből érkező adatot ahhoz, hogy megkapjuk a kimenő adat értékét egy tipikusan nemlineáris függvényt kell alkalmaznunk, melyet aktiváló vagy aktivációs függvénynek nevezzünk. Ezt mutatja be az 3. ábra. [6], [33-34]



3. ábra: A mesterséges neuronok általános felépítése [13]

Ezeknek a hálózatoknak a tanításához szükségünk van adatokra, melyek alapján az algoritmus képes fejlődni. Ezek a modellek az elején rosszul fognak teljesíteni, de a tanulás során elkezdik javítani magukat és fejlődni kezdenek egészen addig amíg képesek meghozni a saját döntéseiket. Hasonlóképpen, mint egy gyerek, aki éppen fejlődésben van. De még a mai legnagyobb mesterséges intelligencia modellek sem érnek fel egy 4 éves gyermek tanulási képességeihez. Az emberi agy ebben a korban csak a látásával képes ötvenszer több információt feldolgozni, mint a mesterséges neurális hálózatok és akkor még nem is vettük a hallást, érintést stb. Mondhatni még jóval le vannak maradva ezek a hálózatok hozzánk képest, de erre halad a világ és kitudja meddig jutnak el. Viszont visszatérve ezek a neurális hálózatok, ha jól vannak feltanítva rendkívül hatékony dolgokra használhatók. Képesek az adatok nagy sebességű osztályzására, csoportosítására. Használhatók beszédfelismerésre, képfelismerésre, melyek az emberek számára rengeteg időbe telnek. Ezek a modellek képesek akár percek alatt ezt elvégezni segítve a munkánkat [2].

2.2.1. Tanítási módok

Egy modell tanítása során alapvetően háromfajta módot alkalmazhatunk. Ez lehet felügyelt tanítás, nem felügyelt tanítás és ennek a kettőnek a keveréke a félig felügyelt. Lényegében itt a fő különbség a tanuláshoz használt adatok feldolgozásán van.

A felügyelt tanítás során felcímkézett, azaz a feldolgozás során megadtuk, hogy az adatok mit tartalmaznak [30]. Így a tanítandó algoritmusunknak van fogalma arról, hogy mi lenne a helyes kimenet.

Ezzel ellentétben a felügyelet nélküli tanítás során az algoritmus önállóan dolgozik és az adatokat belső szerkezetét, mintáit tanulják meg minden másfajta útmutatás nélkül. Így itt semmiféle előfeldolgozásra nincs szükség az adatoknak [35].

A kétfajta tanítási mód vegyítése a félig felügyelt tanítás, ahol elsőnek a modellt egy kisebb mennyiségű felcímkézett adaton megtanítják az alapok felismerésére [36]. Majd ezután ezt a modellt felcímkézetlen adatokon tovább tanítják, hogy felismerje az adatok jellemzőit ezzel tovább finomítva a modell teljesítményét.

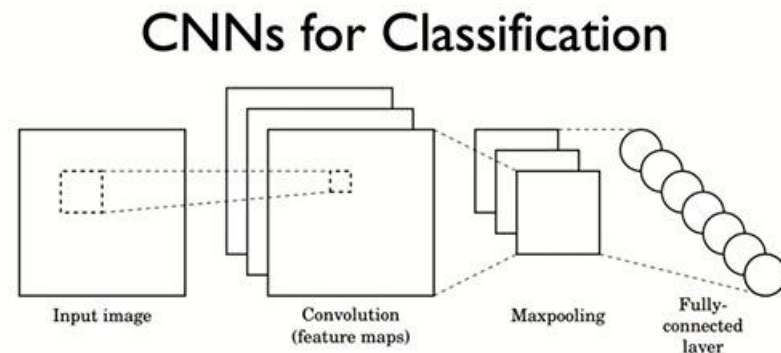
2.3. Konvolúciós neurális hálózat (CNN)

A konvolúciós neurális hálózat egy olyan speciális típusú mesterséges neurális hálózat, ami kifejezetten képfelismerésre, arcfelismerésre, számítógépes látásra és ezekhez hasonló célokra fejlesztettek ki [22]. A konvolúciós neurális hálózatok kiemelkedően hatékonyak a képadatok elemzésében, mert képesek automatikusan megtanulni azokat a jellemzőket, amelyek a képeken található mintázatok, részletek felismeréséhez szükségesek. [22], [37-38]

A CNN legfontosabb része a konvolúciós réteg ez egy speciális matematikai műveletet hajt végre, amelyet konvolúciónak hívnak. Ebben a rétegben úgy mond egy filtert mozgat végig a képen az algoritmus ez a 3. ábrán látható, amit mindig eltol x mennyiségű pixellel. Ez a matematikai művelet minden egyes helyen kiszámolja a filter és a képrészlet közötti pontszorzatot. Az így kapott értékek egy új mátrixot alkotnak, amit jellemző térképnek (feature map-nek) neveznek. Ez tartalmazza az egyes képek jellemzőit. [39-40]

Miután a konvolúciós réteg kiszámítja a jellemző térképet, egy nemlineáris aktivációs függvényt, leggyakrabban a ReLU (Rectified Linear Unit) alkalmazták erre a

célra. A ReLU a negatív értékeket nullára állítja, így könnyítve a bonyolult mintázatok tanulását ezzel is gyorsítva a folyamatot. [3], [39-40]



4. ábra: Konvolúciós neurális hálózat folyamata [3]

A következő lépés a pooling ami arra szolgál, hogy csökkentsük a dimenzióját a jellemző térképnek [3]. Ezáltal csak a legfontosabb információk maradnak meg így gyorsítva a folyamatot. A leggyakrabban használt pooling típus a max pooling, amely megvizsgálja a bemeneti mátrix értékeit és csak a legnagyobb értéket tartja meg [24].

A konvolúciós rétegek és pooling rétegek sorozata után a kimenetet egy vagy több teljesen összekapcsolt réteghez csatlakoztatják. Ezek az összekapcsolt rétegek felelősek a végső predikcióért.

2.4. Keretrendszerek

Ebben az alfejezetben szeretnék kitérni arra, hogy jelenleg milyen keretrendszerek állnak rendelkezésünkre, ha mesterséges intelligenciával szeretnénk foglalkozni. Mivel a Máv SZK Zrt. Információs és technológiai szolgáltatások üzletágnál vagyok duális hallgató és a külső konzulensem is itt dolgozik, aki mesterséges intelligenciával foglalkozik, így én az Ő ajánlásával a TensorFlow-t választottam a Pytorch-val szembe. Itt egy pillanatra kitérnék a programozási nyelv választására. Jelenleg a Python az a nyelv, ami a legelterjedtebb machine learning vagy deep learning modellek készítésének a körében és ez az indok vezetett arra, hogy ezt válasszam a projektem megvalósításához. Egy kimutatás alapján a GitHub-on található gépi tanulási és mély tanulási projektek több, mint 56%-a Python nyelven íródott [25]. Így teljesen egyértelművé vált számomra, hogy ez az a programozási nyelv, amiben érdemes megkezdeni a fejlesztését a projektemnek. Így ez egy remek indokot adott arra, hogy bővítsem a tudásom ebben a nyelvben egy igen versenyképes módon.

2.4.1. TensorFlow

A TensorFlow egy nyílt forráskódú keretrendszer, amelyet a Google Brain Csapata fejlesztett ki gépi tanulási, ezen belül is különösen mély tanulási feladatokra [19]. Széles körben használják mesterséges intelligencia modellek tanítására és futtatására.

A TensorFlow több szempontból is jó választás objektumfelismerési projektekhez. Rendelkezik Object Detection API-val, amely előre betanított modelleket kínál, mint például R-CNN, YOLO, vagy SSD. Ezek a modellek már képesek a tárgyak felismerésére. A továbbtanítás (transfer learning) alkalmazásával a modellek könnyen finomhangolhatóak saját egyedi céljainkra, ami gyorsabb tanulást és jobb teljesítményt jelenthet emellett kevesebb adatra is van szükségünk a tanításukhoz.

Erőforrásigénye viszonylag nagynak mondható viszont támogatja az NVIDIA CUDA-t, mely az NVIDIA GPU-hoz készült segédsoftver, mellyel a GPU erőforrásait igénybe tudjuk venni a tanítás gyorsításához. Viszont a TensorFlow 2.10 utáni verziójával ez Windows alatt nem működik és bárhogya is próbálkozunk a CPU-t fogja igénybe venni, ami viszont igencsak kevésnek minősül a tanítás során. Erre vannak módszerek, hogy ezt kiküszöböljük erre a későbbiekben kitérek még.

A TensorBoard a TensorFlow egyik vizualizációs eszköze, ami lehetővé teszi számunkra, hogy valós időben nyomon követhessük a modellünk tanulását, így gyorsabban észlelhetjük, ha a modellünk alulteljesít vagy éppen túlteljesít. Erről készít nekünk grafikonokat, amelyekkel összehasonlíthatjuk más modelljeinket vagy előző próbálkozásainkat.

2.4.2. Pytorch

A Pytorch a Meta AI csapata által fejlesztett szintén nyílt forráskódú keretrendszer, amely szintén gépi látási, mesterséges intelligencia feladatok megvalósítására lett létrehozva és törekszik arra, hogy egyszerre legyen gyors és jól használható [20]. Egyelőre hasonló népszerűségük van a TensorFlow-val, de talán a Pytorch egyre jobban halad előre és veszi át a piacot.

Pytorchon belül a YOLO modell csomag a legnépszerűbb, amelyből viszonylag egyszerűen tudunk előállítani objektumfelismerő modelleket.

Konfiguráció szempontjából is hasonló a társához. A Pytorch is támogatja a GPU-t és használható vele mind a NVIDIA CUDA és AMD RoCM segédsoftverek. Viszont

ami hatalmas plusz ennél a keretrendszerénél, hogy itt Windows alatt is támogatja a hardvergyorsítókat, mint például a GPU-t [20].

Mondhatni mindkettő könyvtár ugyanazt a célt szolgálja csupán az alfejezet alatti bekezdésben leírtak miatt választottam a TensorFlow-t, viszont a projekt végére úgy gondolom lehet jobb lett volna ezzel a szoftverkönyvtárral kezdeni.

3. Adathalmaz készítése

Mint már említettem egy-egy ilyen modell tanításhoz szükségünk van adathalmazokra. Objektumfelismerés esetén ezeket képek fogják alkotni, melyek az objektumot fogják tartalmazni az osztályukkal ellátva. Ebben a fejezetben bemutatom az algoritmus tanításához való adathalmaz gyűjtési, feldolgozási folyamatát. Továbbá kitérek ezeknek az adathalmazoknak a mesterséges bővítési lehetőségére.

3.1. Kézjelek

Az adatok gyűjtéséhez való első lépés az volt, hogy megismerjem, hogy a jelnyelvben, hogyan néznek ki a betűk, hogyan képzik őket. Két fajta jelnyelvet különböztetünk meg amikor az ábécéről beszélünk az egyik a fonomimikai a másik pedig a daktil. Ezek ismertetésével kezdem ezt az alfejezetet majd kitérek arra, hogy melyiket és miért választottam.

A fonomimikai kézjelek a beszédhangok (fonémákat) tükrözik. Ezeket a jeleket szájmozgatással kombinálva használják annak érdekében, hogy a jelelő személy a beszédhez hasonló fonémákat kézjelekkel jelenítsen meg. Ezeket leggyakrabban a szájról olvasás kiegészítéseként használják. A fonomimikai jelek tehát a beszéd fonémáit követik. Magyarországon ez az elterjedtebb.

A daktil kézjelek a kézzel történő betűzést jelentik, amikor minden egyes kézjel egy adott betűt képvisel az adott nyelv ábécéjéből. Ezzel a módszerrel a jelölő személy betűnként ír le egy szót. Abban az esetben használják, ha egy szónak nincsen megfelelő jele a jelnyelvben. Lényegében itt minden egyes betűt egy kézjel jelent. Ezáltal esett a választásom erre az ábécére hiszen így elkerülhető az, hogy egy mozgást kelljen megtanítani a modellünknek, ami viszonylag komplexebb dolog, illetve a száját se kell figyelniük közben.

Az adatokat osztályoznunk kell ebben az esetben aszerint, hogy az egyes képeken melyik objektum található, illetve ez fogja megadni a lehetséges kimeneteket a modellünk számára. A tanítandó algoritmusunk számára ezek az ábécé betűi lesznek egészen A-tól Y-ig. Sajnos a Z betű a daktil kézjelek között is egy mozgást jelentő betű, amit egyelőre kihagytam, hiszen számomra egy alapvető képekkel tanítandó objektumfelismerő neurális hálózat fejlesztése is egy teljesen új terület volt. Így időszükében ezt hozzáadtam a továbbfejlesztési lehetőségek közé.

3.2. Adatok gyűjtése

Az adatok gyűjtését kétféle szempontból közelítettem meg. Első gondolatom az volt, hogy felkeresem a Kaggle nevű oldalt, ahol találhatunk előre elkészített adathalmazokat felcímkézve készen a tanításra. Viszont én úgy gondoltam, hogy a képek gyűjtése és feldolgozása is a dolgozatomhoz tartozik így a saját adathalmaz készítése mellett döntöttem.

Egy jó minőségű adathalmazhoz igen sok és változatos kép szükséges. Ezeknek az internetről való gyűjtése igen hosszú ideig tartott volna, hogy akár 4-5 ember kezét más-más háttérben megtaláljam. Ennek a megoldására jött az az ötlet, hogy minden betűt felveszek egy rövid videón miközben a kezemmel mutatom. Annak érdekében, hogy változatos legyen az adathalmaz megkértem néhány szaktársam és családtagom, hogy készítsenek ők is ilyen videókat. Így lett minden betűről 4-5 videó. Ezzel máris változatosabbá tettem az adathalmazom hiszen minden egyes ember keze más, valakinek nagyobb, valakinek kisebb és így tovább. Emellett a videókon mindenki mögött más háttér és más fényviszony volt. Ezek kulcsfontosságú szempontok lehetnek például a valós időben történő felismerésnél hiszen ott is pontosan ilyen szituációk lesznek.

A videókból képeket a VLC media player pillanatkép készítő lehetőségével készítettem el. Minden egyes betűről nagy számban készültek képek ez az 1. táblázatban látható szám szerint.

1. táblázat: Az adathalmaz betűnkénti darabszámai

Betűk és darabszámaik					
Betű	Darab	Betű	Darab	Betű	Darab
a	215	k	296	u	260
b	197	l	250	v	220
c	248	m	255	w	200
d	210	n	260	x	180
e	250	o	250	y	199
f	259	p	210		
g	227	q	200		
h	250	r	220		
i	250	s	250		
j	174	t	250		

3.3. Annotáció

Szakedolgozatomban igazán hosszadalmas és időigényes munkája ezen a ponton kezdődött. Összesen 5682 képet kellett a gyűjtést követően, amiket egyesével kellett feldolgoznom. Összehasonlításként az ImageNet nevű modell 3.2 millió annotált képet tartalmaz, amit 5247 kategóriára lehet lebontani [7]. Ehhez képest az enyém eltörpül, illetve már itt is jól látható, hogy a transfer learning mennyire is segíti a munkákat hiszen nincs szükségünk milliós nagyságrendű adathalmazra. Az annotáció során jelöljük és címkézzük fel, hogy melyik képen milyen objektum található. Ebben az alfejezetben bemutatam azt, hogy hogyan is történik a képek feldolgozása és milyen szoftverrel dolgoztam. Továbbá bemutatam a mesterséges adathalmaz bővítési folyamatot.

3.3.1. LabelImg

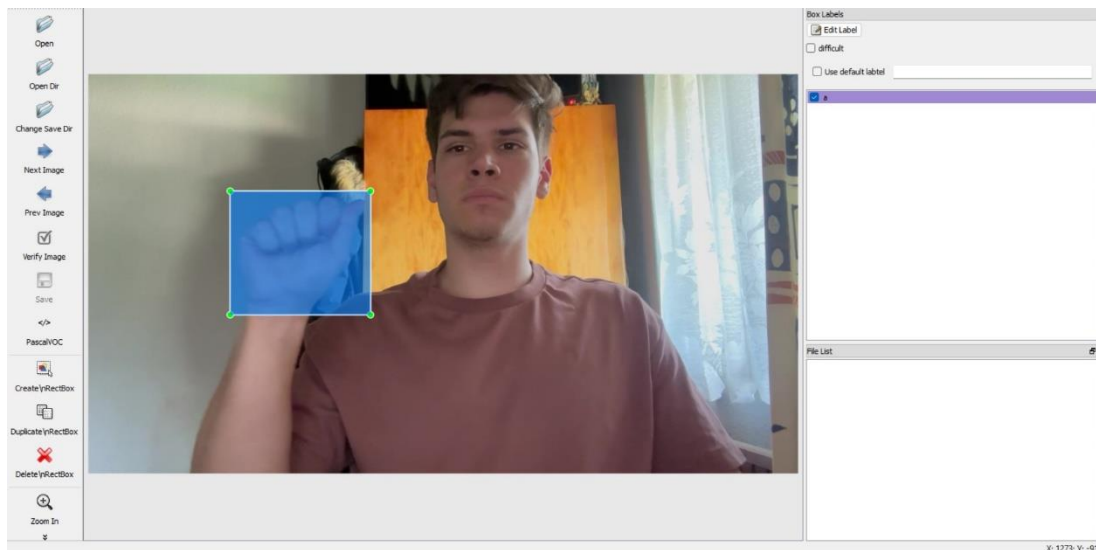
A LabelImg egy nyílt forráskódú szoftver, amit GitHub-ról bárki letölthet. Ez a program lehetőséget nyújt a képek annotációjára. Használata rendkívül egyszerű csupán kiválasztjuk, hogy melyik mappából szeretnénk a képeket betölteni és hova szeretnénk menteni az annotációt tartalmazó fájlt. Én alapvetően minden betűt külön mappába mentettem amikor készítettem a pillanatképeket a videókból ennek csupán az oka, hogy így számomra sokkal átláthatóbb volt az egész folyamat. Emellett könnyebb volt ellenőrizni, hogy kihagytam-e valamelyik képet vagy sem.

Az annotáció során igazából betöltöttem a képeket egyesével és mindegyiken körbe rajzoltam az objektumot ezzel megadva a bounding box elhelyezkedését és ezzel egyhuzamban fel is címkéztem őket. A bounding box az objektum körül lévő téglalap, mellyel jelöljük az objektumot. Ezt reprezentálja az 5. ábra, illetve itt látható a LabelImg felépítése is.

Az annotációkat PascalVOC formátumban mentettem el, ami egy xml fájl. Ez a fájl tartalmazza a bounding box koordinátáját a képeken xmin, ymin, xmax, ymax formátumban, illetve megtalálhatom itt a kép címkéje is. Emellett még más adatok is megtalálhatók itt például a fájl neve, mérete és így tovább.

Kritikus pontja ennek a folyamatnak onnan indult, hogy mivel manuálisan történt minden egyes képfeldolgozása előbb utóbb várható volt, hogy fogok hibázni. Többször volt olyan, hogy rosszul címkéztem fel egy képet, mert bent felejtettem az automatikus címke használatot és nem írtam át a címkét. Ezzel készült jó pár olyan kép, amin nem az

volt, mint ami a címkén szerepelt így dupla munkát kellett végeznem. Szerencsére ez tanítás előtt orvoslásra került.



5. ábra: A LabelImg szoftver felépítése (saját szerkesztésű ábra)

3.3.2. Roboflow

A Roboflow egy olyan weboldal, amely hasznos lehet azok számára, akik objektumfelismerő szoftver szeretnének készíteni. A számomra fontos funkciót teljesen ingyenes elérhetőek, de alapvetően vannak fizetős szolgáltatásai is. Számos hasznos eszközt biztosít az adatok feldolgozásához, annotálásához, augmentációjához, előkészítéséhez, adathalmaz mesterséges bővítéséhez, valamint modellek betanításához és telepítéséhez. Én ezek közül több funkciót is igénybe vettem az adathalmaz további feldolgozására.

Első lépésben feltöltöttem az összes képet erre a Roboflow-ra, ami már itt segítségemre volt, mivel, ha talál olyan képet, ami végül nem lett annotálva akkor azt jelzi és lehetőséget nyújt annak elkészítésére. Ez után beállítottam, hogy az adathalmazom 70% a tanítási adathalmaz legyen, 20% a validációs adathalmaz és a maradék 10% pedig a teszt halmaz legyen. Illetve a feltöltés után a bounding box koordináták megváltoztam x, y, w, h beállításúra.

Ezután megkezdtem az adathalmaz további alakítását. Lehetőségünk van itt beállítani néhány dolgot ezzel is megspórolva néhány sor kódot. Én itt méreteztem át a képeimet 640x640-es méretre fekete szélekkel, így elkerülve a képek széthúzottságát. Erre azért van szükség, mert tanítás közben az ekkora méretű képek is igen sok erőforrást igényelnek nem, hogy az eredeti méretűek. Ezen kívül tettem még rá egy automatikus

kontraszt beállítást ezzel javítva az adathalmaz minőségét, mert így a kezek élei talán jobban kivehetőek az algoritmus számára. Én ezzel letudtam az előkészítési folyamatot, de a Roboflow még bőven kínál erre lehetőségeket.

Ezután igénybe vettem a weboldal által nyújtott augmentációs folyamatot. Az augmentáció során lehetőség van az adathalmazt mesterségen változatosabbá és robusztusabbá tenni. Fontos hozzá tenni, hogy ez csak a tanítási adatokra vonatkozik, ami elosztva 4000 képet foglalt magába. Itt próbáltam olyan lehetőséget választani melyek pozitív irányba tudják változtatni az adathalmazom minőségét. Itt gondolok arra, hogy a szolgáltatás között van olyan opció, hogy kivágás itt a képből kivág egy darab teljesen véletlenszerűen, ami veszélyes lehet és káros képeket generálhat. Mondhatni sok augmentációs lehetőséget biztosít. Én az alábbi funkciókat választottam, ami szerintem még biztonságosan használható ezekre a képekre és javíthatja a valós idejű felismerést:

- Elforgatás -10° és $+10^\circ$ között
- Fényerő -15% és $+15\%$ között
- Mesterséges zaj 1%-ig

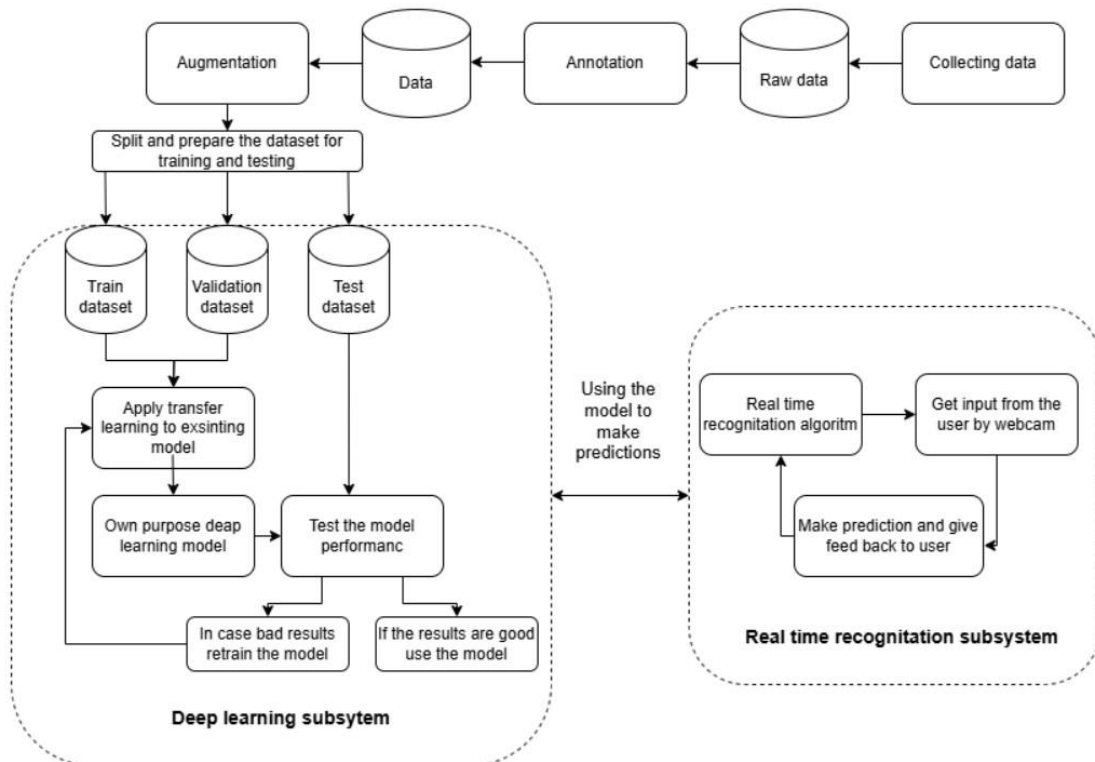
Ezek után kiválaszthattam, hogy mennyire is szeretném növelni a tanítási adathalmazt. Az ingyenes verzióban kétszeres és háromszoros növelésre van lehetőség. Első körben bőven elégnék gondoltam a kétszeres méretet így azt választottam. Ezzel el is készült az adathalmazom, ami így 7966 képpel rendelkezik a tanítási halmazban, ami 2. táblázatban látható szám szerint is. Ez után letöltöttem az adatkészletet az annotációt tartalmazó JSON fájlal együtt, amit a választott COCO formátumban kaptam meg.

2. táblázat: Mesterségesen bővített adathalmaz betűnkénti darabszáma

Betűk és darabszámaik					
Betű	Darab	Betű	Darab	Betű	Darab
a	288	k	418	u	340
b	306	l	368	v	303
c	356	m	334	w	276
d	274	n	378	x	258
e	324	o	351	y	280
f	382	p	278		
g	326	q	234		
h	320	r	286		
i	356	s	358		
j	210	t	340		

4. Kézel felismerő szoftver implementációja

A szoftver fejlesztését három külön osztályba végeztem el. Ezek közül az egyik a modell tanítására szolgál, a másik a modell vizualizációs tesztelésére és a harmadik pedig a valós időben történő felismerést végzi.



6. ábra: A szoftver felépítési terve (saját szerkesztésű ábra)

A szoftver felépítési terve alapvetően a 6. ábrán látható, ahol vizuálisan látható a fejlesztési folyamat. Ebben a fejezetben részletesen bemutatom ezeknek az osztályoknak a fejlesztési folyamatát.

4.1. Modell tanítása

A tanítandó algoritmus az elkészült annotációs JSON fájl és a képek segítségével fogja megtanulni a számunkra szükséges információkat. A COCO (Common Objects in Context) féle JSON fájlok tartalmazzák a képek adatait, mint az objektum lokációja, osztálya és a képek részleteit például fájl név, méret és így tovább. Szóval a tanításhoz való első lépcső ezeknek az adatoknak a betöltése és feldolgozása volt. Alapvetően a Roboflow számunkra külön mappába osztja a tanítási és validációs adathalmazt és mindkettőnek készít annotációs JSON-t. Ez alapvetően nem rossz dolog, viszont ennek

köszönhetően igencsak nagy lenne a kódduplikáció így ezeknek a feldolgozásra függvényeket készítettem. Függvényeknél az egyik fontos dolog, amit megtanultam:

„A függvényeknek egyetlen feladatra kell összpontosítaniuk. Végezzenek pontosan egy dolgot – és ezt tegyék a lehető legjobban.” [8]

Itt alapvetően egy probléma volt, ami a figyelmetlenségem miatt alakult ki. A Roboflow számunkra már előre megkeveri az adatkészletet, amivel én nem számoltam, de ezt hamar javítottam és olyan ciklust készítettem, ami ezt figyelembe veszi és párosítja a képeket. Alapvetően a keverésre nagy szükség van egy modell tanítása során hiszen ezzel is elkerülhető az overfitting. Ha a modell mindig ugyanabban a sorrendben látja az adatokat, akkor előfordulhat, hogy a modell megtanulja az adatok sorrendjét, és nem az adatok belső jellemzőire figyel. Ez növeli a túltanulás kockázatát, ami azt jelenti, hogy a már látott képen jól fog teljesíteni, de olyan adatkészleten, amit sose látott még rosszul fog.

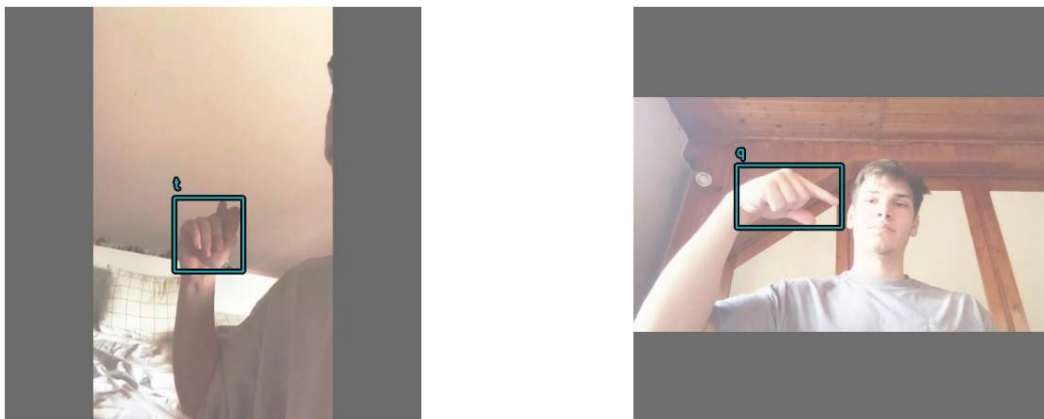
Az adatok betöltése után jöhetett az adatok előkészítése a tanításhoz, ehhez a bounding box-okat, osztályokat és a képek elérési útvonalait tartalmazó változókat TensorFlow alapú konstansokká szükséges alakítani, ami azért fontos, hogy az adatok kompatibilisek legyenek a Dataset API-jával. Ezzel tulajdonképpen a 3 adathalmazból készítünk egy adathalmazt, ami mind 3 adatot tartalmazza. Én biztosra mentem és mind a tanítási mind a validációs adathalmazt újra kevertem. Az így kapott adathalmazok további feldolgozáson estek át és batchekre osztottam és ezáltal elkészült a végleges tanítási és validációs adathalmazom. Amikor batch-ről beszélünk akkor az adatokat csoportosítjuk vagy is megadjuk, hogy egyszerre hány adatot kapjon a modell tanítás során. Nagyobb batch méret felgyorsíthatja a tanulást, illetve a nagyobb méret kevésbé érzékeny a tanulás során lévő ingadozásokra, a kisebb batch mérettel szemben, viszont a kisebb méret jobban tud konvergálni a folyamat alatt. Minél nagyobb a batch méret annál nagyobb az erőforrásigény is. Én 8-as batch méretet választottam az erőforrásomhoz mérten.

Próbáltam tesztelni végig minden egyes lépést, amit csak lehetett gondolok itt arra, hogy megnéztem, hogy a JSON tartalmazza-e az összes osztály címkéjét, jól vannak-e párosítva a képek útvonalai a bounding box-okkal és címkékkel, de az igazán nagy teszt az volt, hogy megnéztem, hogy jól is jeleníti meg a bounding box-okot és címékeket

vizualizáció során. Erre az alábbi Python programozási környezetben megvalósított forráskód részletet alkalmaztam:

```
def visualize_dataset(inputs, value_range, rows, cols, bounding_box_format):
    inputs = next(iter(inputs.take(1)))
    images, bounding_boxes = inputs["images"], inputs["bounding_boxes"]
    visualization.plot_bounding_box_gallery(
        images,
        value_range=value_range,
        rows=rows,
        cols=cols,
        y_true=bounding_boxes,
        scale=5,
        font_scale=0.7,
        bounding_box_format=bounding_box_format,
        class_mapping=class_mapping_train
    )
    plt.show()
```

A KerasCV által nyújtott vizualizációs lehetőséget a matplotlib könyvtár pyplot függvényével kombinálva lehetőségünk nyílik a képek megjelenítésére. Erre azért van szükség, hogy lássuk, hogy sikerült-e jól előkészíteni és feldolgozni a képeket. Hiszen, ha már itt rossz lenne valami akkor nem lenne értelme a kód folytatásának és ezeket a hibákat jobb minél előbb ellenőrizni, javítani. Ennek az eredményét a 7. ábrán láthatjuk, ami igen pozitívnak minősült és ez alapján eddig sikeresen alakultak a dolgok. A fent használt függvény alkalmas a kész modell vizualizációs tesztelésére csupán kisebb átalakítást igényel, de erre majd kitérek a következő alfejezetben.



7. ábra: Annotált kép vizualizációs megjelenítése feldolgozás után (saját szerkesztésű ábra)

Miután megbizonyosodtam arról, hogy sikeres volt az adatok feldolgozása jöhettek a modell tanításához szükséges előkészületek. Mint dolgozatomban elején említettem továbbtanítási módszerrel fogok egy már meglévő modellt a saját célomra áttanítani. A transfer learning segítségével a modell paraméterei már be vannak állítva így nekünk nem kell bajlódniunk az algoritmus mögötti bonyolult matematikával. Erre a ResNet-50 előre tanított modelljét használom fel. Ez a modell már több millió képpel lett tanítva, amelyek több ezer kép kategóriából állnak az ImageNet adatbázis jóvoltából [9]. Ez lesz a háttérret biztosító hálózat (backbone) és a RetinaNet fogja alkotni az objektum detektáló modellt, amely a Resnet-50 által kinyert jellemzőket használja az objektumok felismeréséhez.

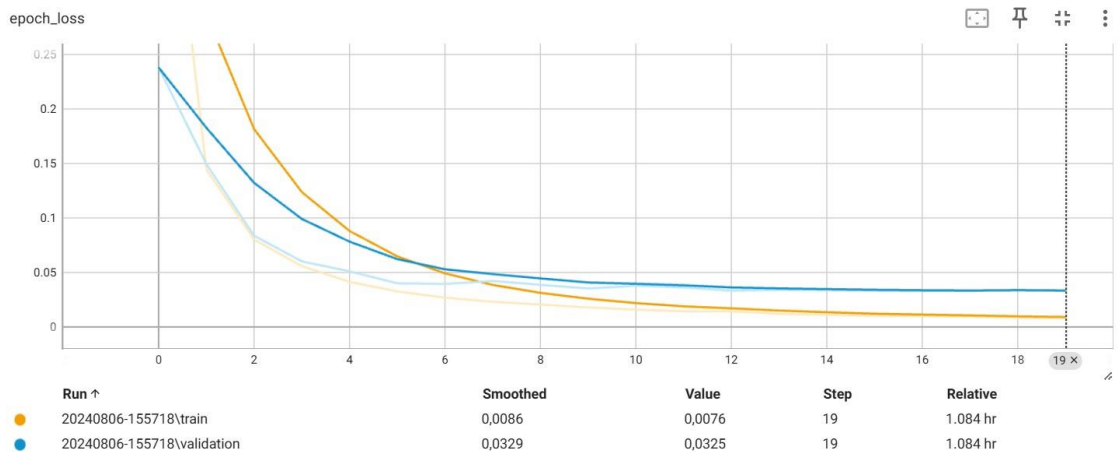
Ezek után már nem sok minden volt hátra a tanítás megkezdéséhez néhány dolgot kellett már csak beállítani köztük az optimalizálót, ami a Stochastic Gradient Descent (SGD) volt. Számítási hatékonysága miatt nagyon gyakran alkalmazzák mély tanítási modellek esetén [10]. Az SGD segíti az algoritmus paramétereinek frissítését minden egyes epoch után. Itt állítottam be továbbá a learning rate-et, momentum-ot és a global clipnorm-ot, melyek mind a tanítást fogják segíteni a maguk módján. Itt talán a kritikus pont a learning rate beállítása, mert a túl kicsi learning rate lassú tanulást eredményezhet míg a túl nagy pedig nem megfelelő konvergálást okozhat [11]. Fontos volt a jó paraméterezés ebben Kirsch Norbi a külső konzulensem volt nagy segítségemre.

Ezek után már csak a modell előkészítése és tanítása volt hátra. Az előkészítés során megadtam a classification loss-t, amely a focal loss veszteségfüggvény alapján fog működni és a box loss-t, ami pedig a smooth L1 loss veszteségfüggvény alapján fog értékelni. Ezek szintén a tanulást segítik, viszont a tesztelés során is fontos információkkal tudnak számunkra szolgálni. A tanításra a lentebb látható Keras fit() függvényt alkalmaztam az alábbi paraméterekkel, amelyet Pythonban megvalósított módon mutatok be:

```
model.fit(  
    train_dataset,  
    validation_data=valid_dataset,  
    epochs=20,  
    callbacks=[tensorboard_callback, checkpoint_callback]  
)
```

Megadtuk számára a tanulási és validációs adathalmazt, illetve azt, hogy hányszor fusson végig a tanító algoritmus az adathalmazokon. Az epoch-ok száma az első próbálkozás

során 40 volt, ami overfittinget eredményezett, mivel egy ponton a validációs halmaz loss értéke nagy ívben elvált a tanítási adathalmaz loss értékétől. Így azt lefelezve már egy sokkal jobb eredményt kaptunk talán lehetett volna még tovább csökkenteni az epoch-ot, mivel így is van egy minimális elválás, viszont mi így is megvoltunk elégedve az eredménnyel.



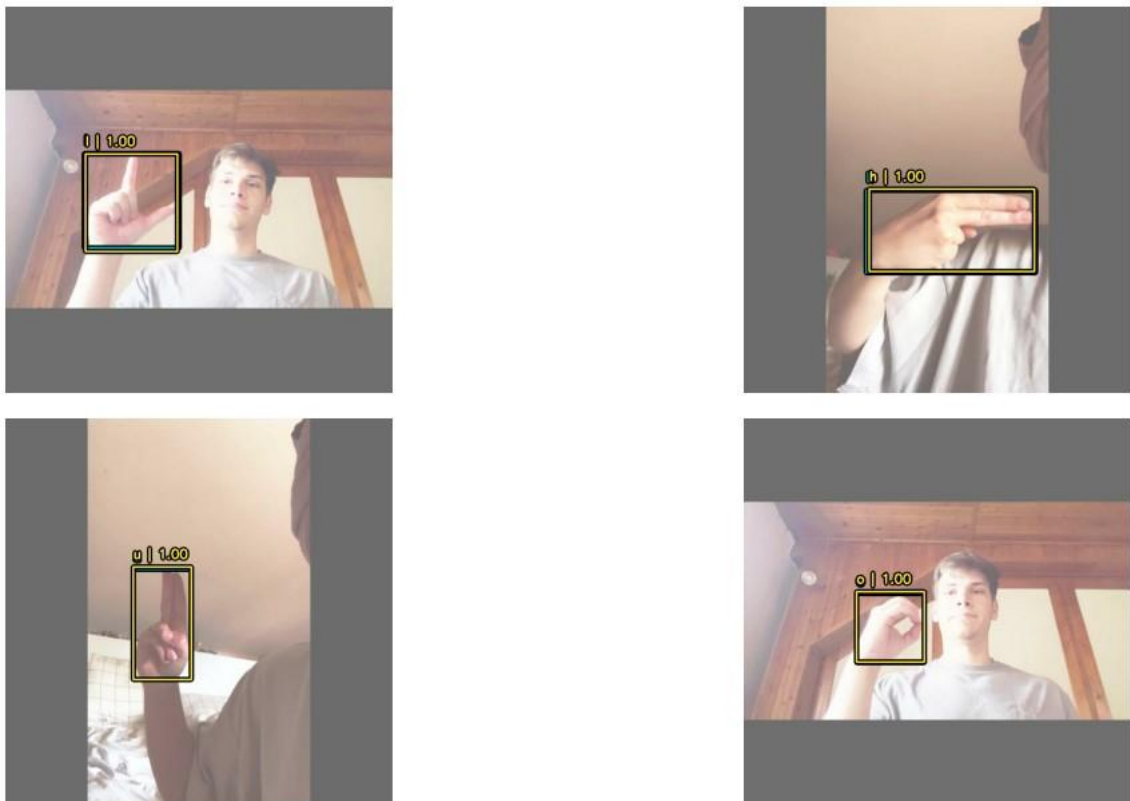
8. ábra: A modell tanulási folyamata (saját szerkesztésű ábra)

A 8. ábrán látható az, hogy az epoch-ok során szépen csökken a loss érték mindkét adathalmazon a narancssárga jelzi a tanítást a kék pedig a validációt. Ezzel könnyen látható és értelmezhető visszajelzést adva arról, hogy a tanítás sikeresen alakul. A tanítás a GPU használatnak köszönhetően megközelítőleg 1 órát vett igénybe körülbelül ez CPU-val közel másfél nap lett volna. Több modell készült, amelyek dolgozatom során több célt fognak szolgálni. Ezt az első modellt, ami az 8-as batch mérettel készült fogom felhasználni a webkamerán keresztül történő felismerésre. A másik modell egy 16-os batch méretű modell, ami ugyanezen az adathalmazon tanult, de érdekes lehet utána járni, hogy ez képes-e mérhető különbséget eredményezni a modellek között.

4.2. Vizualizációs teszt

Ebben az alfejezetben a modell vizualizációs tesztjéről lesz szó. Erre azért volt szükség, hogy tisztában legyek azzal, hogy tényleg jól sikerült-e a tanítás és jó predikciókat jelenít meg. Mérés szintű tesztre majd a 6. fejezetben térek ki ez csak számomra egy gyors visszajelzési céllal készült. Itt is szükséges volt a teszt adathalmaz feldolgozása, de ezt a modell tanítása során már bemutattam és az ott használt függvényekkel ez a folyamat már gyorsan elvégezhető volt.

Az elmenetet keras formátumú modellt betöltöttem és dekódoltam a predikciót. A dekódolás során van egy paraméter a confidence threshold aminek az értéke 0-1 között van. Ezzel adhatjuk meg, hogy a modell milyen érték feletti bounding box-okat jelenítsen meg. Első nekifutásnak 0.5 volt az érték, ami hamar felkerült 0.9-re a modell remek teljesítése miatt. A 9. ábrán látható képeken a sárga bounding box-val jelölt objektumok a modell által jelzett értékek a kék pedig azt mutatja, hogy minek kéne megjelenennie. Ez nem mindenhol látszik az objektumfelismerő modell pontossága miatt.



9. ábra: Modell által jelzett értékek

Erre a vizualizációs megjelenítésre a fentebb használt függvény volt szintén a segítségemre, melyet kisebb mértékben szükséges volt átalakítani, hogy képes legyen megjeleníteni a predikciókat is. Látván, hogy a modell képeken kifejezetten jól, azaz mondhatni 100%-os felismeréssel teljesít bátran vágtam bele a valós időben történő felismerés implementációjának.

4.3. Valós időben történő felismerés

Talán számomra ez jelentette a legnagyobb kihívást több irányból közelítettem meg a célot és ezekből nagyon sok mellékvágányra futott, de végül tiszta fejjel sikerült egy jó megoldást készíteni.

Itt értelemszerűen már nincs szükség az adathalmaz feldolgozására csupán az osztályokra volt szükségem, amit az annotációs JSON fájlból roppant egyszerűen megkaptam. Illetve szükség volt még a modellemre, amit a vizualizációs tesztfájl alapján ugyanúgy töltöttem be itt is. Így meg volt a kész modell és az osztályok. Jöhetett a valós időben történő felismerésnek a megvalósítása.

A webkamera használatára az OpenCV könyvtárat alkalmaztam. Ez egy nyílt forráskódú szoftverkönyvtár kép- és videóelemzéshez, amelyet eredetileg több, mint egy évtizede vezetett be az Intel. Gyakran használják gépi látási feladatok megvalósításához [12]. Előnye, hogy könnyen integrálható TensorFlow-val és Pytorch-val.

A fejlesztés során felmerült problémákra a következő fejezetben szeretnék kitérni, így itt csupán a probléma megoldását és a szoftver működését szeretném bemutatni. Futtatás közben a kamera működést egy roppant egyszerű while végtelen ciklus teszi lehetővé, amiből a Q betűvel tudunk kilépni és ezen belül történik a felismerés végrehajtása. Ennek a ciklusnak a megvalósítása látható a lentebbi nyelvfüggetlen pszeudokód formátumban:

Algorithm 1 Capture frame from video and make prediction for it

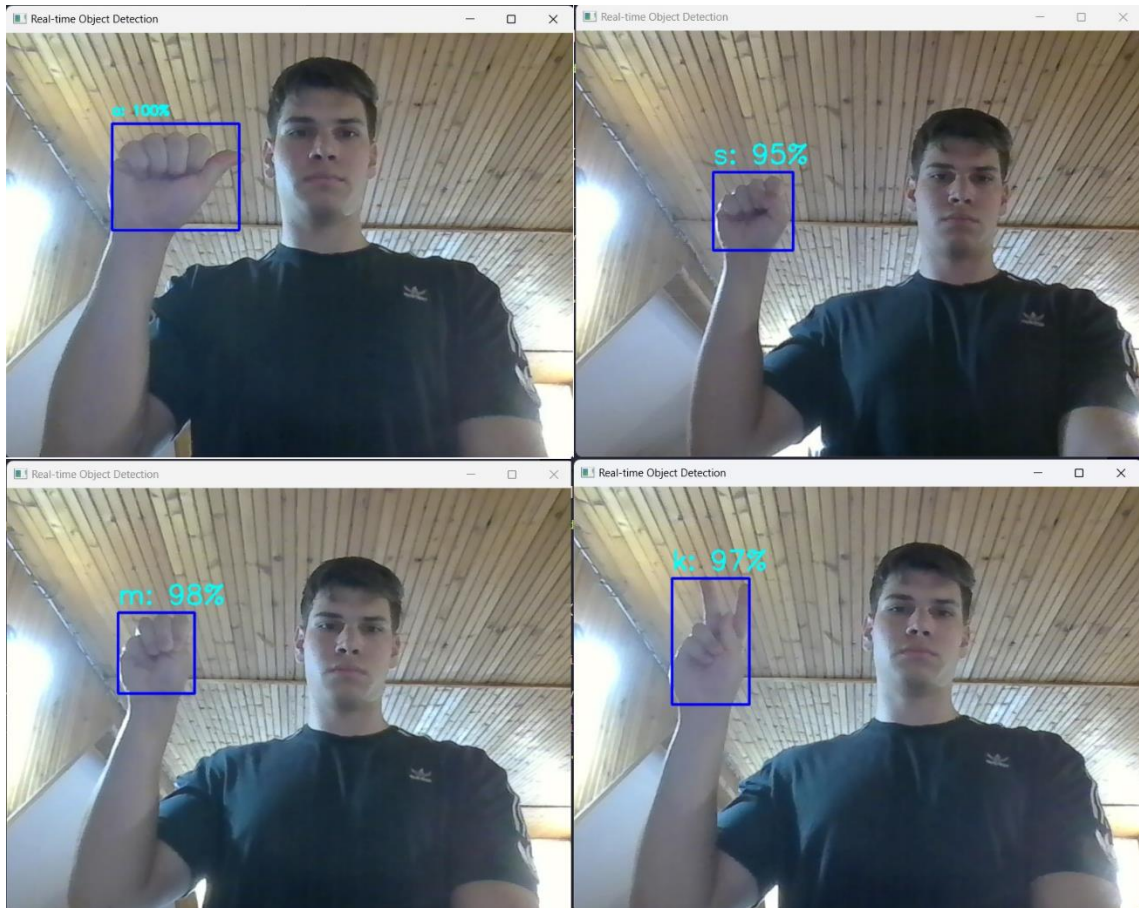
```
1: WHILE true
2:   ret, frame ← capture frame from the webcam
3:   image ← apply preprocess_frame on frame
4:   prediction ← make prediction with loaded_model on image
5:   boxes ← extract boxes from prediction
6:   confidence ← extract confidence from prediction
7:   classes ← extract classes from prediction
8:   num_detections ← extract num_detections from prediction and convert it to
integer
9:   FOR pred IN num_detections
10:    confidence ← get the confidence score for the current object
11:    x, y, w, h ← get the box coordinates for the current object
12:    x1, y1, x2, y2 ← convert x, y, x+w, y+h to integer
13:    class_id ← get the class id for the current object and convert it to integer
14:    class_name ← extract class name from classMapping with class_id if there is no
match then display "Unknown"
15:    DRAW rectangle on frame from (x1, y1) to (x2, y2) with color (255, 0, 0) and
thickness 2
16:    label ← class_name + ": " + confidence * 100 as integer + "%"
17:    DRAW label on frame at (x1, y1 - 10) with font size 1, color (255, 255, 0), and
thickness 2
18:  END FOR
19:  DISPLAY window showing frame with title "Real-time Object Detection"
20:  IF cv2.waitKey(1) is equal to ASCII value of "q"
```

21: **BREAK**
22: **END WHILE**

A modellem már említett 640x640-es méretű képen lett tanítva így a webkamerából kinyert pillanatképeket is ekkora méretre kellett konvertálni. Ez mondhatni egy nagyon kicsi, de fontos lépés, mert enélkül nem nagyon tudnék dolgozni a webkamera által kapott pillanatképpel. Erre a lent látható Python programozási nyelven megvalósított függvény volt segítségemre, ahol a kapott pillanatképet tensor-rá alakítom, mert a TensorFlow legtöbb funkciója tensor-okat vár bemenetként és a KerasCV segítségével átméretezem majd az így kapott képet visszaadom:

```
def preprocess_frame(frame):  
    image = tf.convert_to_tensor(frame)  
    resizing = keras_cv.layers.Resizing(640, 640, bounding_box_format="xywh",  
pad_to_aspect_ratio=True)  
    image = resizing(image[tf.newaxis, ...])  
    return image
```

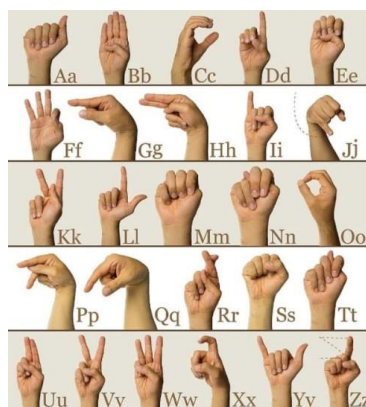
A cikluson belül alkalmazzuk a függvényt a webkamera által kinyert pillanatkép változójára és ezután ezt átadom a modellnek, egy előrejelzésre, hogy az algoritmus szerint melyik betű szerepel a képen. A cikluson belül az OpenCV vizualizációs függvényeinek segítségével a kamerán látható kéz köré megjelenítem a bounding box-ot, az előrejelzést a betű értékével és az előrejelzés pontosságával ezzel majdnem, hogy csak azonnali visszajelzést adva a felhasználónak. A szoftver másodpercenként nyeri ki a képeket a webkamera futása közben és dolgozza fel a modell segítségével, így folyton frissíti a felhasználó számára az előrejelzést. Videó részletet nem tudok mellékelni a szakdolgozatomban így a 10. ábrán látható pillanatképek segítségével próbálom szemléltetni, hogy ez hogyan is nézne ki. Itt is bátran mertem alkalmazni a 0.7-es confidence thresholdot az első futás után látván, hogy jól teljesít a modell.



10. ábra: Valós idejű pillanatképek a szoftver futása közben (saját szerkesztésű ábra)

Próbáltam a 10. ábrán olyan kézjeleket választani, amelyek jelentősen hasonlóak egymáshoz, ilyen az A, az S és az M betű. Ezzel bemutatva azt, hogy a modell jól képes megkülönböztetni hasonló jeleket. A 11. ábrán láthatóak a kézjelek, ezzel igyekszem bizonyítani, hogy valósak az elért eredmények. Emellett itt van a K betű, ahol a kéztartásban a hüvelykujj jelentősen elveszik még is sikerül az algoritmusnak felismernie közel 100 százalékosan. Bizonyos sebességig képes követni a kezét és folyton megjelenítve az előrejelzést, de egy adott mozgatási sebesség után elveszti vagy nem tudja felismerni a gyors helyzetváltás miatt. Viszont ahogy az objektum megáll vagy lassabban mozog képes újra felismerni azokat.

Ezzel a szakdolgozatom szoftveres része a célig kitűzött állapotig elkészült. Ezek után továbbfejlesztések, optimalizálások várnak rá, melyeket egy későbbi fejezetben mutatok be, hogy mik lehetnének azok. Ezekkel akár egy sokkal jobb felhasználói élményt nyújtva, melyeket akár vállalati környezetben alkalmazható célokra is fel lehetne használni.



11. ábra: Kézjelek [18]

Ilyen célok esetében az informatikai rendszerek minőségbiztosítása és auditálása fontos szerepet játszik a szoftverek megbízhatóságában és biztonságában. Az auditálás során a rendszerek biztonságát, hatékonyságát és megfelelőségét vizsgálják, így biztosítva, hogy az adatok integritása és megbízhatósága megfelelően védett legyen. Az informatikai audit segíthet azonosítani a rendszerek gyengeségeit és az adatbiztonsági kockázatait, amelyek javítását követően növelhető a rendszerek teljesítménye, csökkentve a veszteségeket és növelve a rendszerek megbízhatóságát. Az informatikai rendszerek auditálása során a szabványok és irányelvek kulcsfontosságú szerepet játszanak a folyamatok átláthatóságának és ellenőrizhetőségének biztosításában. Ezek a szabványok meghatározzák az auditálási folyamat alapelveit és követelményeit, biztosítva, hogy az auditok eredményei megbízhatók és összehasonlíthatóak legyenek. Figyelembe véve az informatikai auditálási szabványokat nemcsak a funkcionalitást és biztonságot javíthatjuk ezekkel, hanem használatuk hozzájárulhat a vállalati célok hatékonyabb támogatásához is. A minőségbiztosítás továbbá azt is elősegíti, hogy a szoftverek fejlesztése és implementálása során a felhasználói igények maximálisan kielégítésre kerüljenek a hiányosságok feltárásával, ami az egyik legfontosabb szempont, ha megrendelésre készítünk szoftvereket. Ezek a folyamatok biztosítják, hogy a szoftverek megfeleljenek a jogi és szabványos elvárásoknak, ami különösen fontos a vállalati célok elérésében.

5. Fejlesztési során felmerült problémák

A szoftver implementációja során felmerültek olyan problémák, amelyek nem voltak előre várhatóak gondolok itt verzió inkompatibilitásra, kisebb fejlesztési figyelmetlenségekre és a TensorFlow-val való nehézségekre. Ebben a fejezetben ezeket a hibákat mutatom be és azok a sikeres megoldásait.

5.1. GPU használata

Alapvetően TensorFlow-t használva Linux operációsrendszer alatt a GPU használatának semmi akadályja nincs a megfelelő segédsoftverek telepítése mellett. Viszont egy nagyon fontos dolog a videokártya használata. Az én laptopomban egy RTX 3060 6GB-os videokártya van ezzel alapvetően 1 óra az algoritmusom tanítása körülbelül viszont a CPU-t használva ez sokkal hosszabb folyamatnak ígérkezett volna. Egész pontosan 1 epoch lett volna 2,5 óra, ami 20 epoch miatt roppant időigényes. A GPU használata sajnos Windows rendszer alatt közel sem egyszerű bizonyos verzió váltás óta. Egészen pontosan, ha nem 2.10-es verziót használunk TensorFlow-ra akkor nem lehet megvalósítani a GPU használatot.

5.1.1. Verzió váltás

Első próbálkozásként megpróbáltam a verziókat visszaállítani. Én a 2.17-es TensorFlow-t kezdtem el használni a 3.4.1-es Keras-val és a 0.9.0 KerasCV-vel, illetve 3.12.4-es Python verzióval. Miután visszaváltottam 2.10-es TensorFlow-ra minden mást is szükséges volt más verzióban telepíteni. Ez egészen a KerasCV-ig nem is okozott nagyobb problémát, ahol viszonylag sokat kellett visszamenni, de cserébe lehetett tanítani a GPU-t használva. De a KerasCV korábbi verziója miatt le kellett volna mondanom néhány függvényről, ami akkor még nem volt jelen a könyvtárban. Ezáltal inkább lemondtam erről a megoldásról és egy másik lehetőséget kerestem, ahol nincs szükség a verziókkal való harcra. Viszont ez a próbálkozás jó volt arra, hogy kiderüljön az, hogy lehet használni a GPU-t Windows-on, de cserébe le kell mondani néhány funkcióról, amit az újabb verziók nyújtanak.

5.1.2. WSL

A WSL-vel való megoldást a külső konzulensem ajánlotta, hogy ezzel is sikerülhet megoldani a gondomat. A WSL (Windows Subsystem for Linux) lehetővé teszi, hogy külön virtuális gép nélkül Linux alapú rendszert tudjunk futtatni.

A WSL telepítése után nincs más dolgunk, mint elindítani a gépünkön és telepíteni a szükséges segédsoftvereket. Erre rengeteg útmutató van az interneten én is ezek alapján jártam mondhatni majdnem sikerrel. A WSL-ből indított Visual Studio Code-ban hiába volt minden rendben a telepítések során nem akart futni a kód. Mire nagy nehezen rájöttünk, hogy szükséges egy WSL bővítményt telepíteni a VS Code-ba.

Ez után egy következő hibába ütköztem, ami egy váratlan Out Of Memory volt, ami röviden annyit jelent, hogy túl gyenge a GPU-m a tanításhoz. Ezért végül a képek méretét csökkentettem 320x320-asra. Viszont volt egy másik megoldás is, amit a munkatársam tett lehetővé. Még pedig felajánlotta, hogy ha továbbítom neki a forráskódom akkor Ő lefuttatja az otthoni gépén a kódom és elkészíti a 640x640-es képpel a modellem. Így született két algoritmus, ami más-más képméreten tanult. A dolgozatomban én a 640x640-es modellt használtam a valós idejű felismeréshez. Ezzel sikerült megoldani az erőforrás által okozott problémát.

Tanítás során még egy modell kiterjesztési hiba történt, aminek köszönhetően előlről kellett kezdeni az algoritmus tanítását. Ez annak volt köszönhető, hogy én a modell h5 formátumban szerettem volna menteni, ami, mint kiderült már idejét múlt és ezért nem mentette el az algoritmust a tanulási ciklus után. Ezért újra indítottam ezúttal keras kiterjesztésben, ami már megfelelőnek minősült.

5.2. Implementáció alatt felmerült hibák

A fejlesztés során természetesen elkerülhetetlen az, hogy kisebb nagyobb hibákba, problémákba ütközzünk. Ez ebben az esetben sem volt más, hogy voltak olyan szituációk, amiket gyorsan tudtam javítani, de voltak olyanok, amikhez több idő volt szükséges. Alapvetően rengeteg segítséget találunk a felhasznált keretrendszerek dokumentációiban, de ettől függetlenül vannak olyan esetek, amikre nem készültem fel eléggé vagy figyelmetlen voltam. Ebben az alfejezetben ezekre fogok kitérni.

Korábbi fejezetben már említettem kisebb, gyorsan orvosolható problémákat és azoknak a megoldásait, de ide sorolnám még azt is akár, hogy annak ellenére, hogy a Roboflow által generált adatkészlet már 640x640-es méretű képeket nyújt számunkra még is szükség volt a képek átméretezésére a kódon belül is. Ezt azzal lehet magyarázni, hogy hiába vannak már ezen a méreten az adathalmaz képei a formátumuk vagy adatszerkezetük nem feltétlen egyezik az elvártal. A TensorFlow `resize()` függvénye nem csak a méretről gondoskodik, hanem a megfelelő dimenzióról és formátumról, ami lehet

például lebegőpontos tensor is. Emellett tömördek kisebb problémák merültek fel, mint például rossz bounding box formátum megadása, további verziókkal való problémák és így tovább, de ezeket nem tartom olyan érdekesnek, hogy tovább taglaljam.

A fő kihívást a valós idejű felismerés jelentette, ahol voltak már kiemelést érdemlő esetek. Alapvetően a modell tanítása után nem WSL alatt folytattam a fejlesztést, hanem rendszeren Windows alatt. A szoftver első futása után rájöttem, hogy még se tudok a WSL-től elbúcsúzni, mert szintén erőforráshiányban szenvedtem. Konkrétan percenként frissült a videokamera képe, ami így használhatatlanná tette a szoftvert és azt gondoltam, hogy a lassú felismerés miatt nem jelennek meg az előrejelzések. Tudtam, hogy ez a későbbiekben sem lesz jó megoldás hiszen előnyös lenne, ha kisebb erőforrásokkal is képes lenne futni, de tesztelni szerettem volna. Így visszaváltottam WSL-re, ami szintén hamar kudarcba dőlt. Alapvetően, ha OpenCV-vel szeretnénk használni a videokamerát akkor annak a kiválasztását 0, 1, 2, 3 vagy 4-es számmal tehetjük meg vagy hasonló módon a következő elérési útvonallal: /dev/video0. Szerencsére egyik alapján se ismerte fel WSL alatt az OpenCV az alap integrált videokamerát. Hosszas kutakodás és próbálkozás után be kellett látnom, hogy ez nem fog működni WSL alatt. Így most már muszáj voltam javítani a kódom, hogy kisebb igényekkel képes legyen futni Windows-on.

Pár nap után újra neki láttam a szoftver fejlesztésének és más szemszögből közelítettem meg a dolgot. Sokkal több megoldási módot kerestem hasonló problémákra akár Pytorch és YOLO modellek esetében csak, hogy lássam kik milyen módon oldják meg. Sok olyan hibát vétettem, amik magukért beszélnek. Alapul vettem a már meglévő vizualizációs tesztre szolgáló osztályom és egyből szemet szűrt egy probléma. A modellt ugyan betöltöttem viszont a predikciót már nem dekódoltam. Ennek a megoldása az alábbi forráskód részletben látható, melyet Python környezetben mutatok be:

```
loaded_model.prediction_decoder = keras_cv.layers.MultiClassNonMaxSuppression(  
    bounding_box_format="xywh",  
    from_logits=True,  
    iou_threshold=0.5,  
    confidence_threshold=0.7,  
)
```

Ezzel már alapvetően javíthatunk a modell teljesítményén, hiszen a confidence thresholdnak köszönhetően szűrjük a bounding box-ok mennyiségét így a nem kívánatos

előrejelzéseket csökkentve, melyek akár téves előrejelzések, de kicsi százalékkal megjelenhetnek, illetve itt adjuk meg a bounding box formátumot is.

További hibát okozhatott, hogy nem megfelelően készítettem elő a kinyert képeket. Alapvetően erre a célra én az OpenCV függvényeit alkalmaztam első próbálkozásként, amit, mint magától értetődően nem fogja a bemenetet tensor-rá alakítani és az átméretezést is torzítva teszi meg. A 4.3. fejezetben lévő forráskódnak az alkalmazása szolgált ennek a javítására, ami tensor-rá alakítja és a pad to aspect ratio paraméternek köszönhetően nem torzítja a kinyert képeket.

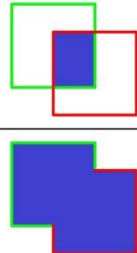
Összeségében ezek voltak a fontosabb változtatások a két szoftververzió között, melyeknek köszönhetően jelentősen optimálisabbá vált a kód és használható módon fut Windows környezetben.

6. Modell tesztelése

Alapvetően a betanított algoritmusomat már teszteltem vizuálisan, viszont ez nem mondható igazi tesztelésnek. Vannak olyan módszerek, melyekkel számszerűsíthetőek a modellek teljesítménye és ez alapján valóban értékelhetőek és összehasonlíthatóak egymással. Ebben a fejezetben ezekre a metrikai mérésekre fogok kitérni, amelyeket gyakran használnak objektumfelismerési algoritmusok esetében. Ezeket a méréseket elvégzem azon a modellen, ami a valós időben történő felismerés mögött van, illetve készítettem egy nagyobb batch méretű modellt, hogy bemutathassam, hogy ezek a kisebb változtatások is milyen hatással lehetnek az elvégzett metrikai mérésekre.

6.1. Teljes modell tesztelése

Objektumfelismerő modellek esetében az alkalmazott metrikák azt mérik, hogy a predikciós bounding box mennyire áll közel a valódi bounding box-hoz (ground-truth bounding box). Ezt a mérést minden osztályra külön-külön végzik el az előre jelzett és a ground-truth területének a metszetéből kiszámítva. [14]

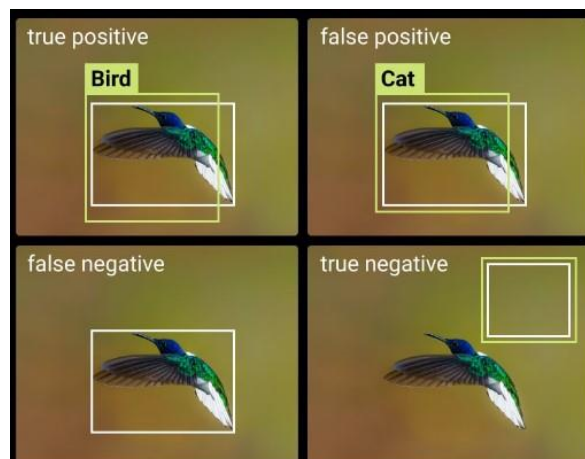
$$\text{IOU} = \frac{\text{area of overlap}}{\text{area of union}} = \frac{\text{IOU}}{\text{IOU}}$$


12. ábra: IOU illusztrációja [14]

Akkor beszélünk tökéletes egyezésről, ha a predikciós és a valódi bounding box területe teljesen lefedik egymást. Ezt a mérést az intersection over union (IOU) segítségével tudjuk elvégezni, amely a Jaccard-indexen, két adathalmaz hasonlósági együtthatóján alapuló mérés. Ennek az illusztrációja látható a 12. ábrán. A mérés akkor minősül tökéletesnek mikor az IOU értéke 1 és akkor mondható értéktelennek, ha az értéke 0 lesz ilyenkor a két bounding box területe semmilyen szinten nem metszi egymást. Az IOU küszöbérték beállításával korlátozhatjuk az észlelés helyesnek vagy helytelennek minősítését. Az egyhez közelebbi küszöbérték szigorúbb köszönhetően annak, hogy így jobb minőségű felismerést igényel a modellünk míg a nullához közelebbi, de nem 0 érték elnézőbbnek minősül tekintve, hogy a kis átfedéseket is figyelembe veszi. A leggyakrabban használt küszöbértékek 50% és 75% közöttiek. [14]

A pontosság (precision) és visszahívás (recall) szintén metrikai mérések, amelyek már a megadott IOU küszöbérték alapján fognak számunkra visszajelzést adni. Ahhoz, hogy ezeket kitudjuk számítani minden detektált bounding box-ot osztályoznunk kell, ami a 13. ábrán látható illusztráció alapján történik. Ezeknek a definíciója található az alábbiakban:

- True Positive (TP): Az objektumot pontosan lokalizálja és ismeri fel az objektum detektáló modell [16],
- False Positive (FP): Olyan objektum észlelése, ami nem szerepel a ground-truth-ban [16].
- False Negative (FN): Nem képes felismerni a ground-truth-ot, ilyenkor a modell nem azonosítja az objektumot [16].
- True Negative (TN): Ezt nem használják objektumfelismerésben, mert az objektum hiányának pontos megerősítésére szolgál [16].



13. ábra: Bounding box osztályozásának illusztrációja. [16]

A pontosság a helyes pozitív előrejelzések százalékos aránya, ami csak a releváns objektumokat azonosítja. A visszahívás pedig a helyes pozitív előrejelzések százalékos aránya az összes megadott ground-truth között. Ez a modell azon képessége, hogy megtalálja az összes releváns ground-truth bounding box-ot. Ezeknek a kiszámítására szolgáló képletek a 14. ábrán találhatóak meg ahol P a pontosság (precision) és R a visszahívás (recall). [14-15], [43]

$$P = \frac{TP}{TP + FP} = \frac{TP}{\text{all detections}},$$

$$R = \frac{TP}{TP + FN} = \frac{TP}{\text{all ground truths}}.$$

14. ábra: Precision (P) és recall (R) képlete [15]

Az átlagos pontosság (AP), egy olyan metrikai mérés, ami egy adott osztályhoz tartozó objektumok felismerésének a pontosságát méri [23]. Tehát ezzel mérhetjük a modellünk teljesítményét osztályonként. Az AP számítása során a Precision x Recall görbe alatti területet integráljuk, így egy számmal mérhetővé válik a pontosság és a visszahívás közötti teljesítmény különböző küszöbértékeknél.

Amíg az átlagos pontosság minden egyes osztálynak kiszámítja az átlagát addig az átlagos pontosság átlaga (mAP) az összes osztály átlagának a kiszámítására szolgál. Ez hasznos lehet olyankor amikor olyan adathalmazunk van, ami sok osztállyal rendelkezik. Az mAP matematikai kiszámítása az alábbi képlet alapján történik (1): [15]

$$mAP = \frac{1}{N} \sum_{i=1}^N \frac{TP_i}{TP_i + FP_i} \quad (1)$$

Az F1 pontszám a pontosság és visszahívás harmonikus átlaga, ami korlátozva van a [0, 1] intervallumon. 0 akkor amikor a pontosság és visszahívás is 0 és 1 amikor mindkettő metrika 1. [15]

Léteznek még további metrikák objektumfelismerő modellek tesztelésére, mint az átlagos visszahívás (AR), átlagos visszahívás átlaga (mAR) és így tovább, de én ezeket már nem alkalmaztam a dolgozatom során csupán csak a fent említett metrikákat használtam fel.

Amikor felosztottam az adathalmazom, készült egy teszt adathalmaz is, ami 450 képből áll. A teszt adathalmaz lényege, hogy olyan képeket tartalmaz az adathalmaz, amelyeket a modell a tanítás során még nem látott így tudjuk értékelni a modell valódi képességeit. Alapvetően a valós idejű felismerés alatt a modell került olyan szituációba, hogy olyan háttérrel látott és olyan emberi kezet, ami egyik tanítási képen se szerepelt, de jól teljesített így is. Viszont ez nem mérhető számokkal és nem teljesen mondható tesztelésnek többek között ezért is szükségesek a metrikai mérések elvégzése. A metrikákkal való megismerkedés után neki láttam egy olyan Python kód elkészítésének, aminek a segítségével tesztelni tudom a modellem. Hosszabb próbálkozás után egy külső

kutatói partner segítségével más szemszögből közelítettem meg a feladatot, mint ami az eredeti felállás volt. Az teszt adathalmaz annotációs JSON adataiból és a predikciós értékekből készítettem egy csv fájlt, amelybe a következők szerint mentettem el az adatokat: elérési útvonal, valós bounding box, valós osztály id, valós osztály, predikciós bounding box, predikciós osztály id, predikciós osztály és a predikció magabiztosságát. Ez a fájl a teljes adathalmaz adatait tartalmazza és ennek köszönhetően jelentősen egyszerűbb az adatok feldolgozása. Amikor az első próbálkozásaim történtek a metrikák kiszámításával akkor én azt úgy kezdtem el, hogy valós időben olvasom be az adatokat és készítem el a predikciókat. Ezzel az volt a probléma, hogy minden egyes futtatásnál több percig kellett várnom mire feldolgozta az adatokat. Így viszont, hogy el vannak mentve egy csv fájlba sokkal gyorsabban feldolgozható. Igaz, hogy a mentés itt is igénybe vesz pár percet, de ha kész van a fájl már csak be kell olvasnunk és végezhetjük rajta a számításokat. A metrikák kiszámítási módja az alábbi pszeudokód formátumban látható lentebb:

Algorithm 2 Calculate metrics for classes

```

1: SET classes TO unique values in "gt_class_id" column of df
2: SET metrics TO an empty dictionary
3: FOR cls IN classes
4:   cls_df ← rows in df where "gt_class_id" equals cls
5:   total_gt ← number of rows in cls_df
6:   preds_cls ← rows in df where "pred_class_id" equals cls
7:   tp ← count of rows in preds_cls where "Detection" is "TP"
8:   fp ← count of rows in preds_cls where "Detection" is "FP"
9:   fn ← total_gt - tp
10:  precision ← tp / (tp + fp) IF (tp + fp) is not 0 ELSE 0
11:  recall ← tp / (tp + fn) IF (tp + fn) is not 0 ELSE 0
12:  f1_score ← (2 * precision * recall) / (precision + recall) IF (precision +
recall) is not 0 ELSE 0
13:  y_true ← array of 1 for "TP" and 0 for "FP" values in "Detection" column of
preds_cls
14:  y_scores ← values in "pred_confidence" column of preds_cls
15:  IF length of y_true > 0
16:    ap ← average_precision_score(y_true, y_scores)
17:  ELSE
18:    ap ← 0.0
19:  metrics[cls] ← dictionary with:
20:    "Precision" ← precision,
21:    "Recall" ← recall,
22:    "F1 Score" ← f1_score,
23:    "Average Precision" ← ap,
24:    "True Positives" ← tp,
25:    "False Positives" ← fp,
26:    "False Negatives" ← fn
27: END FOR
28: mAP ← mean of "Average Precision" values in metrics
29: PRINT "Mean Average Precision (mAP):", mAP
30: mean_iou ← mean of "IoU" values in df
31: PRINT "Mean IoU:", mean_iou
32: FOR cls, m IN metrics
33:   PRINT "Class cls - Letter", uppercase letter from class_id_to_str[cls]
34:   PRINT "Precision:", m["Precision"]
35:   PRINT "Recall:", m["Recall"]

```

```

36: PRINT "F1 Score:", m["F1 Score"]
37: PRINT "Average Precision:", m["Average Precision"]
38: PRINT "True Positives:", m["True Positives"]
39: PRINT "False Positives:", m["False Positives"]
40: PRINT "False Negatives:", m["False Negatives"]
41: END FOR

```

A lényegre térve a 3. táblázatban látható a metrikai mérések eredménye a következők szerint: pontosság (P), visszahívás (R), F1 pontszám (F1), átlagos pontosság (AP), True Positive (TP), False Positive (FP) és False Negative (FN). A táblázatban szereplő adatok 0.5 küszöbértékű IOU beállítással lettek kiszámítva, azaz, ahol a valós és a predikciós bounding box metszete nem haladja meg az 50%-ot akkor azt az előrejelzést False Positiv értéknek számítjuk.

3. táblázat: Modell eredményei 0.5 IOU küszöbértékkel

	P	R	F1	AP	TP	FP	FN
A	1	1	1	1	17	0	0
B	1	1	1	1	13	0	0
C	1	1	1	1	15	0	0
D	1	1	1	1	23	0	0
E	0.96	1	0.98	1	23	1	0
F	1	1	1	1	15	0	0
G	1	1	1	1	14	0	0
H	1	1	1	1	22	0	0
I	1	1	1	1	14	0	0
J	1	1	1	1	14	0	0
K	1	1	1	1	25	0	0
L	1	1	1	1	15	0	0
M	0.96	0.96	0.96	0.99	24	1	1
N	1	1	1	1	20	0	0
O	1	1	1	1	17	0	0
P	1	1	1	1	16	0	0
Q	1	1	1	1	14	0	0
R	1	1	1	1	18	0	0
S	1	1	1	1	16	0	0
T	1	0.96	0.98	1	23	0	1
U	1	1	1	1	25	0	0
V	1	1	1	1	24	0	0
W	1	1	1	1	14	0	0
X	1	1	1	1	13	0	0
Y	1	1	1	1	13	0	0

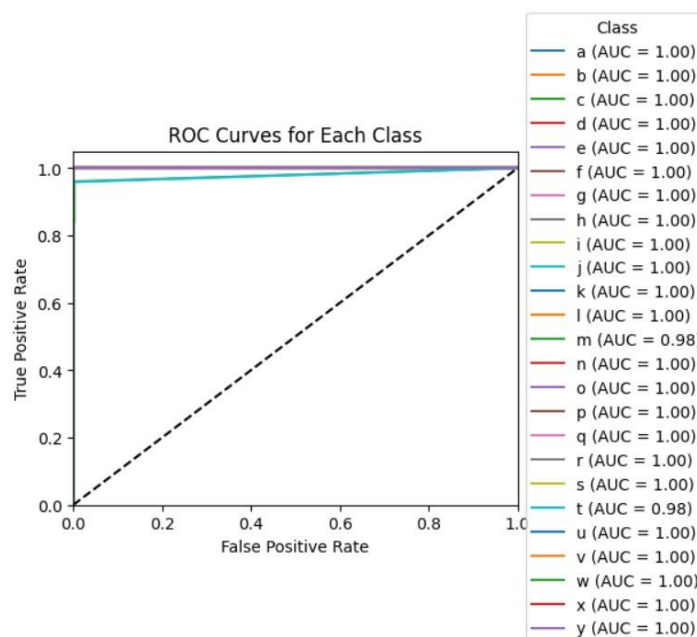
Láthatjuk, hogy a modell roppant jól teljesít a teszt adathalmaz képein is. A TP, FP és FN oszlopok alatt láthatjuk azt, hogy az adathalmazban szereplő képekből mennyi

az a kép az egy adott osztályon belül, ami ezeknek felel meg. A legtöbb esetben az összes TP-nek minősül 1-1 kivétellel. Ezek az eredmények már pozitívnak mondhatóak hiszen ebből is tudunk arra következtetni, hogy a modell jól detektál 0.5-ös IOU alapján is. Alapvetően három kritikus betűnünk van ezek a 3. táblázat alapján az E, M és T betű. Viszont ezek se mondhatóak rossz eredménynek hiszen még csak 90% alá se ment egyik metrikájuk sem. Ezekből a betűkből is csak az M és T objektumnál találhatunk FN minősítésű képet, ami azt jelenti, hogy nem volt képes a modellünk detektálni az objektumot azon az adott képen. Ellenben mind a három betű esetében előfordul hamis detektálás, de mivel ezek eléggé ritkán és kis számban fordulnak elő így nem mondhatóak hatalmas problémának. Nyilván ezek a téves vagy abszolút fel nem ismerések eredményei meglátszanak a metrikák eredményein, de ennek ellenére sincs egyik mérés sem 90% alatt.

Alapvetően tesztelhetném a modellemet esetleg 0.75 IOU küszöbértékkel, de az átlagos IOU értéke 0.92, azaz valószínűleg egyik detektálásnak sincs az IOU értéke 0.75 alatt. Azaz, amikor a modell detektálja az objektumot szinte teljesen pontosan illeszkedik a predikciós bounding box a valós bounding box-hoz. De mivel csak egy szám átírása ennek a tesztelése így végig futtatam a kódot 0.75 IOU értékkel. A vártaknak megfelelően semmiféle eltérés nem volt az IOU 0.5 küszöbértékhez képest. Az osztályokra mért átlag pontosságok átlaga pedig 0.99 értékű lett, ami egy kifejezetten nagyszerű teljesítményre utal. Ez már kezdett túlságosan is szépnek tűnni így felmerült bennem az overfitting gyanúja. Bár a 8. ábrán látható, hogy nincs kritikusnak mondható eltérés a tanulási és validációs loss esetében még is elbizonytalanodtam. Bár a modell képes volt teljesíteni nem látott emberi kézen valós időben szerettem volna biztosra menni. Erre van egy egészen egyszerű módszer, hogy tanítás során minden egyes epoch után elmentjük a modellnek éppen adott állapotát és ezeket bármikor betudjuk tölteni, amelyik epoch-nál csak szeretnénk. Sajnos én ezt a modellem tanításánál nem tettem meg. Így reménykedve abban, hogy ugyanazt a tanulási eredményt adja, ami a 8. ábrán látható újra tanítottam a modellt ugyanazokkal a beállításokkal, mint amit használtam eddig. Itt szintén külső segítségre volt szükségem hiszen az én GPU-m nem képes 8-as batch size-val és 640x640-es képekkel futtatni a tanulást, szerencsére Molnár Áron ezt lehetővé tudta tenni számomra. A tanásítást befejezve láttam, hogy szinte megegyezik a tanítási, validációs loss értéke a 8. ábrán látható modellével így mertem ezen a modellen tesztelni azt, hogy valóban fent áll-e az overfitting. Mivel itt el volt mentve az összes epoch így a 8. iterációt

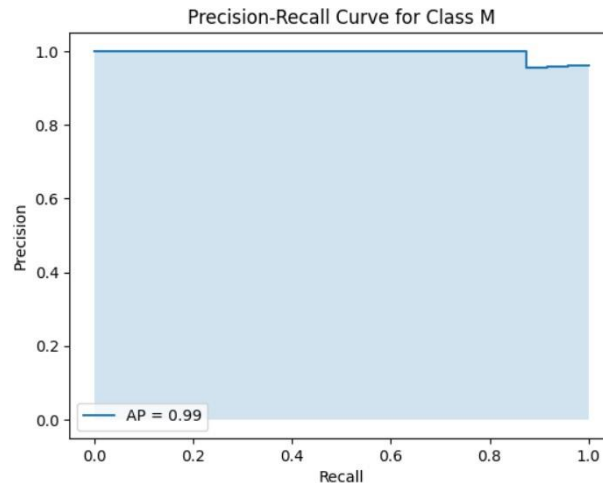
töltöttem be az alapján, hogy talán itt volt az, amikor a validáció picit elvált a tanítástól. Továbbá teszteltem a végső 20. epoch-ot is. Ezeket a modelleket végig futtatva a metrikai teszteken az eredmények alapján nem mondható az, hogy a modellem overfittinggel csupán csak valóban jól teljesít. Erre a modellre csak eddig volt szükségem, hogy képes legyek epoch-ok alapján is tesztelni. A továbbiakban azt a modellt használom, amit eddig is.

A ROC (Receiver operating characteristics) görbe egy grafikus ábrázolás, amely hasznossá válhat számunkra az osztályok rendszerezésének és teljesítménynek vizualizált megjelenítéséhez [17]. A görbe az osztályozási modellek True Positive Rate és False Positive Rate közötti kapcsolatot mutatja meg. A 16. ábrán látható a ROC görbe megvalósítása a modellem esetében. Az AUC (Area Under the Curve) a ROC görbéhez kapcsolódó adat. Ez egy mérőszám, amely megmutatja, hogy a modell milyen jól különíti el a pozitív és negatív osztályokat. Értéke mindig 0 és 1 közötti lesz [17].



15. ábra: ROC görbe (saját szerkesztésű ábra)

Alapvetően mindegyik osztály más színnel van ábrázolva ez a 16. ábra jobb oldalán látható is viszont mivel az értékeik megegyeznek így lefedik egymást. De látható továbbá, hogy van néhány betű, aminek kicsit kisebb az értéke, mint 1.



16. ábra: Pontosság és visszahívás görbe az M osztályra (saját szerkesztésű ábra)

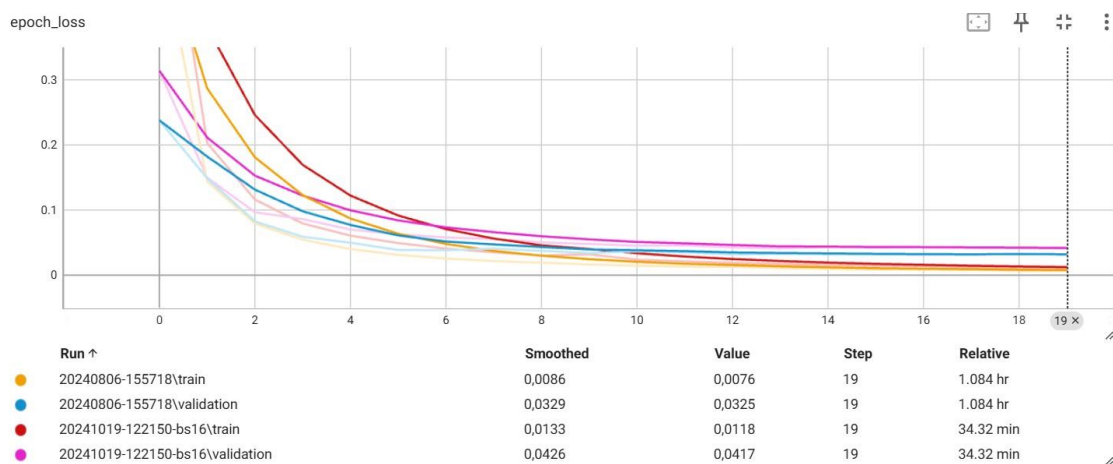
Egy további grafikon a Pontosság és Visszahívás görbe (Precision-Recall Curve), ami szintén használható a modellek teljesítményének mérésére és ezeknek az eredményeknek az összehasonlítására. Egy jó modell esetében a görbe a grafikon felső sávjában halad, vagyis magas a precizitása és a visszahívása is egyszerre. Ez azt jelenti, hogy a modell kevés téves pozitív eredményt ad miközben sok valódi pozitív esetet talál meg. Én a 15. ábrán ezt a görbét a M betű osztályára jelenítettem meg, ahol látható egy kisebb csökkenés a jobb felső sarokban. Ez az egyik olyan betű amire voltak negatív értékelések ezért gondoltam ennek a megjelenítést érdekesebbnek. A többi osztály esetében a felső sötét kék vonal végig halad minden kilengés nélkül.

Összeségében a metrikai teszteredmények alapján kijelenthetővé vált azt, hogy sikerült egy olyan modellt létrehozni továbbtanítás segítségével, ami kifejezetten jól teljesít a saját célú objektumfelismerés esetében.

6.2. Modellek összehasonlítása

Amikor Molnár Áron által lehetőségem nyílt újra tanítani a modellem a fentebb említett okoknál fogva megkértem, hogy esetleg tudná-e futtatni nekem 16-os batch size-val az általam megírt tanuláshoz használt kódot. Az Ő részéről sokkal nagyobb erőforrások álltak rendelkezésünkre így megtette ezt nekem. Szóval lett két modellem melyek más-más batch size-val tanultak. Alapvetően nem ez a leghatékonyabb technika, amivel a modell teljesítményét lehet növelni, de kíváncsi voltam, hogy esetleg egy kétszer akkora batch size képes-e befolyásolni a modell teljesítményét. Sem a kódon, sem az adathalmazokon nem történt változás csupán csak a batch size, ami változott.

Az első összehasonlítási pont az a tanulás eredménye lesz, aminek a grafikus megjelenítését ismét a TensorBoard teszi lehetővé számomra. A 17. ábrán „Run” alatti oszlopában látható, hogy melyik modellhez melyik színnel jelölt tanulási és validációs loss függvény tartozik. Az ábrán látható, hogy a nagyobb batch size által a modell lassabban konvergál a másik algoritmushoz képest ennek az eredményét láthatjuk a „Value” oszlop alatt. Ez a különbség nem hatalmas, de ellenben látványossá teszi, hogy már csak egy kétszer nagyobb batch size is képes befolyásolni a tanulás eredményét és lassítani a modell tanulását. Viszont ellenben az is jól látható, hogy közel fele annyi idő alatt sikerült a tanítást lefuttatni. Ez viszonylag egy várható eredmény hiszen az algoritmus kétszer annyi adatot dolgoz fel. Viszont ez nem teljesen valós eredmény. Az én modellem alapvetően gyengébb GPU által lett tanítva. Mikor megtörtént a modell újra tanítása, hogy rendelkezésemre álljanak az egyes epoch-ok mentései akkor a 8-as batch size-val rendelkező modell 45 perc alatt tanult, de alapvetően ez az algoritmus csak az overfitting tesztelésére lett készítve, viszont én a dolgozatom során végig az első modellem használtam, ami a 8. ábrán is látható így ahhoz is hasonlítom. Így valójában a tanulási idő csak 11 perccel gyorsabb, de már ez is számottevő lehet.



17. ábra: 8-as és 16-os batch méret befolyása a tanulás során (saját szerkesztésű ábra)

Ezek után a 16-os batch size-val rendelkező algoritmust is alávettem a metrikai teszteknek, amelyek eredményei a 4. táblázatban lesznek láthatóak. Itt is 0.5 IOU küszöbértéket alkalmaztam és ugyanúgy minden osztályára végig futtattam. Ebben az esetben viszont már olyan eredmények voltak, amikre azt lehet mondani, hogy jobban teljesít az algoritmus. Alapvetően a másik modell sem teljesített rosszul hiszen láthattuk a 3. táblázatban, hogy szinten minden osztály hibátlan. Ebben az esetben a modell van,

ahol javított, de van, ahol rontott is. De összességében mikor majd kitek az átlagos pontosság átlagára láthatjuk, hogy minimálisan még így is javult.

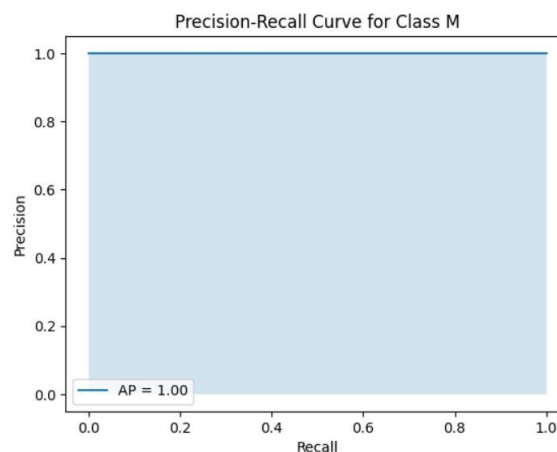
4. táblázat: 16-os batch size-val rendelkező modell metrikai eredményei

	P	R	F1	AP	TP	FP	FN
A	0.94	1	0.97	1	17	1	0
B	1	1	1	1	13	0	0
C	1	1	1	1	15	0	0
D	1	1	1	1	23	0	0
E	1	1	1	1	23	0	0
F	1	1	1	1	15	0	0
G	1	1	1	1	14	0	0
H	1	1	1	1	22	0	0
I	1	1	1	1	14	0	0
J	1	1	1	1	14	0	0
K	1	1	1	1	25	0	0
L	1	1	1	1	15	0	0
M	1	0.96	0.98	1	24	0	1
N	0.95	1	0.98	1	20	1	0
O	1	1	1	1	17	0	0
P	1	1	1	1	16	0	0
Q	1	1	1	1	14	0	0
R	1	1	1	1	18	0	0
S	1	1	1	1	16	0	0
T	1	0.96	0.98	1	23	0	1
U	1	1	1	1	25	0	0
V	1	1	1	1	24	0	0
W	1	1	1	1	14	0	0
X	1	1	1	1	13	0	0
Y	1	1	1	1	13	0	0

A táblázatban pirossal jelöltem azt az osztályt, ahol az algoritmus teljesítménye kisebbnek minősül, mint a kisebb batch size-val rendelkező modellé és zölddel jelöltem azt, ahol viszont jobb eredményt tudott elérni. Alapvetően ezek a teljesítményi hibák érthetőek az M, N és T betű esetében hiszen ezek roppant hasonlóak ennek az eredménye a valós időben történő felismerésnél is érzékelhető némely esetekben. Azt a gondolatot viszont nem feltétlen zárom ki, hogy abban az esetben, ha a nagyobb batch méretű modell esetleg még 1-2 iterációval többet kap akkor lehet, hogy sem az A betűre mért eredmény sem az N betűre mért eredmény nem lenne rosszabb, mint a másik modell esetében. Hiszen ez egy lehetséges példa arra, hogy lassabban konvergált, de ennek ellenére meg érdekes, hogy M és E betű esetében pedig javított.

Ennél a modellnél az átlagos pontosság átlaga (mAP), mint említettem minimálisan jobb lett ez számszerűsítve egész pontosan 0.9998, ami a másik algoritmus esetében 0.9997, ami tényleg egy roppant kicsi különbség, de továbbra is érdekes, hogy már csak a batch méret állításával is lehet különbséget elérni. Viszont az átlagos IOU értéke csökkent, azaz kisebb az átlagos metszet a valós és a predikciós bounding box között. Ennek a modellnek az átlagos IOU értéke 0.9221, míg az alap algoritmus esetében ez 0.9267. Itt se mondható hatalmasnak a különbség, de ebben az esetben rosszabb a 16-os batch méretű modell. Én ezt továbbra is a lassabb konvergálásnak tudnám be hiszen láthatjuk is 17. ábrán, hogy a loss értékekben van különbség 20 epoch alatt és valószínűleg ezt tükrözik az eredmények.

További összehasonlításként ennek a modellnek is készült ROC, Pontosság és Visszahívás görbéje, amelyek közül a Pontosság és Visszahívás, ami érdekesebbnek mondható szintén az M betű osztályra hiszen fentebb is azt tekintettük meg és ez az egyik osztály, ahol javítást is értünk el. Ennek az eredménye a 18. ábrán látható, ahol a sötét kék egyenes, ami egyébként az átlagos pontosság szépen kiegyenesedett és nincs benne beesés már.



18. ábra: 16-os batch méretű modell javítása az M osztályra

6.3. Valós időben történő felismerés tesztelése

Végül úgy gondoltam, hogy érdemes lenne egy olyan tesztet is végezni, ami azt vizsgálja, hogy különböző emberek kezén és más-más háttér, fényviszony esetében, hogy teljesít a modell a valós időben történő felismerés közben. Ebben az alfejezetben ezeknek a méréseknek az eredményére fogok kitérni és ezekből egy következtetést vonok. Alapvetően ez az a tesztelési lehetőség, ami a legkézenfekvőbb, mert semmiféle komplex

programozásra nincs szükség csupán szükséges volt a baráti és családi körben megkérni néhány embert, hogy legyenek tesztalanyok és mutogassák végig az ábécét számomra.

6.3.1. A tesztelők

A valós idejű teszteléshez összesen 30 személyt kértem meg, hogy legyenek a tesztalanyaim a vizsgálathoz. A tesztelők 22 és 48 év közötti emberek volt más-más testi adottságokkal, amelyek befolyással lehetnek a kezek formáira. Gondolok itt arra, hogy valakinek hosszabb és vékonyabb ujjja volt, volt, akinek rövidebb és vastagabb, volt, akinek erősen csontosabb volt és így tovább, illetve rendelkezésemre állt olyan személy is, aki műkörömmel rendelkezett.

6.3.2. A vizsgálat

Az elvégzendő teszt során a megkért tesztalanyokat megkértem, hogy üljenek be az elkészített szoftverem elé és egy kép segítségével, ami a 11. ábrán látható mutogassák el az ábécé betűit. Néha segítségükre voltam olyan értelemben, hogy egy-egy nehezebb kéz jel esetében segítettem beállítani a kéztartásukat. Miközben ezt elvégezték én felírtam az eredményeket és átlagot vontam belőlük, amiknek az eredményeit a következő alfejezetben láthatják.

6.3.3. Tesztelés

A mért adatok az 5. táblázatban láthatóak. A betűk átlag predikciója úgy értendő, hogy 1 tesztelő végig ment az ábécén és a 25 betű értékének az átlaga került feljegyzésre. Azért 25 és nem 26, mert már többször is említett okok miatt a Z betű nem került bele a modellembe.

5. táblázat: A modell teljesítése valós időben átlagosan a betűkre

Tesztelő	Betűk átlag predikciója	Tesztelő	Betűk átlag predikciója	Tesztelő	Betűk átlag predikciója
1. Tesztalany	99,16%	11. Tesztalany	90,96%	21. Tesztalany	82,16%
2. Tesztalany	96,08%	12. Tesztalany	96,88%	22. Tesztalany	95,16%
3. Tesztalany	95,92%	13. Tesztalany	85,04%	23. Tesztalany	95,24%
4. Tesztalany	92,76%	14. Tesztalany	92,88%	24. Tesztalany	89,2%
5. Tesztalany	98,36%	15. Tesztalany	95,92%	25. Tesztalany	96,72%
6. Tesztalany	90,2%	16. Tesztalany	98,28%	26. Tesztalany	95,96%

7. Tesztalany	92,84%	17. Tesztalany	91,84%	27. Tesztalany	96,12%
8. Tesztalany	91,04%	18. Tesztalany	89,84%	28. Tesztalany	97,8%
9. Tesztalany	95,92%	19. Tesztalany	89,12%	29. Tesztalany	88,44%
10. Tesztalany	98,08%	20. Tesztalany	91,32%	30. Tesztalany	92,56%

Az emberek változatosságának köszönhetően több háttér verzió is előfordult ezek mellett más-más fényviszonyok, de alapvetően a fényviszonyról elmondható az, hogy átlagos azért világosabb helyiségekben történt a tesztelés. Voltak sötétebb helyiségek is, amiknek az eredménye a vártaknak megfelelően viselkedett.

Kék sorral azok a tesztalanyok vannak jelölve, akiknek a kezük a tanítási adathalmazban is szerepelt. Piros sorral pedig azok az alanyok rendelkeznek, ahol a valamilyen oknál fogva volt olyan betű, amit nem volt képes felismerni a szoftver így azok a betűk automatikusan 0 értékelést kaptak így jelezve a behatást az átlagon.

A teszt rendkívül hasznosnak volt mondható a továbbfejlesztések feltárásának kapcsolatában. Alapvetően a tesztelés idejére lentebb vettem a confidence threshold értékét 0.7-re így nem csak 90% feletti predikciókat jeleníti meg a modell. A legfontosabb levonható dolog a tesztből az, hogy a lentebb olvasható továbbfejlesztési lehetőségek erősen ajánlottak, mivel egy-egy betű esetében borzasztóan figyelni kell arra, hogy az illető, hogyan tartja a kezét, mert a tanítási adathalmazban nincs elegendő eshetőség erre és ennek az oka volt az, hogy némely betűt nem érzékelte, mert volt aki nem tudta úgy tartani a kezét például rövidebb és vastagabb az ujjá és nem úgy látszik, ahogy például nálam akinek jelentősen nagyobb a keze és hosszabb ujjai vannak. A legkritikusabb betűk az a Q, J, N és T. Ezeknél a betűknél fordult elő leginkább, hogy nem ismerte fel abszolút a szoftver vagy éppen az N és T esetében erősen képes keverni őket az M betűvel. Emellett a fényviszonyok is igen megnehezítették a dolgunkat tesztelés során, mert voltak olyan helyzetek, ahol sötétebb szituáció volt, mint a legsötétebb tanítási adaton. Viszont ennek ellenére összevetve azt, hogy a kék sorokban lévő eredmények nem térnek el annyira óriásian azoktól, amelyek teljesen új kezek nem mondható az, hogy rosszul teljesítene a modell. A legnagyobb szépséghibája az, hogy meg kell állítani a kezét előtte megfelelő módon és akkor nagyon szép eredményt kapunk, de alapvetően ez egy valós időben dolgozó szoftver esetében nem várható el, hogy az objektumnak pár másodpercre meg kelljen állni mielőtt képes lenne felismerni.

Összességében a következtetés, ami levonható, hogy a szoftver jól teljesít egy kezdetleges fázisban lévő vagy úgymond első verziós szoftverhez képest, viszont nagyon erősen igényelné a később olvasható továbbfejlesztési lehetőségeket azok közül is az adathalmaz bővítését és az ezekkel való újratanítást a szépséghibák miatt.

6.4. További tesztelési lehetőségek

Egy érdekesebb tesztelési lehetőség, ami lehetséges, hogy eredményezett volna nagyobb eltéréseket a modellek között az egy olyan ötlet lett volna, hogy a kezeken az ujjakat vagy a körmöket bizonyos jelöléssel ellátni. Például a legegyszerűbb mód erre az lett volna, hogy körömfestéket alkalmazva minden körmet azonos színűre, illetve egy másik esetben pedig más-más színűre festek. Ez azért lett volna érdekes, mert egy piros köröm szín már jóval elűt az alapvető körömágy színtől és a CNN pont ezekre az apróságokra képes jobban figyelni ezzel akár más eredményt is elvérvé. A más-más színű köröm esetén pedig még érdekesebb lehetett volna ez a teszt hiszen ott minden ujj egy színt kap ezzel segítve esetleg a kritikus betűk felismerését. Sok olyan kézjel van, ahol viszonylag az ujjak eléggé közel vannak egymáshoz és nem feltétlen vehetők ki jól bizonyos helyzetekben. Ezzel a módszerrel ez igen jól kiküszöbölhető lett volna mivel egy piros szín jelentősen elűt a kezünk színétől.

Sajnos erre a tesztelési opcióra már nem maradt időm a fejlesztés és a dolgozatom végéhez érve. Szükség lett volna két szinten ekkora adathalmazra, amivel a korábbi modelljeim tanítottam hiszen így kaptunk volna releváns adatokat az összehasonlításhoz, de az rengeteg képfeldolgozással járt volna így ezt a módszert jelenleg nem tudtam véghez vinni.

7. Továbbfejlesztési lehetőségek

Ebben a fejezetben bemutatom a szoftverem továbbfejlesztési lehetőségeit több szempont szerint, melyek akár jobb teljesítményt nyújthatnak egy nagyobb és változatos adathalmaz segítségével vagy éppen felhasználhatóbbá teszik szélesebb körökben.

7.1. Adathalmaz bővítése

Talán a legalapvetőbb és olyan szempontból a legegyszerűbb fejlesztési lehetőség egy tanuló algoritmus esetében az adathalmazunk tovább bővítése és változatosabbá tétele. Azért mondhatjuk ezt a legkönnyebbnek hiszen itt nincs szükségünk programozási beavatkozásokra a fejlesztésekhez, hanem csak további adatok gyűjtését igényli, amely inkább egy sokkal időigényesebb munka.

Az adathalmaz tovább bővítésének az első célja a fent említett augmentációs módszerek nagyobb skálán való felhasználása, amely a valós időben történő felismerést tudná javítani. Ebben az esetben főleg a fényerőnek a nagyobb spektrumú használata, ami sokat tudna segíteni véleményem szerint. A beállított -15% és +15% intervallum sem minősül rossznak viszont vannak olyan helyzetek mikor ez kevésnek minősül hiszen előfordulhat, hogy a helység megvilágítsa, ahol használjuk a szoftvert sötétebb, mint a 19. ábrán látható. Így esetleg ezt növelve 20 vagy 25-ös méretű skálán akár már javíthatnánk ezt a problémát, hogy még felhasználóbarátabbá váljon.



19. ábra: -15% fényerő az augmentáció során (saját szerkesztésű ábra)

Egy másik opció, ami itt hasznossá tud válni az adatok elforgatása. Ezt augmentációval is képesek vagyunk bővíteni, hogy nagyobb fokokban forgassa el jobbra balra, illetve itt már további adatok gyűjtése is aktuális lehet. A mesterséges bővítés alatt

én -10° és $+10^\circ$ fok közötti értéket használtam ezt szerintem nyugodtan állíthatnám akár kétszer ekkorára is ezzel még nem romlana az adatok minősége. Továbbá olyan képek gyűjtése, amelyeken az objektum nem csak jobbra vagy balra van döntve, hanem előrefele és hátrafelé is akár meg van döntve. Ezzel már az adathalmaz mérete is nőne, viszont biztos vagyok benne, hogy a webkamera segítségével történő felismerés során ez rengeteget tudna javítani hiszen nem várhatjuk el a felhasználótól, hogy merev pontosan mutassa a kamerának a kezét állandóan.

A következő ponttal volt a legkevesebb a gond, de ez is számíthat egy olyan szoftver esetében, ami már valós célú felhasználású. Ez pedig nem más, mint az emberi kezek bővítése az adathalmazban. Ugyan nem voltak súlyos problémák idegen kezek felismerésénél, de az általam gyűjtött 5 emberi kéz közel sem mondható optimálisnak egy olyan esetben, ahol a szoftver rengeteg kézzel találkozhat. A valós idejű teszt megerősítette bennem azt, hogy ennek a bővítése is egy fontos és el nem kerülhető feladat. Továbbá szükséges lehet gondolni a bal kezes emberekre, mert tanítás során csak egy bal kezes személy állt rendelkezésemre így azok felismerése nem működik a legoptimálisabban.

Illetve egy olyan augmentáció hozzáadása, amit sajnálok, hogy kihagytam az adathalmazom készítése során ez pedig a homályosítás. Nem kalkuláltam bele azt, hogy nem feltétlen adott mindenki számára egy olyan webkamera, ami tőle képes biztosítani. Ezzel a lehetőséggel viszont vigyázni is kell hiszen akár tönkre is tehetjük vele az adatunkat, tehát nem hatalmas arányban, de minimálisan mindenképpen megérni hozzáadni és tesztelni vele egy adathalmazt.

Ezeknek a fejlesztéseknek is az elvégzése után nagy valószínűséggel érhetnél el teljesítmény javulást a modellemnél, hiszen változatosabbá és robusztusabbá válna.

7.1.1. Gesztusokkal való bővítés

A Z betű mellett a legtöbb szó a jelnyelvben egy adott mozgás elvégzésével fejezhető ki. Így ezeknek az adatgyűjtéséhez már nem elegendőek a képi adatok hiszen azokkal nem tudunk egy mozgást tárolni így rövidebb videókra lenne talán szükségünk ahhoz, hogy képes legyen a modell ezeknek a felismerésére.

Ebben az esetben már más annotációs módszert szükséges alkalmazunk, mint a képeknél és nagy valószínűséggel más módszerrel kéne ezt előkészíteni a modell számára, amely egyelőre még számomra is rejtély, de muszáj leszek utána járni, ha

szeretnék egy jobb minőségű szoftvert készíteni, ami nem csak a betűket képes felismerni, hanem a szavakat is és ezekből így már akár mondatokat is alkotni. Illetve a felhasználóknak is sokkal könnyebbé válna a használata, hiszen nem betűnként kellene a szavakat elmutogatni.

7.2. Szoftverszintű fejlesztések

A legnagyobb probléma a szoftverrel szembe továbbá is az, hogy hiába sikerült csökkenteni az erőforrásigényét még így sem a legoptimálisabb. Erre a tapasztalatim alapján a legjobb orvoslás az lenne, hogy az adathalmaz méretét 640x640-ről csökkenteni kéne akár 220x220-as méretre. Ez azért lenne képes megoldani a problémát, mert amikor egy pillanatképet dolgozunk fel, ami jelentősen nagy méretű sokkal több erőforrást használ fel a programom, mint kisebb képek esetében. Erre akkor lettem figyelmes amikor teszteltem a 340x340-es adathalmazomon a szoftvert és viszonylag kevésbé akadt meg egy-egy pillanatkép feldolgozásakor a valós idejű felismerés. Ebből kifolyólag következtettem arra, hogy egy még kisebb képméretű adathalmaz előfordulhat, hogy már kisebb akadás nélkül képes lenne visszajelzéseket nyújtani számunkra.

Ehhez nincs szükségünk nagyvonalú változtatásokra csupán a már meglévő kódot szükséges optimalizálni, hogy kisebb méretű képekké konvertálja az adott pillanatképeket és ehhez megfelelően jelenítse meg a predikciót és annak az értékét a kezünk körül. Továbbá szükséges egy adathalmaz, ami hasonlóan kisebb méretű képeket tartalmaz, aminek a segítségével elkészíthetjük az új modellt.

Viszont felmerülnek ebben az esetben olyan kérdések, hogy esetleg egy 220x220-as méretű képekkel rendelkező adathalmaz vesztené-e az értékéből és kevésbé lenne pontos, mint esetleg a 340x340-es vagy a 640x640-es. Erre sajnos én jelenlegi tudásom alapján csak teszteléssel tudnék konkrét választ adni, de vannak olyan modellek melyek hasonló méretű képekkel dolgoznak, mint a 220x220 például az áttanított modell is 224x224-es képeken tanult. Így bizakodóak lehetünk, hogy itt se rontana a szoftver minőségén.

További kutatásokkal előfordulhat, hogy más megoldás is lehetséges gyorsabb és folyamatosabb felismerés eléréséhez, de ez több időt igényel számomra, mint amennyi jelenleg volt a dolgozatom elkészítésére így ez a továbbfejlesztések alatt derülhet majd ki.

7.3. Oda és vissza kapcsolat

A dolgozatom vége fele realizáltam egy fontos dolgot, ami az, hogy számunkra visszajelzést ad a szoftver a jelnyelvről, de azoknak, akik jelelnek számukra semmiféle visszajelzési módról nem gondoskodtam. Így egyelőre a szoftver csak egyoldalú, azaz csak azok felé képes fordítani, akik nem értik a jelnyelvet, de a süket némáknak nem ad arról visszajelzést, hogy mi mit szeretnénk kommunikálni velük. Erre a legegyszerűbb megoldás jelenleg egy olyan funkció implementálása lenne, ami a beszélő hangját feldolgozza és megjeleníti írásos formában akár a képernyőn akár egy szövegdobozon belül ezzel gyorsan orvosolhatóvá téve ezt a problémát, de erre lehetnek optimálisabb megoldások is, amik szintén további elmélyülést igényelnek a témában.

Nyilván rengeteg továbbfejlesztési lehetőség létezik még ennek a szoftvernek az esetében, ami lehetséges, hogy a tudásom korlátozottsága miatt vagy csupán még egyelőre fel nem bukkant hiba miatt nem vált világossá, de jelenleg ezek azok a pontok, amikkel tisztában vagyok.

8. Összegzés

Dolgozatom végéhez érve úgy érzem, hogy a kitűzött céломhoz mértén sikerült azt elérnem, amit szerettem volna. Ezalatt azt értem, hogy szerettem volna egy szoftvert készíteni, ami úgymond kapirgálja a mesterséges intelligencia alapjait. Amit úgy gondolok sikerült elérnem, ha nem is egy saját modell készítésével, de egy transfer-learning segítségével tanított modellel már igen. Emellett a legfontosabb mérföldköve a szoftvernek a valós idejű felismerés volt, amit nem hittem volna a dolgozat egyrészénél, hogy sikerül megvalósítani, de ezt a kihívást is leküzdöttem és ez is megvalósult az elvártaknak.

Rengeteg kutatómunkára volt szükség ebben a témában hiszen az egyetemi évek alatt alapvetően nem ez a téma felé irányulunk, de erre volt egy egész nyár, ami mondhatni a minimumhoz elegendő volt így már kicsit jobban értek ehhez a témához és talán hozzá is tudok fűzni dolgokat mikor erről beszélünk egy társaságban. Nyilván mindig van hova fejlődni, amit teljes mértékben érzek. A neurális hálók mögötti matematika elkerülésénél fogva is már sok olyan dolog van, amibe nem mentem bele olyan mélyen. Én úgy tudnám ezt megfogalmazni a legjobban, hogy talán ez a projekt a „Hello World” -je a mesterséges intelligencia világának talán egy picit több, de nem sokkal.

Összeségében a tudásom az biztos, hogy fejlődött talán nem is keveset egy új területen és örülök, hogy végül ez a téma mellett döntöttem így picit magabiztosabban lépek majd ki az egyetemről azáltal, hogy próbálkoztam felzárkózni a mai világhoz.

Irodalomjegyzék

- [1] Murel, J. and Kavlakoglu, E. *What is object detection?*, 2024 IBM [Online]. Elérhető: <https://www.ibm.com/topics/object-detection> (megtekintve 2024.09.26.).
- [2] IBM: *What is a neural network?* 2024 [Online] Elérhető: <https://www.ibm.com/topics/neural-networks> (megtekintve 2024.09.27.).
- [3] MIT 6.S191: Convolutional Neural Networks, Massachusetts Institute of Technology, [Online]. Elérhető: <https://www.youtube.com/watch?v=2xqkSUhmmXU&t=2754s> (megtekintve: 2024.09.25).
- [4] Mesterséges Intelligencia Elektronikus Almanach: – 1.3. A mesterséges intelligencia története [Online]. Elérhető: http://project.mit.bme.hu/mi_almanach/books/aima/ch01s03 (megtekintve: 2024.09.28).
- [5] Google Cloud: *What's the difference between deep learning, machine learning, and artificial intelligence?* [Online]. Elérhető: <https://cloud.google.com/discover/deep-learning-vs-machine-learning> (megtekintve: 2024.09.28).
- [6] Altrichter Márta, Horváth Gábor, Pataki Béla, Strausz György, Takács Gábor, Valyon József, *Neurális hálózatok*. Panem Könyvkiadó Kft., Budapest, 2006
- [7] J. Deng, W. Dong, R. Socher, L.-J. Li, Kai Li, és Li Fei-Fei, "ImageNet: A large-scale hierarchical image database", in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, Miami, FL: IEEE, jún. 2009, o. 248–255.
- [8] R. C. Martin, *Tiszta kód: Az agilis szoftverfejlesztés kézikönyve*. Kiskapu kiadó Kft., Budapest, 2010.
- [9] J. Vajiram és A. Senthil, "Brain MRI detection by Sematic Segmentation models-Transfer Learning approach", 2024, *arXiv*.
- [10] C. Fjellström és K. Nyström, "Deep learning, stochastic gradient descent and diffusion maps", *Journal of Computational Mathematics and Data Science*, vol. 4, p. 100054 2022.
- [11] Q. Sellat, S. K. Bisoy, és R. Priyadarshini, "Semantic segmentation for self-driving cars using deep learning", in *Cognitive Big Data Intelligence with a Metaheuristic Approach*, Elsevier, 2022, o. 211–238.

- [12] I. Culjak, D. Abram, T. Pribanic, H. Dzapo és M. Cifrek, "A brief introduction to OpenCV," in *Proceedings of the 35th International Convention MIPRO*, IEEE, 2012, o. 1725-1730.
- [13] J. A. Yacim és D. G. B. Boshoff, "Impact of Artificial Neural Networks Training Algorithms on Accurate Prediction of Property Values", *Journal of Real Estate Research*, köt. 40, sz. 3, o. 375–418, júl. 2018.
- [14] R. Padilla, W. L. Passos, T. L. B. Dias, S. L. Netto, és E. A. B. Da Silva, "A Comparative Analysis of Object Detection Metrics with a Companion Open-Source Toolkit", *Electronics*, köt. 10, sz. 3, o. 279, jan. 2021.
- [15] R. Padilla, S. L. Netto, és E. A. B. Da Silva, "A Survey on Performance Metrics for Object-Detection Algorithms", in *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*, Niterói, Brazil: IEEE, júl. 2020, o. 237–242.
- [16] Kniazieva, Y. (2024) *Object Detection: Key Metrics for Computer Vision Model Performance*, *Label Your Data* [Online]. Elérhető: <https://labelyourdata.com/articles/object-detection-metrics> (megtekintve: 2024.10.11).
- [17] T. Fawcett, „An introduction to ROC analysis”, *Pattern Recognition Letters*, köt. 27, sz. 8, o. 861–874, jún. 2006.
- [18] AI Medai: *Sign Language Alphabets From Around The World* [Online]. Elérhető: <https://www.ai-media.tv/knowledge-hub/insights/sign-language-alphabets/> (megtekintve: 2024.11.01).
- [19] M. Abadi és mtsai., "TensorFlow: A system for large-scale machine learning", 2016, *arXiv*.
- [20] A. Paszke és mtsai., "PyTorch: An Imperative Style, High-Performance Deep Learning Library", 2019, *arXiv*.
- [21] I. H. Sarker, "Machine Learning: Algorithms, Real-World Applications and Research Directions", *SN COMPUT. SCI.*, köt. 2, sz. 3, o. 160, máj. 2021.
- [22] L. Alzubaidi és mtsai., "Review of deep learning: concepts, CNN architectures, challenges, applications, future directions", *J Big Data*, köt. 8, sz. 1, o. 53, márc. 2021.

- [23] J. Tian, Q. Jin, Y. Wang, J. Yang, S. Zhang, és D. Sun, "Performance analysis of deep learning-based object detection algorithms on COCO benchmark: a comparative study", *J. Eng. Appl. Sci.*, köt. 71, sz. 1, o. 76, dec. 2024.
- [24] B. Pang, E. Nijkamp, és Y. N. Wu, "Deep Learning With TensorFlow: A Review", *Journal of Educational and Behavioral Statistics*, köt. 45, sz. 2, o. 227–248, ápr. 2020.
- [25] H. Li, G. K. Rajbahadur, és C.-P. Bezemer, "Studying the Impact of TensorFlow and PyTorch Bindings on Machine Learning Software Quality", 2024.
- [26] Y. Yu és mtsai., "Techniques and Challenges of Image Segmentation: A Review", *Electronics*, köt. 12, sz. 5, o. 1199, márc. 2023.
- [27] A. B. Amjoud és M. Amrouch, "Object Detection Using Deep Learning, CNNs and Vision Transformers: A Review", *IEEE Access*, köt. 11, o. 35479–35516, 2023.
- [28] A. Hassan, M. Refaat, és A. Hemeida, "Image classification based deep learning: A Review", *Aswan University Journal of Sciences and Technology*, köt. 2, sz. 1, o. 11–35, jún. 2022.
- [29] N. H. Patil, S. H. Patel, és S. D. Lawand, "Research Paper On Artificial Intelligence And It's Applications", *JAZ*, dec. 2023.
- [30] J. A. Nichols, H. W. Herbert Chan, és M. A. B. Baker, "Machine learning: applications of artificial intelligence to imaging and diagnosis", *Biophys Rev*, köt. 11, sz. 1, o. 111–118, febr. 2019.
- [31] M. Islam, G. Chen, és S. Jin, "An Overview of Neural Network", *AJNNA*, köt. 5, sz. 1, o. 7, 2019.
- [32] S.-H. Han, K. W. Kim, S. Kim, és Y. C. Youn, "Artificial Neural Network: Understanding the Basic Concepts without Mathematics", *Dement Neurocognitive Disord*, köt. 17, sz. 3, o. 83, 2018.
- [33] Q. Jiang és M. Liu, "Recent Progress in Artificial Neurons for Neuromodulation", *JFB*, köt. 15, sz. 8, o. 214, júl. 2024.
- [34] S. Agatonovic-Kustrin és R. Beresford, "Basic concepts of artificial neural network (ANN) modeling and its application in pharmaceutical research", *Journal of Pharmaceutical and Biomedical Analysis*, köt. 22, sz. 5, o. 717–727, jún. 2000.

- [35] S. Naeem, A. Ali, S. Anam, és M. M. Ahmed, "An Unsupervised Machine Learning Algorithms: Comprehensive Review", *IJCDS*, köt. 13, sz. 1, o. 911–921, ápr. 2023.
- [36] Y. C A Padmanabha Reddy, P. Viswanath, és B. Eswara Reddy, „Semi-supervised learning: a brief review”, *IJET*, köt. 7, sz. 1.8, o. 81, febr. 2018.
- [37] R. Yamashita, M. Nishio, R. K. G. Do, és K. Togashi, "Convolutional neural networks: an overview and application in radiology", *Insights Imaging*, köt. 9, sz. 4, o. 611–629, aug. 2018.
- [38] S. Indolia, A. K. Goswami, S. P. Mishra, és P. Asopa, "Conceptual Understanding of Convolutional Neural Network- A Deep Learning Approach", *Procedia Computer Science*, köt. 132, o. 679–688, 2018.w
- [39] M. M. Taye, "Theoretical Understanding of Convolutional Neural Network: Concepts, Architectures, Applications, Future Directions", *Computation*, köt. 11, sz. 3, o. 52, márc. 2023.
- [40] X. Zhao, L. Wang, Y. Zhang, X. Han, M. Deveci, és M. Parmar, "A review of convolutional neural networks in computer vision", *Artif Intell Rev*, köt. 57, sz. 4, o. 99, márc. 2024.
- [41] S. Prabu és J. M. Gnanasekar, "A Study on Image Segmentation Method for Image Processing", in *Advances in Parallel Computing*, M. Rajesh, K. Vengatesan, M. Gnanasekar, Sitharthan.R, A. B. Pawar, P. N. Kalvadekar, és P. Saiprasad, Szerk., IOS Press, 2021.
- [42] K. Vaishnavi, G. P. Reddy, T. B. Reddy, N. Ch. S. Iyengar, és S. Shaik, "Real-time Object Detection Using Deep Learning", *JAMCS*, köt. 38, sz. 8, o. 24–32, jún.
- [43] A. Godil, R. Bostelman, W. Shackleford, T. Hong, és M. Shneier, "Performance Metrics for Evaluating Object and Human Detection and Tracking Systems", National Institute of Standards and Technology, NIST IR 7972, júl. 2014.

Ábrajegyzék

1. ábra: Különbféle számítógépes látási lehetőségek [1]	2
2. ábra: Neurális hálózatok [2]	5
3. ábra: A mesterséges neuronok általános felépítése [13]	6
4. ábra: Konvolúciós neurális hálózat folyamata [3].....	8
5. ábra: A LabelImg szoftver felépítése (saját szerkesztésű ábra).....	14
6. ábra: A szoftver felépítési terve (saját szerkesztésű ábra).....	16
7. ábra: Annotált kép vizualizációs megjelenítése feldolgozás után (saját szerkesztésű ábra)	18
8. ábra: A modell tanulási folyamata (saját szerkesztésű ábra).....	20
9. ábra: Modell által jelzett értékek	21
10. ábra: Valós idejű pillanatképek a szoftver futása közben (saját szerkesztésű ábra)	24
11. ábra: Kézjelek [18]	25
12. ábra: IOU illusztrációja [14].....	30
13. ábra: Bounding box osztályozásának illusztrációja. [16].....	31
14. ábra: Precision (P) és recall (R) képlete [15]	32
16. ábra: Pontosság és visszahívás görbe az M osztályra (saját szerkesztésű ábra)	37
15. ábra: ROC görbe (saját szerkesztésű ábra).....	36
17. ábra: 8-as és 16-os batch méret befolyása a tanulás során (saját szerkesztésű ábra)	38
18. ábra: 16-os batch méretű modell javítása az M osztályra	40
19. ábra: -15% fényerő az augmentáció során (saját szerkesztésű ábra).....	44

Táblázatjegyzék

1. táblázat: Az adathalmaz betűnkénti darabszámai.....	12
2. táblázat: Mesterségesen bővített adathalmaz betűnkénti darabszáma.....	15
3. táblázat: Modell eredményei 0.5 IOU küszöbértékkel.....	34
4. táblázat: 16-os batch size-val rendelkező modell metrikai eredményei.....	39
5. táblázat: A modell teljesítése valós időben átlagosan a betűkre	41