[Külső fekete borítólap formátuma]

Széchenyi István Egyetem

Gépészmérnöki, Informatikai és Villamosmérnöki Kar

Informatika Tanszék

**SZAKDOLGOZAT**

**Fieszl Bence**

**Mérnök Informatikus BSc szak**

2022

|  |
| --- |
| [Gerincen:] Hallgató Neve, Évszám {Titkosított} |

**SZAKDOLGOZ****AT**

**Koronavírus kontaktkutatást segítő**

**videó elemző szoftver fejlesztése**

**Fieszl Bence**

**Mérnök Informatikus BSc szak**

**2022**

# Nyilatkozat

Alulírott, Fieszl Bence (ISOJQW), Mérnök Informatikus BSc szakos hallgató kijelentem, hogy a Koronavírus kontaktkutatást segítő videó elemző szoftver fejlesztése című szakdolgozat feladat kidolgozása a saját munkám, abban csak a megjelölt forrásokat, és a megjelölt mértékben használtam fel, az idézés szabályainak megfelelően, a hivatkozások pontos megjelölésével.

Eredményeim saját munkán, számításokon, kutatáson, valós méréseken alapulnak, és a legjobb tudásom szerint hitelesek.

Győr, [beadás dátuma]

hallgató

# Kivonat

Koronavírus kontaktkutatást segítő videó elemző szoftver fejlesztése

Szakdolgozatom fő témája egy neurális hálózattal támogatott koronavírus kontaktkutatásra alkalmas szoftvernek az elkészítése. Azért esett erre a témára a választásom, mivel a koronavírus egy mostani időben elég felkapott és fontos egészségügyi téma. Rengeteg embert érintő problémáról beszélünk, amit valahogyan meg kell fékezni. Emellett a neurális hálók témaköre is viszonylag új területe a számítástechnikának, ezért ennek a felépítése is tartogat kihívásokat és meglepetéseket. A dolgozatom célja, hogy ennek a rendszernek a felépítését és fejlesztésének menetét bemutassa. A munkám elkészítéséhez a számítástechnika és a gépi képfeldolgozás terén már népszerűnek számító neurális hálókat fogom segítségül hívni. A neurális hálózatok remek megoldásai ennek a problémának, mivel bizonyos típusuk képes a bemeneti képen előre megadott osztályokat megkeresni és kategorizálni. A dolgozatomban ezeket a hálózatokat mutatom be, a főbb alkotóelemeiket, működésüket. Emellett a szoftvernek a fejlesztési lépéseit és igényspecifikációját is levezettem a dolgozatomban, illetve a programhoz készített interakciós weboldalnak az elkészítését és működését. A neurális hálózatok közül a YOLOv5 hálózatot választottam a munkámban, ezzel dolgoztam a rendszer fő alkotóelemeként és ezért erről esik a legtöbb szó a dolgozaton belül is. Szerepel a munkámban a hálózatnak a felépítése, működési elve, és kicsit bővebb matematikai háttere, emellett a szoftver felépítése és betanítás. A dolgozat első fejezetében főleg az elméleti és matematikai hátterét fejtem ki, alapvető algoritmusokat, mint konvolúció, neurális hálók különböző rétegei és függvényei. A második fejezetben főként a program tervezéséről és tervezett megvalósításáról van szó, mint pl. adatszerkezetek felépítése, a rendszer strukturális felépítése. és megvalósítási tervezete. A munkám harmadik fejezetében főként az elmélet és a tervezet gyakorlati megvalósításáról, a megvalósítással kapcsolatos eszközökről és módszerekről van szó. A szoftver használatához szükséges felhasználói dokumentációt a negyedik fejezet tartalmazza. Emellett még a munkában szerepelnek a program teljesítményével kapcsolatos teszteredmények, amik az ötödik fejezetben találhatóak.

# Abstract

Development of Video analysis software for Covid contact research

The main topic of my thesis is the development of a software for coronavirus contact research made with neural networks. I choose this subject, because in these days the coronavirus is a trending and important health topic. This problem affects a lot of people life and health, and it should be stopped as soon as possible. In addition, the topic of neural networks is new, so it has a lot of challenge and mysteries The target of my dissertation is to introduce of the planning and the implementation of this development. For my work, I will use neural networks, which is already a popular and a perfectly usable solution for this job, because some types of neural networks can find objects and classify objects based on custom classes. In my work, I will introduce these networks with their basic structure, and their operation. After this, I will introduce the development steps and the software requirements specifications, and the layout of the website which will be used to interact with the user. For the neural network, I choose the YOLOv5 neural network. I used this network to provide the backbone for my software therefore this will be the main topic of my work. I will introduce this networks layout, operation, the mathematical backend, and the training of this network. In the first chapter I will introduce the theoretical background and the basic mathematical backgrounds and algorithms that are needed for the network, like convolution, pooling, network layers, activation, etc. In the second chapter I will explain the planning and the planned implementation of the software included the database structure and the system structural layout. In the third chapter the topic is mainly the implementation of the theory and the plans, and the used tools and methods for this job. The user documentation is written down in the fourth chapter. The fifth chapter contains information about the performance of the software, including test events, and test results.

# Tartalomjegyzék

TARTALOM

[Bevezetés 1](#_Toc98954976)

[1. Elméleti háttér 2](#_Toc98954977)

[1.1. Konvolúciós neurális hálózatok bemutatása és működése 2](#_Toc98954978)

[1.1.1. Konvolúció 2](#_Toc98954979)

[1.1.2. Összevonó rétegek 3](#_Toc98954980)

[1.1.3. Teljesen kapcsolt rétegek 4](#_Toc98954981)

[1.1.4. Aktivációs rétegek 5](#_Toc98954982)

[1.1.5. Tanulás 6](#_Toc98954983)

[1.1.6. Algoritmusok típusai 7](#_Toc98954984)

[1.2. YOLO algoritmus 8](#_Toc98954985)

[1.3. Emberek közötti távolság meghatározása 9](#_Toc98954986)

[2. Tervezés 11](#_Toc98954987)

[2.1. Fejlesztői hardver, hardverkövetelmények 11](#_Toc98954988)

[2.2. Igényspecifikáció 11](#_Toc98954989)

[2.3. Maszkok felismerése 13](#_Toc98954990)

[2.4. YOLO algoritmusok megvalósításai 13](#_Toc98954991)

[2.5. Maszkok és emberek detektálása YOLO algoritmussal 14](#_Toc98954992)

[2.6. Rendszer felépítése 14](#_Toc98954993)

[2.7. Kimeneti adatok tárolásának módja 17](#_Toc98954994)

[2.8. Webes felület 19](#_Toc98954995)

[2.8.1. Branding és logó 20](#_Toc98954996)

[2.9. Használt adatkészletek 21](#_Toc98954997)

[3. Megvalósítás 23](#_Toc98954998)

[3.1. Neurális hálózatok modern számítógépes környezetekben 23](#_Toc98954999)

[3.2. A YOLOv5 algoritmus 23](#_Toc98955000)

[3.3. A detektor 24](#_Toc98955001)

[3.3.1. A detektor által használt neurális hálózat 24](#_Toc98955002)

[3.3.2. A detektor felépítése 32](#_Toc98955003)

[3.3.3. Maszkok feldolgozása a hálózat által 34](#_Toc98955004)

[3.3.4. Emberek és távolságuk feldolgozása a program által 35](#_Toc98955005)

[3.3.5. Az eredmények feldolgozása 36](#_Toc98955006)

[3.4. A kiszolgáló weboldal 37](#_Toc98955007)

[3.4.1. Laravel keretrendszer 38](#_Toc98955008)

[3.4.2. Regisztráció és bejelentkezés 39](#_Toc98955009)

[3.4.3. Videók és Események megtekintése 40](#_Toc98955010)

[3.3.4 Élő videó megtekintése és konfigurálása 43](#_Toc98955011)

[3.5. A kapcsolattartó réteg 43](#_Toc98955012)

[4. Felhasználói dokumentáció 46](#_Toc98955013)

[4.1. A program követelményei 46](#_Toc98955014)

[4.1.1. A program hardveres követelményei 46](#_Toc98955015)

[4.1.2. A program szoftveres követelményei 46](#_Toc98955016)

[4.1.3. A program használata 49](#_Toc98955017)

[5. Tesztelés és eredmények 52](#_Toc98955018)

[5.1. Tesztelési esetek 52](#_Toc98955019)

[5.2. Tesztelési eredmények(RÉGI SÚLYOKKAL, ÚJ TANÍTÁS ALATT) 53](#_Toc98955020)

[5.3. Fejlesztési javaslatok 55](#_Toc98955021)

[6. Összefoglalás 57](#_Toc98955022)

[Irodalomjegyzék 59](#_Toc98955023)

[Mellékletek 62](#_Toc98955024)

# 

# Bevezetés

Az egész világot megrázta egy 2019-ben a Kínai Wuhan városából kiindult halálos vírusfertőzés. Ez a vírus aztán az egész világon terjedésbe kezdett. A koronavírus (SARS-CoV-2) világjárvány során számos ember vesztette életét. A vírus hatalmas tempóban terjed az emberek között, ezenfelül különböző mutációi is megjelentek, amik még veszélyesebbek, mint az eddig ismertek. A korházakat hatalmas nyomás éri a járványhelyzet alatt, rengeteg embernél képes a vírus akár napok alatt nagyon súlyos szövődményeket okozni. A vírus terjedése a legkönnyebben a levegőben, cseppfertőzéssel történik. Ezért a fertőzések elkerüléséhez elengedhetetlen, hogy az emberek megfelelően védekezzenek a vírus ellen. Ezt az egyik leghatékonyabban szájat és orrot eltakaró maszkkal tudják megtenni. Ezenfelül a megfelelő, másfél méteres védőtávolság betartásával is csökkenteni tudják a fertőzésveszélyt. Azonban probléma esetén, ha valaki ezeket az intézkedéseket nem tartotta be, és igazoltan vírusfertőzés áldozata lett, akkor szükséges felkutatni a vele érintkezett személyeket is, akiket szintén meg kell vizsgálni, és szükség esetén elkülönítést kell végezni a közösségi interakcióktól. A dolgozatomban egy ilyen, kontaktkutatást segítő szoftverrendszer tervezését és fejlesztését fogom levezetni. Ez a szoftver vizuális úton, képfeldolgozás segítségével, kamerán keresztül állapítja meg az emberek közötti kontaktokat. A szoftver elkészítése során a mesterséges intelligencia témaköréhez kapcsolódó neurális hálózatokat fogom használni az embereken történő maszkok felismeréséhez. Mindemellett a korábban említett védőtávolság betartásának az ellenőrzését is meg fogja valósítani a szoftver szintén neurális hálózatok, képtranszformációk és vektorszámítások segítségével. Ezeket az adatokat később pedig a felhasználó egy webes felületen vissza tudja tekinteni, amely felület PHP alapokon nyugszik.

# Elméleti háttér

## Konvolúciós neurális hálózatok bemutatása és működése

A gépi látás területén már régóta foglalkoztatja a kutatókat az objektumok felismerésének gondolata. A kétezres évek előtti számítógépes rendszerek azonban túl gyengének bizonyultak az objektumfelismerő algoritmusok számára. A felismerő algoritmusok rengeteg erőforrást emésztettek fel, és a grafikus gyorsítók koráig nem is nagyon tudtak álmodni az emberek ilyen algoritmusokról. Azonban az utóbbi időben a grafikus gyorsítók fejlődésével megnyílt az út az objektumfelismerés felé. Egyre hatékonyabb algoritmusok jelennek meg évről évre, amik különböző megközelítésből indulnak neki ezeknek a problémáknak

Az objektumfelismerő algoritmusoknak rengeteg típusa létezik, azonban az áttörést a CNN (Konvolúciós Neurális Hálózat) típusú algoritmusok hozták meg 2014 környékén.[43]

A konvolúciós neurális hálózatok a nevükből eredően konvolúciókat használnak a kimeneti eredmény meghatározásához.

### Konvolúció

A konvolúció egy speciális matematikai művelet, ami képfeldolgozás esetén egy előre megadott konvolúciós kernelt felhasználva, az adott képpont körüli koordinátákat összeszorozva a kernellel (tulajdonképpen súlyozza), és utána ezeket a szorzatokat összegezve adja meg az adott képpontunk koordinátáit. Ez a művelet az 1. számú képletben látható, ahol K-val jelöljük a konvolúciós kernelt, I-vel a bemeneti képet, és J-vel az új, kimeneti képet

(1)

A konvolúciós műveletnek az eredményét a következő, 2. számú képletben található egyenlőség segítségével számítjuk ki. A képletben a betűk jelentése ugyanaz, mint az előző, 1. képletben.

(2)

A CNN-ek ezt a konvolúciót használják fő műveleteiknek. Ezeknek az algoritmusoknak az általános működését írja le Keiron O’Shea tanulmánya[28]. A tanulmány szerint ezekben az algoritmusokban a konvolúciós kernelek taníthatóak, ezáltal képesek különböző objektumok jellemzőpontjainak meghatározása. Ezek a jellemzők egy úgynevezett aktivációs térképen vizualizálhatóak is, amiken láthatóak, hogy a konvolúciók során a kép mely részleteit találta meg az algoritmus. Azonban a képek széleinél problémát okoz, hogy ott nem végezhető el a konvolúció, ezért sok esetben egy nullákkal feltöltött keretet adunk a képnek, hogy ott is el tudja végezni az algoritmus a műveletet.

### Összevonó rétegek

Az összevonó, vagy idegen szóval pooling rétegek olyan speciális rétegek, amik arra szolgálnak, hogy a számítási kapacitásokat csökkentsük, és ezzel gyorsítsuk az algoritmust. Ezeknek a rétegeknek az a feladatuk, hogy a konvolúciós műveletek után lecsökkentsék a képnek a méretét. Ezt úgy végzik el, hogy a konvolúciós rétegek lefutása után az összevonó réteg megkapja a konvolúciós réteg kimeneti képét, ez lesz az ő bemeneti képe. Ezután ezen a képen egy megadott méretű kernellel végig megyünk, és egy előre kiválasztott matematikai művelet segítségével összevonjuk a kernelbe tartozó képpontokat. A művelet során kapott kimeneti képet adjuk tovább a következő művelet számára.

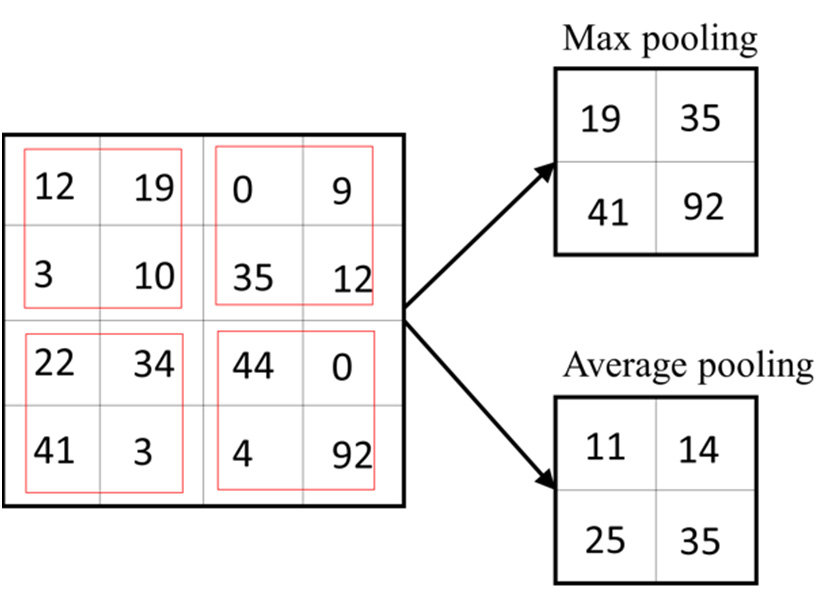
Az algoritmusok erre a műveletre egy méretű kernelt használnak. Én most egy méretű kernel segítségével fogom bemutatni ez a műveletet. Ez a kernel jellegéből adódóan 4 képkockából fog készíteni 1-et, ezzel a bemeneti kép eredeti méretét 25 százalékkal tudjuk csökkenteni. Erre a kernelre azonban többféle műveletet is lehet alkalmazni, ezekből fogok bemutatni két gyakori műveletet

* **maxpooling**

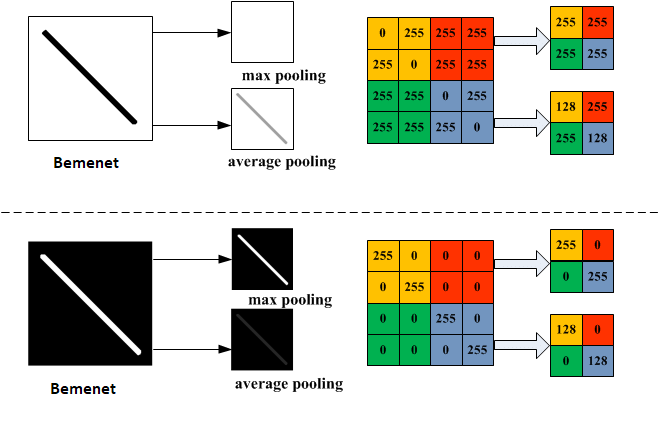
A maxpooling műveletnél a nevéből is adódóan az algoritmus a kernellel lefedett elemek közül megkeresi a legnagyobb elemet, és ez a megtalált, legnagyobb elem lesz az új képen egy képkocka. Ennek a műveletnek a megfordítása a **minpooling**, ami ugyanezen az elven működik, csak nem a legnagyobb hanem a legkisebb elemet választja ki és az lesz az új képen a képkockának az értéke.

* **avgpooling**

Az avgpooling esetén a képen a kernel által lefedett elemeket összeadjuk, és elosztjuk az elemek darabszámával (számtani átlagot veszünk). Ez a megkapott eredmény lesz az új képen egy képkockának az értéke.



1. ábra: maxpooling és avgpooling[8]



2. ábra: Az avgpooling és a maxpooling közötti különbség [42]

### Teljesen kapcsolt rétegek

A detektálás során a konvolúciós és összevonó rétegek többször végrehajtódnak egymás után megadott sorrendben. Azonban a futás legvégén nem ezek szerepelnek, hanem egy teljesen kapcsolt rétegen megy keresztül az adathalmaz, aminek segítségével meghatározza az algoritmus az aktivációs térkép alapján az adott képnek az osztályát, amibe be lehet sorolni. A teljesen kapcsolt rétegben az összes neuron össze van kötve az előző réteg összes neuronjával, a kapcsolatok súlyozva vannak, ezek alapján történik a végeredmény kiszámítása.

### Aktivációs rétegek

A konvolúciós neurális hálózatok esetében az algoritmusok használnak úgynevezett aktivációs rétegeket. Ezek a rétegek arra szolgálnak, hogy a bemenetként kapott adaton egy függvényt hajtanak végre, és a függvény által módosított adatot adják meg a kimenetükön.

Ezek a rétegek különböző függvényeket használhatnak, amikből most fogok felsorolni néhányat. Ezeket a függvényeket Sagar Sharma Towards Data Science oldalon fellelhető cikke alapján gyűjtöttem össze[34].

* **ReLU**

A ReLU függvény a rektifikált lineáris egység függvénynek a rövidítése. Ez a függvény működése szempontjából annyit végez, hogy a negatív elemeket nullára állítja, a többit változatlanul hagyja. Gyakran használt aktivációs függvény, mivel nem igényel bonyolult matematikai műveletet és gyors.

Az eredmény a tartományba esik.

(3)

* **Szivárgó(leaky) ReLU**

A szivárgó ReLU az előzőleg bemutatott ReLU függvény egy hibáját javítja, ami alapján, ha túl nagy eltolást használunk a függvény használata előtt, és az értékek eltolódnak negatív irányba, akkor a függvény mindenre nulla értéket fog visszaadni. Ezért a ReLU függvényt úgy gondolták újra, hogy nem fog nulla értéket adni, hanem egy paramétert (a) építettek be a függvénybe, ami a negatív értékeket is figyelembe veszi.

Az eredmény a tartományba esik.

(4)

* **sigmoid**

A szigmoid függvény egy tartományba eső értéket ad vissza eredményként, ami tökéletes pl. osztályozó algoritmusok számára, mivel itt az osztályvalószínűség pont egy tartományba eső szám.

(5)

* **SiLU**

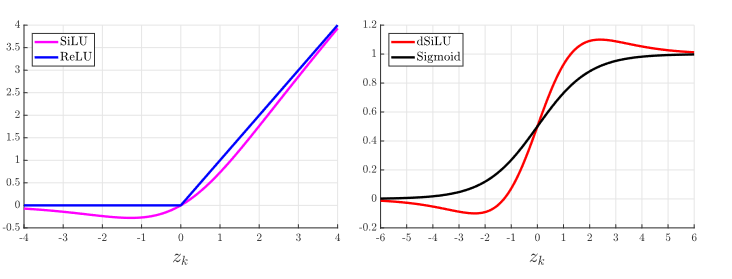
A SiLU[14] függvények a szigmoid függvénynek és a ReLU függvénynek az összemosásának is tekinthetőek. Ezeket a függvényeket abból a célból hozták létre, hogy egy hatékony és gyors aktivációs réteget tudjanak létrehozni. A függvény tulajdonképpen egy súlyozással történő eltolása a fent említett két függvénynek.

(6)

(7)

(8)

A függvény kiszámítása a felső 3 képlet segítségével számolható ki. A függvényekben a a rejtett egység bemenete, a szigmoid függvény az bemenet súlyozása az és a rejtett egység közötti kapcsolat súlyozása és a rejtett egység súlyozása.



3. ábra: a SiLU és a dSiLU függvények képe

### Tanulás

A neurális hálózatoknak egy fő folyamata a tanulás.[27] Ennek a folyamatnak a során alakul ki az algoritmusnak azon képessége, hogy képesek legyenek objektumokat felismerni és osztályozni. Azonban a tanulás nem egyszerű feladat, mivel meg kell találni az algoritmus által elkészített hálózatnak az optimális állapotát, hogy képes legyen minél nagyobb pontossággal, minél több objektumot felismerni. A folyamat során az algoritmus előre meghatározott, és felcímkézett bemeneti képek alapján tanulja meg a szükséges konvolúciós kerneleket és a különböző neuronokhoz tartozó súlyozásokat. A tanulási folyamat során először a hálózat véletlenszerű kernelekkel és súlyokkal töltődik fel, majd ezek az értékek a tanulási folyamat során változnak az algoritmus kimeneti adatainak függvényében.

A tanulás során a neurális hálózatok több matematikai műveletet alkalmaznak. Ezek közül a műveletek közül mutatok be néhányat nagyvonalakban, hogy mi a szerepük, és hogyan működnek.

* **Optimalizáció**

Az optimalizáció a tanulás során használt művelet a neurális hálózat minőségének javítására. Az optimalizáció során azt próbáljuk elérni, hogy a hálózat minél pontosabb eredményeket adjon, minél kevesebb legyen a hiba a hálózaton belül. Ezt úgy érjük el, hogy egy megadott módszerrel ki kell számolni, hogy a hálózat mekkora hibaaránnyal működik (pl. veszteséget számolunk), és ez alapján a hibaarányok alapján előre meghatározott módszerrel módosítanunk kell a hálózatunkon. Erre az optimalizációra többféle megoldás létezik, a népszerűbb megoldások közé tartozik pl. a gradiensereszkedés módszere[6] és az Adam optimalizáció[21].

* **Veszteségfüggvények**

Az optimalizáció során használt veszteségfüggvények[17] azt adják meg, hogy a bemeneti, tanító képeken megadott objektumok és osztályaik (tehát a várt kimenet) és a kimeneti objektumok és osztályaik (tehát a valós kimenet) között mekkora eltérés található. Az algoritmusok ezt az eltérést(veszteséget) minél alacsonyabban próbálják tartani, ezért úgy módosítják a hálózatot, hogy ezt a veszteséget minél jobban tudják csökkenteni. Ezt az eltérést különböző matematikai módszerekkel lehet kiszámolni, mint pl. átlagos négyzetes eltérés (MSE), vagy kereszt-entrópia (CE)[13]

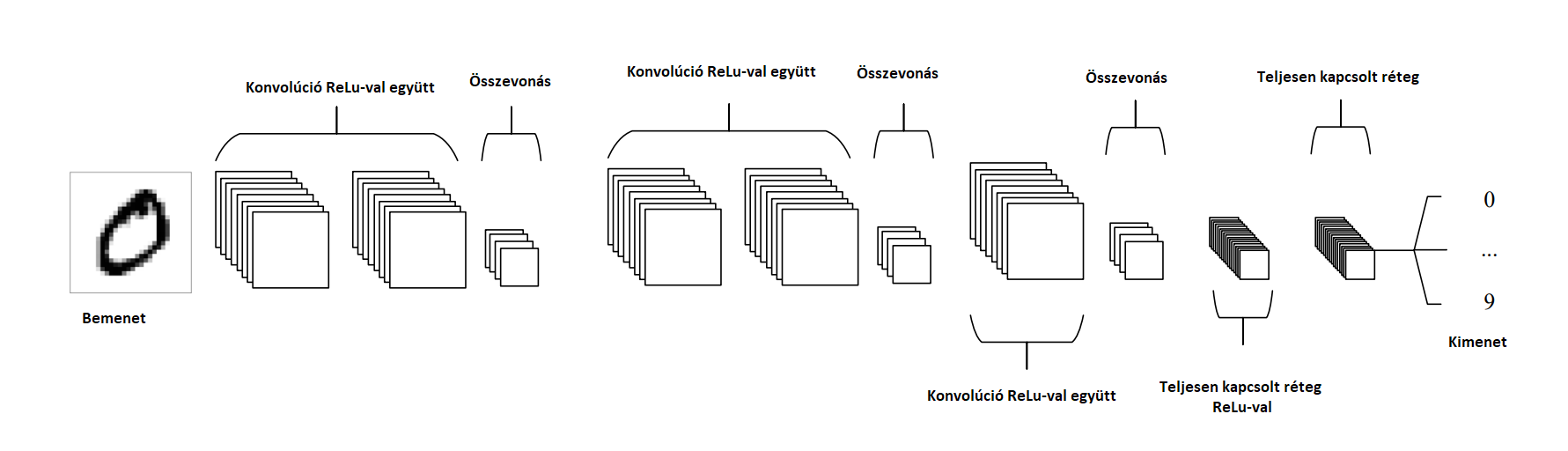
* **Kiejtés**

A kiejtés (dropout[35]) nevű műveletet a túltanulás elkerülésére szokták használni. A túltanulás az a jelenség, amikor a hálózat túl jól alkalmazkodik a tanító adatokhoz, és az azokra kizárólag jellemző formákat tanulja csak meg, ezért csak a hozzá hasonló bemeneten fog működni, más, tőle jobban eltérő bemeneteken sokkal alacsonyabb valószínűséggel fog működni. Ezért alkalmazzuk a kiejtést, ami azt jelenti, hogy bizonyos neuronokat ideiglenesen kizárunk a tanulási folyamatból, nem veszünk figyelembe, ezáltal sokkal általánosabb lesz a modellünk tanulás során. Ezt a kizárást lehet véletlenszerűen, vagy meghatározott matematikai függvény vagy logika mentén végezni.

### Algoritmusok típusai

A korábban tárgyalt CNN algoritmusok közül elsők között csak az egész képet osztályozni tudó algoritmusok léteztek. Azonban idővel megjelentek olyan algoritmusok is, amelyik egy kép régiójáról tudták megmondani az osztályt. Ezek közül is az első híresebb algoritmus az R-CNN[16] volt. Az R-CNN algoritmus első lépésben felosztja a bemeneti képet különböző régiókra (közel kétezer régiótípust ismer), és az alapján végzi az adott régión a konvolúciók után kiválasztásos keresés alapon a címke kiválasztását.

Azonban ez az algoritmus a forrás szerint[43] is lassúnak bizonyult (14s/kép GPU-n), ezért valós időben nem használható. Azonban az idő múlásával újabb algoritmusok is megjelentek. Az R-CNN algoritmusnak is megjelentek jobban optimalizált, gyorsabb változatai.[30] Azonban az igazi áttörést az egyszintű detektorok hozták meg. Ezek az algoritmusok a nevükből adódóan csak egyszer dolgozzák fel a képet és úgy mondják meg az objektumok osztályát. Ezek közül is a két legismertebb algoritmus az SSD (Single Shot Detector)[25] és a YOLO (You Only Look Once).[31]



4. ábra: Példa egy Konvolúciós Neurális Hálózat felépítésére [28]

## YOLO algoritmus

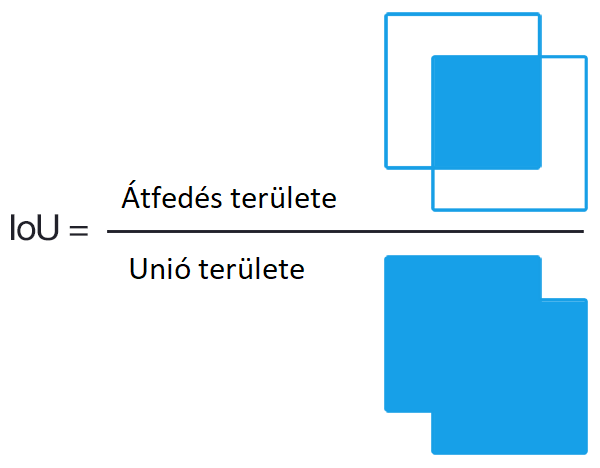
A YOLO a You Only Look Once nevű algoritmusnak a rövidítése. A következőkben ezt az algoritmust fogom bemutatni.[31] Ezt az algoritmust 2016-ban hozták létre abból a célból, hogy alkossanak egy gyors objektumfelismerőt a pontosság megtartása mellett. Ez az algoritmus az objektumfelismerés problémáját regressziós problémaként fogja fel.

Az algoritmus elsőnek számú négyzetekre bontja fel a bemeneti képet. Ezek a négyzetek mindegyike B darab befoglaló geometriát (bounding boxot) tartalmazhat, ami a benne észlelt objektumokat körbeveszi, ha az objektum középpontja a négyzetbe esik. Ezekhez a geometriákhoz tartozó tulajdonságok a

* középponti **pozíciójuk**
* **méretük**
* **confidence level**, hogy mennyire biztos az algoritmus abban, hogy az a geometria tartalmaz-e objektumot, és mennyire ítéli pontosnak az objektumot körülvevő geometriának a méreteit.

Ezt a confidence levelt (bizonyossági pontszámot) a következő, 6. számú képlettel számolja ki, ahol a Pr az objektum tartalmazásának valószínűsége, az IoU pedig az Intersection Over Union rövidítése, ami azt adja meg, hogy a geometriák milyen arányban fedik egymást. A kiszámítását az 5. ábra magyarázza.

(9)

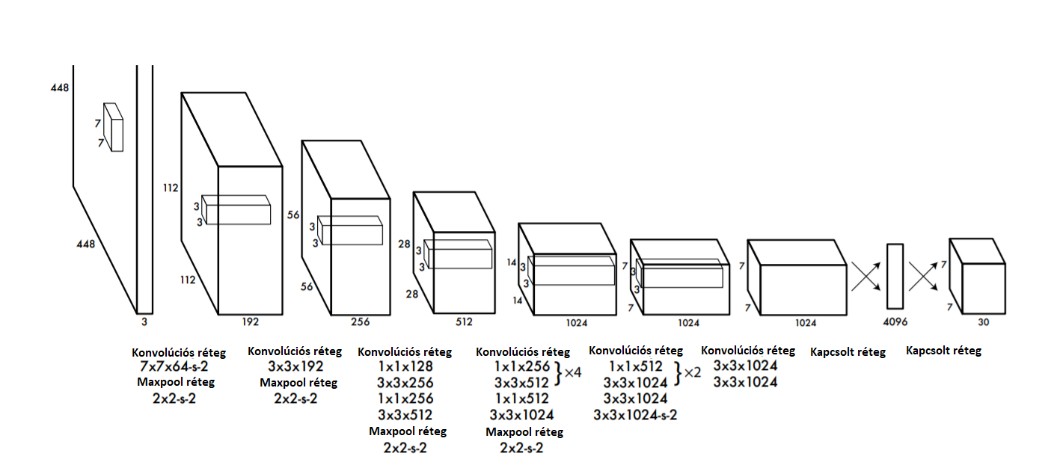


5. ábra: IoU magyarázata[2]

Ezek után azt is meghatározza az algoritmus, hogy az adott négyzet, amit vizsgálunk, az milyen osztályba sorolható elemet tartalmazhat. Ezt a következő, 10. képlettel lehet meghatározni, ahol a Pr a valószínűségeket jelenti az Oszt az adott osztály valószínűségét, az Obj pedig az adott geometriában található objektum valószínűségét.

(10)

Ennek a végén egy méretű tenzort kapunk, aminek mindegyik mezője egy B befoglaló geometriát tárol a bizonyossági pontszámmal, és C darab osztálynak a valószínűségét.



6. ábra: YOLO hálózat felépítése[31]

## Emberek közötti távolság meghatározása

Az emberek közötti távolság meghatározásához szükséges algoritmust már többen levezették korábban.[41] Az említett forrás alapján első lépésben szükséges felismerni az embereket a képen, amit a korábban felvázolt YOLO algoritmus fog elvégezni.[31] Ezek után következik az emberek pozíciójának a meghatározása a képen. A pozíciót a korábban megkapott befoglaló geometriák felhasználásával fogjuk tudni megmondani. A forrás ajánlata szerint a geometriák alsó-középső koordinátáját ajánlott használni, mivel az emberek ott érintkeznek a talajjal, és mivel a talajra közel derékszögben állnak, ezért ott a legpontosabb a köztes távolság mérése. Ennek a kiszámítása a detektáló algoritmus kimeneti egységétől függ, ha a bal felső koordináta és a szélesség, magasság van megadva, akkor a 11. képlet szerint lehet kiszámolni a koordinátákat

(11)

Azonban, ha a kép bal felső (), és jobb alsó () koordinátáit kapjuk meg, akkor pedig a 12. képlet szerint lehet kiszámolni a koordinátákat.

(12)

Ezeknek a koordinátáknak a kiszámítása természetesen minden felismert embernél megtörténik.

A következő lépésben a jelenlegi koordinátákat át kell helyezni 2 dimenziós, madártávlati nézetbe, hogy az emberek közötti távolság mérhető legyen. Ehhez a forrás egy jól ismert inverz homográf transzformációt[12] alkalmaz:

(13)

ahol M egy méretű mátrix, ami a perspektíva transzformációhoz szükséges információkat tartalmazza, a vektornak egy homogenizált reprezentációja, és egy homogenizált változata a madártávlati koordinátáknak. Ebből pedig le lehet származtatni a vektort.

Miután megkaptuk a személyek valós koordinátáit már csak a köztük lévő távolságokat kell meghatározni. Ehhez elég két pontvektornak az euklideszi távolságát venni, ami a 14. képletben található, és ha ez kisebb, mint a beállított távolság, akkor riasztás keletkezik a rendszerben.

(14)

# Tervezés

## Fejlesztői hardver, hardverkövetelmények

A szoftver jellegéből adódóan nem képes minden hardvertípuson futni. Mivel neurális hálót használ, ezért egy grafikus gyorsító szükséges a futásához. Igaz, a neurális háló képes grafikus gyorsító nélkül is, a számítógép processzorán futni, azonban a sebességében hatalmas visszaesés lesz tapasztalható ezesetben. A grafikus gyorsítók a többmagos, párhuzamos feldolgozásukkal sokkal gyorsabban tudnak végezni ezekkel a specifikus feladatokkal. Azonban a gyorsítók közül se mindegy milyen típust választunk. A piacon jelenleg kettő nagyobb gyártó van jelen, az Nvidia és az AMD (régebbi nevén ATI). Azonban ez a kettő gyártó a grafikus kártyán történő számítások elvégzéséhez kettő külön eszközt kínál. Az AMD hardvereknél a HCC/ROCm[4] nevű eszközt szolgáltatja a gyártó, Nvidia részéről pedig a népszerűbb, CUDA[11] elnevezésű eszközt. Az általam készítendő szoftver is ezt, a CUDA elnevezésű eszközt fogja használni futása során. Ehhez azonban szükséges egy Nvidia videókártya, ami az én esetben egy MSI GeForce GTX 960 Gaming 4G OC típusú grafikus vezérlő lesz. Ez a kártya már nem a legfrissebb technológiát használja, azonban a szoftver futásához még elegendő teljesítményt tudj nyújtani. A szoftver szempontjából pedig szintén ezt határozom meg alsó követelménynek.

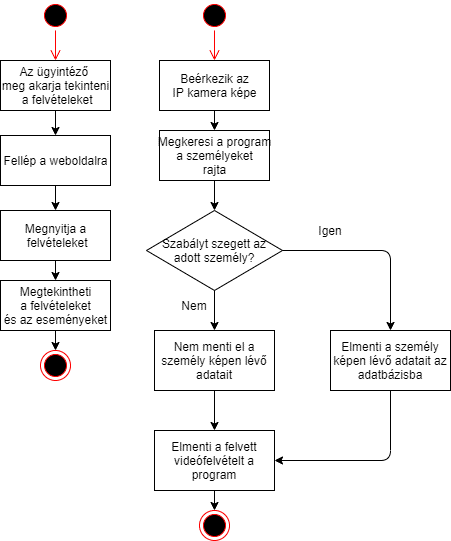
* Nvidia GTX 960 videókártya, vagy újabb (GTX 10xx, GTX 16xx, RTX 20xx RTX 30xx), legalább 4GB VRAM-mal, legalább CUDA 11.4 támogatással
* Intel i5 6. generációs CPU, vagy újabb
* 8GB RAM
* legalább 1,5GB tárhely a szoftvernek
* legalább 2TB tárhely a felvételek tárolására
* Ajánlott dedikált szerveren futtatni
* Legalább 1 MP felbontású kamera 25 fps sebességgel

## Igényspecifikáció

A jelenlegi rendszer szerint az igazoltan beteg személyek esetén személyes bevallás alapján történik meg a kontaktszemélyek felkutatása. A jelenlegi helyzetben az embereknek csak a lelkiismeretén és az emlékezőképességén múlik a betegség terjedésének meghatározása.

Ennek a helyzetnek a javítása érdekében történik ennek a szoftvernek a fejlesztése, hogy esetlegesen közterületen, vagy akár belterületen is (pl. egy gyár területén) megállapíthatóak legyenek a kontaktusok.

A szoftvernek több funkciót is meg kell valósítania. Először is a bejövő IP kamerának a képét fel kell dolgoznia, fel kell ismernie az embereket, és az arcmaszkokat rajta, amiket be kell kategorizálnia csoportokba is. Ezek közül a csoportok közül aztán ki kell válogatnia azokat, ahol az emberek szabályt vétettek (nem viselték rendesen a maszkot, vagy másfél méteres távolságon belül álltak egymástól), és ezeket a szabályszegéseket el kell tárolnia későbbi felhasználásra. Ezen kívül a szoftvernek mentenie kell a nyers videofelvételeket is, hogy később azok is elérhetőek legyenek egyéb feladatok esetén is akár (pl. bármilyen esemény visszatekintése érdekében). Ebből az adatbázisból ezek után egy weboldal segítségével az ügyintézők számára visszatekinthetőnek kell lenniük a tárolt adatoknak, hogy az esetleges kontaktszemélyek megtalálhatóak lehessenek. Ezekből aztán később lehet riportokat készíteni, videórészleteket lementeni.



7. ábra: A szoftver alapfelépítésének terve

## Maszkok felismerése

A szoftvernek a fő feladatai közé tartozik a maszkok és az emberek felismerése. A maszkfelismerés során az algoritmusnak nem szabad függenie a maszk színétől, típusától, kialakításától. Emellett az oldalról történő felimerést is támogatnia kell egy bizonyos mértékig. Azt is szükséges megvalósítani ezen felül, hogy a hibásan hordott, például a személy orrát nem takaró maszkot próbálja meg felismerni. Ezen felül pedig az algoritmusnak azt is figyelembe kell vennie, hogy az emberek milyen távolságra állnak egymástól. A koronavírus szempontjából a távolság fontos, mivel a fertőzésveszélyt nagymértékben tudja csökkenteni a legalább másfél méteres távolság tartása. Ehhez azonban az embereket is fel kell ismernie a programnak, különben a távolságokat nem fogja tudni megállapítani. Az informatikában a gépi látás témakörben az ilyen típusú problémákra már léteznek különféle megoldások. A legjobb megoldás erre a problémára a különböző objektumfelismerő és kategorizáló algoritmusok, amiket a dolgozatom első részében, az elméleti háttér során már bemutattam. Ezek az algoritmusok képesek megállapítani egy bizonyos képen, vagy képrészleten az objektumokat, és képesek az általuk előre betanult címkék alapján kategorizálni őket.

## YOLO algoritmusok megvalósításai

A korábban már bemutatott YOLO algoritmusnak különböző megvalósított verziói léteznek. Ezek a verziók általában azok alapján különböznek, hogy milyen optimalizáción, javításon estek át, esetleg milyen hálózatot használnak gerinchálózatnak a program működése során, milyen keretrendszeren keresztül használják ezeket a hálókat.

Az első és legismertebb YOLO megvalósítás a Darknet. A Darknet egy C alapon írt CUDA-t használó neurális hálózati keretrendszer.[19] Ez a keretrendszer a YOLO-hoz eredetileg írt keretrendszer és a YOLO eredeti implementációja. A keretrendszer képes CPU és CUDA gyorsítóval működni, támogatja az OpenCV könyvtárakat, aminek a segítségével különböző egyéb funkciók is megvalósításra kerültek.

A Darknet mellett természetesen vannak más megvalósításai is az algoritmusnak. Létezik különböző Python alapokon megvalósított megoldásai is, pl. PyTorch vagy Tensorflow alapokon készült megoldások. A program szempontjából a lényeges követelmények a GPU-n történő futás, Python nyelven implementálva és minél jobb eredményeket visszaadva. Ezért a program szempontjából ideális választás a YOLOv5 kiadása[37]

## Maszkok és emberek detektálása YOLO algoritmussal

A maszkok detektálásának folyamatához az algoritmusnak elsők között ismernie kell a főbb kategóriákat a maszkviselési módok között. Ezek közül hármat lehet elkülöníteni, amikor visel maszkot, amikor rosszul viseli a maszkot, és amikor egyáltalán nem visel maszkot. A maszkviselés az az eset, amikor a személy teljesen viseli a maszkját, az orrát és a száját is egyaránt eltakarja. A rossz maszkviselés az az eset, amikor a személy visel, maszkot, csak az nem takarja az orrát, amivel így ugyanúgy fertőzőképes marad, mivel az orrából a nedves cseppek ugyanúgy ki tudnak szállni a maszkon kívülre. A harmadik eset, amikor egyáltalán nem viseli a maszkot, az alatt azt az esetet értjük, amikor se a száját, se az orrát nem takarja a maszk (vagy az állán van). Ez az eset a legrosszabb a három közül, mert így teljesen védtelen a személy.

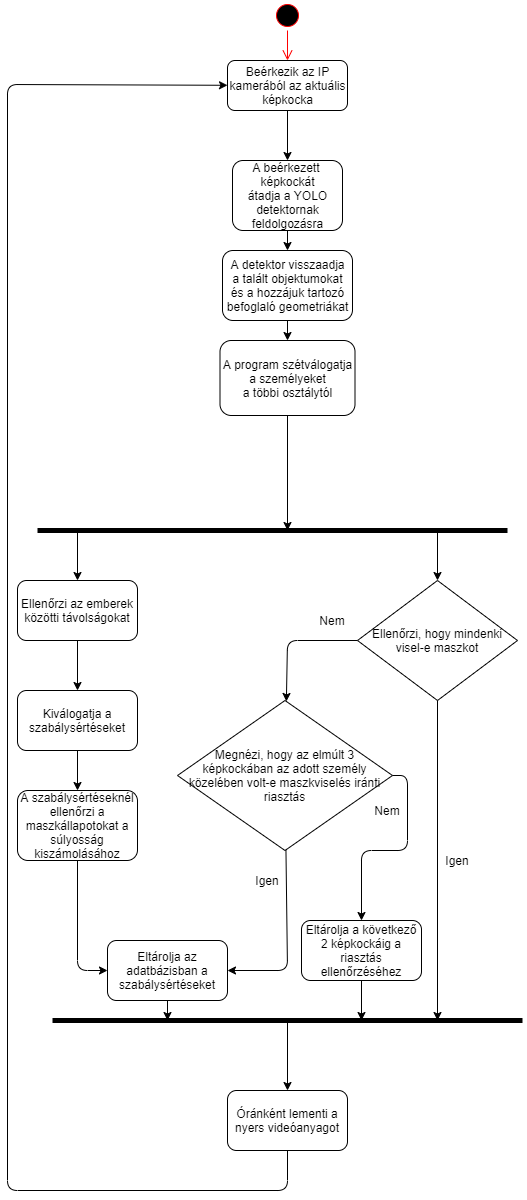
Ezeket az eseteket, hogy fel tudja ismerni az algoritmus első lépésnek meg kell tanítani vele. A betanítás során a hálóba képeket táplálunk, megadva a képrészleteket, amiket vizsgáljon, és azok kategóriáit. Ezek után a neurális háló elvégzi a képek jellemzőinek az összegyűjtését, és eltárolja a különböző súlyokat egy fájlban. Ezt a fájlt azután betöltjük a detektorba, és a neurális háló segítségével az algoritmus megkeresi a különböző objektumokat. Ezeknek az információit is megadja nekünk, az érzékelés valószínűségét, az objektumot körülvevő befoglaló geometria koordinátáit (bal felső koordináta () és jobb alsó ()).

Az emberek detektálása kicsit egyszerűbb feladat, mivel itt csak 1 állapotot kell felismerni. Itt is be kell tanítani az algoritmust, azonban mivel az emberek detektálása régebb óta aktív témakör, ezért sokkal több adatkészlet áll hozzájuk rendelkezésre, mint a maszkfelismeréshez. Sok nagyobb adathalmaz is tartalmaz hozzá adatot, mint pl. az MS COCO adatkészlet.[24]

## Rendszer felépítése

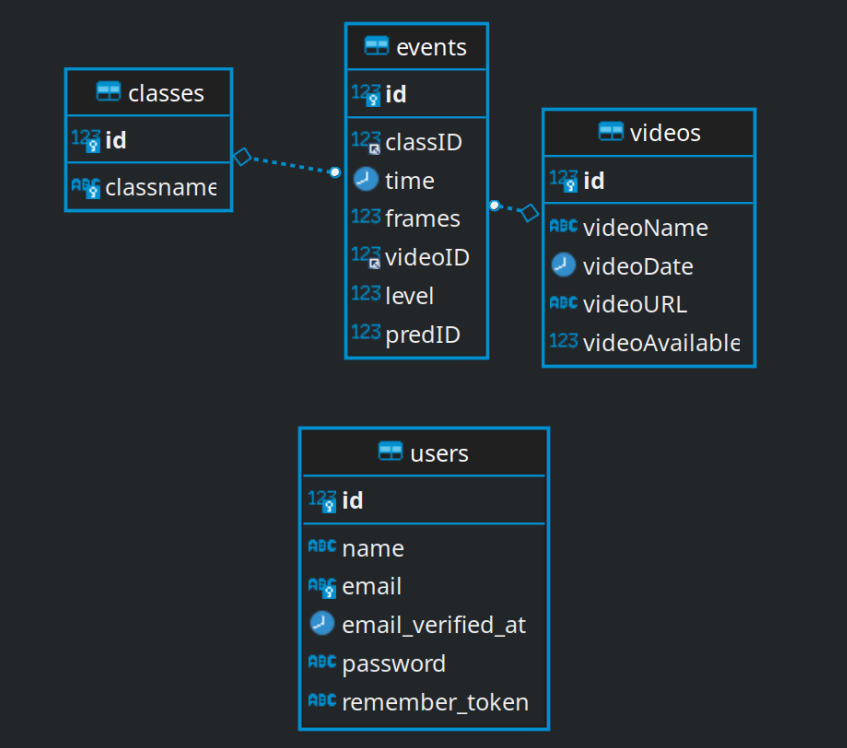
A rendszer az IP kamera beérkező képét képkockánként dolgozza fel. A kamera képkockáit egyesével, egymás után veszi, és adja tovább a YOLO detektornak. A detektor ezek után a neurális hálója segítségével megpróbálja megtalálni a rajta található személyeket és a hozzájuk kapcsolódó arcokat, maszkokat. Ezeknek a megkeresése után a detektor visszaadja a programnak az általa talált eredményt, a befoglaló geometriákat, és a hozzájuk tartozó osztályokat. Ezek után a program megkeresi a képeken található személyeket is, amely után megnézi, hogy minden emberhez tartozik-e valamilyen állapot, visel-e maszkot. Ha nem akkor hozzárendel egy nem eldönthető állapotot.

Ezek után a program kettéválik, praktikusan két külön szálra. Az egyik szálon az emberek maszkviselésének kiválogatása folyik, a másik vonalon pedig az embereknek a távolságmérése. A maszkviselés kiválogatásánál nézi a program, hogy egy szabálysértésnél az elmúlt 3 képkockában volt-e a detektált terület közelében szintén szabálysértés, így próbálja meg kiszűrni a hibás felismeréseket. Ha volt, akkor elmenti az ezeket tároló tömbbe a program a koordinátáit, emellett tovább küldi az adatbázis felé mentésre. A távolságmérés során a korábban felvázolt távolságmérő algoritmust használva dolgozik a program. Ha talál kettő egymáshoz közel tartózkodó embert, akkor ellenőrzi a maszkviselésüket is a veszély szintjének meghatározására. Ezek után ez a szál is eltárolja az adatbázisban a szerzett információkat. A program a legvégén még ellenőrzi, hogy mennyi ideje volt lementve a legutóbbi videofelvétel, és ha 1 órája vagy azon kívül, akkor a jelenlegi fájl lezárja, és új fájlba kezdi el menteni a következő felvételt.



8. ábra: A rendszer felépítésének vázlata

## Kimeneti adatok tárolásának módja



9. ára: Az adatbázis felépítési terve

1. táblázat: A videos tábla terve

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Egyedhalmaz** | **Attribútum** | | **Adattípus** | **Tulajdonság típusa** |
| **Név** | **Szöveges értelmezés** |
| videos | videoID | Azonosító kód | int | idegen azonosító |
| videoName | A videó neve | varchar(20) | leíró |
| videoDate | A videó készültének ideje(kezdőidő) | datetime | leíró |
| videoURL | A videó elérési útvonala | varchar(45) | leíró |
| videoAvailable | A videó elérhető-e | boolean | leíró |

2. táblázat: Az users tábla terve

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Egyedhalmaz** | **Attribútum** | | **Adattípus** | **Tulajdonság típusa** |
| **Név** | **Szöveges értelmezés** |
| users | userID | Azonosító kód | int | azonosító |
| username | Felhaszálónév | varchar(45) | leíró |
| password | Jelszó | varchar(45) | leíró |
| email | E-mail cím | varchar(45) | leíró |
| email\_verified\_at | Email jóváhagyás időpontja | datetime | leíró |
| token | Bejelentkezési token | varchar(45) | leíró |

3. táblázat: Az events tábla terve

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Egyedhalmaz** | **Attribútum** | | **Adattípus** | **Tulajdonság típusa** |
| **Név** | **Szöveges értelmezés** |
| events | eventID | Azonosító kód | int | azonosító |
| classID | Az esemény típusa | int | idegen kulcs |
| time | Az esemény ideje | datetime | leíró |
| frames | Az esemény képkockájának a száma | int | leíró |
| videoID | A videó azonosítója | int | leíró |
| level | A riasztás szintje | int | leíró |
| PredID | A becslés ID-je | int | leíró |

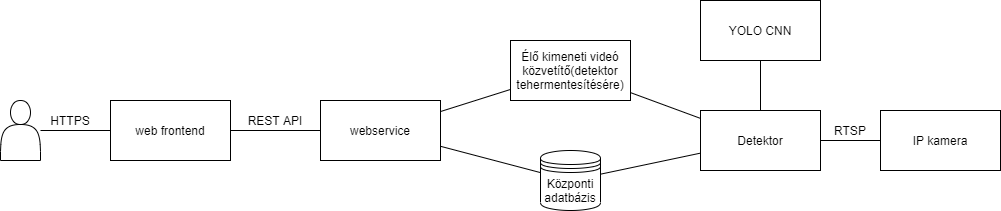
4. táblázat: A classes tábla terve

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Egyedhalmaz** | **Attribútum** | | **Adattípus** | **Tulajdonság típusa** |
| **Név** | **Szöveges értelmezés** |
| classes | classID | Azonosító kód | int | azonosító |
| classname | Esemény neve | varchar(45) | leíró |

A kimeneti adatok tárolásának módjára egy adatbázisrendszer használata szükséges. Ez az adatbázis rendszer a felső, 9. ábra és a fenti táblázatok (1-4.) szerinti táblákat és mezőket (egyedhalmazokat és attribútumokat) tárolná. Az adatok tárolását egy relációs adatbázissal tervezem megvalósítani, azok közül is a MySQL nevű relációs adatbázissal.

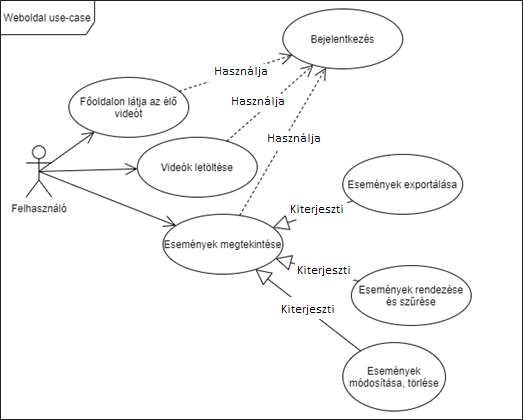
## Webes felület

A program már az előzőekben is említett webes felületen lesz elérhető a felhasználóknak. Ez a webes felület egy MVC típusú architektúrán lesz megvalósítva, REST API segítségével. [3] A frontend felől az API-n keresztül történnek meg a kérések, amit a backend lekezel, és megfelelő választ ad vissza rá. A rendszer Laravel alapokon fog működni, ő szolgálja ki az API-t, válaszol a kérésekre, állítja össze a válaszüzeneteket.



10. ábra: A rendszer felépítése webes kiszolgálóval

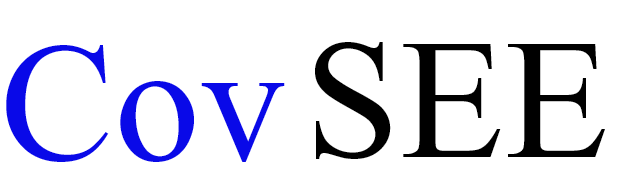
A rendszer felépítése szempontjából a korábban felvázolt detektor megkapja az IP kamerától a képkockákat egy RTSP videófolyamon keresztül. Ezután a detektor a YOLO CNN-nel összekapcsolva elvégzi a korábban felvázolt műveleteit, detektálásait. Ezek után az adatbázisba elmentve az adatokat azzal párhuzamosan az aktuálisan feldolgozott képkockát is kiadja, amit egy köztes réteg megkap. Ez a köztes réteg arra szolgál, hogy a detektor csak a detektálással foglalkozzon, a weboldal leterheltsége esetén is függetlenül tudjon működni hatékonyan a detektor. Emellett ez a köztes réteg azt is meg fogja tudni valósítani, hogy valós időben tudjunk módosítani a detektor tulajdonságain (pl. érzékelési küszöbérték, terület adatai a távolságérzékelésekhez stb.) Ez a köztes réteg, és az adatbázis fogja kiszolgálni a weboldalt, amiből az információkat kiszedve, és összecsomagolva kommunikál a web frontenddel, ami pedig a felhasználóval van kapcsolatban.



11. ábra: Felhasználó use-case diagramja

### Branding és logó

A webes felület megfelelő bemutatásához jónak találtam a programnak egy fantázianevet és egy logót választani, hogy teljes terjedelmében elkészíthető legyen a weboldal. Erre a feladatra a CoVsee fantázianevet találtam ki (a CoVid és a see szavakból összerakva), és a következő, 12-es ábrán látható logót fogom alkalmazni, amit egy SVG rajzoló és generáló programmal hoztam létre. Ezt az emblémát a weboldalon fogom felhasználni, mivel sokkal egyszerűbb ezzel elkészíteni és bemutatni a weboldalt, mintha helyettesítő szöveget vezetnék be a helyére.



12. ábra: Az oldal logója

## Használt adatkészletek

A program megfelelő futásához szükség volt betanítható adatokra. A neurális hálózat számára a betanító adat határozza meg azt, hogy mennyire lesz az a hálózat jó, mennyire lesz az a hálózat hatékony. Ezért ezeket az adatokat úgy kell összeválogatni, hogy a legjobb eredményt lehessen elérni az objektumfelismerés terén, ebből adódóan ezeknek az adatkészleteknek a felépítése és a finomhangolás nagyon nehéz. Én is előre definiált adatkészleteket fogok használni a dolgozatomban, mivel egy több akár tízezer adatból összeálló adatkészlet elkészítése és definiálása rengeteg munka és idő, és nem érné meg ilyen projekt esetében egyéni adatokkal dolgozni, ezért választottam már elérhető adatkészletek közül.

Az adatkészletek esetében a Kaggle[20] nevű adatkészlet megosztó oldalt használtam. Ez az oldal egy nagyon népszerű oldal a gépi tanulás terén, rengeteg féle adatkészlet elérhető itt, rengeteg kategóriában. A neurális hálózattal dolgozó emberek többsége is használja ezt az oldalt adatkészletek beszerzésére. Emellett nem csak adatkészleteket, hanem különböző algoritmusokat, programokat is fel lehet ide tölteni, és meg lehet osztani másokkal, tehát tulajdonképpen egy neurális hálózati „hubnak” mondható, ahol a neurális hálózatokkal foglalkozó szakemberek össze tudják gyűjteni és meg tudják osztani munkásságukat.

Az általam használt adatkészletek Andrewmvd által készített adatkészlet[5], ami 853 képből áll, és 3 különböző csoportra vannak osztva a képek: arcmaszkot visel, nem visel arcmaszkot és rosszul viseli a maszkot, illetve Maali Alharbi[26] által gyűjtött adatkészlet, ami 1420 képből áll, szintén 3 csoportra osztva. Az adatkészleteket egy Roboflow[33] nevű alkalmazás segítségével feljavítottam. A feljavítás alatt azt kell érteni, hogy az eredeti adatkészlet közül néhány képet duplikált a program, és ezeket különböző zavaró tényezőkkel (pl. zaj, forgatás, széthúzás stb.) felruházza, ezért az algoritmus ezeket az eseteket is képes megtanulni, ezáltal sokkal hibatűrőbb lesz.

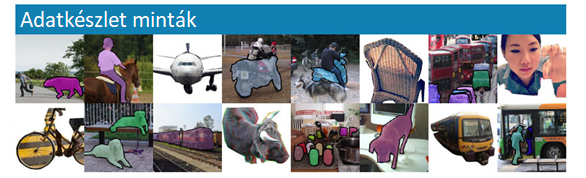
Néhány mintakép az adatkészletből, módosításokkal együtt:

A collage of people

Description automatically generated with medium confidence

13. ábra: Mintaképek az adatkészletből

A személyek felismeréséhez viszont nem egy én fogom betanítani az hálózatot, hanem már egy előre betanított neurális hálózatot veszek alapul, amit COCO [10] adatkészlet segítségével tanítottak be. A COCO egy több objektumot is definiáló adatkészlet, amit nagy vállalatok és szervezetek készítettek és a fő célja az, hogy egy nagy pontosságú és precíz neurális hálózatot lehessen létrehozni vele. Az adatkészlet 123 ezer képet tartalmaz nagy pontossággal előkészítve, ezáltal egy nagyon pontosan működő hálózat tanítható belőle.



14. ábra: Mintaképek a COCO adatkészletből [10]

# Megvalósítás

## Neurális hálózatok modern számítógépes környezetekben

A számítástechnika története során a gépi tanulás és a neurális hálók megvalósításának módja sok változáson ment keresztül.

A kezdetek kezdetén a neurális hálók futtatása kizárólag processzoron történt, különböző egyedi algoritmusokkal megvalósítva. Ez a módszer a GPU gyorsítók érkezése után rohamos léptékben változott meg, mivel a GPU jellegéből adódóan rengeteg, párhuzamos, képfeldolgozással és a neurális hálóhoz szükséges számolásokban erősebb volt, mint egy felső kategóriás CPU is.

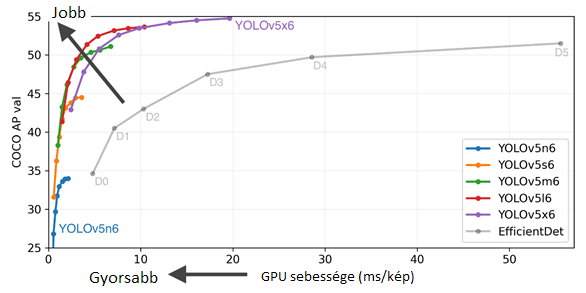
Az utóbbi időben ezeket a neurális hálós környezeteket Python vagy C++ nyelv alatt szokták főként fejleszteni. A Python mostanában népszerűbb, mivel szintaxisa is egyszerűbb, mint a C++-nak emellett sokkal több modul áll rendelkezésre, ami a gépi tanuláshoz és a neurális hálókhoz köthető. Ilyen modulok közé tartozik a Tensorflow vagy a PyTorch. Utóbbit én is fogok alkalmazni dolgozatom során.

## A YOLOv5 algoritmus

A YOLO algoritmus több iteráción is keresztül ment az idő folyamán. Első 3 verzióját az algoritmus kigondolója, Joseph Redmond implementálta főleg C++ nyelv segítségével. Ezeket pontosabban a Darknetnek nevezett programcsomagba foglalta bele, ami különböző képfeldolgozások során használt algoritmusok gyűjteménye volt.

Ez a programcsomag már akár 60,6 mAP (middle average precision – közepes átlagos pontosság) -re is képes volt, ezáltal egy nagyon hatékony algoritmusnak mondható. Azonban idővel az alkotója abbahagyta a kódnak a fejlesztését, és ezt átvette egy másik, Ultralyticsnek nevezett csoport[37], akik továbbra is optimalizálják, és fejlesztik az algoritmust ezáltal még jobbá téve azt.

A YOLOv5 algoritmus is a nevükhöz fűződik. Ez az algoritmus konkrét programkóddal és keretrendszerrel készült már el, ami által sokkal könnyebben implementálható Pythonnal készített mesterséges intelligencia környezetbe.



15. ábra: A YOLOv5 sebessége COCO adatkészlettel mérve[37]

## A detektor

A detektor felépítését tekintve a korábban már említett Ultralytics által kifejlesztett YOLOv5 hálózatot használja alap építőelemként. Az általam készített program is ennek a csomagnak az alap detektorát használja, azonban elég sok módosítással, ezért külön is vettem a programot a csomag többi „segédprogram” részétől teljesen.

A detektor eredeti beállításai szerint parancssori argumentumokkal indult volna el, és több bemeneti hálózati súlyformátumot várt volna, azonban mivel a dolgozat esetében ezek feleslegesek, ezért ezeket kitöröltem a programból, ezáltal egy sokkal átláthatóbb és gyorsabb programot készítve ezzel.

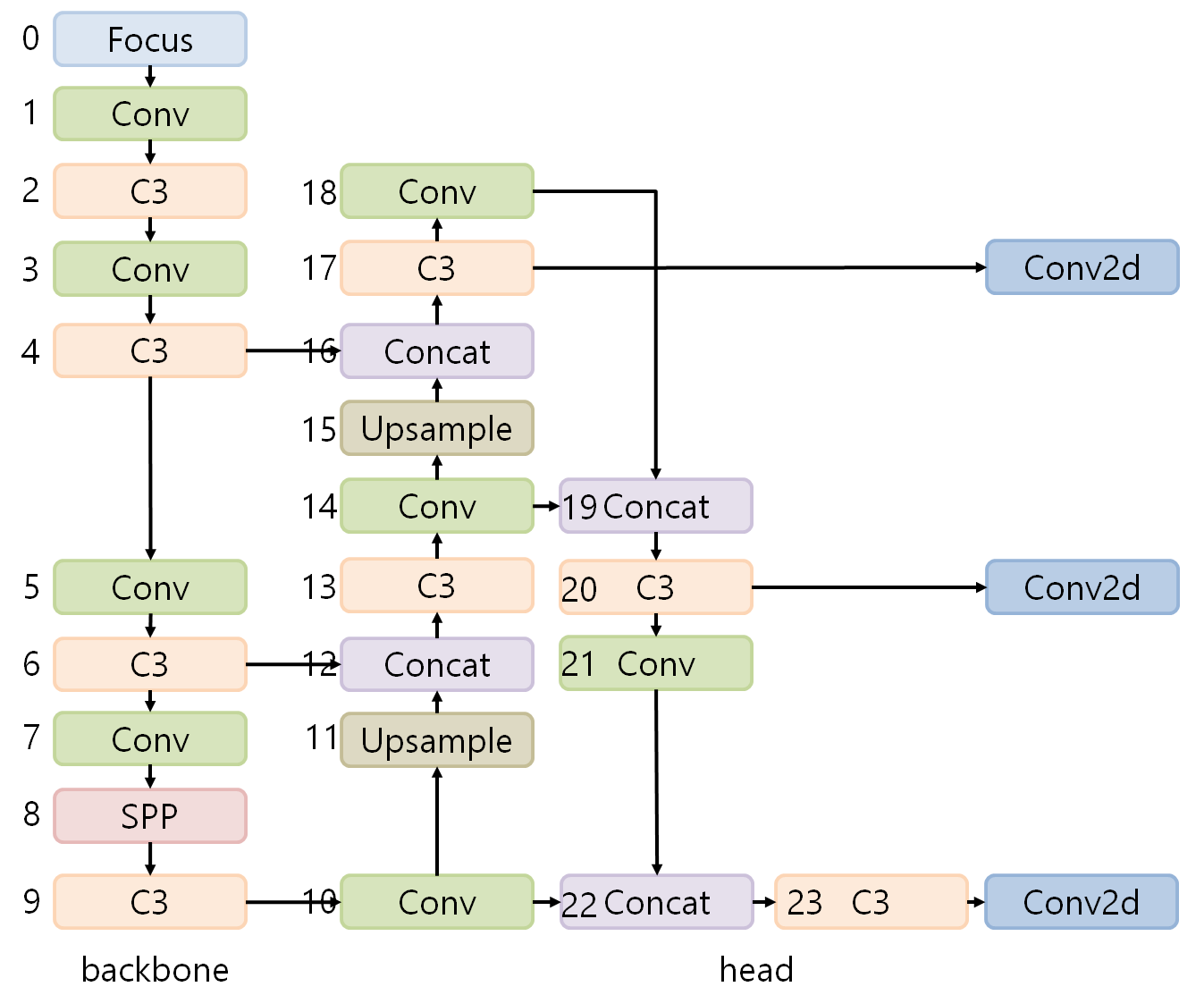
A detektor egy több lépésből működő program, ami több funkciót is lefed egyszerre. A jobb átláthatóság érdekében most levezetem a program működését. A program a **video** almappán belül a **detect.py** fájlban található.

### A detektor által használt neurális hálózat

A detektor által használt neurális hálózatnak a korábban említett YOLO hálózatot fogom használni saját adatkészletekkel betanítva. Ez a hálózat már egy korábban tesztelt, széles körben alkalmazott hálózat, úgyhogy képes kiszolgálni a programot megfelelően. A hálózat felépítését tekintve a korábban felvázolt konvolúciós neurális hálózatok közé tartozik. Az alap, hálózati algoritmusokat tekintve semmi módosítást nem végeztem rajtuk, azok az eredeti YOLOv5 féle algoritmusok, amiket le lehet tölteni az internetről és szabadon fel lehet használni.[37]

A hálózathoz használt algoritmus Python programozási nyelv alatt készült, PyTorch keretrendszer segítségével. A PyTorch egy Mesterséges Intelligenciát támogató keretrendszer, ami arra szolgál, hogy a különböző gépi tanulási, mesterséges intelligenciát fejlesztő személyeknek a munkáját meggyorsítsa. A PyTorch abban segít, hogy a leggyakrabban használt műveleteket, algoritmusokat meghívható függvényekké alakítja, ezáltal a fejlesztőnek nem szükséges minden művelethez újra megírnia ugyanazt a műveletet, elég csak meghívni azt a függvényt, amilyen műveletet el kíván végezni és ezáltal sokkal gyorsabb lesz a fejlesztés is.

A program által használt neurális hálózatnak a felépítését fogom a most következő részben szemléltetni. Az alábbi ábrán grafikusan is látszódik a hálózatnak a felépítése, azonban ezen ábra megértéséhez szükséges a rajta megtalálható műveleteknek a magyarázata.



16. ábra: A YOLOv5 hálózatok felépítése [29]

* **Conv**

A **Conv** műveletek általános konvolúciókat jelölnek, amik egy bemeneti adatból egy előre megadott konvolúciós kernel segítségével kimeneti adatot generálnak. A konvolúciónak a működését a korábbi, elméleti részben már kifejtettem, ezért azt a jelenlegi részben már nem tárgyalnám tovább. A konvolúciók mellett még ebben a rétegben történik egy kötegnormalizálás (batch normalization), és egy SiLU aktiváció is.

* **C3**

A **C3** jelölésű műveletek tömören összefoglalva több konvolúciós műveletnek és egy CSP[39] műveletnek az összefogásai. A C3 műveleteknél 3 darab konvolúció történik (innen a hármas szám is), amik egy megadott sorrend szerint követik egymást. A sorrend pedig úgy írható le, hogy a bemeneti képen az algoritmus elvégez egy konvolóciós műveletet, ezen konvolúciós művelet kimenetét a program továbbítja a CSP Bottleneck algoritmusnak, mint bemeneti kép, ami elvégzi rajta a szükséges műveleteket, amely műveletek végén a kimeneti képet a program visszakapja, és az eredeti, C3 műveletbe beérkező kép egy másik, szintén konvolúción átfuttatott adataival összefűzi (Concat). Ezek után az összefűzött képen még egyszer végrehajt egy konvolúciót, és az ebből kijövő képet szolgáltatja tovább a kimenetén.

Diagram

Description automatically generated

17. ábra: A C3 algoritmus felépítése

class C3(nn.Module):

    # CSP Bottleneck with 3 convolutions

    def \_\_init\_\_(self, c1, c2, n=1, shortcut=True, g=1, e=0.5):  # ch\_in, ch\_out, number, shortcut, groups, expansion

        super().\_\_init\_\_()

        c\_ = int(c2 \* e)  # hidden channels

        self.cv1 = Conv(c1, c\_, 1, 1)

        self.cv2 = Conv(c1, c\_, 1, 1)

        self.cv3 = Conv(2 \* c\_, c2, 1)  # act=FReLU(c2)

        self.m = nn.Sequential(\*(Bottleneck(c\_, c\_, shortcut, g, e=1.0) for \_ in range(n)))

        # self.m = nn.Sequential(\*[CrossConv(c\_, c\_, 3, 1, g, 1.0, shortcut) for \_ in range(n)])

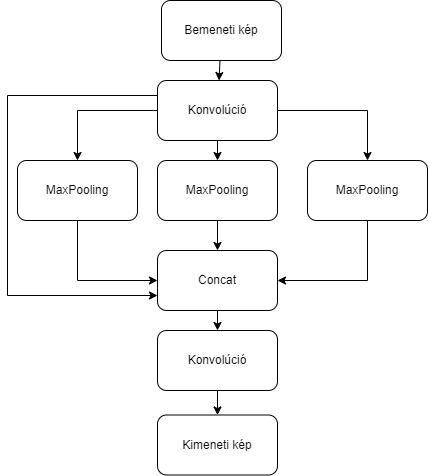
    def forward(self, x):

        return self.cv3(torch.cat((self.m(self.cv1(x)), self.cv2(x)), dim=1))

1. kódrészlet: A C3 algoritmus szoftveres megvalósítása

* **SPP**

Az SPP a Spatial Pyramid Poolingnak, magyarul Térbeli Piramisok Összevonásának a rövidítése. Ez a réteg arra szolgál, hogy a hálózat még hatékonyabban működjön és minél jobb eredményeket érjen el. A réteg fő feladatai közé tartozik, hogy a neurális hálózatok egyik fő problémáját, a bemeneti képeknek a fix méretét kiküszöböljék. Ezt az algoritmus úgy valósítja meg, hogy a bemeneti képen, egy konvolúció után több összevonást (pontosabban maxpoolingokat) végez a képen párhuzamosan, és ezeket az összevonásokat a végén összefűzi a kezdeti konvolúcióval együtt.



18. ábra: Az SPP algoritmus felépítése

class SPPF(nn.Module):

    # Spatial Pyramid Pooling - Fast (SPPF) layer for YOLOv5 by Glenn Jocher

    def \_\_init\_\_(self, c1, c2, k=5):  # equivalent to SPP(k=(5, 9, 13))

        super().\_\_init\_\_()

        c\_ = c1 // 2  # hidden channels

        self.cv1 = Conv(c1, c\_, 1, 1)

        self.cv2 = Conv(c\_ \* 4, c2, 1, 1)

        self.m = nn.MaxPool2d(kernel\_size=k, stride=1, padding=k // 2)

    def forward(self, x):

        x = self.cv1(x)

        with warnings.catch\_warnings():

            warnings.simplefilter('ignore')  # suppress torch 1.9.0 max\_pool2d() warning

            y1 = self.m(x)

            y2 = self.m(y1)

            return self.cv2(torch.cat([x, y1, y2, self.m(y2)], 1))

1. kódrészlet: Az SPP algoritmus szoftveres megvalósítása

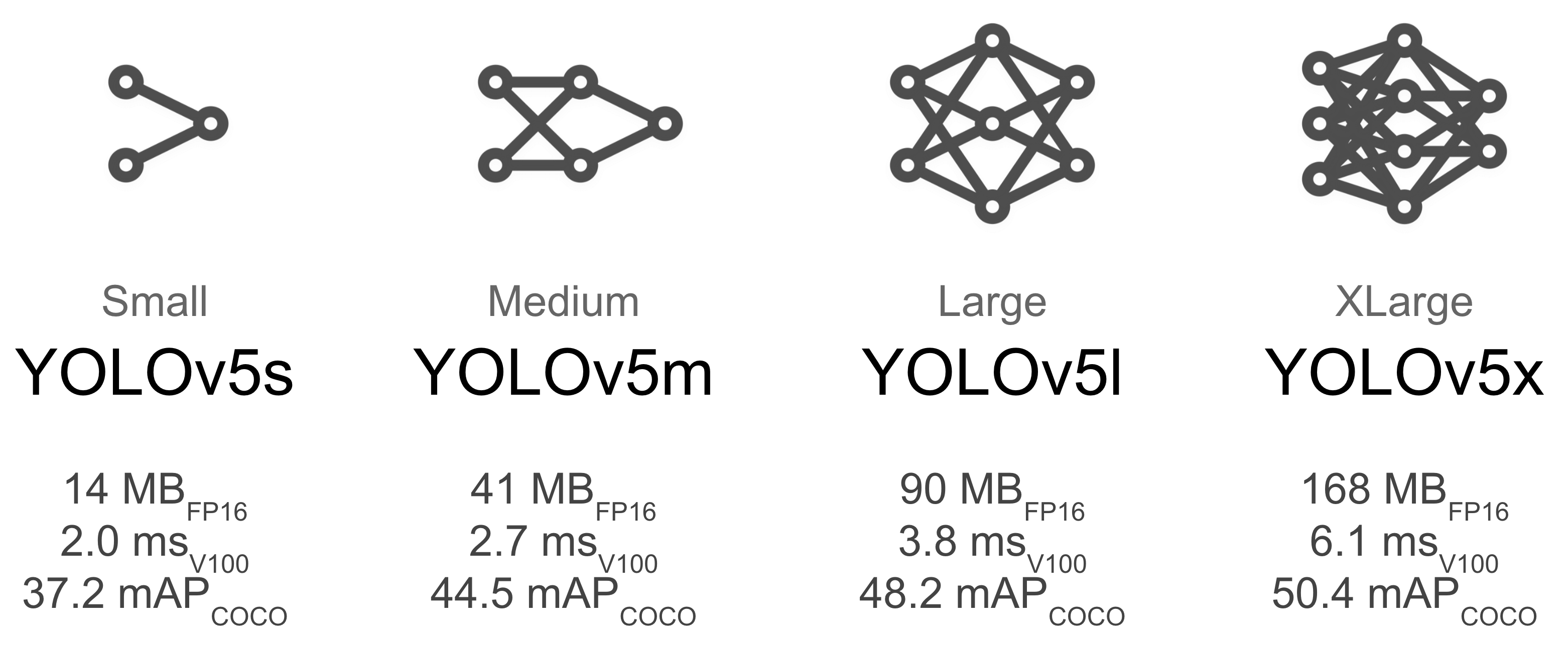
* **Upsample**

Az upsample vagy felskálázás művelet jelentését tekintve a bemenetnek kapott képnek a megnövelése egy megadott arányszám alapján. A megnövelés azt jelenti, hogy pl. egy -ös méretű képből csinálunk 2x-es nagyítással -as méretű képet.

Azonban a nagyítás közben belefutunk egy olyan problémába, hogy a képpontok helyzetét meg tudjuk változtatni, azonban a képpontok között így üres helyek maradnak. Ezeket a helyeket valamivel szükséges kitölteni, hogy értelmezhető legyen a képünk. Az üres helyek kitöltésére különböző algoritmusok és megoldások léteznek, mint például a legközelebbi szomszéd elve, bilineáris, bicubic transzformációk. A dolgozatomban YOLO hálózatok a legközelebbi szomszéd elvét használják a felskálázások során, ami azt jelenti, hogy a kívánt képpontot úgy határozzuk meg, hogy a körülvevő képpontok közül annak az értékét vesszük, amelyik a képponthoz legközelebb esik. Ezt a program a PyTorch keretrendszer Upsample függvényével valósítja meg.

* **A neurális hálózat tanítása**

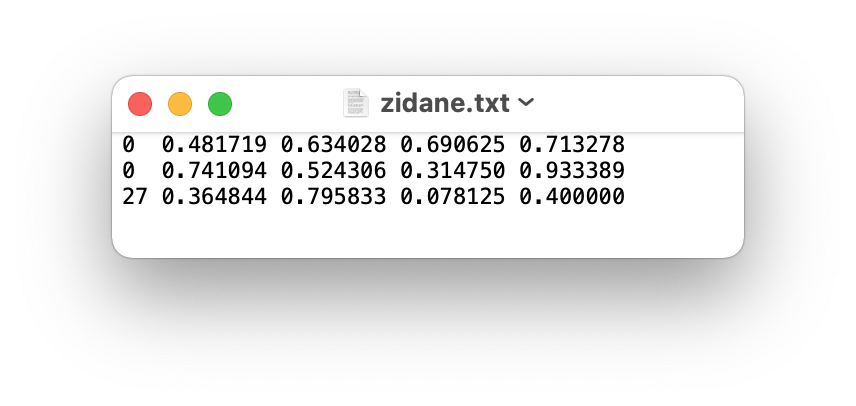
A detektort, mint minden neurális hálózatot tanítani szükséges, ha specifikus felismerési feladatokra szeretnénk használni. A betanítás során osztályozott képeket szolgáltatunk a hálózatunk számára, ami ezeken a képeken megkeresi a jellemzőket, a számára szükséges információkat, és ez alapján készíti el a neurális hálóban szereplő súlyokat. A betanítást a munkámban felhasznált YOLOv5 kódhoz tartozó tanító algoritmussal végeztem el. Ez a tanító algoritmus többféle méretű és tulajdonságú hálózatot képes betanítani, amivel különböző eszközre optimalizált neurális hálózatot lehet felépíteni. Ezek a hálózatok sok esetben már egy előzőleg elkészített alapponttól indulnak, egy úgynevezett checkpointtól, ami azt jelenti, hogy előre be van beléjük táplálva már egy létező adathalmaz, és ezt tanítjuk újra, így még hatékonyabb tud lenni a tanulási folyamat. Természetesen lehet egy teljesen új hálózatot is felépíteni, azonban rengeteg adatra lesz akkor szükség a hálózat optimális működéséhez.



19. ábra: A YOLOv5 detektor néhány betanítható hálózatmérete[37]

Az ábrán említett hálózatok alap felépítése mindegyik esetében azonos, méreteik tekintetében változnak csak, mint a hálózat mélysége és kiterjedése. Ezáltal kevesebb erőforrással rendelkező rendszereken is lehet futtatni a modelleket, ha kisebb hálózatot választunk.

A betanítás megkezdése előtt azonban még szükséges előkészíteni az adatkészleteket. Nem elég csak a képeket betáplálni a hálózatba, azokat természetesen címkézni is szükséges a betáplálás előtt, különben az algoritmus nem is fogja tudni mit keressen. Ezt úgy lehetséges elérni, hogy minden képhez szükséges egy vele azonos nevű txt kiterjesztésű fájlt létrehozni, amiben eltároljuk tabulátorokkal elválasztva a képen található objektumok osztályát (első oszlop), és befoglaló geometriáit. A befoglaló geometriák megadását úgy kell elvégezni, hogy hogy a kép eredeti méretét leredukáljuk egy 0 és 1 közötti skálára (praktikusan minden koordinátát elosztjuk a kép szélességével, avagy magasságával, annak függvényében, hogy x vagy y koordinátáról van szó), és ezen a skálán adjuk meg a befoglaló geometria bal felső pontjának x és y koordinátáit, emellett a szélességet és a magasságot.



20. ábra: A képeket leíró fájlok mintafelépítése[37]

A képeket tartalmazó adatkészletet azonban úgy kell összeállítani, hogy szükséges készíteni egy tanító és egy validációs adathalmazt. Erre a két adathalmazra azért van szükség, mivel az algoritmus teljesen csatolt jellegéből adódóan képes a túltanulásra, ami azt jelenti, hogy túlságosan képes alkalmazkodni egy kijelölt mintára, a betanító adatkészletre, és ezzel az általános felismerésekben sokkal rosszabb teljesítményt tud nyújtani. Ezért szükséges készíteni egy validációs adatkészletet is, amit arra használ a program, hogy a betanított súlyok alapján elkészített hálózatot leteszteli ezekkel a validációs képekkel, és megkeresi rajta azokat az eseteket, amikor vagy hamis pozitív, vagy hamis negatív eredményt ért el a hálózat, és ezek alapján különböző matematikai és logikai műveletekkel próbálja még jobban optimalizálni a hálózatnak a működését.

Miután megtörtént az adathalmaznak az előkészítése és a hálózatnak a kiválasztása, azután kezdődhet a tanulási folyamatnak az elkezdése. A tanulási folyamat egy ciklikusan működő folyamat, ami azt csinálja, hogy a képeket betáplálja a hálózatba, betanítja a súlyokat, ellenőrzi a súlyokat a validációs adatkészlettel és ezek alapján kezdi újra a betáplálást.

A betanítás során néhány paramétert azonban szükséges megadni az algoritmusnak, hogy hogyan végezze a betanítást, milyen tulajdonságokkal fusson. A fontosabb paraméterek közé tartoznak például a batch size, vagyis kötegméret. A kötegméret egy darabszámot ad meg, ami azért szükséges, hogy hatékonyabb legyen a tanulási folyamat. A kötegelt tanításoknál a hálózatba egy művelet alatt nem az összes képet tápláljuk be, hanem csak 1 köteg darabszámával egyenlő képet egyszerre, ezeket végig futtatjuk a hálózaton, majd következik a következő köteg. Ha a program végzett az összes köteggel, tehát az összes tanító adathalmazban lévő képet betáplálta a hálózatba, és utána lefuttatta a validációt a kiejtésekkel, akkor telt le 1 epoch. Az epochot úgy lehetne a legjobban definiálni, hogy 1 iteráció, ami alatt az egész tanító és validációs adathalmaz végigfut egyszer a hálózaton. Ezt az epoch számot érdemes magasan tartani, mert így több iterációt tud a program elvégezni, és jobban tudja optimalizálni a súlyokat is.

Ezeket a paramétereket a programom betanításakor a következő értékekre állítottam:

* Epochok szám: 150
* Kötegméret: 8
* Hálózat típusa: YOLOv5m

A hálózat betanításának eredményei a következőképpen alakultak:

TANÍTÁS ALATT

21. ábra: Megbízhatósági mátrix

TANÍTÁS ALATT

22. ábra: A hálózat teljesítménye az epochok előrehaladásával

### A detektor felépítése

A detektor programjában első lépésben a rendszerszintű csomagimportálások történnek meg. A csomagok között található idő és dátumkezelés (date és datetime) alfolyamat kezelés(subprocess), argumentum értelmező és rendszercsomagok (argparse, os, sys, pathlib), és a program szempontjából fontos képi és neurális hálózatos algoritmusok (opencv, numpy, pytorch).

Ezt követően a program több, különböző YOLOv5-höz tartozó csomagot is beimportál, azonban, hogy ezeket a program be tudja importálni szükséges átlépni a YOLOv5 algoritmusokat tartalmazó mappába. A programot úgy terveztem, hogy ezek a segédprogramok a detektort tartalmazó főkönyvtáron belül egy yolov5 mappában legyenek eltárolva. Ezekről a segédprogramokról készítettem is egy github forkot, hogy esetleg verziófrissítések esetén ne az legyen, hogy a program már el sem indítható, mert sehol nem találhatóak már meg az általa hivatkozott függvények.

Az importok után még történik egy mysql adatbázis kapcsolódás definíció, mivel a program fog adatbázistáblákba is írni a mysql connector csomag segítségével, ezért szükséges egy kapcsolódást létrehozni a táblázattal. Ezek a kapcsolódási adatok csak a programban levő változó módosításával módosíthatóak, de mivel ez nem lesz egy gyakran megváltozó érték, ezért nem is probléma.

Ezek után történik meg a fő függvénybe történő belépés. A függvényben több állítható paraméter is van, az esetleges különböző program beállítási funkciók módosítása érdekében. Ezekhez ritkán kell nyúlni, azonban nem probléma, ha ezek módosíthatóak kicsi belenyúlás közepette, mert lehetnek olyan esetek, amikor finom hangolni kell a detektort, vagy módosítani rajta dolgokat.

A függvénybe való belépést követően történik egy bejövő forrás vizsgálat, ami csak annyit csinál, hogy ellenőrzi, hogy a forrás az streaming vagy videó típusú-e, és ez alapján feltölti a webcam változót egy logikai értékkel.

Ezek után megtörténik a felismerő eszközök (itt grafikus kártya, vagy processzor, attól függ, melyik áll rendelkezésre) inicializálása, és a modell, pontosabban a súlyok betöltése a programba és a hálózatba. Ezt a kódrészletet nagyban módosítottam és egyszerűsítettem, mivel az eredeti detektorban többféle súlyokat is be lehetett táplálni, azonban a program esetében csak egy féle súlykiterjesztéssel fog dolgozni a program, ezáltal nem szükséges a többi súlykiterjesztésnek az ellenőrzése.

    # CPU/Videókártya inicializálása

    device = select\_device(device)

    half &= device.type != 'cpu'  # half precision only supported on CUDA

    # Modell betöltése

    w = str(weights[0] if isinstance(weights, list) else weights)

    classify, suffix, suffixes = False, Path(w).suffix.lower(), ['.pt', '']

    check\_suffix(w, suffixes)  # check weights have acceptable suffix

    stride, names = 64, [f'class{i}' for i in range(1000)]  # assign defaults

    model = attempt\_load(weights, map\_location=device)

    stride = int(model.stride.max())  # model stride

    names = model.module.names if hasattr(model, 'module') else model.names  # get class names

    if half:

        model.half()  # to FP16

    imgsz = check\_img\_size(imgsz, s=stride)  # check image size

1. kódrészélet: Az eszközök inicializálása és a súlyok betöltése

A modell betöltése és inicializálása után pedig a bemeneti képeket/videókat tölti be a program egy objektumba, amit később el lehet érni, és folyamatosan le lehet kérni belőle a következő képkockát.

A modell betöltése elindítása után következne a képkockák betáplálásba a hálózatba, azonban még előtte az algoritmus elmenti a jelenlegi videó adatait az SQL videos táblájába, hogy elkezdte a videó feldolgozását, de még nem végzett vele. Emellett lenulláz a program még pár technológiai változót is, ami arra szolgál, hogy meg lehessen mondani, hogy egymást követő 4 képkockából 3 képkocka esetén volt-e észlelés kis távolságon belül, ha igen akkor az tulajdonképpen egy eseménynek vehető, mivel valószínűsíthetjük, hogy az 1 embernek az arcát ismerte fel maszk nélkül.

Az észlelések összekötése úgy történik, hogy 4 képkockán max 10 észlelés lehet minden képkockán, minden észleléshez tartozik 4 adat, x,y koordináta egy prediction ID, ami összeköti az észleléseket és egy frame number, hogy hány képkockán keresztül tartott az esemény(ez növekszik mindig 1-el ha 1-el több észlelés található)

### Maszkok feldolgozása a hálózat által

A következő lépés a definíciók után a képkockák betáplálása a hálózatba és a detektor végrehajtása. Ezek a képkockák egy növekményes(for) ciklusban táplálódnak be, aminél minden egyes ciklusvégrehajtás 1-1 képkockát jelent. A képkockák feldolgozása elején megnézzük, hogy legalább már megy-e 4 képkockája a felvétel, ha igen, akkor a korábbi észleléseket tartalmazó tömböt 1 képkockányival eltoljuk „hátrafelé”, ezzel helyet csinálva a most feldolgozandó, negyedik képkockának. A mozgatás után betöltjük a képkockát a megfelelő feldolgozóba, átalakítjuk megfelelő formátummá (mivel jelenleg minden képpont 0 és 255 közötti értéket vehet fel, azonban a program 0 és 1 közötti értékekkel dolgozik, ezért ezeket a számokat át kell alakítani lebegőpontos számmá, pontosságnak megfelelően 16 vagy 32 bites lebegőpontos számmá) és ez után normalizálni kell az értékeket egy 0-1 tartományon. Ezt úgy érjük el, hogy minden értéket elosztunk 255-tel.

Ezek után történik meg a képkocka hálózatba betáplálása. Ez már egy előre elkészített függvény segítségével történik, a model függvénnyel, amibe betápláljuk a képet, és még pár egyéb információt, ami a modell szempontjából lényeges. Ez a modell megadja a becsült osztályokat és objektumokat, amikből még egy nem-maximum vágás segítségével meg kell határozni a számunkra releváns és hasznos felismeréseket.

1. kódrészlet: A képek betáplálása a hálózatba és detektálás

    # a hálózat alkalmazása a képkockákon

    if device.type != 'cpu':

        model(torch.zeros(1, 3, \*imgsz).to(device).type\_as(next(model.parameters())))  # run once

    for path, img, im0s, vid\_cap, s in dataset:

        frameCounter += 1

        #észlelések elmozgatása 1 képkockával az észleléseket tartalmazó vektorban

        if(frameCounter >= 4):

            predictionList[0]=predictionList[1]

            predictionList[1]=predictionList[2]

            predictionList[2]=predictionList[3]

            predictionList[3]=np.zeros((10,4))

        #A képek betöltése és átalakítása

        img = torch.from\_numpy(img).to(device)

        img = img.half() if half else img.float()  # uint8 to fp16/32

        img /= 255.0  # 0 - 255 to 0.0 - 1.0

        if len(img.shape) == 3:

            img = img[None]  # expand for batch dim

        # A hálózatba betáplált képkocka végigfuttatása

        visualize = increment\_path(save\_dir / Path(path).stem, mkdir=True) if visualize else False

        pred = model(img, augment=augment, visualize=visualize)[0]

        # Nem-maximum vágás (aktiváció)

        pred = non\_max\_suppression(pred, conf\_thres, iou\_thres, classes, agnostic\_nms, max\_det=max\_det)

A betáplálás és a detektálás után még azonban a program nem végzett, az eredményeket szükséges kiértékelni, és egy SQL táblába lementeni az eredményeket, emellett még a videókat is mentenie kell.

### Emberek és távolságuk feldolgozása a program által

Az emberek és távolságuk feldolgozásához szükséges algoritmusokat már a dolgozat elméleti részében korábban levezettem. Ezen algoritmusok megvalósításához a programon belül is hasonló elvek mentén készült el. Az emberek felismeréséhez a korábban már többször is bemutatott, YOLOv5 algoritmust fogom használni előre betanított súlyokkal, amely súlyokat a COCO adatkészlet segítségével tanítottak be. Ezen hálózat azonban annak ellenére, hogy több osztályt képes felismerni számunkra csak és kizárólag 1 osztálynak a felismerése volt a lényeges, ez pedig az emberek osztálya. Ezt úgy érjük el, hogy a felismeréskor kizárólag a 0-s sorszámú osztályt vesszük figyelembe. A neurális háló betöltéséhez és inicializáláshoz ugyanazokat a függvényeket és a metódusokat használtam, mint az eddigiekben, a maszkok felismerésénél. Ezért a program ezen lépéseit nem is részletezem, mivel teljesen felesleges duplikálás lenne csak.

A rendszer szempontjából fontos algoritmusok és műveletek inkább a perspektíva transzformációk és az emberek távolságának meghatározása. A perspektíva transzformációkat az OpenCV programcsomag segítségével oldottam meg. A perspektíva transzformációhoz szükséges ismerni a helyszínnek a tulajdonságait. Szükséges a képeken kiválasztani azt a területet, amit transzformálni akarunk. A területet úgy kell kiválasztani, hogy a való életben az egy téglalapot adjon. Ennek a téglalapnak a 4 sarkát be kell jelölni a képen, erre használható a mouse.py fájl, amit egy internetes forrásból szereztem. [7] A program megadja a 4 pontnak a koordinátáját, ami nekünk szükséges a perspektíva transzformációhoz. A program ezek után lementi az adatokat egy config.yaml fájlba Ezt ezek után a videos mappába config.yaml néven szükséges elmenteni, és ezt a program behivatkozza. A detektoron belül is a megjelölt forrás alapján készítettem el a perspektívatranszformáció függvényeit is. Azonban ahhoz, hogy a szoftver valós távolságot tudjon mérni, szükséges megadni egy a való életben téglalap 2, egymás melletti oldalának kezdő és végpontjait, hogy a program arányszámokat számolva tudjon távolságot számolni. Az emberek közötti távolság detektálásához pedig kigyűjtöttem az embereknek a befoglaló geometriáit, megkerestem a geometriák alsó-középső pontját, és ezeket a pontokat eltároltam egy listába. Ezek után ebből a listából minden elemet végig ellenőrizve kiszámolom a pontok euklideszi távolságát, és a szabálysértőket eltárolom egy listába. Ezek után pedig csak az elemek ellenőrzése, és adatbázisba írása történik.

def getDistance(matrix,p1,p2,d\_w,d\_h):

    transformed\_downoids\_p1 = compute\_point\_perspective\_transformation(matrix,(p1[0],p1[1]))

    transformed\_downoids\_p2 = compute\_point\_perspective\_transformation(matrix,(p2[0],p2[1]))

    h = abs(p2[1]-p1[1])

    w = abs(p2[0]-p1[0])

    dis\_w = float((w/d\_w)\*200)

    dis\_h = float((h/d\_h)\*200)

    dist = math.sqrt(math.pow((dis\_h),2) + math.pow((dis\_w),2))

    return dist

1. kódrészlet: Az emberek közötti távolság meghatározása

### Az eredmények feldolgozása

Az eredmények feldolgozása is természetesen képkockánként történik, máshogy hosszabb is meg bonyolultabb is lenne megoldani. Így pedig csak minden detektálás után végigmegy az összes becslésen a program.

A feldolgozás elején a program átméretezi a befoglaló geometriákat a kimeneti értékeknek megfelelően, hogy a képen feldolgozhatóak legyenek ezek a geometriák. Ezek után az események kiértékelése történik annak megfelelően, hogy ha az utolsó 4 képkockán legalább 3 összefüggő képkockán keresztül egy bizonyos pixelértéken belül volt a befoglaló geometriák kezdőpontjának (bal felső pontjának) a távolsága (azért elég a kezdőpont, mivel az egy arc esetén nem sokat változik még mozgás esetén sem, ezért elég jó referenciapont), akkor az egy észlelésnek tekinthető, ami egy közös prediction ID alá is tartozik. Ezek után elmenti a képkockához tartozó észleléseket a program (max 10-et), és utána még az adatbázisba azokat az értékeket, ahol észlelés volt található, vagy ha már található olyan prediction id-val egy észlelés akkor csak módosítjuk annak megfelelően, hogy mik voltak az esemény részletei.

Ezek után az észlelt képkockára megtörténik még a befoglaló geometriák felrajzolása, majd ezeknek a képkockáknak a közvetítése és lementése videófelvételekbe.

A képkockák közvetítésére és a videófájlok lementésére egy népszerű, videófeldolgozáshoz és kódoláshoz használt alkalmazást, az FFmpeg-et[15] használja a program. Az FFmpeg program segítségével a program a beérkező képkockákból képes összefűzni egy felvételt, és ezt a felvételt továbbadni videókimenetként, és RTMP közvetítésként a weboldal felé.

## A kiszolgáló weboldal

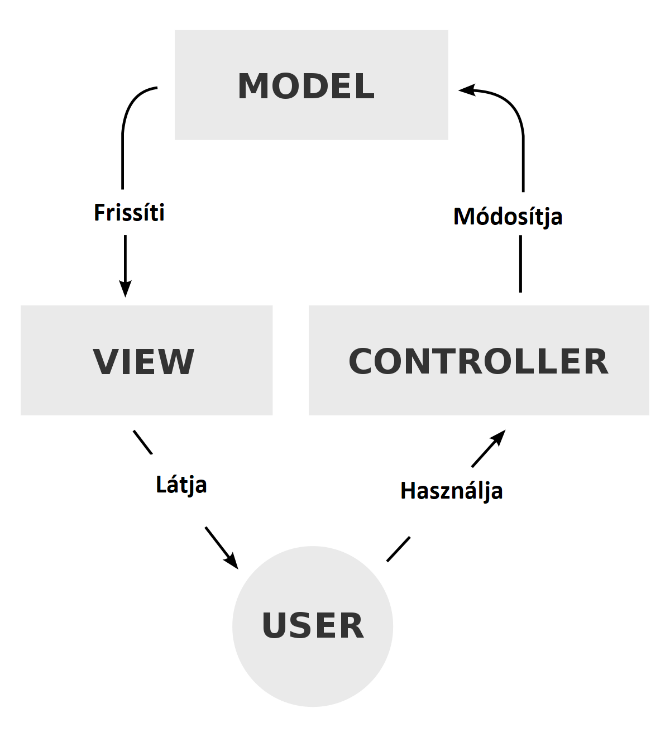
A rendszer fő interakciós eleme a kiszolgáló weboldal lesz. Ez a weboldal tartja a kapcsolatot a felhasználó és a háttérben futó detektorrendszer között. Optimális esetben a felhasználónak csak ezt kell használnia a rendszer működése során, a háttérben húzódó alkalmazáshoz nem szükséges hozzáférnie.

Ezt a weboldalt saját preferenciámból adódóan PHP nyelven kezdtem el megvalósítani. Azért választottam a PHP nyelvet, mivel ebben már van némi tapasztalatom, és a PHP egyik remek keretrendszerével, a Laravellel[22] már van tapasztalatom. Emellett a web egyik legnépszerűbb programozási nyelvről beszélünk, még ha a JavaScript, Rust meg egyéb más programozási nyelvek kezdik kiszorítani, még mindig rengeteg weboldal használja dinamikus megjelenítéshez. A PHP mellett természetesen HTML, CSS és JavaScript nyelveket is használok a weboldal frontendjének elkészítéséhez, a Laravel egyedi, blade template nyelve mellett[23].

### Laravel keretrendszer

A Laravel egy közösség által fejlesztett PHP keretrendszer, ami az egyszerű, gyors, hatékony weboldalkészítésre fókuszál, ami aztán egy robosztus, jól skálázható környezetben működik. A keretrendszer rengeteg különböző elemből áll, amik mintegy építőkockaként működve képesek a webfejlesztés során segíteni a fejlesztőknek a munkáját és gyorsítani a fejlesztés folyamatát. Dolgozatomban is ezeknek az „építőkockáknak” a segítségével fogom összeállítani a felhasználóknak szánt weboldalt.

A Laravel keretrendszer egyik hasznos eleme az MCV szemlélete. Az MVC szemlélet szerint a weboldal során a Model (adatbázis és adatok) és a View (frontend környezet, amit a felhasználó lát) elkülönülő réteget a Controller köti össze, aminek az előnye közé tartozik, hogy sok esetben könnyebb átlátni és elválasztani a különböző programozási szinteket, és nem keveredik a frontend környezetbe adatmanipuláció, hanem mindenről egy backend kontroller gondoskodik.



23. ábra: Az MVC architektúra felépítése [32]

A Laravel keretrendszer másik hasznos tulajdonsága a gyors prototípus készítés. A különböző alapelemekre és már előre meg vannak írva a függvények és a minták, ezeket akár egy egyszerű függvényhíváson keresztül meg lehet hívni, és ezáltal az alap funkciók (mint pl. adatbázis elérés, REST API válaszok, middleware szolgáltatások stb.) nagyon gyorsan elérhetőek és felhasználhatóak, nem kell mindent megírni a nulláról.

### Regisztráció és bejelentkezés

A weboldal jellegéből adódóan védendő, privát információkat tartalmaz. Ezeket az adatokat nem lenne szerencsés eset, ha más, külső emberek is látnák az erre jogosult személyeken kívül, azonban mivel webes alkalmazásról van szó, elég nagy az esély, hogy más, külsős személyek is esetleg elérhetik a weboldalt (kivéve, ha egy elkülönített hálózatban van). Ezért a weboldal tervezése során szükséges valamilyen hitelesítési folyamatot használni, hogy biztos a megfelelő személy fér-e hozzá a szükséges információkhoz.

Ezt a hitelesítést egy egyszerű, felhasználónév, jelszó kombinációs bejelentkezéssel fogom a dolgozatomban megoldani. Azonban a modern szoftverfejlesztés során egy ilyen egyszerű rendszer felépítésének is rengeteg buktatója, biztonsági rése lehet. Ezek közé a rések közé tartozik az egyik legismertebb, SQL injectionnak nevezett támadási forma. Ezen támadási forma során a beviteli mezőkbe olyan futtatható SQL kódot visznek fel speciális formátumban, ami bizonyos műveleteket képes elvégezni az adatbázisban (Akár le is lehet kérdezni az egész felhasználók táblát is vele pl.). Ezek a támadások nagyon veszélyesek, ezért nem is próbálkoztam saját megoldást kifejleszteni, hanem inkább hagyatkoztam a Laravel keretrendszerre. A Laravel keretrendszerben megtalálhatóak előre megírt authentikációs szolgáltatások, amik már az ismert sebezhetőségeket kikerülve kerültek megírásra.

Dolgozatomban is az egyik ilyen szolgáltatást fogom használni. Pontosabban nem csak egy authentikációs szolgáltatást, hanem egy kompletten elkészített alap beléptető és regisztrációs oldalt és oldal kiindulási alapot. Ez a csomag Laravel Jetstream[18] néven érhető el, és egy remek alapot nyújt az alkalmazás elkészítéséhez.

A csomag a Laravel Fortify authentikációs szolgáltatásokat használja, ami szolgáltatja a bejelentkezési, regisztrációs, és még egyéb (pl. jelszóemlékeztető) szolgáltatások gerincét, alapfüggvényeit. Emellett a csomag még tartalmaz egy előre elkészített minta weboldal szerkezetet, amit a dolgozatomban is felhasználtam. Ez az oldalszerkezet egy Laravel Livewire keretrendszer alapján készült, amivel lehetőség nyílik a backend és a frontend keretrendszer összekötésére. A Laravel Livewire rendszere egy dinamikus, DOM manipulációt elősegítő eszköz, ami aktív kapcsolatban van a weboldal backend környezetével. A Livewire segítségével meg lehet oldani, hogy a backenden legenerált információt át lehessen emelni frontend felületre az információ felesleges konvertálásával. Így pl. lehetséges, hogy egy háttérben elvégzett lekérdezés összes adatát átvigyük a frontendet kiszolgáló fájlba, és az oldal legenerálása és kirajzolása során módosítsuk a DOM-ot a bejövő információknak megfelelően.

A Jetstream alapszolgáltatások esetén nem sok dolgon módosítottam, a regisztrációs linket vettem ki egyedül, hogy külső személyek ne tudjanak regisztrálni, így nem is fognak hozzáférni az adatokhoz. Ezt a **web** weboldalt tartalmazó mappán belül **app/http/Controllers/Auth/RegisteredUserController.php** nevű fájl módosításával értem el, ahol a create() függvényen belül az oldal elérését a register view helyett a login viewre írtam át, így a felhasználó még ha be is írja az URL-t hogy ő az example.com/register oldalra akar lépni, akkor is a login felületet hozza be.

### Videók és Események megtekintése

A videók és az események megtekintéséül szolgáló oldalaknak az alap felépítése megegyezik, mivel mind a kettő oldal azonos funkciót lát el, csak más paraméterekkel. Ezáltal az oldalakhoz írt programkód is nagy hasonlóságokat mutat egymással, egy alap mintára épülnek.

Ezeknek az oldalaknak a megvalósítása a **web** mappán belül a **resources/views** mappában történt, ami közül is a **vids.blade.php** és az **events.blade.php** fájlokat jelenti

Ezek az oldalak egy egyszerű táblázatot jelenítenek meg, amiben megtalálhatóak az adatbázis különböző rekordjainak emberek számára olvasható módon történő kiírásai.

A videók megtekintésére szolgáló oldalon a felhasználó számára a következő adatok kerülnek kijelzésre:

* A videó ID-ja
* A videó neve (A név a következő formulából áll össze: ÉÉÉÉHHNNÓÓPPMM)
* A videó készítésének ideje
* A videó elérhetősége (megtörtént már a feldolgozása a rendszer által?)
* 3 funkciógomb a videó megtekintéséhez, letöltéséhez és törléséhhez

Az események megtekintésekor pedig a következő adatok:

* Az esemény ID-ja
* Az esemény típusa (kontakt vagy maszklevétel, vagy mind a kettő egyszerre)
* Az esemény ideje
* Az esemény időtartama
* Az eseményt tároló videó neve
* Az esemény súlyossági szintje
* és 3 funkciógomb az esemény megtekintéséhez, letöltéséhez és törléséhez

Ezeket az adatokat a program a korábban említett MySQL táblában tárolt értékekből kérdezi le (1. és 5. táblázat) SQL lekéréseken keresztül. A táblázat kitöltése egy egyszerű foreach ciklussal történik.

Az SQL lekérések paraméterezhetőek a keresési argumentumoknak megfelelően, tehát a videó készítésének idejének megfelelően. A lekérés GET metóduson keresztül történik, ezáltal akár tárolhatóak, küldhetőek is a különböző keresési beállítások.

    public static function getEvents(Request $request){

        if($request->filled('starttime')){

            $starttime = $request->input('starttime');

        }

        else{

            $starttime =  date("Y-m-d H:i:s", strtotime("-1 hours"));

        }

        if($request->filled('endtime')){

            $endtime = $request->input('endtime');

        }

        else{

            $endtime =  date("Y-m-d H:i:s");

        }

        if($request->filled('level')){

return view('events', ['events' => Events::whereBetween('time', [$starttime, $endtime])->where('level',$request->input('level'))->cursor()]);

        }

        else{

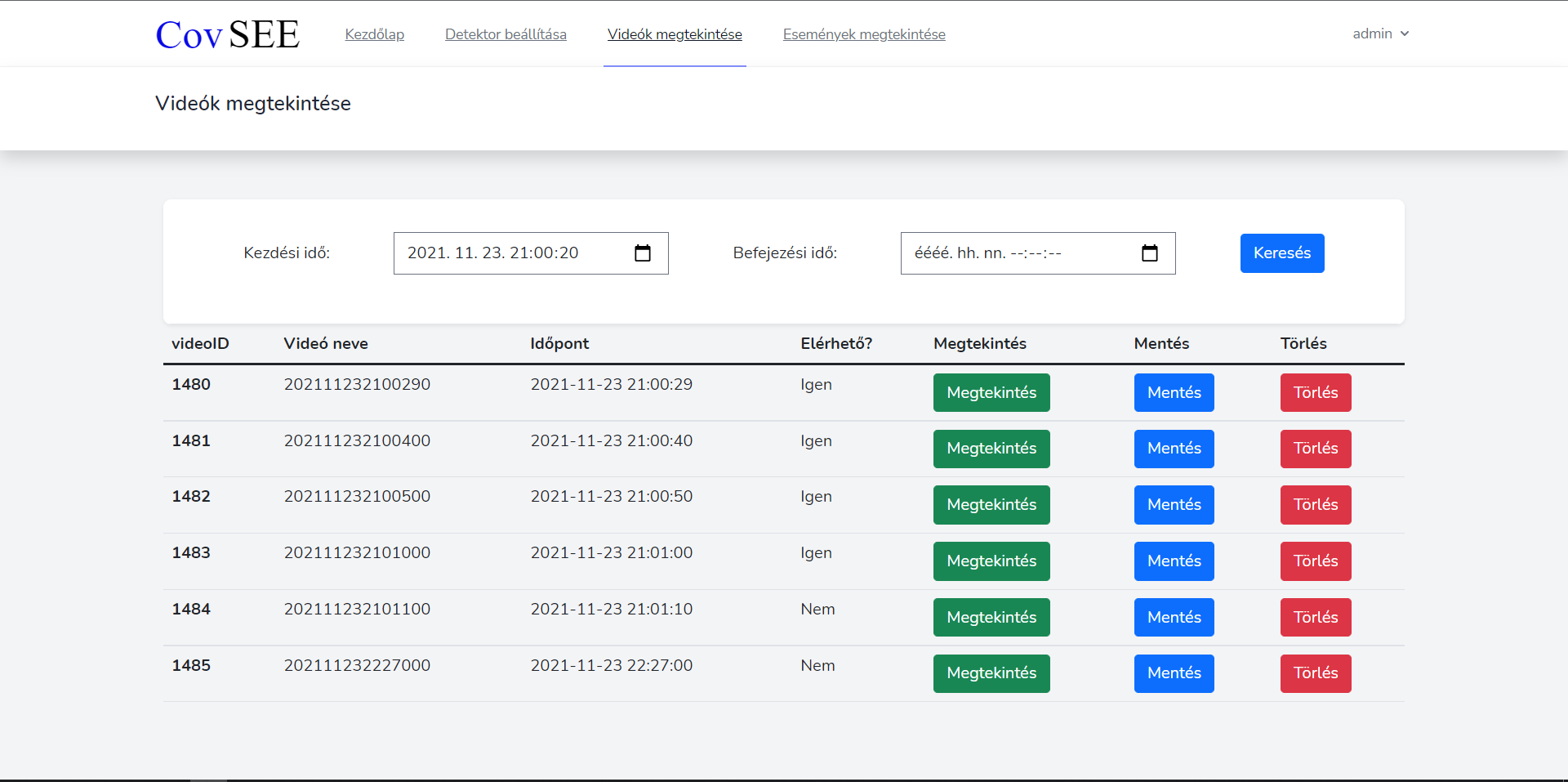
        return view('events', ['events' => Events::whereBetween('time', [$starttime, $endtime])->cursor()]);

        }

    }

1. kódrészlet: Az eseményeket lekérdező Controller függvény

Az oldal tetején még egy keresősáv is szerepel, mivel a sok bejegyzés miatt nagyon körülményes lenne a teljes listát végig nézni (emellett a lista is csak az utolsó 1 óra adatait jeleníti meg mivel nagyon lassú lenne az oldal betöltése, emellett erőforráspazarló is egy teljes SQL lekérdezést végig futtatni), emiatt lehet mind videónál mind eseménynél keresni időintervallumra. Emellett eseményeknél még súlyosság alapján is lehet keresni egy legördülő lista segítségével.



24. ábra: A videók megtekintésére szolgáló felület

Az oldal kódját Laravel blade template segítségével építettem fel. A keretrendszerhez kapcsolódó Laravel Jetstream kiegészítő modul alap mintaoldalát kezdtem el felhasználni és átalakítani a weboldal igényeinek megfelelően, így az oldal egész felépítése és külső kinézete egységes lesz. A felső sávba a keresőmezőhöz egy HTML formot készítettem, amiben oldaltól függően 2, vagy 3 beviteli mező van, a hozzájuk tartozó labelekkel együtt, kettő datetime-local típusú beviteli mező, ami a kezdő és a befejező dátum bevitelére szolgál, és egy legördülő select lista, ami az események visszatekintésénél az esemény súlyosságának a kiválasztására szolgál. A beviteli mezők mellett található egy Keresés elnevezésű submit button is, ami a form elküldésekor visszatér ugyanerre a címre, csak küldi magával a form kitöltésekor bevitt adatokat egy GET kérésben, amely kérést a Laravel a Routing rendszerén és a hozzá elkészített kontrolleren keresztül feldolgozza, a megadott paraméterek alapján végrehajtja az SQL lekérdezést, és visszaadja az oldalt a megfelelő adatokkal egy táblázatban megjelenítve. Ez pontosabban úgy működik, hogy a Routingon keresztül a Laravel elküldi a Controllernek a GET kérést, ami egy SQL lekérdezést végez, ennek a lekérdezésnek az eredményét (ami egy lista lesz) eltárolja egy változóban, amit a weboldal leírófájljának továbbít, ami pedig végigmegy egy ciklussal a lista elemein és minden elemet kiír egy táblázati sorba.

A videók és az események listázása során lehetőség van ezeknek az elemeknek a megtekintésére, lementésére és törlésére. Ezt az oldalon található 3 gomb segítségével lehet megtenni. A megtekintés és a törlés esemény során az oldal egy felugró, Modal ablak segítségével kommunikál a felhasználóval, ami egy modern HTML keretrendszer, a Bootstrap[9] segítségével sikerült megoldani. A gombra kattintáskor az oldal elindít egy JavaScript esemény, ami egy oldalon előre felvitt, azonban rejtett mezőt aktivál, ami szolgál a modal kijelzésére. Ez a JavaScript függvény még arra is képes, hogy meghíváskor módosítsa a modal tartalmát, ezáltal amikor a videókat akarjuk megtekinteni minden kattintáskor újra létrehozza a modalon belül a videó taget és feltölti a kiválasztott videó elérési útvonalával. A törlésnél pedig a törlendő videó nevével, emellett ott a törlés gombra is definiál egy eseményt, ami egy aszinkron JavaScript kérést indít el a webszerver felé az adatbázisból való törlés érdekében. Emellett videó esetén a tárhelyről is törli a videófájlt.

### Élő videó megtekintése és konfigurálása

A felhasználónak lehetősége van arra, hogy megtekintse a program futása közbeni élő videóképet és beállítsa a detektor néhány tulajdonságát. A videófelvételek élő közvetítéséhez a program által éppen feldolgozott képkockákat a program egy RTMP adatfolyamon keresztül elküldi egy feldolgozó és tehermentesítő webszerver részére, ami pedig megosztható formában továbbítja akár több felhasználó számára is.

Az RTMP adatfolyamot a detektor programból egy FFmpeg-nek[15] nevezett, videó feldolgozó és kódoló szoftver segítségével valósítom meg. Ez az ffmpeg nevű program képkockánként kapja meg a feldolgozott videót, amit átalakít a megfelelő kimenetre (esetünkben egy h264 kódolású RTMP streamre), és továbbítja a webszerver felé.

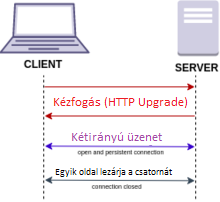
A webszerveren fel van állítva egy listener arra a portra, amin beérkezik a detektor felől az élő videó, és adatfolyam beérkezése esetén továbbítja egy megfelelő, korábban beállított URL-re, mind RTMP közvetítésként, mint pedig m3u8 közvetítésként, hogy webes környezetben is megnyitható legyen.

Az élő videókat feldolgozó oldal linkjének localhostot állítottam be, mivel az egész program helyi futással lett elkészítve, és tesztelve. Természetesen a detektort lehet akár egy külön, főként grafikus gyorsítókat tartalmazó szerverre helyezni, ebben az esetben csak a programban kell átírni egy paramétert, a **detect.py** fájlban az URL-t tartalmazó rtmp\_url változó értékét szükséges módosítani.

## A kapcsolattartó réteg

A kapcsolattartó réteg a program és a weboldal közötti kapcsolattartásra szolgáló réteg. Erre a rétegre azért van szükség, mivel a weboldalról is módosíthatóak a detektor bizonyos paraméterei, értékei, és ezért szükséges ez a réteg, mivel a weboldalról közvetlenül belenyúlni nem lehet a detektor függvénybe (nem is lenne célszerű)

Ezért egy köztes, réteget terveztem a programnak, ami ezt a kapcsolattartást végrehajtja. A weboldallal ez a program WebSocket[36] protokollon végez információcserét. A WebSocket egy kommunikációs protokoll, ami a valós idejű, folyamatos információcserére találtak ki.



25. ábra: A WebSocket protokoll működése[40]

A protokoll TCP kapcsolaton keresztül működik és a folyamatos, teljes duplex információcserére lett kifejlesztve. Ezáltal megfelelő a program számára is, mivel a szerver és a kliens közötti információcserekor folyamatos kapcsolatra van szükség, ameddig a lekérések és a visszaigazolások folynak. Emellett azért is választottam ezt a protokollt, mivel Python és JavaScript programozási nyelven belül ez egy egyszerűen megvalósítható protokoll. A program detektor felőli protokollja a video almappán belül a pysocket.py fájlban található. Ebben a fájlban első lépésben beimportáltam a szükséges Python modulokat, hogy a program működése megfelelő legyen. Az importálás után megtörténik a program által használt alap változók beállítása alapértékekre, hogy a program működőképes legyen. Utána a program elindítja a listenert a websocketre a **8765**-ös porton, és várja a bejövő adatokat. A listener elindításakor egy függvényt hív meg, ami definiálja az async műveleteket. Minden művelet esetén a program egy bejövő üzenetet vár JSON formátumban, amit aztán megpróbál értelmezni és sikeres értelmezés esetén megpróbálja értelmezni az üzenetet. Az üzenet egy „;” -vel elválasztott, 2 elemű json lista, aminek az első fele egy string a második fele viszont vagy egy karakter, vagy egy szintén json lista. A változatok és a műveletek a következőképpen alakulnak:

* **start** üzenet egy json listával, amiben van egy src, conf és min mező
* **change** üzenet egy json listával, amiben van egy src, conf és min mező
* **status** üzenet egy karakterrel (az esetünkben 0 lesz, de igazából lényegtelen mert nem dolgozzuk fel, csak szükséges
* **stop** üzenet egy karakterrel (az esetünkben 0 lesz, de igazából lényegtelen mert nem dolgozzuk fel, csak szükséges

Az üzenetek természetesen a nevüknek megfelelő műveleteket végzik el. A start üzenet elindít egy detektor alfolyamatot, ha még nem a megadott src forrással, conf bizonyossági szinttel (tehát hogy mi legyen a minimum bizonyosság, ami után azt mondja egy elemre, hogy észlelésnek veszi) és min legkisebb embere közötti távolsággal. A change üzenet változtat a detektoron a vele együtt megadott adatokkal, pontosabban leállítja a detektort és a kapott új adatokkal elindít egy új folyamatot vele. A status üzenettel le lehet kérni a detektor jelenlegi állapotát és beállított értékeit. Ezt a weboldalon előtöltés helyett egy adatlekérés gombbal oldottam meg, és így a felhasználóra van bízva az adatok lekérése. A stop üzenet pedig leállítja az éppen futó detektor példányát.

Ezek, a műveletek egyszerűnek tűnnek azonban azért szükség volt több feltételvizsgálatra is ezekkel kapcsolatban, pl., hogy fut-e a detektor, hogy ne indítsunk el 2 példányt belőle, vagy pedig le van állva, vagy még el sem indult egyszer sem (az utolsó 2-t csak programozástechnikai okokból különböztettem meg, nincs különbség a két állapot között lényegesen, csak a változók definiálása így egyszerűbb volt).

# Felhasználói dokumentáció

## A program követelményei

### A program hardveres követelményei

A program hardveres követelményeit már korábban felvázoltam azonban újra felvázolom a következőkben:

* Nvidia GTX 960 videókártya, vagy újabb (GTX 10xx, GTX 16xx, RTX 20xx RTX 30xx), legalább 4GB VRAM-mal, legalább CUDA 11.4 támogatással
* Intel i5 6. generációs CPU, vagy újabb
* 8GB RAM
* legalább 1,5GB tárhely a szoftvernek
* legalább 2TB tárhely a felvételek tárolására
* Ajánlott dedikált szerveren futtatni
* Legalább 1 MP felbontású kamera 25 fps sebességgel

A program külön paraméterezhető akár 2 külön szerveren való futáshoz, a kódba kell egy minimális módosítást végezni, főleg a videók mentése terén és a különböző alrendszerek kommunikációs csatornáit illetőleg (websocket, közvetítés). Ez talán olyan esetben jöhet jól, ha 2 külön szerveren fut a program, amiből az egyik egy nagy skálázhatóságú webszerver, ami főleg az I/O műveletekre van felkészítve (vagy akár 2 szerver, 1 a webnek és 1 az adatbázisnak), a másik pedig egy nagy számítási teljesítménnyel rendelkező, főleg grafikus gyorsítókból álló szerver.

### A program szoftveres követelményei

A program szoftveres követelményeit több csoportra is lehet bontani, annak megfelelően, hogy melyik szoftverösszetevőt nézzük. Külön követelménye van a websocketnek, a detektornak, a weboldalnak és az adatbázisnak is. Ezeket a követelményeket ezért a következőkben csoportosítva vázolom fel az elkülöníthetőség érdekében

A program minden része Ubuntu Linux 20.04 LTS alatt lett fejlesztve és lett kitesztelve.

**A detektor és a WebSocket program szoftveres követelményei:**

* Python 3.8.10 alatt tesztelve
* YOLOv5 detektor # 19c8760 számú commitja (forkot készítettem róla, ami a következő linken érhető el: <https://github.com/b3nc301/yolov5>) A YOLOv5-nek a követelményei a requirements.txt fileban is olvashatóak, de felvázolom őket itt is:
  + matplotlib>=3.2.2
  + numpy>=1.18.5
  + opencv-python>=4.1.2
  + Pillow>=7.1.2
  + PyYAML>=5.3.1
  + requests>=2.23.0
  + scipy>=1.4.1
  + torch>=1.7.0
  + torchvision>=0.8.1
  + tqdm>=4.41.0
  + mysql.connector python plugin
* ffmpeg feldolgozó 4.2.4 alatt tesztelve
* websockets

**Az adatbázis szoftveres követelménye:**

* MySQL adatbázis 8.0.27 alatt tesztelve

**A weboldal szoftveres környezete:**

* PHP 8.0 alatt tesztelve (legalább PHP8.0 kell, felette nincs tesztelve)
  + BCMath
  + Ctype
  + Fileinfo
  + JSON
  + Mbstring
  + OpenSSL
  + PDO
  + Tokenizer
  + XML
  + MySQL
* PHP 8.0 fpm socket
* nginx webszerver

Ezeket a programokat lehet egy szerverre is telepíteni, nem interferálnak egymással, úgy is lett kitesztelve az egész rendszer, azonban lehetséges több szerverre is telepíteni mindegyik modul összetevőit, így minden részelem csak a saját erőforrásait fogyasztja, nem fogyasztja a másik rendszer elől az erőforrásokat.

A szoftver követelményeinek megfelelő programcsomagokat természetesen kézzel is fel lehet telepíteni, azonban a következő parancsokat segítségképpen összeállítottam, hogy használhatóak legyenek. Fontos, hogy a rendszer szintű parancsokat Ubuntu Linux 20.04 alatt készítettem el, tehát más disztribúciókkal nem garantált a működése.

sudo apt install python3

sudo apt install python3-pip

sudo apt install git

git clone <https://github.com/b3nc301/thesis>

cd thesis/video

git clone <https://github.com/b3nc301/yolov5>

cd yolov5

pip3 install -r requirements.txt

sudo apt install ffmpeg

sudo pip3 install wesockets

1. kódrészlet: A detektor és websocket segéprogramjainak letöltése és telepítése

A detektor és az őt kiszolgáló websocket program indításához csak a video mappában található pysocket.py fájlt kell elindítani a **python3 pysotcket.py** paranccsal és innentől kezdve készenáll a detektor indítására websocketen keresztül

sudo apt install software-properties-common

sudo add-apt-repository ppa:ondrej/php

sudo apt update

sudo apt install php8.0

sudo apt install php8.0 php8.0-bcmath php8.0-ctype php8.0-fileinfo php8.0-mbstring php8.0-pdo php8.0-tokenizer php8.0-xml php8.0-mysql php8.0-fpm

sudo apt install nginx-full libnginx-mod-rtmp

1. kódrészlet: A weboldal szoftveres környezetének telepítése

A weboldal szoftveres környezetéhez tartozik egy nginx konfigurációs fájl is, amit szükséges módosítani, hogy a program a megfelelő módon üzemeljen.

A módosításhoz szükséges megnyitni a /etc/nginx/nginx.conf fájlt egy szövegszerkesztővel (pl. nano), és a config fájl végére a következő részletet szükséges beírni (8. kódrészlet):

rtmp{

server{

listen 1935;

chunk\_size 4096

application live{

live on;

hls on;

hls\_path /var/www/html/public/lives;

hls\_fragment 6s;

hls\_keys off

}

}

}

1. kódrészlet: Az nginx.conf fájl bővítése

Ezáltal a webszerverünk újraindítás után az **1935**-ös portra hallgatva fogad RTMP közvetítéseket, amiknek a hls fájljait eltárolja a /var/www/html/public/lives mappában található mappába, ahol létrehoz egy m3u8 lejátszási fájlt, amit most már weben is le lehet játszani, és több szegmenst is még létrehoz, amik szükségesek a közvetítéshez.

Azonban így még csak a közvetítéseket továbbítjuk, a weboldal még nem üzemkész. Úgy tudjuk üzemképessé tenni, az oldalt hogy a kódban található web mappa tartalmát áthelyezzük a /var/www/html/ mappába, a „.env.example” file-t kitöltjük mezőknek megfelelően(főleg az SQL szerverre vonatkozó mezőket) és az nginx mappában található default fájlt áthelyezzük a /etc/nginx/sites-enabled/ mappába kicserélve az eddigi fájlt.(érdemes biztonsági mentést csinálni róla esetleges problémák elkerülése érdekében)

Az adatbázis feltöltéséhez is biztosítottam SQL táblákat, amik a web mappának database mappájában a **tables.sql** fájlban találhatóak. Ebben a fájlban előre van definiálva egy felhasználói fiók is, amivel be lehet lépni a rendszerbe.

### A program használata

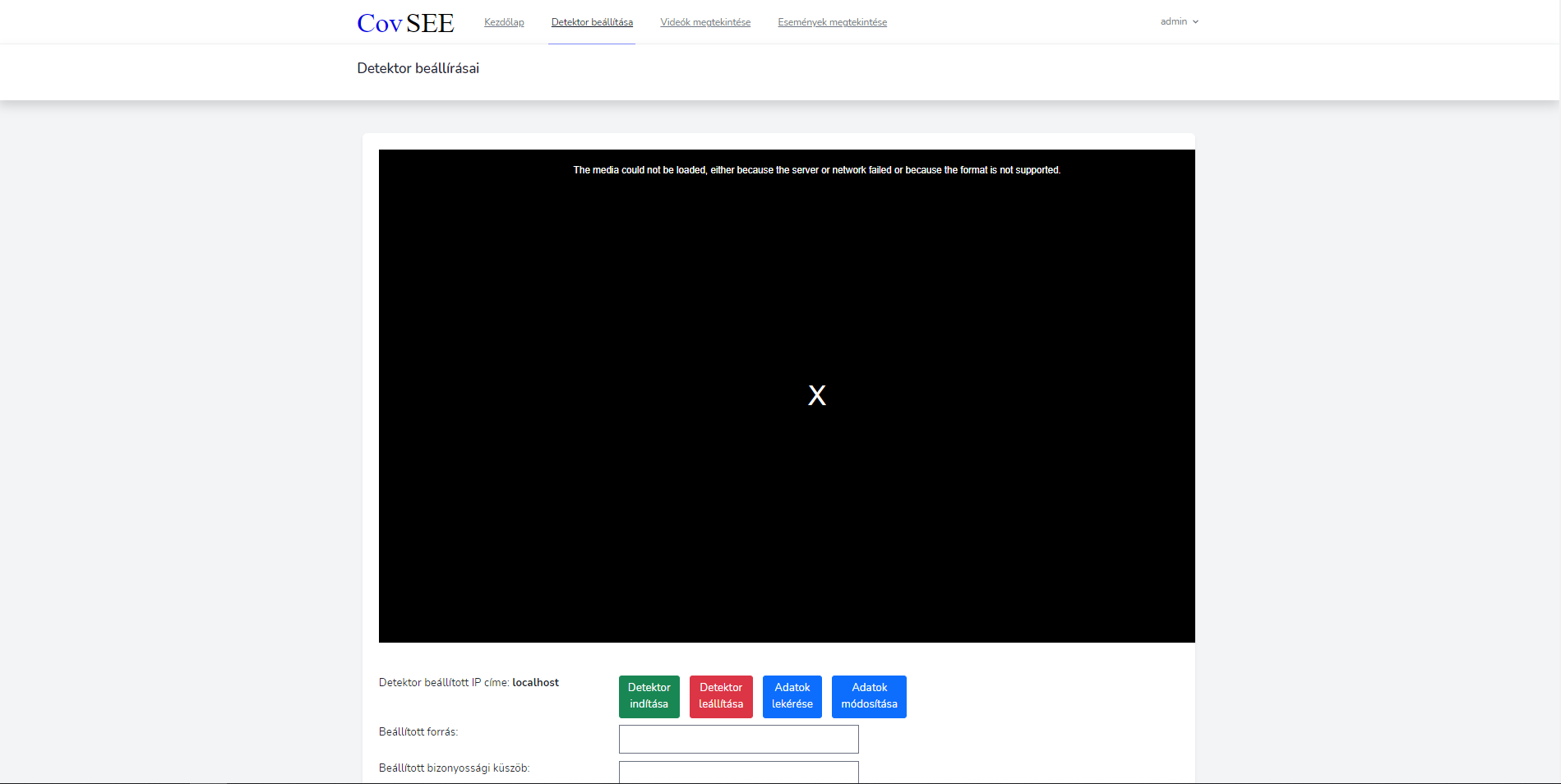
A program használatához első lépésben be kell jelentkezni a feltelepített weboldalra. Ez az alap SQL táblában elkészített admin@admin felhasználó 12345678 jelszó kombinációval lehetséges.

Bejelentkezés után a felhasználó a főoldalon találja magát. A főoldalon semmi információ nincs, csak kiírja a program, hogy bejelentkezett a felhasználó. A funkciókat a felső menüsáv tartalmazza, amin 4 elem található, a kezdőlap, ahol most is tartózkodunk, a Detektor beállító oldal, ahol a detektor élő képét tudjuk nézni, és be tudjuk állítani a különböző beállításait, a videók megtekintésére és kezelésére szolgáló oldal, és az események megtekintésére és kezelésére szolgáló oldal.

A detektor beállításaira vonatkozó oldalon legfelül látunk egy videólejátszót, ami a detektor aktív állapota esetén mutatja az aktuális videót (Igaz sajnos késleltetéssel a sok réteg miatt, amin átmegy a videó). A videólejátszó alatt láthatjuk a detektor websocketjének a beállított címert (ha másik szerveren van, akkor lényeges információ), emellett 4 gombot, ami a következőkre való:

* A detektor indítása gomb elindítja a detektort az itt, oldalon megadott információk alapján. Alapértelmezetten a detektor soha nem indul el, azt kézzel kell, nekünk elindítanunk ezzel a gombbal
* A detektor leállítása gomb leállítja a detektort
* Az adatok lekérése gomb segítségével lekérdezhetjük a beállított adatokat, amik beíródnak a szövegmezőkbe
* Az adatok módosítása gomb pedig a beírt adatokat felviszi a websocketbe és onnan a detektorba

A gombok alatt található még 3 mező, ezek egyértelműen a forrás, bizonyossági küszöb és minimális távolság beállítására vonatkoznak.



26. ábra: A detektor beállítására szolgáló felület

A videók megtekintésére vonatkozó oldalra lépve a rendszer betölti a legutóbbi 1 óra videófelvételeit. Ezt a felső sávban elhelyezett szűrő segítségével tudjuk módosítani, hogy más időpontra is tudjunk szűrni. A videókat emellett meg tudjuk tekinteni egy előugró ablakban, le tudjuk tölteni mp4 formátumban és tudjuk törölni is őket (ez mind az adatbázisból mind a fájlrendszerből törli őket).

Az események megtekintésére szolgáló fülön ugyan azok a lehetőségek és opciók vannak, mint a videók megtekintésére szolgáló oldalon, csak itt nem videókat, hanem eseményeket lehet kezelni.

# Tesztelés és eredmények

## Tesztelési esetek

A program teszteléséhez először azt kell tudni, hogy mire is akarunk tesztelni, milyen teszteseteket és tesztelési szempontokat szeretnénk felállítani. A teszteléseknek a jó eredmények elérése érdekében változatosnak és sokrétűnek kell lennie, hogy megfelelő visszajelzést adjon a program állapotáról. Ezért döntöttem úgy, hogy a tesztelés esetében mind saját, mint publikus adatkészleteket is felhasználok.

A saját adatkészletek készítése viszonylag egyszerű feladat. Kiteszek valahová egy kamerát, és megpróbálok a tesztelési esetnek megfelelő műveleteket elvégezni én magam, ezáltal biztosan tudom kontrollálni a tesztnek a lefutását, hogy minden általam kívánt eshetőségi eset le legyen tesztelve. Azonban egyedül nem feltétlenül vagyok képes olyan szituációkat előállítani, amik sokszor egy publikus teszt adatkészleten szerepelnek. Ezért a dolgozatomban fogok alkalmazni publikus adatkészleteket is. Azonban a publikus adatkészletekkel van egy olyan probléma, hogy mostanában már egyre kevesebb található belőlük, főként a GDPR-nak és a személyiségi jogoknak köszönhetően. Az emberek nem szeretnek publikus videófelvételeken szerepelni, még úgy sem sokszor, hogy ezek esetleges tudományos célból készülnek. Ezért sok publikus adatkészlet már nem érhető el, mivel a képeken szereplő személyek kérték a videó eltávolítását. Emellett a másik említett probléma, a GDPR is megnehezíti a kutatók érdekét. A GDPR szabályozás miatt nem lehet publikus felvételeket közzétenni, amin felismerhető emberek szerepelnek az emberek személyes adatainak védelme érdekében, ezért Európai felvételeket szinte nem is lehet találni az interneten.

A fentebb megemlített indokok miatt internetes forrásból kevés felvételt tudtam csak felhasználni, az egyik ilyen felvétel a VIRAT[1] videó adatkészlet. Ez az adatkészlet különböző biztonsági kamerák képeit tartalmazza, amelyek segítségével a programot lehet tesztelni valós körülmények között. Ez az adatkészlet főként a távolság és az emberek felismeréshez használható, mivel az adatkészlet még a Covid előtti időkből (2011) származik.

A maszkok felismerésére főként saját videókat készítettem, amelyben saját magamon teszteltem különböző körülmények között a programomat. Próbáltam különböző megvilágítások mellett, különböző szögekből, irányokból, különböző távolságokból megközelíteni a kamerát a tesztelés érdekében.

A picture containing text

Description automatically generated

27. ábra: Saját készítésű tesztvideókból mintaképek

Ezen felül kipróbáltam a videóközvetítéseket, hogy azok hogyan működnek. Sajnos egyedül nem voltam képes lemodellezni azokat az eseteket, amikor több ember, távolságon belül áll egymástól maszkokkal vagy azok nélkül, ezért erre az esetre rögtönzött megoldásként Joe Biden elnöki beiktatásából származó videófelvétel egy részletét használtam fel több-kevesebb sikerrel. A tesztvideókat és a teszteseteket a program könyvtárának demo mappájában helyeztem el.

## Tesztelési eredmények(RÉGI SÚLYOKKAL, ÚJ TANÍTÁS ALATT)

* **IMG\_2562.MP4**

Az algoritmus a maszkos és a maszk nélküli eseteket is remekül felismeri. A videó közepén egy áthaladásnál állon lévő maszk esetén összezavarodott, hogy nem jól viselem a maszkot, vagy nem viselem a maszkot. Engem, mint ember objektumot remekül felismert. Végig szemben álltam a kamerával ezen a felvételen. A videó végén a fekete pólómra egy 2-3 másodpercig azt hitte, hogy az is maszkot visel, amikor közelítettem a kamera felé.

* **IMG\_2563.MP4**

A maszkos eseteket remekül felismeri szembefordulás esetén. Oldalra forduláskor elveszíti a maszkot, de utána felismeri újból. A maszklevételkor felismeri, hogy levettem a maszkot. Az államon lévő maszkot először felismerte, hogy levettem, aztán pedig ahogy közelítettem a kamerához látszott, hogy összezavarodik és folyamatosan nem visel-rosszul visel állapot között ugrált. A végén az orrom alá húztam fel a maszkot, amikor közel voltam akkor azt hitte, hogy rajtam van a maszk, viszont ahogy távolodtam érdekes módon összezavarodott, hogy most viselem-e vagy pedig nem jól viselem-e.

* **IMG\_2564.MP4**

A videóban a bevilágító zavaró fényforrásra azt hiszi, hogy maszkot hord, kicsit érdekes jelenség. Hibás maszkviselés esetén megint összezavarodik, de most már a 3 maszkállapot között ugrál folyamatosan.

* **IMG\_2565.MP4**

Itt már csökkentettem a megvilágításon, így felfedeztem egy érdekes jelenséget, hogy az árnyékokat felismeri néha embernek. A maszkviselést felismerte, az állra húzott maszkot megint próbálta kitalálni, hogy rosszul viselt-e vagy nem viselt, de többségben volt a nem viselt eredmény.

* **IMG\_2566.MP4**

Ezen a videón még az eddigieknél is jobban csökkentettem a megvilágításon, ami a fekete maszkomnak hála néha el is veszíti a maszkviselést, de összességében jól szerepelt a maszkviselésen. A többi esetet itt nem próbáltam.

* **IMG\_2567.MP4**

Ezen a felvételen a 2566-os felvételhez hasonló fényviszonyokkal dolgozok, csak elfordított kamerával, így más nézetben is kipróbáltam a kamerát. Sötétben, ha távol állok a kamerától akkor semmit nem ismer fel, csak ha szemben és közel állok hozzá akkor ismer fel bármit is, még ha viselem is a maszkot. Itt is előjön a probléma, hogy az árnyékot személynek detektálja.

A person taking a selfie

Description automatically generated with low confidence

28. ábra: Saját készítésű videók teszteléséből néhány eset

5. táblázat: A program tesztelésének eredményei saját videókkal

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tesztvideó | Viselt maszkot | Nem viselt maszkot | Nem jól viselt maszkot | Személy |
| **IMG\_2562** | 95% | 70% | 70% | 100% |
| **IMG\_2563** | 90% | 60% | 40% | 100% |
| **IMG\_2564** | 65% | 40% | 25% | 95% |
| **IMG\_2565** | 90% | 80% | 50% | 70% |
| **IMG\_2566** | 70% | - | - | 90% |
| **IMG\_2567** | 30% | 0% | 0% | 70% |
| Átlag | **73,3%** | **50%** | **37%** | **87,5%** |

## Fejlesztési javaslatok

A tesztelések alatt a program vegyes eredményekkel szerepelt. A saját videófelvételeken viszonylag pontos volt, nem veszítette el a magabiztosságát csak néhány esetben. Sajnos a „nem jól viseli a maszkot” kategória nagyon rosszul szerepelt, szinte soha nem ismerte fel jól ezt az állapotot, általánosságban vagy azt ismerte fel, hogy nincs rajta maszk, vagy ha az orr közelébe volt helyezve, akkor már azt, hogy fent van a maszk. Ez a probléma következtethető abból, hogy betanítás során az adathalmazban ebből a kategóriából volt a legkevesebb kép, 500 darabnál kevesebb és ezért teljesít ebben a kategóriában gyengén.

Chart, bar chart

Description automatically generated

29. ábra: A felismerési kategóriák megoszlása

A maszkok helytelen hordásának problémáját lehetne orvosolni azzal a megoldással, hogy sokkal több (nagyságrendileg több ezer) új képet adunk hozzá a betanító adathalmazhoz, amivel ez a kategória sokkal nagyobb részarányban vesz majd részt a betanítási folyamat alatt és ezáltal sokkal pontosabb is lesz a felismerés.

# Összefoglalás

A dolgozatomban egy koronavírus kontaktkutatásra alkalmas videó elemző szoftver tervezésnek és elkészítésének a folyamatát vezettem le. A dolgozat készítése során számos forrást felhasználtam, amelyekből ötleteket merítettem a szoftver felépítéséhez. Az algoritmus felépítéséhez a neurális hálózatokat vettem alapul, erre az alaptémakörre épül a dolgozat egésze. Ezek közül a hálózatok közül is a mai világban modern és népszerű, konvolúciós neurális hálózatokat választottam. Ezen hálózatok működését a dolgozatomban is levezettem, mind a működésüket, mind a matematikai hátterüket. A konvolúciós neurális hálózatokon belül is a YOLOv5 algoritmust választottam a detektálási feladatok elvégzéséhez. Ez az algoritmus egy mai, modern, széles körben használt és kedvelt algoritmus, emellett egyszerűen használható, ezért a munkám megvalósításához is tökéletesen alkalmas volt, a dolgozatomban ennek a hálózatnak a kifejtésével is lehet találkozni. A munkám készítése során is tapasztaltam a hálózat ezen jó tulajdonságait, gyorsan és jól használható programokat lehetett vele készíteni. A dolgozat készítése során volt lehetőségem neurális hálózatot is betanítani adatkészletek segítségével, ami elsőre kicsit nehézkes volt, azonban néhány próbálkozás után már tudtam használni az algoritmus ezen funkcióját is. A betanítás során nem feltétlenül értem el a hálózat maximum potenciálját, mivel a rengeteg beállítási lehetőségnek hála, a tökéletes hálózat kialakítása rengeteg időt vett volna igénybe, emellett a betanítási idő is heteket vett volna el, ha az algoritmus készítői által adott utasításokat követtem volna. Ezért a programom nem a legtökéletesebb hálózatot használja, ennek köszönhetően lényegesen többet hibázik, mint amennyit a legjobb beállítások mellett hibázna, viszont ezzel a lépéssel energiát és időt takarítottam meg a dolgozat készítése során. Ha szeretnénk javítani a neurális hálózatnak a hatékonyságát, akkor azt nagyobb adatkészlettel (több akár tízezres nagyságrendű adatról beszélünk), hosszabb betanítással, avagy precízebb finomhangolással lehetne megtenni. Ez a dolgozat egyik jövőbeni fejlődési irányvonalának is tekinthető lenne, hogy hogyan lehetne javítani a dolgozatban felvázolt neurális hálózat felismerési képességeit. Emellett a munkámat még több szempontból is lehetne javítani, fejleszteni.

A fejlesztések közé tartozik például a több kamerának a használata. Jelenleg a program egy kamerát képes kezelni, ami egy valós környezetben nem feltétlenül életszerű helyzet. A biztonsági kamerával felszerelt épületek, területek általában több kamerával rendelkeznek, ami a programom jelenlegi állapota szerint csak úgy lehetnek használni, hogy minden kamerára építünk egy saját detektorszervert, ami lássuk be idő és erőforrás pazarlás. Ezért a programom egy fejlesztési iránya lehet a több kamerának az együttes használata és kezelése. Ezáltal akár egy komplett biztonsági csomaggá is lehetne alakítani a programomat.

Fejlesztési iránynak tekinthető még az interakciós webfelületnek a fejlesztése is. A dolgozatomban elég kevés funkciót szerveztem bele a felhasználóknak szánt felületbe, ez mellé még rengeteg különböző funkciót lehetne beleépíteni, mint pl. jogosultsági szintek, sokkal speciálisabb esemény és videókeresés, és akár egy mobilra letölthető, webes alapú applikáció fejlesztése.

# Irodalomjegyzék

1. "A Large-scale Benchmark Dataset for Event Recognition in Surveillance Video" by Sangmin Oh, Anthony Hoogs, Amitha Perera, Naresh Cuntoor, Chia-Chih Chen, Jong Taek Lee, Saurajit Mukherjee, J.K. Aggarwal, Hyungtae Lee, Larry Davis, Eran Swears, Xiaoyang Wang, Qiang Ji, Kishore Reddy, Mubarak Shah, Carl Vondrick, Hamed Pirsiavash, Deva Ramanan, Jenny Yuen, Antonio Torralba, Bi Song, Anesco Fong, Amit Roy-Chowdhury, and Mita Desai, in Proceedings of IEEE Comptuer Vision and Pattern Recognition (CVPR), 2011.
2. Adrian Rosebrock: Intersection over Union (IoU) for object detection - <https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/> letöltés ideje: 2021 ápr. 24.
3. Alex Rodriguez: Introduction to RESTful Web services - <https://developer.ibm.com/articles/ws-restful> letöltés ideje: 2021 ápr. 24.
4. AMD ROCm - <https://rocmdocs.amd.com/en/latest> letöltés ideje: 2021. ápr. 24.
5. Andrewmvd: Face Mask Detection - <https://www.kaggle.com/andrewmvd/face-mask-detection> letöltés ideje: 2021.11.15
6. Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M. W., Pfau, D., Schaul, T., ... & De Freitas, N.: Learning to learn by gradient descent by gradient descent. arXiv preprint arXiv:1606.04474. 2016.
7. Basile Roth: A social distancing detector using a Tensorflow object detection model, Python and OpenCV - <https://deepnote.com/@deepnote/A-social-distancing-detector-using-a-Tensorflow-object-detection-model-Python-and-OpenCV-KBcEvWejRjGyjy2YnxiP5Q> letöltés ideje: 2021.11.27.
8. Bhatnagar, S., Gill, L., & Ghosh, B.: Drone Image Segmentation Using Machine and Deep Learning for Mapping Raised Bog Vegetation Communities. Remote Sensing, 12(16), 2602., 2020
9. Bootstrap - <https://getbootstrap.com/> letöltés ideje: 2021.11.19
10. COCO Dataset - <https://cocodataset.org/#home> letöltés ideje: 2021. 11. 27.
11. CUDA Zone - <https://developer.nvidia.com/cuda-zone> letöltés ideje: 2021 ápr. 24.
12. D. A. Forsyth and J. Ponce, Computer vision: a modern approach. Prentice Hall Professional Technical Reference, 2002.
13. De Boer, P. T., Kroese, D. P., Mannor, S., & Rubinstein, R. Y..: A tutorial on the cross-entropy method. Annals of operations research, 134(1), 19-67., 2005.
14. Elfwing, S., Uchibe, E., & Doya, K. (2018). Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, *107*, 3-11.
15. FFMPEG <https://ffmpeg.org/> letöltés ideje: 2021.11.10
16. Girshick, R., Donahue, J., Darrell, T., & Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 580-587). 2014.
17. Jason Brownlee: Loss and Loss Functions for Training Deep Learning Neural Networks - <https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/> letöltés ideje: 2021.ápr.29.
18. Jetstream - <https://jetstream.laravel.com/2.x/introduction.html> letöltés ideje: 2021. 11. 20
19. Joseph Redmon: Darknet: Open Source Neural Networks in C - <https://pjreddie.com/darknet/> letöltés ideje: 2021 ápr. 24.
20. Kaggle - <https://www.kaggle.com/> letöltés ideje: 2021.11.15
21. Kingma, D. P., & Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980., 2014.
22. Laravel - <https://laravel.com/docs/8.x#why-laravel> letöltés ideje: 2021.11.10
23. Laravel Blade - <https://laravel.com/docs/8.x/blade> letöltés ideje: 2021.11.12
24. Lin, T. Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., ... & Zitnick, C. L.: Microsoft coco: Common objects in context. In European conference on computer vision (pp. 740-755). Springer, Cham., 2014
25. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. C.: “Ssd: Single shot multibox detector,” in European conference on computer vision. Springer, pp. 21–37., 2016
26. Maali Alharbi: Face Mask Dataset - https://www.kaggle.com/datasets/maalialharbi/face-mask-dataset letöltés ideje: 2022.03.15
27. Matthew Stewart: Simple Introduction to Convolutional Neural Networks - <https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac> letöltés ideje: 2021.ápr.29.
28. O'SHEA, Keiron; NASH, Ryan.: An introduction to convolutional neural networks. arXiv preprint arXiv:1511.08458, 2015.
29. Overview of model structure about YOLOv5 #280 - <https://github.com/ultralytics/yolov5/issues/280> letöltés ideje: 2021.11.25.
30. R. Girshick, “Fast r-cnn,” in Proceedings of the IEEE international conference on computer vision, pp. 1440– 1448., 2015
31. Redmon, J., Divvala, S., Girshick, R., & Farhadi, A.: You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 779-788)., 2016
32. RegisFrey: The model, view, and controller (MVC) pattern relative to the user. - <https://commons.wikimedia.org/wiki/File:MVC-Process.svg> letöltés ideje: 2021.11.19.
33. Roboflow - <https://roboflow.com/> letöltés ideje: 2021.11.27.
34. Sagar Shamra : Activation Functions in Neural Networks - <https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6> letöltés ideje: 2021 ápr. 24.
35. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, 15(1), 1929-1958., 2014
36. The WebSocket API (WebSockets) - <https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API> letöltés ideje: 2021.11.20.
37. Ultralytics : YOLOv5 - <https://github.com/ultralytics/yolov5> letöltés ideje: 2021.10.20
38. Wang, C. Y., Bochkovskiy, A., & Liao, H. Y. M.: Scaled-YOLOv4: Scaling Cross Stage Partial Network. arXiv preprint arXiv:2011.08036., 2020
39. Wang, C. Y., Liao, H. Y. M., Wu, Y. H., Chen, P. Y., Hsieh, J. W., & Yeh, I. H. (2020). CSPNet: A new backbone that can enhance learning capability of CNN. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops* (pp. 390-391).
40. Websocket - <https://upload.wikimedia.org/wikipedia/commons/1/10/Websocket_connection.png> letöltés ideje: 2021. 11. 23
41. Yang, D., Yurtsever, E., Renganathan, V., Redmill, K. A., & Özgüner, Ü.: A vision-based social distancing and critical density detection system for covid-19. arXiv preprint arXiv:2007.03578, 24-25., 2020
42. Yu, D., Wang, H., Chen, P., & Wei, Z.: Mixed pooling for convolutional neural networks. In International conference on rough sets and knowledge technology (pp. 364-375). Springer, Cham., 2014
43. Zou, Z., Shi, Z., Guo, Y., & Ye, J.: Object detection in 20 years: A survey. arXiv preprint arXiv:1905.05055. 2019.

# Mellékletek

1. [A dolgozat mellékletei, ha vannak]