[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

[1 oldalas, magyar nyelvű tartalmi kivonat]

# Abstract

[Title in English]

[1 oldalas, angol nyelvű kivonat]

# Tartalomjegyzék

TARTALOM

[Bevezetés 1](#_Toc70632424)

[1. Elméleti háttér 2](#_Toc70632425)

[1.1. Konvolúciós neurális hálózatok bemutatása és működése 2](#_Toc70632426)

[1.1.1. Konvolúció 2](#_Toc70632427)

[1.1.2. Pooling rétegek 3](#_Toc70632428)

[1.1.3. Teljesen kapcsolt rétegek 4](#_Toc70632429)

[1.1.4. Aktivációs rétegek 4](#_Toc70632430)

[1.1.5. Tanulás 5](#_Toc70632431)

[1.1.6. Algoritmusok típusai 7](#_Toc70632432)

[1.2. YOLO algoritmus 7](#_Toc70632433)

[1.3. Emberek közötti távolság meghatározása 9](#_Toc70632434)

[2. Tervezés 11](#_Toc70632435)

[2.1. Fejlesztői hardver, hardverkövetelmények 11](#_Toc70632436)

[2.2. Igényspecifikáció 11](#_Toc70632437)

[2.3. Maszkok felismerése 13](#_Toc70632438)

[2.4. YOLO algoritmusok megvalósításai 13](#_Toc70632439)

[2.5. Maszkok és emberek detektálása YOLO algoritmussal 14](#_Toc70632440)

[2.6. Rendszer felépítése 14](#_Toc70632441)

[2.7. Kimeneti adatok tárolásának módja 17](#_Toc70632442)

[2.8. Webes felület 20](#_Toc70632443)

[Irodalomjegyzék 22](#_Toc70632444)

[Mellékletek 24](#_Toc70632445)

# 

# Bevezetés

Az egész világot megrázta egy 2020-ban 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 a szabályszegéseket. 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, amit egy webszerver fog futni Python alapokon.

# 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.[24]

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

### Konvolúció

A konvolúció egy speciális művelet, ami képfeldolgozás esetén egy előre megadott konvolúciós kernelt felhasználva, az adott képpont körüli koordinátákat összeszorozva a kernellel (tulajdonképpen súlyozva), 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)

És ezt a 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[17]A tanulmány szerint ezekben az algoritmusokban a konvolúciós kernelek taníthatóak, ezáltal képesek különböző objektumok jellemzőpontjainak meghatározása. Ezek a jellemzők egy ún. 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 konvolúciót.

### Pooling rétegek

A 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 a pooling 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 után kapott kimeneti képet adjuk tovább a következő művelet számára.

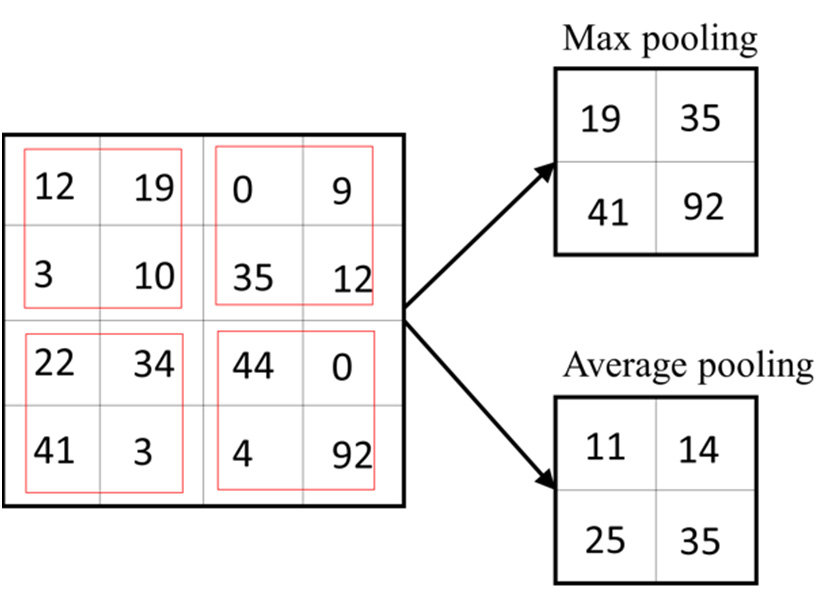
Az algoritmusok erre a műveletre egy méretű kernelt használnak. Én most egy méretű kernel segítségével fogom bemutatni ez a műveletet. Ez a kernel jellegéből adódóan 4 képkockából fog készíteni 1-et, ezzel a bemeneti kép eredeti méretét 25 százalékkal tudjuk csökkenteni. Erre a kernelre azonban többféle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

* **maxpooling**

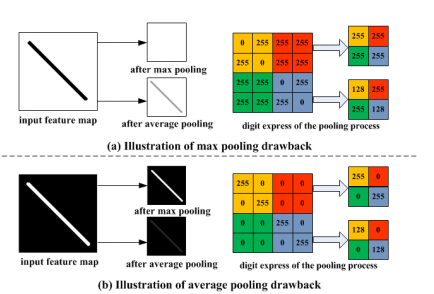
A maxpooling műveletnél a nevéből is adódóan az algoritmus a kernellel lefedett elemek közül megkeresi a legnagyobbat és ez a legnagyobb elem lesz az új képen egy képkocka. Ennek a műveletnek a megfordítása a **minpooling**, ami ugyanezen az elven működik, csak nem a legnagyobb hanem a legkisebb element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

* **avgpooling**

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



1. ábra: maxpooling és avgpooling[2]



2. ábra: Az avgpooling és a maxpooling közötti különbség [23]

### Teljesen kapcsolt rétegek

A detektálás során a konvolúció és a pooling rétegek többször végrehajtódnak egymás után megadott sorrendben. Azonban a futás legvégén nem ezek, 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 ún. 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[11].

* **ReLU**

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

Az eredmény a tartományba esik.

(3)

* **Szivárgó(leaky) ReLU**

A szivárgó ReLU az előbb 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)

### Tanulás

A neurális hálózatoknak egy fő folyamata a tanulás.[12] 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 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 valamilyen módszerrel ki kell számolni, hogy a hálózat mekkora hibaaránnyal működik (pl. veszteséget számolunk), és ez alapján a hibaarányok alapján valamilyen módszerrel módosítanunk kell a hálózatunkon. Erre az optimalizációra többféle megoldás létezik, a népszerűbb megoldások közé tartozik pl. a gradiensereszkedés módszere[1] és az Adam optimalizáció[14].

* **Veszteségfüggvények**

Az optimalizáció során használt veszteségfüggvények[8] 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)[4]

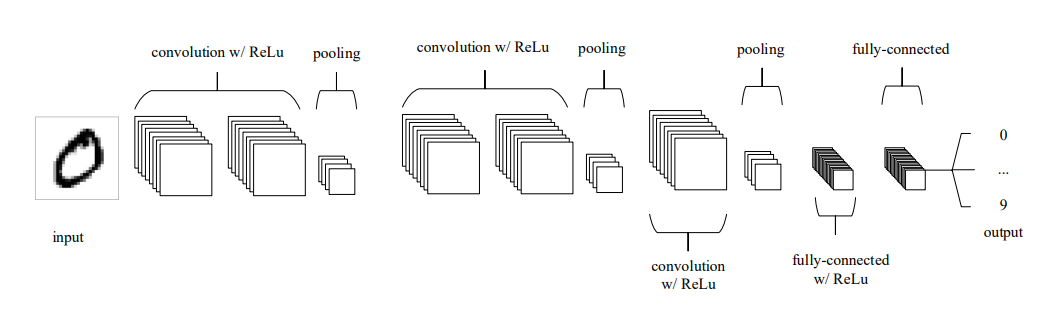
* **Kiejtés**

A kiejtés (dropout[20]) 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 valamilyen matematikai függvény vagy logika mentén végezni.

### Algoritmusok típusai

Ezek közül a CNN algoritmusok közül elsők között csak olyan típusúak léteztek, amik csak az egész képet tudták osztályozni. Azonban idővel megjelentek azok is, amelyik egy kép régiójáról tudták megmondani az osztályt. Ezek közül is az első híresebb algoritmus az R-CNN[5] 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[24] 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.[18] 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)[16] és a YOLO (You Only Look Once).[19]



3. ábra: Példa egy Konvolúciós Neurális Hálózat felépítésére [17]

## YOLO algoritmus

A YOLO a You Only Look Once nevű algoritmusnak a rövidítése. A következőkben ezt az algoritmust fogom bemutatni.[19] 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 a 4. ábra magyarázza, aminél az Area of Overlap a két geometria metszetének a területét adja meg, az Area of Union pedig a két geometria uniójának a területét.

(6)

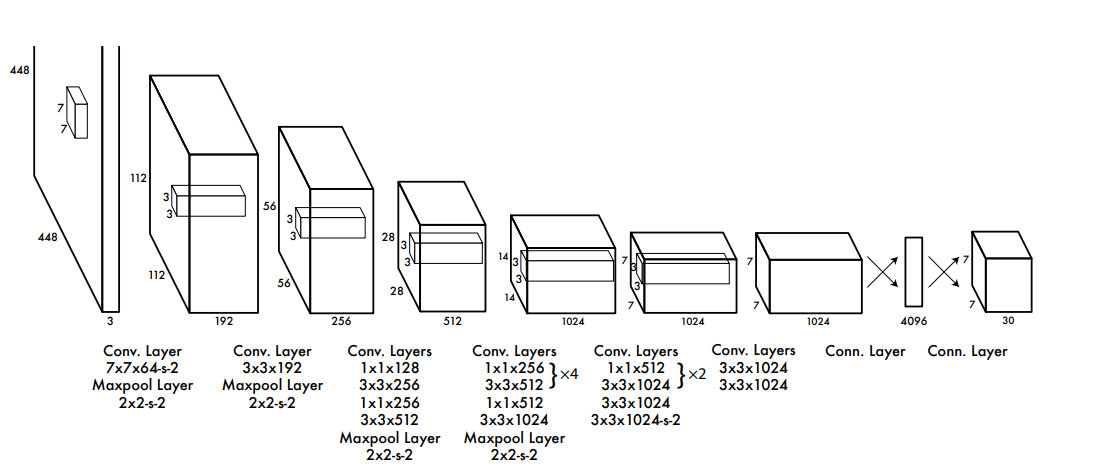


4. ábra: IoU magyarázata[13]

Ezek után azt is meghatározza az algoritmus, hogy az adott négyzet, amit vizsgálunk, az milyen osztályba sorolható elemet tartalmazhat. Ezt a következő, 7. képlettel lehet meghatározni, ahol a Pr a valószínűségeket jelenti az Oszt az adott osztály valószínűségét, az Obj pedig az adott geometriában található objektum valószínűségét.

(7)

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



5. ábra: YOLO hálózat felépítése[19]

## Emberek közötti távolság meghatározása

Az emberek közötti távolság meghatározásához szükséges algoritmust már többen levezették korábban.[22] A forrás alapján első lépésben szükséges felismerni az embereket a képen, amit a korábban felvázolt YOLO algoritmus fog elvégezni.[19] Ezek után következik az emberek pozíciójának a megtalálása 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 8. képlet szerint lehet kiszámolni a koordinátákat

(8)

Azonban, ha a kép bal felső (), és jobb alsó () koordinátáit kapjuk meg, akkor pedig a 9. képlet szerint lehet kiszámolni a koordinátákat.

(9)

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[3] alkalmaz:

(10)

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 11. képletben található, és ha ez kisebb, mint a beállított távolság, akkor riasztás keletkezik a rendszerben.

(11)

# Tervezés

## Fejlesztői hardver, hardverkövetelmények

A szoftver jellegéből adódóan nem képes minden hardvertípuson futni. Mivel neurális hálót használ, ezért egy dedikált videókártya szükséges a futásához. Igaz, a neurális háló képes grafikus kártya nélkül is, a számítógép processzorán futni, azonban a sebességében hatalmas visszaesés lesz tapasztalható ezesetben. A videókártyák a többmagos, párhuzamos feldolgozásukkal sokkal gyorsabban tudnak végezni ezekkel a specifikus feladatokkal. Azonban a videókártyák 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[10] nevű eszközt szolgáltatja a gyártó, Nvidia részéről pedig a népszerűbb, CUDA[7] 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.

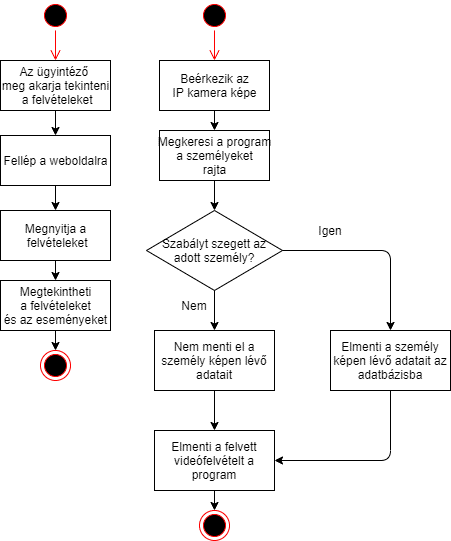
* Nvidia GTX 9xx videókártya, vagy nagyobb, legalább 4GB VRAM-mal, CUDA támogatással
* Intel i5 6. generációs CPU, vagy újabb
* 8GB RAM
* legalább 1,5GB tárhely a szoftvernek
* legalább 2TB tárhely a felvételek tárolására
* Ajánlott dedikált szerveren futtatni
* Legalább 1 MP felbontású kamera 25 fps 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.



6. 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 szögig. Azt is szükséges megvalósítani ezen felül, hogy a hibásan hordott, például orrot nem takaró maszkot felismerje. 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. 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 bemutatott YOLO algoritmusnak különböző megvalósított verziói léteznek. Ezek a verziók általában azok alapján különböznek, hogy milyen optimalizáción, javításon estek át, esetleg milyen hálózatot használnak backbone hálózatnak a program működése során, milyen keretrendszeren keresztül használják ezeket a hálózatokat.

A YOLO algoritmusoknak különböző verziói is léteztek. Mindegyik verzió valamilyen újítást hozott az előzővel kapcsolatban.

Az első és legismertebb YOLO megvalósítás a Darknet. A Darknet egy C alapon írt CUDA-t használó neurális hálózati keretrendszer.[9] 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 YOLOv4 kiadása[21]

## 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.

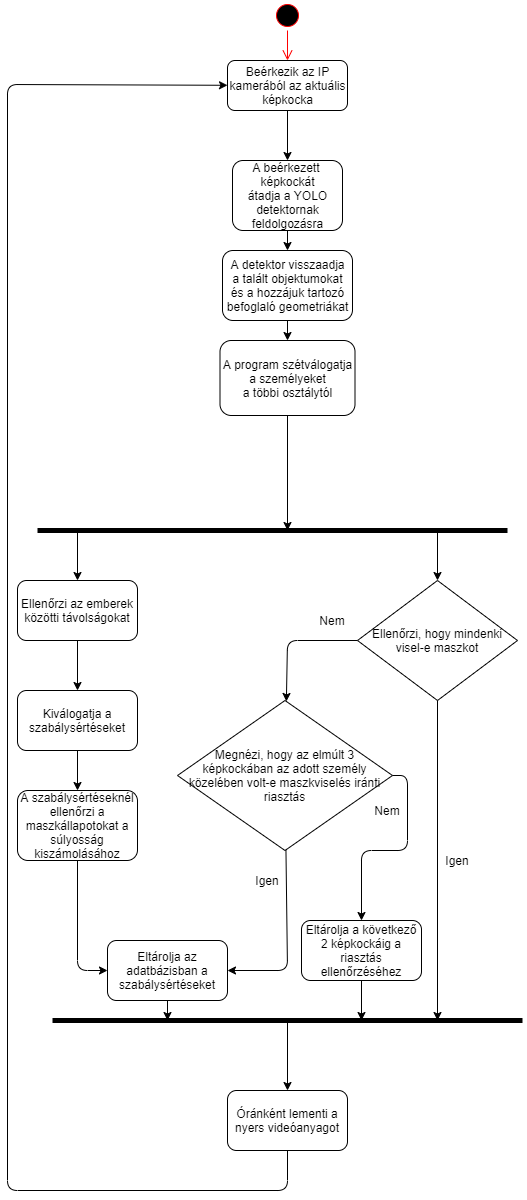
Ezeket az eseteket, hogy fel tudja ismerni az algoritmus első lépésnek meg kell tanítani vele. A betanítás során a hálóba képeket táplálunk, megadva a képrészleteket, amiket vizsgáljon, és azok kategóriáit. Ezek után a neurális háló elvégzi a képek jellemzőinek az összegyűjtését, és eltárolja a különböző 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.[15]

## 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 szétválogatja a személyeket, hogy a következő lépések gyorsabban végre tudjanak hajtódni. Első lépésnek 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 bontódik. 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.



7. ábra: A rendszer felépítésének vázlata

## Kimeneti adatok tárolásának módja

Graphical user interface, application

Description automatically generated

8. 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 | int | leíró |
| videoDate | A videó készültének ideje | datetime | leíró |
| videoURL | A videó elérési útvonala | varchar(45) | leíró |
| videoAvailable | A videó elérhető-e | nvarchar(20) | 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 |
| token | Bejelentkezési token | varchar(45) | leíró |

3. 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 |

4. táblázat: A loginhistory tábla terve

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Egyedhalmaz** | **Attribútum** | | **Adattípus** | **Tulajdonság típusa** |
| **Név** | **Szöveges értelmezés** |
| loginhistory | loginID | Azonosító kód | int | azonosító |
| userID | Azonosító kód | int | idegen azonosító |
| ip | Bejelentkezési IPv4 cím | char(15) | leíró |
| time | időpont | datetime | leíró |

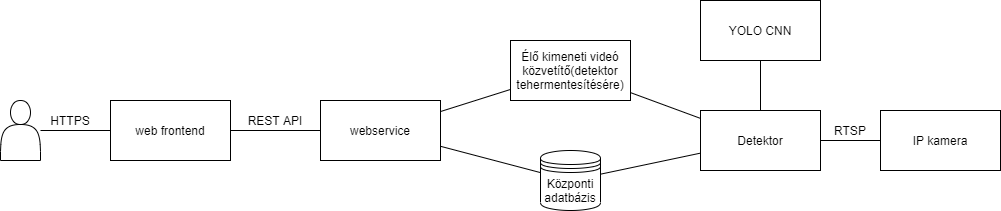
5. táblázat: Az events tábla terve

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Egyedhalmaz** | **Attribútum** | | **Adattípus** | **Tulajdonság típusa** |
| **Név** | **Szöveges értelmezés** |
| events | eventID | Azonosító kód | int | azonosító |
| xBox | A bounding box x koordinátája | float | leíró |
| yBox | A bounding box y koordinátája | float | leíró |
| wBox | A bounding box szélessége | float | leíró |
| hBox | A bounding box magassága | float | leíró |
| classID | Az esemény típusa | int | idegen kulcs |
| time | Az esemény ideje | datetime | leíró |
| videotime | Az esemény ideje a videóban másodpercben kifejezve | int | leíró |
| videoID | A videó azonosítója | int | leíró |
| level | A riasztás szintje | int | leíró |

A kimeneti adatok tárolásának módjára egy adatbázisrendszer használata szükséges. Ez az adatbázis rendszer a felső, 8. ábra és a fenti táblázatok (1-5.) 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. [6] A frontend felől az API-n keresztül történnek meg a kérések, amit a backend lekezel, és megfelelő választ ad vissza rá. A rendszer Django framework alapokon fog működni, ő szolgálja ki az API-t, válaszol a kérésekre, állítja össze a válaszüzeneteket.



9. ábra: A rendszer felépítése webes kiszolgálóval

A rendszer felépítése szempontjából a korábban felvázolt detektor megkapja az IP kamerától a képkockákat egy RTSP videostreamen 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. threshold, 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.

Diagram

Description automatically generated

10. ábra: Felhasználó use-case diagramja

Diagram

Description automatically generated

11. ábra: Adminisztrátor use-case diagramja

# Irodalomjegyzék

1. 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.
2. 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
3. D. A. Forsyth and J. Ponce, Computer vision: a modern approach. Prentice Hall Professional Technical Reference, 2002.
4. 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.
5. 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.

1. [https://developer.ibm.com/articles/ws-restful](https://developer.ibm.com/articles/ws-restful/) letöltés ideje: 2021 ápr. 24.

1. <https://developer.nvidia.com/cuda-zone> letöltés ideje: 2021 ápr. 24.

1. <https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/> letöltés ideje: 2021.ápr.29.

1. <https://pjreddie.com/darknet/> letöltés ideje: 2021 ápr. 24.

1. <https://rocmdocs.amd.com/en/latest> letöltés ideje: 2021. ápr. 24.

1. <https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6> letöltés ideje: 2021 ápr. 24.

1. <https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac> letöltés ideje: 2021.ápr.29.

1. <https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/> letöltés ideje: 2021 ápr. 24.
2. Kingma, D. P., & Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980., 2014.
3. 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
4. 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
5. O'SHEA, Keiron; NASH, Ryan.: An introduction to convolutional neural networks. arXiv preprint arXiv:1511.08458, 2015.
6. R. Girshick, “Fast r-cnn,” in Proceedings of the IEEE international conference on computer vision, pp. 1440– 1448., 2015
7. 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
8. 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
9. Wang, C. Y., Bochkovskiy, A., & Liao, H. Y. M.: Scaled-YOLOv4: Scaling Cross Stage Partial Network. arXiv preprint arXiv:2011.08036., 2020
10. 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
11. 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
12. 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]