# ETDK-dolgozat

Faragó Richard

Fekete Árpád-Bence

Ferencz Dániel

**XXV. reál- és humántudományi Erdélyi Tudományos Diákköri Konferencia (ETDK)**

**Kolozsvár, 2022. május 19–22.**

# Adrenaline Eye

## Automatikus fényképkészítés kalandpark látogatók számára gépi látás segítségével

**Szerzők:**

**Faragó Richard**

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar, Informatika szak, III. év

**Fekete Árpád-Bence**

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar, Adatelemzés és modellezés szak, II. év

**Ferencz Dániel**

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar, Informatika szak, III. év

**Témavezetők:**

**dr. Sulyok Csaba,** egyetemi adjunktus

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar

**Patakfalvi Örs-Krisztián,** szoftverfejlesztő,

Codespring

**Ráduly Zalán,** szoftverfejlesztő,

Codespring

**Tunyogi Rudolf-Bálint,** szoftverfejlesztő,

Codespring

CODESPRING

OUR CODE YOUR SUCCESS

## Kivonat

Kalandparkokban a pályák kialakítása miatt gyakran körülményes a látogatókról fényképeket készíteni, miközben átkelnek egy-egy akadályon. Megfelelő számú fényképész alkalmazása és a számukra szükséges infrastruktúra biztosítása jelentős többletköltségekkel járhat.

Az Adrenaline Eye projekt célja egy automatikus fényképezőrendszer megvalósítása a kalandparkokban felszerelt kamerák segítségével. Első lépésként a kamerák képeket készítenek, amelyeken egy neurális háló sisakfelismerést végez. A detektált sisakon a rendszer felismeri a számot, társítva a képeket a megfelelő felhasználóhoz sisakszám és látogatási idő alapján. Az elkészült album megtekinthető egy webes felületen keresztül.

A rendszer több modulból áll, melyek üzenetsorok segítségével kommunikálnak. Két komponens a kamerák kezelésével és a fotók készítésével foglalkozik. További kettő a képeket dolgozza fel gépi látást alkalmazva, valamint egy szolgáltatás gondoskodik a csoportosításról és tárolásról.

# Tartalomjegyzék

**Bevezető** **1**

**1. Az Adrenaline Eye projekt** **3**

1.1. Funkcionalitások . . . . . 3

1.2. A rendszer felépítése . . . . . 4

**2. Mikroszolgáltatások** **6**

2.1. Mikroszolgáltatás orientált architektúra . . . . . 6

2.2. Kameravezérlő . . . . . 7

2.3. Fényképkészítő . . . . . 10

2.4. Sisakdetektáló . . . . . 11

2.5. Számfelismerő . . . . . 16

2.6. Központi szerver . . . . . 18

2.7. Webalkalmazás . . . . . 20

**3. Számítógépes látás** **23**

3.1. Elméleti áttekintés . . . . . 23

3.2. Probléma feltárása . . . . . 33

3.3. Sisakdetektálás . . . . . 33

3.4. Számfelismerés . . . . . 43

**4. Az alkalmazás működése** **47**

**5. Eszközök és munkamódszerek** **50**

**Következtetések és továbbfejlesztési lehetőségek** **56**

# Bevezető

A kalandparkok pályái általában erdős területeken, fák között vannak kialakítva, amelyeken sorra egy-egy személy kelhet át. Ezért nehéz feladat jó fényképeket készíteni a résztvevőkről miközben átkelnek az akadályokon, főleg úgy, hogy egyszerre többen is lehetnek a parkokban. Több fényképész alkalmazása, és a nekik a biztonságos környezet kialakítása sok munkával és többletköltségekkel járhat.

Az Adrenaline Eye projekt célja egy olyan rendszer fejlesztése, ami automatizálja a kalandpark résztvevőinek a fényképezését, illetve a képek eljuttatását a megfelelő személyekhez. A rendszer segítségével a kalandpark alkalmazottai bevezethetnek különböző kamerákat, amelyek meghatározott időintervallumonként képeket készítenek egy pályáról. Ezek a képek feldolgozásra kerülnek, a képek készítési időpontja, illetve a rajtuk levő sisakon felismert számjegy alapján a rendszer társítja a pályán résztvevő személyekhez. A kalandpark pályáinak kipróbálása után a résztvevők megtekinthetik a róluk készült képeket egy webalkalmazáson keresztül.

Léteznek különböző megvalósítások személyek beazonosítására vidámparkokban, ilyen például az AIPIX [15], ami arcfelismerés által azonosítja be a látogatókat. Másik megvalósítás pedig RFID[1] bevezetésével történő azonosító rendszer: Sniper Action Photo [53] vagy az Argus[61]. Az RFID címkék és az azt olvasó antennák használata, amely esetében az azonosítás könnyebb, mint gépi látást alkalmazó módszerekkel, viszont az antennák és a címkék karbantartása időigényes, körülményes és nem teljesen automatikus, hiszen a látogatók az RFID címkéket hozzá kell érintsék az olvasókhoz az akadály pályákon.

Az Adrenaline Eye rendszer egy kalandparkban használt sisakok számjegye és a látogatási időpont alapján társítja a megfelelő személyekhez az elkészült képeket. A rendszer mikroszolgáltatás orientált architektúrára épül, amely lehetővé teszi, hogy a szolgáltatások különböző jól meghatározott feladatokat hajtsanak végre, illetve újratelepíthető legyen anélkül, hogy veszélyeztetné az alkalmazás egységét. A projekt elsősorban a Kolozsvár melletti Adrenalin Park[2] számára készül, alapötlete Szélyes Leventétől származik, illetve a követelmény-lista együttműködésével alakul ki.

A dolgozat első fejezetében bemutatásra kerülnek a rendszer fő funkcionalitásai és architektúrája. A második fejezetben részletezve van az alkalmazás felépítése, az azt felépítő

[1] Radio-frequency identification
[2] https://adrenalinpark.ro/hu/

mikroszolgáltatások, ezeknek a működése és a köztük lévő kommunikáció. A harmadik fejezet bemutatja a gépi látáshoz kapcsolódó mikroszolgáltatások elméleti hátterét, elért eredményeket és a felhasznált módszereket. A negyedik fejezet az alkalmazás működését mutatja be, míg az ötödik fejezet érinti a projekt fejlesztése során alkalmazott munkamódszereket és a fejlesztést segítő eszközöket. Az utolsó fejezetben a következtetésekről és továbbfejlesztési lehetőségekről van szó.

A projekt fejlesztése 2021 nyarán kezdődött a Codespring nyári gyakorlat[3] keretén belül. Köszönet illeti Dr. Sulyok Csabát a koordinálásért, Patakfalvi Örs-Krisztiánt, Ráduly Zalánt és Tunyogi Rudolfot a mentorálásért és szakmai segítségért, a kapcsolattartó személyt, Szélyes Leventét és az Adrenalin Parkot a szükséges infrastruktúra biztosításáért, valamint a csoportos projekt tantárgy kereteiben csatlakozott diáktársakat, Garfield Adrienne-t, Katona Hunort és Péntek Alízt a hozzájárulásért.

[3] https://edu.codespring.ro/

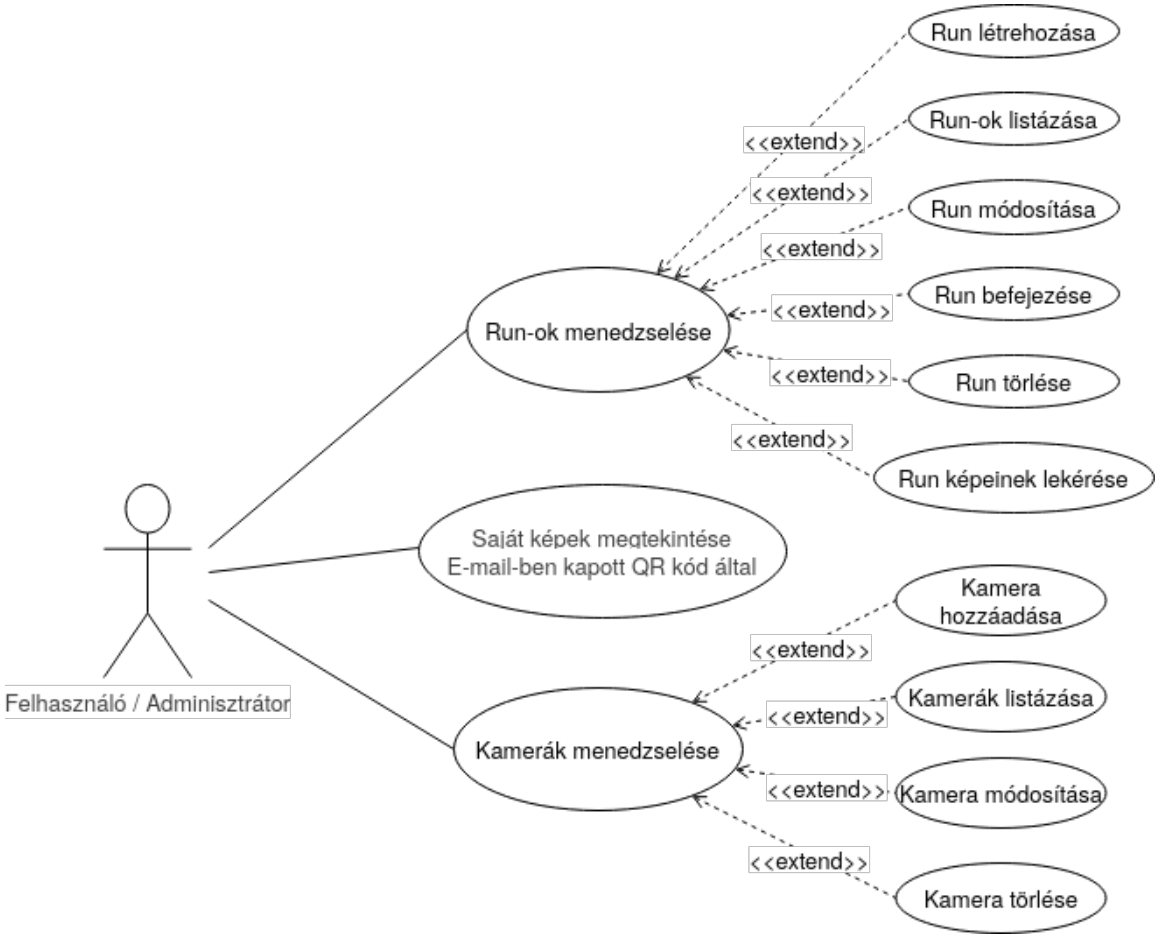
# 1. Az Adrenaline Eye projekt

Az Adrenaline Eye egy kalandparkok számára létrehozott szoftverrendszer, amely automatikusan képeket készít a látogatókról miközben azok átkelnek egy-egy akadályon. Öt különálló mikroszolgáltatásból és egy webes felületből épül fel. Jelen fejezet bemutatja a rendszer főbb funkcionalitásait és felépítését.

## 1.1. Funkcionalitások

A rendszer jelenlegi állapotában egyetlen típusú felhasználó létezik, egyelőre a funkcionalitásokhoz való szerep alapú hozzáférés nem lett megvalósítva, de várhatóan a funkcionalitások fel lesznek bontva adminisztrátorra és felhasználóra. A felhasználó számára elérhető az összes funkcionalitás az 1. ábrán látható.

A run entitás egy személy látogatását jelenti a parkban. Egy runt egy időintervallum



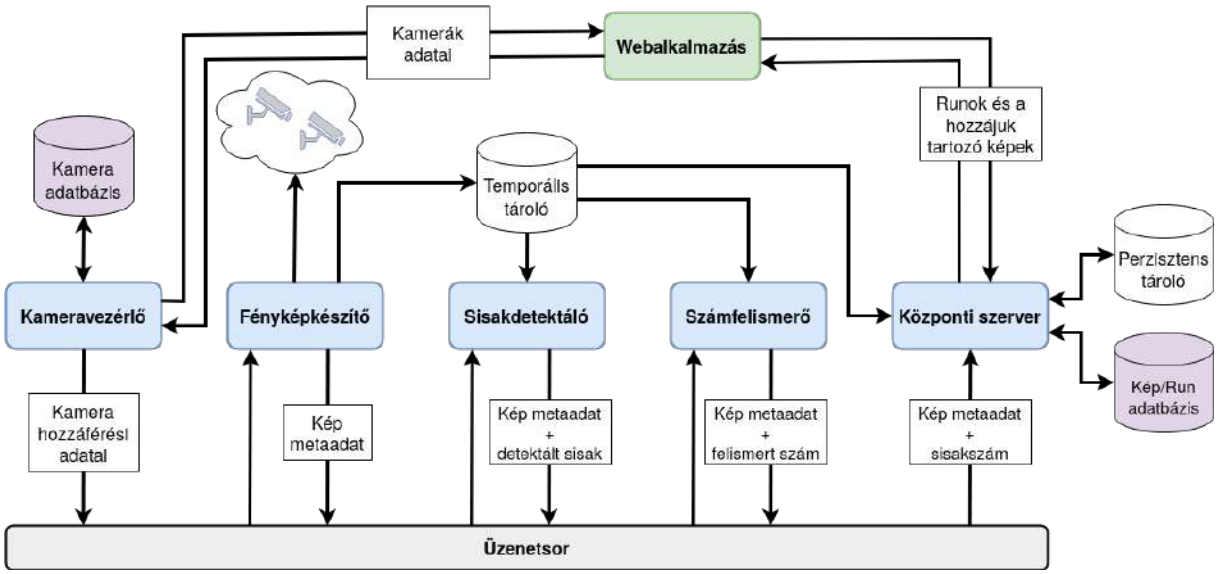

1. ábra. A rendszerben elérhető funkcionalitások listája. A funkcionalitások a runok és kamerák menedzselése, illetve a saját képek megtekintése E-mail-ben kapott QR kód által.

és egy sisakszám határoz meg, és hozzá tartozik a látogató e-mail címe, aki ebben az időintervalummban az adott sisakot viselte. Egy kamera, mint entitás, pedig tartalmazza a rendszerben felvitt, kalandparkokban felszerelt kamerák elérhetőségeit és beállításait.

A kamerák és runok menedzselése adminisztrátor szerepkörhöz tartozó funkcionalitások. Kamerákat lehet létrehozni, kilistázni, egyenként módosítani és törölni a rendszer adatbázisából. Runokat úgyszintén lehet létrehozni, kilistázni, külön-külön módosítani, lezárni, törölni és a hozzátartozó képeket megtekinteni. A kalandpark látogatók megtekinthetik a róluk készített képeket egy weboldalon keresztül, mely elérhető egy e-mailben kiküldött QR kód beszkennelésével, amely egy elérési útvonalat tartalmaz a webes felületen a felhasználó képeit tartalmazó galériához. A galéria QR kód nélkül is elérhető, közvetlenül az elérési útvonalon keresztül.

## 1.2. A rendszer felépítése

Az Adrenaline Eye alkalmazás architektúrája mikroszolgáltatás orientált, ezáltal külön modul felelős minden önálló újrahasznosítható feladat elvégzéséért. A komponensek a 2. ábrán láthatóak és a köztük lévő kapcsolat üzenetsorokon és API[4] kéréseken keresztül valósul




2. ábra. Az Adrenaline Eye mikroszolgáltatásai, balról jobbra indulva ahogyan az üzenetek áramlása is történik. Az áramlás a fényképkészítőtől indul. Miután a kameravezérlő inicializált egy kamerát, elkezdődik az üzenetek időközönkénti létrehozása a fényképkészítő által. A létrehozott üzenet sorra eljut minden mikroszolgáltatáshoz, amelyek a kimenetük alapján módosítják azt. A központi szerver elmenti a képet a háttértárolóra és az adatbázisba, nem továbbít üzenetet. A képeket a webalkalmazáson keresztül tekintheti meg a felhasználó.

[4] Application Programming Interface

meg. A kamerák menedzselésére létrehozott komponens a kameravezérlő, mely egy REST[5] API és a kamera adatait továbbítja a fényképkészítőnek. A fényképkészítő modul a fizikai kamerával létesít kapcsolatot és készít képet. A sisakdetektáló a képeken sisakokat lokalizál. A számfelismerő a sisakokon lévő számot azonosítja be. A központi szerver elmenti az azonosított képeket, valamint a webalkalmazás megjeleníti őket.

A szolgáltatások két tárhelyet használnak a képek elmentésére és olvasására. Egy temporális tárolót, ahova a képek addig tárolódnak, ameddig nem lesz beazonosítva rajta a sisak és a rajta lévő szám. A beazonosítás folyamata után a kép átkerül a perzisztens tárolóra, ahonnan elérhető a központi szerver számára.

Az alkalmazás ezen kívül két külön adatbázist használ: a kameravezérlő és központi szerver számára, amelyek a modellekből származó entitások elmentéséért felelősek.

A szolgáltatások részletes leírására, a köztük lévő kommunikációra és a felhasznált adatbázisok bemutatására a 2. fejezetben kerül sor.

[5]Representational State Transfer

# 2. Mikroszolgáltatások

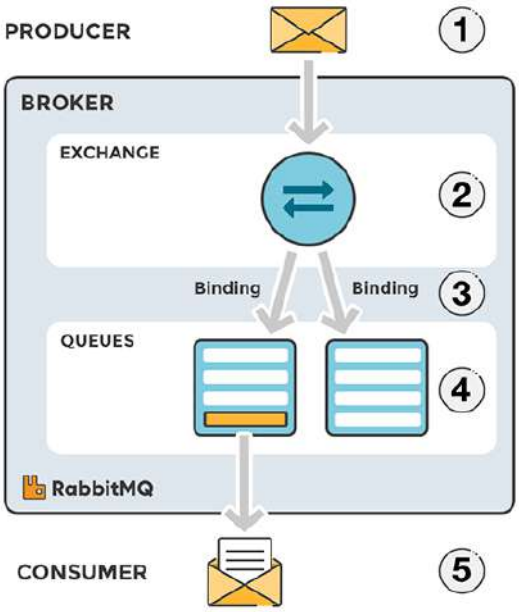
Jelen fejezetben bemutatásra kerül az alkalmazás részletesebb architektúrája, a különböző mikroszolgáltatások és a köztük lévő kommunikációért felelős üzenetsor.

## 2.1. Mikroszolgáltatás orientált architektúra

Az Adrenaline Eye alkalmazás mikroszolgáltatás orientált architektúrára [40] épül, amely nagy rendszerek építésére elterjedten használt. A komponensei egy jól meghatározott szerepet töltenek be, egymástól jól elhatárolhatóak és a köztük lévő függőségek minimálisak. A mikroszervizek ezenfelül függetlenül fejleszthetőek, skálázhatóak és különböző technológiák felhasználásával megvalósíthatóak. A köztük lévő kommunikáció jelentős üzenetalapú, mely üzenetbrókerrel van megvalósítva, amely biztosítja az üzenetek továbbítását, a konkurens feldolgozást, irányítja az üzenetek folyását és igazgatja azok állapotát.

### RabbitMQ üzenetbróker

A projekten belül a *RabbitMQ* [54] bróker biztosítja az üzenet alapú kommunikációt a szolgáltatások között . *AMQP*[6] előíráson alapszik, mely meghatározza az üzenetek orientációját,




3. ábra. Az ábra illusztrálja egy producer és egy consumer közötti üzenetcserét és az AMQP protokoll 3 absztrakt komponensét, az exchanget, bindingot és queuet.

[6] Advanced Message Queuing Protocol

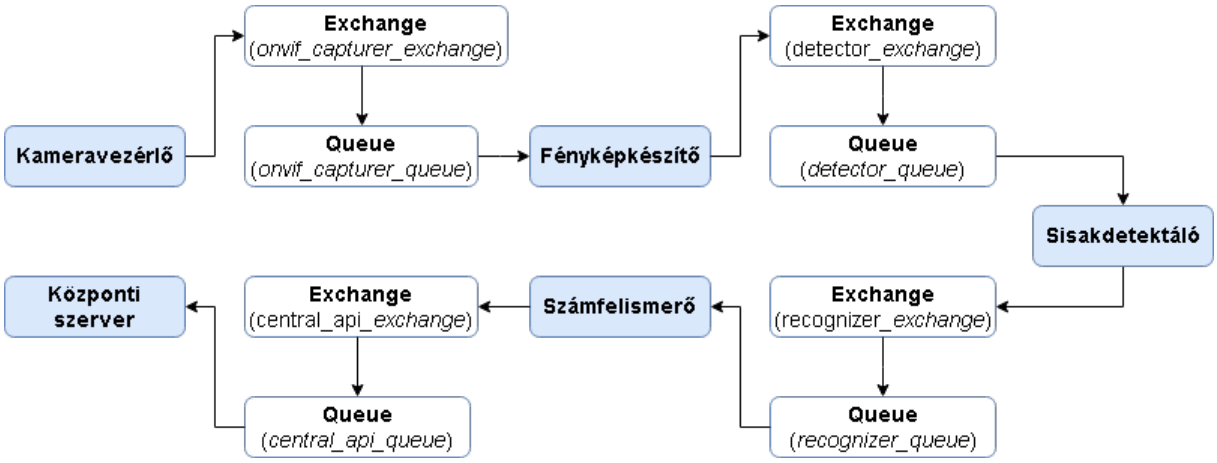
sorba helyezési módját és irányítását. A protokoll 3 absztrakt komponenst foglal magába (3. ábra[7]). Az *exchange* az üzenetek irányításával foglalkozik, a *queue* az üzenetek tárolásához használt FIFO[8] típusú adatstruktúra, valamint a *binding* leírja az *exchange*-k és *queue*-k közötti szabályokat, azaz, hogy az üzenetek melyik üzenetsorba legyenek irányítva.

#### Architektúra és üzenetbróker alkalmazása

A komponensek között egyirányú kommunikáció a *RabbitMQ* által biztosított irányítók, üzenetsorok és szabályok segítségével történik, mindez megfigyelhető A 4. ábrán. A folyamatábra bal oldalán lévő kameravezérlő szolgáltatás *producer*-ként működik, azaz üzeneteket állít elő. Az alatta elhelyezkedő komponens, a központi szerver, előtte lévő komponens üzeneteit dolgozza fel, tehát *consumer*-ként funkcionál. A fényképkészítő, sisakdetektáló és számfelismerő mikroszolgáltatások ugyanakkor *prosumer*-ek, vagyis egyaránt gyártanak és fogyasztanak is.

## 2.2. Kameravezérlő

A kameravezérlő szerver a kamerák menedzseléséért felelős REST API, valamint ütemező feladatkörrel is rendelkezik. Funkcionalitásai közé sorolhatók új kamerák bevezetése, meglévők módosítása vagy törlése. A felsorolt műveletek a `/api/cameras` végpontra küldhető HTTP[9]




4. ábra. A diagramon az Adrenalin Eye komponensei közötti egyirányú kommunikáció, illetve az ehhez felhasznált exchangek, queuek és bindingok láthatóak.

[7]Forrás: https://www.cloudamqp.com/img/blog/exchanges-bidings-routing-keys.png, utolsó megtekintés dátuma: 2022-04-26

[8]First In, First Out

[9]HyperText Transfer Protocol

kéréseken keresztül érhetőek el. Egy kamera létrehozásával együtt létrejön egy időzített taszk is, mely megadott időintervallum alapján küld üzenetet a fényképkészítő szolgáltatásnak, hogy az rögzítsen egy képet. Egy előzőleg bevezetett kamera entitás változtatásakor az ütemező módosítja, illetve törlés során megszünteti a kamerához rendelt feladatot.

### Ütemező és kimenő üzenetek

Az ütemező által indított taszkok megadott időközönként az üzenetbrókeren keresztül kéréseket továbbítanak a fényképkészítő modulnak. Az üzenetek tartalmazzák a kamerák adatait, melyek szervizhez tartozó az adatbázisban vannak lementve. Az adatbázisban tároljuk a kamera azonosítóját, a kép rögzítéséhez szükséges interfészt biztosító protokollt, az elérési címet és hitelesítő adatokat, például felhasználónév és jelszó, melyek előzőleg említett protokolltól függenek. Az üzenetek felépítése látható az 5. ábrán.

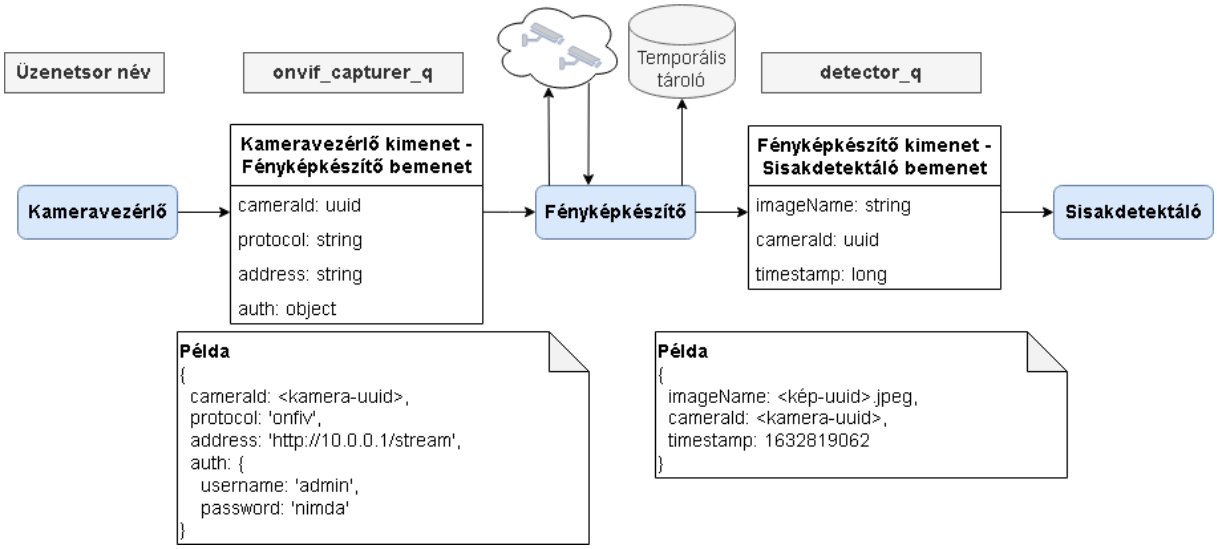
### A szolgáltatás müködése

A kameravezérlő reprezentálja az alkalmazás folyamatának kiindulási pontját az ütemezőnek köszönhetően. A modul elindításakor működésbe lép az ütemező, mely inicializálja az adatbázisban eddig perzisztens kamerákhoz tartozó taszkokat. A *Task Scheduler* a taszkok memóriába való tárolásához a Java által biztosított szinkronizált adatszerkezetet használja, a *ConcurrentHashMap*-t, mely garantálja a szálbiztos működést, azaz zárolást és egyidejűséget enged az atomi metódusainak köszönhetően. Az üzenetbrókerhez csatlakozva, az aktív taszkok folyamatosan üzeneteket küldenek a fényképkészítő komponensnek, ameddig a kamera entitás nem törlődik vagy a szolgáltatás nem áll le. Törlés vagy leállás során az ütemező automatikusan felszabadítja a függőben lévő erőforrásokat.

### Felhasznált technológiák

A webszerver *Java* programozási nyelvben lett implementálva a *Spring keretrendszer* [21] felhasználásával, mely lehetőséget ad az adatelérési réteg, üzleti logika és megjelenítési réteg megfelelő szétválasztására, ezáltal támogatva a háromrétegű architektúrát. A *Spring Data JPA*[10] segítségével lett az adatelérési réteg megvalósítva. Az üzleti logikához tartozó service osztály megteremti a kapcsolatot a perzisztencia és megjelenítési réteg között. Az utóbbi rétegben a *Spring Boot Starter Web* modul használata *RESTful API* segítségével van megvalósítva.

[10] Java Persistence API




5. ábra. Az ábrán a fényképkészítő komponens kommunikációs diagramja látható feltüntetve a fizikai kamerákkal való kommunikáció is. Részletezve vannak példákkal a kameravezérlőtől érkező és sisakdetektálónak továbbított üzenetek.

A *Mapstruct* [3] csomag segítségével létrehozott mapper, a modell osztályt különböző reprezentációkba alakítja át egy entitás létrehozásakor, frissítésekor, vagy lekérdezésekor, a DTO[11] mintát követve. A *Spring Boot Starter Validation* könyvtár az objektumok validálásáért felelős. A *Lombok* pedig megkönnyíti az ismétlődő kódrészletek elkerülését. Az ütemezés megvalósítását a *Spring Task Scheduler* modul biztosítja. A *Spring Cloud Stream* modul interfészeket szolgáltat az üzenet alapú kommunikáció megkönnyítésére, illetve módot ad RabbitMQ üzenetbróker integrálásához, melynek beállításai testreszabhatók a Spring globális konfigurációs állományán vagy környezeti változókon keresztül, ezzel ragaszkodva az *External Configuration Store* mintához is. A *Spring keretrendszer* a *DI*[12] minta segítségével valósítja meg az *IoC*[13] elvet és további programtervezési mintákat használ, melyek megfigyelhetők a szolgáltatás implementációjában, például *Singleton* és *Factory* minta.

A szolgáltatás által menedzselt kamera entitások egy *PostgreSQL* [16] adatbázisban vannak eltárolva, mely rendelkezik *UUID*[14] beépített típussal és az újonnan létrehozott entitásokat ellátja egy automatikusan generált azonosítóval.

[11] Data Transfer Object
[12] Dependency Injection
[13] Inversion of Control
[14] Universally Unique Identifier

## 2.3. Fényképkészítő

A rendszer hardver közeli része a fényképkészítő mikroszolgáltatás. Szerepe, amint a neve is elárulja, képek készítése megadott kamerákkal, illetve a rögzített képek lementése egy temporális tárolóra. A feladat elvégzéséhez szükséges adatokat üzenetsoron át kapja meg a kameravezérlő egységtől, majd API kérést küld a kamerának, hogy az képet készítsen és továbbítja válaszként. Minden sikeresen elkészített fotó adatait továbbítja a sisakdetektáló komponensnek.

### A szolgáltatás müködése, bemenet és kimenet

A komponens mozzanatai és üzenetek felépítései láthatóak az 5. ábrán. A modul induláskor kapcsolódik az üzenetbrókerhez, illetve létrehozza a szükséges bemeneti és kimeneti üzenetsorokat, ha azok az indulás pillanatában nem léteznek. Miután létrejött a bemeneti üzenetsor, figyeli és fogyasztja az arra beérkező üzeneteket. Egy üzenet tartalmazza a kamera azonosítóját, támogatott protokollját, elérési címét és hitelesítő adatait. Az üzenetben kapott adatokat felhasználva HTTP kérést küld a megfelelő kamerának, melyre válaszként egy a kamera által készített képet kap `image/jpeg` formátumban. Ha a kamerához tartozó cím nem elérhető időtúllépés lép fel és egy megfelelő hibaüzenettel jelzi ezt. Sikeres kérés esetén a kapott kép egyedi névvel lesz ellátva, amely alapján később beazonosítható és el van mentve az ideiglenes tárolóra. Az ideiglenes tároló tartalmazza azon képeket, amelyeken még nem történt látogató beazonosítása sisakszám alapján. A folyamat utolsó lépéseként, a szerviz a kimeneti üzenetsorba elküldi a kép nevét, a kamera azonosítóját és a rögzítés időpontját.

A fényképkészítő modul, a microservice architektúrának és az üzenetalapú kommunikációnak köszönhetően, a többi komponenstől függetlenül fut, egy saját konténerben, ami által skálázható. A szolgáltatás viselkedését a konténer környezeti változóinak a beállításával lehet konfigurálni, melyek meghatározzák a használandó bemeneti és kimeneti üzenetsorokat, valamint a temporális tároló útvonalát, ahova a fotók lementésre kerülnek.

### Kamerák és protokoll

A rendszer teszteléséhez használt térfigyelő kamera Hikvision márkájú, 4 MP képminőséggel rendelkezik és internethez Wi-Fi-n keresztül kapcsolódik. Támogatja ISAPI[15]-t és besorolható az ONVIF S és G profiljába. Az interfészek által biztosítva van a kamerával

[15]Internet Server Application Programming Interface

való kommunikáció, egy HTTP kérés küldéséhez szükséges a kamera média moduljához kötött specifikus URL[16], mely szerkezete a következő részekből áll: képlekéréshez használt protokoll, a kamera hosztneve és portja, valamint a média profil cím béli útvonala és neve. A kiszolgált kép eléréséhez kötelező *digest authentication*, amely egy előzőleg meghatározott felhasználónév és jelszó alapján hitelesíti a kérést. A hitelesítő adatokat fel lehet vinni a kamera saját admin felületén keresztül. A felületen belül szükséges az ONVIF engedélyezése is.

#### Kamera szimulátor

A szolgáltatás tesztelése érdekében egy kamera szimulátor modul felelős, amely az ONVIF-hez hasonló HTTP kérésekre hallgat és visszaküld egy képet. A szolgáltatás továbbá rendelkezik képfeltöltés és törlés funkcionalitással is. A fényképkészítő komponens kéréseire véletlenszerűen kiválasztott képet küld válaszul, így pótolva a térfigyelő kamerákat komponens tesztelések esetén. A feltöltött képek különböző szcenáriumokat tükrözhetnek, ezzel is megkönnyítve a tesztelés folyamatát.

#### Felhasznált technológiák

A szolgáltatás *Singleton* mintán alapszik és *Python 3*-ban van implementálva, amely egy általános célú szkript nyelv. Az API kérések küldését a HTTP hívásokra alkalmas Requests könyvtár segítségével valósítja meg. Az üzenetbrókerrel való kapcsolat megteremtéséért felelős a *Pika*, amely egy AMQP könyvtár. Egyedi azonosítók generálásakor a *UUID* csomagot használja fel. Az üzenetek szerializációját a *JSON*[17] csomag segítségével oldja meg, a validálást pedig a *Schema validator* modul által nyújtott funkcionalitásokkal.

## 2.4. Sisakdetektáló

A képfeldolgozást végző folyamathoz tartozó mikroszolgáltatások között az első a sisakdetektáló, amely a fényképkészítő mikroszolgáltatás által készített képeken elsősorban sisakokat ismer fel. A felismerés számítógépes látás segítségével, egy YOLOv5 [2] architektúrájú detektorral történik. A kamerák által rögzített fotók sorra bemenetet képeznek a beépített neurális háló számára, amelyet egyesével dolgoz fel. A háló képes fejeket és sisakokat felismerni és lokalizálni, az adott objektumot behatároló téglalap meghatározásával. Ha a

[16] Uniform Resource Locator
[17] JavaScript Object Notation

bemeneti képen több sisak és fej található, a detektor képes az összeset felismerni. Minden felismert objektumhoz a következő tulajdonságokat társítja:

- Behatároló téglalap bal felső és jobb alsó pixel koordinátái, ezek az ún. bounding box koordináták.
- Konfidencia érték 0 és 1 között, amely azt reprezentálja, hogy a neurális háló milyen magabiztossággal ismerte fel az adott objektumot. Például egy sisak esetében mennyire "biztos" a háló abban, hogy a téglalapon belül felismert objektum egy sisak.
- Objektumosztályt képviselő egész szám, amely meghatározza, hogy a háló valójában milyen objektumot ismer fel. Ez esetben a 0 sisakot, az 1 pedig fejet reprezentál.

A komponens két fő konfigurációval rendelkezik: CPU[18] alapú, melyben minden számítást a processzor végez, vagy GPU[19] alapú, amelyben maga az objektumfelismerés a grafikus kártya használatával történik. A jobb teljesítmény érdekében a GPU alapú konténer van alapértelmezetten használva, mivel a grafikus kártyán sokkal gyorsabb a neurális háló kiértékelése. Ha a környezet, amelyben a konténer fut, nem biztosít grafikus kártyát, a CPU alapú verziót kell használni.

A mikroszolgátatásokon alapuló architektúrának megfelelően, ez a szolgáltatás is üzenetek segitségével kommunikál a rendszer többi elemével, és azoktól független, saját konténerben fut. Környezeti változókkal konfigurálható a detektor, amelyek a konténer indulásakor kerülnek beállításra.

#### Bemenet és kimenet

Az üzenetsoron keresztül két szolgáltatással van kapcsolat létesítve: a *fényképkészítővel* és a *számfelismerővel*. A fényképkészítőtől érkező üzenet tartalmazza a fénykép nevét, amely UUID-ból és fájl kiterjesztésből tevődik össze. Emellett jelen van a kép rögzítésének időbélyege és a képet készítő kamerának az azonosítója. Ez a bemenet JSON formátumú. Az üzenetben lévő adatok közül csak a fénykép nevét használja fel a szolgáltatás, a többi mezőre más mikroszolgáltatásnak van szüksége, amelyek a sisakdetektáló után lépnek működésbe. Ha a bemenet egy nem létező kép nevét tartalmazza, akkor a szolgáltatás ennek megfelelő log

[18] Central Processing Unit
[19] Graphical Processing Unit

üzenetet ír a konzolra, illetve log fájlba, de a működése nem áll le. A hibás kérést nem dolgozza fel.

Ha a bemeneti képen sikerült a detektornak lokalizálnia egy sisakot, akkor a fennebb említett formátumban elküld egy üzenetet a számfelismerőnek. Ez ugyanaz a JSON objektum, mint ami a bemenet, viszont egy új mezővel rendelkezik, amely tartalmazza a sisakot behatároló téglalap bal felső és jobb alsó pixel koordinátáit. A szolgáltatás ezt a JSON objektumot küldi tovább a számfelismerő által figyelt üzenetsorra. Az üzenet csak akkor továbbítódik a számfelismerőnek, hogyha legalább egy sisak lokalizálása sikeres volt. Ha a detektor nem ismer fel egyetlen sisakot sem, a fogadott üzenet feldolgozottnak tekinthető és a szolgáltatás várakozási állapotban kerül, a következő feldolgozási kérést várja a fényképkészítőtől.

### A szolgáltatás működése

A sisakdetektor működésének első mozzanata a konfiguráció inicializálása a környezeti változók alapján. Ezután a konfigurációt felhasználva kiválasztja a CPU-t vagy a GPU-t mint eszközt, amelyen a neurális háló kiértékelése történni fog. A detektor betöltéséhez szükség van a háló súlyait tartalmazó fájlra, amely a konfigurációban megadott elérési útvonalon található. Amennyiben a meghatározott súlyfájl nem található, automatikusan letöltődik egy környezeti változóban megadott URL-ről és lementődik a tárolóra. A kapott súlyok alapján inicializálódik a neurális háló, azaz a rendszer felépíti a hálót a memóriában a súlyok betöltésével. Az inicializáció utolsó lépése az üzenetsorra való csatlakozás. Ezáltal létrejön egy várakozási sor, amelyen a szolgáltatás fogadja a képeket. Ha ez a négy előkészítő lépés sikeres, akkor elkezdődik a képek fogadása.

Amikor egy üzenet érkezik a figyelt üzenetsoron, a kamera által rögzített kép beolvasásra kerül a megosztott tárolóról, ahová a fényképkészítő szolgáltatás korábban elmentette azt. A megfelelő képfájl az üzenetben található UUID segítségével azonosítható be. Ennek a szakasznak az utolsó lépése a feldolgozandó kép betöltése a háló kiértékelést végző eszköz memóriájába.

A következő lépés a kép feldolgozása, amely egy három lépcsős folyamat: kép előfeldolgozása, neurális háló kiértékelése és az NMS[20]. A kiértékelés részleteit és az NMS-t a 3.3. alfejezet tárgyalja. Előfeldolgozáskor a kép átméreteződik, a BGR színcsatornákról átkonvertálódik standard RGB színcsatornákra, valamint a 0-255 tartományban lévő eredeti

[20]Non-Maximum Suppression

pixelértékek visszaskálázódnak 0 és 1 közé. Az NMS és a kiértékelés folyamata a 3.3 alfejezetben kerül részletezésre. A feldolgozás eredményeképpen minden detektált objektum rendelkezni fog a behatároló téglalap koordinátáival, a detektálásra vonatkozó konfidencia értékkel és az objektumosztályt képviselő számmal.

Az utolsó lépés az eredmény meghatározása. A rögzített képeken akár több, a sisakdetektáló által felismerhető objektum is lehet, a modulnak ki kell választania egyet, amelynek a koordinátái a végső kimenetet fogják képezni. A sisakdetekor a felismert objektumokhoz csatolt tulajdonságok alapján egy rangsort állít fel. Konfigurációkor beállítható, hogy a felismert objektumokat mi alapján rangsorolja, a következő kimeneti módok közül lehet választani:

1. *Konfidencia*: A legnagyobb konfidencia értékkel rendelkező sisak koordinátáit küldi tovább.

2. *Méret*: A behatároló téglalap mérete szerint rangsorol, azaz a képen lévő „legnagyobb" sisak kerül tovább.

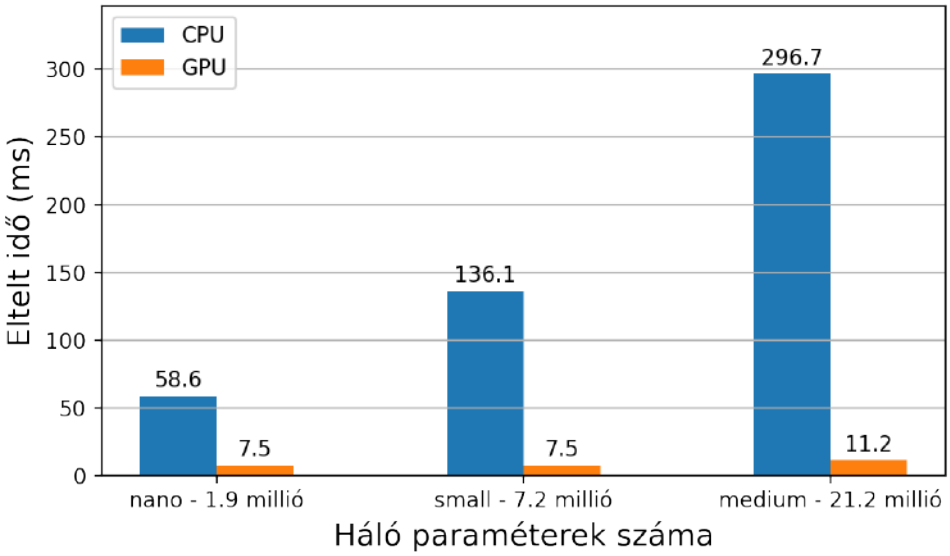
3. *Távolság a kép közepétől*: A kép közepéhez legközelebb lévő sisakot küldi tovább, kép közepe és a behatároló téglalap közepe közötti euklidészi távolság alapján.

4. *Vegyes*: Minden felismert objektumhoz egy pontszámot is csatol, amely a három tulajdonság súlyozott számtani közepének kiszámolásával jön létre. A súlyok tetszés szerint állíthatóak a szolgáltatás konfigurálásakor. A legnagyobb pontszámú objektum koordinátáit választja ki a kimenetnek. Ez az alapértelmezett kimeneti mód, a konfidenciához, mérethez és kép közepétől való távolsághoz tartozó súlyok alapértelmezett értékei rendre `0.3`, `0.5` és `0.2`.

Ha működés közben a szolgáltatás valamilyen hibába ütközik, akkor megfelelő hibaüzenet logol és megáll. Ilyen hiba lehet például az, hogy nem elérhető a GPU vagy nem lehet letölteni a súlyfájlt a megadott URL-ről.

### A szolgáltatás teljesítménye

A skálázhatóság érdekében a beépített neurális hálónak három különböző méretű változata van. A legkisebb `nano` méretű háló 1.9 millió paraméterrel rendelkezik és ez a leggyorsabb, viszont a legpontatlanabb is. Ennél nagyobb változat a `small` méretű, 7.2 millió paraméterrel, amely lassúbb, de a felismerés terén jobban teljesít. Végül létezik egy `medium` méretű háló




6. ábra. A sisakdetektálást végző neurális háló átlagos kiértékelési ideje GPU és CPU használatával. A YOLOv5 [2] architektúrának a háló paraméterek számának függvényében három különböző mérete van, konfigurációkor ezek közül lehet választani. CPU-n történő kiértékelés esetén egy kisebb háló használata nagyban lerövidíti a kiértékelés idejét.

21.2 millió paraméterrel, amely a leglassabb a három közül, viszont ezzel érhetőek el a legjobb felismerések. A három változat kiértékelési sebessége a 6. ábrán van feltüntetve, a felismerések pontosságával kapcsolatos metrikák a 3.1 alfejezetben kerülnek bemutatásra.

Teszteléskor a sisakdetektáló várakozási sorára felkerült 300 darab üzenet, véletlenszerűen választott képekkel, melyek között voltak olyanok is, amelyeken nincs sisak. Mindhárom háló méret esetében átlagolva van az inferencia időtartama. A tesztek kétszer vannak elvégezve, először CPU, majd GPU használatával. A mérések során egy Intel i7-4720HQ típusú processzoron és egy Nvidia GeForce GTX 1080 típusú videókártyán fut a sisakfelismerés. Mindkét esetben az előfeldolgozást és az NMS-t a CPU végzi, ennek a két lépésnek az átlagos időtartama rendre `2.61` és `0.55` milliszekundum.

A YOLO[21] architektúra nagy áteresztőképességet biztosít, `medium` méretű háló és GPU használatával egy kép feldolgozása átlagosan `14.36` milliszekundumba telik. Ez azt jelenti, hogy a sisakdetektáló a videókártyán futtatva megközelítőleg 69 képet képes feldolgozni másodpercenként.

[21] You Only Look Once

#### Felhasznált technológiák

A szolgáltatás a *Python3* programozási nyelvben van implementálva, az üzenetsoron keresztüli kommunikációt a *Pika* könyvtár segítségével valósítja meg. A YOLOV5 [2] architektúrájú neurális háló a *PyTorch* [48] keretrendszerben van létrehozva, továbbá a képek előfeldolgozásához a szolgáltatás az *OpenCV* [10] könyvtárat, a matematikai számításokhoz a *NumPy* [22] könyvtárat használja.

## 2.5. Számfelismerő

A számfelismerő a második szolgáltatás, ami képfeldolgozást hajt végre a *sisakdetektálótól* kapott detektált sisakot tartalmazó kép és a hozzátartozó információk alapján. A számfelismerés az *EasyOCR* [46] könyvtár segítségével történik, amely egyesével értékeli ki a képeket, és az eredménytől függően küld vagy nem küld további üzenetet a *központi szerver* által figyelt üzenetsorba. A kommunikáció üzenet brókeren keresztül történik, előre meghatározott csatornáról kap és csatornába küldi az üzeneteket.

#### Bemenet és kimenet

Az üzenetsoron keresztül két szolgáltatással van kapcsolat létesítve: a sisakdetektálóval és a központi szerverrel. A sisakdetektálótól érkező üzenet egy JSON formátumú objektum, amely tartalmazza a fénykép nevét, a kép készítésének időpontját, a képet készítő kamera azonosítóját, illetve a sisakot behatároló téglalap bal felső és jobb alsó pixel koordinátáit. Az adatok közül a szolgáltatás fénykép nevét és a sisakot meghatározó koordinátákat használja fel, a többi információ pedig a további üzenet esetén van felhasználva más komponens által.

Sikeres számfelismerés esetén a beérkező adatokhoz hozzácsatol két mezőt: a sisakon felismert szám, illetve a predikcióhoz tartozó *konfidencia* érték. A kiegészített üzenetet a központi szerver által figyelt csatornára továbbítja. Sikertelen számfelismerésnek tulajdoníthatóak a következő esetek: sikertelen képbetöltés, sikertelen számfelismerés az adott sisakon, illetve a konfidencia nem halad meg egy küszöb értéket. A kép feldolgozásának a folyamata megszakad sikertelen felismerés esetén és a temporális tároló egységen marad későbbi feldolgozás esetére, illetve a szolgáltatás nem küld tovább üzenetet a soron következő komponens számára.

### A szolgáltatás működése

A számfelismerő működésének első mozzanata, hasonlóan a sisakdetektorhoz, a konfigurációs értékek betöltése a környezeti változókból. Következő mozzanatként a komponens betölti az EasyOCR által használt *neurális hálót* és kapcsolatot létesít az üzenetbrókerrel, ahonnan egy dedikált csatornán érkeznek az üzenetek.

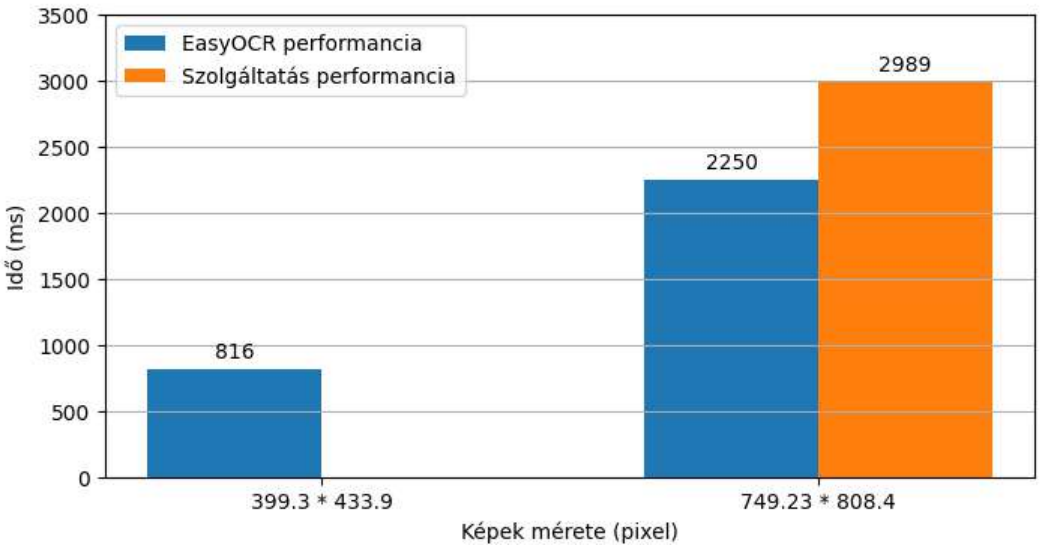
Egy üzenet érkezésekor a kép beolvasásra kerül a megosztott tárolóról, majd a sisakot meghatározó téglalap kivágása után átméreteződik a kép. A feldolgozott képeket a neutrális háló kiértékeli. Ha az adott képen nem található számjegy, akkor -30 és 30 fokos szöggel elforgatva újra próbálja értékelni, mert megtörténhet, hogy az adott képen a számjegy nincs vízszintes vagy a közeli pozícióban. Az EasyOCR a legjobb teljesítményt vízszintesen pozícionált karakterekre éri el. Az eredmény több predikciót is tartalmazhat, ezek közül a legbiztosabbat tartja meg. Ha a predikciónál kapott konfidenciaszint meghaladja a konfigurációban levő küszöb értéket, akkor az eredmény tovább küldődik a központi szerver által figyelt üzenetsorba. Sikertelen predikció esetén nem küld tovább üzenetet. Kiértékelés után a szolgáltatás figyeli tovább a dedikált csatornát, hogy újabb üzenetet dolgozzon fel.

### A szolgáltatás teljesítménye

A szolgáltatás teljesítménye függ az EasyOCR hálójának a teljesítményétől és a számítógép teljesítményétől. A számfelismerés kiértékelési ideje, illetve a szolgáltatás kiértékelési ideje külön van tesztelve két 30 képet tartalmazó adathalmazt használva. A két adathalmaz egy kisebb méretű teszt halmazból, illetve ugyan ezeket a képeket tartalmazó, de átméretezett képekből álló adathalmazból tevődik össze. A szolgáltatás a bejövő képeket előfeldolgozáskor átméretezi, így performancia csak a nagyobb pixel méretű képhalmazra van számolva. A mérések során egy Intel i5-6400 típusú processzoron történik a számfelismerés. Az előfeldolgozást és a hálóval való kiértékelést is a CPU végezi.

### Felhasznált technológiák

A számjegyfelismerő funkcionalitásai a *Python 3.9* programozási nyelv segítségével valósulnak meg. A kommunikációt a többi mikroszolgáltatással üzenet brókerrel valósítja meg, amihez felhasznált könyvtár a *Pika*. A képek betöltéséhez és előfeldolgozásához az *OpenCV* [10] és *Numpy* [45] könyvtárak vannak használva. A számfelismerés pedig az EasyOCR könyvtár segítségével van megvalósítva.



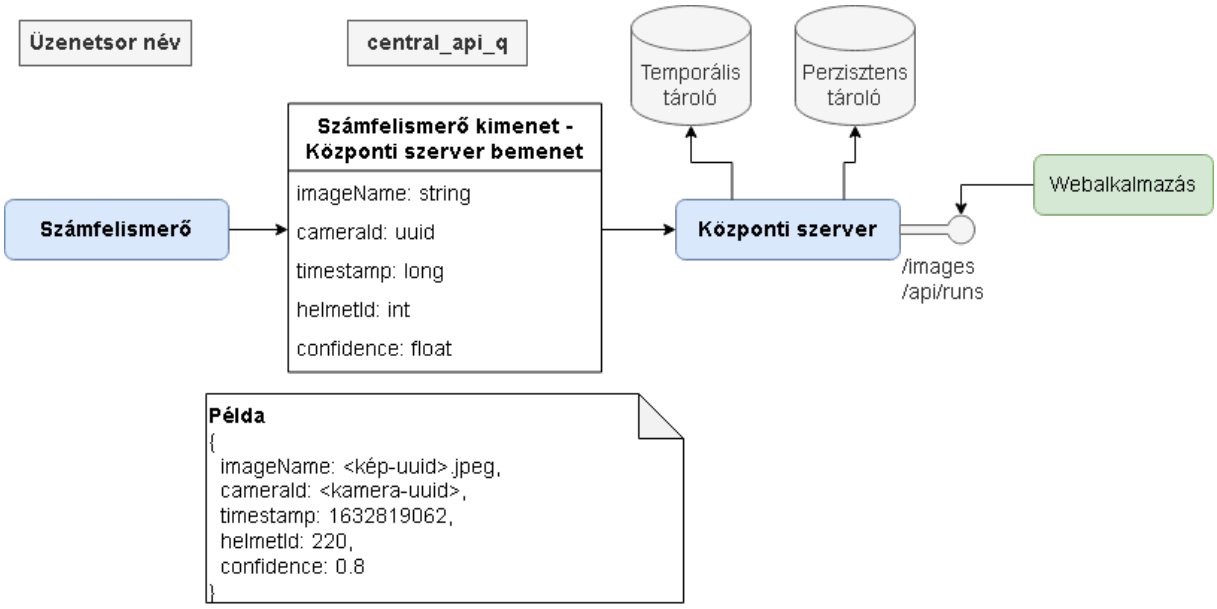

7. ábra. Megtekinthető az EasyOCR, illetve az azt felhasználó szolgáltatás átlagos kiértékelési idejük két különböző méretű adathalmazon.

## 2.6. Központi szerver

A központi szerver feladata a runok és a beazonosított fotók menedzselése, valamint a képek fizikai tárolása és elérhetősége a webalkalmazás számára. A komponens feladatköréhez tartozik szintén a QR kódok generálása és küldése e-mail formájában a látogatók számára egy run létrehozásakor.

### Bemenet és kimenet

A komponens a webalkalmazás API kéréseit szolgálja ki. Egy run entitás tartalmazza a látogató által megadott e-mail címet, hozzárendelt sisak számát és indulási időpontját, amelyhez a run befejezéskor egy végzési időpont is hozzá fűződik. A `/api/runs` végpont által biztosított a runok létrehozása, lekérése, módosítása, törlése és egy runhoz tartozó beazonosított képek listájának lekérdezése. A szerver az `/images` végponton keresztül teszi elérhetővé a lementett képeket egyedi azonosítóik szerint statikus tartalomként. A szolgáltatás mindemellett fogadja üzenetsoron keresztül a számfelismerőtől érkező üzeneteket, melyek feldolgozása során kihat a tárhelyen elhelyezett fotókra is, átmásolva az üzenethez tartozó képet a temporális könyvtárból egy megadott perzisztens tárolóhelyre.




8. ábra. A központi szerver kommunikációs diagramja, bemeneti üzenet példával és API végpontokkal, illetve a felhasznált tárolók feltüntetésével.

### A szolgáltatás müködése

A központi szerver müködésének folyamata és kommunikációs ábrája látható a 8 diagramon. Induláskor kapcsolódik az adatbázishoz és az üzenetsorhoz, majd létrehozza a temporális és perzisztens könyvtárakat, ha azok az indulás pillanatában még nem léteznek. Inicializálás után fogyasztja a számfelismerő felől érkező üzeneteket. Egy üzenet tartalmazza egy kép adatait, a kamera azonosítóját, amely készítette azt, a rögzítés idejét, a képen felismert sisakon lévő számot és a beazonosítás megbízhatóságának értékét. Az adatokat lementi az adatbázisba és a hozzátartozó fotót átmásolja a perzisztens tárolóra. Run létrehozásakor a látogató a megadott e-mail címére megkapja a képei megtekintéséhez tartozó QR kódot, melyen keresztül elérhető a webalkalmazás galériája run azonosítója szerint.

### Felhasznált technológiák

A szolgáltatás API része a kameravezérlőhöz hasonlóan *Java*-ban, illetve *Spring keretrendszer* használatával van megvalósítva, felhasználva a 2.2. alfejezetben megemlített technológiákat, eszközöket és függőségeket. A kamervezérlő adatbázisa mellett a központi szervernek is biztosítva van egy külön *PostgreSQL* alapú adatbázis. A QR kód előállítása a *ZXing*[22] [67] könyvtár segítségével biztosított, mely egy- vagy kétdimenziós vonalkódok generálására és felismerésére képes. Az e-mail küldés a *SendGrid* [12] *Java* kliensű

[22]Zebra Crossing

API könyvtárán keresztül történik. A *Spring* konfigurációs mechanizmusának köszönhetően lehetővé teszi az előzőleg említett két csomagra épülő szolgáltatások beállítását környezeti változók alapján.

A *SendGrid* egy felhőalapú *SMTP*[23]-szolgáltató, mely templatek segítségével dinamikus tartalommal képes ellátni a elküldendő e-maileket. Jövevénye a *SaaS*[24]-ra épülő tranzakciós és marketing e-mail kézbesítő szolgáltatásoknak.

## 2.7. Webalkalmazás

Az Adrenaline Eye a felhasználók számára egy webalkalmazás segítségével teszi lehetővé a kamerák, runok menedzselését. A kalandpark látogatók az alkalmazáson keresztül tekinthetik meg az kamerák által elkészített képeket. Ezen adatokat a központi szerver és a kamera vezérlő szolgáltatások biztosítják, melyek módosíthatóak.

### Kommunikáció a szerverekkel

A kommunikáció a két szerver között *REST API* [17] konvenciók alapján valósul meg az *Axios* [4] könyvtár által, aszinkron módon. Az Axios *HTTP* kéréseket küld és Promise-ként fogadja a válaszokat. A Promise az egy olyan objektum, ami helyettesít egy aszinkron feladatot, ami még nem volt kiértékelve. Két külön környezeti változó felel a központi szerver és a kamera vezérlő felé irányuló kérések elérési útvonaláért. A könyvtár segítségével a CRUD[25] műveleteket valósítja meg a szerverekkel: objektum létrehozása esetén POST, törlés esetén a DELETE, frissítés esetén PATCH, illetve listázás esetén a GET metódussal küld kérést. A runok menedzseléséért és a galéria megtekintéséért a központi szerverrel 2.6, illetve a kamerák menedzseléséért a kameravezérlővel 2.2 kommunikál. A POST és a PATCH kérések body-jában található az adat létrehozás vagy módosítás esetén, *JSON* formátumban, amik a könyvtár segítségével alakulnak át objektummá, és fordítva üzenetté. A kapott adatokat dinamikusan tárolja el és jeleníti meg az applikáció.

### Megjelenítési réteg

A megjelenítési réteg a *pages* [41] és *components* [50] részekből tevődik össze. A components rész saját, testre szabott elemeket tartalmaz, amik felhasználnak React

[23] Simple Mail Transfer Protocol
[24] Software as a Service
[25] Create Read Update Delete

komponenseket. A React komponensek előre definiált és stilizált elemek, amelyek újrahasználhatóak. A pages tartalmazza a különböző oldalakat, amik több saját, illetve alap komponensből tevődik össze. A webalkalmazás bal oldalán található egy menüsor, amivel navigálhat a felhasználó az oldalak között. Egy komponenst egy oldal többször is felhasználhat, illetve van olyan elem, amit több oldal használ fel. Ezeknek a komponenseknek lehet lokális állapotuk, ez általában az oldal betöltésekor, vagy gomb megnyomásakor változik. A *Next.js* [64] intézi a *navigációt* [18], a pages mappában található fájlok nevei lesznek az elérési útvonal nevei, illetve almappákban találhatók az egymásba ágyazott oldalak fájljai. A dinamikus útvonalválasztást sajátos fájlnévvel történik, paraméter lesz a szögletes zárójel közti név, ezáltal egyedi kéréseket küldhetünk a szervernek. Az oldalak közti átirányítás a *Link* komponenssel történik.

### Felhasználói felület

A felhasználói felület a *MUI* [51] könyvtár komponenseiből tevődik össze. Az MUI egy széles körben elterjedt React alapú UI[26] könyvtár, amely egységes kinézetet garantál előre definiált komponensek segítségével. Ezek a komponensek reszponzív és felhasználóbarát működést biztosítanak egy applikációnak. Lehetőséget ad a könyvtár ezeknek a komponenseknek a testre szabására, felhasználásukkal saját elemeket lehet definiálni. Ezek elhelyezkedése az oldalakon a *Grid* elem segítségével történik, négyzetrácsos beosztást készít az alkotóelemeknek, amik alkalmazkodnak a megjelenítési eszköz méretéhez. A webalkalmazás kellemes kinézetéért egy egyedi téma is definiálva van, ez felelős a komponensek és oldalak közti egységes kinézetért. A témázáson belül a szín paletta, a betűtípus, körvonalazás és az elemek közti térköz van beállítva. A szín paletta pedig beállítja az elsődleges és másodlagos színeket, amik a komponensek egységes kinézetét határozzák meg, illetve a különböző informatív üzenetek színei is meg vannak benne határozva. A *MUI* rendelkezik *ThemeProvider* elemmel is, ami ráruházza a bevezetett témát az egész alkalmazásra, így minden komponens stílusa megegyezik.

### Felhasznált technológiák

A webes alkalmazás funkcionalitásai a *TypeScript* [28] programozási nyelv segítségével valósulnak meg. A TypeScript a *Microsoft* által kifejlesztett programozási nyelv, ami

[26] User Interface

a *JavaScript* programozási nyelvet egészíti ki, lehetőséget adva típusok és interfészek használatára. Mivel a JavaScript bővítése, ezért a bevezetett funkcionalitások opcionálisok, minden helyes JavaScript kód helyes TypeScript kódnak számít. A típusdefiníciók alapján a TypeScrip kompilálási időben ellenőrzi a típusok használatának helyességét.

A *React* [1] a Facebook által kifejlesztett nyílt forráskódú keretrendszer, ami *Single Page Application* (egy oldallal rendelkező applikáció) építésére alkalmas. Az SPA lényege, hogy a felhasználóval történő interakció során nem új oldalak töltődnek be, hanem az adott oldal összetevői változnak. A komponensek általában XML szintaxishoz hasonló JSX, a projektben TSX-ben íródnak, JavaScript vagy TypeScript kiegészítése HTML kód használatával. Ezek az előfeldolgozás során átalakulnak megfelelő JavaScript kódrészletre.

A webalkalmazás Next.js keretrendszer felhasználásával íródott, ami React-ra épül, és kiegészíti azt több funkcionalitással, mint például szerver oldali rendereléssel és statikus export-tal. Lehetővé teszi a React komponensek szerver oldali renderelését mielőtt elküldi a kliensnek a *HTML*[27]-t, így gyorsabban töltődik be az oldal. Lehetőség van prefetching funkcionalitásra is. A Link átirányító komponensnek lehet adni egy olyan tulajdonságot, hogy azon az oldalnak, amire mutat a link, annak az előforrásait előre töltse be.

[27] Hyper Text Markup Language

# 3. Számítógépes látás

A rendszerben két mikroszolgáltatás van, amely számítógépes látást alkalmaz: a sisakdetektáló, amely a kamerák által készített képeken sisakokat lokalizál, valamint a számfelismerő, amely a képeknek azon részén próbál meg számokat felismerni, ahol a detektor sisakokat észlelt.

Jelen fejezetben bemutatásra kerülnek a két szolgáltatás által felhasznált módszerek, illetve ezen módszerek által elért eredmények.

## 3.1. Elméleti áttekintés

A sisakdetektáló a YOLO [52] detektorok családjába tartozó YOLOv5 architektúrát alkalmazza, amely egy konvolúciós neurális hálót specifikál. A számfelismerő a karakterek pozíciójának megállapításához egy konvolúciós hálót alkalmaz, az egymásutáni karakterek kiolvasásához pedig egy LSTM[28] hálót.

A következő áttekintés összegzi a konvolúciós hálók és az LSTM hálók megértéséhez szükséges elméleti alkotóelemeket az egyszerű neuronoktól és aktivációs függvényektől kezdve a neurális hálók felépítésén és tanításán át. Az alfejezet végén szó lesz a sisakdetektálással és számfelismeréssel kapcsolatos metrikákról is.

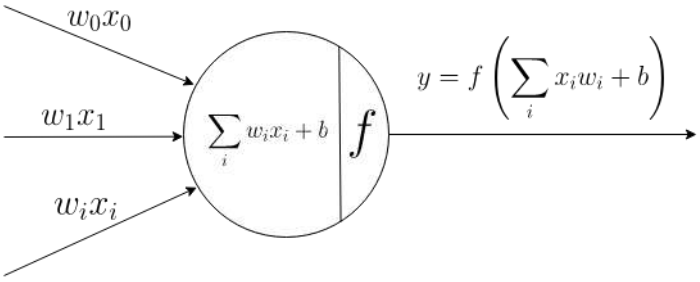
### Neuronok és aktivációs függvények

Egy neurális háló építéséhez felhasznált alapegység a mesterséges neuron (9. ábra[29]), amely a bemenetre vonatkozó súlyok (weights) és a küszöbérték (bias) kis változtatását megfelelően tükrözi a kimenet kis mértékű változásával. Hasonlóak a perceptronokhoz, viszont a kimeneti értéket egy aktivációs függvény felhasználásával számolják ki, melynek eredménye általában egy 0 és 1 közötti valós szám. Ez esetben a súlyösszeghez hozzáadódik a küszöbérték és ez az összeg bemenetet képez az aktivációs függvény számára.

Aktivációs függvények közé tartoznak a szigmoid, tanh, SiLU vagy ReLU függvények. Egy ilyen neuron esetében a paraméterek beállítása könnyebb, mivel a kimenet nem egy 0 vagy 1 szélső érték, hanem egy valós szám. Ennek a számnak a felhasználásával a súlyok változtatásakor pontosan megállapítható, hogy egy-egy paraméter állítása milyen irányban és

[28] Long Short Term Memory

[29] Forrás: saját szerkesztés https://cs231n.github.io/neural-networks-1/ alapján, utolsó megtekintés dátuma: 2022-04-21




9. ábra. Egy neuron $f$ aktivációs függvénnyel. Más neuronoktól érkező bemenetek ($x_i$) és a bias ($b$) alapján meghatározza a kimenetet ($y$). Minden bemenethez tartozik egy $w_i$ súly, amely egy véletlenszerű értékkel inicializálódik és tanítás alatt változik.

mekkora mértékben mozdította el a neuron kimeneti értékét.

### Neurális hálók

A mesterséges neurális hálók (ANN[30]) az emberi agy működését utánozzák. Több neuron strukturált összekapcsolása révén olyan modellek építhetőek, amelyek képesek procedurális úton nehezen megközelíthető feladatok megoldására, mint például egy objektum felismerése egy képen.

Olyan problémákra nyújtanak megoldást, amelyeket a klasszikus programokkal nehéz megoldani. Ezek a modellek egy teljesen más irányból közelítik meg az adott feladatot, nincsenek beprogramozva egy adott feladat megoldására és nincsenek ellátva a feladatra vonatkozó szabályokkal. Egy lehetséges megoldást tanítás során, a belső paraméterek állításával kapnak.

A neurális hálókat univerzális approximátoroknak is nevezik [26], hiszen képesek bármely függvény értékeinek a megközelítésére. Ez esetben a probléma reprezentálható egy ismeretlen függvény értékeinek a meghatározásaként.

Egy neurális háló tanításához szükség van egy módszerre, amely meghatározza, hogy a neurális háló a súlyainak egy adott állapotában mennyire jól tudja elvégezni a kitűzött feladatot. Ezt egy hibaaránnyal lehet kifejezni, amely a költségfüggvény felhasználásával számolható ki. A költségfüggvény értéke megadja, hogy mekkora különbség van a modell elvárt és aktuális kimenete között.

A költségfüggvény megválasztása a feladattól függ. Regresszió esetén gyakori az MSE[31] költségfüggvény használata, több osztályú klasszifikáció esetén elterjedt a kereszt entrópia

[30] Artificial Neural Network
[31] Mean Squared Error

(cross entropy).

Egy neurális háló betanítása olyan súly értékek megkeresését jelenti, amelyekre a költségfüggvény értéke minimális. Tanítás során a modell a bemeneti adatok feldolgozásával és az eredmény kiértékelésével lépésenként tanítható be. Egy ilyen folyamat során minden egyes iterációval változnak a neurális háló súlyai, egészen addig, amíg a modell konvergál, azaz a költségfüggvény megközelíti a minimumot. Ebben az állapotban a modell betanítottnak tekinthető, hiszen elfogadható hibaaránnyal képes megoldani az adott feladatot.

A költségfüggvény értékének a minimalizálását a gradiens leereszkedés (gradient descent) optimalizációs módszerrel éri el . A költségfüggvény szerinti hiba visszaterjesztésével (error backpropagation) [56] és a láncszabály (chain rule) [66] alkalmazásával kiszámíthatóak a gradiensek, amelyek a költségfüggvénytől visszafele a háló első rétegéig megadják, hogy a súlyokat milyen irányban kell változtatni annak érdekében, hogy a hiba minimalizálódjon. Ennek a változtatásnak a mértékét egy hiperparaméter, a tanulási ráta (learning rate) szabályozza, amely úgy értelmezhető, mint a negatív gradiens irányba való lépés nagysága. Általánosan a súlyok lépésenkénti változtatása a következőképpen írható le:

$$w_{actual} = w_{prev} - \alpha \nabla f(w_{prev}) \tag{1}$$

ahol $w$ a súlyokat, $\alpha$ a tanulási rátát, $f$ az aktivációs függvényt és $\nabla f(w)$ a függvény gradiensét jelöli.

A legelterjedtebb optimalizációs módszer a sztochasztikus gradiens leereszkedés (SGD[32]), melynek több változata létezik: SGD-Momentum, Nesterov momentum, RMSProp, AdaGrad vagy Adam [55].

A megfelelő tanulási ráta kiválasztása kulcsfontosságú, ha túl kicsi, nagyon kis lépéseket fog tenni a modell a költségfüggvény minimuma fele. Ha túl nagy, akkor előfordulhat, hogy átugorja a költségfüggvény minimumát és a minimum körüli tartományban ragad.

Egy neurális háló betanítása egy adathalmazon a következő lépésenkénti folyamatot jelenti: egy felcímkézett adatpontra (például egy képre) a háló kiértékelése, ezzel meghatározva a költségfüggvény értékét, a hiba visszaterjesztése, azaz a gradiens meghatározása, majd a súlyok változtatása a gradiens szerinti negatív irányba. A tanítást több epoch-ból áll, egy epoch során az előbb leírt folyamat végbemegy a tanítási adathalmaz minden elemére, amelyek kötegenként (batch) kerülnek feldolgozásra. Minden epoch végén kiértékelődik a háló teljesítménye egy

[32]Stochastic Gradient Descent

validációs adathalmazon. Ezt az adathalmazt a megfelelő hiperparaméterek megállapítására is szokás használni. Végül a tanítás után a háló teljesítménye egy teszt adathalmazon van lemérve. A három adathalmaz egyenként különálló, páronként nem tartalmaznak közös adatpontokat.

### Konvolúciós neurális hálók

A konvolúciós hálók (CNN[33]) [49] a neurális hálók azon speciális változatai, amelyek konvolúciós rétegeket alkalmaznak. Képesek adaptívan megtanulni a felismerni kívánt objektum vizuális jellemzőit és ezekből egy hierarchiát kialakítani. Például egy arcfelismerést végző konvolúciós háló esetében a háló elején lévő rétegek megtanulnak felismerni egy kontűrt vagy egy ívet, a közepén lévők egy fület, egy szemet, egy szájat, míg a háló végén lévő rétegek megtanulnak ezen jellemzők alapján felismerni egy arcot. A hagyományos hálóktól csak felépítésükben térnek el, így az eddig bemutatott neuron szerkezet, aktivációs függvények és optimalizációs módszerek ugyanazok. Architekturális jellemzőikből kifolyólag kevesebb súly paraméterrel rendelkeznek mint a hagyományos hálók. Széles körben elterjedtek színes képek feldolgozására, az elmúlt években dominánsá váltak a számítógépes látással kapcsolatos feladatok megoldásában, elsősorban az objektumfelismerés témakörében [62].

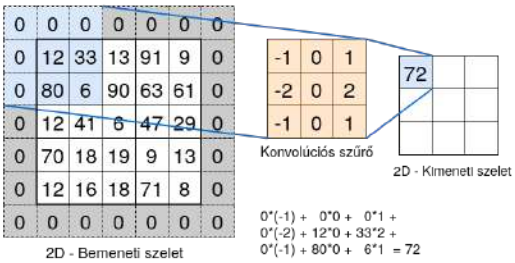
Központi alkotóelemük a konvolúciós kernel, amely egy tanítható súlyokat tartalmazó mátrix. A kernelt végig léptetve a két dimenziós bemenet pixelein, létrejön az ugyancsak kétdimenziós kimenet, a kernel alatti pixel értékek súlyzott összegének a kiszámításával. A 10. ábra[34] szemléltet egy példát a műveletre. Ha a bemeneti adat három dimenziós, az ennek megfelelő kernelnek is van egy harmadik, mélységi dimenziója. Ahhoz, hogy a széleken lévő pixelek fölött is elhaladhasson a kernel, nullával szokás kipótolni a kép széleit, ezzel semlegesítve a kipótolt terület befolyását a súlyösszegre.

Egy konvolúciós réteg több kernelt felhasználva végig konvolválja a bemeneti adatot a kimenetet létrehozásához, amelynek nem csak szélessége és magassága, hanem mélysége is van. A kimenet mélységét a kernelek száma határozza meg, minden kernel létrehoz a kimenetben egy mélységi dimenziót kitöltő szeletet.

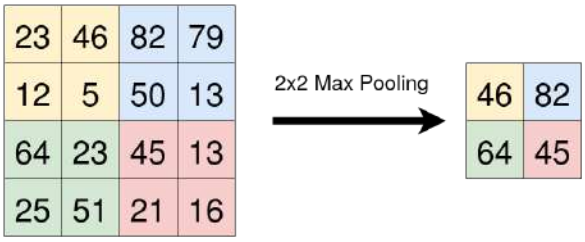
Konvolúciós rétegek között pooling rétegek helyezkednek el, amelyek a konvolúcióhoz hasonlóan, egy szűrő segítségével összegzik minden mélységi dimenzióban a vizsgált terület alatti pixelek értékeit, csökkentve a háló paramétereinek a számát. Ez az összegzés általában

[33]Convolutional Neural Networks

[34]Forrás: saját szerkesztés https://cs231n.github.io/convolutional-networks/ alapján, utolsó megtekintés dátuma: 2022-04-21




10. ábra. Konvolúciós kernel működése. Az 5x5-ös bemeneti szelet ki van egészítve 7x7 méretűre, így a 3x3-as kernel 2-es lépésközt használva végig konvolvál azon, létrehozva a 3x3-as kimeneti szeletet. A kernel alatti értékek beszorzódnak a szűrő megfelelő elemével, majd az így kapott értékek összegzése megadja a kimenetet.




11. ábra. Maximum pooling szűrő működése. A kijelölt 2x2-es terület alatti legnagyobb pixel érték kerül a pooling réteg utáni konvolúciós rétegbe. Ez a réteg 2-es lépést (stride) haszálva negyedeli az őt megelőző konvolúciós réteg paramétereinek számát, miközben a legfontosabb értékek továbbkerülnek a kimenetbe.

maximum kiválasztást vagy áltagszámolást jelent. Például maximum pooling esetében csak a vizsgált terület alatti legnagyobb pixel érték kerül a pooling réteg kimenetébe. A max pooling réteg működése a 11. ábrán[35] van szemléltetve. A konvolúciós és pooling rétegekhez tartozik egy hiperparaméter (stride), amely szabályozza, hogy mekkora lépéseket kell tenni a kernelekkel a kép való konvolváláskor.

A konvolúciós hálók utolsó rétege általában egy sűrűn összekötött réteg (FC[36]), amely a hagyományos neurális hálókban található. Ennek a rétegnek a kimeneti dimenziói megegyeznek a lehetséges objektum osztályok számával. Példa képeket osztályozó konvolúciós hálóra a 12. ábrán[37] található.

Népszerű konvolúciós neurális architektúrák közé tartozik a LeNet[32], AlexNet[29], VGGNet[59] és a ResNet [23]. Ezen architektúrákkal kapcsolatos kutatások elsősorban az ImageNet[14] adathalmazt használják.

### Detektorok

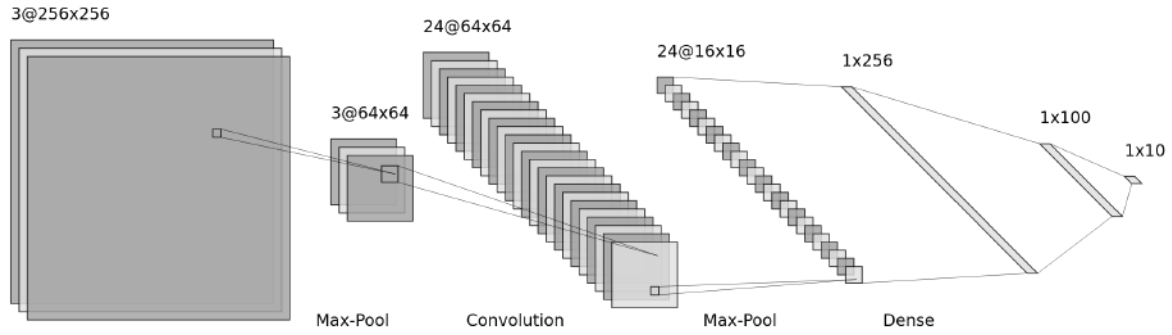
A detekorok olyan modellek, amelyek képesek fényképeken felismerni és lokalizálni különböző osztályokba tartozó objektumokat. Két nagyobb kategóriába sorolhatóak: klasszikus megközelítés alkalmazó módszerek, mint például a Haar kaszkád [65], a HOG[38][13] vagy a

[35]Forrás: saját szerkesztés https://cs231n.github.io/convolutional-networks/ alapján, utolsó megtekintés dátuma: 2022-04-21

[36]Fully Connected

[37]Forrás: saját szerkesztés https://alexlenail.me/NN-SVG/LeNet.html segítségével, utolsó megtekintés dátuma: 2022-04-22

[38]Histogram of Oriented Gradients




12. ábra. Példa egy osztályozást végző konvolúciós háló architektúrájára. Az első 16x16-os, 16-os lépést alkalmazó maximum pooling réteg 1:16 arányban csökkenti a három csatornás bemeneti kép méretét, amelyből az utána következő konvolúciós réteg 24 szeletet hoz létre. Ezt követi még egy maximum pooling réteg, amely ugyancsak 1:16 arányban csökkenti a méretet, majd három sűrűn összekötött réteg. A legutolsó sűrű réteg kimenete 1x10-es méretű, tehát a háló a 10 kategória egyikébe sorolja a bemeneti képen lévő objektumot.

SIFT[39][42], és a neurális hálókat felhasználó, gyors detektálást végző modellek:

1. R-CNN[40] [19] hálók, amelyek két lépésben végzik el a feladatot: először téglalap alakú ROI[41] területeket választanak ki a képen, majd minden téglalapon objektumfelismerést végeznek CNN hálókkal.

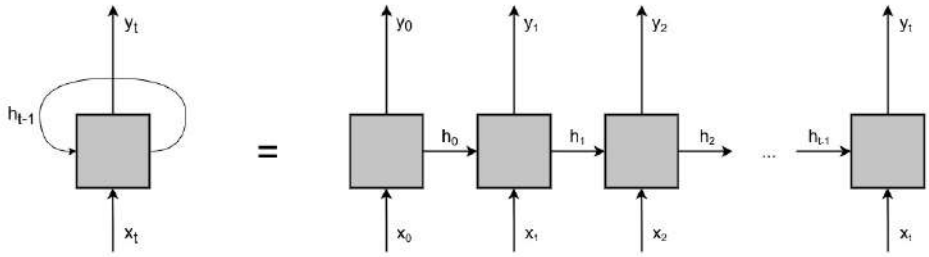
2. SSD[42] detektorok [36], amelyek a VGG architektúrát egészítik ki hierarchikus konvolúciós rétegekkel. Úgynevezett kép piramis reprezentációkat csatolnak a VGG végére.

3. YOLO [52] családba tartozó modellek, amelyek régiókra osztják a képet, majd objektumokat behatároló téglalapokat és hozzájuk tartozó konfidencia szinteket, valamint régiókhoz tartozó objektumosztály valószínűségeket prediktálnak. A két procedúra párhuzamosan történik, a kimenet a két predikció eredményének összefűzéséből jön létre. Gyorsabb mint az R-CNN megközelítés, mivel a CNN háló kiértékelése csak egyszer meg végbe, nem szükséges külön-külön minden kép régió elemzése.

### Visszacsatolásos neurális hálók

A visszacsatolásos neurális hálók (RNN[43]) [35] képesek változó hosszúságú adatszekvenciák feldolgozására, úgy hogy az aktuális kimeneti érték meghatározásakor

[39] Scale Invatiant Feature Transform
[40] Region-based Convolutional Neural Networks
[41] Region of Intereset
[42] Single Shot Multibox Detector
[43] Recurrent Neural Networks




13. ábra. RNN háló idő szerint kibontva. $h_{t-1}$ jelöli a memória értékét, amely minden iterációban befolyásolja az $y_t$ kimenet értékét és újraszámolódik a $t+1$-dik iteráció számára.

figyelembe veszik a szekvencia korábbi elemeire kiszámolt kimeneti értékeket is. Többek közt karakterfelismeréskor használatosak, a kiolvasandó szöveg karakterei közötti strukturális információt is képesek felhasználni. Ilyen strukturális jellemző lehet az, hogy egy magánhangzó után gyakran egy mássalhangzó következik vagy a karakterek valamilyen mintát követve alkotnak szavakat.

Ezeket a hálókat olyan neuronok alkotják, amelyek figyelembe veszik a bemeneti szekvencia $t-1$-dik elemére kiszámolt kimenetetet a $t$-dik bemeneti elem feldolgozásakor, így a modell által már korábban levont következtetések befolyásolják az aktuális kimenetet. Úgy is értelmezhető, hogy ezeknek a neuronoknak van memóriája, amelyben képesek egy korábbi lépésben kiszámolt eredményt tárolni. Egy ilyen modell idő szerinti bontása a 13. ábrán[44] látható.

Az RNN hálók beleütközhetnek az információ lemorzsolódásának problémájába [24], amely abban áll, hogy egy korábbi memória érték befolyása egyre kisebb és kisebb lesz az aktuális kimenetre nézve, ahogy iterációnként egyre távolabb kerül az aktuális lépéstől. Például az $n$-dik iterációban a kimeneti értékre az $n-1$-ben kiszámolt $h_{t-1}$ memóriaértéknek lesz a legnagyobb hatása, $n-2$-dik lépésben meghatározott $h_{t-2}$ memóriának már kisebb befolyása van, hiszen azt a modell már módosította egyszer a $n-1$-dik lépésben. Következmény képpen az RNN modelleknek csak rövid távú memóriájuk van.

### LSTM neurális hálók

Az LSTM neurális hálók [25] a visszacsatolásos hálók továbbfejlesztett változatai, amelyek rövid- és hosszútávú memóriával is rendelkeznek. A kimenet meghatározásához mindkét memóriatípust felhasználják és a bemeneti szekvencia lépésenkénti feldolgozása közben

[44]Forrás: saját szerkesztés https://colah.github.io/posts/2015-08-Understanding-LSTMs/ alapján, utolsó megtekintés dátuma: 2022-04-25

mindkettő változhat.

Egy ilyen háló a klasszikus RNN hálókkal ellentétben nem azt figyeli, hogy az egyel korábbi lépés hogyan befolyásolja a jelenlegit, hanem inkább azt határozza meg, hogy az eddig megtanult memóriában tárolt értéket hogyan kell változtatni a jelenlegi bemenet alapján. Ha az aktuális bemenetnek nagy jelentősége van, akkor nagy súlyt rendel hozzá és így nagyobb mértékben fog változni a hosszú távú memória állapota.

### Felhasznált metrikák

A következőkben bemutatásra kerülnek a sisakdetektálással és a számfelismeréssel kapcsolatos metrikák.

Három metrika van megfigyelve a számfelismerő neurális háló működésénél: pontosság, precizitás és visszahívás (recall). A pontosság a helyes predikciók és az összes predikció aránya. A precizitás a visszatérített (pozitív) példányok és a releváns példányok aránya, a felismerés a helyesen visszatérített elemek és a releváns elemek aránya.

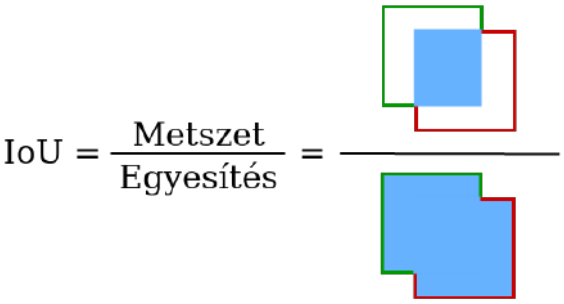
Az objektumdetektálás klasszifikációból és lokalizációból áll. Ahhoz, hogy a detektálást sikeresnek lehessen tekinteni, a helyes kategorizálás mellett a lokalizáció is pontos kell legyen, azaz közel kell essen az objektumot behatároló téglalap tényleges és modell által meghatározott pozíciója. Erre használatos metrika az IoU[45] (14. ábra[46]), amely megadja, hogy mennyire helyes a lokalizáció. A valódi és a prediktált téglalapok metszetének és egyesítésének az aránya, amelyhez tartozik egy 0 és 1 közötti szabadon választható küszöbérték. Az IoU arányt összehasonlítva a küszöbértékkel, eldönthető hogy egy adott detektálás helyes-e:

1. IoU $\geq$ küszöbérték $\Rightarrow$ Helyes pozitív (true positive), azaz a modell pontosan határolta be az objektumot, ahhoz hogy a detektálást sikeresnek lehessen tekinteni.

2. IoU $<$ küszöbérték $\Rightarrow$ Helytelen pozitív (false positive), a prediktált behatároló téglalap nem illeszkedik eléggé az objektum valódi helyzetéhez, a detektálás sikertelen.

Helytelen negatív (false negative) esetben a modell rosszul kategorizál vagy nem ismer fel egyetlen objektumot sem a képen. Objektumdetektálás tematikájában a helyes negatív (true negative) eseteket nem szokás figyelembe venni, hiszen ezek a képnek azon részeit jelenti, ahol nincs felismerhető objektum.

[45] Intersection over Union
[46] Forrás: saját szerkesztés [47] alapján.




14. ábra. IoU metrika kiszámításhoz el kell osztani a behatároló téglalap valódi (zöld) és a prediktált (piros) területének metszetét a két terület egyesítésével.

A globális, minden osztályt figyelembe vevő átlagos precizitás (mAP[47]) az objektumdetektálás alapvető mérőszáma. Megállapításához kiszámolódik az osztályonkénti 11 pontos interpolált átlagos precizitást (AP[48]) [38]. Ez a mérőszám egy adott objektumosztály esetében egyenlő a precizitás-visszahívás görbe alatti területtel (AUC[49]), amelynek megközelítő értékét interpolációval lehet kiszámolni. Ez a terület minél nagyobb, annál jobban teljesít a modell. Első lépésként 11 ponttal fel kell osztani a 0 és 1 közötti visszahívás tartományt egyenlő részekre. Ezután a görbén lévő „fűrészfogakat“ ki kell simítani, úgy hogy adott visszahívás értékhez társítani kell a tőle jobbra lévő legközelebbi maximális percizitást. Az így kapott visszahívás szinteken $(0, 0.1, 0.2, \ldots 0.9, 1.0)$ meg kell mérni a maximális precízitást, majd ezeket átlagolni kell. Az eredmény az adott osztályra vonatkozó átlagos precizitás, kiszámolását a 15. ábra szemlélteti. A folyamatot az alábbi képlet összegzi:

$$AP = \frac{1}{11} \sum_{r \in \{0, 0.1, \ldots, 0.9, 1\}} P_{interpolated}(r)$$

*Jelölés:* $AP$ *- egy osztályra vonatkozó átlagos precizitás,* $r$ *- visszahívás küszöb,* $P_{interpolated}(r)$ *-* $r$ *visszahívás küszöbhöz tartozó maximális precizitás*

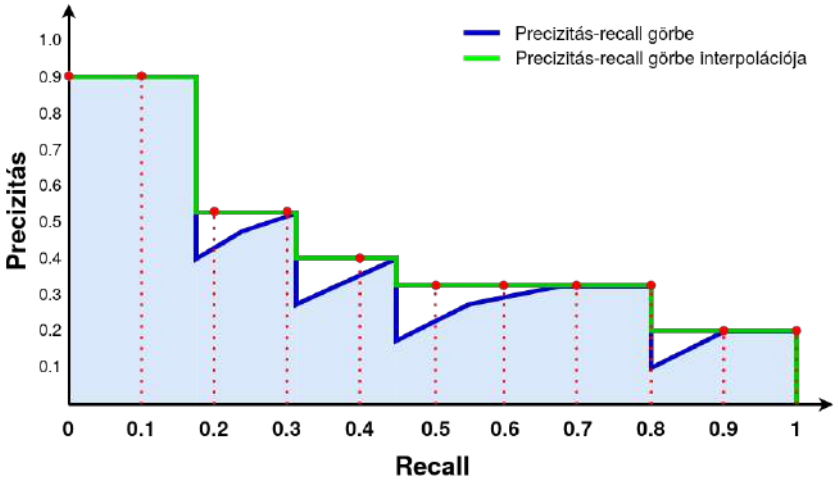
A globális mAP kiszámolásához ezeket az osztályonkénti AP értékeket kell átlagolni, a következő képlettel számolható:

$$mAP = \frac{1}{n} \sum_{k=1}^{n} AP_k$$

*Jelölés:* $n$ *- osztályok száma,* $AP_k$ *- k-dik osztályra vonatkozó átlagos precizitás*

Az eddigi mAP csak osztályozási metrika, ahhoz hogy a lokalizáció pontosságát is

[47] mean Average Precision
[48] Average Precision
[49] Area Under Curve




15. ábra. Példa egy osztályra vonatkozó precizitás-visszahívás görbe alatti terület (AUC) meghatározására 11 pontos interpolációval. A görbe alatti világoskék terület nagysága megközelítőleg a 0, 0.1, ..., 0.9, 1 visszahívás küszöbökön mért maximális precizitás értékek átlagolásával számolható ki. Megközelítés, mivel a görbe (kék) és az interpolációja (zöld) közé eső terület hozzáadódik az AUC-hez.

figyelembe vegye, szükség van egy IoU küszöbre, amely szerint el lehet dönteni, hogy egy objektum detektálása helyes pozitív vagy helytelen pozitív kategóriába sorolható precizitás és visszahívás megállapításakor. Globális átlagos precizitáshoz leggyakrabban használt IoU küszöb a $0.5$, így jön létre a $mAP@0.5$[50] metrika, amely a legfontosabb objektumdetektáláshoz használt mérőszám. A kategorizálás helyességét és a lokalizáció pontosságát is méri. Értéke 0 és 1 közé esik, minél nagyobb, annál jobban teljesít a detektor.

A mAP@0.5 iker metrikája a *mAP@0.5:0.95*, amely $0.5$, $0.55$, $0.60$, ..., $0.95$ IoU küszöbökkel számolt globális átlagos precizitások átlagolásának az eredménye. A következő képlet szerint lehet kiszámolni:

$$\text{mAP@0.5:0.95} = \frac{1}{10} \sum_{IoU \in \{0.5,\, 0.55,\, ...,\, 0.90,\, 0.95\}} mAP@IoU$$

A mAP@0.5-nél szigorúbb mérőszám, mivel magas IoU küszöböknél csak nagyon pontos lokalizáció esetében fogad el egy detektálást helyesnek. A magas IoU küszöbbel számolt tagok általában lefele húzzák az átlagot.

[50] mean Average Precision at 0.5 IoU threshold

## 3.2. Probléma feltárása

A rendszer egyik fő funkcionalitása a kamerák által rögzített képek társítása egy-egy látogatóhoz. Ennek megvalósításához a látogatási idő mellett szükség van a képen megjelenő sisak számának meghatározására.

A rendszer a sisakon lévő szám felismerését két lépésben hajtja végre. Először az objektumfelismerést végző komponens egy sisakot lokalizál a kamera által rögzített képen. Ha ez sikeres, akkor a számfelismerő megpróbálja beazonosítani a sisakon lévő számot. Ehhez a képnek csak azon részét vizsgálja, amelyen sisak található.

Ez alapján a megoldandó probléma két részre osztható: sisakdetektálás egy képen, valamint számfelismerés a bemeneti kép egy kisebb régióján, ahol egy sisak detektálva van. A következő két alfejezetben ezeknek az alproblémáknak a lehetséges megoldásai, valamint a rendszer által alkalmazott módszerek kerülnek bemutatásra.

A sisakon lévő szám kiolvasását végezhetné egyből a számfelismerő modul, viszont a nagy méretű bemeneti kép a feldolgozása lassú lenne. Mindemellett az esetek többségében a felismerés hibás lenne, hiszen a képnek csak egy kis régiója releváns a sisakszám kiolvasásához. A számfelismerő nem képes megfelelőképpen behatárolni ezt a részt. Ezért szükséges a sisakdetektáló is, amely a számfelismerőnél sokkal hatékonyabban és pontosabban képes elvégezni ezt a feladatot.

Felismerés és detektálás esetében is lehetséges, hogy a felismerendő objektumok közül egy sincs a képen. Ebben az esetben a detektálásnak nincs kimenete, hiszen a modell nem észlel objektumot a képen. A felismerésnek mindig van kimenete amely egy osztályt jelölő címkéből és az ehhez tartozó magabiztossági értékből áll. A modell kis konfidenciával, de bekategorizálja a bemeneti képet az egyik osztályba. Az ehhez hasonló téves kimeneteket ki lehet szűrni egy konfidencia küszöb használatával: egy felismerés csak akkor lesz érvényes, ha a konfidenciája nagyobb a küszöbnél.

## 3.3. Sisakdetektálás

A rendszerben a sisakdetektálás célja a kép azon régiójának meghatározása, amelyen a számfelismerő modul hatékonyan kereshet számokat. Ezáltal a sisakdetektálás egy elengedhetetlen előfeldolgozási lépésnek tekinthető. Ez a rész a 2.4. alfejezetben bemutatott mikroszolgáltatás neurális hálóját részletezi.

### Lehetséges megoldások

A sisakot tartalmazó régió meghatározására több számítógépes látást alkalmazó módszer létezik:

1. *Póz becslés*, amely megadja a képen lévő emberi alak végtagjainak és fejének pozícióját. Ez alapján könnyen meghatározható a sisak pozíciója is.

2. *Pixel szintű szegmentáció*, amelynek felhasználásával eldönthető, hogy konkrétan mely pixelek tartoznak a sisakhoz.

3. *Objektum detektorok* alkalmazása, amelyek egy fényképen fel tudnak ismerni objektumokat és meg tudják adni az objektumot behatároló téglalap pozícióját.

### YOLOv5 háló architektúra

A rendszerben a sisakdetektáló modul YOLOv5 neurális hálót alkalmaz [2], amely a YOLO modellek családjába tartozik, a YOLOv4 [8] továbbfejlesztett változatának tekinthető. A projektben használt változata PyTorch v1.7 keretrendszerben íródott.

A hálózatot két részre lehet osztani: gerinc (backbone) és a rá csatolt fej (head). A gerincet a CSP-Darknet53[51] [8] konvolúciós háló alkotja, amely köteg normalizációt (batch normalization) és SiLU aktivációs függvényt alkalmazó konvolúciós rétegekből, valamint három konvolúciós réteget magába foglaló Conv3 rétegekből áll. A YOLOv5-ben alkalmazott DarkNet háló utolsó rétege egy SPPF[52] bottleneck elem, amely három hierarchikus max-pooling művelet eredményét illeszti össze. A fej is hasonló Conv3 rétegekből és konvolúciós rétegekből áll, emellett alkalmaz felskálázó rétegeket (upsample) és felhasználja a gerinc néhány rétegének kimenetét is. A rétegek szélességi és mélységi újraskálázásával változtatható a súly paraméterek száma, az Adrenaline-Eye projekt `nano`, `small` és `medium` méretű hálókat használ.

A tanításhoz használt költségfüggvény három komponens összegéből áll:

1. *Lokalizációs hiba*, amely a behatároló téglalap valódi és prediktált pozíciója közti különbség alapján számolható.

2. *Klasszifikációs hiba*, amely bináris kereszt-entrópiával számolódik és azt számszerűsíti, hogy a modellnek sikerült-e helyes osztályba sorolnia a lokalizált objektumokat.

[51] Cross Stage Partial Network
[52] Spatial Pyramid Pooling - Fast

| | | Tanítás | Validálás | Tesztelés | Összesen |
|---|---|---|---|---|---|
| Tanítási halmaz | Sisak objektumok száma | 18196 | 4285 | 5455 | 27936 |
| | Fej objektumok száma | 76852 | 18757 | 21671 | 117280 |
| | Képek száma | 8045 | 2011 | 2515 | 12571 |
| Finomhangolási halmaz | Sisak objektumok száma | 450 | 95 | 94 | 639 |
| | Fej objektumok száma | 0 | 0 | 0 | 0 |
| | Képek száma | 199 | 49 | 63 | 311 |
| | Képek százalékos eloszlása | 64% | 16% | 20% | 100% |

1. táblázat. Sisakdetektor tanításához, validáláshoz és teszteléshez felhasznált adathalmazok összetétele.

3. *Objektum találati hiba*, amely azzal foglalkozik, hogy a behatároló téglalap tartalmaz-e felismerendő objektumot. A modell ez alapján tanulja meg elkülöníteni a detektálandó objektumokat a háttértől.

A YOLO hálók kimenete több behatároló téglalapot is tartalmazhat ugyanarra a detektált objektumra. A modell egy objektumhoz tartozó végső téglalap kiválasztását az NMS algoritmussal végzi, amely ugyanarra az objektumra vonatkozó téglalapok közül kiválasztja a legnagyobb konfidenciával rendelkezőt. IoU mérőszám alapján dönti el azt, hogy két téglalap ugyanarra az objektumra mutat-e. Ha két téglalap közti IoU nagyobb mint $0.50$, a kisebb konfidenciával rendelkező törlődik.

A YOLOv5 bizonyul a legjobb megoldásnak az alternatívák közül, mivel ez a dekektor képes megfelelő gyorsasággal és pontossággal megállapítani a fényképeken lévő sisak helyét, úgy hogy más objektumdetektálókhoz képest jóval kevesebb számítási kapacitást használ.

### Adathalmaz

A hálók tanításához két adathalmaz van létrehozva: egy nagyobb betanítási halmaz, amely főként építkezéseken készült képeket tartalmaz és egy kisebb finomhangolásra használt adathalmaz, amely kalandparkban készült képekből áll. Utóbbi a projekt keretében került összeállításra.

A betanítási halmaz a Safety Helmet Detection [57] és a Safety Helmet Wearing [58] adathalmazok összeolvasztásából jött létre. Összesen 12571 színes képet tartalmaz, amelyek mind 416x416-os felbontásúak. Két objektumosztály van felannotálva: munkások fején lévő sisakot reprezentáló `helmet` osztály és sisak nélküli fejet jelölő `head` osztály. Ezekből az objektumokból rendre 27936 és 117280 található az összes képen egybevéve.

A finomhangoláshoz használt adathalmaz a kolozsvári Adrenalin Parkban készült képeket, valamint saját tesztvideókból kivágott képeket tartalmaz. Összesen 311 fényképből áll, csak sisak objektumok vannak felcímkézve, amelyből összesen 639 található. Sokkal nagyobb felbontásúak a képek, mint a betanítási halmazban, a parkban készült képekre átlagban a 3000x4000 méret a jellemző, a videóból kivágottak mind 2560x1440-es méretűek.

Az adathalmazok címkéi eredetileg PascalVOC formátumúak, viszont előfeldolgozás során átkonvertálódtak YOLO formátumba. Mindkettő 64:16:20 arányban van felosztva tanulási, validációs és teszt halmazokra, ennek egy részletes bontása a 1. táblázatban olvasható. A betanítási halmaznak egy olyan verziója is létezik, amely csak sisakok annotációit tartalmazza.

### A tanítás részletei

A hálók tanítása két szakaszból áll: betanítás és finomhangolás. Mindkét szakaszban az elvégzett tanítási kísérleteket a következő szempontok alapján lehet kategorizálni:

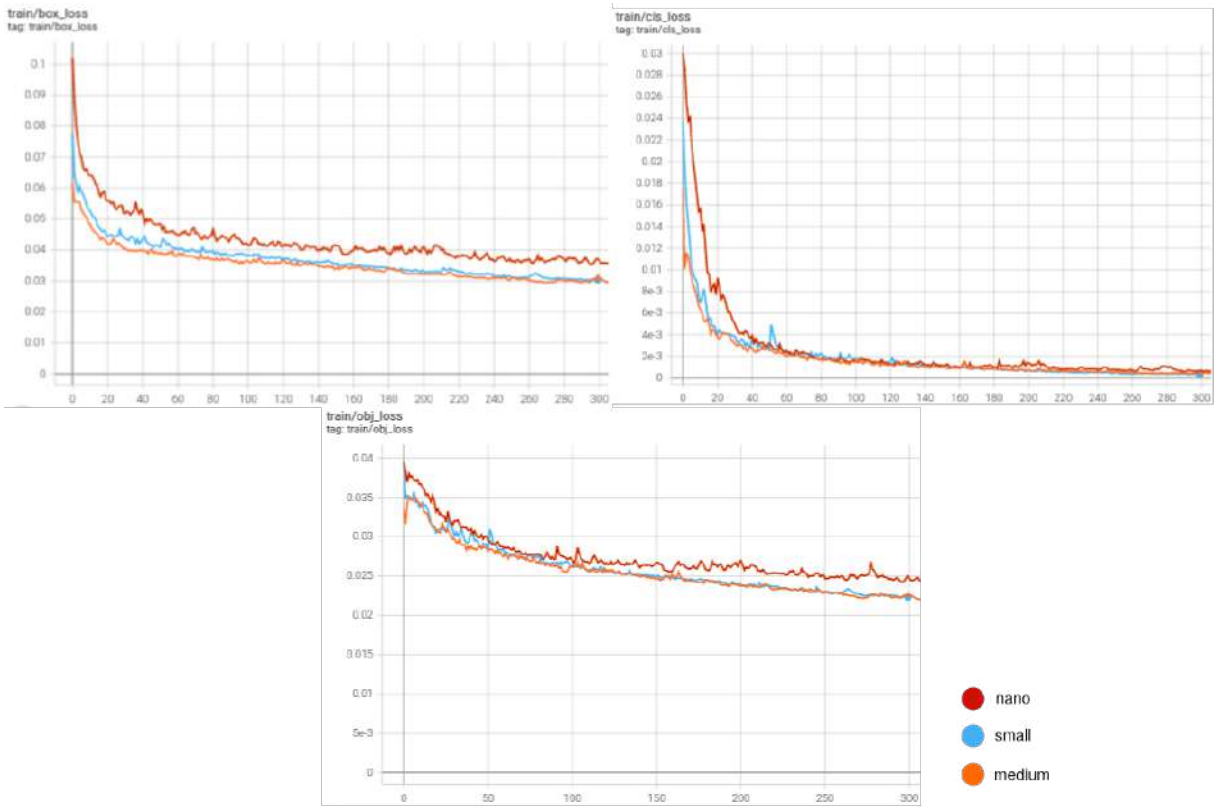
- *Háló mérete*: `nano` - 1.9 millió paraméterrel, `small` - 7.2 millió paraméterrel és `medium` - 21.2 millió paraméterrel.

- *Augmentáció típusa*: A YOLOv5 szerzői által megadott alapértelmezett augmentációs értékekkel végzett tanítások vagy megnövelt augmentációs paraméterekkel végzett kísérletek. Az alapértelmezett típus a következő augmentációkat tartalmazza: HSV színcsatornákon lévő értékek növelése rendre 1.5%-al, 70%-al és 40%-al[53], véletlenszerű 10%-os transzláció[54] és vertikális tengely szerinti tükrözés 50%-os eséllyel. A megnövelt augmentációs beállítás ugyancsak tartalmazza a HSV értékek változtatását, mindegyik értéket 100%-al növeli, a véletlenszerű 50%-os transzlációt és a vertikális tengely szerinti tükrözést 50%-os eséllyel. Emellett minden képre alkalmaz 25 és -25 fokos forgatást, képmozaik összeállítást, összekeverését (mix up) és képszegmensekből való kép összeállítást (segment copy-paste).

- *Objektumosztályok száma*: A felhasznált adathalmaz címkéi tartalmaznak sisak és fej osztályokat, vagy csak sisak osztályt. Utóbbi esetben a fej annotációk nincsenek figyelembe véve.

[53] A `hue` növekedik 1.5%-al, `saturation` 70%-al és `value` 40%-al.

[54] Például egy 100x100-as méretű kép elmozdul 10 pixellel jobbra. A kép jobb oldalán lévő 10 pixel szélességű sáv eltűnik, a bal oldalon megjelenik egy 10 pixel széles szürke sáv a jobbra csúszott pixelek helyére.

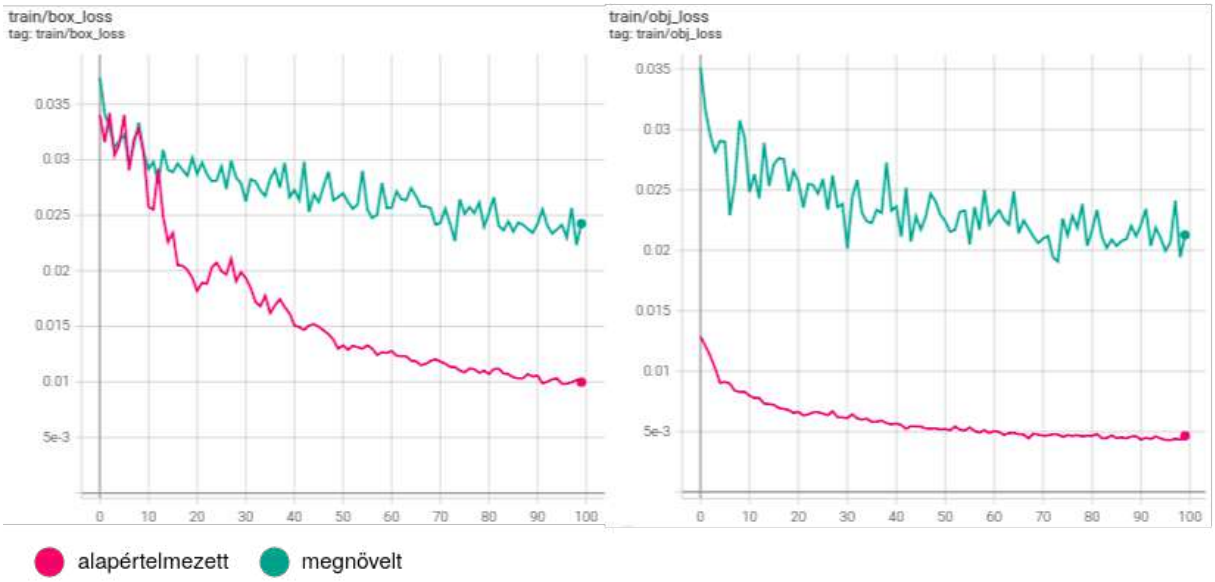



16. ábra. A költségfüggvény komponenseinek változása a sisak és fej osztályokra betanított hálók esetében, alapértelmezett augmentációs hiperparamétereket használva. Az első 40-50 epoch-ban gyorsan tanul a háló és eredményképpen gyorsan csökken a költségfüggvény értéke.

A kísérletek egy Nvidia GeForce GTX 1080-as videokártyával, Intel i5-6400 processzorral és 64 GB RAM-al ellátott számítógépen vannak elvégezve, amelyet a Codespring cég biztosít. A gyorsabb feldolgozás érdekében tanítás előtt az adathalmazok teljes terjedelmükben betöltődnek a RAM memóriába és nem a merevlemezről kerülnek beolvasásra.

A tanítások előre tanított súlyokból indulnak, amelyek 300 epoch-ot vannak tanítva az MS-COCO adathalmaz 80 osztályt tartalmazó 2017-es változatán [34]. A három háló méretre mindkét augmentációs beállítással le van futtatva egy újabb 300 epoch-os tanítás. A kísérletek el vannak végezve a betanítási adathalmazhalmaz sisakot és fejet tartalmazó és csak sisak osztályt tartalmazó változatára is, ez összesen 12 betanítást jelent. Ebben a szakaszban az adathalmazban lévő 416x416-os felbontású képek nincsenek újraméretezve. Az augmentációtól és az objektumosztályok számától függetlenül, az említett infrastruktúrán egy `nano`, `small`, és `medium` méretű neurális háló 300 epoch-os tanítása rendre 3, 6 és 14 órát vesz igénybe, úgy hogy a képek rendre 32-es, 64-es és 128-as méretű kötegekben (batch size) kerülnek feldolgozásra.

A finomhangolások az előző szakaszban betanított hálók súlyaiból indulnak, 100 epoch hosszúságúak és ugyanazt az augmentációt alkalmazzák, amelyekkel a tanítás történt, tehát egy




17. ábra. A költségfüggvény komponenseinek változása a sisak osztályokra finomhangolt `small` méretű háló továbbtanításakor, alapértelmezett és megnövelt augmentációs hiperparamétereket használva. Megnövelt augmentáció esetén az értékek nagyobbak és nagy mértékben változnak két epoch között. Ez a viselkedés minden kísérletnél jellemző megnövelt augmentációs paraméterek esetén, betanításnál és finomhangolásnál egyaránt. A klasszifikációhoz tartozó költségfüggvény komponens nincs feltüntetve, mivel egyetlen osztály esetén az értéke mindig nulla.

megnövelt augmentációval tanított háló finomhangolása megnövelt augmentációval folytatódik. A költségfüggvény viselkedése egy `small` méretű háló finomhangolási procedúrája közben a 17. ábrán látható. Ebben a szakaszban csak a sisak osztályra betanított hálókkal folytatódnak a kísérletek, mivel a finomhangolási adathalmaz csak sisakok annotációit tartalmazza. Ez esetben is mindhárom háló méret alapértelmezett és megnövelt augmentációs hiperparaméterekkel van továbbtanítva, amely összesen 6 további kísérletet jelent. Ebben a szakaszban a változó méretű képek 640x640-es felbontásra vannak átméretezve. A `nano`, `small`, és `medium` méretű hálók 100 epoch-os finomhangolása rendre 4, 8 és 17 percet vesz igénybe, 16-os kötegméretet használva.

Mindkét esetben Adam optimalizációs módszer van használva, $0.937$-es $\beta$ értékkel. A kezdetleges tanulási ráta $0.001$, amelyet OneCycle [60] eljárást alkalmazó tanulási ráta ütemző változtat. A súlycsökkenés (weight decay) értéke $0.0005$. Minden kísérlet 3 warmup epoch-al kezdődik, $0.8$-as kezdeti momentummal és $0.1$-es kezdeti bias tanulási rátával. A warmup epoch-ok és a tanulási ráta ütemező miatt a tanítások elején a hálók gyorsan tanulnak, mert kezdetben a ráta magas.

A hordozhatóság érdekében a tanítás egy Docker konténerben fut egy YAML formátumú

konfigurációs fájl alapján. Konténeren belül a teljes háló tanítási procedúra automatikusan történik a konfiguráció alapján, a folyamat elindításához elég egyetlen konfigurációs fájlt megszerkeszteni. Egy ilyen procedúra a következő lépésekből áll: annotációk átalakítása, adathalmaz felosztása tanítási, validálási és tesztelési halmazokra, háló tanítása, betanított háló tesztelése és statisztikák összeállítása, végül pedig a betanított háló és a statisztikák exportálása konténeren kívülre a felhasználó számára.

### A tanítás eredményei

A sisakdetektorok tesztelése során a precizitás, visszahívás (recall), mAP@0.5 és mAP@0.5:0.95 metrikák vannak megfigyelve. Először a betanítási szakaszban létrejött hálók tesztelésének statisztikái kerülnek bemutatásra, majd a finomhangolási szakaszban végzett kísérletek eredményei.

A sisak és fej objektumokat tartalmazó betanítási adathalmazon végzett 300 epoch-os tanítások eredménye 6 neurális háló, az ezekhez tartozó teszt adathalmazon elért eredmények a 2. táblázatban láthatóak. A megnövelt augmentációval tanított közepes méretű háló teljesít a legjobban. Minden háló esetében a precizitás magas és a visszahívásban, valamint a mAP értékekben nincs nagy eltérés. A megnövelt augmentáció átlagosan $0.22$-vel növelte meg a visszahívást, $0.105$-el a mAP@0.5-ot és $0.025$-el a mAP@0.5:0.95 értékét, viszont a precizitást $0.025$-el rontotta. Ebben a szakaszban tanított hálók jobban megtanulják felismerni a sisakokat, amelyből $4.22$-szer kevesebb van az adathalmazban, mint fejből. A kezdeti, 300 epoch hosszú tanításkor megfigyelhető hogy a nagyobb háló méretek jobban teljesítenek, viszont finomhangoláskor és csak sisak osztállyal végzett kísérletek során ezek a különbségek eltűnnek.

A 3. táblázat alapján megfigyelhető, hogy a sisak osztályra betanított hálóknál is az erősebb augmentáció javítja a teljesítményt. Ezek a hálók csak egyetlen osztályra kell specializálódjanak és elvárható, hogy jobban megtanulják azt az osztályt. Ez nem tükröződik a metrikákban, nincs számottevő hatása annak, hogy a hálók csak egyetlen osztályra kell figyeljenek. Átlagban $0.004$-el alacsonyabb a mAP@0.5, $0.012$-el magasabb a mAP@0.5:0.95, $0.001$-el nagyobb a precizitás és $0.008$-al kisebb a visszahívás a két osztályon tanult hálókhoz képest.

A második szakaszban elvégzett kísérletek alapján kijelenthető, hogy a hálók finomhangolása nagyban javítja a teljesítményt. A 4. táblázat tartalmazza a betanítási halmaz sisak osztályára 300 epoch-ot tanított hálók teljesítményét a finomhangolási teszt adathalmazon, a 5. táblázatban vannak feltüntetve a 100 epoch-ot finomhangolt hálók teljesítménye a

| Háló méret | Augmentáció | Osztály | Precizitás | Recall | mAP @0.5 | mAP @0.5:0.95 |
|---|---|---|---|---|---|---|
| medium | alapértelmezett | sisak és fej | **0.975** | 0.513 | 0.747 | 0.504 |
| | | sisak | 0.973 | 0.754 | 0.870 | 0.619 |
| | | fej | 0.977 | 0.272 | 0.624 | 0.390 |
| | megnövelt | sisak és fej | 0.954 | **0.792** | **0.884** | **0.553** |
| | | sisak | 0.954 | 0.888 | 0.937 | 0.640 |
| | | fej | 0.954 | 0.696 | 0.831 | 0.467 |
| small | alapértelmezett | sisak és fej | 0.967 | 0.530 | 0.753 | 0.502 |
| | | sisak | 0.961 | 0.774 | 0.877 | 0.618 |
| | | fej | 0.973 | 0.286 | 0.629 | 0.386 |
| | megnövelt | sisak és fej | 0.951 | 0.747 | 0.860 | 0.530 |
| | | sisak | 0.957 | 0.859 | 0.922 | 0.617 |
| | | fej | 0.945 | 0.635 | 0.797 | 0.442 |
| nano | alapértelmezett | sisak és fej | 0.970 | 0.514 | 0.746 | 0.495 |
| | | sisak | 0.972 | 0.725 | 0.855 | 0.607 |
| | | fej | 0.969 | 0.303 | 0.636 | 0.384 |
| | megnövelt | sisak és fej | 0.930 | 0.678 | 0.818 | 0.494 |
| | | sisak | 0.940 | 0.828 | 0.903 | 0.598 |
| | | fej | 0.920 | 0.528 | 0.732 | 0.390 |

2. táblázat. A sisak és fej osztályokra betanított hálók teljesítménye a tanítási adatok teszt halmazára. Mindkét osztály figyelembe vételével mért értékek közül az egyes metrika szerinti legnagyobb értékek ki vannak emelve.

| Háló méret | Augmentáció | Precizitás | Visszahívás | mAP@0.5 | mAP@0.5:0.95 |
|---|---|---|---|---|---|
| `medium` | alapértelmezett | 0.956 | 0.677 | 0.824 | 0.564 |
| | megnövelt | 0.964 | **0.904** | **0.946** | **0.677** |
| `small` | alapértelmezett | 0.977 | 0.685 | 0.834 | 0.595 |
| | megnövelt | 0.959 | 0.884 | 0.935 | 0.663 |
| `nano` | alapértelmezett | **0.982** | 0.773 | 0.882 | 0.638 |
| | megnövelt | 0.928 | 0.855 | 0.915 | 0.638 |

3. táblázat. A sisak osztályra betanított hálók teljesítménye a tanítási adatok teszt halmazára.

finomhangolási teszt adathalmazon. A tesztek alapján a finomhangolási teszt adathalmazon a finomhangolt hálók visszahívása átlagban $0.154$-el, a mAP@0.5 $0.08$-al és a mAP@0.5:0.95 $0.322$-vel növekedik meg a betanítási halmaz sisak osztályával tanított hálókhoz képest. A precizitás átlagosan csak $0.003$-al nő. Ez annak tudható be, hogy az értékek már tanítás után magasak, a finomhangolás nem tudja még fennebb tolni őket. Enyhén itt is jelen van az erősebb augmentáció pozitív hatása, kivéve a mAP@0.5:0.95 esetében, amely átlagban $0.018$-al alacsonyabb. Összesítve a precizitás, visszahívás és mAP@0.5 rendre 0.016-al, 0.071-el és 0.036-al növekedett.

| Háló méret | Augmentáció | Precizitás | Visszahívás | mAP@0.5 | mAP@0.5:0.95 |
|---|---|---|---|---|---|
| `medium` | alapértelmezett | 0.952 | 0.638 | 0.807 | 0.460 |
| | megnövelt | 0.976 | **0.872** | **0.933** | 0.476 |
| `small` | alapértelmezett | 0.971 | 0.713 | 0.850 | **0.478** |
| | megnövelt | 0.975 | 0.819 | 0.905 | 0.444 |
| `nano` | alapértelmezett | 0.984 | 0.660 | 0.827 | 0.430 |
| | megnövelt | **0.987** | 0.819 | 0.908 | 0.469 |

4. táblázat. A *betanítási* halmaz sisak osztályára betanított hálók teljesítménye a *finomhangolási adatok* teszt halmazán.

| Háló méret | Augmentáció | Precizitás | Visszahívás | mAP@0.5 | mAP@0.5:0.95 |
|---|---|---|---|---|---|
| `medium` | alapértelmezett | 0.987 | 0.840 | 0.920 | 0.791 |
| | megnövelt | **1.000** | **0.968** | **0.984** | 0.790 |
| `small` | alapértelmezett | 0.955 | 0.894 | 0.944 | **0.801** |
| | megnövelt | 0.989 | 0.936 | 0.967 | 0.769 |
| `nano` | alapértelmezett | 0.965 | 0.883 | 0.937 | 0.780 |
| | megnövelt | 0.967 | 0.926 | 0.959 | 0.759 |

5. táblázat. A sisak osztályra *finomhangolt* hálók teljesítménye a *finomhangolási adatok* teszt halmazán.

### Sisakkövetés

A rendszer csak olyan képhez tud számot társítani, amelyen a sisakon lévő szám látszik. Akkor is megállapítható a szám, amikor a sisak olyan szögben áll, hogy egyáltalán nem látszik rajta a szám. Ha a rendszer ilyen esetben követni tudja egyik képről a másikra a sisak pozícióját, akkor eldöntheti, hogy az aktuális képen egy korábban felismert, szám alapján beazonosított sisak van-e vagy sem. Ez alapján a képhez társíthatja a korábban kiolvasott számot. A sisak helyzetét is meghatározhatja, amely hasznos megkülönböztető információ lehet, ha a képen több sisakot is be kell azonosítani. Ez a beazonosítás addig működhet, amíg a sisak a kamera látóterében van. A felmerülő probléma a sisak pozíciójának képről képre való követése, amelyre a lehetséges megoldások közé tartoznak az alábbiak:

1. *MOSSE*[55] [9]: Objektumkövetés adaptív korrelációs szűrő segítségével. Első lépésben manuálisan meg kell határozni a képen lévő követendő objektumot tartalmazó téglalapot (ablak), amely alapján a szűrő megtanulja az objektum kinézetét. Az algoritmus lépésenként mozgatja az ablakot a szűrő előző, $t-1$ lépésbeli és aktuális $t$ kimenete alapján.

[55] Minimum Output Sum of Squared Error

2. *Meanshift és CAMshift*[56] [11]: Objektumkövetés képkockáról képkockára klaszter középpont számolással és centroid eltolással. A klaszter egy ablak alatti pixel értékek összességét jelenti. Minden lépés elején az ablak az előző lépésben meghatározott centroid alapján elmozdul, majd újraszámolódik. Első iterációban az ablakot a követendő objektumra kell helyezni.

3. *Kálmán-szűrő* [33]: Objektumok pozíciójának optimális becslése folyamatos, képkockák közötti mérések alapján.

4. *ROLO*[57] [44]: Objektumkövetés YOLO és LSTM hálók kombinációjának felhasználásával.

5. *Optikai folyam (optical flow)* [7]: Mozgás érzékelése egy videó egymás utáni képkockái között pixel szinten, a pixelek értékváltozásának modellezésével.

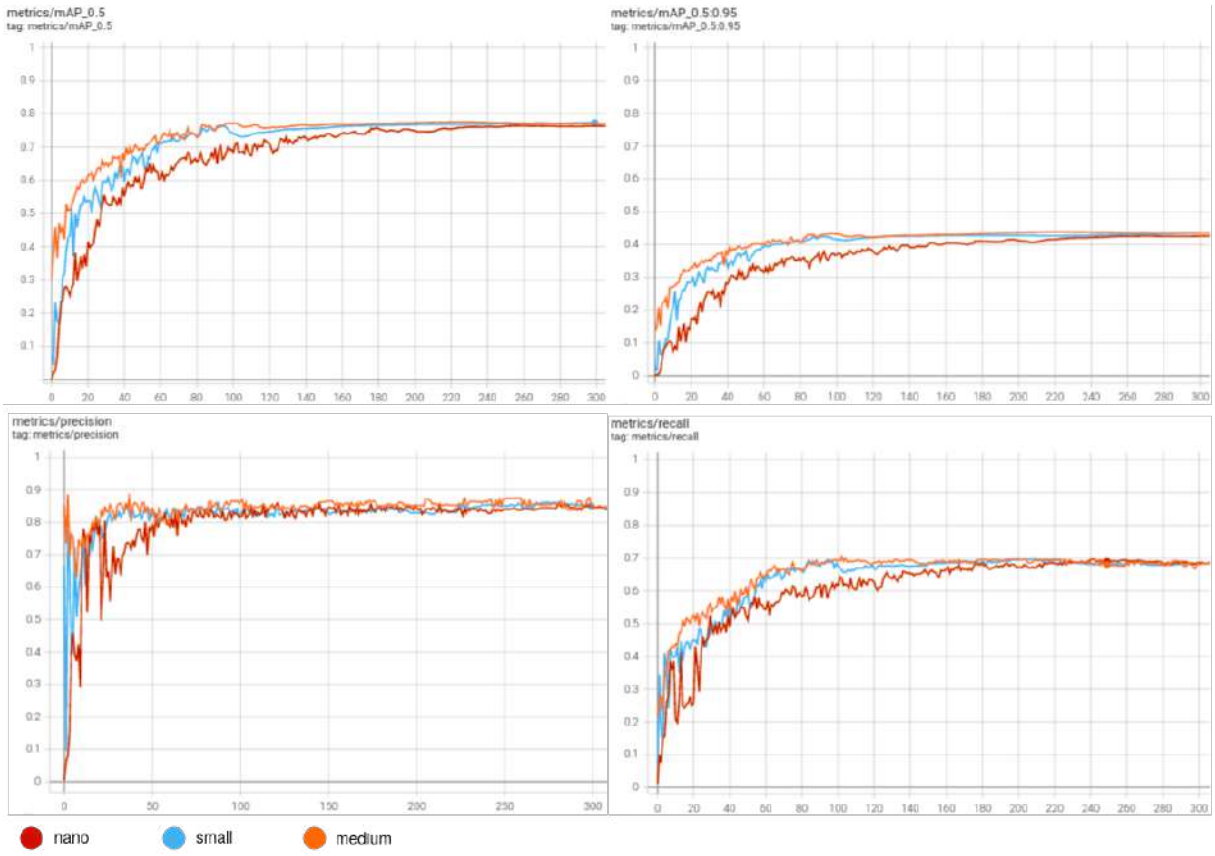
A felsorolt lehetőségek közül a MOSSE van tesztelve két forgatókönyv szerint: először egy 20 FPS-el felvett teszt videón követ egy sisakot, majd a kamerával 0.18 másodpercenként készült képsorozaton, amely megközelítőleg egy 5 FPS-el felvett videónak felel meg. Első esetben képes követni egy sisakot egészen addig, amíg a sisak nem kerül túl közel a kamerához és a mérete nem nagyobb a szűrő ablakánál. Második esetben inicializáció után csak néhány képkockán keresztül képes meghatározni a sisak pozícióját, mivel túl nagy eltérés van két egymás utáni kép között. Ez esetben a szűrő alól teljesen kicsúszik a sisak, ezért az algoritmus nem tudja korrigálni az ablak helyzetét. A fényképkészítő által rögzített képek 0,18 másodpercnél nagyobb időközökkel érkeznek, ezért élesben feltehetően még rosszabbul teljesítene a módszer. A MOSSE nem működik alacsony FPS mellett, ezért a rendszer esetében nem alkalmazható.

**Következtetések**

Az elvégzett kísérletek alapján sikerült különböző YOLOv5 architektúrájú hálókat betanítani, a legjobban teljesítő modell esetében a mAP:0.5 értéke elérte a $0.984$-et. Ez a `medium` méretű háló be van építve a sisakdetektáló mikroszolgáltatásba.

Egyetlen objektumosztályra való specializációjuknak nincs számottevő hatása a felismerésére és finomhangolással tovább növelhető a vizsgált modellek teljesítménye.

[56] Continuously Adaptive Meanshift
[57] Recurrent YOLO




18. ábra. A metrikák változása 300 epoch-os betanítás közben, sisakot és fej tartalmazó adathalmazon, alapértelmezett augmentációs hiperparaméterekkel. A mérőszámok minden epoch végén, a validációs halmazon vannak kiszámolva.

Felépítésükből adódóan más detektorokhoz képest gyorsabbak, ezért alkalmasak könnyen skálázható rendszerek építésére.

A 18. ábrán látható, hogy a betanítás során a hálók az első 20-30 epoch-ban gyors ütemben tanulnak. 100-150 epoch után a súlyok elérik az optimális értékeiket, ezután a hálók teljesítménye nem javul. Ez a viselkedés minden betanítási kísérletre jellemző. Erős augmentációkat alkalmazva lehet elérni a legjobb háló tanításokat.

## 3.4. Számfelismerés

A projekt egyik legfontosabb feladata a fényképek helyes kiosztása a résztvevők között. Ezt a sisakon található egyedi számok és látogatási időpont segítségével lehet megoldani. A legnagyobb probléma viszont az, hogy a pályákon részt vevő személyek nem a kamerára, hanem az akadályokra figyelnek, ezért a képeken sokszor nem ideális pózban jelennek meg. Ennek következménye az, hogy a számok különböző pozícióban, vagy egyáltalán nem jelennek meg. Ebben az alfejezetben a számfelismerő szolgáltatás 2.5 karakter felismerés része kerül

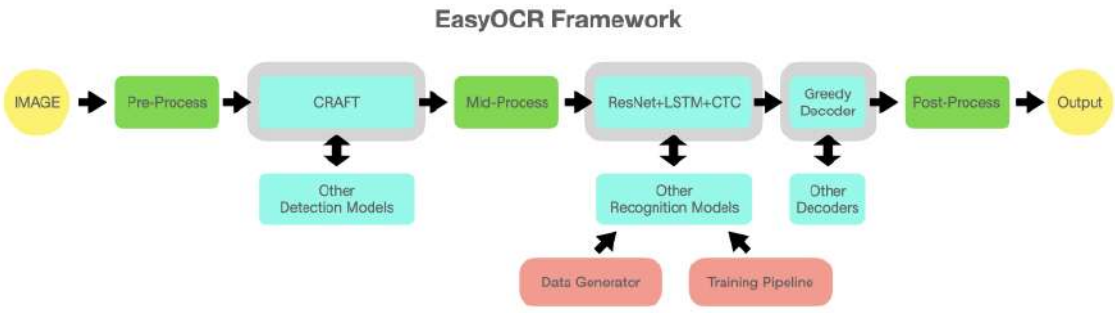
bemutatásra.

### Optikai karakterfelismerés

Az Optikai karakterfelismerés (OCR) [39] egy olyan módszer, amely a kézzel írt, vagy nyomtatott szövegek átalakítására szolgál. Olyan formára hozza, ami elektronikusan szerkeszthető és kereshető. Leginkább szkennelt dokumentumok, képek digitalizálására használják, a gépi látás és mesterséges intelligencia kutatási területe.

Az EasyOCR (19. ábra[58]) egy olyan nyílt forráskódú karakter felismerő könyvtár, ami több, mint 80 nyelvet és több elterjedt betűtípust támogat (latin, arab, cirill, stb.). A karakterek felismeréséhez a kép egy összetett folyamaton megy keresztül. Előfeldolgozásnak a képet normalizálja és átméretezi, hogy a szélessége és a magassága pixelekben számolva osztható legyen 32-vel. A következő lépes a karakter detektálás, ami a *CRAFT* [6][59] algoritmussal történik. Ez olyan területeket határoz meg, ahol a karakterek elhelyezkednek a képeken. A karakterek felismerésére *CRNN*[60] modellt használ, ami rész 3 komponensből áll össze: feature extraction (tulajdonság megállapítás) *ResNet* [63] segítségével, szekvencia címkézés *LSTM* [27] hálóval, illetve *CTC* [20] dekódolás. A modellek felépítése a PyTorch keretrendszerre alapszik.

A szolgáltatás egy előre betanított hálót használ a számok felismerésére. Ennek betöltéséhez szükséges megadni, hogy milyen típusú karaktereket kell felismerjen, majd a képeket átadja a `readText` nevű függvénynek, ami az eredményt téríti vissza. A képeken több helyen is felismerhet szöveget, ezért egy tömböt térít vissza benne a predikciókkal, és a hozzájuk tartozó konfidenciával és koordinátákkal.




19. ábra. Az EasyOCR architetkúrájának szemléltetése. Itt látható a képek kiértékelésének folyamata.

[58]Forrás: `https://github.com/JaidedAI/EasyOCR`, utolsó megtekintés dátuma: 2022-04-26
[59]Character-Region Awareness For Text detection
[60]Convolutional Recurrent Neural Network

Az EasyOCR előnye, hogy meg lehet adni paraméterként azt, hogy az adott képen milyen karaktereket ismerjen fel a háló. Az alkalmazásban csak a számjegyekre van testre szabva a háló karakter halmaza, de ez bővíthető a sisakokon lévő írásoknak megfelelően.

A felhasznált háló egy 30 képből álló képhalmazon van tesztelve. A képek kivágott sisakokat tartalmaznak, amin számok találhatóak. Ezek az Adrenaline Kalandpark weboldaláról, illetve Google képekből tevődnek össze, átlagméretük 399.3*433.9 pixel.

### Képek elforgatása

A számok pozíciója nagy mértékben befolyásolja az EasyOCR hálójának predikcióját. Az említett adathalmaz tesztelése során kiderült, hogy nagyobb pontosságot ér el, ha a számok vízszintes pozícióba vannak helyezve egy képen. Mivel a képeken nem tudni, hogy pontosan az illető személy milyen szöggel van elforgatva a kamerához képest, ezért a rendszerbe egy általános forgatás van bevezetve. A képek sorra 0, 30 és -30 fokkal elforgatva értékelődnek ki. Ha meghaladja az egyik predikció konfidencia szintje a küszöb értéket valamelyik kiértékelésnél, akkor már nem folytatódik a kiértékelés a többi változatra. Ez a módszer segít a teszt adathalmazon 85%-os pontosság helyett 93.3%-ot elérni.

A képek elforgatása egy általános, de nem biztos megoldás a számok vízszintre hozására. A kalandpark teljesítése során lehet, hogy a személy nagyobb szöggel el van fordulva a kamerához képest, ilyenkor már nem segít ez a megoldás. A forgatások esetén megtörténhet, hogy a számok állása meghaladja a 90 fokot a vízszinthez képest, így rosszul ismerhet fel számokat a rendszer, különösebb gond felmerülhet a 6-os illetve 9-es esetén. Ezáltal az általános forgatás, amit használ a rendszer, sem túl nagy.

### Eredmények

A fent említett adathalmaz teszteléséből a következő metrikák vannak mérve: pontosság, precizitás és visszahívás (recall). Több fázisban lettek kiszámítva ezek: átméretezés nélkül, átméretezéssel, illetve átméretezéssel és általános forgatással. Az átméretezés az átlag 399.3*433.9 pixelből 749.23*808.4 pixel van megtartva a magasság szélesség arányát a képeknek. Mindegyik tesztnél a felismerési küszöb érték 0.7, ha ezt meghaladja a predikció konfidencia értéke, akkor elfogadott adott predikció.

Átméretezés nélkül az EasyOCR által használt LSTM háló 73%-os pontosságot, 88%-os precizitást és 81%-os visszahívást ér el. Míg átméretezéssel 77%-os pontosságot, 85%-os

precizitást és 88%-os felismerést ér el. A legjobb eredményt a 3 különböző szöggel vett elforgatással adott: 93%-os pontosság, 93%-os felismerés, 100%-os visszahívás.

**Következtetések**

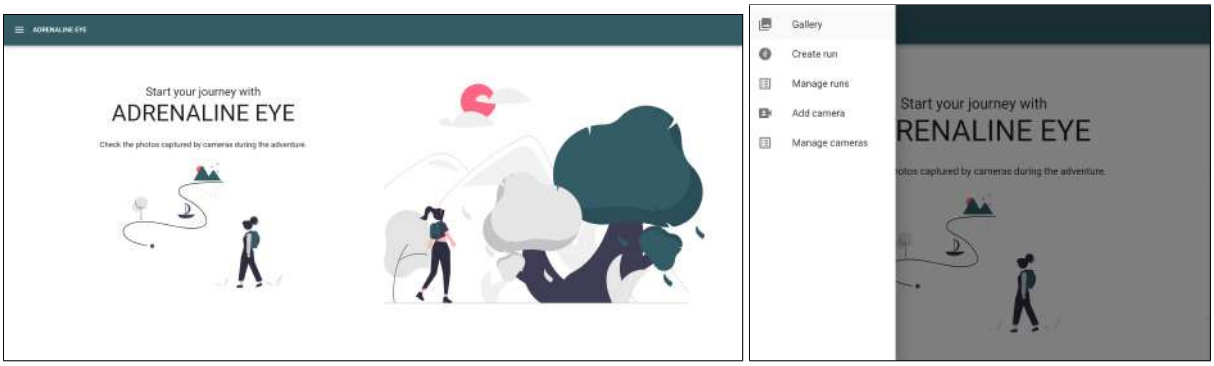
Az EasyOCR LSTM hálója legjobb eredményeként 93%-os pontosságot, 93%-os felismerést, 100%-os visszahívást ér el az általános 0, 30 és -30 fokos forgatások segítségével. Ezen módszer van felhasználva a számfelismerő szolgáltatásban.

# 4. Az alkalmazás működése

Az alkalmazás indításakor a felhasználót egy kezdőoldal fogadja, amely látható a 20a-s ábrán. Ahogy a 20b-s ábrán látszik, a weboldal rendelkezik egy navigációs sávval, amely a baloldalon lévő ikon segítségével érhető el.

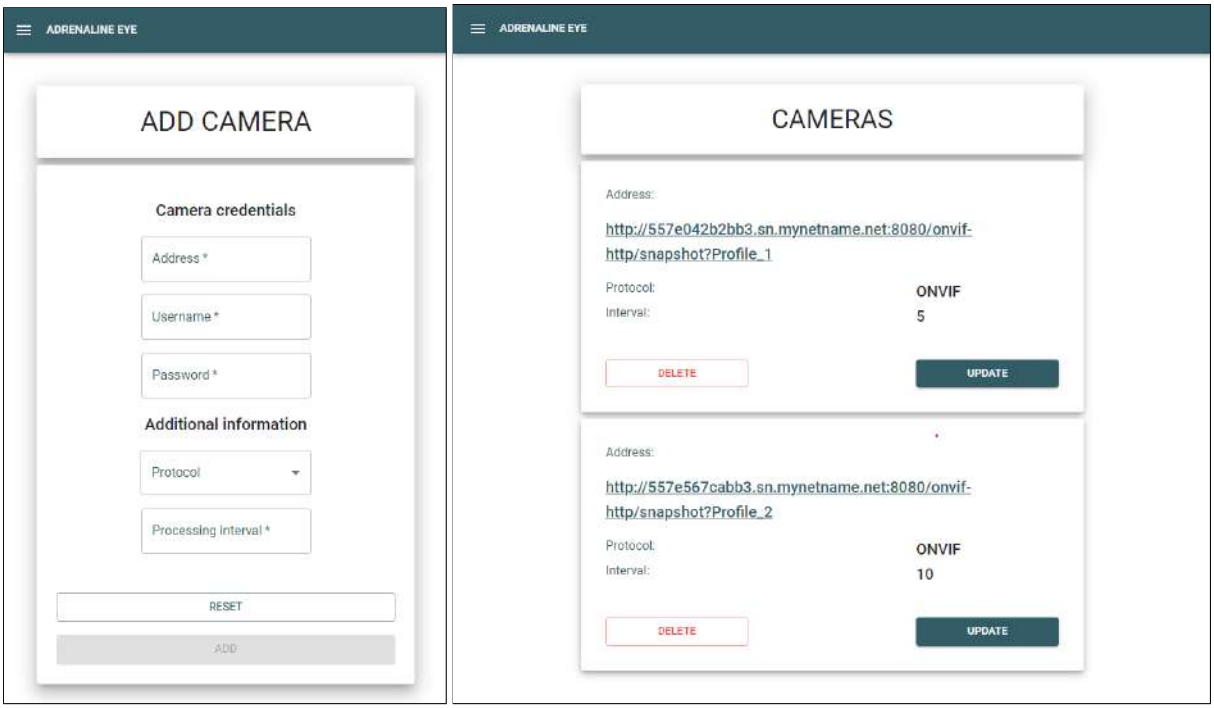
A webalkalmazás első funkcionalitása a kamerák menedzselése. Ehhez a funkcionalitáshoz több oldal is kapcsolódik, az egyik a kamerák hozzáadása, a másik a kamerák listázása. Kamerák hozzáadásához egy űrlapot kell kitölteni, amely a 21a. ábrán látszik. Egy kamera létrehozásához meg kell adni annak az elérési adatait, és az azonosításhoz szükséges adatokat, amelyek ebben az esetben felhasználónév és a jelszó. További információként meg kell adni, hogy a kamerát milyen protokoll alapján lehet elérni, illetve szükséges információ az, hogy hány másodpercenként készítsen képet a pályáról. Ezek kitöltése után az *Add* gomb segítségével elküldi az információkat az alkalmazás a kameravezérlő szolgáltatásnak. A feltöltést követően a felhasználó visszajelzést kap a webalkalmazástól, egy üzenet formájában. A visszajelzés *Camera added* üzenet lesz amennyiben a feltöltés sikeres, ellenkező esetben pedig egy hibaüzenet. Egy sikeres kamera létrehozást követően elkezdődik a kamerától kapott képek feldolgozása a rendszer többi része által. Az image capturer lementi a képeket a temporális tárolóra, és küld üzenetet a kép információiról a sisak detektálónak. Sikeres sisak és számfelismerés esetén a központi szerver áthelyezi a képet a permanens tárolóra, hozzátársítva a megfelelő runt.

A navigációs sávon a *Manage cameras* gomb megnyomásával a rendszer átirányítja a felhasználót egy olyan oldalra, ahol a kamerák listázva vannak. Ezen az oldalon megtalálhatóak a szükséges információk a kamerákról, amiket az *Update* gomb megnyomásával




(a) Kezdőoldal (b) Navigációs sáv

20. ábra. Weboldal betöltésekor megjelenő kezdőoldal

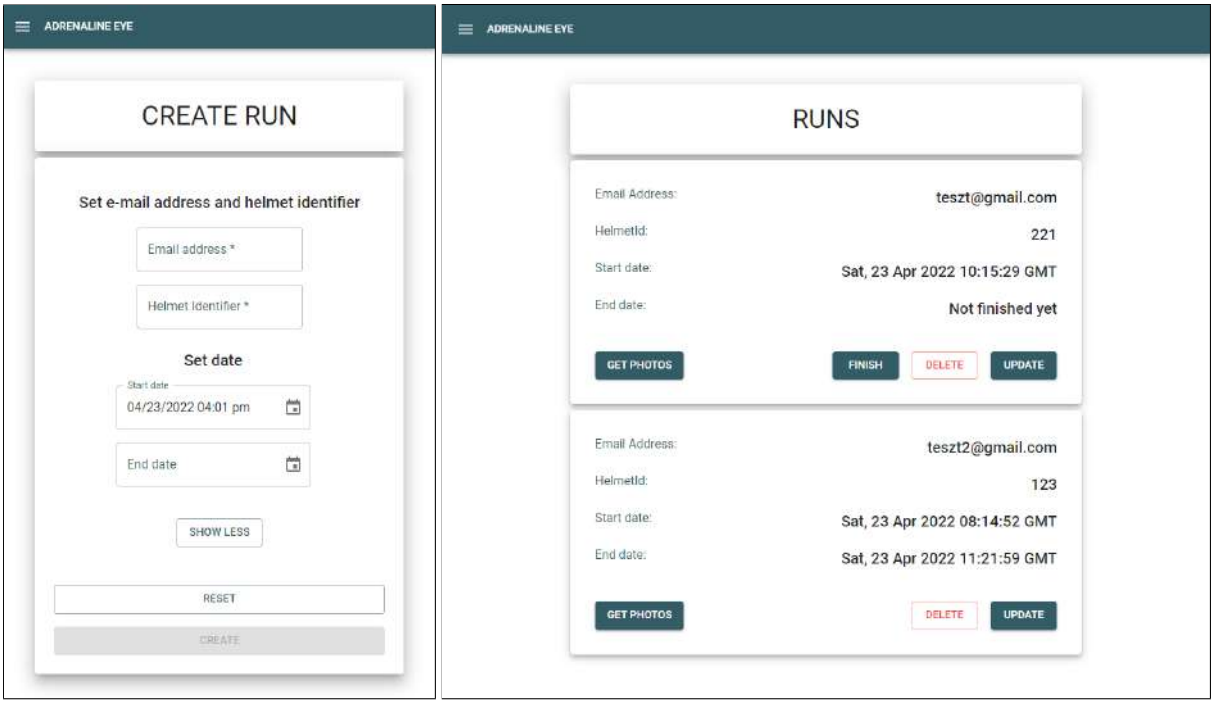



(a) Kamera hozzáadásának űrlapja (b) Kamerák listázása

21. ábra. Kamerák menedzselésére szolgáló oldalak

az előző űrlaphoz (lásd 21) hasonló módon szerkeszteni lehet, ugyanakkor a *Delete* gomb megnyomásával lehetőség van a kamerák törlésére is.

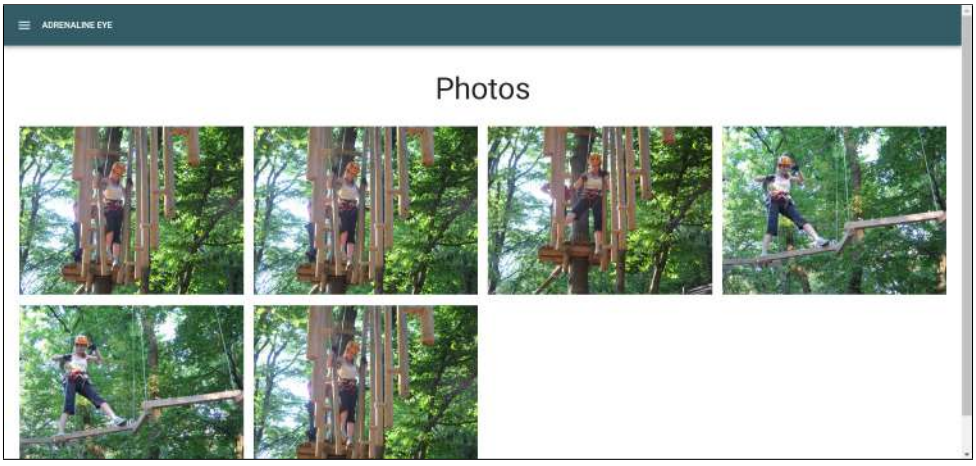
A következő fontos funkcionalitás a kalandpálya részvételek, azaz runok, felvezetése a rendszerbe. A *Create run* menüpont átvisz egy másik űrlapra, ami egy run létrehozására alkalmas. Az oldalon meg kell adni a résztvevő e-mail címét és sisakszámát. Az űrlap alapból beállítja a run kezdődátumát az oldal betöltésének időpontjára, a befejezési időpontot pedig üresen hagyja. Ezeket viszont meg lehet változtatni a *Show more* gomb megnyomása után, illetve a befejezési időpontot a kalandpálya teljesítése után is meg lehet adni. A *Create* gomb megnyomásakor a weboldal elküldi az információkat a központi szervernek, és visszajelzést kap a folyamat sikerességéről. A run bevezetésekor a központi szerver elmenti az információkat, és elkezd a számfelismerőtől kapott üzenetek alapján hozzátársítani képeket. Ekkor a szerver a résztvevő e-mail címére elküld egy QR kódot, ami egy elérési útvonal a galériához, ami a run-hoz csatolt képeket tartalmazza. A *Manage runs* oldalon megtekinthetőek a már létrehozott run-ok, amelyeket módosítani és törölni is lehet ugyanitt, a megfelelő gombok segítségével. A *Finish run* gomb segítségével lezárhatóak a még folyamatban lévő runok, aminek a hatására a jelenlegi idő lesz elküldve a szervernek mentésre. Ezen felül a runokhoz tartozó galériákat is el lehet érni innen a *Get photos* gomb segítségével.




(a) Run hozzáadásának űrlapja (b) Runok listázása

22. ábra. Runok menedzselésére szolgáló oldalak

A webes alkalmazás funkcionalitásaihoz tartozik még a galéria megtekintése. Ezt el lehet érni az előbb említett run listázási oldalról a *Get photos* gomb megnyomásával, vagy rá is lehet keresni a run egyedi azonosítója alapján.




23. ábra. Képek megjelenítése a galériában

# 5. Eszközök és munkamódszerek

A fejezet bemutatja a projekt fejlesztése alatt alkalmazott munkamódszereket és felhasznált fontosabb eszközöket.

### Agilis projektmenedzsment és szoftverfejlesztés

Az agilis projektmenedzsment [31] egy irányított folyamat, amely egy termék vagy szolgáltatás értékesítését jelenti az üzleti világban. Magába foglalja a létrehozást, folyamatos fejlesztést, eszközök és technikák használatát, valamint a tervezői csapat hozzájárulását egy helyesen működő termékhez, ami megfelel az ügyfél igényeinek.

A szoftverfejlesztés terén az agilis megközelítés egy empirikus módszer, mely a projektben feltűnő konkrétumokon alapszik. Gyakorlatban a csapat összes tagja követi és hozzásegít a folyamathoz. A fejlesztés során rendszerességgel értékelik és tesztelik a szolgáltatást, valamint hirtelen változtatások esetén alkalmazkodnak a helyzethez, hogy minimalizálják a később feltűnő problémákat.

### Scrum keretrendszer

Az Adrenaline Eye projekt fejlesztésén belül a *Scrum* keretrendszert [30] van használva, mely egyike az agilis munkamódszereknek. Úgynevezett sprint iterációkon alapszik és a Scrum csapatokhoz rendelt szerepekből, munkaanyagokból és eseményekből tevődik össze.

Egy *Scrum* csapat a következő főbb szerepkörökből áll:

- A *product owner* (terméktulajdonos) a megrendelőt képviseli és a projekt üzleti szempont béli igényeit fogalmazza meg.
- A *development team* (fejlesztő csapat) tagjai a napi munkát elkötelezetten végzik és a projekten belül több szakterületen is van készségük.
- A *scrum master* (scrum mester) pedig a csapat támogatásáért, megszervezéséért és a szabályok betartásáért felelős, valamint emellett célja a munkakörnyezet folyamatos fejlesztése.

A *Scrum* által előírt munkaanyagok:

- A *product backlog* (követelménylista) a teljes követelmény lista, amely meghatározza magát a terméket, a funkciói leírását és a hozzá tartozó ötleteket. A terméktulajdonos szerkeszti a tartalmát és dönti el a lista elemeinek megfelelő sorrendjét.

- A *sprint backlog* (teendőlista) meghatározza az iterációba helyezett taszkokat és a product backlog-al ellentétben ezt csak a fejlesztő csapat változtathatja.

- A *product increment* (növekmény) egy potenciálisan használható funkcionalitás, amely teljesen működőképes és késznek nyilvánítható.

A *Scrumon* belüli események:

- A *sprint* egy iteráció, amely bekebelezi az összes többi eseményt. Az iterációk rövid ciklusok, egy- vagy kéthetesek, nem többek 1 hónapnál és visszajelzést nyújtanak a csapatnak, hogy folyamatosan fejlődő képes maradjon a munkamenet.

- A *sprint planning* (tervezés) egy megbeszélés a sprint elején, mely során a csapat megtervezi a teendőlistát.

- A *daily standup* (napi megbeszélés) naponta megejtett 15 percnél nem hosszabb gyűlés, amelyiken a csapat tagjai elmondják külön-külön, hogy az előző alkalom óta mit sikerült megvalósítani, mit fog a következőkbe tenni és ha vannak akkor megemlíti a lehetséges akadályokat.

- *Sprint review*-al (áttekintés) zárul egy iteráció, mely révén a csapat megbeszéli az eddig elkészült funkcionalitásokat és bemutatja azokat a terméktulajdonosnak. A megbeszélés hossza függ a sprint terjedelmétől.

- *Sprint retrospectivet* (visszatekintés) úgyszintén az iteráció végén tartják. A csapat tagjai megbeszélik egymás között, hogy az elmúlt sprintbe mi ment jól, mi ment rosszul és hogy mit kellene változtatni, hogy a jövőben javuljon a munkamenet.

A fentebb említett munkamódszereknek egy sajátos verziója a projekt fejlesztése közben is alkalmazva volt. A mentorok a scrum master és product owner szerepét töltötték be a fejlesztő csapat számára. A projekt fejlesztésnek a folyamata két nagyobb szakaszra osztható, a nyári gyakorlatra és az azt követő időszakra. A nyári gyakorlat során a scrum a következőképpen zajlott: a munka kéthetes sprintekre volt osztva, a sprintek planningel kezdődtek, majd naponta egy standup keretén belül megbeszélés történt a felmerülő problémák megoldása érdekében, a

sprintet pedig egy review és retrospective megbeszélés zárta. A fejlesztés második szakaszában egy sprint hossza egyhónapos volt, heti egyeztetésekkel. A sprint elején és végén a első időszakhoz hasonlóan lettek megtartva a sprint események.

### Verziókövetés, Folyamatos integráció

Az Adrenaline Eye projekt mikroszolgáltatásait külön *GitLab* [5] repositorykban vannak tárolva *Git* [37] verziókövetést használva.

A *Git* egy decentralizált verziókövető, irányítja és követi a fájlokon való változtatásokat egy megadott perióduson keresztül. Lehetővé teszi a csapat számára, hogy együtt fejlesszenek anélkül, hogy negatívan kihassanak egymás munkájára.

A *GitLab* egy DevOps platform, melyen belül létrehozott *repositoryk* tárhelyként szolgálnak a *Git* számára, így nem csak a jelen variánsát mentik le a projekt fájlrendszerének, hanem visszamenőleg minden egyes változtatást, ami *commit*-olva és *push*-olva lett. Ezentúl egy *Wiki* repositoryval is rendelkezik, amely a projekt dokumentációját, specifikációit és diagramjait tárolja. Lehetőséget ad feladatok kitűzésére és rendszerezésére egy tábla formájában, mely elősegíti a *Scrum* munkamódszert is felvázolva vizuálisan a feladatok állapotát.

A *GitLab* rendelkezik egy CI/CD[61] eszközzel, mely a folyamatos integrációért és folyamatos kitelepítésért szolgál. Automatizálja a munkamenetet és biztosítja a jobb minőségű kódot iterálva a buildelést, tesztelést és kitelepítést. Segít elkerülni vagy egy korai fázisban kiküszöbölni a hibákat.

A *GitLab CI/CD* a következő elemekből épül fel:

- A *pipelineok* a legfelsőbb komponensei a folyamatos integrációnak, *jobok* kollekciója, amelyek *stagekbe* vannak csoportosítva. Általánosan egy automatikus folyamat, de szükség esetén manuálisan is indítható.
- A *Jobok* feladatokat határoznak meg, mint például kód tesztelése, lefordítása vagy kitelepítése.
- A *Stagek* megszabják a feladatok elvégzésének sorrendjét, csoportosítva azokat.
- A *Runnerek*, azok a komponensek, amelyek a feladatokat végzik.

A projekten belül fejlesztett szolgáltatások rendelkeznek saját GitLab repositoryval, build eszközökkel és csomagkezelőkkel, amelyek szintén hozzá segítenek a munkafolyamat

[61] Continuous Integration / Continuous Deployment

automatizálásához. A Java programozási nyelvben implementált szolgáltatások a *Gradlet*, Pythonban írt komponensek a *Pip*-et és Javascriptre alapuló modulok a *Yarnt* használják függőség kezelő és build eszközként. Az eszközök által elérhető funkcionalitások scriptként felhasználhatók az automatizálásban egy gitlab-ci.yml konfigurációs fájlon keresztül, melyet minden egyes repository tartalmaz. Az előzőleg említett jobok és stagek szintén ebben az állományban definiálhatóak.

### Kódelemzők

A kódelemzők, más szóval linterek, szerepe a tiszta kódolás elveinek és konvencióinak betartásának biztosítása. A projekt fejlesztése során használt linterek nagy mértékben konfigurálhatóak és beágyazhatóak a GitLab folyamatos integrációjába, így garantálva a termék minőségét és hibamentességét.

A felhasznált kódelemzők programozási nyelvekre felosztva a következők:

- Java:
  - A *Checkstyle* egy egységes kódolási szabvány biztosít.
  - A *PMD* a programkód futtatása nélkül megtalálja a gyakori programozási hibákat.
  - A *SpotBugs* hibaminták keresésén alapszik.
- Python:
  - A *PyLint* hibákat keres, javaslatokat tesz és kiértékelést nyújt a kódról.
  - A *Flake8* három másik kódelemzőt összegez, PyFlakes-t, a pycodestyle-t és a Ned Batchelder McCabe-szkriptjét, amelyek alapján a PEP8 kódolási stílust szabályozza, programozási hibákat keres és ellenőrzi a program ciklomatikus komplexitását.
  - A *MyPy* egy statikus típusellenőrző.
  - A *Bandit* a kódban gyakori biztonsági problémákat keres és jelentést tesz a folyamat végén.
- Typescript:

– A *ESlint* statikus kódelemző, ami képes automatikusan kijavítani a megtalált hibákat. Támogatja az *ECMAScript* jelenlegi szabványait és a kód típusosságát is ellenőrzi a TypeScript plugin-jai segítségével.

– A *Prettier* a formázásáért felelős, segítségével egységes stílus alkalmazódik a forráskódra, valamint rendelkezik egy beépíthető modullal is az ESLint statikus kódelemzőbe.

### Kitelepítés

A GitLab CI/CD folyamatához, de főként a kitelepítéshez nagy mértékben hozzájárult a *Docker* [43], ami lehetővé teszi a szoftver virtuális konténerbe helyezését és működéséhez szükséges minden függőséget megteremt, legyen szó könyvtárakról, konfigurációkról vagy egyéb eszközökről, így az könnyedén kitelepíthető lesz a végfelhasználó számára. Továbbá segít a dinamikus skálázhatóságba, hiszen a konténerek gyorsan beüzemeltethetők és kevés erőforrást használnak.

A *Docker* egy saját kifejező nyelvet is nyújt, melyen keresztül irányítható a szoftver konténerbe ágyazása. A nyelv párosítva az előző alfejezetben említett build eszközökkel és csomagkezelőkkel, deklaratívvá, megismételhetővé és főleg robusztussá teszi a kitelepítés folyamatát.

A *Docker* absztrakciót kínál és nem használ virtuális hardvert. A programok nagy része elkülönülten futhat egy pehelysúlyú, egyszerű, erőforrásokra optimalizált Linux disztribúción, mely gyorsítja a rendszerindítást és csökkenti a konténer helyfoglalását. Ilyen például az *Alpine Linux*.

A kitelepítés *build* szakaszában egy Dockerfile konfigurációs állomány alapján létrejön egy *Docker image*, mely tartalmazza a szoftverhez tartozó fájlokat és metaadatokat, majd az feltöltődik *deploy* szakasz során a megadott *container registrybe*, melyet szintén a *GitLab* szolgáltatja.

Az Adrenaline Eye szoftver lokális tesztelése a *Docker-compose* eszköz segítségével történik, amely képes megteremteni a környezetet több modulból álló alkalmazások futtatására. Egy YAML[62] fájlon keresztül megengedi a szolgáltatások *Docker image* általi definiálását, saját hálózati interfészek kezelését, illetve perzisztens volume-ok létrehozását. A hálózatok és volume-ok hozzájárulnak a komponensek közötti kapcsolatteremtéshez. A kitelepítéssel

[62] YAML Ain't Markup Language

kapcsolatos, Docker-compose által felhasznált konfigurációs fájlok egy külön repositoryban vannak. A konfigurációkban felhasznált Docker image-ek a projekt GitLab *container registryjéből* töltődnek le automatikusan. A Compose által biztosított automatizáció révén a teljes rendszer elindítható egyetlen `docker-compose up` paranccsal.

# Következtetések és továbbfejlesztési lehetőségek

A dolgozatban bemutatott Adrenaline Eye projekt főbb célkitűzései megvalósultak, létrejött egy alkalmazás, amely a felvezetett kamerák által készített képek alapján felismeri a sisakokon megjelenő számokat, és hozzátársítja időpont alapján egy adott runhoz.

A rendszer mikroszolgáltatás alapú architektúrával jött létre, melynél a kameravezérlő szerviz képes a kamerák menedzselésére és a fényképkészítési feladatok ütemezésére, a fényképkészítő szolgáltatás megvalósítja a képek készítését és mentését a megadott kamerákkal. A képfeldolgozást végző mikroszolgáltatások közül a sisakdetektáló képes fejeket és sisakokat felismerni és lokalizálni, illetve a számfelismerő a képeken lokalizált sisakokon képes számokat felismerni. A központi szerver megvalósítja a runok és beazonosított fotók menedzselését, tárolását, és ezek elérhetőségét a webalkalmazás számára.

A webes applikáció lehetővé teszi a kamerák és runok menedzselését, illetve az utóbbihoz társított képek megjelenítését galéria formában. Sikerült megvalósítani a szolgáltatások közti üzenet alapú kommunikációt RabbitMQ bróker segítségével, illetve ezek kitelepítését Docker segítségével.

A különböző mikroszolgáltatások tervezése és fejlesztése során felmerült több továbbfejlesztési lehetőség:

- Központi szervernél és webalkalmazásnál felhasználómenedzsment: adminnak és kalandpályán résztvevő személyeknek megfelelő szerepkörök elkülönítése, illetve különböző szerepkörökkel rendelkező felhasználók létrehozása. Az admin felhasználó menedzselheti a kamerákat és résztvevőket, a résztvevő pedig listázhatja a saját runjait, és megtekintheti az azokhoz tartozó képeket.

- Webalkalmazás esetén a runok szűrése és rendezése időpont alapján: külön lehetne választani azokat a runokat amik még nem fejeződtek be, hogy az adminok jobban tudják monitorizálni a jelenleg kalandparkban tartózkodó vendégeket.

- Mobil alkalmazás bevezetése admin felhasználóknak az rendszer könnyebb menedzselése érdekében.

- Számfelismerő szolgáltatás esetében élek felismerése a számok vízszintes helyzetbe hozásának érdekébe, vagy olyan OCR könyvtár alkalmazása, ami figyelembe veszi a karakterek elforgatási szögét.

- Sisakdetektáló szolgáltatás kiegészítése egy biztonsági funkcionalitással, amely sisak nélküli látogatók felismerésekor riasztja a parkban dolgozókat.
- Sisakkövetés felhasználása, amely révén a rendszer olyan képhez is tud látogatót társítani, amelyen nem látszik a sisakszám. Abból a feltételezésből kiindulva, hogy egy pályán egyszerre egy ember megy végig, akkor ha van két időpillanat amikor a sisakot sikerül beazonosítani, feltételezhethető, hogy az időintervallumban készült összes képen ugyanaz a sisak látható. A rendszer így egy run időszakán belül a megfelelő személyhez társítja a képeket.
- Az ONVIF protokollt támogató térfigyelő kamerák egy része képes egy eseményt kibocsájtani a protokoll *analytics* és *event* moduljainak segítségével mozgásérzékelés vagy személyfelismerés pillanatában. Ezeket az eseményeket eventbrókeren keresztül is küldheti, ha az konfigurálva van. Ez a megközelítés javíthat a fényképkészítés ütemezésének a kifinomultságán.

# Hivatkozások


[1] *A JavaScript library for building user interfaces*. URL: https://en.reactjs.org/ (elérés dátuma: 2022. ápr. 12.)

[2] Glenn Jocher et. al. *ultralytics/yolov5: v6.0 - YOLOv5n 'Nano' models, Roboflow integration, TensorFlow export, OpenCV DNN support*. v6.0. verzió. 2021. okt. DOI: 10.5281/zenodo.5563715. URL: https://doi.org/10.5281/zenodo.5563715.

[3] P.H. Ansari. *Learn MapStruct: Give You Enough Understanding on the Various Functionalities of Mapstruct with Suitable Examples*. Independently Published, 2021. URL: https://books.google.ro/books?id=uCN0zgEACAAJ.

[4] *Axios HTTP library*. URL: https://axios-http.com/docs/intro (elérés dátuma: 2022. ápr. 12.)

[5] J. van Baarsen. *GitLab Cookbook*. EBL-Schweitzer. Packt Publishing, 2014. ISBN: 9781783986859. URL: https://books.google.ro/books?id=ANIGBgAAQBAJ.

[6] Youngmin Baek és mások. „Character region awareness for text detection". *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, 9365–9374. oldal.

[7] John Barron, David Fleet és S. Beauchemin. „Performance Of Optical Flow Techniques". *International Journal of Computer Vision* 12 (1994. febr.), 43–77. oldal. DOI: 10.1007/BF01420984.

[8] Alexey Bochkovskiy, Chien-Yao Wang és Hong-Yuan Mark Liao. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. 2020. DOI: 10.48550/ARXIV.2004.10934. URL: https://arxiv.org/abs/2004.10934.

[9] David Bolme és mások. „Visual object tracking using adaptive correlation filters". 2010. jún., 2544–2550. oldal. DOI: 10.1109/CVPR.2010.5539960.

[10] G. Bradski. „The OpenCV Library". *Dr. Dobb's Journal of Software Tools* (2000).

[11] Gary R. Bradski. *Computer Vision Face Tracking For Use in a Perceptual User Interface*. 1998.

[12] R. Costa és D. Hodun. *Google Cloud Cookbook*. O'Reilly Media, 2021. ISBN: 9781492092865. URL: https://books.google.ro/books?id=oAtHEAAAQBAJ.

[13] Navneet Dalal és Bill Triggs. „Histograms of oriented gradients for human detection". *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*. 1. kötet. Ieee. 2005, 886–893. oldal.

[14] Jia Deng és mások. „Imagenet: A large-scale hierarchical image database”. *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, 248–255. oldal.

[15] *Face recognition based system for amusement parks*. URL: http://www.polingametechnologies.com/electronic-accessories/aipix.aspx (elérés dátuma: 2022. ápr. 23.)

[16] L. Ferrari és E. Pirozzi. *Learn PostgreSQL: Build and manage high-performance database solutions using PostgreSQL 12 and 13*. Packt Publishing, 2020. ISBN: 9781838986896. URL: https://books.google.ro/books?id=WXUCEAAAQBAJ.

[17] Roy T Fielding. *Architectural styles and the design of network-based software architectures*. University of California, Irvine Irvine, USA, 2000. 5. fejezet.

[18] *File-system Routing*. URL: https://nextjs.org/docs/routing/introduction (elérés dátuma: 2022. ápr. 12.)

[19] Ross Girshick és mások. *Rich feature hierarchies for accurate object detection and semantic segmentation*. 2013. DOI: 10.48550/ARXIV.1311.2524. URL: https://arxiv.org/abs/1311.2524.

[20] Alex Graves és mások. „Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks”. *Proceedings of the 23rd international conference on Machine learning*. 2006, 369–376. oldal.

[21] Felipe Gutierrez, Gutierrez és Anglin. *Pro Spring Boot 2*. Springer, 2019.

[22] Charles R. Harris és mások. „Array programming with NumPy”. *Nature* 585.7825 (2020. szept.), 357–362. oldal. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[23] Kaiming He és mások. „Deep residual learning for image recognition”. *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, 770–778. oldal.

[24] Sepp Hochreiter. „The vanishing gradient problem during learning recurrent neural nets and problem solutions”. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6.02 (1998), 107–116. oldal.

[25] Sepp Hochreiter és Jürgen Schmidhuber. „Long Short-Term Memory”. *Neural Computation* 9.8 (1997), 1735–1780. oldal.

[26] Kurt Hornik, Maxwell Stinchcombe és Halbert White. „Multilayer feedforward networks are universal approximators”. *Neural networks* 2.5 (1989), 359–366. oldal.

[27] Zhiheng Huang, Wei Xu és Kai Yu. „Bidirectional LSTM-CRF models for sequence tagging”. *arXiv preprint arXiv:1508.01991* (2015).

[28] *JavaScript with syntax for types*. URL: https://www.typescriptlang.org/ (elérés dátuma: 2022. ápr. 12.)

[29] Alex Krizhevsky, Ilya Sutskever és Geoffrey E Hinton. „Imagenet classification with deep convolutional neural networks”. *Advances in neural information processing systems* 25 (2012).

[30] M.C. Layton és D. Morrow. *Scrum For Dummies*. Wiley, 2018. ISBN: 9781119467649. URL: https://books.google.ro/books?id=KI9XDwAAQBAJ.

[31] M.C. Layton és S.J. Ostermiller. *Agile Project Management For Dummies*. Wiley, 2017. ISBN: 9781119405733. URL: https://books.google.ro/books?id=ZkozDwAAQBAJ.

[32] Y. Lecun és mások. „Gradient-based learning applied to document recognition”. *Proceedings of the IEEE* 86.11 (1998), 2278–2324. oldal. DOI: 10.1109/5.726791.

[33] Qiang Li és mások. „Kalman Filter and Its Application”. *2015 8th International Conference on Intelligent Networks and Intelligent Systems (ICINIS)*. 2015, 74–77. oldal. DOI: 10.1109/ICINIS.2015.35.

[34] Tsung-Yi Lin és mások. *Microsoft COCO: Common Objects in Context*. 2014. DOI: 10.48550/ARXIV.1405.0312. URL: https://arxiv.org/abs/1405.0312.

[35] Zachary C. Lipton, John Berkowitz és Charles Elkan. *A Critical Review of Recurrent Neural Networks for Sequence Learning*. 2015. DOI: 10.48550/ARXIV.1506.00019. URL: https://arxiv.org/abs/1506.00019.

[36] Wei Liu és mások. „Ssd: Single shot multibox detector”. *European conference on computer vision*. Springer. 2016, 21–37. oldal.

[37] J. Loeliger és M. McCullough. *Version Control with Git: Powerful Tools and Techniques for Collaborative Software Development*. Oreilly and Associate Series. O'Reilly Media, Incorporated, 2012. ISBN: 9781449316389. URL: https://books.google.ro/books?id=ZkXELyQWf4UC.

[38] Christopher Manning, Prabhakar Raghavan és Hinrich Schütze. „Introduction to information retrieval”. (2008), 145–147. oldal.

[39] Shunji Mori, Hirobumi Nishida és Hiromitsu Yamada. *Optical character recognition*. John Wiley & Sons, Inc., 1999.

[40] I. Nadareishvili és mások. *Microservice Architecture: Aligning Principles, Practices, and Culture*. O'Reilly Media, Incorporated, 2016. ISBN: 9781491956250. URL: https://books.google.ro/books?id=BvUEvgAACAAJ.

[41] *Next.js pages*. URL: https://nextjs.org/docs/basic-features/pages (elérés dátuma: 2022. ápr. 12.)

[42] Thao Nguyen és mások. „Object detection using scale invariant feature transform”. *Genetic and evolutionary computing*. Springer, 2014, 65–72. oldal.

[43] J. Nickoloff és S. Kuenzli. *Docker in Action, Second Edition*. Manning, 2019. ISBN: 9781617294761. URL: https://books.google.ro/books?id=bmVwwQEACAAJ.

[44] Guanghan Ning és mások. „Spatially Supervised Recurrent Convolutional Neural Networks for Visual Object Tracking”. *arXiv preprint arXiv:1607.05781* (2016).

[45] Travis E Oliphant. *A guide to NumPy*. 1. kötet. Trelgol Publishing USA, 2006.

[46] *Open source optical character recognition*. URL: https://github.com/JaidedAI/EasyOCRs (elérés dátuma: 2022. ápr. 13.)

[47] Rafael Padilla, Sergio Netto és Eduardo da Silva. „A Survey on Performance Metrics for Object-Detection Algorithms”. 2020. júl. DOI: 10.1109/IWSSIP48289.2020.

[48] Adam Paszke és mások. „PyTorch: An Imperative Style, High-Performance Deep Learning Library”. *Advances in Neural Information Processing Systems 32*. Szerkesztette H. Wallach és mások. Curran Associates, Inc., 2019, 8024–8035. oldal. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[49] Rachana Patel és Sanskruti Patel. „A Comprehensive Study of Applying Convolutional Neural Network for Computer Vision”. *International Journal of Advanced Science and Technology* 6 (2020. jan.), 2161–2174. oldal.

[50] *React Components*. URL: https://reactjs.org/docs/components-and-props.html (elérés dátuma: 2022. ápr. 12.)

[51] *React UI framework*. URL: https://mui.com/ (elérés dátuma: 2022. ápr. 12.)

[52] Joseph Redmon és mások. „You only look once: Unified, real-time object detection”. *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, 779–788. oldal.

[53] *RFID enabled automated photograpy*. URL: https://sniperactionphoto.com/ (elérés dátuma: 2022. ápr. 23.)

[54] G.M. Roy és J. Titcumb. *RabbitMQ in Depth*. Manning, 2017. ISBN: 9781638353225. URL: https://books.google.ro/books?id=1zkzEAAAQBAJ.

[55] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. 2016. DOI: 10.48550/ARXIV.1609.04747. URL: https://arxiv.org/abs/1609.04747.

[56] David E Rumelhart, Geoffrey E Hinton és Ronald J Williams. „Learning representations by back-propagating errors”. *nature* 323.6088 (1986), 533–536. oldal.

[57] *Safety Helmet Detection Dataset*. URL: https://www.kaggle.com/datasets/andrewmvd/hard-hat-detection (elérés dátuma: 2022. ápr. 21.)

[58] *Safety Helmet Wearing Dataset*. URL: https://github.com/njvisionpower/Safety-Helmet-Wearing-Dataset (elérés dátuma: 2022. ápr. 21.)

[59] Karen Simonyan és Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014. DOI: 10.48550/ARXIV.1409.1556. URL: https://arxiv.org/abs/1409.1556.

[60] Leslie N. Smith és Nicholay Topin. *Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates*. 2017. DOI: 10.48550/ARXIV.1708.07120. URL: https://arxiv.org/abs/1708.07120.

[61] Zsolt Szécsi, Károly Simon és Levente Szélyes. „Argus: Hardware and software system for automatic or semi-automatic photo taking”. *2015 IEEE 13th International Symposium on Intelligent Systems and Informatics (SISY)*. 2015, 37–41. oldal. DOI: 10.1109/SISY.2015.7325407.

[62] Christian Szegedy és mások. „Going deeper with convolutions”. *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, 1–9. oldal.

[63] Sasha Targ, Diogo Almeida és Kevin Lyman. „Resnet in resnet: Generalizing residual architectures”. *arXiv preprint arXiv:1603.08029* (2016).

[64] *The React Framework for Production*. URL: https://nextjs.org/ (elérés dátuma: 2022. ápr. 12.)

[65] Paul Viola és Michael J Jones. „Robust real-time face detection”. *International journal of computer vision* 57.2 (2004), 137–154. oldal.

[66] P.J. Werbos. „Backpropagation through time: what it does and how to do it”. *Proceedings of the IEEE* 78.10 (1990), 1550–1560. oldal. DOI: 10.1109/5.58337.

[67] M. Winter. *Scan Me - Everybody's Guide to the Magical World of Qr Codes*. Westsong Publishing, 2011. ISBN: 9780965900034. URL: https://books.google.ro/books?id=s5ZxqwwYKk8C.