ETDK-dolgozat

Marton Szilárd

Szabó Albert-Balázs

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

Kolozsvár, 2021. május 27–30.

Greensy: Újrahasznosítási szimbólumokat felismerő és a szelektív hulladékgyűjtést elősegítő alkalmazás

Szerzők:

Marton Szilárd

Sapientia Erdélyi Magyar Tudományegyetem, Marosvásárhelyi Kar, Informatika szak, III. év

Szabó Albert-Balázs

Sapientia Erdélyi Magyar Tudományegyetem, Marosvásárhelyi Kar, Informatika szak, III. év

Témavezetők:

dr. Iclănzan Dávid, docens

Sapientia Erdélyi Magyar Tudományegyetem, Marosvásárhelyi Kar

dr. Sulyok Csaba, egyetemi adjunktus

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar

Patakfalvi Örs-Krisztián, szoftverfejlesztő,

Codespring

Ráduly Sándor, szoftverfejlesztő,

Codespring

Ráduly Zalán, szoftverfejlesztő,

Codespring

CODESPRING

OUR CODE YOUR SUCCESS

Kivonat

A mai társadalom egyik égető problémája a már nem használt termékek megfelelő újrafelhasználása. A műanyag 79 százalékát hulladéklerakókban tárolják vagy az óceánba öntik, 12 százalékát elégetik és csak 9 százalékát hasznosítják újra. A termékek nem megfelelő tárolása és alkalmatlan, eredménytelen újrahasznosítása hatalmas károkat okoz. Évente több mint százmillió tengeri élőlény hal meg a műanyag hulladékokkal való szennyezés következtében.

A Greensy alkalmazás a szelektív hulladékgyűjtési folyamatot segíti elő az újrahasznosítható szimbólumok felismerésével, amely által a felhasználók információkat kaphatnak a hulladékok megfelelő tárolásával kapcsolatosan; illetve a szemétgyűjtő cégekről való elérhetőségi információk egy egységes felületen történő megjelenítésével.

A rendszer három részből áll: egy iOS applikációból, egy központi szerverből és egy gépi látásért felelős modulból. Az újrahasznosítási szimbólumok felismerésére modern gépi tanulási módszerek vannak alkalmazva.

Tartalomjegyzék

Bevezető **1**

1. A Greensy projekt **3**

1.1. Funkcionalitások 3

1.2. A rendszer felépítése 4

1.3. Eszközök és munkamódszerek 5

2. A központi szerver **7**

2.1. Felhasznált technológiák és eszközök 8

2.2. Adatmodell 10

2.3. Adathozzáférési réteg 11

2.4. API 11

3. Az iOS alkalmazás **13**

3.1. Felhasznált technológiák és eszközök 13

3.2. Felhasználói felület megvalósítása 16

3.3. Kommunikáció a szerverrel 17

3.4. Újrahasznosítható szimbólumok felismerése 18

3.5. Az alkalmazás használata 18

4. Számítógépes látás **22**

4.1. Elméleti áttekintés 22

4.2. Adathalmaz gyűjtése és előfeldolgozása 24

4.3. Felhasznált technológiák és eszközök 27

4.4. Kipróbált módszerek 28

4.5. A tanítás részletei 30

4.6. Tanítási eredmények 32

Következtetések és továbbfejlesztési lehetőségek **37**

Bevezető

Manapság az újrahasznosítható hulladékok túlnyomó többsége nem kerül újrahasznosításra. A 2018-as év nyertes statisztikája [2] az RSS (Royal Statistical Society) szerint a műanyagok újrahasznosítási arányát célozza meg. 2018-ban a nem újrahasznosított műanyagok aránya 90.5%, amelyből körülbelül 12% elégetésre került és 79% szemétlerakókban végezte. Romániában 2017-ben 4.58 millió tonna hulladékból 641,200 tonna hulladék került újrahasznosításra [3] vagy komposztálásra, amely mindösszesen 14%-nak felel meg.

Az idők során egyre inkább javuló tendenciát mutat a hulladékok újrahasznosítása [3], azonban egy jelentős változáshoz, a technológiai fejlesztések és az alternatív megoldások kidolgozása mellett kiemelten fontos a fogyasztók alapos tájékoztatása és a környezettudatos szemlélet fejlesztése. A felmérések szerint [54] az emberek újrahasznosítással kapcsolatos problémái közé tartozik, hogy túl nagy erőfeszítéssel jár, sok időt és energiát igényel, mert a lakóhelyekhez közel kevés újrahasznosító központ található. Sok esetben előfordul, hogy az embereknek nincs alapos tájékozottságuk a hulladékok újrahasznosítási lehetőségeiről. Bár a termelők és a kereskedők egyre gyakrabban tüntetnek fel oktató jellegű információkat a csomagoláson, vagy esetenként kisebb pénzjutalommal is próbálják ösztönözni az embereket a szelektív hulladékgyűjtésre. Sajnos a gyakorlatban ez korántsem tükröződik jól.

Számos felületen léteznek már kivitelezések, alkalmazások az újrahasznosítás elősegítésére. Többek között az *InfoDeseuri* [6] applikáció, amely lehetőséget nyújt a felhasználók számára részt venni újrahasznosítási programokban, kampányokban. Az *iRecycle* [18] alkalmazás lehetőséget nyújt különböző újrahasznosítási kategóriák szerint csoportosítani a hulladékokat. Emellett gyűjtőközpontok megjelenítését is támogatja az említett kategóriák szerint. A legtöbb mobil- vagy webalkalmazás biztosítja az alapvető funkcionalitásokat, mint a gyűjtőközpontok térképen történő megjelenítését, az újrahasznosítható termékek csoportokba való sorolását, illetve a cégekkel való kapcsolatok megteremtését. Viszont egyik alkalmazás sem rendelkezik olyan funkcionalitással, amellyel a felhasználó könnyedén azonosítani tudná egy hulladékról, hogy újrahasznosítható-e, és amennyiben igen, akkor milyen kategóriába tartozik.

Az újrahasznosítást segítő alkalmazások csoportjába tartozik a *RecycleNation* [20], amely hangfelvétel alapján azonosítja, hogy a felhasználó milyen hulladékról, anyagtípusról vagy szimbólumról szeretne információt kapni. Egy hasonló alkalmazás a *Recycle Mate* [53], amely segítségével egy tárgyról fényképet lehet készíteni. A fénykép elkészítése után, az applikáció beazonosítja a tárgyat, hogy milyen újrahasznosítási kategóriába tartozik.

A Greensy alkalmazás a termékeken lévő újrahasznosítási szimbólumok automatikus felismerésével és osztályozásával próbálja segíteni a felhasználókat, egy hulladékról készült fénykép alapján. Az újrahasznosítási szimbólumok alapján pontosabban meg lehet állapítani egy hulladék anyagának típusát, mivel azok csomagolásai sokszor megtévesztőek [24] lehetnek akár egy ember számára is, viszont a hulladékok környezetvédelmi termékjel (resin code) [5] alapú megkülönböztetése egyértelmű. A felismert szimbólumokról megjelenített információk elősegítik a hulladékok megfelelő tárolását, illetve az alkalmazásban megtalálható gyűjtőközpontok információi segítségével a felhasználók kapcsolatba tudnak lépni a cégekkel és elszállíthatják a hulladékokat.

A projekt fejlesztése 2020 augusztusában kezdődött a U-Hub Mentorprogram[1] keretén belül a Codespring cég munkatársainak mentorálása alatt. Az alkalmazás ötlete a cég által meghirdetett nyári gyakorlat keretein belül fogalmazódott meg. A kivitelezéshez hozzájárultak dr. Iclănzan Dávid docens és dr. Sulyok Csaba egyetemi adjunktus, valamint Patakfalvi Örs-Krisztián, Ráduly Zalán és Ráduly Sándor szoftverfejlesztők. Köszönet illeti a felsorolt személyeket a fejlesztési folyamat során nyújtott szakmai és mentori segítségnyújtásért.

A dolgozat 1. fejezete egy áttekintést ad a teljes projektről és annak elkészüléséről, kifejtésre kerülnek a funkcionalitások, a rendszer architektúrája, a felhasznált eszközök és munkamódszerek. A 2. fejezetben a központi szerver működésének és felépítésének részletes bemutatása történik. Az ezt követő, 3. fejezet az iOS alkalmazás megvalósításába nyújt betekintést, amely a szimbólum felismerését végzi és kilistázza az újrahasznosítással foglalkozó cégeket. A 4. fejezet a szimbólumok felismeréséhez használt gépi tanulásos módszereket, valamint a felhasznált adathalmazt ismerteti. Az utolsó fejezet összefoglalja a dolgozat tartalmát és a projekt két részének, az alkalmazásnak és a számítógépes látás modulnak a továbbfejlesztési lehetőségeit mutatja be.

[1] https://u-hub.ro/mentorprogram/

1. A Greensy projekt

A fejezet bemutatja az alkalmazáson belüli felhasználó szerepkörhöz társított funkcionalitásokat, valamint részletezi a felépített alkalmazásrendszer architektúráját. Legvégül a fejlesztés során felhasznált eszközök és munkamódszerek kerülnek tárgyalásra.

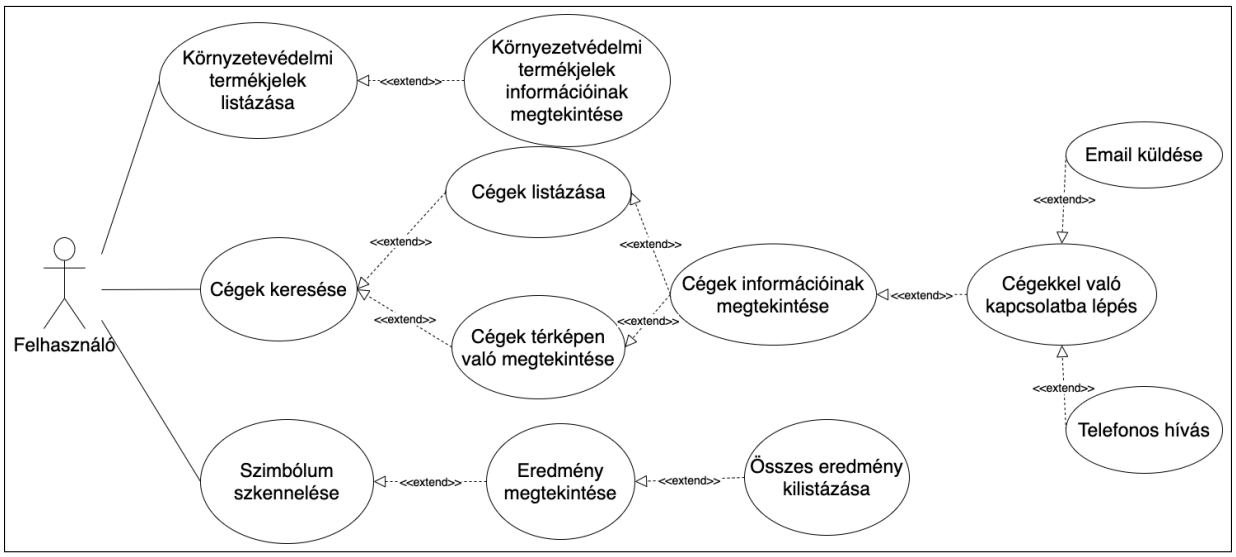
1.1. Funkcionalitások

A rendszer jelenleg egy felhasználói szerepkört tartalmaz, az egyszerű felhasználót, amely az alkalmazás nyújtotta összes funkcionalitáshoz hozzáférést kap (ld. 1. ábra).

A felhasználó számára lehetőség adódik az újrahasznosítással foglalkozó cégek megtekintésére és kilistázására, ezen funkcionalitások a beépített térkép nézet és a cégek listában való megjelenítésével van megvalósítva. Továbbá megtekinthetőek a cégek részletes adatai (mint például címe, gyűjtött termékek kategóriája) és elérhetőségei is.

A felhasználónak egy wiki oldal is rendelkezésére áll, ahol lehetősége adódik a kategóriákba csoportosított a környezetvédelmi termékjelek megtekintésére, illetve részletes információk böngészésére egy adott szimbólumról.

A fennebb felsorolt funkcionalitások mellett a felhasználó számára elérhető az alkalmazáson belül a környezetvédelmi termékjelek szkennelése, amelyhez szükséges egy fénykép készítése, majd az alkalmazás kiértékeli azt. A kiértékelés eredménye tartalmazza az adott termékről, hogy milyen újrahasznosítási kategóriába tartozik, illetve információt nyújt abban az esetben is, ha nem sikerült felismerni az adott szimbólumot.



1. ábra. A funkcionalitások egy szerepkör függvényében: a felhasználóknak cégek kilistázásához, a Wiki oldal megtekintéséhez és a szimbólum szkenneléshez van jogosultsága.

1.2. A rendszer felépítése

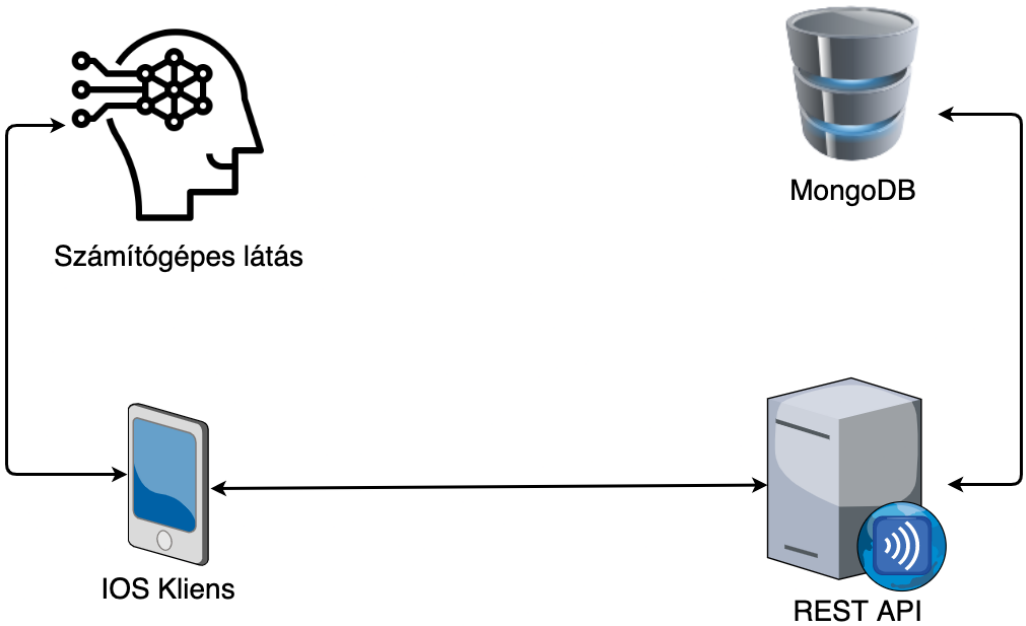
A rendszer felépítése négy fő részből áll: az iOS mobilalkalmazás, a számítógépes látás modul, illetve a REST API, amely a MongoDB adatbázissal kommunikál. A rendszer architektúrai felépítését a 2. ábra szemlélteti.

Az iOS mobilalkamazás a kliens oldali feladatok ellátásáért felelős. Itt találhatóak azok a funkcionalitások, amelyek egy felhasználó számára gördülékenyebbé tehetik az újrahasznosítási folyamatok elvégzését. Ezen belül található egy számítógépes látás modul is, amely a szimbólumok szkennelésének szerves részét képezi. A mobilalkalmazás tényleges felépítése és használata részletesebben a 3. fejezetben olvasható.

A számítógépes látás modul a 3.4 alfejezetben kifejtett szimbólum felismerés modulért felelős. Ezen modul nyújt segítséget a felhasználó által szkennelni kívánt hulladék környezetvédelmi termékjelének a felismeréséért és a megfelelő adatok kinyeréséért (mely termékjelet milyen százalékban ismer fel).

A központi szerver, amely a 2. fejezetben taglalt REST API-t valósítja meg. Ez a szerveralkalmazás felelős a *MongoDB* [13] adatbázissal való kommunikáció felépítéséért, illetve a kliensalkamazástól érkező *HTTP* [23] kérések kiszolgálásáért.

A *MongoDB* adatbázis felelős az adatok sémaszerű eltárolásáért. Az adatok gyűjteményekben (collection), azon belül pedig dokumentumokban helyezkednek el.



2. ábra. Az alkalmazás architektúrai felépítése négy fő komponensből áll: a számítógépes látás modul, az iOS klienalkalmazás, a REST API és a MongoDB adatbázis.

Mindegyik dokumentum egy entitást tárol.

1.3. Eszközök és munkamódszerek

Ebben a fejezetben bemutatásra kerülnek azok a munkamódszerek és eszközök, amelyek a projekt gördülékeny fejlesztéséhez járulnak hozzá.

Verziókövetés

A verziókövetés a Git [31] nyílt forráskódú, osztott verziókövetésre alkalmas szoftverrel történik. A Git osztott architekúrájából adódóan a projekt fejlesztése, fejlesztési története nem csak egy központi tárolón elérhető, hanem a csapat összes tagja rendelkezik a saját másolatával a kódbázisról, amely által a funkcionalitások párhuzamos fejlesztése van megoldva. A projekt fel van osztva tárolókra, amelyek szétválasztják azt három részre (szerver, iOS alkalmazás, számítógépes látás). Az így szétosztott tárolók nem csak az esetleges párhuzamos fejlesztés szempontjából segíthetnek, hanem átlátható részekre bontják fel a projektet. Ezenkívül minden tárolón belül *branch*-eknek nevezett elágazások vannak, amelyek mindegyike egy különálló funkcionalitást kíván implementálni a GitFlow [15] konvenció szerint.

A *branch*-ek kezelése GitFlow konvenció alapján történik. Használata által könnyedén követni lehet a kód alakulását és ezáltal a hibák javítása is egy egyszerűbb feladat. A GitFlow munkamenet lényege *branch*-eknek nevezett elágazások kialakítása, ahol két fő ág, a *master* és *develop* ág létezik. A *master* ágban tárolódik a végleges kód, amelyet a kód stabil verziójaként is tartanak számon, a *develop* ág pedig a fejlesztés alatt álló funkcionalitásoknak és hibajavításoknak ad helyet. A *develop* ágból kiinduló ágak többféle típusúak lehetnek, mint például a *feature*, ami egy új funkcionalitást jelképez. A *develop* ágból kiindult ágak a fejlesztés elkészültével *merge request*-eknek nevezett művelet által visszacsatolódnak a *develop* ágba. A *merge request*-ek GitLab felületen való egységes megjelenítése lehetőséget ad a visszacsatolni kívánt kód más csapattagok általi ellenőrzésére és észrevételek hozzáfűzésére. A csapattagok átnézik a *merge request*-et és az elfogadás után a merge-elni kívánt *feature* ág visszacsatolódik a *develop* ágba.

A Git tárolók könnyebb kezelhetőségére, a folyamatos integráció és kitelepítés használatára a GitLab [32] nyílt forráskódú DevOps (Developer Operations) életciklust kezelő eszköz nyújt lehetőséget. A GitLab egy olyan eszköz rendszer, ahol Git tárolók menedzselését lehet átlátható verziókövetés által könnyedén végezni. A projekt megvalósítása során egységes felületen

létrehozott feladatokat, azok megoldását és ellenőrzését is nagyban segíti. Ezenkívül a GitLab alkalmas a különféle dokumentumok, avagy a projekt megtervezésére használt diagramok és különböző képek tárolására.

Projektmenedzsment

A projekt megvalósítása során a Scrum [68], egy agilis szoftverfejlesztési modell van alkalmazva. A Scrum egy olyan szoftverfejlesztési modell, amelynek nincsenek szigorúan behatárolt szabályai, mindössze útmutatást ad egy projektben résztvevő csapattagok hatékony együttműködésére és egy projekt gördülékeny fejlesztésére azáltal, hogy megengedi a kísérletezést a problémák megoldásában. A kéthetes sprint időszakok alatt dinamikusan történik a megvalósítandó funkcionalitások elkészítése, illetve az elvégzendő feladatok végrehajtása. A sprintek elején mindig az elvégzendő feladatok leszögezése és tervezése történik. A sprintek végén egy demo keretein belül bemutatásra kerül az előző sprint haladásai és visszatekintés is történik a sprint pozitívumaira és negatívumaira. A visszatekintés segít a munkamenet és a csapatmunka javításában.

A feladatok a GitLab beépített Kanban boardján vannak rendszerezve, ahol minden feladathoz tartozik egy becsült idő, ami a csapattagok közös becslésén alapszik.

Folyamatos integráció és kitelepítés

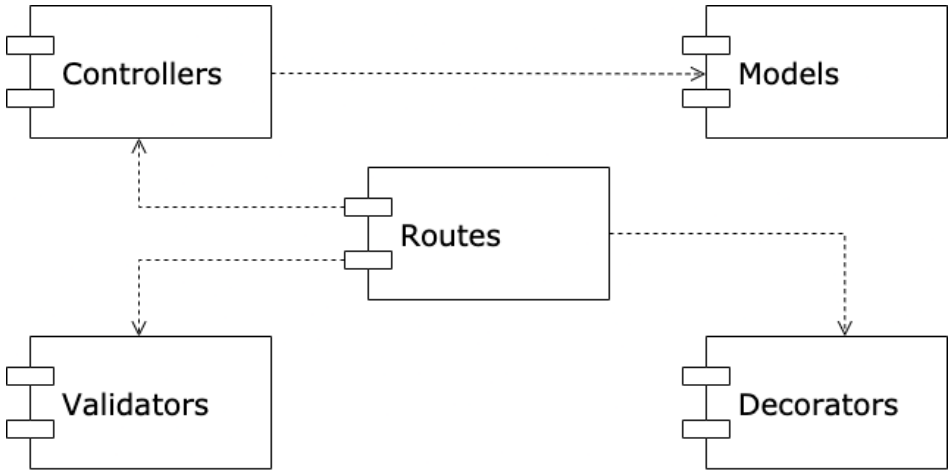
A *CI (continuous integration)*, folyamatos integráció egy iteratív folyamat, amely csökkenti a kódban fellépő hibákat az előre meghatározott tesztelési folyamatok automatikus futtatásának következtében minden egyes új kódverziónál, ezzel megkönnyítve a fejlesztők munkáját. A *CD (continuous delivery)*, azaz a folyamatos kitelepítés, célja a fejlesztési, tesztelési és kiadási folyamatok felgyorsítása és gyakoriságának növelése. A folyamatos integráció és kitelepítés a szerveralkalmazás szerves részét képezi, részletesebben 2.1 alfejezetben olvasható.

2. A központi szerver

Az alkalmazás központi szervere *JavaScript* [28] programozási nyelvben íródik, a *Node.js* környezet felhasználásával. Az adatok tárolása és lekérése a *MongoDB* nyílt forráskódú dokumentumorientált adatbázis szoftver segítségével valósul meg.

A szerveralkalmazás egy RESTful (Representational State Transfer) [57] szoftver architektúra stíluson alapuló API rendszer, melynek segítségével a *HTTP* (Hypertext Transfer Protocol) [23] kérések kerülnek kiszolgálásra. Az architektúra tulajdonságai közé tartozik az alacsony erőforrásigény, egyszerű megépíthetőség és karbantarthatóság, valamint a gond nélküli horizontális skálázhatóság. A szerveralkalmazás öt részre bontható: controllers, models, routes, validators, decorators (ld. 3. ábra)

A *controllers* modulban találhatóak az alapműveletek funkcionalitásainak implementációi. Itt hajtódnak végre a kérések és jönnek létre, kérődnek le vagy éppen frissülnek a meglévő adatok. A *models*-ben találhatóak az adatbázis sémák definíciói, meghatározva a lehetséges mezőket, azok típusát és egyéb jellemzőit. A *routes* modulban az API által biztosított elérési útvonalak valamint a teljesítésükhöz használt függvények (middleware) sorozata van meghatározva. A *validators* modul az adatok ellenőrzéséért felelős, amennyiben nem megfelelő adatok érkeznek a szerverhez, a visszatérített válasz egy hibaüzenet lesz, ami tartalmazza szükséges módosítási javaslatokat. A *decorators* modul kiküszöböli a kód duplikációt azáltal, hogy ismétlődő feladatokat, mint egy entitás azonosító alapú kikeresése, kiemel újra



3. ábra. Komponens diagram, melyen a szerver öt fő modulja látható, a *routes*, mely a kérések fogadásáért és irányításáért felelős, a *validators* ahol opcionálisan ellenőrzések hajtódnak végre, a *decorators* ami előre elvégez általános műveleteket, ezzel könnyítve a következő *controllers* réteg munkáját, amely utolsó lépésként a *models* modulban definiált sémák segítségével végrehajtja a kért műveleteket.

felhasználható függvényekbe és a kikeresett információval kiegészíti (dekorálja) a kéréseket. Ezen rétegek működésének részletesebb leírása a 2.4 alfejezetben olvasható.

2.1. Felhasznált technológiák és eszközök

Node.js [69] egy nyílt forráskódú, cross-platform *JavaScript* futtatási környezet, ahol a *JavaScript* kód futtatásáért a Google által fejlesztett V8 JavaScript-motor felelős. A *Node.js* egy szálon fut, működési alapelve a ciklikus eseménykezelés (*event-loop*) és az aszinkron műveleteket részesíti előnyben. Az *event-loop* [74] lényege a végrehajtásra váró feladatok és kérések egymás utáni sorakoztatásában rejlik, ahol a környezet rendre végzi el ezeket feladatokat, így drasztikusan csökkentve az alkalmazás erőforrásigényét. A futtatási környezet 12.18.4 verziója van használva.

Npm, teljes nevén Node Package Manager [71], a *Node.js* [69] futtatási környezet csomagkezelője. Segítségével könnyedén kezelhetőek és telepíthetőek a szerverhez használt könyvtárak és egyéb függőségek.

Express [30] a *Node.js*-hez [69] használt egyik legelterjedtebb keretrendszer *HTTP* [23] szerver megvalósítására. A könyvtár hasznos és robusztus funkciókat kínál, melyek nagy segítséget nyújtanak webalkalmazások fejlesztésénél. Ilyen például a *HTTP* szerver egyszerű konfigurálása, a kérések irányítása, alapvető hibaesetek lekezelése és úgynevezett *middleware*-ek írásának támogatása, melyek a feldolgozás modularizálásban játszanak fontos szerepet.

ESLint [75] egy statikus kódelemző eszköz, amely felismeri a helytelen mintákat tartalmazó JavaScript kódrészleteket. Személyre szabhatóak a beállításai és mintái, így segítségével folyamatosan fenntartható a kód minősége és stílusának konzisztenciája. Az eszközzel együttműködő *Prettier* bővítmény a kód automatikus formázásában nyújt segítséget a meghatározott minták alapján, ezzel megkönnyítve a kódolási folyamatot és a formázási hibák elkerülését.

MongoDB [13] egy nyílt forráskódú dokumentumorientált adatbázis szoftver, amely a NoSQL adatbázisszerverek csoportjába tartozik, ahol adatbázis sorok helyett dokumentumokról beszélhetünk melyek táblák helyett kollekciókba vannak csoportosítva. Az adatokat az adatbázis BSON (*Binary JavaScript Object Notation* [29]) formátumban tárolja el, míg interakció során (adatok írása és visszatérítése) olvasható és szélesen eltejedt JSON (*JavaScript Object Notation* [61]) formátumot használ. A MongoDB alkalmas választásnak bizonyul, mivel

kiválóan teljesít szöveges információk tárolásában és támogatja a földrajzi (*geospatial*) típusú adatokat és azok hatékony lekérését.

Mongoose ODM [37] (*Object Document Mapping*) egy olyan könyvtár, amelynek segítségével kapcsolatot lehet teremteni a Node.js szerver és a MongoDB adatbázis között objektum modellezési szinten. A könyvtár támogatja az aszinkron környezetekben történő feladatok elvégzését, és használata sok előnyt nyújt:

- *automatikus típuskonverzió* - a kérésekben megkapott szöveges adatokat a meghatározott típusokra alakítja át
- *validálás* - minden egyes kérésnél egy ellenőrzés megy végbe, ami során a kapott paraméterek helyessége kerül ellenőrzésre
- *sémák létrehozása* - lehetőséget nyújt a dokumentum alapú adatok rendszerezésére és csoportosítására

A könyvtár segítségével felépített sémák beépített függvények sorát teszik elérhetővé az adatok létrehozására, szűrésére, módosítására és törlésére.

A szerveralkalmazásban továbbá a következő fontosabb csomagok vannak felhasználva:

- **Helmet** - segít biztonságosabbá tenni a API-t, különféle *HTTP* fejléceket kezelésével
- **Morgan** - az API-hoz befutó kérések automatikus logolását végzi
- **Dotenv** - lokális fejlesztés során a környezeti változók betöltéséért felelős egy *.env* kiterjesztésű fájlból
- **Nodemon** - egy olyan fejlesztési eszköz, amely automatikusan újraindítja a szervert minden fájl változást követően

Docker [56] egy virtualizációs eszköz, amely operációs rendszer szintű virtualizációt használ szoftverek konténerekben történő futtatásához. A konténerek elkülönítve léteznek egymástól és egyenként tartalmazzák az adott szoftver futtatásához szükséges könyvtárakat és forrásfájlokat. Egy konténerbe csomagolt alkalmazás futtatható bármilyen UNIX alapú rendszeren, amely rendelkezik *Docker Engine*-nel.

Docker Compose [42] azon eszközök közé tartozik, amely képes több konténerből álló Docker alkalmazásokat menedzselni. Konfigurálása YAML (*YAML Ain't Markup Language*

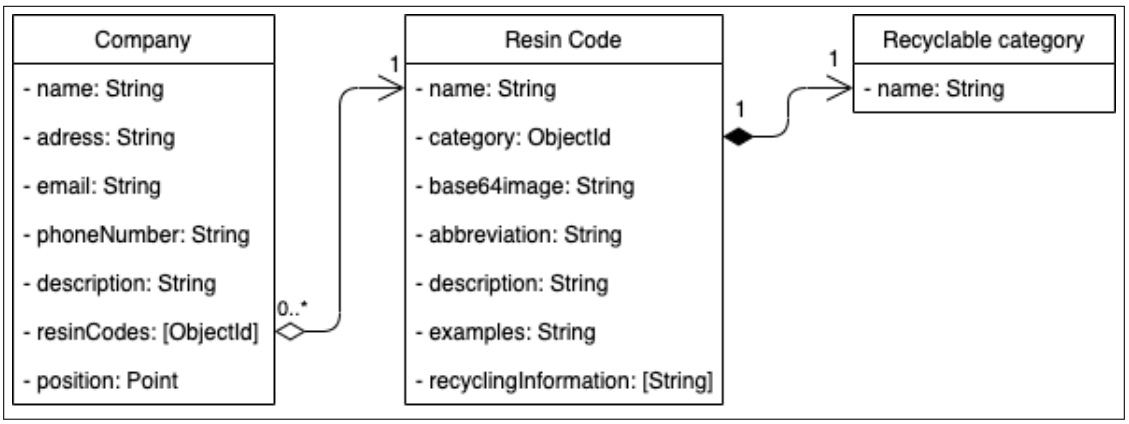
[8]) állományok segítségével történik, amelyekben az egyszerre futtatni kívánt konténerek definícióit és beállításait lehet meghatározni. *Docker Compose* segítségével van megvalósítva a központi szerver és a *MongoDB* adatbázis konfigurálása, egyidejű futtatása, illetve a kettőjük közt levő kapcsolat felépítése.

GitLab CI/CD [62] egy beépített eszköze a GitLabnak [32], amely megfelelően konfigurálva automatizálni tudja a szoftverfejlesztés számos lépését, pontosabban a folyamatos integrációt, amely részletesebb leírása megtalálható 1.3 alfejezetben. A dolgozat írásának pillanatában három automatizált feladat van beépítve: *lint*, *build* és *deploy*. A *lint* fázis a kód formázásának és szintaktikai helyességének ellenőrzéséért felelős az *ESLint* statikus kódelemző segítségével. A *build* a szerveralkalmazás Docker imagebe történő csomagolását végzi. A *deploy* fázis pedig a csomagolt Docker image elindítására, leállítására és újraindítására biztosít manuálisan elindítható feladatokat.

2.2. Adatmodell

A központi szerverben az adatok kezelésére három fő és egy segéd séma van definiálva: *Company, Recyclable category, Resin code* és *Point* (4. ábra).

A **Company** séma tartalmazza az összes felvezetett újrahasznosítással foglalkozó céget. A sémában definiált adattagok közé tartozik a cég neve, címe, e-mail címe, telefonszáma, általános leírása, egy lista amely tartalmazza, hogy milyen típusú termékeket hasznosítanak újra, illetve a koordináták, ahol a cég található. Ezen adatok kötelező jelleggel jelen kell legyenek annak érdekében, hogy a kliensalkalmazás egy leegyszerűsített formában lehetőséget tudjon adni a felhasználók számára, a cégekkel történő kapcsolatteremtésre.



4. ábra. A szerveroldalon három fő séma van definiálva, a *Company*, mely az újrahasznosítási cégek adataival foglalkozik, a *Recyclable category* amely a lehetséges újrahasznosítási kategóriákat tartalmazza, valamint a *Resin codes* a környezetvédelmi termékjelek tárolására.

A **Recyclable category** séma tartalmazza az összes támogatott újrahasznosítási kategóriát (mint műanyag, papír, fém). A kollekció sémája egy név mezővel rendelkezik, amely kötelező és a kategóriát reprezentálja.

A **Resin code** sémában található az összes Európában elismert újrahasznosítási szimbólum, hivatalos nevén környezetvédelmi termékjel. A séma mezői között megtalálható a szimbólum neve, a kategóriája, amelybe tartozik, az ikonja *Base64* [38] kódolásban, a kategóriájának rövidítése és rövid leírása, amely információt tartalmaz az esetleges újrahasznosítási módszerekről és lehetőségekről. Ezek mellett a séma lehetőséget biztosít példák felsorakoztatására is arról, hogy milyen gyakori termékek készülnek az adott szimbólum által jelölt anyagból és hogy maga az anyag újrahasznosítható-e vagy sem.

A **Point** egy segéd séma, mely földrajzi koordináták mentésére alkalmas és a *Company* modellben van felhasználva a cégek koordinátáinak tárolására. A *MongoDB* által meghatározott *geoposition* formát követi, ami szükséges a földrajzi lekérdezések megvalósításához. A sémában két szám kerül tárolásra egy listában, amelyek a szélességi és hosszúsági fokokat jelentik. Külön segéd séma lévén a cégek pozícióján túl máshol is felhasználható, ahol koordináta alapú adattárolásra van szükség.

2.3. Adathozzáférési réteg

Az adatbázis és a szerver közti kommunikáció a 2.1. fejezetben bemutatott Mongoose ODM segítségével valósul meg. Az ODM lehetőséget ad az adatok sémákba való rendezésére és a sémák által azok egyszerű kezelésére előre megírt függvények segítségével. Az adatbázisban a sémáknak megfelelően kollekciókat hoz létre a könyvtár, amelyekben belül az adatok úgynevezett dokumentumok formájában kerülnek tárolásra. A Mongoose által biztosított metódusokkal lehet dokumentumokat elmenteni, frissíteni, törölni és lekérdezni, mindezt anélkül, hogy MongoDB specifikus szkripteket és lekérdezéseket kellene írni. Minden adatmodellen négy alapművelet (CRUD) végrehajtását teszi lehetővé az API: létrehozás, olvasás, módosítás és törlés. Ezen műveletek elvégzéséért a *controllers* rétegben található függvények felelősek, amelyeknek keresztül hozzáférhetünk az adatbázisban levő adatokhoz.

2.4. API

Az API a hozzá intézett kérésekre adott válaszokban az adatbázisban található dokumentumokat használja fel. Az újrahasznosítással foglalkozó cégek adatainak az

összegyűjtése jövőbéli terv, így jelenleg *mock* adatokkal van feltöltve az adatbázis ezen kollekciója. Az újrahasznosítási kategóriák, illetve a környezetvédelmi termékjelek a Wikipédián [78] elérhető adatokat felhasználva lettek létrehozva.

A beérkező kérések feldolgozásának a folyamata a következő: a kliensalkalmazás egy HTTP kérést küld a már futó központi szerver felé, amelyet az fogad, feldolgoz, ellenőriz, elvégzi a kért műveletet és visszatéríti a megfelelő választ.

Kezdetben a kérés a *routes* rétegbe érkezik, ahol a szerver eldönti a beérkezett HTTP URI szerkezetének és a kérés típusa alapján, hogy milyen műveletek szükségesek a kérés kielégítéséhez. Ezt követően kerülnek meghívásra az adott végpontok definícióinál felsorakoztatott műveletek (middleware). A *validators* réteg ellenőrzi a beérkező kérés paramétereit és adatait, például egy *POST* kérés esetében ellenőrzi a kérés törzsében levő adatok létezését valamint azok típusait, és csak a helyesség ellenőrzése után hívódik meg amennyiben szükséges a *decorators* réteg. Ha szükséges, akkor a megfelelő dekorátor kikeresi az adatbázisból a keresett indexű dokumentumot és továbbadja azt a következő *controllers* rétegnek, ezzel egyszerűsítve annak munkáját. A *controllers* rétegben vannak az adatmanipulációs műveletek, melyek a *models* rétegben definiált sémákat, és azok által biztosított beépített függvényeket használják. A szerver az adatokat *JSON* formátumban várja el és az API által visszatérített információk is *JSON* formátumot követnek. A kérések 4 típusát támogatja a szerveralkalmazás, amelyek az Adathozzáférési réteg alfejezetben említett CRUD műveletek alapján kerültek megírásra: *GET*, *POST*, *DELETE*, *PUT*.

Az API szerver *GET* kéréseihez tartoznak:

- */api/companies* - visszatéríti az adatbázisban szereplő összes cég adatait
- */api/categories* - visszatéríti az összes újrahasznosítási kategóriát
- */api/resin-codes* - visszatéríti az összes Európában elismert újrahasznosítási szimbólumot és ahhoz tartozó adatokat
- */api/resin-codes?categoryId=* - visszatéríti az adott kategóriához tartozó újrahasznosítási szimbólumok adatait

Az adatok manipulációja *PUT* és *DELETE* műveleteken keresztül valósul meg, a *GET* kérésekhez hasonlóan, annyi különbséggel, hogy a elérési útvonal vége egy azonosítóval bővül (*/api/companies/companyId*), amely meghatározza az adatbázisban keresendő dokumentumot.

3. Az iOS alkalmazás

Az iOS alkalmazás valósítja meg a termékeken megjelenő újrahasznosítással kapcsolatos szimbólumok felismerését, illetve az újrahasznosító központok megjelenítését, a felhasználó tartózkodási helye környezetében. Jelen fejezet részletezi a fejlesztés során felhasznált technológiákat és eszközöket, illetve ismerteti az alkalmazás implementációs részleteit.

3.1. Felhasznált technológiák és eszközök

A jelenlegi alfejezet a telefonos alkalmazás fejlesztése során felhasznált technológiákat és eszközöket mutatja be. A felsorolt technológiák és környezetek szerves részét képezik a fejlesztési folyamatnak.

Swift

A **Swift** [27] (v5.4) egy nyílt forráskódú, hatékony és intuitív programozási nyelv többek között iOS operációs rendszerekre történő fejlesztésre. A *Swift* kód írása interaktív és könnyen átlátható, a szintaxisa tömör, mégis kifejező. A Swift kód tervezése szerint biztonságos, ugyanakkor villámgyorsan futó szoftvert is gyárt.

CocoaPods

A **CocoaPods** [22] (v1.10.1) egy függőségmenedzsment eszköz *Swift* [27] és *Objective-C* [39] alapú projektek esetében. A projekt függősségeit a `Podfile` állomány tartalmazza (ld. 1. kódrészlet), amelyek dinamikusan töltődnek be az alkalmazás indulása során.

```
...
  use_frameworks!

  pod 'FloatingPanel', '2.1.0'
  pod 'iCimulator', '1.5'
  pod 'TensorFlowLiteSwift', '2.0.0'
end
...
```

1. kódrészlet. Az alkalmazás függőségeinek definicióit a Podfile állomány tartalmazza.

Xcode

Az **Xcode** [40] (v12.5) egy fejlesztői környezet *macOS, iOS, iPadOS, watchOS* és *tvOS* operációs rendszerekre, amelyet az Apple fejlesztett ki. Az *Xcode* beépített *LLDB* [45] komponense grafikus hibakeresési opciót kínál a fejlesztő számára, ahol forráskód, valamint gépi kód szinten lehet kiszűrni a felmerülő problémákat. Fő funkcionalitásai közé tartozik, hogy egyedi interfész készítő modullal rendelkezik, amellyel könnyedén létrehozhatóak egyedi nézetek. Ezen nézetek grafikus felülete könnyedén személyre szabhatóak az alkalmazáson belül.

Az *Xcode* egy beépített iPhone szimulátor alkalmazással is rendelkezik, amellyel a fejlesztés alatt álló iOS alkalmazás egyszerűen tesztelhető, még mielőtt egy valós eszközre kerülne. Egy kifogásolható része többek között a kamera szimulációja, ami elmaradottnak számít az *Android* fejlesztői környezettel szemben.

MapKit

Az alkalmazás egyik fő részét képezi az újrahasznosító központok térképen való megjelenítése, amelyre a *MapKit* [59] ad lehetőséget. Ez egy interaktív térkép, amely segítségével szabadon lehet navigálni a világ bármely pontján, mindemellett elhelyezhetőek annotációk, amely cégek székhelyeit szimbolizálják és információt közvetítenek a felhasználó felé.

AVFoundation

Az **AVFoundation** [55] egy teljeskörű keretrendszer, amelyet audió-vizuális médiával való együttműködéshez lehet használni. A keretrendszer egy *AVCaptureSession* osztállyal rendelkezik, amely kezeli a rögzítési tevékenységet, és koordinálja az adatfolyamot a bemeneti eszközről a kimeneti rétegbe. Az *AVCapturePhotoOutput* egy felületet biztosít az állóképekkel kapcsolatos munkafolyamatok rögzítéséhez. Az állóképek alapvető rögzítésén túl a fénykép kimenet támogatja a *RAW* formátumú rögzítést. Ez fontos elsősorban a kép *Data* típussá való átalakításában és tárolásában. A rögzített fényképeket egy *CVPixelBuffer* típusban adja vissza, amely bemenetként szolgál a gépi tanulásos modellnek.

TensorFlow Lite

A **TensorFlow Lite** [64] egy nyílt forráskódú, mobil eszközökre optimalizált, gépi tanulással kapcsolatos műveletek elvégzését támogató keretrendszer. *FlatBufferek*et [72] használ a hálózati modellek adatszerializációs formátumaként, elkerülve a szabványos modellek által használt *Protocol Buffers* formátumot. Ez számos előnyt nyújt, mint például a gyorsabb kiértékelés, amely lehetővé teszi az alkalmazás számára, hogy korlátozott erőforrások használatával is hatékonyan működjön. Az integrációó részletesebb leírása a 4.3 alfejezetben olvasható.

iCimulator

Az **iCimulator** [77] (v1.5) könyvtár segítségével lehetővé lehet tenni az iMAC kamera kimenetének az átirányítását az *Xcode* fejlesztői környezetbe beépített szimulátor kamerájához. A könyvtár egy *UDP* [12] szerver működésén alapszik, amely létrehozza a kapcsolatot a két fél között és megoldja a képkockák átirányítását. Ennek segítségével sikeresen tesztelni lehet szimulátoron egy alkalmazás kamera funkcióját.

Floating panel

A **Floating panel** [80] egy azon UI elemek közül, amely dinamikusan felugró nézetként funkcionálnak. Jelen van a térkép nézet annotációra való érintésekor, amikor a képernyő alsó részéből ugrik fel egy kis részt letakarva a képernyőn. A felugró ablak fel- és lehúzható, jelenleg három állapottal rendelkezik: kis méretben, félig és teljesen kihúzott. Mindegyik állapot az adott ablak képernyőn elfoglalt méretéért felelős.

Scrollable segmented control

A *segmented control* egy UI komponens, amely nézetek közti váltásért felelős, azaz szegmens alapú navigációt biztosít. A **ScrollableSegmentedControl** [60] egy specifikus UI komponens, amely alapjaiban a *segmented control* elemre hasonlít. Fő tulajdonságai a horizontális görgethetőségében rejlik, így egymás mellett az adatok számosságától függően jeleníthető meg annyi elem, amennyire szükség van. A **ScrollableSegmentedControl** használata kedvezőbb a beépített *segmented control* komponenssel szemben, abban az esetben ha több szegmens elemet szükséges megjeleníteni, és azok nem férnek ki képernyőn.

3.2. Felhasználói felület megvalósítása

A felhasználói felület több komponensből tevődik össze. Az *assets* tartalmazza az alkalmazásban felhasznált egységes színskálákat és egyedi ikonokat. A *UI Elements* könyvtár tartalmazza az újrafelhasználható UI komponenseket, mint például a horizontálisan görgethető szegmens vezérlőt, illetve a kinézeteken megjelenő lekerekített gombot.

A felhasználói felület megvalósítása főként a *Storyboard* [46] segítségével történik. A *Storyboard* magába foglalja azokat a nézeteket, amelyekkel az alkalmazás rendelkezik, illetve definiálja a nézetek közötti navigáció típusát. A *Storyboard*on belüli nézetek *segue*ek segítsével vannak összekötve, amely által meg van határozva az alkalmazás navigációs statikus folyama. Fő navigációs elemként *bottom navigation bar* szolgál, amely a *Recyclers*, *Scan* és *Wiki* nézetek közötti váltásért felel. Emellett az alkalmazás rendelkezik olyan nézetekkel is, amelyek a *tab bar*tól függetlenül jelennek meg, támogatva a *stack* alapú navigációs folyamatokat.

Az alkalmazás esetében úgy a *Recyclers*, mint a *Wiki* nézetben, egyaránt megtalálható egy horizontálisan görgethető szegmens vezérlő. A *Recyclers* nézet esetében, kulcsfontosságú szerepet játszik a térkép, illetve lista közti váltásért, illetve a *Wiki* jellemzően megnyilvánul a görgethető funkcionalitása, ahol több újrahasznosítási kategória létezik és ezek nem férnek ki a képernyőn horizontálisan.

Egy másik felhasznált UI komponens a *Floating panel* [80], amely az adatok felugró ablakként történő vizualizálásban nyújt segítséget. A *Recyclers* nézetben az újrahasznosítással foglalkozó cégek adatainak, míg a *Scan* nézetben a felismert szimbólum eredményének megjelenítéséért felelős.

A *Recyclers* navigációs elem megérintésével legelőször a térkép nézet töltődik be. Ezen nézet a már említett horizontálisan görgethető szegmens vezérlőből és egy konténer dobozból épül fel, ahol a konténer doboz tartalmazza a szegmens vezérlő által váltogatható térkép, illetve lista nézetet. A térkép nézet a *MapKit* beépített könyvtárral valósul meg, a térképen annotációk jelennek meg. Három típusú annotáció van implementálva: a cégek megjelenítésére szolgáló, a felhasználó helyzetét jelölő, illetve a klaszterek megjelenítésére szolgáló annotációk. Ezen annotációk egyedi kinézettel rendelkeznek, illetve a térképbe való közelítés mértékétől függően klasztereződnek. A klaszterezett annotációk nem reagálnak érintésre, viszont feltüntetik hogy hány annotáció csoportosul az adott klaszter alatt.

Az annotációk a cégek térképen való megjelenítéséért felelősek és egyedi animációval rendelkeznek, amely az annotáció megérintésekor lép életbe, ezt követően megjelenítődik egy

felugró ablak. A felugró ablak sémaszerűen jeleníti meg a kiválasztott cég adatait egy táblában, kezdetben a képernyő alján egy kis méretben a cég neve és címe tárul a felhasználó elé, teljesen kihúzott állapotában több információ kerül előtérbe. A táblán belül minden egyes cella egyedi kinézettel rendelkezik, és más-más adat megjelenítéséért felelős, mint például az elérhetőségek és újrahasznosítható termék kategóriák. A felsorolt újrahasznosítható kategóriák egy *collection view* UI komponensben vannak elhelyezve címkékként, ahol egy címke egy *XIB* vizuális felületben megvalósított újrafelhasználható UI elemként szolgál.

A *Wiki* navigációs elem alatt szintén egy listanézet található. Egy lista elem egy képpel és három címkével rendelkezik, ahol a kép a környezetvédelmi termékjel ikonját, míg a három címke a környezetvédelmi termékjel kategóriájának rövidítését, nevét és mindennapi életben megtalálható termékek példáit írja le. A környezetvédelmi termékjel adatlapját egy statikus táblanézet alkotja, testreszabott cellák használatával, ahol a listaelemek dinamikusan méreteződnek az adatok hosszától függően.

3.3. Kommunikáció a szerverrel

A telefonos alkalmazás *HTTP* [23] kéréseken keresztül kommunikál a központi szerverrel. A kliens oldali kérések megvalósítását a *RestManager* osztály valósítja meg, amely egy *URLRequest* adattípussal építi fel a kérést. A *Swift* támogatja az aszinkron műveletek végrehajtását, így könnyedén kezelhetővé válik a kérések válaszainak lekezelése. Az aszinkron műveletek *escaping block*okon keresztül továbbítják a szervertől kapott adatokat a UI fele.

A *HTTP* kérésekre *JSON* [61] formátumban érkeznek a válaszok a REST API-tól. A válasz megérkezése után egy ellenőrzési szakasz lép életbe, első sorban vizsgálatra kerül, hogy a kommunikáció során történt-e hiba, majd a visszatérítettt HTTP kód értélődik ki. Abban az esetben, ha a visszatérítettt HTTP kód a megefelő tartományba esik a szerver által küldött adatok elemzése következik. Amennyiben egyik ellenörzési rész is hibával tér vissza, a kérés sikertelennek minősül és a *mock* adatok kerülnek betöltésére a felületen.

A *RestManager* a feladatokat további menedzsereknek osztja le, úgy hogy a másodlagos menedzserek öröklik a *RestManager* tulajdonságait: *CompanyRestManager*, *CategoryRestmanager*, *ResinCodeRestManager*. Mindegyik osztály egy adott sémának megfelelő adat átalakításért felelős, ahol *JSON* formátumú adatot a megfelelő entitássá alakítja át.

A teljes folyamat fő csomópontja a *DataProvider* osztály, amelynek a fő feladata a

UI vezérlőktől érkező kérések feldolgozása és továbbítása a megfelelő menedzsereknek. A *DataProvider* osztály ellenőrzi az internetkapcsolat állapotát, annak megléte nélkül betölti a *mock* adatokat. A *mock* adatok „dummy„ adatok, amelyek csak a funkcionalitások működésének reprezentálásához vannak létrehozva, jelenleg nem valós adatokat szolgáltatnak a felhasználó számára, csak az alkalmazás tesztelésének célját képezik.

Az adatok UI vezérlőkhöz való érkezésekor a környezetvédelmi termékjelek esetében egy kiegészítő lépéssel egészítődnek ki: a termékjelek ikonjainak *base64* [38] dekódolásával. Amennyiben a kapott dekódolt *base64* karakterlánc érvénytelen mintát tartalmaz, az esetben egy *placeholder* kép fog bekerülni az ikon helyett.

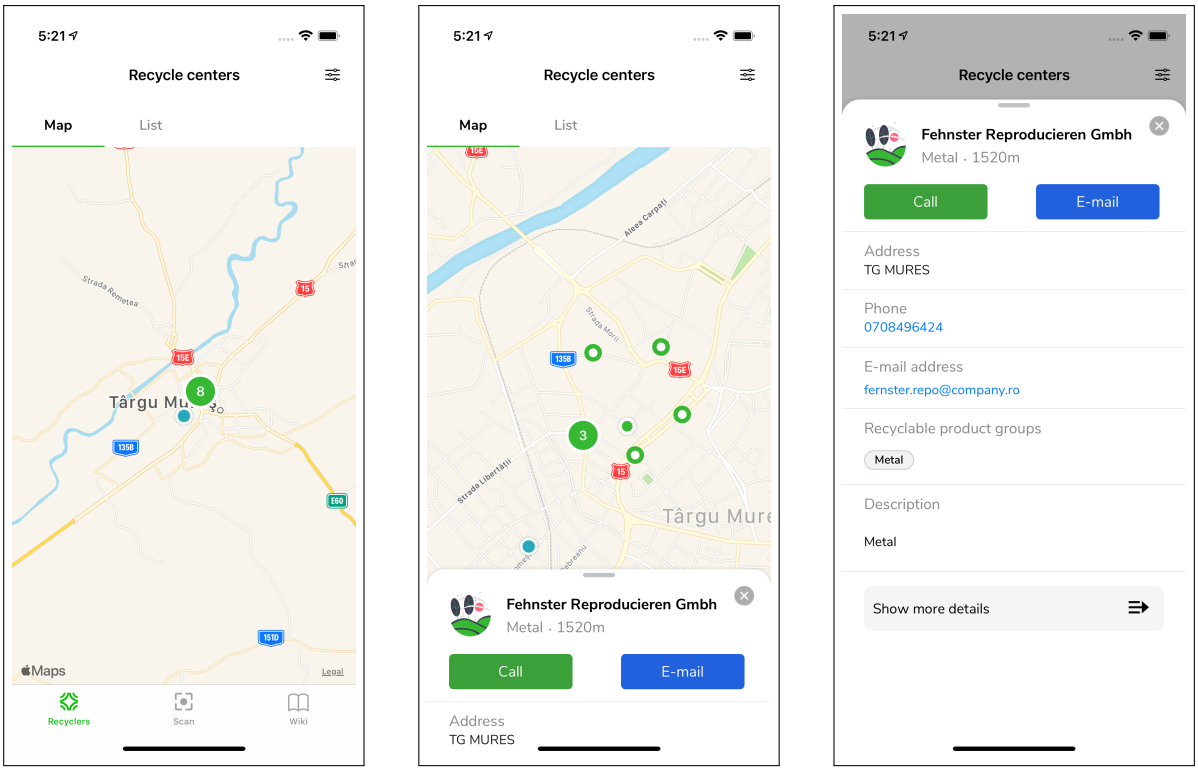
3.4. Újrahasznosítható szimbólumok felismerése

A *Scan* nézetre való váltáskor automatikusan betöltésre kerül egy mély tanulásos *TensorFlow Lite* [64] modell és a hozzá tartozó statikusan hozzáadott szimbólumok halmaza. Egy iOS alkalmazás csak *CoreML* modelleket képes feldolgozni, itt kerül képbe a *TensorFlowLite*, amely megkönnyíti a gépi tanulásos modell betöltését és futtatását. A gépi tanulásos modell új, *TensorFlow 2* változata nem támogatja a modellek *CoreML*be való átalakítását.

Az alkalmazás *scan* funkciójának elindítása következtében elindul a felismerési folyamat, ahol a kép átalakítására kerül sor *CVPixelBuffer* típusból egyszerű *Data* típusra. A konverzió minden egyes kép készítés esetében szükséges azért, mert a gépi tanulásos modell bemeneti értékének a dimenziója meg kell egyezzen a kép dimenziójával. A kép átalakítása *Data* típusba megkönnyíti az adat struktúrájának módosítását. Ezt követően kezdődik el az adat kiértékelése és az eredmények meghatározása. Eredményként egy lista jelenik, ahol mindegyik elem egy *Inference* típus lesz, amely rendelkezik egy osztálynév és pontosság mezővel. Ez a lista kerül továbbküldésre az adatokat megjelenítő réteghez.

3.5. Az alkalmazás használata

Az alkalmazás a nézetek közti navigáláshoz egy *tab bar*t biztosít, ahol az alkalmazás fő funkcionalitásai érhetőek el: gyűjtőközpontok térképen és listában való megjelenítése, a környezetvédelmi termékjelek szkennelése, illetve a kategóriákba sorolt termékjelek wiki oldalának megtekintése.



5. ábra. Az első képen a térkép nézet látható, ahol a cégek térképen való elhelyezkedésük annotációi klaszterezve vannak. A ráközelítés után megjelennek a cégek pontos annotációi, amelyek érintése során felugrik a képernyő alján látható ablak, és ezt a nézetet felhúzva látható részletesebb információ az adott cégről. A felugró ablakot teljesen lehúzva vagy az „x„ gombra érintve lehet eltüntetni.

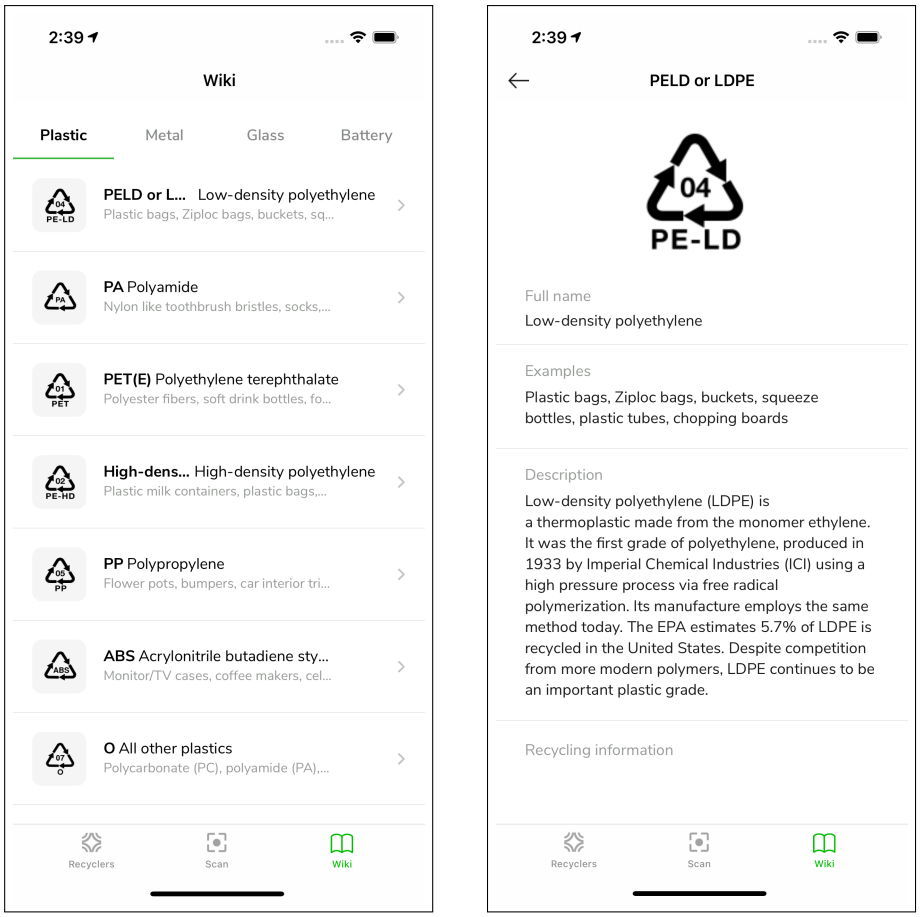
A **Recyclers** nézet, ahogyan az 5. ábra is mutatja, a gyűjtőközpontok térképen és listában való megjelenítéséért felel. A térkép legelső megjelenésekor az alkalmazás engedélyt kér a jelenlegi GPS pozíció használatára, annak érdekében, hogy a felhasználót lokalizálja és az ő közelében lévő cégeket jelenítse meg. A térképen feltüntetett cégek helyszíneinek annotációi ki-, illetve beközelítéssel klaszteresődnek dinamikusan. Egy cég annotációjának megérintéskor a cégnek adatait lehet előhívni, a képernyő aljából felugró ablak segítségével. Ezen ablak elsősorban a cég nevét, típusát, címét jeleníti meg és a felhasználó távolságát az adott cégtől, emellett két kapcsolatba lépési opciót kínál fel: hívás és e-mail írása. Az ablak teljesen felhúzott állapotában megjelenik a cég teljes adatlapja, ahol további információ található az elérhetőségekről, cég leírásáról, azon termékkategóriákról, amelyek újrahasznosításával foglalkozik a cég és egy *„more info„* gomb, amely megérintésével lehetőség adódik átmenni egy még részletesebb nézetre, ami bővebb információkkal szolgál az adott cégről a felhasználó számára. A térkép nézet mellett található egy listanézet is az összes cég adataival, amely egy átláthatóbb nézetet nyújt. Egy listaelemet kiválasztva egy újabb nézet jelenik meg a képernyő

6. ábra. A scan gomb lenyomásával fénykép készül, majd a fénykép kiértékelődése után megjelenik a képernyő alján látható egy nézet, amelyen látható az összes felismert szimbólum ikonja, neve, kategóriája és pontossága. Itt szintén lehetőség adódik egy részletesebb nézet megtekintésére.

jobb oldalából, a cég részletesebb adataival.

A **Scan** nézetben rejlik az alkalmazás fő funkcionalitása, az újrahasznosíthatási szimbólumok felismerése (ld. 6. ábra). A nézetre való ugráskor a felhasználónak egy kameranézet jelenik meg, ahol szükséges engedélyezni az alkalmazás számára a telefon kamera hozzáférését, ellenkező esetben lehetetlenné válik a szkennelés. Az engedély megtagadása következtében minden újonnani alkalmazás elindításakor felugrik egy ablak, ahol újból engedélyt kér az alkalmazás a felhasználótól. A megfelelő engedélyek megadása után a felhasználó elkezdheti a szkennelési fázist, ami egy fénykép készítéssel történik. Ezt követően a háttérben az alkalmazás kiértékeli a képet és visszatéríti a felismert szimbólumot, annak nevét, képét és pontosságát. Amennyiben a felhasználó további adatokat szeretne megtudni az eredményről lehetőség nyílik egy részletesebb nézet megjelenítésére a felugró ablakban megérintve a *„more info„* gombot. A részletesebb nézetben a felismert környezetvédelmi termékjelről is található kiegészítő információ.

A **Wiki** nézet tartalmazza az összes újrahasznosítási szimbólumot kategóriákra lebontva. Az



7. ábra. Az első képen látható lista tartalmazza az összes rendelkezésre álló újrahasznosítási szimbólumot és annak nevét, kategóriájának hivatalos rövidítését, példáit a mindennapokban fellelhető termékekről. Egy elemet kiválasztva megtekinthető az adott szimbólum részletesebb leírása.

7. ábra első képén látható listaelemekként jelenítődnek meg a környezetvédelmi termékjelek és azoknak ikonja, megnevezése, kategóriája, leírása és felhasználási területei. A felhasználó útmutatás érdekében itt szerezhet bő információt egyes szimbólumokkal kapcsolatosan, illetve tájékozódhat edukatív jelleggel a részletezett leírás és mindennapi termékekben való felhasználási példák segítségével.

4. Számítógépes látás

A vizuális megkülönböztetése az objektumoknak segítette, illetve segíti az emberiséget túlélni és fejlődni. A számítógépes látás az emberi látáshoz hasonlóan működik [10]. Mind az emberi, mind pedig a számítógépes látás három nagy részre osztható: A fény átalakítása valamiféle jellé, amit az adott rendszer értelmezni tud. Az embernél ez az átalakítás a neuronok által valósul meg, a számítógépeknél a mesterséges neuronok felelősek ezért; ezt a jelet egy mechanizmus segítségével a mesterséges neuronok továbbközlik, hasonlóan mint az emberben található neuronok. A végső fázis a továbbvitt jel értelmezése, amely a gépi látásban mesterséges neuronok súlyozott szavazatán alapszik.

4.1. Elméleti áttekintés

A számítógépes látás egy interdiszciplináris tudományág, amely automatizálni kívánja a képek illetve a videókból való információ kinyerését. Az automatizálása ezen folyamatnak számítógépekkel történik, amelyek különböző algoritmusok segítségével nyerik ki a képekből a kívánt információt, mint például a Histogram of Oriented Gradients (HOG) [16], Scale-Invariant Feature Transform (SIFT) [50] vagy éppen konvolúciós neurális hálók általi [49] kinyerése a jellemzőknek. A képekből kinyert információknak egyik jelentős felhasználása a vizuális felismerése az objektumoknak. A vizuális felismerés többféle lehet az objektumok számát és azok felismerésének módszerét tekintve [70]:

- *Képosztályozás*, ahol egy kép teljes egészéről döntődik el, hogy melyik osztályba tartozik
- *Szegmentálás*, ahol több osztályt is reprezentálhat egy kép, és minden pixelhez hozzárendelődik egy osztály
- *Objektum detektálás*, amely hasonló a szegmentáláshoz, annyiban tér el, hogy nem rendelődik minden pixelhez osztály, hanem egy határolókeret (bounding box) által lokalizálódik minden felismerendő objektum.

Konvolúciós neurális hálók

A számítógépes látás téren a jelenlegi legelterjedtebb (vagy legkorszerűbb) módszert a konvolúciós neurális hálók képviselik [7]. A 2012-ben az AlexNet [43] által az ImageNet [17] adathalmazon elért kimagasló eredmények után váltak a számítógépes látás alapköveivé. A

konvolúciós neurális háló konvolúciós rétegekből áll, amelyek szerepe a különböző méretű és elhelyezkedésű jellemzők kinyerése konvolúció által az adott képből. A konvolúció a bemenet lokális régiói és egy adott szűrő skaláris szorzatát jelenti. A szűrők elemei ebben az esetben úgy kezelhetőek, mint a háló tanítható paraméterei, amelyek a teljesen összekötött háló esetében a súlyok. Mivel egy bemenethez csak egy szűrő van használva konvolúciónak, ezért a bemenet neuronjai megosztoznak ezeken a súlyokon, ezáltal a teljesen összekötött hálóhoz képest drasztikusan lecsökkentve a tanítható paraméterek számát. A konvolúciónak köszönhetően a háló rétegei a bemenet regionális területeinek relációit szűrik ki, ellentétben a teljesen összekötött hálókkal, ahol minden bemeneti neuron kapcsolódik minden kimeneti neuronhoz.

Objektum detektálás

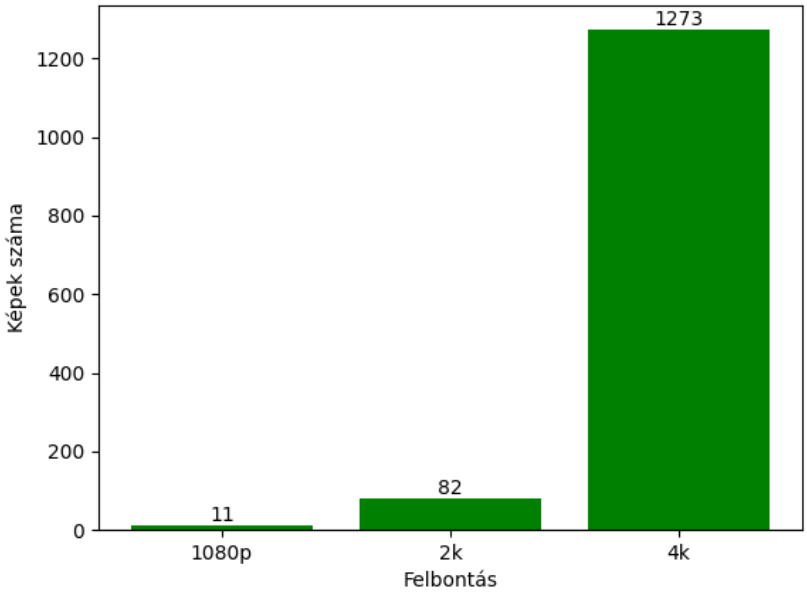
Az objektum detektálás a számítógépes látás azon szakterülete, amely a tárgyak lokalizálását és osztályozását ötvözi. Ezek lokalizálása határolókeretek (bounding box) segítségével történik, amelyek alakja egy téglalap. Ahhoz, hogy egy konvolúciós háló beazonosítsa a potenciális tárgyakat, többféle módszer létezik. Az objektum detektálást meg lehet közelíteni mint egy osztályozási feladat, ahol a határolókeretek és a hozzájuk tartozó osztályok külön, osztályozással határozódnak meg. Az ilyen típusú konvolúciós neurális háló architektúrák egyike a Region Proposal Network (RPN) [25] típusú hálók. A másik megközelítése a regresszió, ahol az adott architektúrában nincs különválasztva a tárgyak lokalizálása és osztályba sorolása, ilyenek például a You Only Look Once (YOLO) [63] és Single Shot Detector (SSD) [51] hálók. A két típusú architektúrának megvannak az előnyei és hátrányai. Mivel az RPN típusú hálók egy külön konvolúciós hálót használnak a határolókeretek kiszűrésére és osztályozására, ezért pontosabbaknak bizonyulnak a legtöbb esetben, mint a YOLO vagy SSD. Azonban a YOLO és SSD hálók tanítása és kiértékelése gyorsabb, mivel egy egységes hálót használnak.

Jelen cikkben kétfajta algoritmus van kipróbálva újrahasznosítási szimbólumok detektálására, egyik amely mesterséges neurális hálók segítségével lokalizálja és osztályozza az újrahasznosítási szimbólumokat, a másik egy Template Matching (Sablonillesztés)-nek nevezett módszer, amely nem használ mesterséges intelligencia alapú osztályozást.

4.2. Adathalmaz gyűjtése és előfeldolgozása

A felkutatott szakirodalomban és egyéb internetes forrásokban jelenleg nem található olyan publikus adathalmaz, amely az újrahasznosítási szimbólumokra fektetné a hangsúlyt. Ezen probléma miatt egy adathalmaz kialakítása és publikálása elősegíti az újrahasznosítási szimbólumok felismerésének problémáját. Az adatok OnePlus 6T, Huawei P40 Lite, OnePlus Nord és iPhone 7 mobiltelefonok kamerái segítségével vannak begyűjtve, manuálisan, természetes körülmények között. Az adathalmazban szereplő újrahasznosítási szimbólumok nagy része a szimbólumot képező háromszög avagy kör alakban egymásba mutató nyilak (Möbius-hurok) és a benne szereplő számból állnak, amely az Európai Unióban 1-98-ig terjed [5]. Tartalmaz azonban olyan szimbólumokat is, ahol nem található szám, ehelyett egy, a környezetvédelmi termékjelnek megfelelő rövidítés lelhető a Möbius-hurok környékén.

Az adathalmaz összesen 1366 felcímkézett képből áll, amelyeken a szereplő objektumok 57 osztályba sorolhatók az Európai Bizottság által definiált környezetvédelmi termékjelek alapján. A képek különböző felbontásúak ahogy az a 8. ábrán is látható és változó fénykörülmények között vannak készítve. Az osztályok eloszlását tekintve az adathalmaz osztály kiegyensúlyozatlansággal (class imbalance) rendelkezik, mivel egyes szimbólumokból



8. ábra. Az adathalmaz pixelszámban mért felbontásbeli eloszlása, ahol az 1080p az 1920x1080 felbontásnál nagyobb vagy egyenlő, de 2560x1440-nél kisebb képeket jelöli, a 2k a 2560x1440 felbontásnál nagyobb vagy egyenlő, de 4096x2160-nél kisebb képek, míg a 4k a 4096x2160 felbontásnál nagyobb vagy egyenlő képek.

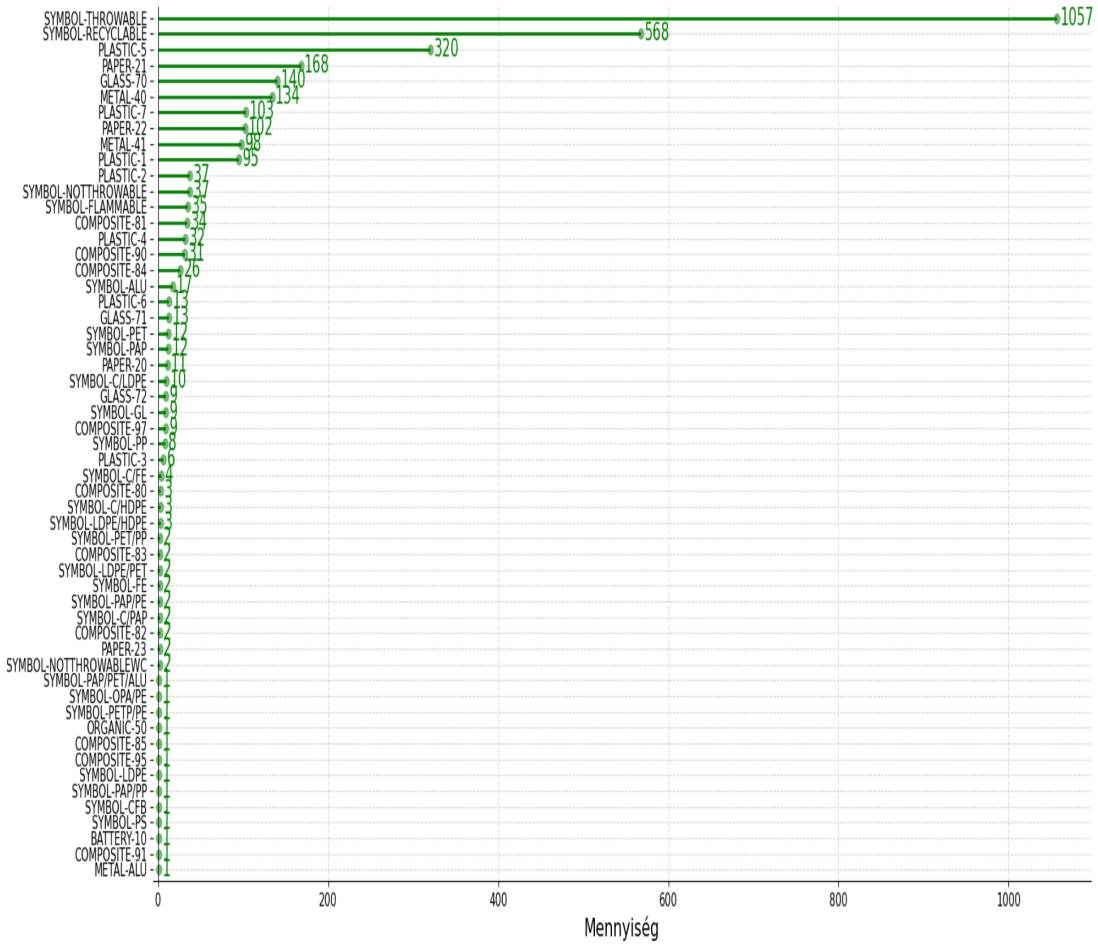
nagyságrendileg több található mint másokból.

A képek címkézése a **LabelImg** [1] nevezetű szoftverrel történt. Ennek segítségével, vizuális felületen, kurzorral történő négyzetek kijelölésével és az osztály megadása által történik a címkék létrehozása. Az objektumok kijelölése után minden egyes címkézett képfájlhoz egy *xml* kiterjesztésű címke fájl generálódik, amiben többek között a határolókeretek koordinátái és a hozzájuk tartozó osztály található meg. A többféle címkézési formátum közül, mint például a COCO [4], YOLO és Pascal VOC [21], az utóbbi van használva a címke fájlok kigenerálási formátumaként. A Pascal VOC formátum más, nagyobb adathalmaz kialakításához is volt használva, ezek közé tartozik az ImageNet. A formátum célja a standardizált adathalmazok kialakítása. Mivel a formátum kompatibilis a *TensorFlow* keretrendszerrel, megkönnyíti a tanítást és kiértékelést bármilyen Pascal VOC formátumban címkézett adathalmazon. A képek oly konvenció szerint lettek címkézve, hogy minden címke a legjobban reprezentálhassa az adott osztályt. Az olyan szimbólumok címkéje amelyek a Möbius-hurok belsejében nem tartalmaznak számot a *SYMBOL* előtaggal rendelkeznek, majd ezt követve egy kötőjellel, maga a szimbólum neve található. Azon szimbólumok amelyek a Möbius-hurok belsejében számot tartalmaznak, előtagként az anyag megnevezését és kötőjellel elválasztva a hozzájuk tartozó környezetvédelmi termékjel kódját kapják. Az adathalmazban szerepelnek olyan szimbólumok is, amelyeknek nincs hivatalosan meghatározott nevük, ezért ezeknek egyedi megnevezések kitalálására van szükség, ilyenek például a *SYMBOL-THROWABLE* és *SYMBOL-RECYCLABLE* annotációk.

Az adathalmaz osztályok közötti kiegyensúlyozatlansága annak köszönhető, hogy a két legnagyobb számossággal rendelkező osztály, amely az *újrahasznosítható* és *eldobható*, majdnem minden csomagoláson megtalálható (lásd 9. ábra). Ezenkívül a számosságban ezeket követő szimbólumok a műanyagot jelölők, amelyeknek nagy számuk egyszerűen annak tulajdonítható, hogy sokkal több termék csomagolása műanyag. Ebből következik az is, hogy az adathalmazban a kevés számban előforduló osztályokból kevesebb lelhető a bevásárlóközpontokban, üzletekben.

A kialakított adathalmazban szereplő minden egyes kép úgy van elforgatva, hogy a rajtuk levő szimbólumok álló pozícióban szerepeljenek szimbólumok rotációját tekintve. Az ilyen szintű egységesítéssel megkönnyíthető a későbbi tisztítása, augmentációja és különböző feladatokra való felhasználása az adathalmaznak.

Az adathalmazban szereplő szimbólumok a képek méretéhez viszonyítva kisméretűek. Az adathalmaz olyfajta átalakítása, amelyben a szimbólumok nagyobb méretűek a képhez



9. ábra. Az adathalmaz osztályok szerinti eloszlása, ahol egyes szimbólumok száma nagyban eltér a többitől. Bizonyos szimbólumok majdnem minden csomagoláson szerepelnek (SYMBOL-THROWABLE, SYMBOL-RECYCLABLE), míg egyesek ezzel ellentétben nagyon kevés csomagoláson vannak jelen (SYMBOL-PE, SYMBOL-CFB). Ez a tulajdonsága az adathalmaznak osztályok kiegyensúlyozatlanságához vezet.

viszonyítva, egyértelműen segítené ezek lokalizálását és osztályozását. A képekből e módszer szerint vannak kivágva részek, úgy, hogy kevesebb pixelterület maradjon a háttérből és ezáltal növelve a kép pixelterületének és szimbólumok pixelterületének arányát.

Az adathalmazban szereplő képek nagy része véletlenszerű Gauss zajt tartalmaz, amit a szimbólumok felismerése érdekében indokolt erre alkalmas szűrőkkel eltávolítani. Ebben az esetben többféle homályosító szűrő ki van próbálva, mint például a Gauss, Medián és Bilaterális szűrők. A felsorolt szűrők közül a Bilaterális szűrő bizonyul a leghatékonyabbnak abból a szempontból, hogy az éleket nem mossa el, ellentétben a Gauss és Medián szűrőkkel. Az élek

megmaradása és a zaj eltávolítása fontos a további képfeldolgozási módszerek alkalmazása és szimbólumok felismerése szempontjából. Továbbiakban több fajta kombináció van kipróbálva a képek feldolgozását illetően, ilyen a Canny éldetektáló [11], amely kiemeli az éleket. Azonban sok információ elveszik ami alapján a szimbólumok felismerése megtörténhetne. A végső képfeldolgozási módszer-kombináció, az alapján, hogy mennyi lényeges információt, zajt, illetve mennyi felesleges információt tartalmaz szimbólumok felismerését tekintően a kép feldolgozás után: Bilaterális szűrő, Adaptív Vágás (Adaptive Threshold), Black Hat morfológiai transzformáció. Az Adaptív Vágás használatának a célja az élek kiszűrése, a Black Hat transzformációnak a túlságosan fényes háttérben levő sötét élek/objektumok kiemelése.

4.3. Felhasznált technológiák és eszközök

A különböző programok elkészítése és felhasználása a **Python** programozási nyelv 3.8.5-ös verziójával történik. A képek feldolgozására, illetve képek manipulációjához az **OpenCV** (Open Source Computer Vision Library) [33] könyvtárcsomag 4.5.1.48-as verziója van használva. Az *OpenCV* egy nyílt forráskódú könyvtárcsomag, amelynek fő felhasználása a számítógépes látás és gépi tanulás terén van.

Az előre a Common Objects in Context (COCO) [48] adathalmazon betanított modellek beszerzése, ezek tanítása és kiértékelése a **TensorFlow 2** könyvtárcsomag és a hozzátartozó Object Detection API-jának segítségével történik. A modellek tanítása előtt egy egységes formátumba kell alakítani a kép és címke párokat, amelyek a *TensorFlownak* a TFRecord kiterjesztésű fájljait takarják jelen esetben. A TFRecord a *TensorFlow* fájlformátuma, amely megkönnyíti az adathalmazokkal való műveleteket és gyorsítja azokat. A TFRecord fájlokban binárisan vannak eltárolva az adatok, ezáltal kisebb helyet foglalnak el és emiatt sokkal gyorsabb ezen fájlok memóriából való olvasása is.

Az Object Detection API megkönnyíti a detektorok tanítását illetve kiértékelését azáltal, hogy elégséges beállítani a felhasznált modelleket illetve a hozzátartozó konfigurációkat egy erre szánt fájlban az egyéni detektor elkészítéséhez. A modellek kiexportálása is több, az API-ban található szkriptek által történik.

A TensorFlow Lite [34] a *TensorFlow* mobiltelefonokra, beágyazott és IoT (Internet of Things) eszközökre szánt kiterjesztése, amely segítséget nyújt többek között modellek átkonvertálásában és futtatásában. Ahhoz, hogy mobiltelefonon futtatható legyen a modell, a SavedModel [35] formátumba kiexportált modelleket át kell konvertálni TFLite formátumba,

amely már futtatható telefonos felületen. A SavedModel formátumba konvertált modellek tartalmazzák a tanításhoz használt *TensorFlow* programot, a tanított paramétereket és magát a modell architektúráját, amelyet emiatt nem kell külön felépíteni kód szintjén. A TFLite formátum által nem csak futtatható lesz telefonon az adott modell, hanem kompaktabb lesz méretét tekintve és akár kiértékelési idő csökkenés is megfigyelhető.

A modell metrikáinak vizualizálása a *TensorFlow* könyvtárcsomag TensorBoard vizualizációs eszköztárával és a **Matplotlib** könyvtárcsomag segítségével valósul meg. A metrikák megjelenítése a TensorBoard webes felületén történik, ahol a vizualizációkat könnyedén kilehet exportálni SVG formátumba. A TensorBoard által megjelenített vizualizációk nem nyújtanak elég szabadságot, ezért jelen cikkben a *Matplotlib* segítségével elkészített vizualizációk találhatóak meg.

A tanításhoz használt eszköz a Codespring cég által van biztosítva, amely egy GeForce GTX 1080-as videokártyával, Intel(R) Core(TM) i5-6400 @ 2.70GHz processzorral és 64 GB RAM-mal történik. A tanítás **Docker** környezetben zajlik videókártya felhasználásával. Ezáltal az eredmények reprodukálása könnyedén megtörténhet, ezenkívül az esetleges kitelepítés és hordozhatóság is fontos szempontok a használatában.

4.4. Kipróbált módszerek

Az újrahasznosítási szimbólumok felismerése nem egy triviális feladat a 4.2 szekcióban felsorolt tényezők miatt, ezért több módszer is ki van próbálva és végül a legigéretesebb módszer van kiválasztva részletesebb tanulmányozás szempontjából. A módszer kiválasztását nem csak az általa kínált újrahasznosítási szimbólumok osztályozásának pontossága befolyásolja, hanem annak gyorsasága is egy fontos tényező. A felismerés telefonon is futtatható kell legyen valós időben, ezért számítani kell a kisebb teljesítményű eszközökre is, hogy rövid időn belül megtörténjen a felismerés.

Optikai karakterfelismerés

Az optikai karakterfelismerés (OKF) [19] a számítógépes látás egyik alterülete, amely a kézzel írott vagy nyomtatott karakterek számítógép általi felismerésével foglalkozik. Az OKF több részre osztható a karakterek felismerésének módszerét tekintve. Az egyik ilyen módszer a klasszikus számítógépes látásban használt technikákat veszi alapul, mint például szűrők használata a karakterek éleinek kiszűrésére, a karakterek élek általi szegmentálása és

a szegmentált karakterek osztályozása. Más módszerek a mélytanulás alapjain fekszenek. Az egyik mélytanulási megközelítése ennek a karakterek detektálása és osztályozása detektor által [44]. Egy másik mélytanulási megközelítés a rekurrens neurális hálók általi [66] osztályozást veszi igénybe.

Egy kipróbált módszer a rekurrens neurális hálók általi OKF amely az *OpenCV* könyvtárcsomagban található Tesseract [36] optikai karakterfelismerő segítségével történik. A Tesseract 4.0 egy továbbfejlesztett rekurrens neurális hálót, Long Short Term Memory (LSTM)-t [14] használ karakterek felismerésére. Az optikai karakterfelismerője nagyon érzékeny a képeken levő zajra, illetve a nem tisztán kivehető karakterekre, ezért alapos előfeldolgozást igényelnek a képek. Ezen képek előfeldolgozása aszerint történik, hogy a felsorolt zavaró tényezők minimálisak legyenek, emellett nagyban segíti a hálót az is, ha valamilyen éldetektáló algoritmus van alkalmazva a képeken mielőtt osztályozásra lennének átadva.

Osztályozás előtt ki vannak vágva a képek azon részei, amelyen csak az újrahasznosítási szimbólumok láthatóak. A kivágást követően a képek feldolgozása következik, ahol nem sikerül egy olyan feldolgozási módszert találni, amely hatékonyan működne az adathalmaz összes elemére. A 61 kivágott kép alapos feldolgozása ellenére a Tesseract 9.83%-os pontossággal osztályozza azokat. Az eredmények fényében a rekurrens neurális hálók általi osztályozása az újrahasznosítási szimbólumoknak elvetésre kerül.

Template Matching

A képeken levő objektumok keresésének az egyik legegyszerűbb módja a Template Matchingnek nevezett módszer [9]. A Template Matching az *OpenCV*-be épített függvények [58] által van elvégezve. A módszer lényege, hogy egy template-nek nevezett képet csúsztatnak végig azon a szürke vagy egy színcsatornás képen ahol az objektumot keresik (forráskép). A forrásképen minden egyes pontban kiszámolódik a template és az általa a forrásképen lefedett terület pixeljei között egy korreláció, amely többféle lehet és egyik változata a következőképpen számítható ki:

$$cor = \frac{\sum_{i=0}^{N-1}(x_i - \overline{x}) \cdot (y_i - \overline{y})}{\sqrt{\sum_{i=0}^{N-1}(x_i - \overline{x})^2 \cdot \sum_{i=0}^{N-1}(y_i - \overline{y})^2}}$$

ahol x a template kép pixelei, $\overline{x}$ a template kép pixelintenzitásainak az átlaga, y a forráskép

pixelei, $\overline{y}$ a forráskép pixelintenzitásainak az átlaga, N a template képen található pixelek száma. A korreláció minden egyes ponton egy -1 és +1 között mozgó érték, amely a template kép és forráskép adott pixelközépponton levő hasonlóságát írja le.

A Template Matchingnek több hátránya is van működéséből kifolyólag, az egyik hátrány, hogy nagy számítási kapacitás szükséges a minta és forráskép közötti korreláció kiszámításához a forráskép minden egyes pontjában, amely azt jelentené, hogy mobiltelefonon nagyon lassú lenne. Egy másik hátránya, hogy nagyon érzékeny kis változásokra a forrás képen, mint például fényviszony változása és objektumok más-más szögekben való megjelenése.

A felsorolt hátrányok és az ebből következő eredmények miatt nem bizonyul egy általános megoldásnak az újrahasznosítási szimbólumok felismerésére a Template Matching.

Objektum detektálás

A Single Shot típusú detektorok széleskörű felhasználása az objektum detektálás terén leginkább két tényezőnek köszönhető, az egyik, hogy gyors kiértékelési idővel rendelkeznek és a másik, hogy a gyors kiértékelés ellenére pontosság szempontjából is jól teljesítenek. A kiértékelési idő nagyban elősegíti azt a felhasználását a hálónak, hogy mobiltelefonon futtatható legyen elfogadható idő alatt.

A Single Shot architektúra (lásd 11. ábra[2]) két részre bontható, amelyek az úgynevezett gerinc modell és a Single Shot Detektor feje. A gerinc modell egy konvolúciós neurális háló, amelynek célja a bemeneti kép fontosabb jellemzőinek kivonása amelyet továbbít a fej résznek és egy osztályozó rétegnek. A fej, azaz az SSD rétegek ugyancsak konvolúciós rétegek, amelyeknek feladata a határolókeretek többféle méretben való kinyerése és minden határolókerethez egy osztály rendelése, a fejben található rétegek ugyancsak egy osztályozóhoz kapcsolódnak, mivel többféle méretű jellemző, avagy határolókeret osztályozásra kerül.

Mivel az adathalmazban található képeken a szimbólumok többféle méretben is megtalálhatóak, ezért is lehet kedvező az SSD architektúra választása a szimbólumok detektálására.

4.5. A tanítás részletei

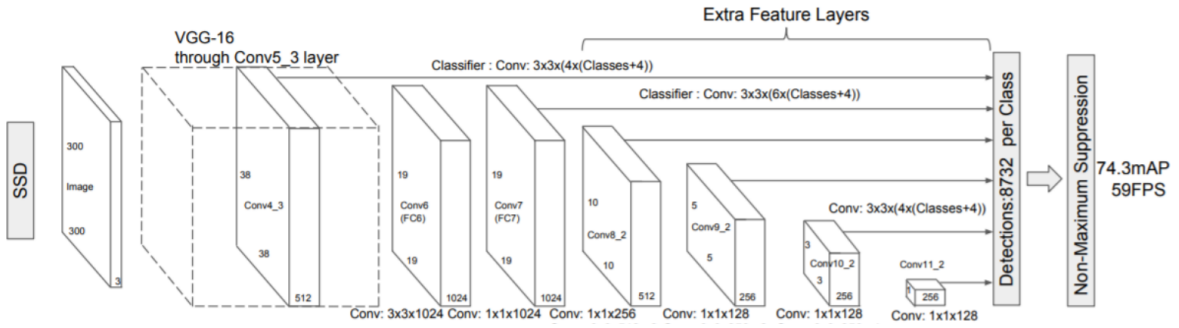
A mélytanulási detektor modellek tanítása nagy számítási kapacitást igényel, ezért a felhasznált videokártyával egyetlen tanítás 10-25 órát is eltarthat. Emiatt csak egyetlen

[2]forrás: https://arxiv.org/pdf/1512.02325.pdf

architektúrájú modellt sikerült részletesebben, többféle hiperparaméterrel kipróbálni. Főként a SSD MobileNet V2 FPNLite 640x640 detektor eredményei vannak a jelen cikkben kifejtve, amely egy Single Shot típusú detektor, melynek a jellemző kinyerője a MobileNet V2 [65]. Ez egy elterjedt architektúra, mivel kedvező kiértékelési idővel és pontossággal rendelkezik.

A tanítás előtt a képeket és a hozzájuk tartozó címkéket TFRecord kiterjesztésű fájlba kell csoportosítani, mivel a tanítás csak ilyen formában lehetséges az Object Detection API által. A TFRecordba való alakítása a képeknek és címkéknek egy publikusan elérhető szkript által történik [73], amelynek bemenete a címkék és képek helye a lemezen és kimenete egy TFRecord kiterjesztésű fájl. A tanítással és kiértékeléssel kapcsolatos beállításokat és hiperparamétereket az Object Detection API egy konfigurációs fájlban fogadja el. A bemeneti képek mérettől függetlenül 640x640 méretre vannak átalakítva tanítás előtt, hogy megegyezzenek a MobileNet V2 bemeneti rétegének dimenziójával.

A hiperparaméterek megválasztása pszeudo véletlen módon történik, aszerint, hogy egyes paraméterek milyen irányban és mértékben befolyásolják a modellek kiértékelési metrikáit. Ezek közül a fontosabbak, amelyek szerint optimalizálva vannak a betanított modellek, az a tanítási együttható, ami egy koszinusz alapú tanítási együttható csökkentővel [67] minden lépésben csökkentve van; és a regularizáció ami a tanítás ideje alatt konstans. Minden modell 8-as batch mérettel van tanítva, ami azt jelenti, hogy a háló paraméterei minden lépésben egyszerre nyolc kép alapján vannak állítva. Kétféle optimalizálóval történik kísérletezés a modellek tanítását illetően, ezek a Sztochasztikus Gradiens Süllyedés Momentummal (SGDM) [52] és az Adam [41] optimalizálók. A lokalizációs veszteségnek a Súlyozott Huber veszteségfüggvény van használva, ami specifikusan a Single Shot detektorok körében elterjedt



10. ábra. Az SSD architektúra, ahol látható bizonyos szinten található konvolúciós rétegek osztályozó rétegbe való csatolása, ami által a határolókeretek és a hozzájuk tartozó osztályok meghatározása történik. A több konvolúciós réteg szintjén osztályozó rétegbe való csatolásnak köszönhetően többféle méretű objektum detektálására is alkalmas.

[79], mivel egyensúlyban tartja az osztályozási és lokalizációs veszteséget, nem engedve, hogy a gradiens csakis az osztályozás fele vigye el a tanulást, ami által a modell gyenge teljesítményt nyújtana lokalizáció terén. Osztályozási veszteségnek Súlyozott Szigmoid Fokális (Weighted Sigmoid Focal) veszteség van használva, amelyet az osztály kiegyensúlyozatlanság típusú problémák kezelésére használnak [47], mivel kisebb súlyt társít jól osztályozott példányokhoz mint az olyan példányokhoz amelyek osztályozása nehezebb avagy kisebb pontossággal történik. A totális veszteség a lokalizációs és osztályozási veszteség súlyozott összegeként van kiszámolva, ahol a súly mindkét veszteségnek egyenlő.

Az adathalmaz 80 százaléka van felhasználva a modellek tanítására, illetve 20 százaléka a kiértékelésükre. A tanított modellek mindegyike transzfer tanulás segítségével van továbbtanítva a COCO adathalmazról. A transzfer tanulás segítségével kevesebb lépés alatt elkezdődik a generalizációja a modelleknek, ezáltal kevesebb lépést igényelnek a tanítások. A modellek tanítva vannak az adathalmaz augmentált és nem augmentált változatával is. A bővítése a képeknek kétféleképpen van elvégezve. Az egyik módszer az Object Detection API által történik, ahol a korábban említett konfigurációs fájlban meghatározandó opciók szerint végződik a képek sokszorosítása. A konfigurációs fájlban megtalálható augmentációk a következők: véletlenszerű 90 fokos forgatás, véletlenszerű kontraszt és fényerő állítás, véletlenszerű vágás a kép egy részéből, véletlenszerű horizontális és vertikális tükrözés. Egy másik módszer azért indokolt, mivel az Object Detection API nem rendelkezik más, fontos bővítési technikákkal. Ezt akkor kell elvégezni, amikor még nincsenek TFRecord formátumba alakítva a felhasznált adathalmazban található képek. Az ehhez a módszerhez tartozó augmentációk a következők: skálázás, transzláció, gamma és szigmoid kontraszt.

4.6. Tanítási eredmények

A mélytanulási modellek teljesítménye a COCO Detection Challenge által meghatározott metrikák [4] alapján vannak mérve, amelyek közül kiemelendő: Mean Average Precision (mAP) - Átlagos Középérték Precizitás, Mean Average Recall (mAR) - Átlagos Középérték Visszahívás. A COCO Detection Challenge által meghatározott metrikákon kívül a veszteség is figyelembe van véve a modellek teljesítményét tekintően. Az mAP értéke a modell azon tulajdonságát tükrözi, hogy csak a releváns objektumokat ismeri fel és háromféle IoU (Intersection over Union) értéken számolódik ki, ezért háromféle IoU szerint vannak a mérések elvégezve: IoU=.50:.05:.95, ahol 50%-tól ötös lépésekkel egészen 95%-ig az mAP érték ezen

IoU értékek AP (Average Precision) átlaga lesz, IoU=.50, ahol a minimum IoU érték 50% kell legyen a határolókeretek lokalizálását tekintően és IoU=.75, ahol az IoU érték minimum 75% kell legyen, hogy a detektált határolókeret figyelembe legyen véve az mAP számításánál. Az mAP számítása ezenkívül az objektumok nagyságán is eloszlik, ugyancsak három részre: Kis objektumokra számolt mAP, amelyeknek területe kisebb mint 32^2, közepes objektumokra, amelyeknek területe nagyobb mint 32^2 és kisebb mint 96^2, a nagy objektumok területe pedig nagyobb mint 96^2. Mivel az adathalmazban csak nagy és közepes objektumok szerepelnek, ezért jelen esetben csak a nagy és közepes objektumok mAP értéke van figyelembe véve kiértékelésnél.

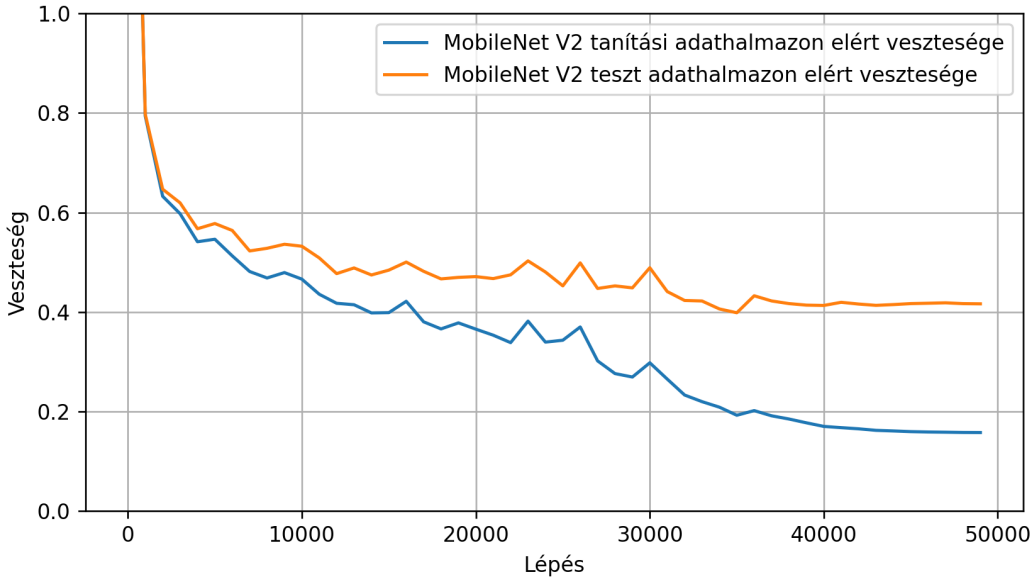
Az mAR érték jelentése, hogy a határolókeretek közül hány volt helyesen detektálva. Ezen érték kiszámolása megtörténik a detektálások számának figyelembe vételével és ahogy az mAP esetében is, az objektumok nagysága itt is három részre osztja az mAR kiszámolását. A detektálások számát illetően három részre osztható az mAR: 1, 10 és 100, ahol ezek a számok a detektálások számát jelölik és maximum ennyi detektálásra átlagolódik egy képen az AR(Average Recall) érték.

A cikkben megjelenő eredmények IoU=.50:.05:.95 és tárgyak számát tekintve 100 detektálást vesznek figyelembe az mAP és mAR értékek kiszámolásához.

Az osztály kiegyensúlyozatlanság jelenségnek köszönhetően a tanított modellek a túltanulás (overfit) jeleit mutatják, mivel a 11. ábrán is látható, hogy a 36.000 lépéstől kezdődően a teszt adathalmazon mért osztályozási veszteségtől elhatárolódik a tanító adathalmazon mért osztályozási veszteség, azáltal, hogy a teszt veszteség ettől a lépéstől kezdődően már nem csökken.

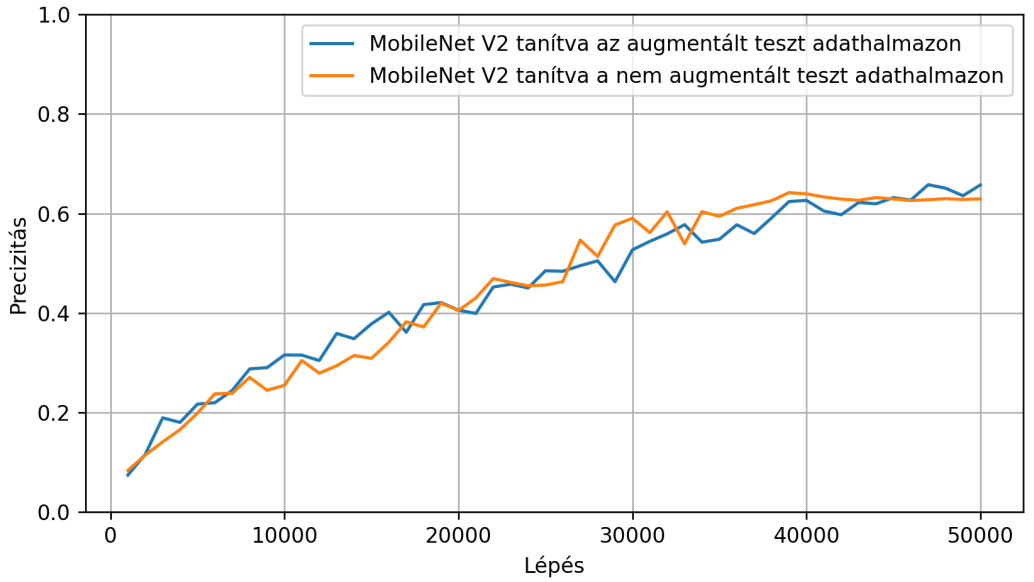
Az osztály kiegyensúlyozatlanság befolyásának gyengítésére használt augmentálás 2.54%-al növelte a teszt adathalmazon elért mAP-t (lásd 12. ábra), illetve 3.91%-al csökkentette a tanítási adathalmazon mért mAP-t.

Az adathalmaz módosítását követően, ahol azon osztályok amelyek számossága nem éri el a 10-et és meghaladja az 500-at eltávolításra kerülnek, ahogy a 13. ábrán is látható, szignifikánsabb javulás érhető el mAR metrikát tekintve. Meghagyva az 500-nál nagyobb számossággal rendelkező két osztályt, mAP-ben való növekedés és mAR-ben való csökkenés történik. Ez az összefüggés arra enged következtetni, hogy a két legnagyobb számossággal rendelkező osztályt a modell túlságosan megtanulja, ezáltal háttérbe szorítva a kisebb számossággal rendelkezőket.

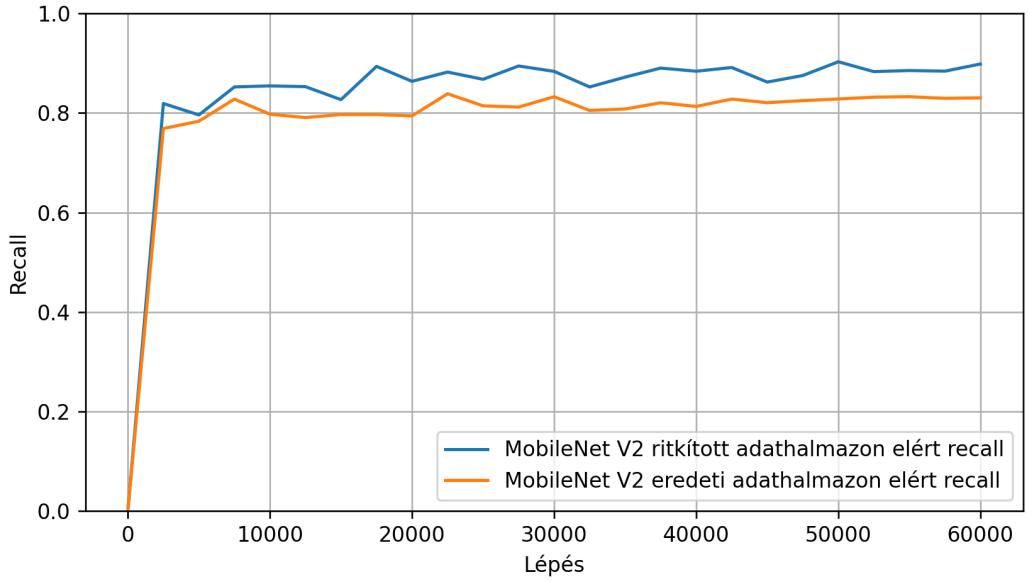


11. ábra. A MobileNet V2 tanítási és teszt halmazon elért veszteségek egyike. A vízszintes tengelyen a lépések száma, míg a függőleges tengelyen az összesített veszteség értéke látható.

Az SGDM optimalizálóval 21 tanítás ment végbe, amelyből 5 tanítás a ritkított adathalmazon történt. A legjobb eredményt az eredeti adathalmazon 0.05399 tanulási együtthatóval, 5.999e-5 regularizációval, 60.000 lépés alatt sikerült elérni, ahol az mAP értéke a teszt adathalmazon 71.29%, míg az mAR értéke 83.05%. A kipróbált legjobb paraméterekkel

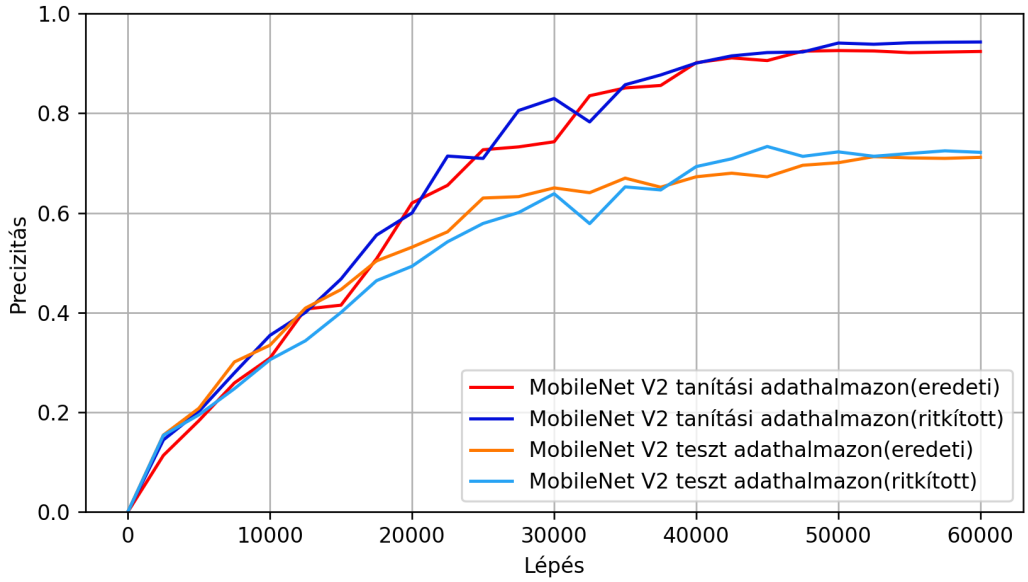


12. ábra. A MobileNet V2 tanítva augmentált és nem augmentált adathalmazon. Látható, ahogy az adathalmaz bővítését követően precizitásbeli növekedés történik.



13. ábra. A recall növekedése osztályok eltávolításával. A ritkított adathalmazon elért mAR maximális értéke 90.3%, míg az eredeti adathalmazon elért mAR maximális értéke 83.8%. A mérések mindkét esetben a teszt adathalmazon történtek.

is csak minimalizálni sikerült a túltanulás mértéket, eltüntetni nem. A ritkított adathalmazon a legjobb eredményt ugyanazokkal a hiperparaméterekkel sikerült elérni mint az eredeti adathalmazon. A teszt adathalmazon 72.16% mAP-t és 88.00% mAR-t sikerült elérnie a



14. ábra. A ritkított és az eredeti adathalmazon történt kiértékelések mAP-ben. A kiértékelések a teszt és tanítási adathalmazon történtek. Látható, ahogy a kiegyensúlyozatlan adathalmazon kissebb a túltanulás mértéke.

modellnek, azonban a túltanulás mértéke nagyobb mint az eredeti adathalmazon, ahogy az a 14. ábrán is látható. Ez annak is tulajdonítható, hogy a ritkított adathalmazon nem volt elég tanítás a megfelelő hiperparaméterek megtalálásában.

Az Adam optimalizálóval 7 tanítás történt a nem ritkított adathalmazon. Az SGDM-hez képest, ahhoz, hogy generalizálódni tudjon a modell, jóval kisebb tanulási együttható megválasztására van szükség, ezért a legjobb eredményt 0.003999 tanulási együtthatóval, az SGDM-el is használt regularizációs együtthatóval, 60000 lépés alatt érte el a tanított modell. Az így kapott legmagasabb mAP érték 69.41% lett, amely társul egy 83.63%-os mAR értékkel.

Következtetések és továbbfejlesztési lehetőségek

A Greensy alkalmazásrendszer fejlesztése során elkeszült egy iOS kliensalkalmazás, amely a felhasználókat a szelektív hulladékgyűjtés folyamatában segíti. A megvalósult főbb funkcionalitások a következők: a gyűjtőközpontok megtekintése térképen és listanézetben, a környezetvédelmi termékjelekkel kapcsolatos információk böngészése egy wiki használatával és nem utolsó sorban a környezetvédelmi termékjelek felismerése a felhasználó által készített fénykép segítségével.

Az összegyűjtött adathalmaz 1366 képből áll, amely egy jó kiindulópontot nyújtott az újrahasznosítási szimbólumok felismerését tekintően. Az adathalmazban szereplő szimbólumok felismeréséhez használt módszerek közül (OCR, Template Matching, detektor) a leghatékonyabbnak a detektorok bizonyultak. A detektorok finomhangolását több optimalizálóval is sikerült kipróbálni, melyek közül a leghatékonyabbnak az SGDM bizonyult, mind a ritkított, mind pedig az eredeti adathalmazon. Az eredeti adathalmazon 71.29% mAP, és 83.05% mAR, míg a ritkítottan 72.16% mAP és 88.00% mAR értékeket sikerült elérni a teszt adathalmazon.

Az iOS alkalmazás funkcionalitások szempontjából még nem teljes, ennek következtében pedig sok továbbfejlesztési lehetőség merül fel jövőbeli tervek között:

- a felhasználó menedzsment megvalósítása, amely által a regisztrált és bejelentkezett felhasználók számára további funkcionalitások érhetőek el;
- a gyűjtőközpontok fele intézett hulladék elszállítások, valamint az azokhoz tartozó programálások kezelése és megjelenítése egy naptár nézetben, értesítések küldése a programálásokkal kapcsolatosan;
- egy „Home” nézet létrehozása, ahol a felhasználó újrahasznosításhoz való hozzájárulási statisztikái jelennének meg;
- főként a gyűjtőközpontok számára egy egységes felület fejelsztése egy webalkalmazás keretében, ahol a különböző cégek bevezethetnék adataikat a rendszerbe;
- a környezetvédelmi termékjelek kibővítése amerikai, illetve kínai termékjelekkel;
- egy kredit rendszer bevezetése, amely a felhasználó számára az újrahasznosított termékek mennyiségétől függően pontszámokat írna jóvá a rendszerben, később ezeket a pontszámokat a felhasználó tárgyi értékekre válthatná jutalomként.

A szimbólumok felismerésének megbízhatóságát is javítani kellene a jövőben. Ennek eléréséhez többféle ötlet merült fel:

- a konfuziós mátrix és legjobb k statisztikák elkészítése a modell teljesítményének jobb megértésére;
- egy nagyobb architektúrájú háló kipróbálása, amely egy API-n keresztül kommunikálna a mobilos alkalmazással;
- az adathalmaz bővítése, amely a Unity játékmotor segítségével szintetikus adatok generálása által [76] vagy GAN (General Adversarial Network)[26] segítségével is megtörténhet;
- az adathalmazban szereplő osztályok számosságuk szerint csoportokra osztása és minden egyes csoporthoz egy külön detektor rendelése.

Hivatkozások

[1] 2021. URL: https://github.com/tzutalin/labelImg (elérés dátuma: 2021. ápr. 28.)

[2] URL: https://rss.org.uk/news-publication/news-publications/2018/general-news/statistics-of-the-year-2018-winners-announced/ (elérés dátuma: 2021. ápr. 29.)

[3] URL: https://ambasadasustenabilitatii.ro/romania-are-a-doua-cea-mai-slaba-rata-de-reciclare-a-deseurilor-din-ue/ (elérés dátuma: 2021. máj. 4.)

[4] URL: https://cocodataset.org/#format-data (elérés dátuma: 2021. máj. 3.)

[5] *A BIZOTTSÁG HATÁROZATA a csomagolóanyagok azonosítási rendszerének a csomagolásról és a csomagolási hulladékról szóló, 94/62/EK európai parlamenti és tanácsi irányelv értelmében történő meghatározásáról*. 1997. URL: https://eur-lex.europa.eu/legal-content/HU/TXT/PDF/?uri=CELEX:31997D0129&from=EN (elérés dátuma: 2021. ápr. 27.)

[6] Asociația pentru Dezvoltarea Comunității Rurale ADCR. *InfoDeseuri*. URL: https://play.google.com/store/apps/details?id=ro.infodeseuri.devapp (elérés dátuma: 2021. márc. 26.)

[7] Md Zahangir Alom és mások. „A state-of-the-art survey on deep learning theory and architectures”. *Electronics* 8.3 (2019), 292. oldal.

[8] Oren Ben-Kiki, Clark Evans és Brian Ingerson. „Yaml ain’t markup language (yaml™) version 1.1”. *Working Draft 2008-05* 11 (2009).

[9] Roberto Brunelli. *Template matching techniques in computer vision: theory and practice*. John Wiley & Sons, 2009.

[10] Charles F Cadieu és mások. „Deep neural networks rival the representation of primate IT cortex for core visual object recognition”. *PLoS Comput Biol* 10.12 (2014), e1003963.

[11] John Canny. „A computational approach to edge detection”. *IEEE Transactions on pattern analysis and machine intelligence* 6 (1986), 679–698. oldal.

[12] Chran-Ham Chang és mások. „High-performance TCP/IP and UDP/IP networking in DEC OSF/1 for Alpha AXP”. *Proceedings of 3rd IEEE International Symposium on High Performance Distributed Computing*. IEEE. 1994, 35–42. oldal.

[13] Kristina Chodorow. *MongoDB: the definitive guide: powerful and scalable data storage*. " O’Reilly Media, Inc.", 2013.

[14] Junyoung Chung és mások. „Empirical evaluation of gated recurrent neural networks on sequence modeling”. *arXiv preprint arXiv:1412.3555* (2014).

[15] Johan Cwiklinski. *git-flow Documentation*. URL: https://readthedocs.org/projects/git-flow/downloads/pdf/latest/ (elérés dátuma: 2021. ápr. 25.)

[16] 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.

[17] 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.

[18] Inc. Earth911. *iRecycle*. 2009. URL: https://earth911.com/irecycle/ (elérés dátuma: 2019. júl. 20.)

[19] Line Eikvil. „Optical character recognition”. *citeseer. ist. psu. edu/142042. html* 26 (1993).

[20] Inc. Electronic Recyclers International. *RecycleNation*. 2020. URL: https://recyclenation.com/ (elérés dátuma: 2021. ápr. 12.)

[21] M. Everingham és mások. „The Pascal Visual Object Classes (VOC) Challenge”. *International Journal of Computer Vision* 88.2 (2010. jún.), 303–338. oldal.

[22] Jesse Feiler. „Using CocoaPods”. *Beginning Reactive Programming with Swift*. Springer, 2018, 17–27. oldal.

[23] Roy Fielding és mások. *Hypertext transfer protocol–HTTP/1.1*. 1999.

[24] Plastic Soup Foundation. *GREENPEACE USA: RECYCLING SYMBOLS ON PLASTIC PACKAGING ARE MISLEADING*. 2020. URL: https://www.plasticsoupfoundation.org/en/2020/04/greenpeace-usa-recycling-symbols-on-plastic-packaging-are-misleading/ (elérés dátuma: 2021. máj. 2.)

[25] Ross Girshick és mások. „Rich feature hierarchies for accurate object detection and semantic segmentation”. *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, 580–587. oldal.

[26] Ian J Goodfellow és mások. „Generative adversarial networks”. *arXiv preprint arXiv:1406.2661* (2014).

[27] James Goodwill és Wesley Matlock. „The swift programming language”. *Beginning Swift Games Development for iOS*. Springer, 2015, 219–244. oldal.

[28] Arjun Guha, Claudiu Saftoiu és Shriram Krishnamurthi. „The essence of JavaScript”. *European conference on Object-oriented programming*. Springer. 2010, 126–150. oldal.

[29] Cornelia Győrödi és mások. „A comparative study: MongoDB vs. MySQL”. *2015 13th International Conference on Engineering of Modern Electric Systems (EMES)*. IEEE. 2015, 1–6. oldal.

[30] Evan Hahn. *Express in Action: Writing, building, and testing Node.js applications*. Manning Publications, 2016, 31–49. oldal.

[31] *Hivatalos Git dokumentáció:* URL: https://git-scm.com/doc (elérés dátuma: 2021. ápr. 25.)

[32] *Hivatalos GitLab dokumentáció:* URL: https://about.gitlab.com/stages-devops-lifecycle/continuous-integration/ (elérés dátuma: 2021. ápr. 25.)

[33] *Hivatalos OpenCV dokumentáció:* URL: https://docs.opencv.org/4.5.1/ (elérés dátuma: 2021. ápr. 28.)

[34] *Hivatalos Tensorflow Lite dokumentáció:* URL: https://www.tensorflow.org/lite (elérés dátuma: 2021. ápr. 28.)

[35] *Hivatalos Tensorflow SavedModel dokumentáció:* URL: https://www.tensorflow.org/guide/saved_model (elérés dátuma: 2021. ápr. 28.)

[36] *Hivatalos Tesseract dokumentáció:* 2021. URL: https://github.com/tesseract-ocr/tessdoc (elérés dátuma: 2021. ápr. 28.)

[37] Simon Holmes. *Mongoose for Application Development*. Packt Publishing Ltd, 2013.

[38] Simon Josefsson és mások. *The base16, base32, and base64 data encodings*. Technikai jelentés. RFC 4648, October, 2006.

[39] Dawid Karczewski. *Swift vs Objective-C: Which Should You Pick For Your Next iOS Mobile App?* 2020. URL: https://www.ideamotive.co/blog/swift-vs-objective-c-which-should-you-pick-for-your-next-ios-mobile-app (elérés dátuma: 2021. ápr. 25.)

[40] Maurice Kelly és Joshua Nozzi. *Mastering Xcode: Develop and Design*. Peachpit Press, 2013.

[41] Diederik P Kingma és Jimmy Ba. „Adam: A method for stochastic optimization”. *arXiv preprint arXiv:1412.6980* (2014).

[42] Kitti Klinbua és Wiwat Vatanawood. „Translating tosca into docker-compose yaml file using antlr”. *2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)*. IEEE. 2017, 145–148. oldal.

[43] Alex Krizhevsky, Ilya Sutskever és Geoffrey E Hinton. „Imagenet classification with deep convolutional neural networks”. *Advances in neural information processing systems* 25 (2012), 1097–1105. oldal.

[44] Huseyin Kusetogullari és mások. „DIGITNET: A Deep Handwritten Digit Detection and Recognition Method Using a New Historical Handwritten Digit Dataset”. *Big Data Research* 23 (2021), 100182. oldal.

[45] Keith Lee. „Using LLDB”. *Pro Objective-C*. Springer, 2013, 415–434. oldal.

[46] Rory Lewis, Yulia McCarthy és Stephen M Moraco. *Beginning IOS Storyboarding: Using Xcode*. Apress, 2012.

[47] Tsung-Yi Lin és mások. „Focal loss for dense object detection”. *Proceedings of the IEEE international conference on computer vision*. 2017, 2980–2988. oldal.

[48] Tsung-Yi Lin és mások. „Microsoft coco: Common objects in context”. *European conference on computer vision*. Springer. 2014, 740–755. oldal.

[49] Yi-zhou Lin, Zhen-hua Nie és Hong-wei Ma. „Structural damage detection with automatic feature-extraction through deep learning”. *Computer-Aided Civil and Infrastructure Engineering* 32.12 (2017), 1025–1046. oldal.

[50] Tony Lindeberg. „Scale invariant feature transform”. (2012).

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

[52] Yanli Liu, Yuan Gao és Wotao Yin. „An improved analysis of stochastic gradient descent with momentum”. *arXiv preprint arXiv:2007.07989* (2020).

[53] Adaptation Environmental Support Pty Ltd. *Recycle Mate*. 2019. URL: https://play.google.com/store/apps/details?id=com.recycle_mate&hl=en&gl=US (elérés dátuma: 2021. ápr. 12.)

[54] John A McCarty és LJ Shrum. „The recycling of solid wastes: Personal values, value orientations, and attitudes about recycling as antecedents of recycling behavior”. *Journal of business research* 30.1 (1994), 53–62. oldal.

[55] Bob McCune. *Learning AV Foundation: A Hands-on Guide to Mastering the AV Foundation Framework*. Addison-Wesley Professional, 2014.

[56] Adrian Mouat. *Using Docker: Developing and Deploying Software with Containers*. " O'Reilly Media, Inc.", 2015.

[57] Shyue Ping Ong és mások. „The Materials Application Programming Interface (API): A simple, flexible and efficient API for materials data based on REpresentational State Transfer (REST) principles”. *Computational Materials Science* 97 (2015), 209–215. oldal.

[58] *Opencv Template Matching dokumentáció:* 2021. URL: https://docs.opencv.org/3.4/df/dfb/group__imgproc__object.html#ga586ebfb0a7fb604b35a23d85391329be (elérés dátuma: 2021. ápr. 28.)

[59] Eduardo Eloy Loza Pacheco és mások. „Swift UI and Their Integration to MapKit Technology as a Framework for Representing Spatial Information in Mobile Applications". *GIS LATAM Conference*. Springer. 2020, 80–91. oldal.

[60] Goce Petrovski. *ScrollableSegmentedControl*. 2016. URL: https://github.com/GocePetrovski/ScrollableSegmentedControl (elérés dátuma: 2021. ápr. 25.)

[61] Felipe Pezoa és mások. „Foundations of JSON schema". *Proceedings of the 25th International Conference on World Wide Web*. 2016, 263–273. oldal.

[62] Alexander Poth, Mark Werner és Xinyan Lei. „How to Deliver Faster with CI/CD Integrated Testing Services?": *European Conference on Software Process Improvement*. Springer. 2018, 401–409. oldal.

[63] 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.

[64] Brennan Saeta és Denys Shabalin. „Swift for TensorFlow: A portable, flexible platform for deep learning". *Proceedings of Machine Learning and Systems* 3 (2021).

[65] Mark Sandler és mások. „Mobilenetv2: Inverted residuals and linear bottlenecks". *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, 4510–4520. oldal.

[66] Naveen Sankaran és CV Jawahar. „Recognition of printed Devanagari text using BLSTM Neural Network". *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*. IEEE. 2012, 322–325. oldal.

[67] Tom Schaul, Sixin Zhang és Yann LeCun. „No more pesky learning rates". *International Conference on Machine Learning*. PMLR. 2013, 343–351. oldal.

[68] Ken Schwaber és Mike Beedle. *Agile software development with Scrum*. 1. kötet. Prentice Hall Upper Saddle River, 2002.

[69] Stefan Tilkov Steve Vinoski. *Node.js: Using JavaScript to Build High-Performance Network Programs*. 2010. URL: https://ieeexplore.ieee.org/document/5617064 (elérés dátuma: 2021. ápr. 25.)

[70] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[71] Pedro Teixeira. *Professional Node. js: Building Javascript based scalable software*. John Wiley & Sons, 2012.

[72] TensorFlow. *TensorFlow Lite*. URL: https://www.tensorflow.org/lite/guide (elérés dátuma: 2021. ápr. 25.)

[73] *TFRecord generálása:* 2020. URL: https://github.com/douglasrizzo/detection_util_scripts/blob/master/generate_tfrecord.py (elérés dátuma: 2021. ápr. 28.)

[74] Stefan Tilkov és Steve Vinoski. „Node. js: Using JavaScript to build high-performance network programs". *IEEE Internet Computing* 14.6 (2010), 80–83. oldal.

[75] Kristın Fjóla Tómasdóttir, Mauricio Aniche és Arie Van Deursen. „The adoption of javascript linters in practice: A case study on eslint". *IEEE Transactions on Software Engineering* 46.8 (2018), 863–891. oldal.

[76] Jonathan Tremblay és mások. „Training deep networks with synthetic data: Bridging the reality gap by domain randomization". *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2018, 969–977. oldal.

[77] Yuiga Wada. *iCimulator*. 2019. URL: https://github.com/YuigaWada/iCimulator (elérés dátuma: 2021. ápr. 25.)

[78] Wikipedia. *Recycling codes*. URL: https://en.wikipedia.org/wiki/Recycling_codes (elérés dátuma: 2021. ápr. 25.)

[79] Shengkai Wu és mások. „Iou-balanced loss functions for single-stage object detection". *arXiv preprint arXiv:1908.05641* (2019).

[80] Shin Yamamoto. *Floating Panel*. 2020. URL: https://github.com/scenee/FloatingPanel (elérés dátuma: 2021. ápr. 25.)