COVID-19 védelmi mutató (CVM) dashboard

Angyal Olivér (KHQVVI)

Csomor Balázs Patrik (J85ZRU)

Schmelczer András (CVTNXD)

Radnai László (YJ5DMW)

Képernyőkép a CVM-dashboardról működés közben

Bevezetés

Az új koronavírus elleni védekezést segítendő, kidolgoztunk egy merőszámot, ami a szociális távolságtartás betartását méri. Ez a COVID-19 védelmi mutató, röviden CVM. A fejlesztésünk gyakorlatban is alkalmazható, ehhez a CVM értékét mérő és visszajelző alkalmazást hoztunk létre, ami a CVM-dashboard nevet viseli. Ennek segítségével adott helyszínen mérhető a terjedés valószínűsége és ezzel együtt az ott tartózkodóknak visszajelezhető a viselkedésük felelősségteljessége. Az elkészült webalkalmazást közzétettük, így azt bárki kipróbálhatja és használhatja. Ez a https://cvm-dashboard.ml/ címen érhető el.

Eredményeinket, illetve a tervezés és implementáció részleteit ebben a dokumentumban összesítettük. Ezen felül mind egy felhasználói, mind pedig egy fejlesztői dokumentáció is elérhető a projekt GitHub oldalán, a következő címen: https://github.com/angyaloliver/cvm-dashboard. A következőkben a választott technológiákat, az azok mellett szóló érveket és az egyéni megoldásainkat mutatjuk be.

Technológiák és implementáció

Alkalmazás kerete

Számos megközelítés létezik modern alkalmazások implementációjára, de közülük nagyon kevés ténylegesen platformokon átívelő és könnyen hozzáférhető a végfelhasználók számára. Az egyetlen járható útnak a webalkalmazás tűnik, mivel a böngészők többnyire ugyanazoknak a webes specifikációknak felelnek meg és minden platformon elérhetők. Továbbá, a linkre kattintás messze a legegyszerűbb módja egy alkalmazás elérésének. A webalapú implementáció azonban további kérdéseket vet fel.

Egy teljes szerver-kliens implementáció felesleges komplexitást vezetett volna be a projektünkbe, ezért egy elsődlegesen kliensoldali megközelítést választottunk. A kiszolgálást a Firebase Hosting [1] szolgáltatására bíztuk, a programunk logikáját pedig a kliensen hajtjuk végre. Ez a döntés természetesen kompromisszumokkal jár: a számítások a végfelhasználó eszközét terhelik, így a performancia erősen függ az adott hardver teljesítményétől. Cserébe a projekt komplexitása szignifikánsan kisebb és a skálázhatóság kérdése is megoldottnak tekinthető.

Úgy véljük, hogy a típusok statikus ellenőrzése kulcsfontosságú a megbízható alkalmazások létrehozásához. Ez segíti az integrált fejlesztői környezetet (IDE) és összességében javítja a fejlesztői élményt. Ezért döntöttünk úgy, hogy a webalkalmazást TypeScript [2] nyelven implementáljuk.

Az alkalmazásunk egyik alapkövetelménye emberek felismerése egy videó bemeneten. Számos megoldás létezik ennek a megvalósítására is, mi egy neurális hálókon alapuló megközelítést választottunk. Ehhez az egyik legnépszerűbb gépi tanulás keretrendszert, a Tensorflowt, pontosabban ennek JavaScript nyelven írt implementációját [3] használtuk, amely lehetővé teszi betanított modellek alkalmazását böngészőkben is.

Emberek felismerése videóról

Az alkalmazásunk céljának eléréshez elengedhetetlen egy képi bemenetről az azon szereplő emberek felismerése. Ennek megvalósítására több megoldást is kipróbáltunk.

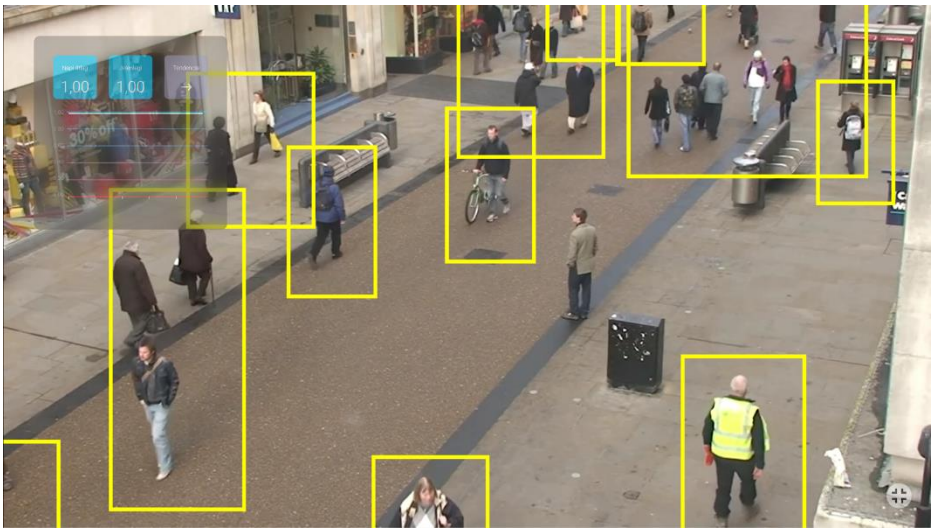
Egyszerű, algoritmikus megoldás

Dr. Vajta László Képfeldolgozás c. kurzusán átadott tanácsait követve első megoldásként igyekeztünk egy minél egyszerűbb algoritmust választani. Arra az alapvető felismerésre építünk, hogy ha $t \to \infty$, a kamera által látott képen lévő összes ember megjelent már korábbi képkockákon is, mozgó állapotban. Az emberek ezen mozgások által kerültek jelenlegi pozíciójukba. Így a mozgó régiókat követve képesek leszünk az emberek helyzetét számontartani.

A létrehozott feldolgozó rendszerünk az OpenCV [4] nyújtotta tradicionális képfeldolgozási lépésekből építkezik. Sajnos az OpenCV webes implementációja (OpenCV.js) nem támogatja a hardveres gyorsítást, így első lépésként a bemenet nagymértékű leskálázása szükséges a valós idejű élmény biztosítása érdekében. Ezután egy magasszintű lépés következik: a Gaussian Mixture-based Background/Foreground Segmentation Algorithm [5] [6] segítségével szegmentáljuk a képet. Ehhez a mostani és az előző pár képkockát használja fel az algoritmus.

Az eredményt megfelelően választott küszöbbel binarizáljuk, majd a zajcsökkentés végett mediánszűrjük. A kapott képre a morfológia zárás műveletét alkalmazzuk az esetlegesen több szegmensre széteső emberek összeillesztése céljából. A végeredményből már csak a mozgó területek pozícióit kell kiolvasni, a *cv.findContours*-t alkalmazzuk erre. A túlzottan kisméretű

régiók eldobásra kerülnek, a nagyobbakra pedig (a területnél egy kicsit nagyobb) befoglaló téglalapot illeszt a program.



Képernyőkép a fenti algoritmus eredményéről

Ahogy az ábrán látszik, pár nem elfogadható hiba is következik az elvégzett műveletekből. Egyrészt minden mozgó objektumot embernek hisz, ami nem minden környezetben fogadható el. Másrészt, az egymáshoz túl közel haladó, esetleg érintkező embereket nem különbözteti meg egymástól, őket egy objektumnak érzékeli. Ezen problémák megoldása, a valósidejűség megtartása mellett, a fenti algoritmus fejlesztésével nem tűnik megoldhatónak. Ebből kifolyólag egy komplexebb, szofisztikáltabb hozzáállást választottunk.

Mély neurális háló alapú megoldás

Miután megállapítottuk, hogy az egyszerű algoritmikus megoldás nem nyújt elegendő pontosságot, mély neurális hálókkal kezdtünk kísérletezni. Felmerült a két módszer együttes használata is, olyan módon, hogy a fentebb részletezett algoritmus kimenetét felhasználva kisebb képrészletet adnánk át egy neurális hálónak, azonban erről a feldolgozási sebesség érdekében lemondtunk.

Manapság gépi tanulás során a legnépszerűbb programnyelv a Python, a legnagyobb keretrendszerek (Tensorflow, Torch) ezen a nyelven írt könyvtárakkal rendelkeznek. A Tensorflownak azonban létezik egy JavaScript nyelven írt implementációja is, a korábban említett

Tensorflow.js. Ez a könyvtár a felhasználók eszközeinek grafikus gyorsítóját is ki tudja használni, így akár valós idejű feladatokra is alkalmazható.

A neurális hálókkal történő tárgyfelismerés egy széles körben kutatott terület, így bárki számára elérhetőek nagy pontosságú, előre tanított modellek. Saját hálónk tanítása helyett ezek közül igyekeztünk választani. Egy modellel szemben egyértelmű elvárás a gyorsaság és az objektum felismerés magas pontossága. Esetünkben ezen kívül a méret is döntő szempont volt, hiszen gyakoriak a több száz megabájtos modellek, amiket azonban nem célszerű kliens oldalon használni a megnövekedett adatforgalom és a lassú betöltés miatt. Ezt figyelembe véve elsősorban olyan megoldásokat kerestünk, amelyek böngészőben, vagy mobilalkalmazásban történő használatra készültek.

Három modellt vizsgáltunk meg, a COCO-SSD és a YOLO neurális hálókat, amelyek objektum felismerést valósítanak meg, illetve a PoseNet modellt, amely személyeket és azok testhelyzetét képes felismerni. Az alábbiakban látható összehasonlító táblázatban közelítő értékek szerepelnek, hiszen a mérések egyetlen eszközön és egy videófolyamon készültek, illetve minden esetben csak a lehető legpontosabb eredményt adó paraméterezést vettük figyelembe. A pontosságot a fals eredmények gyakorisága alapján empirikus úton állapítottuk meg.

Modell	**Méret**	**Sebesség**	**Pontosság**
COCO-SSD	67MB	137 ms (7 fps)	+++
Tiny YOLOv3	34MB	117 ms (8 fps)	++++
PoseNet	47MB	446 ms (2 fps)	+++++

Végül a Tiny YOLOv3 modellre esett a választásunk, amely a leggyorsabb feldolgozást szolgáltatta megfelelő pontosság mellett. A modellel való interakciót egy nyílt forráskódú JavaScript könyvtárra [7] alapoztuk, amely interfészt szolgáltat a modell betöltésére és az objektum felismerés futtatására Tensorflow.js segítségével. Annak érdekében, hogy ne blokkoljuk a felhasználói felület végrehajtási szálát, a modell használata egy web workerben [8] történik.

A neurális háló bemenete a videófolyam aktuális képkockája, a kimenete pedig címkékkel ellátott befoglaló téglalapok. A kimenetek közül a továbbiakban a *person* címkével ellátott,

megfelelően nagy bizonyossággal prediktált eredményeket használtuk. A bizonyosságra vonatkozó határértéket manuális teszteléssel, több különböző videófolyamon produkált eredmény összevetése alapján igyekeztük ideális értékre állítani. Végül az így előállított befoglaló téglalapokat transzformáltuk a későbbi feldolgozáshoz szükséges koordinátarendszerbe.

A táblázatban található sebesség értékekből látszik, hogy a YOLO modell is csak 8 képkocka feldolgozására képes másodpercenként. Annak érdekében, hogy a felhasználói felületen ennél gyakrabban tudjuk megjeleníteni az emberek pozícióját, a befoglaló téglalapok pozícióját extrapoláltuk.

A neurális háló által felismert személyek

Kamera paraméterek automatikus meghatározása

A kamera kalibrációja tipikusan egy emberi beavatkozást igénylő feladat, azonban megfelelő esztimációkkal ez elkerülhető. A felhasználói élmény növelése érdekében a manuális beállítást igénylő paraméterek számát nullára csökkentettük. Ebben a fejezetben az automatizált megoldásunk kerül bemutatásra.

A weboldal betöltésekor az emberek távolságának meghatározásához elengedhetetlen paraméterek kerülnek kiszámításra. Ezek rendre a kamera magassága, dőlése és a „Field of View”

(röviden FoV). Amennyiben elérhető és a felhasználó engedélyezi, az algoritmus felhasználja a mobiltelefonba épített giroszkópot is a kamera szögének meghatározásához. Az algoritmus bemenetét a neurális háló által felismert emberek képen lévő befoglaló téglalapja szolgáltatja.

A kamera paramétereket próbálgatással határozzuk meg. Amelyik a legvalószínűbb paraméterhármas, azt választjuk. Feltételezzük, hogy az emberek átlagmagassága 170 centiméter.

A magasság viszont számolható a kamera paramétereiből. A számolt magasság átlagtól való eltérésének négyzete határozza meg a hibát. Sok felismert emberre összesítsük ezt a hibát. Ahol a legkisebb a hiba, az a legvalószínűbb paraméterezés (bár elképzelhető hiba), ezért az ahhoz tartozó paraméterezést választjuk.

A feladat innentől az észlelt emberek magasságának meghatározása a vélt paraméterek alapján. A befoglaló téglalap alsó pontja az észlelt ember lábát jelöli (ami egy földön lévő pont), a teteje pedig a fejét. Egy tetszőleges (x, y, z) pont pin-hole vetítésre felírt kényszer két egyenletet ad.

x_s, y_s legyen egy képpont (-1 és 1 értelmezési tartománnyal, a $(0,0)$ a középső pixel). A kamera paraméterei y_e, φ, f rendre a kamera magassága, dőlése (vízszintes esetben 0) és a FoV értéke. φ értékét v_y és v_z segítségével is fel lehet írni, ahol a v vektor a (0,0) képpontra vetített egyenes irányvektora. Azaz $v_y := \cos\varphi$ és $v_z := \sin\varphi$.

A kép koordináta alapján egy egyenest úgy tudunk felírni, hogy $(0, y_e, 0) + t \cdot v'$, ahol az előbbi tag az egyenes egyik pontja (a kamera), v' pedig felírható v, x_s és y_s segítségével: $v' = v + f \cdot x_s \cdot (1,0,0) + f \cdot y_s \cdot (0, v_z, -v_y)$. Ez három, páronként merőleges vektor lineáris kombinációja. Az első a kamerasíkra merőleges komponens, a második a vízszintes eltolást határozza meg, a harmadik pedig a kamerasíkon a felfelé néző vektor komponens.

A $(0, y_e, 0) + t \cdot v' = (x, y, z)$ bevezet egy új paramétert (t), miközben x, y és z-re ad egy-egy egyenletet. Ezáltal lényegében két kényszeregyenletet kapunk.

A befoglaló téglalap aljához tartozó pontról továbbá tudjuk, hogy $y = 0$, a felső pontról, hogy $y = 1.7m$. Tudjuk, hogy a két pont X és Z koordinátája megegyezik. A két vetítési egyenlet egyértelműen meghatározza ezeket a paramétereket. A hiba $(y - 1.7m)^2$. Megjegyzendő, hogy

ugyan nem minden ember 170 centiméter magas, de ez a hiba kiegyenlítődik sok ember felismerése után.

Miután megismertük a paramétereket, minden ember helyzetét meg tudjuk határozni a kamerához képest az X-Z vízszintes síkon. Ez a pin-hole kényszeregyenletekből triviálisan kiszámolható.

CVM érték számítása

Ahogy az korábbi fejezetekben említésre került, az általunk definiált CVM érték a fő mutatónk a detektált emberek vírussal szembeni védettségnek a kifejezésére. Ebben a fejezetben az érték meghatározásához kapcsolódó megfontolások kerülnek bemutatásra. A következő elveket tartottuk szem előtt a CVM érték formulájának meghatározása során:

- Egy személynek a CVM értéke legyen 0, ha legalább
  - egy 0.5 méteren belüli szomszédja van.
  - három 1 méteren belüli szomszédja van.
  - hat 1.5 méteren belüli szomszédja van.
- A CVM érték legyen 1, ha egy személynek 1.5 méteren belül nincs szomszédja.

Egy egyén veszélyeztetettsége tehát a szomszédai (1.5 méternél közelebbi személyek) számától és közelségétől függ. Ezt az alábbi módon fejeztük ki (ahol d az n szomszéd távolságának növekvő sorrendbe rendezett tömbje):

$$\mathrm{CVM}_0 = \sqrt{\mathrm{d}_1 - 0.5} - 0.4 \cdot \sum_{i=2}^{n} 1.5 - \sqrt{d_i - 0.5}$$

- ha $d_1 \geq 0.5$
- ha $d_1 < 0.5$, akkor $\mathrm{CVM}_0 = 0$

A legközelebbi szomszéd távolsága kapja a legnagyobb súlyt a CVM meghatározásában. Minden további szomszéd a távolság függvényében tovább csökkenti a CVM értéket. A négyzetgyök függvény drasztikusabb változást eredményez alacsony távolság értékek (mint például 0.6 méter) esetén, mint nagyobb távolságok esetén. A függvény a távolságot a [0,1] intervallumba képzi le, ami egyben a CVM mutató értékkészlete is.

A képlet az implementációban kiegészült a történelmi értékek felhasználásával, illetve a távolság skálázásával, hogy zajos detekció esetén is megbízható eredményt generáljon a dashboard.

Felhasználói felület

Szoftverfejlesztőként bármennyire is szeretnénk kizárólag a program és az algoritmusok helyes és gyors működésére fókuszálni, ez napjainkban már nem elég. Ahhoz, hogy gyakorlatban alkalmazható végterméket kapjunk, a megfelelő felhasználói felület elkészítésének is prioritást kell élveznie.

Szerencsére a modern web nyújtotta lehetőségekkel kifejezettebb nehézség nélkül tudtuk a frontend alapjait implementálni a következő megfontolásokkal. Mindvégig szem előtt tartottuk az egységes dizájn nyelvezetet: a pozitív eseményeket kék szín jelöli, míg a negatívakat piros. Ezen felül fontos számunkra, hogy különböző felhasználási esetekre gondolva, minden érdeklődésű felhasználónak biztosítani az információ könnyű elérését.

A gyors áttekintést szolgálandó, a fő mutatók (a CVM átlagos és jelenlegi értéke, illetve a változás tendenciája) központi pozícióban helyezkednek el. A CVM értékek előzményét egy egyszerű diagramon ábrázoljuk. Ennek megjelenítéséhez az *ApexCharts* [9] könyvtárat választottuk.

Az oldal támogatja a teljes képernyős megjelenítést, ahol a kamera képe kerül kiemelésre. Gondolva arra, hogy kamera nélkül is kipróbálható legyen a dashboard, egy demó mód (és ahhoz tartozó kapcsoló elem) is része a weboldalnak. Demó módban előre felvett képet jelenít meg és dolgoz fel a program.

Az alkalmazás felületének leglényegesebb eleme egy CVM hőtérkép, ami a videó bemenetre kerül kompozitálásra. Az esztétikum és hasznosság egyensúlyát céloztuk a hőtérkép tervezésénél. Minden ember CVM értéke annak veszélyességét jelző szín segítségével egy kör alakú színátmenetként kerül kirajzolásra. Ezen grádiensek a közelség függvényében egymásba mosódhatnak. Mind a szín, mind pedig az átlátszóság dinamikusan változik a védekezési mutató függvényében.

Ahhoz, hogy egy gyors és látványos vizualizációt hozzunk létre, a hőtérkép optimális implementációja elsődleges fontossággal bírt. Ezért ez a WebGL technológia segítségével kerül kirajzolásra, amit szerencsére a használatban lévő eszközök 98%-a támogat [10]. Megfelelő árnyaló kód írásával és kellően kis felbontáson történő rajzolással, gyakorlatilag nullára csökkentettük a grádiens rajzolásának költségét.

Folyamatos integráció és telepítés

A folyamatos integráció és telepítés (CI/CD) egy modern projekt elengedhetetlen eleme. Számos probléma megelőzhető egy megfelelő csővezeték kiépítésével, ami gondoskodik az automatikus tesztelésről, telepítésről, vagy akár egy egységes fejlesztői környezetről. A fejlesztés egyes fázisaiban különböző eszközöket vettünk igénybe, ebben a fejezetben ezeket fogjuk bemutatni.

A tervezéshez a TeamGantt [11] eszközt használtuk, ami egy intuitív kezelőfelületet nyújtott a specifikációnkban definiált munkatervként szolgáló Gantt-diagramnak. Az alapvető eszköztárunkba tartozott még a GitHub verziókezelő, valamint a Visual Studio Code is. Utóbbit bővítményekkel (pl. Remote-Containers [12]) egészítettük ki a közös munka megkönnyítésének és egységesítésének érdekében. Teszteléshez a Jest keretrendszert [13] használtuk.

A tesztjeink automatikus futtatásáról, illetve az automatikus telepítésről is a GitHub Actions [14] gondoskodott. Workflow fájlokban definiáltunk műveleteket, amiket a fő (main) ág frissítése során (push művelet) minden esetben szerettünk volna elvégezni. Ilyen művelet a tesztek automatikus futtatása, ami ha hiba nélkül ér véget, a program új verziója is automatikusan elérhetővé válik a felhasználók számára. Utóbbi műveletet egy külön workflow fájlban definiáltuk, ami – a többi konfigurációs fájl mellett – megtalálható a projekt GitHub tárában.

Ahogy az már említésre került, kiszolgáláshoz Firebase Hostingot alkalmaztuk, amely egy tartalomszolgáltató hálózaton (CDN) keresztül szolgálja ki a felhasználókat az egész világon. Az alapértelmezett domain nevet (cvm-dashboard.firebaseapp.com) lecseréltük egy könnyebben megjegyezhető, elegánsabb névre (cvm-dashboard.ml), amit szerencsére térítésmentesen tudunk használni. A kiszolgálás mellett monitorozási lehetőségeket is nyújt a platform, ami hasznos metrikákat tesz számunkra elérhetővé.

Felhasználói dokumentáció

Felhasználói szemszögből az alkalmazásunk használata igen egyszerű – mindössze egy kamera és egy internetkapcsolattal rendelkező mobiltelefon vagy számítógép szükséges a működéséhez. Néhány megkötés persze érvényes a felsorolt követelményekre, ezeket fogjuk ebben a fejezetben részletezni.

Fontos, hogy a kamera paraméterei változatlanok maradjanak a program futása közben, tehát a megfelelő pozíció megtalálása után onnan elmozgatni a készüléket nem javasolt. Az ideális működés érdekében érdemes az eszközt 3-10 méter közötti magasságba helyezni, vízszintes horizonttal, semleges vagy enyhe negatív dőlésszögű (lefelé néző) beállítást alkalmazni.

A számítógépre, illetve mobiltelefonra vonatkozólag az erősebb hardverrel (elsődlegesen a GPU-t figyelembe véve) rendelkező készülékeken jobb felhasználói élményre lehet számítani. Az oldal megtekintéséhez javasolt a Google Chrome, Firefox, Opera, vagy Chromium-alapú Edge használata. Az Apple termékei jelenleg nem támogatottak.

A felsorolt követelmények teljesítése esetén a https://cvm-dashboard.ml/ oldalra navigálva (a kamera használatának engedélyezését követően) elérhetővé válik a dashboard és a kiértékelt kamerakép. A prezentációs célzattal implementált, jelenleg alapértelmezésben aktív “Demó mód” egy előre rögzített kamerakép segítségével mutatja be a szoftver működését. Értelemszerűen, valós alkalmazás esetén ez a funkció az oldalon található csúszka segítségével deaktiválandó.

Eredmények

Ebben a fejezetben a projekt elvégzése során összegyűjtött tapasztalatainkat fogjuk összefoglalni. Fontosnak tartjuk a tanulságok levonását, hogy a jövőbeni kihívásainkban az itt elkövetett hibákat elkerülhessük, a bevált gyakorlatokat pedig újra alkalmazhassuk.

A specifikációban definiált munkaterv nagyon hasznosnak bizonyult az időmenedzsment szempontjából. A Gantt-diagram kiválóan alkalmas kis csapatok munkájának összehangolására, a határidők és függőségek követésére. A félév során általában sikerült tartanunk az ütemtervet, a feladatait jellemzően mindenki időben elvégezte. Ezt annak is köszönhettük, hogy csapatunk tagjai

egyenletesen voltak terhelve – optimálisan sikerült elosztanunk a teendőket. Az egyedüli nehézséget az egyes feladatok időtartamának megbecsülése okozta, ami olykor pontatlanságokhoz vezetett, de szerencsére ezeket sikerült kompenzálni, így elértük a kitűzött céljainkat.

A Képfeldolgozás c. tárgy keretei között elsajátított tudás egy részét a gyakorlatban is sikerült kipróbálnunk, ami által mélyebb megértésre tehettünk szert. Különösen az emberek felismerésének kézi implementációja, majd neurális háló alapú megoldása és a kamera kalibrációja során tudtuk a tanultakat alkalmazni.

Összességében, egy gyakorlatban alkalmazható és a közösség érdekét szolgáló megoldást hoztunk létre. Ráadásul ez nemcsak egyszerűen elérhető bárki számára, hanem még kellően pontos is, ami a termék iránti bizalom kiépülését és a könnyű adaptációt is segíti.

Hivatkozások

[1] Google, „Firebase Hosting Documentation,” [Online]. Available: https://firebase.google.com/docs/hosting. [Hozzáférés dátuma: 13. december 2020.].

[2] Microsoft, „TypeScript Documentation,” [Online]. Available: https://www.typescriptlang.org/docs/. [Hozzáférés dátuma: 13. december 2020.].

[3] Google, „Tensorflow.js Documentation,” [Online]. Available: https://js.tensorflow.org/api/latest/. [Hozzáférés dátuma: 13. december 2020.].

[4] Intel Corporation, „OpenCV Documentation,” [Online]. Available: https://docs.opencv.org/4.5.0/. [Hozzáférés dátuma: 13. december 2020.].

[5] Z. Zivkovic, „Improved adaptive Gaussian mixture model for background subtraction,” in *IEEE*, 2014.

[6] Z. Zivkovic és F. van der Heijdenb, „Efficient adaptive density estimation per image pixel for the task of background subtraction,” 2006.

[7] „tfjs-yolo,” [Online]. Available: https://github.com/shaqian/tfjs-yolo. [Hozzáférés dátuma: 13. december 2020.].

[8] Mozilla, „Web Workers API,” [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API. [Hozzáférés dátuma: 13. december 2020].

[9] ApexCharts, „ApexCharts Documentation,” [Online]. Available: https://apexcharts.com/docs/. [Hozzáférés dátuma: 13. december 2020.].

[10] A. Deveria, „Can I use WebGL?,” [Online]. Available: https://caniuse.com/?search=webgl. [Hozzáférés dátuma: 13. december 2020.].

[11] TeamGantt, „TeamGantt,” [Online]. Available: https://www.teamgantt.com/. [Hozzáférés dátuma: 13. december 2020.].

[12] Microsoft, „Remote-Containers,” [Online]. Available: https://marketplace.visualstudio.com/items?itemName=ms-vscode-remote.remote-containers. [Hozzáférés dátuma: 13. december 2020.].

[13] Facebook Inc., „Jest Documentation,” [Online]. Available: https://jestjs.io/docs/en/getting-started. [Hozzáférés dátuma: 13. december 2020.].

[14] GitHub Inc., „GitHub Actions Documentation,” [Online]. Available: https://docs.github.com/en/free-pro-team@latest/actions. [Hozzáférés dátuma: 13. december 2020.].