Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Németh Gergely

OTTHONBIZTONSÁG MEGTEREMTÉSE ELÉRHETŐ BEÁGYAZOTT RENDSZER SEGÍTSÉGÉVEL MESTERSÉGES INTELLIGENCIA ALAPOKON

KONZULENS: Dr. Forstner Bertalan

BUDAPEST, 2022.10.31.

Tartalomjegyzék

Összefoglaló

Napjainkban egyre több és egyre fontosabb szerepet játszik az okosotthon. A háztartásokban már minden eszközt felszereltek okos funkciókkal, hogy azok az emberek életét egyszerűsítsék. Az okoskaputelefon az előbb említett célt szolgálva segíti a ház tulajdonosát a biztonság megteremtésében további hasznos információkkal kiegészítve. Az elmúlt évek hardveres fejlődése, valamint ezen eszközök széles hozzáférhetősége lehetővé teszi, hogy korábban elképzelhetetlen számításigényű megoldásokat ültessünk át hétköznapi területekre.

A kutatás célja annak bizonyítása, hogy az olcsón elérhető számítástechnikai platformok alkalmasak a területen való alkalmazásra mesterséges intelligencia felhasználásával. Egy ilyen rendszertől elvárható, hogy képes legyen az arcokat felismerni, azonosítani és az arc észrevétele után külön okosotthon funkciókat végrehajtani (pl: értesítést küld a telefonra, ha ismert vagy ismeretlen arc áll az ajtó előtt). Az okosotthon megteremtése céljából a szoftver további információkat biztosít a felhasználó számára, így a biztonság megteremtése érdekében az alkalmazás minden érzékelt arcról felvételt készít, amelyet a felhasználó vissza tud követni. Egy ilyen rendszer a környezetből szerzett információk alapján képes figyelni a otthonunkat és értesítést küldeni a felhasználónak, ha érkezett postai küldemény, csomag, illetve, ha már elvitték a szemetet.

A dolgozat első felében bemutatásra kerülnek a különböző területen elérhető mesterséges intelligenciát alkalmazó arcfelismerő technológiák. Továbbá a kutatás különbséget tesz a technológiák között és bizonyos szempontok alapján kiválasztja a legmegfelelőbbet. A tanulmány továbbiakban kitér a kutatás eredményének validálására egy prototípus elkészítésével.

Abstract

Smarthomes play more and more roles nowadays. In our homes, every tool is already installed with smart functions, because it simplifies people's lives. The smart doorphone serves the purpose mentioned earlier, helps the owner of the house to create security, supplemented with additional useful information. The hardware development of recent years, as well as the wide availability of these devices, make it possible to implement previously unimaginable computing-intensive solutions into ordinary areas.

The aim of the research is to prove that inexpensive computing platforms are suitable for using them in the field using artificial intelligence. Such a system can be expected to be able to control more complex functions in addition to face recognition, such as recognize and identify faces and after noticing the face, it performs special smart home functions. (e.g. sending a notification to the phone, if a known or unknown face is standing in front of the door). In order to create a smart home, the software provides the user with additional information. In order to create security, the application takes a picture of every detected face, which the user can look back to. The smart doorbell also monitors the environment and sends a notification to the user, if mail or a package has arrived or, if the trash has already been collected.

In the first half of the thesis, facial recognition technologies using artificial intelligence available in different fields are presented. The research differentiates between technologies and selects the most appropriate one based on certain aspects. The study further covers the validation of the research vessel by making a prototype.

1 Bevezetés

Napjainkban a biztonság egyre nagyobb szerepet játszik. Már minden üzletben, bankban, utcákban és még az autókban is található kamera, hogy bármi probléma esetén vissza lehessen követni a történéseket. A ma épülő modern házak szintúgy fel vannak szerelve különböző biztonsági és okosotthon funkciókkal, amelyek biztosítják a tulajdonosok kényelmét és jó közérzetét. A különböző technológiák napról napra fejlődnek, azonban az újabbnál újabb eszközökért különösen magas árat kell fizetni.

1.1 Problémafelvetés

A biztonság megteremtése mindig is kiemelkedő szempont volt az emberiség történelmében. Családunk védelme érdekében dolgozunk, hogy egy kényelmes otthonban tudhassuk szeretteinket, azonban nem tudjuk minden pillanatban felügyelni a házat. Amíg távol vagyunk lakhelyünk körül zajlik az élet, nem tudván, hogy kik állnak meg házunk előtt és hogy milyen emberek figyelik a lakásunk bejáratát. A biztonság érdekében okosotthon kamerák százai állnak a tulajdonosok rendelkezésére, melyek makulátlanul ellátják feladatukat, azonban többségük nem megbízható és egyáltalán nem felhasználóbarát. Erre bizonyíték a 2021-ben történt Hikvision botrány [1]. A hiba a webes felületen mutatkozott meg, ugyanis elérhető volt a kamera képe az interneten keresztül. A támadó a tulajdonosnál is nagyobb admin jogot tudott így szerezni, ezzel átvéve az uralmat a kamera felett. Ekkor, ez a probléma több mint ötmillió kamerát érintett.

Egyik fontos szempont egy okos eszköz megvásárlásánál az ár. A piac fel van szerelve megannyi készülékkel, amely a kényelmünket és biztonságunkat biztosítja, azonban a megbízható eszközök többsége túlárazott és az emberek nagyobb százaléka nem engedheti meg magának ezeket. Ha példának vesszük a ma kapható okoskamerák egyik top eszközét az Arlo 4 Pro-t [2], megfigyelhető, hogy bár remek funkciókkal rendelkezik, 200 dolláros ára ma már egy középkategóriás telefonnal is vetekszik.

Egy okos eszköztől elvárható, hogy különböző funkciókkal segítse tulajdonosát. Az okoskamerák többsége biztonságot nyújt, azonban zöme nem képes plusz információval ellátni a gazdáját. Ha már kifizettünk annyi pénzt, mint egy telefon ára a lakóhelyünk védelme érdekében, igazán számíthatunk egy kis kényelemre is, amelyek mindennapi életünket segítik

plusz információkkal, mint például, hogy elvitték-e már a szemetet vagy érkezett-e postai küldemény.

1.2 Célkitűzés és motiváció

A fő motiváció, hogy képesek vagyunk a ma elérhető technológiával egy olyan terméket készíteni, amely egy olcsó, kevés erőforrással rendelkező készüléken is makulátlanul fut úgy, hogy a felhasználó számára könnyen kezelhető és megannyi plusz információval segíti a mindennapokban.

Mivel az elérhetőség a cél, a tanulmány egy olcsó eszközt tesztel, amely ma már minden háztartásban megtalálható. A gyorsan fejlődő világban már megszokottá vált, hogy telefonunkat legfeljebb két évente lecseréljük egy újabb, jobb modellre. Ha körbenézünk otthonunkban biztosan találunk egy polcon porosodó Android eszközt, amelynek már nem vesszük hasznát, ezért vizsgálom ezeket a készülékeket, melyekre már nincs szükségünk, de remek szolgáltatást tudnának nyújtani egy okoskamera szerepében. Mivel régebbi eszközökről beszélünk, a tanulmány elsődleges célja, hogy olyan arcfelismerő és arcazonosító alkalmazást lehessen készíteni, amely kevés erőforrást igényel, mégis kiemelkedő teljesítménnyel rendelkezik.

Az okoskamera eszközök magas árára a példa az alábbi cikk [32]. Ebben az írásban arról beszélnek, hogy miért ilyen drágák a Google Nest biztonsági kamerák. A cikk írója nem csak azt vázolja, hogy mennyibe is kerül egy készülék, de a Google Nest további plusz összegért prémium csomagot is ad. Ez azt jelenti, hogy a Nest kamerák nem képesek videót rögzíteni, amig nem vagyunk előfizetve a bónuszcsomagra.

A OneSmart oldalán [33] készült cikkben az okos kapu zárakról ír és elemzi az előnyüket és hátrányukat. A cikk írója felsorolja kapukamera előnyeit, mint például, hogy magasfokú biztonságot kínál, mindemellett nem lesz szükségünk a kulcsunkra sem, hogy kinyissuk az ajtót, ugyanis a készülék képes kinyitni az ajtót magától. Remek funkciókkal rendelkezik, amelyek kényelmesebbé teszik mindennapi életünket, azonban egy költséges hátránya is van. Az ilyen típusú okos kapu zárok vételi ára mellett gondolnunk kell a beszerelési költségre is, ugyanis a telepítést érdemes egy szakemberre bízni.

1.3 Dolgozat felépítése

A dolgozatban a kutatás eredményét fejtem ki. A cél elérése érdekében tesztelem és kipróbálom a különböző gépi látású technológiákat és arcfelismerő algoritmusokat, amelyeket a dolgozat első felében részletezek, mindemellett külön figyelmet fordítok ezeknek az erőforrásigényére. Összesen két fő könyvtárat mutatok be, illetve prezentálásra kerülnek a kiválasztási szempontok. Az ezt követő részben kifejtem a döntésem okát.

A tanulmány második fele a kutatásom validálásról szól. Egy prototípust készítek, amelyben megvalósítom a kiválasztott technológiát, ezzel érvényesítve a kutatás mögött lévő elméletet. A dolgozat végén a prototípust különböző készülékeken tesztelem és vizsgálom az erőforrás használatukat, amivel kimutatom az elért eredményeket.

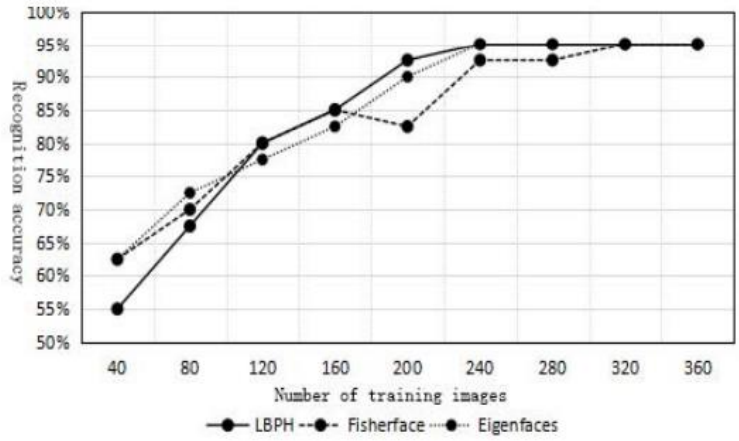
2 Kapcsolódó irodalom

A kutatás a témával kapcsolatos irodalmak feldolgozásával kezdődik. Fontos számomra, hogy feltérképezzem a jelenlegi technológiákat és hogy milyen eredményekkel zárultak más kutatók tanulmányai.

2.1 Az arcaznosításban elért eddigi eredmények

Az [3] által bemutatót kutatásban a célkitűzés egy olyan alkalmazás megalkotása volt, amely képes felváltani az emberi erőforrásokat a gépi látásra, ezzel leküzdve a kézi jelenlét hátrányait. Az alkalmazás az arcok alapján vezeti az órán való részvételt, ezzel helyettesítve a kézi aláírásos jelenléti ívet. A tanulmányban az arcfelismerő jelenléti rendszer három funkcionális modulra oszlik: információbevitelre, jelenléti bejelentkezésre és jelenléti nyilvántartásra. Az első modul a felhasználók alapvető információinak bevitelére szolgál, így begyűjti a diákok arcképeit és ezekre az adatokra egy arcfelismerő osztályt tanít. A második modul ennek a rendszernek az alapvető funkcióját valósítja meg. Ha a felhasználót sikeresen azonosította a program az arc beolvasásával, akkor az felismeri a diák jelenlétét, de ha nem sikerült azonosítani, automatikusan lefényképezi, majd manuálisan rögzíti a személyről készült képet. Az alapvető arcfelismerési folyamat három részre osztható: arcfelismerés, osztályozó betanítás és arcazonosítás. Az arcfelismerés célja az arcot tartalmazó terület felismerése a képen. Ebben a folyamatban a bemeneti képet formálják, majd az Adaboost osztályozó [4] felismeri az arc részét. A második lépés az LBP-jellemzők kinyerése az arcterületből, illetve az LBP-hisztogramjának kiszámítása. Az LBP (Local Binary Pattern) egy operátor, amelyet a kép textúrák jellemzőinek leírására használnak [5]. Az LBP alapötlete az, hogy összeadja a pixelek és a környező pixelek közötti különbség eredményét, hogy egy pixelt vegyen középpontnak, ezzel küszöbértékeket hasonlítson össze a szomszédos pixelekkel. A harmadik szakaszban, az észlelt arcterület alapján az LBP-jellemzőket kinyerik és beadják az osztályozóba az egyes felhasználók azonosításához. A kutatásban Adaboost-ot alkalmaznak, amely elsősorban osztályozási és regressziós feladatokhoz használható. Ez egy iteratív algoritmus, melynek lényege, hogy egy erős tanuló halmazt hozzon létre a gyengén betanított osztályozó halmazok segítségével. Az algoritmus a gyenge halmazokat osztályozási minták szerint súlyozza és az eredmények alapján egymásra építi. Az képzésekből nyert összes halmaz végül egy végső erős tanuló halmazba integrálódik.

Ez a cikk a szakirodalomban javasolt reprezentációs módszert alkalmazza és az LBP jellemző spektrumának statisztikai hisztogramját használja jellemző vektorként az osztályozáshoz és felismeréshez. Első lépésben egy képet több régióra osztanak fel és a régió minden pixeléhez kivonják az LBP jellemzőket. Ily módon egy régiót le lehet írni egy statisztikai hisztogrammal, amelyeket lokális bináris módú hisztogramoknak nevezünk. A kép régiókra bontásával és az LBP hisztogram külön kiszámításával bizonyos mértékig csökkenthető a kép nem megfelelő igazításából adódó hibák. Eközben a különböző régiókhoz különböző súlyok rendelhetők. Például a középső rész nagyobb súllyal állítható be, mint az élrész, így a középső rész jelentősebb a képillesztésben és felismerésben. Az LBP algoritmus érvényességének igazolására két klasszikus arcfelismerő algoritmust tesztelnek, a Fisherface-t és az Eigenface-t. Ezen túlmenően a cikk az arcfelismerő osztályozó képzésben eltöltött időről statisztikát készít, különböző számú kép mellett három algoritmus segítségével.



3.2.1.1. ábra: Algoritmusok teljesítményének összehasonlítása [3]

Az 3.2.1.1. ábra mutatja, hogy a tanuló képek számának növekedésével az arcfelismerés pontossága jelentősen javult. Személyenként öt képpel tesztelve az arcfelismerés pontossága elérheti a 95%-ot is. 3+ kép esetén az LBP algoritmus jobb eredményt ért el, mint a másik két algoritmus. Ebben a cikkben megvalósítottak egy jól működő arcazonosítást különböző algoritmusokkal összehasonlítva. Ezzel igazolja, hogy Android platformon lehetséges az arcok azonosítása, azonban sajnos a cikk nem tér ki az erőforrás használatra.

A következő irodalomban [6] az arc azonosítási technológiákat mutatnak be és hasonlítanak össze. Ebből a legérdekesebb az személy felismerése a Tensorflow Lite segítségével. Három különböző modellt mutat be, a FaceNet-et [7], a MobileFaceNet-et [8]és az InceptionResNet-et [9]. Mindhárom modell különbözőképpen került felépítésre, ezért a cikk

írója vizsgálja az erőforrás használatot és keresi a legmegfelelőbbet. Az első vizsgálati szempont a teljesítményét vizsgálta. A teszt során a legjobb eredményt a FaceNet mutatta, ugyanis a tanulás közben mindössze 3%-ot csökkent a teljesítménye, míg a másik kettő esetében ez meghaladta a 10%-ot is. A következőkben a következtetési időt vizsgálta, melyben a MobileFaceNet biztosul a leggyorsabbnak. Összesen 98 milliszekundumba került neki az érzékelés, ami jóval kisebb, mint az InceptionResNet 210 milliszekundum ideje. Az előrejelzési időt elsősorban a modell kimeneti mérete befolyásolja. A tanulmány szerint a legjobb idővel rendelkező modell ismételten a MobileFaceNet. Mindössze 3.6 milliszekundum időre volt szüksége, míg a másik kettőnek az ideje 6 és 8 milliszekundum között mozgott. Ez azzal magyarázható, hogy a MobileFaceNet kimeneti vektor mérete csökkentett, mindössze 112 tulajdonsággal rendelkezik, míg a FaceNet-nek és az InceptionResNet-nek 160 tulajdonsága van. Összességében mindhárom modell effektív, mivel az idejük 10 milliszekundum alatt van. Az utolsó tesztben a tanulmány az erőforrás használatot méret. A legrosszabb eredményt az InceptionResNet modell adta, közel 538 MB memóriát igényelt, míg a MobileFaceNet 359 MB memória használattal dolgozott. A processzor használatban mindhárom hasonló eredményeket mutattak. Végeredményül a tanulmány célja az volt, hogy kiválassza a legjobb opciót az arc azonosításhoz, hogy egy alkalmazást készítsen Android eszközre. Az arc érzékeléshez az ML Kit rendszerét használta, mivel ez mutatta a legjobb eredményeket egy személy felismerésében. A cikk írója szerint a legfontosabb aspektus az arcazonosításban az a teljesítmény. A FaceNet mutatta a legjobb eredményeket a teljesítményben, azonban alul maradt a gyorsaságban a MobileFaceNethez képest. Az InceptionResNet mindkét esetben a legrosszabb eredményt mutatta, így teljesítmény alapon a FaceNet-tel valósította meg az alkalmazását. A tanulmány végül az ML Kit-et és a FaceNet-et ajánlja, mint a legjobban teljesítő opció az arcazonosítás terén.

3 Arcfelismerés és arcazonosítás

Az arcfelismerés egyszerű feladat az ember számára. Már egy három napos baba is képes megkülönböztetni az ismert arcokat. Az automatikus arcfelismerés lényege, hogy kivonjuk a képből a számunkra jelentős jellemzőket és valamilyen osztályozást hajtsunk végre rajtuk. A következőkben, különböző technológiákat és algoritmusokat vizsgálok, kutatva a legegyszerűbb, leggazdaságosabb módszert a megfelelő arcazonosítás érdekében.

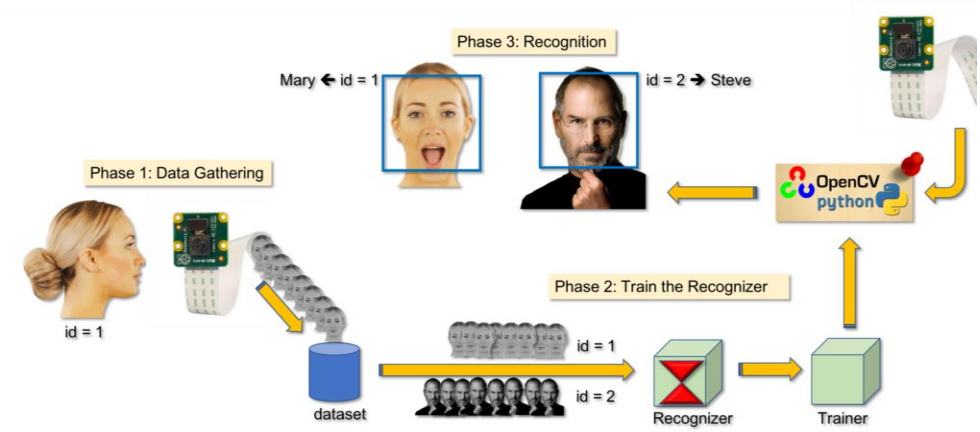
3.1 Arcfelismerés és arcazonosítás

Első lépésben érdemes megkülönböztetni az arcfelismerést és az arcazonosítást. Sokan ezt a két meghatározást egy és ugyanazon fogalmaknak gondolják, viszont valójában két külön feladat. Az arcfelismerés csak az arc felismerését foglalja magában, ami egyszerűen azt jelenti, hogy az arcfelismerő rendszer képes azonosítani, hogy emberi arc szerepel e a videón vagy képén, de nem tud nevet párosítani a személyhez. Néhány meglehetősen jelentős különbség van az arcfelismerés és az arcazonosítás között, azonban a kettő elválaszthatatlanul összefügg. Az arcfelismerés az arcazonosító rendszerek egyik összetevője, mivel az arcazonosítás első szakasza az emberi arc jelenlétének észlelése. A fő különbség, hogy az arcfelismerés olyan rendszerre utal, amely egyszerűen észleli az emberi arc jelenlétét, viszont az arcazonosítás ezt a következő szintre emeli és képes felismerni és azonosítani az észlelt személyt egy adatbázisban tárolt egyezés alapján. [10]

3.2 OpenCV

Az első kódkönyvtár, amivel foglalkozom az a gépi látásban már remek eredményeket elért OpenCV. Az OpenCV (Open Source Computer Vision Library) [11] egy nyílt forráskódú könyvtár, amit azért hoztak létre, hogy egy közös infrastruktúrát biztosítsanak a gépi látással foglalkozó alkalmazásokhoz.

Az OpenCV tartalmaz mindent, ami az arc felismeréséhez és azonosításához szükséges, mivel rendelkezik érzékelővel és beépített tanuló algoritmusokkal. A 3.2.1.1. ábra bemutatja az OpenCV-vel való arcazonosítás menetét. A jelenleg elérhető algoritmusok az Eigenfaces, a Fisherfaces és a Local Binary Patterns Histograms. [12]



3.2.1.1. ábra: OpenCV arcazonosítása [29]

Az arc azonosításához elengedhetetlen első lépés egy megfelelő adatkészlet elkészítése, amely nélkül nem lennénk képesek arcokat felismerni. A kódkönyvtár biztosítja az egyedi modellek tanítását, de az OpenCV-hez készültek már jól működő adatkészletek. Ilyen például az AT&T Facedatabase, amely egy egyszerűbb, gyorsabb megoldást biztosít, azonban mindössze tíz különböző kép alapján különbözteti meg az arcokat. Mivel kevés képpel dolgozik, a pontosság alacsony, amivel nem képes sok arcot összehasonlítani. Abban az esetben, ha megbízhatóbb adatforrást szeretnénk használni érdemes a Yale Facedatabase A-t vagy a Extended Yale Facedatabase B-t alkalmazni, amelyek több képből dolgoznak, viszont hátrányuk, hogy méretük nagy és erőforrásigényük magas. A kutatás során a törekvés egy olyan metódus megtalálása, amely olcsó és kevés erőforrást igényel, így az AT&T Facedatabase bizonyul a legmegfelelőbbnek.

A következőkben két tanuló algoritmust mutatok be, amiket a tanulmány során tesztelek. Az egyik legfőbb probléma a képi ábrázolással, hogy túl nagy kiterjedésű. Erre ad megoldást az Eigenfaces, amely Principal Component Analysis (PCA)-t használ. A kétdimenziós p×q képek m=p×q dimenziós vektorteret ölelnek fel, tehát egy 100×100 pixeles kép már 10 000 dimenziós képtérben fekszik. Ennél nagyobb pixel arányok használata esetén a képtér már kezelhetetlenül nagy lesz, illetve a megfelelő azonosítás érdekében nem minden dimenzióra van szükségünk.

Az Eigenfaces a PCA-t használva az esetleges korrelált változókat egy kisebb, nem korrelált változók halmazává alakítja. Egy nagy kiterjedésű adatkészletet gyakran korrelált változók írnak le és csak néhány értelmes dimenzió adja az információ nagy részét, így a PCA módszer megkeresi az adatokban a legnagyobb szórással rendelkező irányokat. A PCA leképezés után az algoritmus a következő lépéseket hajtja végre: Kivetíti az összes képzési

mintát és lekérdezési képét a PCA alterébe, majd megkeresi a legközelebbi szomszédot a kivetített képzési képek és a kivetített lekérdezési kép között.

Principal Component Analysis a jellemzők lineáris kombinációját találja meg, amely maximalizálja az adatok teljes varianciáját. Bár ez egyértelműen egy hatékony módja az adatok megjelenítésének, nem vesz figyelembe semmilyen külső szereplőt, így sok megkülönböztető információ elveszhet. Könnyen előfordul olyan helyzet, amikor az adatok eltérését egy külső forrás zavarja, példának okán egy fény. Mivel a PCA által azonosított komponensek egyáltalán nem tartalmaznak megkülönböztethető információkat, így a kivetített minták összekenődnek és lehetetlenné válik az osztályozás. Erre a problémára ad megoldást a Fisherfaces algoritmus, amely Linear Discriminant Analysis (LDA)-t használ. Az osztályok között legjobban elválasztó jellemzők kombinációjának megtalálása érdekében a Lineáris Diszkriminancia Analízis maximalizálja az osztályok közötti és az osztályokon belüli szórások arányát. Ugyanazon osztályok szorosan egymás mellé csoportosulnak, míg a különböző osztályoknak a lehető legtávolabb kell lenniük egymástól. A Fisherfaces metódus megtanul egy osztályspecifikus transzformációs mátrixot, így a diszkriminancia elemzés helyet, az arcvonások tesznek különbséget a személyek között.

3.2.1 Tapasztalat

A két metódust a következő cikkben [13] teszt alá vetik egy ugyanolyan adatkészlettel, hogy kiderítsék melyik rendelkezik jobb teljesítménnyel. Az ebben a kísérletben használt adatkészlet a Labelled Faces in the Wild (LFW) [34] névre hallgat, amely egy benchmark adatkészlet [35]. 13 233 arcképet tartalmaz, amelyek a személyek neveivel vannak elmentve. A képek 250 x 250 pixeles JPEG formátumban érhetők el. Az adatbázis 5749 különböző személy képét tartalmazza, ebből 1680 embernél kettő vagy több kép szerepel az adatbázisban. A fennmaradó 4069 emberről csak egyetlen kép van az adatbázisban. Az LFW-adatkészletből a legtöbb képpel rendelkező tíz személy kerül kiválasztásra az adatkészlet korlátozott verziójának elkészítéséhez. A tíz személyről 40 kép van tanulásra és négy kép tesztelésre. Ez összesen 440 kép, 400 tanulásra és összesen 40 tesztre vannak fordítva.

Az eredmények az egyénenkénti teljesítményt mutatják, ahol tíz különböző személyt állapítottak meg. Az egyes technikák átlagos pontszámait az alábbi 3.2.1. táblázat mutatja.

Név	Pontosság	Pontosság	Felidézés	F-Score	Kiesés
Eigenfaces	83,1 %	15,34 %	15,50 %	14,53 %	9,39 %
Fisherfaces	83,87 %	19,27 %	19,35 %	18,34 %	8,96 %

3.2.1. táblázat: A technikák átlagos százaléka

Ebben a tanulmányban olyan mérőszámokat használnak, mint a felidézés, a pontosság és az F-score annak érdekében, hogy szélesebb körben és jobban megértsük a munkában összehasonlított technikák/algoritmusok teljesítményét. Az eredmények azt mutatják, hogy az Eigen és a Fisherface se produkál magas értéket a felidézés, a pontosság és az F-score tekintetében, ennek egyik oka az lehet, hogy mindkettő az alapértelmezett beállításokat használja. A teljesítmény növelni lehet különböző k-értékek bevezetésével. Az Eigen és a Fisher 83% pontossággal rendelkezik, csak századokban térnek el egymástól, amely nem jelentős különbség. Azonban, ha a többi eredményt vizsgáljuk kimutatható, hogy a Fisher jobban teljesít, mint az Eigen. Az F-score a legjobb mérőszám annak meghatározására, hogy egy módszer mennyire képes helyes és helytelen előrejelzéseket végezni. Az Eigen 14,53% teljesítményével rosszabb eredményt produkál, mint a 18,34%-kal rendelkező Fisher. A kiesés egy téves riasztás vagy egy hibás előrejelzés valószínűsége, ennek optimális értéke 0%. A legjobb bukást szintén Fisher produkálja a 8,96%-os eredményével. Az is megállapítható, hogy osztályozótól függetlenül az eredmények azt mutatják, hogy a Fisherface jobban teljesít, mint az Eigenface algoritmus.

Név	Tanulási idő	Jóslás idő
Eigenface	3.68 s	0.0982 s = 98.2 ms
Fisherfaces	4.19 s	0.0075 s = 7.5 ms

3.2.2. táblázat: Tanulás és jóslás idő

Ami az időt illeti, az OpenCV implementációk betanítása lényegesen hosszú időt vesz igénybe. Az Eigenface a 3,68 másodperc tanulási idejével jobb eredményt ad, mint a Fisherface, azonban a jóslási időben nagyon alul marad, ugyanis a Fisherface 13-szor gyorsabb idővel rendelkezik, ami azt mutatja, hogy jelentős mértékben gyorsabban ismeri és azonosítja fel az arcokat.

3.3 Android ML Kit, Tensorflow Lite

ML Kit egy mobil SDK [14], amit a Google fejlesztett gépi tanulás céljából. Az ML Kit API lehetővé teszi, hogy valós időben hajtson végre a program olyan gép tanulást, amely segítségével egyszerűvé válik az objektum vagy az arcok érzékelése. A technológia egy teljes körű csomagot nyújt a célom elérése érdekében, ezért foglalkozom vele a következő részben.

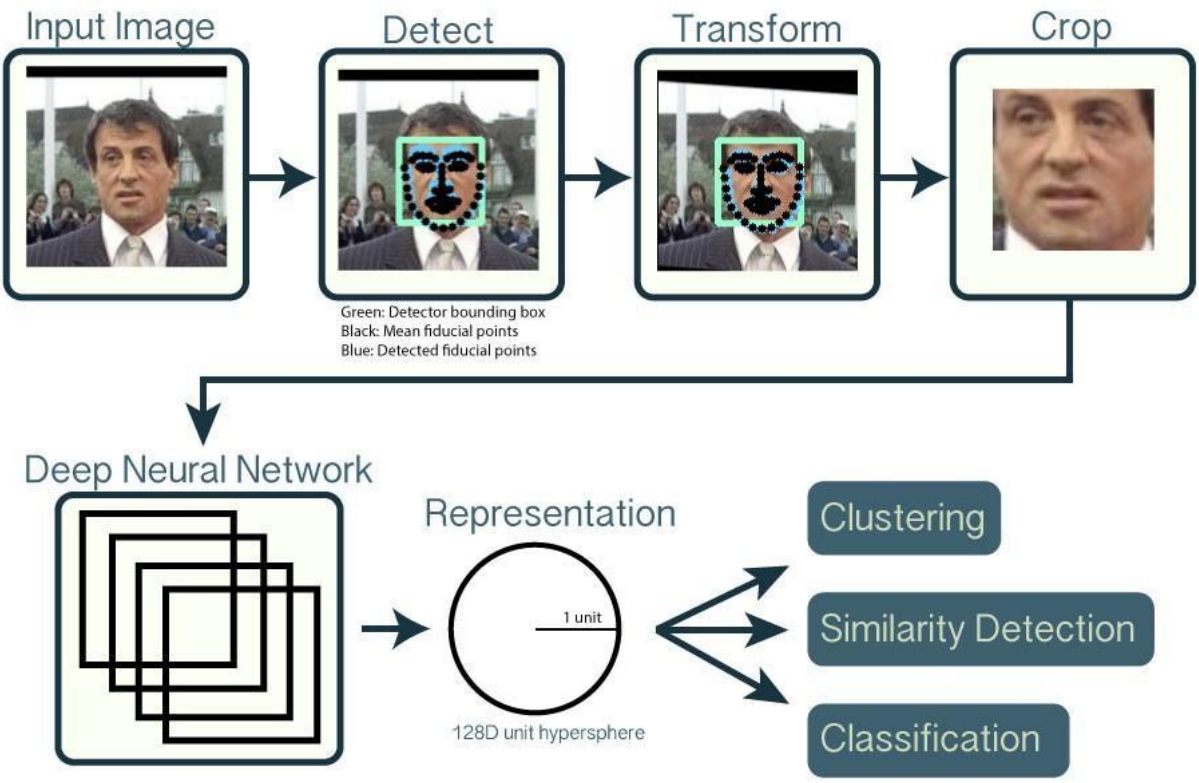
3.3.1 Arcazonosítás menete

Az ML Kit arcfelismerő API-jával [15] felismerhetjük az arcokat a képen, azonosíthatjuk a legfontosabb arcvonásokat és lekérhetjük az észlelt arcokat. Az API használatához elengedhetetlen a CameraX könyvtár [16], amely biztosít egy egyszerű kamera kezelést. A könnyű használat érdekében az Android fejlesztők use case-kez vezettek be, melyek a következők: Preview, Image Analysis, Image Capture és a Video Capture. A Preview egy felületet biztosít, amelyre ki lehet rajzolni a kamera képét. Ez egy kamera előkép, amely könnyen formázható és beépíthető egy Layout-ba. Az Image Analysis az egyik legfontosabb use case az arc érzékelés szempontjából. Ez a ‘use case’ határozza meg az elemzéshez szükséges algoritmust, hogy a kamera képén gépi tanulást is végre lehessen hajtani. Az Image Capture, mint ahogy neve is mondja rögzít egy adott pillanat képét és képes azt elmenteni, illetve a Video Capture, amely videók rögzítésére szolgál.

Az arcok azonosításához további kódkönyvtárakra van szükségünk. A Tensorflow Lite [17] biztosítja a mesterséges intelligenciát, aminek segítségével képesek vagyunk modellek betanítására. A Tensorflow Lite modell számos előnyt nyújt, mint például a csökkentett méretet, továbbá a gyors hozzáféréssel rendelkezik, illetve az adatokhoz közvetlenül hozzá lehet férni külön elemzési vagy kicsomagolási lépés nélkül. A Lite modell lehetővé teszi, hogy korlátozott számítási és memória forrásokkal rendelkező eszközökön hatékonyan futtasson bármely tanuló modell. A Tensorflow Lite több lehetőséget biztosít modellek generálására:

- Létező Tensorflow Lite modell használata
- Saját Tensorflow Lite modell készítése a Tensorflow Lite Model Maker segítségével

A továbbiakban bemutatásra kerül az arc azonosításának logikája az előbbiekben ismertetet könyvtárakkal [18]. Ennek lépései az 3.3.1.1. ábra mutatja be:



3.3.1.1. ábra: Egy arcfelismerő rendszer csővezetéke [18]

1. Érzékeljük az arcot.
2. Az arc transzformálása azért, hogy a szemek mindig ugyanabban a pontban legyenek.
3. Az arc újra méretezésé, hogy a bemeneti képek nagysága egyforma legyen és kielégítse a Deep Learning modell feldolgozási méretét.
4. Deep Neural Network, azaz a mély neurális háló

Az első lépés az arcok felismerése, amit remekül megold az ML Kit Face Detection. A process() függvény meghatározza a bemeneti képen észlelt arcokat és visszaadja azokat Face objektumként. Ezt követően a képek transzformálása szükséges, mivel minden egyes érzékelt arc szeme ugyanabba a pontba kell essen, így biztosítva a gépi tanulás minél nagyobb pontosságát. Ennek érdekében Bitmap-eket használunk és azokat transzformáljuk. A második lépés után a talált arcok újra méretezésé következik. A neurális háló bizonyos méretű képeket képes feldolgozni, mivel úgy van kialakítva, hogy ne használjon fel sok erőforrást egy bejövő kép azonosítására. Az utolsó lépésben a gépi tanulás segítségével visszakapunk egy vektort, amit az arc beágyazási pontjának nevezünk. Ez a beágyazás úgy jön létre, hogy vesszük a két F1 és F2 arc közötti hasonlóságot, amely egyszerűen kiszámítható az E1 és E2 beágyazások közötti euklideszi távolságaként:

$$Similarity(F1, F2) = \|DNN(F1) - DNN(F2)\|_2 = \sqrt{\sum_{i=1}^{D}(E1_i - E2_i)^2}$$

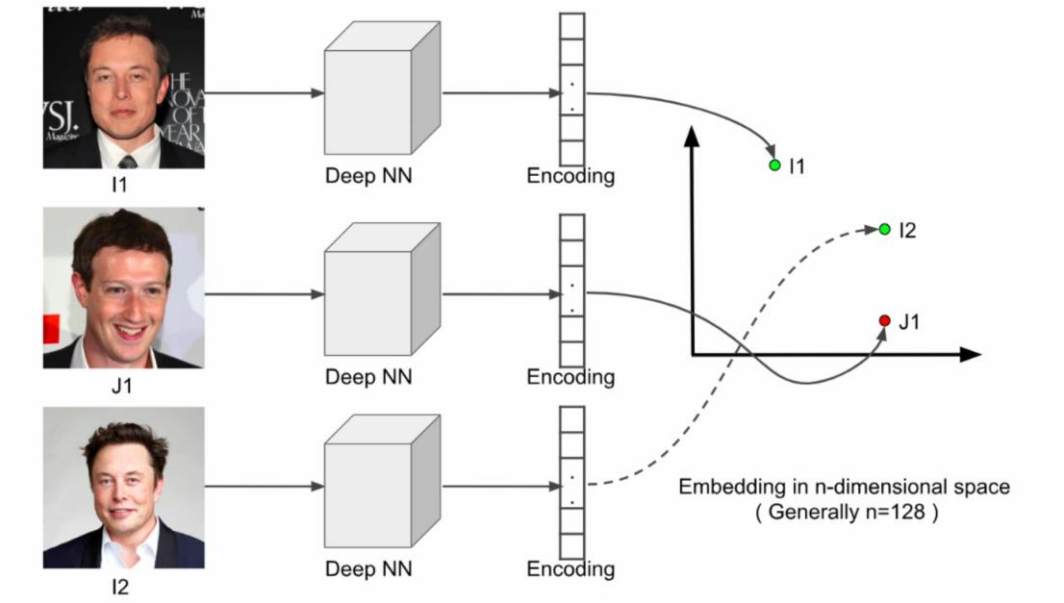
3.3.1.2. ábra: Az arcok közötti hasonlóság kiszámítható a beágyazódások közötti euklideszi távolságként [18]

Az 3.3.1.2. ábra illusztrálja az egyenletet, amivel két arcot lehet összehasonlítani úgy, hogy kiszámítjuk a hasonlóságát, majd ellenőrizzük valamilyen küszöbértékhez képest. Ha a kiszámított érték alacsony, akkor azt mondhatjuk, hogy mindkét arc ugyanattól a személytől származik.

3.3.2 Az arc azonosítása

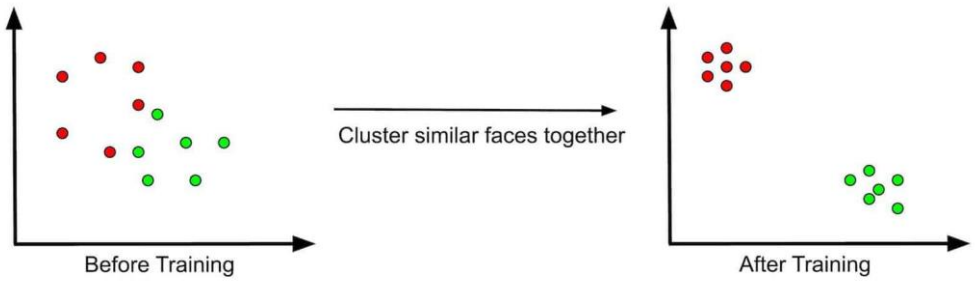
A mély tanulás érdekében, a metódusnak meg kell tanulnia egy nagydimenziós jellemzőteret, amely képes megkülönböztetni a személyek arcát. Ha megvan a betanított modell, akkor minden archoz egyedi vonásokat generálhatunk, így az azonosítás érdekében össze tudjuk hasonlítani az új arcok jellemzőit az ismert arcokkal. Abban az esetben, ha új személyt szeretnénk hozzáadni az ismert arcok adatbázisához, egyszerűen meghatározzuk a jellemzőket és hozzáadjuk az adatbázishoz. [19]

A konvolúciós neurális hálózatok (CNN-ek) [20] nagyon jó tulajdonság kinyerők, mivel ez a mély tanuló architektúra közvetlenül tanul az adatokból, így nincs szükség manuális tulajdonságok meghatározására. Az első lépés egy arcábrázolás létrehozása a bemeneti arcképekből. Tegyük fel, hogy van egy CNN-ünk, amely egy arcképet vesz bemenetként és egy „n” számból álló tömböt állít elő kimenetként. Az alábbi ábrán láthatjuk a CNN működését n = 2 esetében.



3.3.2.1. ábra: Jellemzőtér vizualizációja [19]

A 3.3.2.1. ábra bemutatja az egyes képekhez generált kódolásokat és illusztrálja, hogy hogyan néz ki egy 2-dimenziós jellemzőtér. A grafikonon látható pontokat arcbeágyazódásnak nevezzük, mivel az arc pontjai egy jellemzőtérbe vannak beágyazva. Ezek a beágyazások az adatbázisban lévő minden arcképhez létrejönnek, így mutatva az arcok egymáshoz való közelségüket. Abban az esetben, ha egy képzetlen hálózatot használunk az ugyanazon személy képeihez generált arcbeágyazások (I1 és I2) közeliek, vagy távoliak lehetnek. Még az is előfordulhat, hogy az azonos arc egyik pontja közelebb áll egy másik személyhez, mint ahogy az ábra is mutatja az I2 és J1 ponttal. A modell tanításának célja egy jellemzőtér megtanulása, ahol a zöld pontok egymáshoz közel és a piros pontoktól távol helyezkednek el, mindez úgy, hogy fordítva is igaz legyen. Így az első személy összes arcbeágyazása közel lesz egymáshoz és távol a másik arc beágyazásaitól. Ezt a 3.3.2.2. ábra illusztrálja.

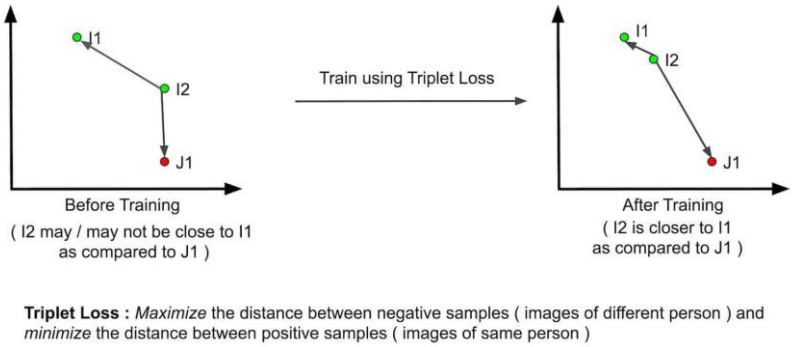


3.3.2.2. ábra: Tanulás előtt és tanulás után [19]

Annak érdekében, hogy képes legyen a modell a tanulásra, meg kell mondani a hálózatnak, hogy milyen jól teljesít, hogy a következő iterációban jobban teljesítsen. Így definiálunk egy veszteségi mérőszámot, amely a hálózat mutatójaként szolgál arra vonatkozóan, hogy mennyire képes elérni a célt. Összesen két feladatot szeretnénk megvalósítani:

- Hozza közelebb ugyanazon személy arcát.
- Tolja távolabb a különböző személyek beágyazásait.

A veszteség mutatónak jeleznie kell a hálózat számára, hogy csökkentse az I1 és I2 közötti távolságot és növelje az I2 és J1 közötti távolságot. Ez egy neurális hálózat tanítása során a kései tanítással történik. Az 3.3.2.3. ábra mutatja, hogy mi történik a veszteség mutató tanulása után.



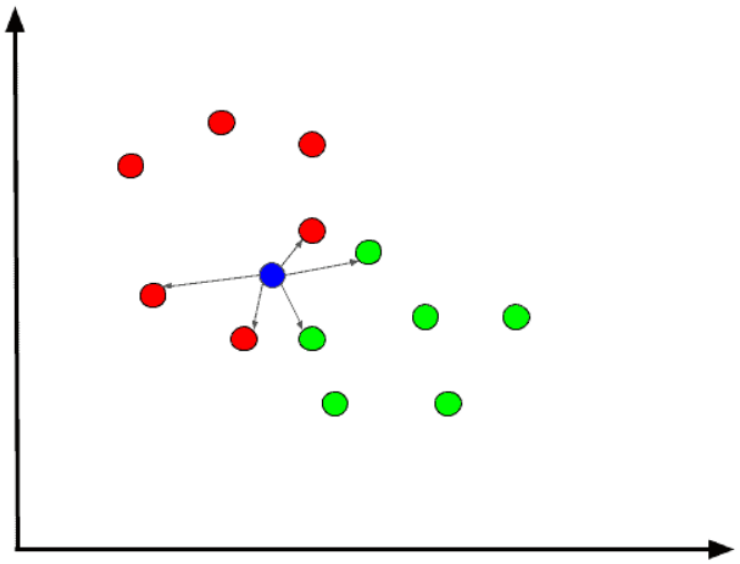
3.3.2.3. ábra: Veszteség mutató tanulás előtt és tanulás után [19]

Ezt a veszteség mutatót jogosan nevezik hármas veszteségnek, ami az egyik legszélesebb körben használt veszteség mutató az arcfelismeréshez. Így a CNN-eket jellemző adatgyűjtést és hármas veszteségmérőt használva a hálózat képes arra, hogy megtanuljon egy arcbeágyazási teret, ahol ugyanazon személy arcai össze vannak csoportosítva és a különböző személyek arcai el vannak választva. Az így kapott beágyazásoknak ki kell számítani a távolságát az adatbázisban található összes többi pontjától. A megjósolt személy az lesz, akinek a beágyazása a legközelebb áll az új arc beágyazásához. Az ehhez az összehasonlításhoz használt leggyakoribb távolságmérték az L2 távolság [21], amely egy számítási módot ad az n-dimenziós tér két pontja közötti euklideszi távolságra.

Vannak jobb módszerek is a következtetések levonására. Az eddigiekben vázolt példa alapján az új beágyazási pontokat összehasonlítjuk az adatbázisban található már rögzített

összes ponttal, ami nagy adatkészlet esetén lassú lehet. Az SVM [22] segítségével betanítható egy több osztályú osztályozó úgy, hogy az arcbeágyazásokat bemenetként használjuk. Ez azt jelenti, hogy minden osztály más-más személynek felel meg, így amikor egy új arcbeágyazást szeretnénk besorolni, egyszerűen összehasonlíthatjuk a támogatási vektorokkal és megjósoljuk, hogy melyik osztályhoz tartozik az új személy. Ez nagymértékben csökkenti a számítási költségeket, mivel nem kell összehasonlítania az új beágyazást az adatbázis több száz vagy több ezer beágyazásával. Bár az SVM megközelítés gyorsabb, van egy hátránya. Az SVM egy parametrikus gépi tanulási módszer, ami azt jelenti, hogy ha új személy kerül az adatbázisba, előfordul, hogy a régi paraméterek nem működnek, ezért újra kell képeznünk az SVM modellt. Erre a problémára nyújt megoldást a k-Nearest Neighbour (k-NN) [23] algoritmus. A k-NN az egyik legegyszerűbb gépi tanulási algoritmus, mivel nem kötelezi mindenáron a távolság kiszámítását a következtetési időpontban, hanem minden új pontnál csak összehasonlítja a "k" legközelebbi szomszédokat és többségi szavazási sémát alkalmazva hozza meg a döntést.

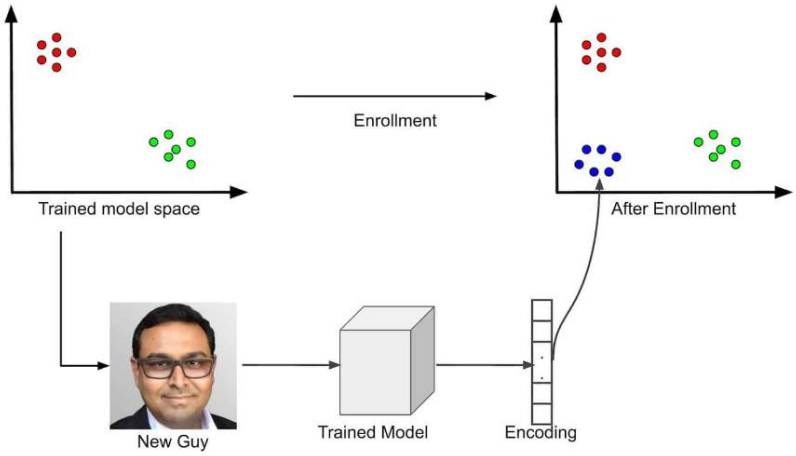
Vegyük például az alább látható beágyazásokat. Ha k = 5-öt használunk, akkor a kék pontot (új beágyazás) összehasonlítjuk a szomszédjaival és közülük többségi szavazást veszünk. A szomszédok közül hárman a piros osztályra, ketten a zöldre szavaznak, így a végső jóslat piros. Ezt vizualizálja a 3.3.2.4. ábra.



3.3.2.4. ábra: Jóslás k-NN-t használva [19]

Új személyek felvételekor ugyanazt a modellt tudjuk használni, nem kell minden alkalomkor újra generálni beágyazási területet. Mivel a modell már betanított és egy másik személy esetén a generált beágyazások messze eltérnek az adatbázisban lévő többi személyétől,

így csak ennek megfelelően frissítjük az ismert arcok adatbázisát. Az új személy kezelését az 3.3.2.5. ábra szemlélteti.



3.3.2.5. ábra: Új személy kezelése [19]

3.3.3 Mély neurális háló kiválasztása

A képlet megismerése után az azonosítás kulcs szereplőinek meghatározása történik. A kutatás alatt három lehetséges mély neurális hálózatot vizsgálok. Először egy a python kódnyelven létrehozott nn4.small2 [24] nevű előtanult modellt tanulmányozom. A modell körülbelül 14 FPS átviteli sebességet ér el egy MacBook Pro-n, továbbá a modell teljesítménye egy 8 magos 3,70 Ghz-es processorral 58,9 ms/frame. A modell további kérdések vett fel miszerint belefér-e az okostelefon memóriai határába, ugyanis az nn4.small2 több mint 30 MB fájl mérettel rendelkezik, mindemellett nem lehet a pontosságát megítélni, mivel vannak korlátai és hátrányai.

A következő a FaceNet [7], egy Google fejlesztésű modell, amely a legkorszerűbb eredményeket ért el. A mintadarab nagyon jó implementáció rendelkezik, valamint az előre betanított modellt már publikálták szabad felhasználásra. Az alábbi cikkben teszteket végeztek el a modellen, amiket a továbbiakban vázolók. Az első tesztben a beágyazások megegyeztek az eredeti modellek megfelelőivel, illetve azt is észre lehetett venni, hogy a Lite verzió sokkal könnyebb és gyorsabb. A következő lépésben importálni kellett a Lite modellt az Android Studio projektbe. A tapasztalat alapján, a mintadarab jól működik, de körülbelül 3,5 másodpercbe telik mire megállapítja program az arcok között lévő különbséget egy Google Pixel 3-as eszközön. A következtetés az, hogy egy remek modell offline arcok

összehasonlításához, illetve az összehasonlítás garantáltan pontos, viszont az egyetlen probléma az, hogy a lassúsága miatt nem lehet valós időben használni.

Az utolsó modell, amit vizsgálok, a Watchdata Inc., Beijinh, China nevű cég munkája, a MobileFaceNets [8]. A cég sikeresen bemutatott egy nagyon hatékony CNN-modellt (Convolutional Neural Network - Konvolúciós Neurális Hálózat), amelyet kifejezetten a mobileszközökre terveztek, megvalósítva a nagy pontosságú valós idejű arc-ellenőrzését. Lenyűgöző sebességet értek el nagyon nagy precizitással mindössze 4,0 MB-os fájlméretű modellben. Ez a méret tökéletes egy mobilalkalmazáshoz.

3.4 Kiválasztási szempontok

A cél teljesítésének érdekében kiválasztási szempontokat vázolok fel, annak érdekében, hogy a legmegfelelőbb technológia mellett tudjak dönteni. A tanulmány elején letisztáztam, hogy Android eszközre tervezem az alkalmazást, így teljesítve a könnyen elérhetőséget. Az első kérdés, amit felvettek a vizsgált gépi látásnál, hogy lehet-e futtatni az Android platformon. Fontos, hogy valós időben mindemellett offline működjön az arcok azonosítása, ezért felül kell vizsgálni az összes algoritmust, hogy beágyazható-e egy applikációba.

A második szempont a teljesítmény és az erőforrásigény. Annak érdekében, hogy bármely eszközön futtatható legyen a program, plusz elérje a megfelelő gyorsaságot és megbízhatóságot, a rendszernek az erőforrás igénye alacsony, mindemellett a teljesítménye magas kell legyen. Abban az esetben, ha a memória és processzor igénye magas a gépi látásnak, a modell tanulási ideje növekedhet ezzel lassítva az alkalmazást. Túl magas értékek esetén előfordulhat, hogy nem futtatható a technológia az eszközön.

A harmadik szempont, ami alapján vizsgálom a mesterséges intelligenciákat, a plusz funkciók telepítésének egyszerűsége. Mivel egy olyan alkalmazás megalkotása a cél, amely nem csak az alap arcazonosítás funkciókkal rendelkezik, hanem további információkkal tudja kiegészíteni a felhasználók mindennapjait, mint például jött-e posta vagy elvitték-e már a szemetet, így vizsgálom, hogy a technológiát mennyire lehet kiegészíteni tárgyak felismerésére.

3.5 Döntés

A döntés során az előző 3.4 pontban felvetett kérdésekre keresem választ, majd ezek alapján választom ki a legalkalmasabbat.

Az első szempont, hogy a technológia telepíthető Android mobil eszközön, hiszen, ha ez nem lehetséges, akkor feleslegesen vizsgáljuk tovább a módszert. Az ML Kit és TF Lite nem vett fel ezzel kapcsolatban kérdést, ugyanis ezek a könyvtárak Android eszközre lettek optimalizálva, így telepítésük egyszerű, kezelésük könnyű. Az OpenCV veti fel a kérdést, hogy lehetséges-e, amelyre a válasz igen, csak bonyolult és nem programozó barát [25]. Elsősorban le kell tölteni az OpenCV Android könyvtárát, amit az Android Studio-ban egy modulként kell beimportálni. Ezzel még nincs gond, azonban további fejfását tud okozni a különböző error-ok kezelése, illetve a további teendők miszerint külön natív könyvtár hozzáadása is szükséges.

A megfelelő teljesítmény elérése érdekében összehasonlítottam a két ismertetett metódust hatékonyság és kapacitás terén. Az OpenCV-ben a két vizsgált metódus közül a Fisherface produkált jobb eredményeket, így azt hasonlítom össze az ML Kit-tel és a MobileFaceNet-tel. Míg a Fisherface 83,87% pontosságot ad, addig a MobileFaceNet 99,55%-ra képes. Ez jelentős különbség, ugyanis a MobileFaceNet majdnem a 100%-os precizitást tud elérni, így messze megbízhatóbb, mint a Fisherface. A gyorsaság tanulmányozásával megállapítható, hogy a Fisherface jobb jóslási sebességgel szolgál, mint a TF Lite modell. Az OpenCV technológia 7,5 milliszekundum sebességre képes, ez több mint háromszor gyorsabb, mint a MobileFaceNet.

A harmadik szempont alapján végigvizsgálom a plusz funkciók telepítésének lehetőségét és egyszerűségét. Az ML Kit egy remekül felszerelt SDK, amely már magában tartalmazza a tárgyak érzékelését, így ez biztosítja a legkönnyebben a program további funkciókkal való kiegészítését. Az ML Kit Object Detection [26][26][26][26] szintén a CameraX [16] analizis függvényét használva előre betanított TF Lite modellek és azok beágyazással képes megkülönböztetni a tárgyakat, így a programban egy pár sor írásával elérhetővé válik például a kukásautók érzékelése. Mivel az arcazonosítás és a tárgyak felismerése közösen párosul az ML Kit-ben, ezért biztosabb megoldást ad, mint az OpenCV. Ahhoz, hogy az OpenCV-vel az arcokon kívül tárgyakat is érzékelni és megkülönböztetni tudjunk az alkalmazásunkat egy újabb technologiával kell kiegészíteni. A YOLO egy olyan algoritmus, amely neurális hálózatokat használ a valós idejű objektum észlelés biztosítására. A gyorsasága és pontossága miatt népszerű, így különféle rendszerekben használták már, mint például közlekedési jelzések, emberek, parkolóórák és állatok észlelésére. Az algoritmus további kérdéseket vet fel, mint hogy futtatható-e Android készüléken, illetve a gyengébb eszközök elbírják-e párhuzamosan az OpenCV arcazonosítást és a YOLO könyvtárakat. [27][28]

A vizsgálat végén arra a következtetésre jutottam, hogy az ML Kit könyvtárat választom egy előre betanított TF Lite modellel. Ez a gépi látás kielégíti azt, amire nekem szükségem van. Biztosítja az egyszerű Android alkalmazás készítését, ugyanis olyan megoldássokkal rendelkezik, amelyek a programozó munkáját könnyítik meg. A CameraX és az ML Kit Android eszközökre van optimalizálva, így futtatása és megbízhatósága megkérdőjelezhetetlen. Az előre betanított TF Lite modell biztosítja az offline, valós idejű működést, illetve a kevés erőforrás igénye miatt kielégíti azt a feltételt, hogy a program képes legyen régi, gyenge eszközökön is jól futni. Végül az ML Kit magába foglalja a tárgyak azonosítását is, így az alkalmazást egyszerűen lehet további funkciókkal kiegészíteni.

4 Prototípus elkészítése

A tanulmány validálásának érdekében elkészítek egy prototípust, amelyben megvalósítom az előző pontban ismertetett és választott technológiát. Bemutatom az implementálás folyamatát és szemléltetem a fejlesztés során felmerülő problémákat.

4.1 CameraX és ML Kit implementálása

A CameraX implementációja egyszerű, ez egy könnyen kezelhető könyvtár remek útmutatókkal. A laza kezelés érdekében létrehozok egy CameraManager osztályt, ami egy helyen kezeli a kamera életciklusait. A működésének érdekében a ProcessCameraProvider életciklusára, amit a CameraX biztosít, fel kell iratkoztatni a kamera moduljait. A prototípus során használom a könyvtár három fő modulját, mint például a Preview-t, ami egy élő képet ad a kamera képről. Méretezése engedett, így egy fragment layout felépítése könnyű. A prototípusban alkalmazom az ImageCapture modult a pillanatképek rögzítése érdekében, továbbá felhasználom az egyik legfontosabb modult a tanulmány szempontjából az Image Analysis-t. Ez az egység biztosítja a bejövő kamerakép analizálását egy analiyzer függvényben. Az érzékelés folyamatát itt veszik át az ML Kit különböző érzékelői. A mesterséges intelligencia könyvtár könnyen telepíthető, csak a megfelelő plugin-t hozzá kell adni a projekthez és egy ahhoz adott Detector-t kell létrehozni a feladathoz szükséges beállításokkal.

4.2 Tensorflow arcazonosítás megvalósítása

Az személyek azonosításához a 3.3-as pontban ismertettet elméletet programozom le, hogy megválaszoljam a tanulmány során felvetett kérdéseket.

4.2.1 Arcazonosítás

Ahhoz, hogy az arcokat képesek legyünk a mély neurális háló segítségével azonosítani, először át kell formálni őket. Képek esetén a legkönnyebben Bitmap-ekkel dolgozunk, így a CameraX-től kapott képet Bitmap-é formáljuk, majd a kamera döntése szerint forgatjuk, hogy minden arc egy vonalban legyen. Mivel az ML Kit Face Detection biztosítja az arc körvonalát egy határoló keretben, így könnyen le tudjuk kérni tőle és egyszerűen ki tudjuk vágni az elforgatott képből az arcot. Az utolsó formázási lépés a kép méretezése. A mély tanuló egy 112*112-es méretezésű képet vár el, így a Bitmap-et erre a méretre formáljuk. Ezzel a három

lépéssel teljesül a mély tanuláshoz szükséges feltétel, miszerint minden képnek egyforma méretűnek kell lennie, illetve a szemeknek ugyanabban a pontban kell elhelyezkedniük. [30]

Az így kapott Bitmap átadásra kerül egy függvénynek, ami megállapítja, hogy ismeri e az arcot. A függvény első lépése a normalizálás. Erre azért van szükség, mert ez biztosítja, hogy minden bemeneti paraméter hasonló adat eloszlású legyen. Ez gyorsítja a hálózatot tanítás közben. A normalizálás a következőképpen történik:

```
val imgData = ByteBuffer.allocateDirect(1 * inputSize * inputSize * 3 * 4)
imgData.order(ByteOrder.nativeOrder())
intValues = IntArray(inputSize * inputSize)
bitmap.getPixels(intValues, 0, bitmap.width, 0, 0, bitmap.width, bitmap.height)
imgData.rewind()
        for (i in 0 until inputSize) {
            for (j in 0 until inputSize) {
                val pixelValue = intValues[i * inputSize + j]
                if (isModelQuantized) {
                    imgData.put((pixelValue shr 16 and 0xFF).toByte())
                    imgData.put((pixelValue shr 8 and 0xFF).toByte())
                    imgData.put((pixelValue and 0xFF).toByte())
                } else {
                    imgData.putFloat(((pixelValue shr 16 and 0xFF) - IMAGE_MEAN) /
                                        IMAGE_STD)
                    imgData.putFloat(((pixelValue shr 8 and 0xFF) - IMAGE_MEAN) /
                                        IMAGE_STD)
                    imgData.putFloat(((pixelValue and 0xFF) - IMAGE_MEAN) /
                                        IMAGE_STD)
                }
            }
        }
```

Először létrehozunk egy ByteBuffer-t, hogy könnyebben el tudjuk tárolni a normalizált adatokat, majd kinyerjük a pixel értékeket a Bitmap-ből, ezután beletöltjük különböző formázással az ByteBuffer-be. Az ezt következő lépésben az arcbeágyazási pontok kiszámítása történik, kihasználva a MobileFaceNet Tensorflow Lite modellt:

```
val inputArray = arrayOf<Any>(imgData)
val outputMap: MutableMap<Int, Any> = HashMap()
embeedings = Array(1) { FloatArray(OUTPUT_SIZE) }
outputMap[0] = embeedings
MainActivity.tfLiteFace.runForMultipleInputsOutputs(inputArray, outputMap)
```

A beágyazási pontok egy Array<FloatArray> listába vannak eltárolva. Ezeket hasonlítja össze az algoritmus és keresi a k-NN alapján a legközelebbi pontot. A legközelebbi pont megállapítása az alábbi kóddal történik:

```
private fun findNearest(emb: FloatArray): List<Pair<String, Float>?> {
  val neighbour_list: MutableList<Pair<String, Float>?> = ArrayList()
  var ret: Pair<String, Float>? = null
  var prev_ret: Pair<String, Float>? = null
  for ((name, value) in registered) {
      val knownEmb: FloatArray = ((value.extra) as Array<*>)[0] as
                                  FloatArray
      var distance = 0f
      for (i in emb.indices) {
          val diff = emb[i] - knownEmb[i]
          distance += diff * diff
      }
      distance = Math.sqrt(distance.toDouble()).toFloat()
      if (ret == null || distance < ret.second) {
          prev_ret = ret
          ret = Pair(name, distance)
      }
  }
  if (prev_ret == null) prev_ret = ret
  neighbour_list.add(ret)
  neighbour_list.add(prev_ret)
  return neighbour_list
}
```

Az így visszaadott szomszédok listájából megállapítható, hogy melyik beágyazási ponthoz van közel az újonnan felismert arc. Ehhez egy távolság mértékegységet határozunk meg és abban az esetben, ha ez a távolság kevesebb mint egy, akkor ugyanarról az arcról beszélünk. Ez a kódban a következőképpen történik:

```
if (registered.size > 0) {
        val nearest = findNearest(embeedings[0])
        if (nearest[0] != null) {
            val name = nearest[0]!!.first
            distance_local = nearest[0]!!.second

            if (distance_local < 1.0f) {
                recognitionInfo.text = name
            } else {
                recognitionInfo.text = "Unknown"
            }
        }
    }
```

4.2.2 Arckezelés és mentés

Minden érzékelt arcot a FaceManeger osztály kap meg. Ez az osztály felelős az arcok kezeléséért, mentéséért és további események indításáért. A FaceManager jelenleg két fontos funkcióval rendelkezik, melyek közül az egyik az új arc felvétele. A mentés során kétféle technológiát használok. Az egyik a Room [31] adatbázist, mellyel képes vagyok elmenteni az arcokat képpel, dátummal és névvel, hogy ezek egy galériában visszatekinthetőek legyenek. Ez egy egyszerű megvalósítás, mivel a Room egy adatbázis-objektum-leképezési könyvtár, amely megkönnyíti az adatbázis elérését az Android alkalmazásokban. A másik mentési technika a legfontosabb az arcok kezelésekor. Itt kerülnek elmentésre a konkrét érzékelt arcok egy JSON fájlba. Ez egy HashMap létrehozásával történik, amit aztán a GSON segítségével egy JSON-be mentünk. Erre azért van szükség, mert ezzel a módszerrel könnyíteni lehet az alkalmazáson és nem függ internet kapcsolattól, mindemellett a tárhelyigényét alacsonyan lehet tartani. Ez a HashMap kerül mindig beolvasásra kamera indításakor, hogy az algoritmus érzékelni tudja a már elmentett arcokat. Ez a következőképpen történik:

```
private fun editJson(context: Context, map: HashMap<String?,
                                SimilarityClassifier.Recognition>) {
    val jsonString = Gson().toJson(map)
    val sharedPreferences = context.getSharedPreferences("HashMap",
                                AppCompatActivity.MODE_PRIVATE)
    val editor = sharedPreferences.edit()
    editor.putString("map", jsonString)
    editor.apply()
}
```

A második funkció, amit a FaceManager lát el, az az érzékelt arcok kezelése. Ez a következőképpen történik: Egy személyt akkor vesz érzékeltnek, ha a kamera az arcot 15 másodpercen belül legalább három másodpercig azonosít. Abban az esetben, ha ez megtörténik a rendszer névvel, vagy ha ismeretlen arcról van szó akkor név nélkül elmenti és egy dátumot társít hozzá a visszanézhetőség miatt. A mentés során ismét a Room rendszerét használom. Erre az időkorlátra azért van szükség, hogy ne riasszon be minden egyes arcra, ami elhalad a kamera előtt. Miután egy bizonyos arcról mentés készült, utána a program két percig nem kezd vele semmit, hogy ne legyen teletömve az előzmény.

5 Értékelés

5.1 Kutatás validálása

A kutatásom validálásának érdekében különböző Android okostelefonokon tesztelem a prototípust. Elsősorban a működést vizsgálom, hogy minden funkció ez elvárt módon működik-e. Továbbiakban az erőforrások használatát ellenőrzöm, ami azt jelenti, hogy megnézem a CPU, illetve a memória használatot és méréseket végzek az FPS számmal kapcsolatban. Az FPS-t, azaz a másodpercenkénti képkockák számát egy külön módszerrel lehet megmérni, ami azt jelenti, hogy használni kell a CameraX Image Analysis modulját és egy saját számlálót szükséges elhelyezni a kódban. A kalkuláció módszerét az alábbi kód mutatja. Az előzőekben említett mérőszámokat egyszerűen ki lehet olvasni az Android Studio Profiler segítségével, ami monitorozza az eszközt az alkalmazás futtatása közben.

```
val currentTime = System.currentTimeMillis()
     frameTimestamps.add(currentTime)
     while (frameTimestamps.size >= frameRateWindow) {
             frameTimestamps.removeFirst()
      }
     val timestampFirst = frameTimestamps.first() ?: currentTime
     val timestampLast = frameTimestamps.last() ?: currentTime
     framesPerSecond = 1.0 / (
         (timestampLast - timestampFirst) /
             frameTimestamps.size.coerceAtLeast(1).toDouble()
         ) * 1000.0
```

Az első tesztkészülékem egy Oneplus 8T eszköz, ami az erős/közép kategóriába tartozik, az ára körülbelül 160.000 forint. Nem egy könnyen elérhető eszköz, a kutatásom nem ezt a kategóriát célozza meg, azonban remek kiindulási értékeket ad az alacsonyabb értékű eszközökkel szemben. A Oneplus 8T egy 48 MP-es kamerával rendelkezik, belül pedig 3 x 2.84 GHz processzor, illetve 8 GB memória található. Maga az eszköz 2 éves, azonban ez a működésében nem látszik. Az első teszt során nem botlunk hibába, mivel minden funkció úgy működik rajta, ahogy kell. Az erőforrás vizsgálata közben megfigyelhető, hogy a CPU-t átlagban csak 24%-ban veszi igénybe, míg a memóriából körülbelül 350 MB-ot használ. Ezek

az adatok azt mutatják, hogy az erőforrást kis mértékben veszi igénybe. Kamera indítás után az FPS szám 30-at mutat, mivel a készülék ebben az állapotában még nem látott arcot, de abban a pillanatban amikor a látószögébe kerül egy személy az FPS szám leesik 12-re, majd 10 és 12 között ingadozik. Ezen látszik, hogy a mély tanulás közben igénybe veszi még az erősebb eszközt is, ez azonban szabad szemmel nem látható, így a felhasználói élményt nem rontja.

A második teszteszköz egy Xiaomi Redmi Note 9 Pro termék, ami már hat éve használatban van. Ez az készülék egy remek alanya a tanulmánynak, ugyanis egy régi, már használaton kívüli termék. Újonnan az ára 90.000 forint, azonban használati ideje miatt elértéktelenedett. A Redmi Note 9 Pro egy 64 MP-es kamerával rendelkezik, belül 2 x 2.3 GHz processzor és egy 6 GB-os memória található. A prototípus remekül működik az eszközön, az arcokat felismeri és azonosítja úgy, hogy a felhasználói élményből nem veszít. Az erőforrás vizsgálatnál megfigyelhető, hogy 30%-ban veszi igénybe a processzort és a memória használata igen alacsony, összesen 200 MB körül mozog. Az FPS szám a kamera indítása után 20 és 30 között ingadozik. Egy arc érzékelése után az FPS egy nagyot esik és azonosítás közben átlagban négy és hat között ugrál. Ezen látszik, hogy igénybe veszi az eszközt azonban ez működésben nem látszik meg. Ez a szám már vészesen kicsi, de az érzékelést és az azonosítás pontosan elvégzi.

Az utolsó teszteszköz egy Huawei P30 Lite, amit már három éve használnak. A felhasználója annakidején 75.000 forintért vásárolta, tekintettel a korára és a kihasználtságára, körülbelül fele annyit érhet most, ami olcsóbb, mint a legtöbb ma kapható okoskamera. A P30 Lite egy 48 MP-es kamerával rendelkezik, belül pedig 4 x 2.2 GHz Cortex-A73 processzorral és 6 GB-os memóriával szerelték fel. A prototípus elindítása után észrevehető, hogy a gombnyomásra lassabban reagál, a kamera előképe néha akadozik. Az új arc felvétele közben az alkalmazás lelassul és idő kell neki még végrehajt egy műveletet. Ennek ellenére a felhasználói élményt csak kis mértékben rontja, mivel a funkciói működnek. Az erőforrásokat vizsgálva, azonban csak az FPS számban változott az előző készülékekhez képest. A processzort 35%-ban használja, míg átlagban 300 MB memóriát vesz igénybe. Az FPS a kamera indítását követően alacsony számot mutat, hat és nyolc között ingadozik. Egy arc érzékelését követően ez a szám leesik és átlagosan 2-3 között mozog. A prototípus érezhetően lassabban írja ki a képernyőre a felismert arcot, azonban az azonosítást jól elvégezte.

	Oneplus 8T	Xiaomi Redmi Note 9 Pro	Huawei P30 Lite
Processzor	3 x 2.84 GHz	2 x 2.3 GHz	4 x 2.2 GHz
Operatív memória	2-8 GB	2-6 GB	2-6 GB
Processzor használat	~24%	~30%	~35%
Memória használat	350 MB	200 MB	300 MB
FPS - arc nélkül	30	20-30	6-8
FPS - arc érzékelésével	10-12	4-6	2-3

5.1.1. táblázat: Tesztkészülékek összehasonlítása

A táblázat alapján megállapítható, hogy vannak különbségek az eszközök között. Míg egy drágább eszköz simán működteti a prototípust, addig a gyengébb, öregebb eszközök, bár működésének megfelelően, azonban lassabban futtatják azt. A nagy különbségek az FPS számban mutatkoznak meg, az alap erőforrás használatban, mint a processzor vagy a memória, nem jön létre nagy eltérés. Ha egy eszközt vizsgálunk egyszere kimutatható, hogy sok erőforrást igényel egy személy azonosítása, ugyanis még az erős készülékeknek is csökken az FPS száma arc érzékelés közben. Megfigyelhető, hogy a két gyengébb eszközön a képkocka szám kritikus pontokat ér el, amivel a működést lassíthatják, illetve a felhasználói élményt ronthatják, ennek ellenére a tesztelés során az arcazonosítás működik.

Összességében a tanulmány túllép a ma kapható okoskamerák többségén, mivel egy olyan megoldást biztosít, ami elérhető, felhasználóbarát, bármely otthonban megtalálható, már nem használt Android telefonon futtatható és az arc azonosítás mellett képes egyéb funkciókkal ellátni a felhasználót. Mint, ahogy a Problémafelvetés1.1-es fejezetben is vázoltam, a kutatás egy megoldást biztosít, amivel biztonságosabbá tehetjük otthonunkat.

5.2 Továbbfejlesztési lehetőségek

A tanulmány tovább vihető több irányba is. Hetente jelennek meg újabb arcazonosító modellek, amelyek egyre pontosabban képesek érzékelni az arcokat. Abban az esetben, ha az új technológia jobb eredményeket mutat, érdemes lehet a régi modell lecserélésén elgondolkodni.

Az arcok érzékelése mellett plusz statisztikákkal ki lehet egészíteni a felhasználónak adott információkat, mint például egy nap hányszor érzékeli ugyanazt az arcot, melyik napszakokban jellemző az ismert arc érzékelése vagy feltűnik e ugyanaz az ismeretlen arc ugyanazon a napon több alkalommal is.

Számos okosotthon funkcióval is ki lehet bővíteni az prototípust. Az arcokhoz hozzá lehet rendelni feladatokat, miszerint, ha érzékeli a tulajdonos arcát a kapuban, akkor a rendszer szól az okostelevíziónak, ami elindítja kedvenc filmünket, vagy akár szól kávégépnek, amely főz egy kávét. lefőz egy kellemes friss kávét, mire beérünk otthonunkba.

A könnyebb információ hozzáférés érdekében, az alkalmazáshoz Google Assistance funkciókat lehet telepíteni, így a felhasználó egy kérdéssel hozzá tudna jutni az adatokhoz, mint például, hogy jött e postai küldemény.

Egy további biztonsági funkció lehet, hogy az arcokon kívül érzékelni és leolvasni lehetne az autók rendszámát, melyek a házunk előtt állnak meg. Így vissza lehet nézni és figyelni a forgalmat és a biztonságot otthonunk előtt.

Tekintettel arra, hogy a prototípus Android alapokon íródott, így egyszerűvé válik egy Android alapú raspberry pi-re a kitelepítése. Ezáltal megvalósítható egy külső eszköz, amely az internet segítségével kommunikál az alkalmazással.

6 Irodalomjegyzék

[1] hvg.hu, „Nagy baj van: egy hiba miatt 23 ezer magyar web- és biztonsági kamera fölött lehet átvenni az irányítást”, szeptember 2021, [Online]. Available: https://hvg.hu/tudomany/20210921_hikvision_biztonsagi_kamera_webkamera_serulekenyseg_hiba

[2] arlo.com, [Online]. Available: https://www.arlo.com/en-us/cameras/pro/arlo-pro-4.html

[3] Xiaojun Bai, Feihu Jiang, Tianyi Shi, Yuang Wu, „Design of Attendance System Based on Face Recognition and Android Platform”, szeptember 2020, [Online]. Available: https://ieeexplore.ieee.org/document/9239722

[4] Zongying Ou, Xusheng Tang, Tieming Su, Pengfei Zhao, „Cascade AdaBoost Classifiers with Stage Optimization for Face Detection”, [Online]. Available: http://link.springer.com/content/pdf/10.1007/11608288_17.pdf

[5] Zhenhua Guo, Lei Zhang, David Zhang, „A Completed Modeling of Local Binary Pattern Operator for Texture Classification”, március 2010, [Online]. Available: https://ieeexplore.ieee.org/abstract/document/5427137

[6] Salinda Hettiarachchi, „Analysis of different face detection and recognition models for Android”, 2021, [Online]. Available: https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1574943&dswid=1963

[7] Florian Schroff, Dmitry Kalenichenko, James Philbin, „FaceNet: A Unified Embedding for Face Recognition and Clustering,” 2015, [Online]. Available: https://www.cv-foundation.org/openaccess/content_cvpr_2015/html/Schroff_FaceNet_A_Unified_2015_CVPR_paper.html

[8] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam, „MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications”, április 2017, [Online]. Available: https://arxiv.org/abs/1704.04861

[9] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, Alex Alemi, „Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning”, február 2016, [Online]. Available: https://arxiv.org/abs/1602.07261

[10] NEC New Zealand, “Face Detection vs Facial Recognition – what’s the difference?”, június 2022. [Online]. Available: https://www.nec.co.nz/market-leadership/publications-media/face-detection-vs-facial-recognition-whats-the-difference/

[11] OpenCV, „Introduction”, [Online]. Available: https://docs.opencv.org/4.x/d1/dfb/intro.html

[12] OpenCV, “Face Recognition with OpenCV”, [Online]. Available: https://docs.opencv.org/3.4/da/d60/tutorial_face_main.html

[13] Ismail Aliyu, Muhammad Ali Bomoi, Maryam Maishanu, „A Comparative Study of Eigenface and Fisherface Algorithms Based on OpenCV and Sci-kit Libraries Implementations”, február 2022, [Online]. Available: https://www.mecs-press.org/ijieeb/ijieeb-v14-n3/IJIEEB-V14-N3-4.pdf

[14] Google Developers, „ML Kit”, [Online]. Available: https://developers.google.com/ml-kit

[15] Google Developers, „ML Kit Face Detection”, október 2022, [Online]. Available: https://developers.google.com/ml-kit/vision/face-detection

[16] Google Developers, „Android Jetpack CameraX”, október 2022, [Online]. Available: https://developer.android.com/training/camerax

[17] „Tensorflow Lite”, május 2022, [Online]. Available: https://www.tensorflow.org/lite/guide

[18] Esteban Uri. “Real time face recognition with Android + TensorFlow Lite”, június 2020. [Online]. Available: https://medium.com/@estebanuri/real-time-face-recognition-with-android-tensorflow-lite-14e9c6cc53a5

[19] Vikas Gupta, Satya Mallick, “Face Recognition: An Introduction for Beginners”, április 2019. [Online]. Available: https://learnopencv.com/face-recognition-an-introduction-for-beginners/

[20] “CS231n Convolutional Neural Networks for Visual Recognition”, [Online]. Available: https://cs231n.github.io/transfer-learning/

[21] Jason Brownlee, “Gentle Introduction to Vector Norms in Machine Learning”, február 2018. [Online]. Available: https://machinelearningmastery.com/vector-norms-machine-learning/

[22] Wikipedia, „Support vector machine”, [Online]. Available: https://en.wikipedia.org/wiki/Support_vector_machine

[23] Wikipedia, „Nearest neighbor search”, [Online]. Available: https://en.wikipedia.org/wiki/Nearest_neighbor_search#Approximation_methods

[24] OpenFace, “Models and Accuracies” [Online]. Available: http://cmusatyalab.github.io/openface/models-and-accuracies/

[25] Elvis Chidera, “A Beginner’s Guide to Setting up OpenCV Android Library on Android Studio”, augusztus 2018. [Online]. Available: https://medium.com/android-news/a-beginners-guide-to-setting-up-opencv-android-library-on-android-studio-19794e220f3c

[26] Google Developers, „ML Kit Object detection and tracking”, október 2022, [Online]. Available: https://developers.google.com/ml-kit/vision/object-detection

[27] Adrian Rosebrock , “YOLO object detection with OpenCV”, november 2018. [Online]. Available: https://pyimagesearch.com/2018/11/12/yolo-object-detection-with-opencv/

[28] Shantam Sultania “RealTime Object detection using YOLO and OpenCV”, január 2020. [Online]. Available: https://medium.com/analytics-vidhya/realtime-object-detection-using-yolo-and-opencv-8380e5537f2a

[29] Marcelo Rovai, “Real-Time Face Recognition: An End-To-End Project”, március 2018. [Online]. Available: https://towardsdatascience.com/real-time-face-recognition-an-end-to-end-project-b738bb0f7348

[30] Real-Time Face Recognition Android Github, [Online]. Available: https://github.com/atharvakale31/Real-Time_Face_Recognition_Android

[31] Google Developers, „Save data in a local database using Room”, október 2022, [Online]. Available: https://developer.android.com/training/data-storage/room

[32] Ryne Hager, „Why the hell are Google's Nest cameras so expensive?”, június 2021, [Online]. Available: https://www.androidpolice.com/2021/06/19/why-the-hell-are-googles-nest-cameras-so-expensive/

[33] Carl, „Are Smart Locks Really Worth The Investment?”, [Online]. Available: https://onesmartshelter.com/smart-locks-worth/

[34] Gary B. Huang, Manu Ramesh, Tamara Berg, Erik Learned-Miller, „Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments”, [Online]. Available: http://vis-www.cs.umass.edu/papers/lfw.pdf

[35] Bruno Veloso, João Gama, Rita P. Ribeiro, Pedro M. Pereira, „A Benchmark dataset for predictive maintenance”, július 2022, [Online]. Available: https://arxiv.org/abs/2207.05466