Matematikai és Informatikai Intézet

Digitális személyazonosítás mesterséges intelligenciával

Készítette
Kovács Gábor
Programtervező informatikus Bsc

Témavezető
Dr. Kovásznai Gergely
Tanszékvezető, egyetemi docens

Eger, 2025

Tartalomjegyzék

Bevezetés

A digitalizáció és a mesterséges intelligencia fejlődése az elmúlt évtizedekben alapjaiban változtatta meg mindennapi életünket és az üzleti világ működését. Az olyan technológiák, mint a mobilalkalmazások, a felhőalapú rendszerek és a mesterséges intelligencia, lehetővé tették, hogy az információk gyorsabban és hatékonyabban legyenek elérhetők, mint valaha. Ezek a technológiák nemcsak az ipari folyamatokat, hanem a személyazonosítás és az adatkezelés területét is forradalmasították.

A személyazonosítás hagyományos módszerei, mint például a papíralapú igazolványok, egyre inkább háttérbe szorulnak a digitális megoldásokkal szemben. Az e-személyi igazolványok és a biometrikus azonosítási technológiák, mint például az arcfelismerés vagy az ujjlenyomat-azonosítás, nemcsak kényelmesebbé, hanem biztonságosabbá is teszik az azonosítási folyamatokat. Az Európai Unió által bevezetett eIDAS és EUDI rendeletek is azt mutatják, hogy a digitális személyazonosítás a jövő egyik kulcsfontosságú területe.

A szakdolgozat célja egy olyan mobilalkalmazás fejlesztése, amely a modern technológiák, például a React Native, az Expo, a Node.js és a MongoDB segítségével lehetővé teszi a felhasználók számára személyazonosságuk digitális igazolását. Az alkalmazás egy neurális hálózatot használ az optikai karakterfelismerés (OCR) megvalósítására, amely képes a személyi igazolványokról készült képekből kinyerni a releváns adatokat. Az így kinyert információk egy biztonságos adatbázisban kerülnek tárolásra, biztosítva a felhasználók adatainak védelmét és a rendszer megbízhatóságát.

A dolgozat során bemutatom a személyazonosítás fejlődését, a digitális megoldások előnyeit és kihívásait, valamint a mesterséges intelligencia és a neurális hálók szerepét az adatfeldolgozásban. Részletesen ismertetem a projekt során használt technológiákat, a backend és frontend fejlesztési folyamatokat, valamint a rendszer tesztelését és dokumentálását. A cél egy olyan átfogó megoldás bemutatása, amely nemcsak technológiai szempontból innovatív, hanem a felhasználók számára is könnyen használható és biztonságos.[30]

1. fejezet

Személyazonosítás fejlődése és digitalizációja

1.1. A személyi igazolványok története

Ebben a fejezetben röviden végigmegyünk a személyazonosító igazolványok történetén. A *személyazonosítás* igénye évezredekre nyúlik vissza, hiszen a társadalmak mindig is szerették volna *hiteles módon* azonosítani tagjaikat. Az ókori birodalmakban *pecsétes levelek*, *ujjlenyomatos agyagpecsétek* és különböző azonosító jegyek szolgáltak erre a célra. A középkorban a *nemesi kiváltságokat* vagy *állampolgárságot* igazoló dokumentumokat használtak, például a pápai bullákat vagy a királyi rendeleteket. [1, 2]

A modern értelemben vett *személyi igazolványok* a *19. és 20. században* terjedtek el, amikor az államok egyre inkább szükségét érezték annak, hogy állampolgáraikat hivatalos dokumentumokkal azonosítsák. Magyarországon az első személyi igazolványokat a *második világháború* után vezették be, és azóta számos változáson mentek keresztül, mind biztonsági, mind technológiai szempontból. [3]

1.2. A digitális azonosítás megjelenése

A 21. században a *digitális technológia* fejlődésével a személyazonosítás egyre inkább az *elektronikus rendszerekre* helyeződött át. Az internet elterjedésével növekedett az igény a *biztonságos online azonosításra*, amely a hagyományos személyi igazolványok digitális megfelelőit hívta életre.

Számos országban, mint például Észtországban bevezették az *elektronikus személyi igazolványokat* (eID), amelyek *beépített chippel* rendelkeznek, és különböző *biometrikus adatokat* is tárolhatnak, mint *ujjlenyomat* vagy *arcfelismerési információ*. Ezek az azonosítási módszerek jelentősen javítják a biztonságot és megkönnyítik az online szolgáltatásokhoz való hozzáférést. [3]

1.3. A mobiltechnológia és az azonosítás

Az *okostelefonok* és a *mobilalkalmazások* terjedésével az azonosítás folyamata tovább egyszerűsödött. A felhasználók ma már egyetlen kattintással vagy *biometrikus azonosítással* (pl. *Face ID*, *ujjlenyomat-olvasó*) hitelesíthetik magukat különböző szolgáltatásokhoz.

A *mobiltechnológia* lehetővé tette a digitális személyazonosítás gyorsabb, kényelmesebb és biztonságosabb formáit. A *blockchain*[1] alapú azonosítási rendszerek pedig tovább fokozzák az *adatok védelmét*, lehetőséget adva a felhasználóknak, hogy nagyobb kontrollt gyakoroljanak személyes adataik felett.

1.4. A digitalizáció jövője a személyazonosításban

A jövőben a személyazonosítás módszerei tovább fejlődnek, egyre inkább az *automatizált* és *AI-alapú rendszerek* irányába. Az *arcfelismerés*, a *hangalapú azonosítás* egyre népszerűbbé válik.

A digitális személyazonosítás előnye, hogy *gyorsabb* és *hatékonyabb* az offline módszereknél, ugyanakkor komoly *adatbiztonsági kihívásokat* is felvet. A jövő fejlesztései során kulcsfontosságú lesz a *magánszféra védelme* és az *etikus adatkezelés* biztosítása.

A *digitális személyi igazolványok* az állampolgárok azonosításának egyre népszerűbb eszközei világszerte. Ezek az okmányok nemcsak a hagyományos, fizikai igazolványok elektronikus megfelelői, hanem további funkciókkal is rendelkezhetnek, például *online hitelesítésre*, *elektronikus aláírásra* vagy egyes állami és magánszolgáltatások elérésére.

Bár a digitális személyazonosítás rengeteg *előnyt* kínál, számos *kihívás* is társul hozzá, amelyek megoldása kulcsfontosságú a széleskörű elterjedéshez.

1.5. Előnyök

A *digitális személyi igazolványok* lehetővé teszik az azonosítás és hitelesítés *gyors* és *kényelmes* módját. Az állampolgárok anélkül igazolhatják magukat, hogy fizikai okmányt kellene magukkal hordaniuk, hiszen az azonosító adatok tárolhatók egy *mobiltelefonon* vagy egy *biztonságos szerveren*. Magyarországon a DÁP[2] valósítja ezt meg.

Például egy *e-személyi igazolvány* segítségével egy banki ügyintézés vagy egy online regisztráció néhány kattintással elvégezhető, míg a hagyományos személyazonosítás esetében papírokat kell kitölteni, aláírni és személyesen bemutatni.

[1] Az adatok „blokkokba” vannak tárolva, amelyek egymáshoz vannak láncolva, így egy folyamatos, biztonságos láncot alkotnak.

[2] Digitális Állampolgárság Program

A digitális személyazonosító igazolványok a legmodernebb *titkosítási technológiákat* alkalmazzák, így *nehezebben hamisíthatók*, mint a hagyományos plasztikkártyák. Egy jól megtervezett digitális rendszerben az adatok ellenőrzése és tárolása *szigorú biztonsági szabványok* szerint történik, csökkentve az illetéktelen hozzáférés vagy visszaélés esélyét. Az Európai Unióban az Európai Bizottság felelős a digitális személyazonosításra vonatkozó szabályozásért, melynek keretében az *eIDAS* és az *EUDI* rendeletek meghatározó szerepet töltenek be a biztonságos digitális azonosítási keretrendszer kialakításában. [4, 5]

Biometrikus azonosítókkal (például *arcfelismerés* vagy *ujjlenyomat*) kombinálva az e-személyik még biztonságosabbá válhatnak, hiszen a felhasználó azonosítása egyértelmű és nehezen másolható.[13]

A digitális személyi igazolványok hozzájárulhatnak a *papíralapú ügyintézés csökkentéséhez*, ami nemcsak a *környezetvédelmet* szolgálja, hanem a közigazgatási rendszerek *hatékonyságát* is növeli. A kevesebb nyomtatás, postázás és manuális adatfeldolgozás hosszú távon jelentős *költségmegtakarítást* eredményezhet az állam és a vállalatok számára is.

1.6. Kihívások

A digitális személyazonosító rendszerek egyik legnagyobb kihívása az *adatbiztonság*. Mivel ezek az igazolványok személyes adatokat tartalmaznak, kiemelt célpontjai lehetnek *kibertámadásoknak* és *adatlopásoknak*. Egy esetleges rendszerfeltörés vagy adatvédelmi incidens súlyos következményekkel járhat az érintettek számára.

Ennek elkerülése érdekében olyan technológiákat kell alkalmazni, mint a *többfaktoros hitelesítés*, az adatok *titkosítása* és a *decentralizált adattárolás*. Az *adatvédelmi jogszabályok* is szigorúan szabályozzák, hogy a digitális személyi igazolványokhoz kapcsolódó információkat hogyan lehet kezelni és tárolni.

Bár a digitális személyi igazolványok kényelmesek lehetnek a technológiailag fejlett országokban, nem mindenki fér hozzá megfelelő eszközökhöz vagy internetkapcsolathoz. Az *idősebb generációk*, a *technológiai ismeretekkel kevésbé rendelkezők* vagy a *hátrányos helyzetű régiókban élők* számára nehézséget jelenthet az új rendszer használata.

Emellett az *infrastruktúrának* is fel kell készülnie a digitális személyazonosítás kezelésére. Például egy digitális személyi igazolvány csak akkor használható széles körben, ha az intézmények és vállalkozások rendszerei *kompatibilisek* vele.

A digitális azonosítás *jogi szabályozása* még sok helyen gyerekcipőben jár. Az *eltérő nemzeti szabályozások* és az adatvédelmi törvények miatt nehéz olyan *univerzális rendszert* kialakítani, amely minden országban elfogadott és kompatibilis lenne a helyi előírásokkal.

Például egy olyan digitális személyi igazolvány, amely egy adott országban teljes

körű hitelesítésre képes, nem biztos, hogy egy másik országban is elfogadott. A *nemzetközi standardok* és *együttműködések* kialakítása kulcsfontosságú lehet a rendszer globális elterjedése szempontjából.

2. fejezet

A mesterséges intelligencia szerepe az adatfeldolgozásban

A *mesterséges intelligencia* (MI) forradalmasította az adatfeldolgozást és az automatizált döntéshozatalt. A hagyományos, manuális adatrögzítési módszerekkel szemben az MI-alapú rendszerek képesek *nagy mennyiségű információ gyors* és *pontos* feldolgozására, ami különösen hasznos a dokumentumok digitalizálása és azonosítási rendszerek fejlesztése terén.

2.1. Mi az az OCR (Optikai karakterfelismerés)?

Az *Optical Character Recognition* (OCR) egy olyan technológia, amely képes nyomtatott vagy kézírásos szövegeket *digitális formátumba* alakítani. Az OCR segítségével a dokumentumokról készült fényképeken található szöveg *gépileg olvasható* formátummá konvertálható, így az adatok feldolgozása és tárolása *automatizálható*.

Az OCR működésének lépései:

1. **Kép előfeldolgozása** – A képből eltávolítják a *zajokat* (például árnyékokat vagy torzításokat), hogy a karakterek élesebben felismerhetők legyenek.

2. **Karakterek felismerése** – Az MI-alapú OCR rendszerek általában *neurális hálók* segítségével azonosítják az egyes betűket és számokat.

3. **Szöveg átalakítása és értelmezése** – A felismerés után a szöveget *szerkezetileg elemzik*, hogy a megfelelő adatokat lehessen kinyerni belőle (például név, születési dátum, igazolványszám).

Az ilyen rendszerek ***gépi tanulással*** folyamatosan fejleszthetők: minél több dokumentumot dolgoznak fel, annál pontosabb lesz a felismerés és az adatkinyerés.

2.2. Hogyan segíthet az MI a dokumentumok feldolgozásában?

A mesterséges intelligencia a dokumentumfeldolgozás több aspektusában is jelentős segítséget nyújt:

- ***Automatizált adatkinyerés*** – Az MI-alapú rendszerek az igazolványokról gyorsan és pontosan kivonják az adatokat, csökkentve a manuális bevitel szükségességét.
- ***Hibajavítás és adatellenőrzés*** – Az MI képes felismerni és javítani a karakterfelismerési hibákat, valamint ellenőrizni az adatok érvényességét (pl. egy születési dátum valós lehet-e).
- ***Biztonsági ellenőrzések*** – Az MI algoritmusok kiszűrhetik a hamis vagy manipulált igazolványokat azáltal, hogy összehasonlítják azokat ismert mintákkal és adatbázisokkal.
- ***Adatvédelmi és titkosítási megoldások*** – Az MI segítségével érzékeny személyes adatok anonim módon tárolhatók és dolgozhatók fel, csökkentve az adatlopás kockázatát.

2.3. A mesterséges intelligencia jövője az adatfeldolgozásban

Ahogy az MI-technológiák egyre fejlettebbé válnak, az adatfeldolgozás pontossága és hatékonysága tovább javulhat. A jövőben várható fejlesztések közé tartozik:

- ***Még pontosabb OCR és természetes nyelvfeldolgozás*** (NLP) – Az MI képes lesz még bonyolultabb dokumentumokat is pontosan értelmezni.
- ***Valós idejű adatfeldolgozás*** – Az azonosítási folyamatok azonnali ellenőrzése és feldolgozása még gyorsabbá válhat.
- ***Blokklánc és decentralizált identitáskezelés*** – Az MI és a blokklánc kombinálásával a személyazonosító adatok még biztonságosabbá tehetők.

Összességében a mesterséges intelligencia egyre fontosabb szerepet tölt be a személyazonosító dokumentumok *digitális feldolgozásában*. Az *automatizálás*, a *pontosság* és a *biztonság* növelésével hozzájárul ahhoz, hogy a felhasználók kényelmesebben és gyorsabban intézhessék ügyeiket a digitális világban.

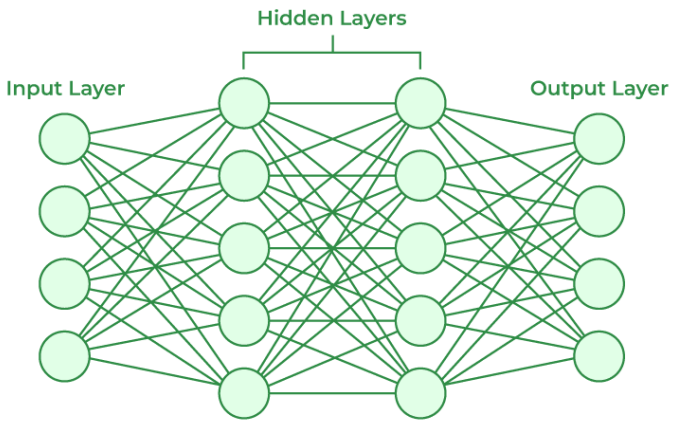
3. fejezet

Neurális hálók

A *mély neurális hálózatok* több egymásra épülő rétegből állnak, amelyek *mesterséges neuronokat* vagy *csomópontokat* tartalmaznak. A 3.1 ábrán szemléltetett módon ezek a csomópontok különböző *matematikai műveleteket* végeznek el, jellemzően *lineáris transzformációkat*, amelyek révén az információ feldolgozása és továbbítása történik.

Ezek a hálózatok három fő rétegből épülnek fel: a *bemeneti rétegből*, a *rejtett rétegekből* és a *kimeneti rétegből*. Az első réteg, a bemeneti réteg, fogadja és előkészíti az adatokat, amelyeket aztán a rendszer a következő szintek felé továbbít. A köztes rétegek, az úgynevezett rejtett rétegek, az adatok feldolgozását végzik. Ezek a rétegek az információkat *nemlineáris aktivációs függvények* segítségével alakítják át, és fokozatosan kinyerik a releváns jellemzőket a bemenetekből. [6]

A rejtett rétegek működése közvetlenül nem megfigyelhető, mivel a bennük található paraméterek – azaz a *súlyok* és *eltolások* – a *tanulási folyamat* során finomhangolódnak. Ezek a paraméterek lehetnek kezdetben véletlenszerűek, de előfordulhat, hogy előre meghatározott értékekkel indulnak, amelyeket korábbi tanulási tapasztalatok alapján állítottak be. A rejtett rétegek célja, hogy az adatok *összetett mintázatait* felismer-



3.1. ábra. Neurális háló, forrás: [7]

jék és olyan jellemvonásokat azonosítsanak, amelyek hozzájárulnak a végső döntések meghozatalához, például egy predikció vagy osztályozás formájában. [6]

A rejtett rétegek által előállított információ végül eljut a kimeneti réteghez, amely a hálózat végső eredményeit biztosítja. Az adott feladattól függően ez a réteg lehetővé teszi az *osztályozást*, *előrejelzést* vagy akár új minták generálását. Az adatok ilyen módon történő feldolgozását és továbbítását *előreterjesztésnek* nevezzük, amely a 3.1. ábrán is szemléltetett módon zajlik. [6]

3.1. TensorFlow – A Mélytanulási Keretrendszer

A *TensorFlow* egy nyílt forráskódú, fejlett *gépi tanulási* és *mesterséges intelligencia* (AI) keretrendszer, amelyet a Google fejlesztett ki és tett elérhetővé 2015-ben. Az egyik legnépszerűbb eszközként szolgál a mesterséges intelligencia alkalmazások fejlesztésében, különösen a *mélytanulási modellek* létrehozásában. A TensorFlow egy olyan rugalmas és hatékony platform, amely lehetővé teszi a kutatók és fejlesztők számára, hogy széleskörű gépi tanulási modelleket és alkalmazásokat hozzanak létre és futtassanak különböző környezetekben, beleértve a *felhőt*, a *mobil eszközöket* és a *beágyazott rendszereket*. [15]

A TensorFlow kifejlesztésének fő célja, hogy megkönnyítse a gépi tanulási modellek létrehozását és implementálását, ugyanakkor a számítási feladatokat egy *skálázható*, *párhuzamosítható* grafikus számítási modellben kezelje. Az eszközt eredetileg a Google Brain csapata készítette, hogy támogassa a Google-hoz tartozó projektek, például a keresőoptimalizálás, képfeldolgozás, gépi fordítás, és más mesterséges intelligencia alkalmazások fejlesztését. [15]

A TensorFlow támogatja a különböző gépi tanulási algoritmusokat, mint például a *felügyelt* és *felügyelet nélküli tanulás*, valamint a mélytanulás különböző típusait, mint a neurális hálózatok, *konvolúciós neurális hálózatok* (CNN) és *„ismételd" neurális hálózatok* (RNN). A TensorFlow képes hatékonyan futtatni modelleket *CPU*-n, *GPU*-n és más dedikált hardvereken, mint például a *Tensor Processing Unit* (TPU), amely a Google által kifejlesztett egyedi hardver az AI számítási feladatok gyorsítására. [15]

A TensorFlow egyik legnagyobb előnye az, hogy skálázható, tehát képes kis mértékű alkalmazásoktól kezdve, a világ legnagyobb felhőalapú rendszereit is kiszolgálni. Emellett a TensorFlow lehetővé teszi a modellek mobil eszközökön való futtatását, így a fejlesztők egyszerűen vihetik a mélytanulási megoldásaikat a mobil alkalmazások világába is. [15]

3.2. Konvolúciós neurális hálózatok és a Keras könyvtár

A *konvolúciós neurális hálózatok* (CNN-ek) a mélytanulás egyik leggyakrabban alkalmazott architektúrái, különösen *képfeldolgozási* feladatokban. Felépítésük a vizuális információk feldolgozására lett optimalizálva, így kiválóan alkalmazhatók például *arcfelismerésre*, *nem-* és *korosztály-becslésre*, vagy más *biometrikus minták* feldolgozására. [10]

A CNN-ek működésének alapja a *konvolúciós réteg*, amely képes a bemeneti képekről *jellemzőket* (éleket, mintázatokat, textúrákat stb.) kinyerni úgy, hogy a teljes képet nem kell egydimenziós vektorrá alakítani, hanem *lokális régiókat* vizsgál. Ezután általában következik egy *pooling réteg*, amely az adatok térbeli méretét csökkenti, miközben a legfontosabb jellemzőket megtartja. [9]

A klasszikus *teljesen összekötött rétegek* (fully connected layers) csak a feldolgozás legvégén jelennek meg, amikor már a rendszer „megtanulta", hogy milyen tulajdonságokat fontos figyelembe venni. A tanítás során a CNN megtanulja azokat a *szűrőket*, amelyek a feladat szempontjából releváns mintázatokat emelik ki. [11]

A CNN-ek használata különösen előnyös olyan esetekben, amikor az adatok erősen vizuális természetűek, például arc- vagy dokumentumképek, amelyeket az alkalmazás is feldolgoz. A hálózatok képesek tanulni a bemenetekből kiolvasható jellemzőket, így automatikusan felépíthetnek egy hatékony osztályozási logikát. [12]

A CNN-ek implementálásához az egyik leggyakrabban használt eszköz a ***Keras***, amely a TensorFlow magas szintű API-jaként működik. A Keras *deklaratív stílusban* teszi lehetővé a modellek gyors és egyszerű definiálását, betanítását és értékelését, így különösen alkalmas *prototípus-készítésre*, mobilalkalmazásokhoz való modellek fejlesztésére, vagy oktatási célokra. [16]

A dolgozatban bemutatott alkalmazás fejlesztése során is használatra került a Keras, amelynek segítségével könnyedén létre lehetett hozni a konvolúciós rétegekből és teljesen összekapcsolt (dense) rétegekből álló hálózatot, valamint alkalmazni lehetett az *Adam optimalizáló algoritmust* is, amely gyors és stabil tanulást biztosít. [17]

4. fejezet

Backend-en használt technológiák

4.1. Node.js

A *backend fejlesztés* kulcsszerepet játszik a modern web- és mobilalkalmazások működésében, hiszen ezen a rétegen történik az *adatok kezelése*, *tárolása* és *kiszolgálása*. A szerveroldali technológiák közül a *Node.js* az egyik legnépszerűbb választás, amely lehetőséget biztosít arra, hogy az alkalmazás teljes fejlesztése *JavaScript* nyelven történjen. A Node.js egy gyors és hatékony futtatókörnyezet, amely *eseményvezérelt* és *nem blokkoló* működésének köszönhetően különösen alkalmas nagy teljesítményt igénylő alkalmazások készítésére.

4.1.1. A Node.js alapjai és működési modellje

A Node.js a *Google V8 JavaScript motorjára* épül, amely lehetővé teszi a JavaScript kód gyors végrehajtását szerveroldalon. Az egyik legfontosabb sajátossága az *aszinkron* és eseményvezérelt működés, amely lehetővé teszi, hogy a szerver egyszerre több kérést is kezeljen anélkül, hogy az egyes műveletek blokkolnák egymást. Mivel a Node.js nem használ többszálú feldolgozást, hanem *egyetlen szálon* fut, a skálázhatóságot egy úgynevezett *eseményhurok* biztosítja, amely a beérkező kéréseket folyamatosan fogadja és kezeli. [18]

Az aszinkron működés egyik alapvető eszköze a *visszahívási függvények* (callback), amelyeket később az *ígéretek* (Promises) és az *async/await* konstrukciók egészítettek ki.

4.1.2. A Node.js előnyei és kihívásai

A Node.js egyik legnagyobb előnye a *teljesítménye* és *skálázhatósága*. Az eseményvezérelt architektúra miatt a szerver könnyedén kezelhet nagy számú *egyidejű kapcsolatot* anélkül, hogy túlterhelődne. Ez különösen előnyös olyan alkalmazások esetében,

amelyek *valós idejű interakciót* igényelnek, például csevegőalkalmazások vagy élő adatstreaming megoldások.

Egy másik fontos előny a JavaScript használata mind a *frontend*, mind a *backend* fejlesztés során. Ez lehetővé teszi, hogy a fejlesztőcsapatok egységes technológiai környezetben dolgozzanak, ami jelentősen csökkenti a fejlesztési időt és egyszerűsíti a kód karbantarthatóságát.

A Node.js azonban nem minden esetben ideális választás. A *CPU-intenzív műveletek*, például nagy mennyiségű számítási feladatokat végző algoritmusok vagy mesterséges intelligencia modellek futtatása esetén a Node.js teljesítménye korlátozott lehet. Mivel egyszálú környezetben működik, a nagy számítási igényű feladatok blokkolhatják az egész szerver működését. Ilyen esetekben érdemes *külső szolgáltatásokat* vagy *külön szálakon futó háttérfolyamatokat* használni a terhelés elosztására.

4.1.3. Backend fejlesztés Express.js segítségével

A Node.js önmagában is alkalmas szerverek létrehozására, de a fejlesztést jelentősen megkönnyíti az *Express.js*, amely egy minimalista és rugalmas webkeretrendszer. Az Express lehetőséget biztosít *API végpontok* kialakítására, *HTTP kérések* kezelésére és *middleware-ek* használatára. [19, 20]

4.1.4. Adatbázis-kezelés Node.js-ben

A backend fejlesztés egyik alapvető feladata az *adatkezelés*, amelyhez különböző adatbázisokat lehet használni. A Node.js kompatibilis mind a *relációs (SQL)*, mind a *NoSQL* adatbázisokkal. A relációs adatbázisok, például a *MySQL* vagy a *PostgreSQL* strukturált adatkezelést biztosítanak, míg a NoSQL megoldások, mint a *MongoDB*, rugalmasabb adatmodellezést tesznek lehetővé.

Egy MongoDB alapú adatbázis kapcsolat létrehozására a *Mongoose* csomag használható, amely egy *objektumorientált adatmodellezést* biztosító eszköz. A Mongoose használata lehetővé teszi az adatbázis-interakciók egyszerű kezelését, miközben biztosítja az adatok *validálását* és *strukturálását*. Az objektumorientált modellnek köszönhetően a JavaScript objektumok könnyedén átalakíthatók adatbázis-bejegyzésekké. [21]

4.2. MongoDB

A *MongoDB* egy *nyílt forráskódú*, *dokumentum-orientált adatbázis-kezelő rendszer* (DBMS), amelyet a MongoDB, Inc. fejlesztett. Az adatokat *dokumentumok* formájában tárolja, amelyek általában *JSON-szerű BSON* formátumban (Binary JSON) kerülnek mentésre. Ez lehetővé teszi, hogy az adatokat *rugalmasan*, *skálázható* módon tároljuk, így ideális választás *nagy mennyiségű*, *változó szerkezetű* adat kezelésére.

A MongoDB-t 2007-ben alapították, és gyorsan elnyerte a fejlesztők és cégek körében a népszerűséget a hagyományos relációs adatbázisokkal szembeni előnyei miatt. Mivel a MongoDB nem követeli meg a *szigorú séma* használatát, az adatokat szabadon, dinamikusan tárolhatjuk, amely különösen hasznos a gyorsan változó vagy nem strukturált adatokkal dolgozó alkalmazások számára.

A MongoDB-t széles körben használják különböző típusú alkalmazásokban, beleértve a *webalkalmazásokat*, *analitikai platformokat*, valamint a *Big Data* és a *gépi tanulási* megoldásokat is. A NoSQL adatbázisok közé tartozik, amelyek az újabb alkalmazásfejlesztési igényeknek megfelelően nem használják a relációs adatmodellek korlátait, például az előre meghatározott táblákat és a szigorú séma-rendszert.

4.2.1. A MongoDB működési elve

A MongoDB működésének alapja a *dokumentum-orientált adattárolás*. A dokumentumok egyszerű *kulcs-érték párokat* tartalmaznak, de sokkal *összetettebb adatstruktúrákat* is képesek tárolni, például tömböket, beágyazott dokumentumokat és más összetett típusokat. Az adatok nem táblákban és sorokban, hanem dokumentumokban vannak tárolva, amelyek egy-egy adatbázison belül *kollekciókban* helyezkednek el. [22]

A MongoDB által használt adatmodell lehetővé teszi az adatbázisok *szabad struktúráját*, amely *rugalmasságot* biztosít a fejlesztők számára, hiszen nem szükséges előre definiálni az adatbázis séma szerkezetét. Ez különösen hasznos akkor, amikor az adatok változnak, fejlődnek vagy különböző forrásokból származnak. [22]

Mivel a MongoDB adatokat dokumentumokban tárolja, amelyek JSON-szerű formátumban vannak, az adatok *hierarchikus struktúrában* szervezhetők. Az adatmodell nemcsak a gyors fejlesztést és a rugalmas adattárolást teszi lehetővé, hanem a különböző adatfeldolgozási műveletek (például lekérdezések, frissítések, törlés) egyszerűsítését is. [22]

4.2.2. A MongoDB különleges jellemzői

A MongoDB kiemelkedő jellemzője a *skálázhatóság*, amely lehetővé teszi a rendszer számára, hogy adatokat nagy mennyiségben kezeljen anélkül, hogy teljesítménybeli problémák merülnének fel. A MongoDB többféle skálázási lehetőséget kínál, mint például a *sharding*, amely az adatokat több szerverre osztja el, és a *replikáció*, amely biztosítja az adatok biztonságos és megbízható tárolását. [23]

A sharding azt jelenti, hogy a MongoDB képes az adatokat különböző szerverek között megosztani (például egy szétosztott rendszerben), így képes nagyobb adatbázisokat kezelni. Mindez a rendszer teljesítményét nem befolyásolja, mivel minden egyes szerver csak egy részét tárolja az adatoknak. A replikáció segítségével a MongoDB biztosítja az adatok elérhetőségét, mivel az adatok több példányban vannak tárolva, és

ezek automatikusan szinkronizálódnak a szerverek között. Ez a mechanizmus biztosítja az adatbázisok *megbízhatóságát* és *rendelkezésre állását*. [23]

A *másodlagos indexek* és a *full-text keresés* lehetővé teszik a MongoDB számára, hogy gyorsan és hatékonyan végezzen kereséseket az adatok között, így megfelelő választ adva a komplex lekérdezési igényekre is. Az indexek a lekérdezések sebességét növelik, míg a full-text keresés lehetővé teszi a szöveges adatok gyors feldolgozását. [23]

4.2.3. A MongoDB előnyei és hátrányai

Mivel a MongoDB nem követeli meg a szigorú séma alkalmazását, lehetőséget biztosít a *változó adatstruktúrák* kezelésére. Az adatbázis szerkezete bármikor változtatható, új mezők és adatpontok hozzáadása nem igényli az egész adatbázis átdolgozását.

A MongoDB képes hatékonyan kezelni *nagy mennyiségű adatot*, akár több szerverre is szétosztva a sharding mechanizmus segítségével, így biztosítva a *magas rendelkezésre állást* és *megbízhatóságot*.

A MongoDB *gyors adatolvasást és -írást* biztosít, különösen a nagy mennyiségű, változatos adatokat kezelő alkalmazások esetében.

A MongoDB *rugalmas adatmodellt* kínál, amely lehetővé teszi a *gyors iterációt* és az alkalmazások gyors fejlesztését. A séma nélküli adatbázisok különösen hasznosak olyan dinamikus alkalmazások esetén, ahol az adatok gyorsan változnak.

Bár a MongoDB folyamatosan fejlődik ezen a téren, a relációs adatbázisokhoz képest még mindig gyengébb a *tranzakciókezelés*, különösen komplex, több lépéses tranzakciók esetén.

Mivel a MongoDB nem használja a hagyományos relációs adatbázisokat, a fejlesztőknek nem minden esetben van elegendő tapasztalatuk ezzel a technológiával, és bár a közösség növekszik, még mindig kisebb, mint a hagyományos adatbázisoké.

Mivel a MongoDB nem használ előre meghatározott sémát, az *adatok szerkezete* nem mindig van megfelelően standardizálva, ami problémákat okozhat a nagy rendszerek integrálásakor.

4.2.4. A MongoDB alkalmazásai

A MongoDB különösen akkor válik hasznossá, amikor az alkalmazás olyan *Big Data* vagy *valós idejű adatfeldolgozási* igényekkel rendelkezik, amelyek *nagy volumenű* és *folyamatosan változó adatokat* igényelnek. Ilyen típusú alkalmazások közé tartoznak például a *webalkalmazások*, *mobilalkalmazások*, *e-kereskedelmi rendszerek*, big data elemzések, valamint az *IoT rendszerek*. A MongoDB *nagy sebességű*, *dinamikus adatfeldolgozási* képessége miatt ideális választás olyan modern alkalmazásokhoz, amelyek gyorsan reagálnak az adatváltozásokra, és nem igényelnek szigorú adatmodell-szerkezetet.

5. fejezet

Frontend-en használt technológiák

5.1. React Native

A *React Native* egy *nyílt forráskódú keretrendszer*, amelyet a Facebook fejlesztett ki, és amely lehetővé teszi *natív mobilalkalmazások* fejlesztését *JavaScript* és *React* használatával. A React Native célja, hogy egyszerűsítse a mobilalkalmazások fejlesztését, miközben a natív alkalmazások teljesítményét és élményét biztosítja. A keretrendszer alapja a React, amely egy JavaScript könyvtár a *felhasználói felületek* (UI) építésére, de a React Native lehetővé teszi, hogy *ugyanazt a kódot* használjuk mind az *Android*, mind pedig az *iOS* platformokon. Ez különösen vonzó azoknak a fejlesztőknek, akik keresnek egy *költséghatékony* és *időtakarékos* megoldást a mobilalkalmazások fejlesztésére. [14, 24]

5.1.1. A React Native működési elve

A React Native működésének alapja a *natív komponensek* és *JavaScript csomag* kombinációja. Amíg a React webalkalmazásoknál HTML és CSS használatával építi fel a felhasználói felületet, addig a React Native *natív iOS- és Android-komponenseket* használ. Ez azt jelenti, hogy bár a fejlesztő JavaScript-ben írja meg a kódot, a végén az alkalmazás natív komponensekben jelenik meg a mobil eszközökön, így biztosítva a natív alkalmazásokhoz hasonló teljesítményt és élményt. [14, 24]

A React Native alkalmazások a következőképpen működnek:

- *JavaScript szál*: Az alkalmazás logikáját és az *állapotkezelést* JavaScript kódban valósítják meg, amely a React-tel működik. Ez a kód kezelni fogja az *eseményeket*, az *adatokat*, és irányítja a felhasználói felületet.

- *Natív szál*: A React Native alkalmazások natív szálat használnak az Android és iOS platformok natív komponenseihez való hozzáféréshez. Az alkalmazás natív

elemei, mint például a gombok, szövegdobozok, listák és más UI komponensek, teljes mértékben az adott platform *natív funkcióit* használják.

Ez a megoldás lehetővé teszi, hogy a fejlesztők a natív alkalmazások *sebességét* és *reakcióképességét* érjék el, miközben csak egy kódot kell írniuk mindkét platformra. [14, 25]

5.1.2. A React Native Jellemzői

A React Native több fontos jellemzőt kínál, amelyek a modern mobilalkalmazások fejlesztésének alapvető szükségleteit szolgálják ki. Ezek közé tartozik a *gyors fejlesztés*, a *natív élmény*, az *eszközökhöz való közvetlen hozzáférés* és a *közösségi támogatás*. [14, 25]

A keretrendszer egyik legnagyobb előnye, hogy ugyanazt a JavaScript kódot használhatjuk az Android és iOS alkalmazásokhoz is. Ez jelentős *időt és erőforrást takarít meg*, mivel nem szükséges különböző nyelveken és fejlesztési környezetekben dolgozni a két platform számára. Az alkalmazás *üzleti logikáját* és felhasználói felületét ugyanazon a *kódalapú megoldáson* fejleszthetjük, így a fejlesztési ciklusok gyorsabbak és hatékonyabbak lesznek. [14, 25]

Bár a React Native egy JavaScript alapú keretrendszer, a natív komponensek használatának köszönhetően az alkalmazások teljesítménye közel áll a natív alkalmazásokéhoz. A keretrendszer lehetőséget ad arra, hogy közvetlenül hozzáférjünk az eszköz *hardveréhez és szoftveréhez*, mint például a *kamera*, a *GPS*, az *érintőképernyő* és más *natív API-k*. Ezen kívül lehetőség van *natív modulok* létrehozására, amelyek még jobb teljesítményt biztosítanak a kritikus alkalmazás részek számára. [14, 25]

A React Native mögött hatalmas *fejlesztői közösség* áll, amely folyamatosan bővíti és fejleszti a keretrendszert. Ennek eredményeként számos *könyvtár* és *eszköz* érhető el, amelyek segíthetnek a gyakori feladatok, mint például az *adatkezelés*, a *navigáció* vagy az *animációk* kezelésében. A közösség által nyújtott támogatás rendkívül hasznos a fejlesztők számára, mivel gyorsan válaszokat és megoldásokat találhatnak a problémákra. [14, 25]

5.2. Expo

Az *Expo* egy olyan nyílt forráskódú *keretrendszer* és *eszközkészlet*, amely megkönnyíti a React Native alkalmazások fejlesztését, tesztelését és telepítését. Célja, hogy gyorsabbá és kényelmesebbé tegye a mobilalkalmazások fejlesztési folyamatát, anélkül hogy *natív kóddal* kellene dolgozni. Az Expo biztosít egy *előre konfigurált fejlesztői környezetet*, amely csökkenti a beállításokkal és függőségekkel járó problémákat, így különösen népszerű kezdők és *gyors fejlesztési ciklusokat* igénylő projektek körében. [26]

5.2.1. Expo előnyei

Az Expo egyik legnagyobb előnye, hogy a fejlesztőknek nem kell natív kódot írniuk vagy külön fejlesztői eszközöket telepíteniük Androidra és iOS-re. Az alkalmazások közvetlenül futtathatók egy mobileszközön az *Expo Go* alkalmazás segítségével, amely lehetővé teszi a *gyors iterációt* és *valós idejű módosításokat*. Az Expo emellett több *beépített API-t* is biztosít, például *kamera*, *helymeghatározás*, *értesítések* és *fájlkezelés* támogatását, anélkül hogy külső függőségeket kellene telepíteni. [26, 27]

5.2.2. Expo CLI és Managed Workflow

Az *Expo CLI* egy *parancssori eszköz*, amely lehetővé teszi a React Native projektek egyszerű létrehozását, fejlesztését és buildelését. Az „expo-cli" *npm csomag* telepítése után könnyen indíthatók új projektek az „*npx create-expo-app@latest*" paranccsal, ami jelentősen leegyszerűsíti a fejlesztési folyamat kezdeti lépéseit. Az Expo CLI biztosítja a *hot reloading* funkciót, a *QR kód* alapú eszköz-csatlakoztatást és számos más fejlesztést segítő eszközt.

Az Expo két fő módot kínál a projektek kezelésére: a *Managed Workflow* és a *Bare Workflow*. A Managed Workflow az Expo által kezelt megoldás, ahol nincs szükség natív kód szerkesztésére, míg a Bare Workflow lehetőséget biztosít natív kód írására is, ha speciális funkciókra van szükség. A fejlesztők többsége a Managed Workflow-t használja, mert ezzel a módszerrel gyorsan beállítható és futtatható egy React Native alkalmazás. Az Expo CLI mindkét workflow típust támogatja, de a Managed Workflow esetén nyújtja a legtöbb *automatizált megoldást*, mint például az *OTA* (Over-The-Air) frissítések kezelése vagy a natív függőségek automatikus konfigurálása. [26, 27]

5.2.3. Expo alkalmazás publikálása

Az Expo lehetőséget biztosít az alkalmazások *könnyű publikálására* is. Emellett az Expo *EAS* (Expo Application Services) platformot is biztosít, amely támogatja az *OTA* (Over-The-Air) frissítéseket, így a fejlesztők anélkül tudnak új verziókat kiadni, hogy a felhasználóknak frissíteniük kellene az alkalmazást az áruházból. [26, 27]

6. fejezet

A projekt

A projekt célja egy olyan *mobilalkalmazás* fejlesztése volt, amely lehetőséget biztosít a felhasználók számára *személyazonosságuk igazolására*. A rendszer működése során a felhasználó először *regisztrál* az alkalmazásban, majd ezt követően a személyazonosító okmányáról egy *fényképet készít*. Az így kapott kép feldolgozása egy *neurális hálózat* segítségével történik, amely azonosítja és kinyeri a *releváns adatokat* az okmányról. Az így kinyert információkat az alkalmazás egy *biztonságos adatbázisban* tárolja, biztosítva ezzel a felhasználó azonosításának *megbízhatóságát* és *hatékonyságát*.

6.1. Adathalmaz generáló program

A neurális háló hatékony betanításához megfelelő mennyiségű és változatosságú *adathalmazra* (dataset) volt szükség. Mivel nem létezett olyan nyilvánosan elérhető adatforrás, amely magyar *személyazonosító igazolványok* képeit tartalmazta volna, ezért a szükséges *adatkészletet* saját magamnak kellett előállítanom. Az adathalmaz generálása során az alábbi lépéseket követtem.

Az első lépés egy megfelelő alapként szolgáló személyazonosító igazolvány képének beszerzése volt. Az interneten keresve találtam egy olyan *példányt*, amely megfelelő *referenciát* biztosított a további példányok előállításához. Az így kiválasztott minta a 6.1 ábrán látható:

6.1. ábra. Minta személyazonosító igazolvány, forrás: [8]

Mivel az eredeti képen valós *személyes adatok* szerepeltek, ezért azokat el kellett távolítanom. Ehhez az *Adobe Photoshop* szoftvert használtam, amellyel kitöröltem az összes szöveget és egyéb személyes információt, így egy üres, *adatmentes igazolványt* kaptam. A megtisztított igazolvány a 6.2 ábrán tekinthető meg:

6.2. ábra. Minta személyazonosító igazolvány retusálva, forrás: saját kép

Miután a kártyán szereplő eredeti adatokat eltávolítottam, a következő lépés az adatmezők *véletlenszerű kitöltése* volt. Ehhez először egy előre összeállított *névadathalmazt* használtam, amelyből a nevek generálása történt. A személyazonosító igazolványon szereplő fényképekhez egy előre összegyűjtött, *szelfikből álló képadatbázist* vettem alapul. A további adatok, mint például a *CAN azonosító*, az *okmányazonosító*, az *érvényességi idő*, a *születési dátum* és a *nemre* vonatkozó információ teljesen véletlenszerűen lettek generálva.[28, 29]

A képek módosításához a Python programozási nyelvhez elérhető *Pillow csomagot* alkalmaztam, amely lehetőséget biztosított arra, hogy a képen szöveget helyezzek el, valamint új *képrétegeket* illesszek az eredeti alapra. A megfelelő vizuális megjelenés érdekében a szöveg pontos pozícióját, tartalmát, betűtípusát és színét előzetesen meg kellett határozni. Ehhez a *Vanilla Extract Font* betűtípust választottam, mivel ez hasonlít leginkább a személyazonosító igazolványokon alkalmazott karakterkészlethez.

A generált adatok és képek további feldolgozása során minden adatmezőhöz kiszámítottam a megfelelő *koordinátákat*, amelyek a szöveg elhelyezéséhez szükségesek. A hatékonyság növelése érdekében függvény segítségével meghatároztam az egyes adatmezők *határoló téglalapjait*. A generált igazolványok további *augmentációkat* is kaptak, amely során enyhe *elforgatást*, *fényerő*- és *kontrasztmódosítást*, valamint *zaj hozzáadását* alkalmaztam, hogy az adatkinyerő modell számára változatosabb adathalmaz álljon rendelkezésre.

A képek elforgatásához *függvényeket* használtam, amelyek a kép középpontja körül elforgatták a határoló dobozokat. A zajgenerálás során minden pixel értékéhez egy *véletlenszerű értéket* adtam hozzá, amelynek mértéke -10 és 10 közé esett. A fényerő és kontraszt módosításához a Pillow *ImageEnhance* osztályát alkalmaztam. Az augmentált képeket végül *JSON formátumú* fájlokba mentettem, amelyek tartalmazták a határoló dobozok koordinátáit és a hozzájuk tartozó szöveges értékeket.

A végső lépésként az előállított személyazonosító igazolványokat és a hozzájuk tartozó *címkézett adathalmazt* fájlba mentettem, amelyet a 6.1 kódrészlet mutat be.

```
augmented_image, augmented_boxes, angle =
    ↪ augment_image_and_boxes(id_card, bounding_boxes)
if augmented_image.mode == "RGBA":
    augmented_image = augmented_image.convert("RGB")

label_data = {
    "image": f'eszemelyi_with_{person[1]}_{person[0]}.jpg',
    "labels": []
}

for idx, field in enumerate(["Name", "Doc No", "Birthday",
    ↪ "Expiry", "CAN", "Gender", "Image"]):
    box = augmented_boxes[idx]
    x1, y1, x2, y2 = box

    text_value = ""

    if field == "Name":
        text_value = name
    elif field == "Doc No":
        text_value = doc_no
    elif field == "Birthday":
        text_value = birthday_str
    elif field == "Expiry":
        text_value = expiry_str
    elif field == "CAN":
        text_value = can_str
    elif field == "Gender":
        text_value = "F" if gender=="M" else "N"
```

```
28
29      label_entry = {
30          "label": field,
31          "text": text_value,
32          "box": [[x1, y1], [x2, y2]]
33      }
34
35      label_data["labels"].append(label_entry)
36
37  output_image_path =
        ↪ f'./data/Generated_cards/eszemelyi_with_{person[1]}{person[0]}.jpg'
38  output_json_path =
        ↪ f'./data/Generated_cards/json_labels/{person[1]}{person[0]}.json'
39  augmented_image.save(output_image_path)
40
41  with open(output_json_path, 'w') as json_file:
42      json.dump(label_data, json_file, indent=4)
```

6.1. kód. Augmentáció és fájlmentés

Az így előállított és *címkézett adathalmaz* megbízható alapot nyújtott a *neurális hálózat betanításához*, lehetővé téve az igazolványokon szereplő adatok *pontos felismerését*. A 6.3 ábrán egy generált személyi igazolvány látható, amelyen a piros keretek a *határoló téglalapokat* jelölik. Fontos megjegyezni, hogy ezek kizárólag a *szemléltetés céljából* kerültek megjelenítésre, az éles alkalmazás során nem láthatók.

6.3. ábra. Egy generált személyazonosító igazolvány, forrás: saját kép

6.2. A projektben használt neurális hálók

6.2.1. OCR modell

Az *optikai karakterfelismerés* (OCR) megvalósításához a „01 image to word” projektet használtam fel, amely a *Machine Learning Training Utilities* (MLTU) *könyvtárra* épül. Az eredeti *implementáció* egy előre meghatározott adathalmazon működik, azonban

a személyi igazolványokon található adatok *pontosabb felismerése* érdekében a *saját adathalmazommal* tanítottam be a modellt. [31]

```
1   import os
2   from datetime import datetime
3   from mltu.configs import BaseModelConfigs
4
5   class ModelConfigs(BaseModelConfigs):
6       def __init__(self):
7       super().__init__()
8       self.model_path =
            ↪ os.path.join("Models/1_image_to_word",
            ↪ datetime.strftime(datetime.now(), "%Y%m%d%H%M"))
9       self.vocab =
            ↪ "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"
10      self.height = 32
11      self.width = 128
12      self.max_text_length = 23
13      self.batch_size = 1024
14      self.learning_rate = 1e-4
15      self.train_epochs = 100
16      self.train_workers = 20
```

6.2. kód. Konfigurációs fájl

A rendszer működéséhez több komponens *összehangolt működésére* volt szükség. Az egyik legjelentősebb a 6.2. ábrán látható *konfigurációs fájl*, amely meghatározza a modell legfontosabb paramétereit, például a *képek méretét*, a *megengedett karakterek halmazát*, a *maximális szöveghosszt* és a *tanulási sebességet*. Ezeket az értékeket az igazolványokon szereplő adatok felismeréséhez optimalizáltam.

A neurális hálózat egy *konvolúciós neurális hálózatot* (CNN) és egy *bidirekcionális hosszú rövid távú memóriával rendelkező* (BiLSTM) réteget egyesít. A CNN rétegek a képekből *jellemzőket vonnak ki*, míg az LSTM rétegek az *időbeli összefüggéseket* elemzik. Az utolsó réteg egy *teljesen összekapcsolt* (Dense) réteg, amely az előrejelzett karaktereket adja vissza. A tanítás során a *Connectionist Temporal Classification* (CTC) algoritmus felel a karakterek helyes sorrendjének rekonstruálásáért.

A tanítás során a rendszer az általam előkészített *adathalmazon* tanult. A tanítási folyamat az adatok *betöltését*, *előfeldolgozását* és a neurális hálózat *tanítását* végzi. Az eredeti kódban egy nyilvános OCR adathalmaz szerepelt, azonban ezt lecseréltem *saját igazolványos képeimre* és azokhoz tartozó *címkékre*. Az adathalmaz előkészítése során minden egyes képhez hozzárendeltem a rajta található szöveges információt, amelyet címkeként használt a modell a tanulás során.

A betanítás folyamán különböző *adattranszformációs lépések* biztosították, hogy a rendszer hatékonyan tudja feldolgozni a bemeneti adatokat. A képeket megfelelő mé-

retre alakítottam, a karaktereket *számmá konvertáltam*, és gondoskodtam róla, hogy a címkék *egységes formátumban* szerepeljenek. A modell az *Adam optimalizáló algoritmust* és a *CTC veszteségfüggvényt* használta a hatékonyabb tanulás érdekében.

A tanítás során különböző *visszacsatolási mechanizmusok* segítették az optimális modellparaméterek megtalálását. Az *EarlyStopping* mechanizmus például megakadályozta a *túlilleszkedést* azáltal, hogy időben leállította a tanulási folyamatot, ha az *érvényesítési hiba* egy bizonyos számú iteráció után nem csökkent tovább. A tanítási folyamat végén a modell a *legjobb teljesítményt* nyújtó állapotában került elmentésre.

A tanítás után az elkészült modellt teszteltem *éles adatokon*. Ehhez a 6.3. kódban látható függvényt használtam, amely képes egy *bemeneti képből szöveges információt visszaadni*. A képek betöltése és előfeldolgozása után a modell futtatásra kerül, majd a CTC algoritmus *dekódolja* a felismerési eredményt, és visszaadja a szöveget.

```
1   from mltu.inferenceModel import OnnxInferenceModel
2   from mltu.utils.text_utils import ctc_decoder, get_cer
3
4   class ImageToWordModel(OnnxInferenceModel):
5       def __init__(self, char_list: typing.Union[str, list],
            ↪ *args, **kwargs):
6       super().__init__(*args, **kwargs)
7       self.char_list = char_list
8
9       def predict(self, image: np.ndarray):
10      image = cv2.resize(image,
            ↪ self.input_shapes[0][1:3][::-1])
11
12      image_pred = np.expand_dims(image,
            ↪ axis=0).astype(np.float32)
13
14      preds = self.model.run(self.output_names,
            ↪ {self.input_names[0]: image_pred})[0]
15
16      text = ctc_decoder(preds, self.char_list)[0]
17
18      return text
```

6.3. kód. Modell tesztelése

A kiértékelési fázisban az OCR rendszer teljesítményét a *Character Error Rate* (CER) segítségével mértem, amely azt mutatja meg, hogy a modell mennyire *pontosan* képes *felismerni a szöveget*. A tesztelés során kiderült, hogy a *saját adathalmazzal tanított modell* jelentősen *pontosabb eredményeket* produkált a személyi igazolványokon található adatok felismerésekor, mint az *eredeti verzió*.

6.2.2. Határoló doboz detektáló modell

Ez a *neurális hálózat* felel azért, hogy a *személyi igazolványokon* található *releváns adatokat* – mint a *név*, *személyi azonosítószám*, *CAN kód*, *fénykép*, *lejárati dátum* és *születési dátum* – *automatikusan felismerje* és azok köré *határdobozt* (bounding box) helyezzen. A rendszer célja, hogy *pontosan azonosítsa* ezeknek a *mezőknek a helyzetét* a képeken, lehetővé téve azok *további feldolgozását* az *OCR modellel*.

A modell *betanításához* a 6.1. fejezetben bemutatott *adathalmazt* használom. Ahogy a 6.4. kódban látható, az *adatelőkészítő modul* minden egyes *képhez hozzárendeli* a megfelelő *JSON fájlt*, amely tartalmazza az egyes *mezők sarokpontjait*. Az adatok *normalizálásra kerülnek*, hogy a *modell számára könnyebben kezelhető* formát öltsenek, így a *kimeneti koordináták* a kép szélességéhez és magasságához viszonyított *relatív értékekként* jelennek meg.

```
1  import os
2  import json
3  import numpy as np
4  from tensorflow.keras.preprocessing.image import load_img,
       ↪ img_to_array
5
6  def load_data(image_dir, json_dir):
7      images, boxes = [], []
8
9      for json_file in os.listdir(json_dir):
10         if not json_file.endswith('.json'):
11             continue
12         with open(os.path.join(json_dir, json_file), 'r') as f:
13             data = json.load(f)
14
15             img_path = os.path.join(image_dir, data['image'])
16             img = load_img(img_path, target_size=(378, 600))
17             img_array = img_to_array(img) / 255.0
18             images.append(img_array)
19
20         image_boxes = []
21         for label in data['labels']:
22             for point in label['box']:
23                 x, y = point
24                 image_boxes.extend([x / 600, y / 378])
25
26             boxes.append(image_boxes)
27
28     return np.array(images), np.array(boxes)
```

6.4. kód. Adathalmaz betöltése

A *neurális hálózat* egy *konvolúciós architektúrát* követ, amely a képeken lévő *mintázatok felismerésével* azonosítja az adott mezők elhelyezkedését. Az első néhány réteg *konvolúciós* és *max-pooling műveletekkel* szűri ki a *vizuális jellemzőket*, míg a *teljesen összekötött* (dense) réteg ezeket feldolgozva kiszámítja a *határdobozok sarokpontjait*. A modell végső kimenete egy *vektor*, amely minden egyes mezőhöz tartalmazza a megfelelő *koordinátákat*.

A *tanítási folyamat* során az *Adam optimalizálót* használjuk a gyorsabb konvergencia érdekében, míg *veszteségfüggvényként* az *átlagos négyzetes hibát* (MSE) alkalmazzuk, mivel a bounding box koordináták *folytonos numerikus értékek*. A rendszer több *visszahívási mechanizmussal* (callback) rendelkezik: a *korai leállítás* (early stopping) segít elkerülni a *túlilleszkedést*, míg a *modellellenőrzési pontok* (checkpointing) biztosítják, hogy a legjobb teljesítményű modellek automatikusan elmentésre kerüljenek. Emellett a tanítás során egy *speciális metrikát* is alkalmazunk, amely a határoló dobozok pontosságát méri. A 6.5. kódban látható „*bounding box accuracy*" függvény a modell által *előrejelzett és a valós koordináták* közötti eltérést számítja ki, és egy adott *toleranciaérték* (0,03) alapján határozza meg, hogy a becslés mennyire pontos. Az *eltérések abszolút értékét* veszi, majd összehasonlítja a *megengedett hibahatárral*, végül pedig ezek *átlaga* adja meg a *modell pontosságát*. Ez a metrika segít abban, hogy ne csupán az MSE alapján értékeljük a modellt, hanem figyelembe vegyük azt is, hogy a detektált koordináták a gyakorlatban mennyire használhatóak.

```
1    def bounding_box_accuracy(y_true, y_pred):
2        tolerance = 0.03
3        diff = tf.abs(y_true - y_pred)
4        correct = tf.cast(diff < tolerance, tf.float32)
5        return tf.reduce_mean(correct)
```

6.5. kód. Határoló doboz pontosság kiszámítása

6.2.3. Nemfelismerő neurális hálózat

Ez a *neurális hálózat* a *személyi igazolványokon* szereplő *arcképek* alapján határozza meg, hogy a képen látható személy *nő vagy férfi*. Ez a rendszer is a 6.1. fejezetben bemutatott *adathalmazt* használja fel, amely többek között minden egyes igazolványképhez tárolja a releváns *határoló dobozokat* és a *nemi információt*.

Az *adatok előkészítése* során az *json fájlokban* meghatározott pozíciók alapján kivágásra kerül az *arcterület*, amelyet *normalizálunk* és egy *egységes méretre* (190×270 pixel) alakítunk. Ez biztosítja, hogy a neurális hálózat *konzisztens bemeneti adatokat* kapjon. Az adatokat ezt követően három részre osztjuk: *tanítási, validációs és teszt adathalmazra*, így garantálva a modell *objektív kiértékelését*.

A neurális hálózat egy *konvolúciós architektúrát* követ, amely *több rétegű mintá-*

zatfelismerést alkalmaz. A *bemeneti kép* a *konvolúciós rétegeken* keresztül fokozatosan *magasabb szintű reprezentációkká* alakul, miközben a *max-pooling rétegek* segítenek csökkenteni a *számítási komplexitást* és az *overfittinget*. A *kimeneti réteg egyetlen neuronból* áll, amely *sigmoid aktivációs függvényt* használ, így a modell *valószínűségi értéket* ad vissza a *nemi besorolásra*.

```
1   def build_model():
2
3       model = keras.Sequential([
4       layers.Conv2D(32, (3, 3), activation="relu",
            ↪ input_shape=(*IMAGE_SIZE, 3)),
5       layers.MaxPooling2D(2, 2),
6       layers.Conv2D(64, (3, 3), activation="relu"),
7       layers.MaxPooling2D(2, 2),
8       layers.Conv2D(128, (3, 3), activation="relu"),
9       layers.MaxPooling2D(2, 2),
10      layers.Flatten(),
11      layers.Dense(128, activation="relu"),
12      layers.Dropout(0.5),
13      layers.Dense(1, activation="sigmoid")
14      ])
15
16      model.compile(
17      optimizer="adam",
18      loss="binary_crossentropy",
19      metrics=["accuracy"]
20      )
21
22      return model
```

6.6. kód. Nemfelismerő modell

A tanítási folyamat során az *Adam optimalizálót* alkalmazom, mivel *gyors konvergenciát* biztosít, míg *veszteségfüggvényként* a *bináris keresztentrópiát* használjuk, amely jól illeszkedik a *kétosztályos osztályozási feladathoz*. A *túlilleszkedés* elkerülése érdekében a modell *dropout réteget* is tartalmaz, amely *véletlenszerűen deaktiválja* a *neuronok* egy részét a *teljes összekapcsolt rétegben*. A tanulás során itt is azokat a *visszahívási mechanizmusokat* (callback) használjuk, mint a 6.2.2. fejezetben bemutatott modellben.

A modell kiértékelése a *teszt adathalmazon* történik, ahol a *pontosság* (accuracy) alapján mérjük a *teljesítményt*. A *végleges modellt* elmentjük, így a későbbiekben közvetlenül *betölthető* és használható lesz új képek nemének meghatározására.

6.3. Node.js Backend

A projekt backend rendszere *Node.js* környezetben készült, valamint az *Express.js keretrendszerre* épül. Az Express *moduláris struktúrája* lehetővé teszi, hogy a szolgáltatásokat jól *elkülönített komponensekre* bontsuk. Fő moduljaink a *felhasználói azonosítást*, *képfeldolgozást*, *személyazonosító igazolványok kezelését* és a *felhasználói adatok kezelését* szolgálják.

Az alkalmazás adatait *MongoDB adatbázisban* tároljuk. A *rugalmas adatséma* lehetővé teszi a *komplex adatstruktúrák* hatékony tárolását. Az adatbázis kapcsolatot egy *singleton mintát* követő modulban valósítottuk meg, amely biztosítja, hogy csak *egyetlen kapcsolat* jöjjön létre az alkalmazás futása során. A kapcsolódási adatokat *környezeti változókban* tároljuk, ami növeli a *biztonságot* és megkönnyíti a *konfigurációs beállítások* kezelését különböző környezetekben.

A felhasználók hitelesítését *JWT* (JSON Web Token) technológiával oldottam meg, amely *biztonságos* és *állapotmentes* módon teszi lehetővé a felhasználók azonosítását. A jelszavakat *bcrypt algoritmussal* titkosítom, amely biztonságos és ellenáll a *brute force támadásoknak*. A bejelentkezési folyamat során ellenőrizni kell a felhasználó hitelesítő adatait, majd sikeres azonosítás esetén egy *időkorlátos JWT tokent* állítunk ki, amelyet a kliens minden további kérésnél elküld a szervernek.

A *middleware-ek* fontos szerepet játszanak az Express.js alkalmazásokban, lehetővé téve a *kérések előfeldolgozását* és *validálását*. Az alkalmazásban több middleware-t is használtunk, például a kérések jogosultságának ellenőrzésére. A „*protect*" middleware ellenőrzi a kéréshez csatolt JWT token érvényességét, és csak *hitelesített felhasználóknak* engedélyezi a *védett végpontok* elérését. Ez biztosítja, hogy az *érzékeny adatokhoz* és műveletekhez csak *jogosult felhasználók* férhessenek hozzá.

Az alkalmazás egyik kulcsfontosságú funkciója a *képfeldolgozás*, amely során a feltöltött személyi igazolványok adatai *automatikusan kinyerésre* kerülnek. Ezt a funkciót a *Node.js backend* és a *Python Flask backend* közötti kommunikáció teszi lehetővé. A *multer könyvtár* segítségével kezeljük a *feltöltött képeket*, amelyeket ezután továbbítunk a Python Flask szervernek, ahol a *neurális hálózatok* segítségével megtörténik az *adatok kinyerése*. Az *aszinkron kommunikációt Promise.all* segítségével oldjuk meg, amely lehetővé teszi a *párhuzamos feldolgozást*.

A *jelszó-visszaállítási* folyamathoz *e-mail küldési* funkcióra is szükség volt, amelyet a *nodemailer könyvtár* segítségével valósítottam meg. A könyvtár lehetővé teszi a *biztonságos e-mail küldést* különböző szolgáltatásokon keresztül. Az alkalmazásban Gmail szolgáltatást használunk, és a hitelesítési adatokat *környezeti változókban* tároljuk.

A backend szolgáltatások könnyen telepíthetőek és *skálázhatóak Docker konténerek* segítségével. A projektben *Docker* és *Docker Compose* használatával biztosítjuk a *konzisztens futtatási környezetet* és az *egyszerű telepítést*. A Docker Compose fájl

definiálja az alkalmazás szolgáltatásait és azok kapcsolatait. A Node.js backend és a Python Flask backend *külön konténerekben* futnak, de kommunikálhatnak egymással, ami *tiszta szeparációt* és *jobb skálázhatóságot* biztosít.

6.4. Flask Backend

A projekt másik kulcsfontosságú komponense a Python *Flask keretrendszerre* épülő backend, amely a *neurális hálózatok futtatását* és a *képfeldolgozási feladatokat* látja el.

A Flask alkalmazás fő feladata a neurális hálózati modellek futtatása és elérhetővé tétele *HTTP API-n* keresztül. A *NodeJS backend* számára biztosít interfészt, hogy a feltöltött képeken elvégezhesse a szükséges *képfeldolgozási*, *szövegfelismerési* és *nemfelismerési* feladatokat, így egyesítve a *két különböző technológia erősségeit.*

A Flask backend az előző fejezetekben bemutatott *három neurális hálózati modellt* integrálja.

A *modellek betöltése* során figyelembe kellett venni a különböző specialitásokat: a bounding box modellhez egy *egyéni metrikát* használtunk, az *OCR modell ONNX formátumban* került exportálásra, míg a *nemfelismerő modell* egy standard *Keras modell.* A nemfelismerő modell betanítása során a *képek méretét* (190×270 pixel) és a *normalizálási folyamat* is gondosan meg lett tervezve, hogy a személyi igazolványokon látható arcképek *pontosan feldolgozhatók* legyenek.

A Flask backend számos *képfeldolgozási funkciót* valósít meg, amelyek a neurális hálózatok előtt *előkészítik*, majd azok után feldolgozzák a képeket. Az egyik ilyen kulcsfontosságú funkció a *perspektív transzformáció*, amely a *határoló dobozban* azonosított területet *torzításmentessé* alakítja.

```
1   def four_point_transform(image, pts):
2
3       rect = np.zeros((4, 2), dtype="float32")
4
5       s = pts.sum(axis=1)
6       rect[0] = pts[np.argmin(s)]
7       rect[2] = pts[np.argmax(s)]
8
9       diff = np.diff(pts, axis=1)
10      rect[1] = pts[np.argmin(diff)]
11      rect[3] = pts[np.argmax(diff)]
12
13      widthA = np.sqrt(((rect[2][0] - rect[3][0]) ** 2) +
            ↪ ((rect[2][1] - rect[3][1]) ** 2))
14      widthB = np.sqrt(((rect[1][0] - rect[0][0]) ** 2) +
            ↪ ((rect[1][1] - rect[0][1]) ** 2))
15      maxWidth = max(int(widthA), int(widthB))
16
```

```
17          heightA = np.sqrt(((rect[1][0] - rect[2][0]) ** 2) +
                ↪ ((rect[1][1] - rect[2][1]) ** 2))
18          heightB = np.sqrt(((rect[0][0] - rect[3][0]) ** 2) +
                ↪ ((rect[0][1] - rect[3][1]) ** 2))
19          maxHeight = max(int(heightA), int(heightB))
20
21          dst = np.array([
22          [0, 0],
23          [maxWidth - 1, 0],
24          [maxWidth - 1, maxHeight - 1],
25          [0, maxHeight - 1]
26          ], dtype="float32")
27
28          M = cv2.getPerspectiveTransform(rect, dst)
29          warped = cv2.warpPerspective(image, M, (maxWidth,
                ↪ maxHeight))
30
31          return warped
```

6.7. kód. Határoló dobozzal határolt kép transzformációja

A 6.7. kódban látható függvény fogadja a képet és a *négy sarokpontot*, majd egy olyan *transzformációt* alkalmaz, amely a területet egy *téglalap alakúvá* alakítja, megszüntetve a *torzításokat*, amelyek a *fényképezés szögéből* adódhatnak. Ez rendkívül fontos az *OCR pontosságának* javításához.

A *nemfelismerő modell* az igazolványokon található *arcképből automatikusan meghatározza* a személy *nemét*. Ez egy fontos *kiegészítő információ*, amely növeli a rendszer használhatóságát. A nemfelismerő függvény a *bounding box modell* által azonosított *arcképterületet* dolgozza fel. A képet először *átméretezi* a modell által elvárt méretre (*190×270 pixel*), majd *normalizálja*, hogy a *pixelértékek 0 és 1 közé* essenek. A betanított modell *előrejelzést tesz* a személy nemére vonatkozóan, és visszaadja mind a *felismert nemet*, mind a *konfidencia értékét*.

A rendszerben fontos szempont a *biztonság*, ezért a *Flask API* csak megfelelő *API kulcs* birtokában érhető el. Ez biztosítja, hogy csak a saját *NodeJS backenddel* kommunikálhasson, és ne legyen kitéve *illetéktelen külső hozzáférésnek*. Ez az egyszerű, de hatékony *biztonsági mechanizmus* minden *API kérés előtt* ellenőrzésre kerül. Az API kulcs értéke egy *konfigurációs fájlban* található, és mindkét backenddel meg van osztva.

A rendszer *központi funkciója* a teljes *képfeldolgozási folyamat*, amely magában foglalja a *határdobozok felismerését*, az *adatmezők kiegyenesítését*, az *OCR feldolgozást* és a *nemfelismerést* is.

```
1       def process_image(image_path):
2           image = cv2.imread(image_path)
3           img_height, img_width = image.shape[:2]
4
```

```
5           resized_image = cv2.resize(image, (IMG_WIDTH,
                ↪ IMG_HEIGHT))
6
7           norm_image = resized_image / 255.0
8
9           pred_boxes =
                ↪ bbox_model.predict(np.expand_dims(norm_image,
                ↪ axis=0))[0]
10
11          pred_boxes_original = []
12          for i in range(0, len(pred_boxes), 2):
13              x = int(pred_boxes[i] * img_width)
14              y = int(pred_boxes[i+1] * img_height)
15              pred_boxes_original.append([x, y])
16
17          fields = {}
18          field_names = ["Név", "Személyi szám", "CAN",
                ↪ "Fénykép", "Születési dátum", "Lejárati dátum"]
19
20          face_image = None
21
22          for i, name in enumerate(field_names):
23              pts = np.array(pred_boxes_original[i*4:(i+1)*4],
                    ↪ dtype="float32")
24
25              warped = four_point_transform(image, pts)
26
27              if name == "Fénykép":
28                  face_image = warped
29                  continue
30
31              text = ocr_model.predict(warped)
32              fields[name] = text
33
34          if face_image is not None:
35              gender, confidence = detect_gender(face_image)
36              fields["Nem"] = gender
37              fields["Nem_konfidencia"] = confidence
38
39          return fields
```

6.8. kód. Képfeldolgozási folyamat

A 6.8. kódban látható *komplex függvény* több lépésben dolgozza fel a képet:

1. Betölti és *előkészíti* a képet a *bounding box modell* számára
2. Felismeri a *határdobozokat* a személyi igazolvány különböző mezői körül

3. Az egyes mezőket *kiegyenesíti* a *perspektív transzformáció* segítségével

4. A szöveges mezőkből *OCR* segítségével *kinyeri az információt*

5. Az arckép mezőt felhasználja a személy *nemének felismerésére*

6. Az eredményeket *strukturált formában* adja vissza

A Flask alkalmazás egyetlen, de *komplex API végpontot* biztosít a képfeldolgozáshoz. A végpont fogadja a *NodeJS backendtől* érkező képfájlokat, ellenőrzi az *API kulcsot*, *ideiglenesen elmenti* a képet, elvégzi a feldolgozást, majd visszaadja az eredményeket *JSON formátumban*. A rendszer gondoskodik az *ideiglenes fájlok törléséről*, függetlenül attól, hogy a feldolgozás sikeres volt-e vagy sem.

A Python Flask backend *Docker konténerben* fut, ami lehetővé teszi a *függőségek egyszerű kezelését* és a szolgáltatás *izolált futtatását*. A Docker konténer tartalmazza az összes szükséges függőséget, beleértve az *OpenCV-t*, a *TensorFlow-t* és a különböző *modellfájlokat*.

A *Docker Compose* segítségével a Flask és a NodeJS backend *együtt indítható*, biztosítva a két rendszer közötti *zökkenőmentes kommunikációt*. Ez a *konténerizált megközelítés* jelentősen egyszerűsíti a rendszer *telepítését* és *karbantartását*, mivel minden függőség és konfiguráció a Docker konténereken belül található.

6.5. React Native frontend

A projekt frontend része *React Native* keretrendszerben készült, amely lehetővé teszi, hogy *egyetlen kódbázisból* mind *Android*, mind *iOS* platformra natív alkalmazást fejleszthessünk. Az alkalmazás az *Expo platformot* használja, amely számos *beépített funkciót* biztosít, például a *kamera* kezeléséhez és a *biometrikus azonosításhoz*.

Az alkalmazás *navigációs rendszere* a *React Navigation* könyvtárra épül, amely strukturált és könnyen kezelhető navigációt biztosít a különböző képernyők között. A felhasználók egyszerűen válthatnak az alkalmazás fő funkciói között, amelyek logikusan elkülönített felületeken keresztül érhetők el.

Az alkalmazásba való belépés alapvetően *két módszerrel* történhet: hagyományos *e-mail és jelszó* megadásával, vagy *biometrikus azonosítással*. A fejlesztés során nagy hangsúlyt fektettem a biztonságos hitelesítésre, emellett az alkalmazás lehetőséget biztosít új felhasználók regisztrációjára is, ahol a személyes adatok megadását követően létrehozható egy új fiók.

A felhasználói adatok biztonságos kezeléséért az *AsyncStorage* felel, ahol az alkalmazás ideiglenesen tárolja a bejelentkezési információkat és a feldolgozott adatokat. Az alkalmazás feltöltési és szerkesztési funkciói az *Axios könyvtár* segítségével kommunikálnak a *backend API-val*, *JWT token alapú hitelesítéssel*.

Az alkalmazás egyik központi funkciója a *személyi igazolványok kezelése*. Itt a felhasználók választhatnak a *manuális adatbevitel* és a *kamera-alapú adatkinyerés* között. A kamera funkció két módon érhető el: közvetlenül az alkalmazáson belül készített felvétellel vagy a készülék galériájából választott kép segítségével.

Az alkalmazás beépített *képszerkesztési funkcióval* rendelkezik, amely lehetővé teszi a felhasználó számára, hogy a kép elkészítése során annak megfelelő részét vágja ki, így biztosítva a személyi igazolvány optimális feldolgozását. A kivágást az Expo *ImagePicker* modulja teszi lehetővé.

A képek elkészítése után a rendszer feltölti azokat a szerverre, ahol a *neurális hálók* felismerik és kinyerik a releváns adatokat. A felhasználók aztán ellenőrizhetik és szükség esetén módosíthatják ezeket az automatikusan kitöltött mezőket a dokumentum véglegesítése előtt.

Az adatok bevitele során az alkalmazás intuitív *dátumválasztót* biztosít a különböző időpontok, például a *lejárati dátum* vagy a *születési idő* megadásához. Az űrlapok validációval rendelkeznek, ami biztosítja a helyes adatformátumok használatát.

A felhasználók kezelhetik a *személyes profiljukat* is, ahol módosíthatják az elérhetőségi adataikat és biztonsági beállításaikat. Itt lehetőség van a *biometrikus bejelentkezés* be- és kikapcsolására is.

Az alkalmazás *felhasználói felülete* modern, letisztult dizájnnal rendelkezik. A különböző képernyőkön konzisztens vizuális elemeket használtunk, mint például gombokat, űrlapmezőket és ikonokat, amelyek harmonikusan illeszkednek egymáshoz. A *FontAwesome* ikonrendszer használata egységes vizuális élményt biztosít, míg a *React Native stíluslapok* lehetővé teszik a következetes megjelenést az egész alkalmazásban.

A felhasználói élményt tovább javítják a *visszajelző mechanizmusok*, mint például a betöltési animációk és az állapotjelzők, amelyek információt nyújtanak az aktuális folyamatokról. Az alkalmazás *reszponzív tervezése* biztosítja, hogy minden funkció megfelelően működjön a különböző képernyőméreteken és tájolásban.

A frontend fejlesztés során különös figyelmet fordítottunk a *felhasználóbarát élmény* kialakítására, minimalizálva a szükséges lépések számát, és intuitív módon vezetve a felhasználót a különböző funkciók között.

7. fejezet

Tesztelés

A szoftverfejlesztési folyamat egyik legfontosabb lépése a *tesztelés*, amely biztosítja, hogy az elkészült rendszer megfeleljen a *kitűzött követelményeknek*, és *hibamentesen* működjön a különböző felhasználási forgatókönyvekben. Ebben a fejezetben bemutatom, hogyan teszteltem a projekt különböző *komponenseit*, beleértve a *neurális hálózatokat*, a *frontend* és *backend* rendszereket, valamint a teljes rendszer *integrációját*. A tesztelési folyamat során különös figyelmet fordítottam a *neurális hálózatok pontosságának* mérésére, a *felhasználói élmény* biztosítására, valamint a rendszer *stabilitására* és *teljesítményére*.

A fejezet első részében a *neurális hálózatok tesztelését* mutatom be, amely magában foglalja az *OCR* (optikai karakterfelismerés), a *határoló doboz detektáló modell* és a *nemfelismerő modell* pontosságának kiértékelését. Ezt követően részletezem a *frontend* és *backend* komponensek tesztelési módszereit, majd bemutatom az *integrációs* és *rendszertesztek* eredményeit. Végül egy *használati útmutatót* is mellékelek, amely tartalmazza a rendszer működésének leírását és egy *use-case diagramot*.

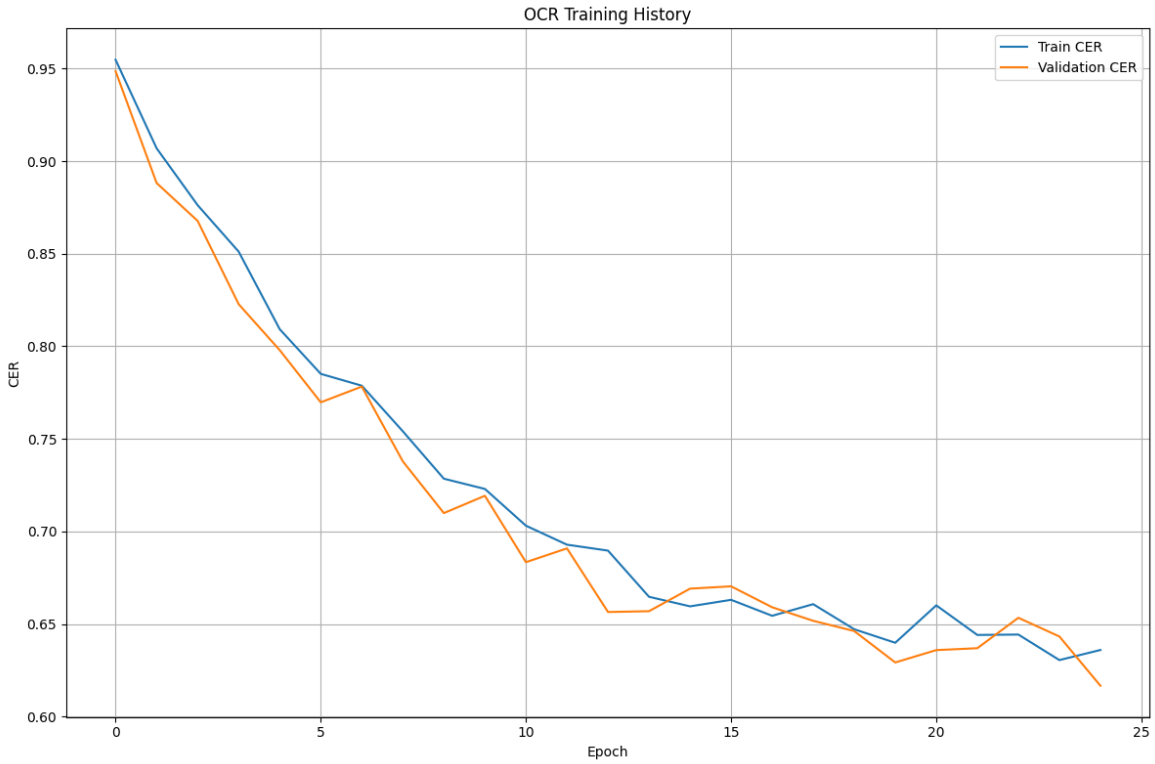
7.1. Neurális hálózatok tesztelése

A projektben három különböző *neurális hálózatot* használtam: az *OCR modellt*, a *határoló doboz detektáló modellt* és a *nemfelismerő modellt*. Ezek mindegyike *kulcsszerepet* játszik a személyi igazolványok adatainak *automatikus feldolgozásában*.

7.1.1. Pontosság és metrikák

A modellek teljesítményének kiértékeléséhez különböző *metrikákat* használtam, valamint elemeztem a tanulási folyamat során rögzített értékeket.

Az *OCR modell* tanítása során a CER értéke folyamatosan csökkent, ahogy a a 7.1 ábrán látható. A kezdeti 0,95-ös értékről a 25. epochra 0,62-re javult, ami jelentős előrelépés, bár még mindig magas hibaarányt jelez.



7.1. ábra. OCR modell tanítási és validációs karakterhiba-arány (CER) értékei, forrás: saját kép

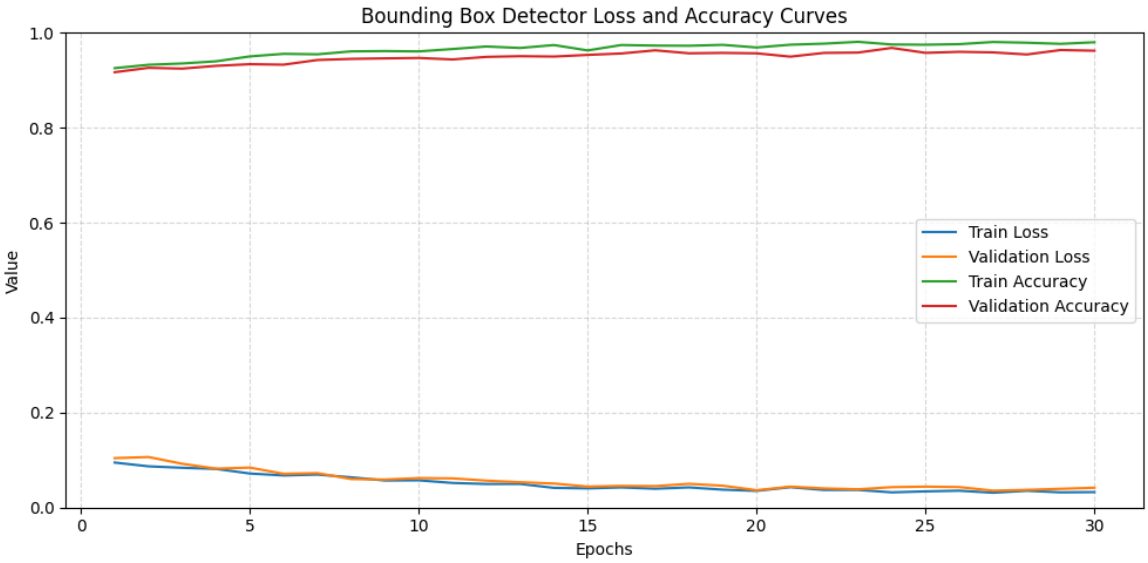
A *tesztelés* során kiderült, hogy a modell jelentős problémákkal küzd az *ékezetes karakterek* felismerése terén. Különösen az „á”, „é”, „ő” és „ű” betűk okoztak nehézséget, amelyeket gyakran ékezet nélküli megfelelőikként azonosított. Emellett a *szóközök felismerése* is következetlen volt, ami különösen a nevek feldolgozásánál okozott problémákat, mivel ezeket gyakran egybeírta vagy indokolatlanul több részre tagolta.

A *határoló doboz detektáló modell* jóval meggyőzőbb eredményeket mutatott, amint az a a 7.2 ábrán látható. A veszteségfüggvény értékei egyenletesen csökkentek, míg a pontosság már a korai epochokban is 90% feletti értéket ért el, majd 98% körül stabilizálódott.

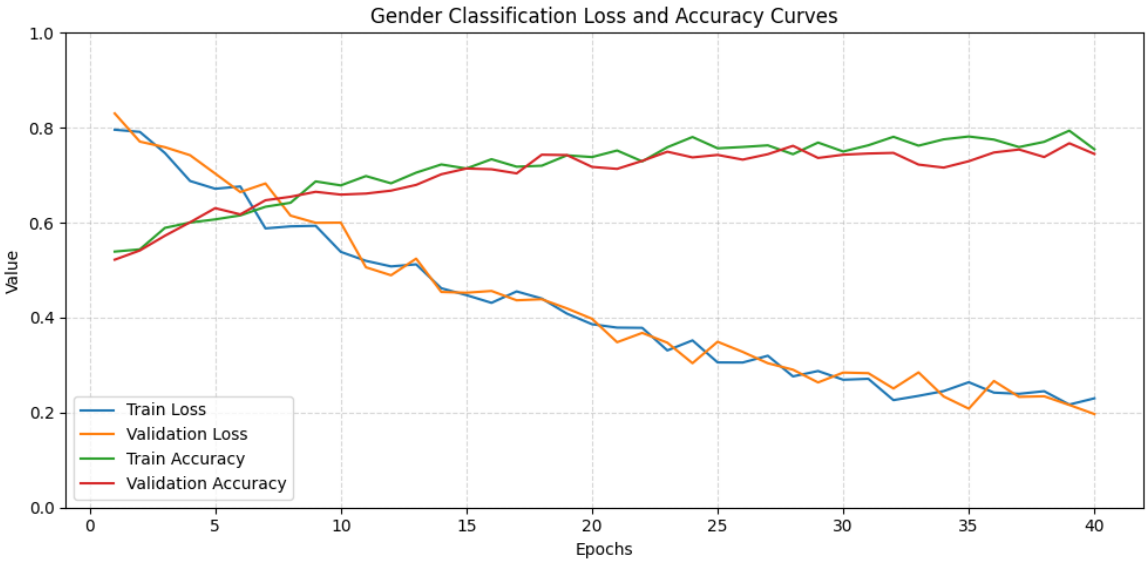
Ugyanakkor a tesztelés során kiderült, hogy a modell érzékeny a *perspektívatorzításra*. Amikor a felhasználók *ferdén fotózták* a személyi igazolványt (20 foknál nagyobb szögben), a határdobozok jelentősen *elcsúsztak*, ami az OCR modell számára tovább nehezítette a pontos szövegfelismerést. Különösen a kártya alsó részén található mezők (lejárati dátum, CAN kód) esetében volt észlelhető ez a probléma.

A *nemfelismerő modell* teljesítménye a a 7.3 ábrán látható. A validációs pontosság csak lassan emelkedett, és a görbék mutatnak némi instabilitást. Az accuracy értéke a 40. epochra is csak 75% körül stabilizálódott.

A nemfelismerő modell jelentős nehézségekkel küzdött bizonyos esetekben. A *hosszú hajú férfiakat* gyakran nőként azonosította, míg a *rövid hajú nőket* férfiként. Ez a je-



7.2. ábra. Határoló doboz modell tanítási folyamata – pontosság és veszteség, forrás: saját kép



7.3. ábra. Nemfelismerő modell tanítási és validációs metrikái, forrás: saját kép

lenség rávilágít, hogy a modell túlságosan a kulturális sztereotípiákra (pl. hajhosszra) támaszkodik a nemek meghatározásánál, ahelyett hogy az arc más jellemzőit venné figyelembe. Idősebb személyek esetében is magasabb volt a téves osztályozás aránya.

7.1.2. Eredmények kiértékelése

Az *OCR modell* kezdeti tesztelésekor alacsony pontosságot ért el. A *felismert karakterek* közel harmadában előfordult valamilyen *hiba*, különösen az *ékezetes betűk* esetében. A többszöri *finomhangolás* után azonban sikerült a teljesítményt jelentősen javítani.

A *határoló doboz detektáló modell* magas átlagos pontosságot ért el, ami összhangban van a tanulási görbén látható kiváló eredményekkel. A modell tanulási folyamata rendkívül stabil volt, már a korai szakaszban is magas teljesítményt mutatott. A perspektívatorzítás problémája azonban továbbra is fennáll, ami a valós használat során komoly korlátot jelenthet.

A *nemfelismerő modell* kezdeti *gyenge teljesítményét* jól tükrözi a tanulási görbe lassú emelkedése. Több javítási kísérlet után javult a pontosság, de még mindig elmarad az ideálistól. A modell túlzottan támaszkodik a felszíni jellemzőkre, mint a hajhossz vagy az arcforma, ami különösen problémás a nem szokványos megjelenésű személyek esetében.

A tanulási görbék elemzése alapján egyértelműen a *határoló doboz detektáló modell* teljesített a legjobban, míg a nemfelismerő modell maradt a leggyengébb láncszem. Az OCR modellnél a CER értéke még a tanítás végén is viszonylag magas maradt, ami jelzi, hogy további fejlesztésekre lenne szükség.

A *fejlesztési javaslatok* között szerepel az *OCR modell* számára nagyobb és változatosabb, különösen ékezetes karaktereket tartalmazó adathalmaz használata, a *nemfelismerő modellhez* komplexebb architektúra kidolgozása, amely kevésbé támaszkodik sztereotipikus megjelenési jellemzőkre, valamint a *határoló doboz modellhez* automatikus perspektíva-korrekció beépítése, amely kompenzálja a ferdén készített fényképek torzítását.

Ezek a fejlesztések együttesen javíthatják a rendszer *megbízhatóságát* és *pontosságát* valós használati körülmények között, bár a jelenlegi állapotában is használható alapot nyújt a személyi igazolványok feldolgozásához, különösen emberi ellenőrzés mellett.

7.2. Frontend tesztelése

A *frontend tesztelése* során a *React Native* alapú alkalmazás különböző *komponenseit* és *funkcióit* vizsgáltam. A cél az volt, hogy biztosítsam a *felhasználói felület helyes működését*, a *navigáció folyamatosságát*, valamint az *API-kal való megfelelő kommunikációt*. A tesztelési folyamat során *manuális* és *automatizált tesztelési módszereket*

alkalmaztam.

A *komponens szintű tesztelés* során a *React Native Testing Library* segítségével ellenőriztem az egyes *képernyők* és *funkciók* működését. Például az *IdCardDetailsScreen* esetében teszteltem, hogy a képernyő helyesen jeleníti-e meg a *betöltési állapotot*, a *hibaüzeneteket*, valamint a *személyi igazolvány adatait*. Az alábbi kódrészlet bemutatja, hogyan ellenőriztem, hogy a képernyő megfelelően kezeli a különböző *állapotokat*, például amikor *nincs adat*, vagy amikor a *betöltés sikeres*.

7.1. kód. Funkció teszt

```
1   import React from 'react';
2   import { render, screen, fireEvent } from '@testing-
        ↪ library/react-native';
3   import IdCardDetailsScreen from '../app/screens/
        ↪ IdCardDetailsScreen';
4
5   jest.mock('axios', () => ({
6       get: jest.fn(() => Promise.resolve({ status:
            ↪ 200, data: mockData })),
7   }));
8
9   const mockData = {
10      id_number: '123456789',
11      first_name: 'Gábor',
12      last_name: 'Kovács',
13      sex: 'férfi',
14      date_of_expiry: '2025-04-30',
15      place_of_birth: 'Budapest',
16      mothers_maiden_name: 'Nagy␣Anna',
17      can_number: '987654321',
18      date_of_birth: '1990-01-01',
19  };
20
21  describe('IdCardDetailsScreen', () => {
22      it('megjeleníti␣a␣betöltési␣állapotot', () => {
23          render(<IdCardDetailsScreen />);
24          expect(screen.getByText('Adatok␣betölté
                ↪ se...')).toBeTruthy();
25      });
26
27      it('megjeleníti␣a␣személyi␣igazolvány␣adatait',
            ↪  async () => {
28          render(<IdCardDetailsScreen />);
29          const name = await screen.findByText('
                ↪ Kovács␣Gábor');
30          expect(name).toBeTruthy();
31          expect(screen.getByText('123456789')).
```

```
                        ↪ toBeTruthy();
32                  expect(screen.getByText('Budapest')).
                        ↪ toBeTruthy();
33              });
34
35              it('megjeleníti␣a␣hibaüzenetet,␣ha␣nincs␣adat',
                    ↪  async () => {
36                  jest.mock('axios', () => ({
37                          get: jest.fn(() => Promise.reject
                                ↪ ({ response: { status: 404
                                ↪  } })),
38                  }));
39                  render(<IdCardDetailsScreen />);
40                  const errorMessage = await screen.
                        ↪ findByText('Nincs␣még␣feltöltött␣
                        ↪ személyi␣igazolvány␣adat');
41                  expect(errorMessage).toBeTruthy();
42              });
43      });
```

Az *integrációs tesztelés* során a *képernyők közötti navigációt* és az *API-kal való kommunikációt* vizsgáltam. Például ellenőriztem, hogy a *CameraScreen* helyesen navigál vissza az *IdCardScreen*-re a kép elkészítése után. A *manuális tesztelés* során különböző *forgatókönyveket* szimuláltam, például a *személyi igazolvány adatok feltöltését*, az *értesítések működését*, valamint a *hibakezelést*. Ezek a tesztek biztosították, hogy az alkalmazás helyesen működjön, és a felhasználók számára *zökkenőmentes élményt* nyújtson.

7.3. Backend tesztelése

A *backend tesztelése* során a cél az volt, hogy biztosítsam az *API végpontok helyes működését*, az *adatbázis műveletek megbízhatóságát*, valamint a rendszer *stabilitását* különböző *terhelési* és *hibás adatokkal kapcsolatos forgatókönyvekben*. A tesztelési folyamat során *manuális* és *automatizált teszteket* is alkalmaztam, hogy a rendszer minden komponense megfelelően működjön.

Az *API végpontok teszteléséhez* a *Postman* és az *automatizált tesztelési keretrendszerek*, például a *Jest* és a *Supertest* kombinációját használtam. A *Postman* segítségével *manuálisan ellenőriztem* az egyes végpontok válaszait, míg a *Jest* és a *Supertest* lehetővé tette az *automatizált tesztek* futtatását, amelyek *gyors visszajelzést* adtak a fejlesztési folyamat során.

A 7.2. kódban bemutatom, hogyan teszteltem a *képfeltöltési végpontot*, amely a feltöltött képeket továbbítja a *Flask backendnek* feldolgozásra.

7.2. kód. Backend tesztelés

```
1   const request = require('supertest');
2   const app = require('../app'); // Az Express alkalmazás
3   const path = require('path');
4
5   describe('POST␣/api/image/upload', () => {
6       it('sikeresen␣feltölti␣a␣képet␣és␣visszaadja␣az
        ↪ ␣eredményeket', async () => {
7           const response = await request(app)
8           .post('/api/image/upload')
9           .set('Authorization', 'Bearer␣
            ↪ valid_token') // Teszt token
10          .attach('images', path.resolve(__dirname
            ↪ , './test-image.jpg')); // Tesztk
            ↪ ép
11
12          expect(response.status).toBe(200);
13          expect(response.body).toHaveProperty('
            ↪ message', 'Images␣uploaded␣and␣
            ↪ processed␣successfully!');
14          expect(response.body).toHaveProperty('
            ↪ results');
15          expect(response.body.results.length).
            ↪ toBeGreaterThan(0);
16      });
17
18      it('hibaüzenetet␣ad␣vissza,␣ha␣nincs␣kép␣feltö
        ↪ ltve', async () => {
19          const response = await request(app)
20          .post('/api/image/upload')
21          .set('Authorization', 'Bearer␣
            ↪ valid_token');
22
23          expect(response.status).toBe(400);
24          expect(response.body).toHaveProperty('
            ↪ message', 'At␣least␣one␣image␣is␣
            ↪ required!');
25      });
26  });
```

Ez a teszt két *forgatókönyvet* vizsgál: az egyikben egy *érvényes képet* töltünk fel, és ellenőrizzük, hogy a *válasz tartalmazza a megfelelő eredményeket*, míg a másikban azt teszteljük, hogy a rendszer *megfelelően kezeli a hiányzó képfájlokat*. Az ilyen tesztek biztosítják, hogy a *backend API stabilan* és *megbízhatóan* működjön, még *váratlan helyzetekben* is.

7.4. Rendszerintegráció tesztelése

A *rendszerintegráció tesztelése* során az alkalmazás különböző *komponenseinek* (*frontend*, *backend*, *neurális hálózatok*) *együttműködését* vizsgáltam. A cél az volt, hogy biztosítsam az *adatok helyes áramlását* a *felhasználói felülettől* a *backendig*, majd a *neurális hálózatok* által végzett *feldolgozás* után vissza a *frontendig*. Ez a folyamat magában foglalta a *képfeltöltést*, az *adatok feldolgozását* és a *felhasználói értesítések kezelését*.

A tesztelés során különböző *forgatókönyveket* szimuláltam, például egy *személyi igazolvány képének feltöltését*, amelyet a *backend* továbbított a *Flask alapú képfeldolgozó modulnak*. A *neurális hálózatok* által kinyert adatokat a *backend API* visszaküldte a *frontendnek*, ahol azok *megjelenítésre kerültek*. Az *integrációs tesztek* során ellenőriztem, hogy a rendszer minden lépése *megfelelően működik*, és a felhasználó számára *érthető visszajelzést* nyújt.

A 7.3.kód bemutatja, hogyan teszteltem a *képfeltöltési folyamatot* és az *adatok visszaküldését* a *frontend* számára:

7.3. kód. Rendszerintegráció tesztelés

```
1   const request = require('supertest');
2   const app = require('../app'); // Az Express alkalmazás
3   const path = require('path');
4
5   describe('Rendszerintegráció␣teszt:␣Képfeltöltés␣és␣
        ↪ adatfeldolgozás', () => {
6       it('feldolgozza␣a␣feltöltött␣képet␣és␣visszakü
            ↪ ldi␣az␣adatokat', async () => {
7           const response = await request(app)
8           .post('/api/image/upload')
9           .set('Authorization', 'Bearer␣
                ↪ valid_token') // Teszt token
10          .attach('images', path.resolve(__dirname
                ↪ , './test-image.jpg')); // Tesztk
                ↪ ép
11
12          expect(response.status).toBe(200);
13          expect(response.body).toHaveProperty('
                ↪ message', 'Images␣uploaded␣and␣
                ↪ processed␣successfully!');
14          expect(response.body).toHaveProperty('
                ↪ results');
15          expect(response.body.results.length).
                ↪ toBeGreaterThan(0);
16          expect(response.body.results[0]).
                ↪ toHaveProperty('extracted_data');
17      });
18  });
```

Ez a teszt a rendszer teljes folyamatát lefedi, a *képfeltöltéstől* kezdve az *adatok feldolgozásán* át azok *visszaküldéséig*. Az ilyen tesztek biztosítják, hogy a rendszer minden *komponense megfelelően kommunikáljon* egymással, és a felhasználók számára *hibamentes élményt* nyújtson. Az *integrációs tesztelés* során feltárt hibák gyors javítása hozzájárult a rendszer *stabilitásának növeléséhez*.

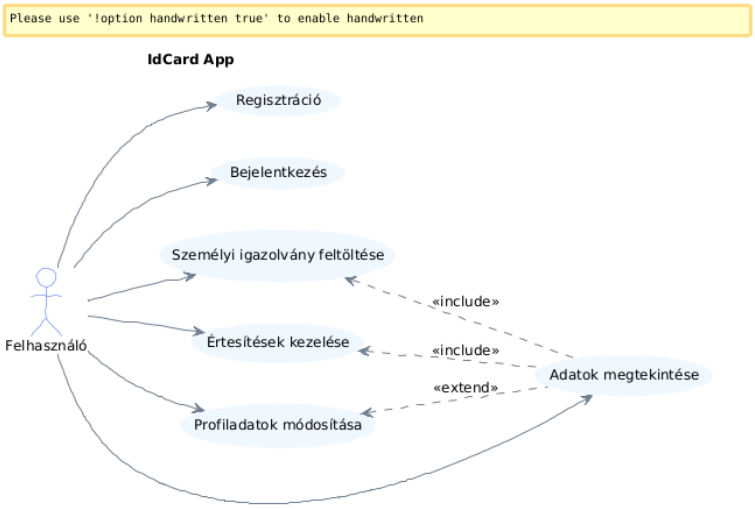
7.5. Használati útmutató

Az alkalmazás célja, hogy a felhasználók *egyszerűen* és *gyorsan* kezelhessék *személyi igazolványaik adatait*, *értesítéseket* kapjanak a *lejárati dátumokról*, és *biztonságosan tárolják* azokat. Az alábbiakban bemutatom az alkalmazás *telepítésének* és *használatának* lépéseit, valamint a leggyakoribb *felhasználói forgatókönyveket*.

Az alkalmazás *telepítése* egyszerű folyamat, amelyhez először le kell tölteni az alkalmazást. A telepítés után az alkalmazás első indításakor a felhasználónak *regisztrálnia* kell egy fiókot, vagy be kell jelentkeznie egy *meglévő fiókkal*. A *regisztráció* során meg kell adni az *e-mail címet*, *jelszót* és egyéb alapvető adatokat. A *bejelentkezés* után az alkalmazás *automatikusan szinkronizálja* a felhasználó adatait a *szerverrel*. Az alkalmazás működéséhez *internetkapcsolat* szükséges, valamint a *push értesítések* engedélyezése ajánlott, hogy a felhasználó időben értesüljön a személyi igazolvány *lejárati dátumáról*. Az értesítések engedélyezéséhez az alkalmazás első indításakor megjelenő kérésre kell pozitívan válaszolni.

Az alkalmazás számos *tipikus forgatókönyvet* támogat, amelyek a felhasználók mindennapi igényeit szolgálják. A *használati esetek* láthatók a 7.4. ábrán. Az egyik leggyakoribb *használati eset* a *személyi igazolvány adatainak feltöltése*. Ehhez a felhasználó a főmenüből elérheti a *"Feltöltés"* képernyőt, ahol *manuálisan megadhatja az adatokat*, vagy *fényképet készíthet* az igazolványról. A fénykép készítése után az alkalmazás *automatikusan feldolgozza a képet*, és *kinyeri az adatokat*, amelyeket a felhasználó *ellenőrizhet* és szükség esetén *módosíthat*. Egy másik gyakori forgatókönyv az *értesítések kezelése*. Az alkalmazás *automatikusan értesítést küld*, ha a személyi igazolvány *lejárati dátuma* egy hónapon belül van. Az értesítésre kattintva a felhasználó közvetlenül az *"Adataim"* képernyőre navigálhat, ahol ellenőrizheti a lejárati dátumot és szükség esetén frissítheti az adatokat. A felhasználók továbbá *módosíthatják profiladataikat*, például az *e-mail címüket* vagy *jelszavukat*. Ehhez a *"Profil"* képernyőt kell megnyitniuk, ahol a szükséges módosítások elvégezhetők. Az alkalmazás minden *adatot biztonságosan tárol*, és a változtatásokat *azonnal szinkronizálja* a szerverrel.

Az alkalmazás használata során előfordulhatnak kisebb *problémák*, amelyek gyorsan orvosolhatók. Ha például a *fényképezőgép nem működik*, ellenőrizni kell, hogy az alkalmazás rendelkezik-e a szükséges *engedélyekkel*. Ezt a készülék beállításaiban lehet



7.4. ábra. Használati eset diagram, forrás: saját kép

megtenni. Ha az alkalmazás *nem küld értesítéseket*, győződjön meg róla, hogy az *értesítések engedélyezve vannak*, és az *internetkapcsolat* megfelelően működik. Amennyiben a felhasználó *nem tud bejelentkezni*, érdemes ellenőrizni az *e-mail címet* és a *jelszót*. Ha a probléma továbbra is fennáll, a *"Jelszó visszaállítása"* funkcióval új jelszót lehet kérni. Ha az alkalmazás *nem szinkronizálja az adatokat*, próbálja meg *újraindítani* az alkalmazást, vagy ellenőrizze az *internetkapcsolatot*.

Az alkalmazás célja, hogy a felhasználók számára *egyszerű* és *intuitív élményt* nyújtson, miközben biztosítja az *adatok biztonságos kezelését* és a *fontos értesítések* időben történő kézbesítését.

7.6. Dokumentáció

A projekt *dokumentációja* kettő fő területre terjed ki: a *forráskód dokumentálására*, és az *API végpontok részletes leírására*. A cél az volt, hogy a rendszer működése a fejlesztők számára *átlátható* és *könnyen érthető* legyen. A dokumentáció biztosítja a projekt *hosszú távú fenntarthatóságát*, és segíti a későbbi fejlesztéseket.

A *forráskód dokumentálása* során *következetes stílust* alkalmaztam, hogy a kód *könnyen olvasható* és *karbantartható* legyen. A *TypeScript* és *JavaScript* fájlokban *JSDoc szintaxist* használtam, amely lehetővé teszi a *fejlesztői dokumentáció automatikus generálását*. Minden *függvény*, *osztály* és *modul* részletes magyarázatot kapott, amely tartalmazza a *funkcionalitás leírását*, a *paraméterek* és *visszatérési értékek* részleteit, valamint a *használat példáit*. A *JSDoc kommentekből* automatikusan generálható *HTML dokumentáció* a *jsdoc eszköz* segítségével. A generált dokumentációt a fejlesztők *könnyen böngészhetik*, ami különösen hasznos a nagyobb projektek esetén.

Az *API dokumentáció* különösen fontos a fejlesztők számára, mivel bemutatja az *elérhető végpontokat*, azok *működését*, a *szükséges paramétereket*, valamint a *lehetséges válaszokat*. A dokumentációt *OpenAPI (Swagger)* formátumban készítettem el, amely *JSON vagy YAML fájlban* tárolható, és könnyen integrálható *Swagger UI* vagy más eszközökkel.

A *dokumentációs fájlok karbantartása* és *frissítése* biztosítja, hogy a rendszer mindig *naprakész* legyen, és a fejlesztők *gyorsan hozzáférjenek* a szükséges információkhoz.

Összegzés

A szakdolgozat során bemutattam a személyazonosítás fejlődését és digitalizációját, különös tekintettel a modern technológiák, például a mesterséges intelligencia és a mobilalkalmazások szerepére. A dolgozat célja egy olyan rendszer fejlesztése volt, amely a digitális személyazonosítás kihívásaira nyújt megoldást, miközben a felhasználók számára gyors, biztonságos és kényelmes élményt biztosít.

A dolgozat első részében ismertettem a személyazonosítás történetét, a digitális azonosítás megjelenését, valamint a mobiltechnológia és a biometrikus azonosítás szerepét. Ezt követően bemutattam a mesterséges intelligencia alkalmazását az adatfeldolgozásban, különös tekintettel az optikai karakterfelismerésre (OCR) és a neurális hálók működésére. A TensorFlow és a Keras könyvtárak segítségével olyan modelleket fejlesztettem, amelyek képesek a személyi igazolványokról készült képekből releváns adatokat kinyerni.

A backend fejlesztés során a Node.js és az Express.js keretrendszerekre építettem, amelyek lehetővé tették a gyors és skálázható adatkezelést. Az adatbázis-kezeléshez a MongoDB-t használtam, amely rugalmas és hatékony megoldást nyújtott a változó szerkezetű adatok tárolására. A frontend fejlesztéshez a React Native és az Expo keretrendszereket alkalmaztam, amelyek lehetővé tették a platformfüggetlen mobilalkalmazás fejlesztését.

A projekt során sikerült egy olyan rendszert létrehozni, amely a személyi igazolványok digitalizálását és azonosítását valósítja meg. Az alkalmazás képes a felhasználók által feltöltött képek feldolgozására, az adatok kinyerésére és azok biztonságos tárolására. A rendszer emellett értesítéseket küld a felhasználóknak a személyi igazolványuk lejárati idejéről, ezzel is növelve a felhasználói élményt és a hatékonyságot.

Bár a projekt során elértem a kitűzött célokat, a rendszer továbbfejlesztésére számos lehetőség kínálkozik. Például a neurális hálók pontosságának növelése további adatok bevonásával, a kép augmentálási technikák alkalmazása a modell általánosítási képességének javítása érdekében, valamint a rendszer teljesítményének optimalizálása a felhasználói élmény további javítására. Ezek a fejlesztések hozzájárulhatnak ahhoz, hogy a rendszer még hatékonyabbá és szélesebb körben alkalmazhatóvá váljon.

Irodalomjegyzék

[1] Rosta Erzsébet, *A daktiloszkópia története*, 2009 https://erzsebetrosta.hu/borlec-rajzolatok-dermatoglyphia/borrajzolatok-tudomanya/a-daktiloszkopia-tortenete.html(Utoljára ellenőrizve: 2025.04.22.)

[2] Múlt-kor, *Tetoválástól az arcfelismerésig: a személyazonosítás története*, 2024 https://m.mult-kor.hu/tetovalastol-az-arcfelismeresig-a-szemelyazonositas-tortenete-20241206(Utoljára ellenőrizve: 2025.04.22.)

[3] Kiss Tibor, Szegő Tamás *A személyazonosítás múltja, jelene és jövője*, 2017 https://www.kozszov.org.hu/dokumentumok/UMK_2017/2/06_A_szemelyazonositas_multja.pdf(Utoljára ellenőrizve: 2025.04.22.)

[4] Európai Unió, *Az Európai Parlament és a Tanács (EU) 910/2014 rendelete (2014. július 23.) az elektronikus azonosításról és a bizalmi szolgáltatásokról a belső piacon történő elektronikus tranzakciókhoz*, Az Európai Unió Hivatalos Lapja, L 257/73, 2014. (Utoljára ellenőrizve: 2025.04.22.)

[5] Európai Unió, *Az Európai Parlament és a Tanács (EU) 2024/1183 rendelete (2024. április 11.) az európai digitális személyazonossági keret létrehozásáról és a 910/2014/EU rendelet módosításáról*, Az Európai Unió Hivatalos Lapja, L 257/1, 2024. (Utoljára ellenőrizve: 2025.04.22.)

[6] A.D.Dongare, R.R.Kharde, Amit D.Kachare, *Introduction to Artificial Neural Network*, 2012 https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=04d0b6952a4f0c7203577afc9476c2fcab2cba06 (Utoljára ellenőrizve: 2025.04.22.)

[7] GeeksforGeeks, *Artificial Neural Networks and its Applications*, 2024 https://www.geeksforgeeks.org/artificial-neural-networks-and-its-applications/ (Utoljára ellenőrizve: 2025.04.22.)

[8] eSzemélyi, *AZ eSZEMÉLYI*, é. n. https://eszemelyi.hu/az-eszemelyi/ (Utoljára ellenőrizve: 2025.04.22.)

[9] Ian Goodfellow, Yoshua Bengio, Aaron Courville, *Deep Learning*, MIT Press, 2016. (Utoljára ellenőrizve: 2025.04.22.)

[10] Yann LeCun, Yoshua Bengio, Geoffrey Hinton, *Deep learning*, Nature 521, 436–444, 2015. (Utoljára ellenőrizve: 2025.04.22.)

[11] Viola, P. and Jones, M., *Rapid object detection using a boosted cascade of simple features*, CVPR, 2001. (Utoljára ellenőrizve: 2025.04.22.)

[12] Adrian Rosebrock, *Deep Learning for Computer Vision with Python*, PyImageSearch, 2017. (Utoljára ellenőrizve: 2025.04.22.)

[13] Jain, A. K., Nandakumar, K., and Ross, A., *50 Years of Biometric Research: Accomplishments, Challenges, and Opportunities*, Pattern Recognition Letters, 2016. (Utoljára ellenőrizve: 2025.04.22.)

[14] Mezei István, *Mobilalkalmazások fejlesztése React Native keretrendszerben*, Debreceni Egyetem, 2022. (Utoljára ellenőrizve: 2025.04.22.)

[15] TensorFlow, *TensorFlow Documentation*, 2025 `https://www.tensorflow.org/` (Utoljára ellenőrizve: 2025.04.22.)

[16] Keras, *Keras Documentation*, 2024 `https://keras.io/` (Utoljára ellenőrizve: 2025.04.22.)

[17] Kingma, D. P., Ba, J., *Adam: A Method for Stochastic Optimization*, arXiv preprint arXiv:1412.6980, 2014. (Utoljára ellenőrizve: 2025.04.22.)

[18] Tilkov Stefan, Vinoski Steve, *Node.js: Using JavaScript to Build High-Performance Network Programs*, IEEE Internet Computing, 2010. (Utoljára ellenőrizve: 2025.04.22.)

[19] Cantelon, M., Harter, M., Holowaychuk, T.J., Rajlich, N., *Node.js in Action (2nd Edition)*, Manning Publications, 2020. (Utoljára ellenőrizve: 2025.04.22.)

[20] Holmes, Alex, *Full-Stack Web Development with MongoDB and Express.js*, Packt Publishing, 2018.

[21] MongoDB, Inc., *Mongoose Documentation*, 2025 `https://mongoosejs.com` (Utoljára ellenőrizve: 2025.04.22.)

[22] MongoDB, Inc., *MongoDB Documentation*, 2025 `https://www.mongodb.com/docs/` (Utoljára ellenőrizve: 2025.04.22.)

[23] Shannon Bradshaw, Eoin Brazil, Kristina Chodorow, *MongoDB: The Definitive Guide (3rd Edition)*, O'Reilly Media, 2019.

[24] Meta Platforms, Inc., *React Native Documentation*, 2025 `https://reactnative.dev/docs` (Utoljára ellenőrizve: 2025.04.22.)

[25] Bonnie Eisenman, *Learning React Native (2nd Edition)*, O'Reilly Media, 2017. (Utoljára ellenőrizve: 2025.04.22.)

[26] Expo, *Expo Documentation*, 2025 `https://docs.expo.dev` (Utoljára ellenőrizve: 2025.04.22.)

[27] Adam Boduch, Roy Derks, *React and React Native (3rd Edition)*, Packt Publishing, 2020. (Utoljára ellenőrizve: 2025.04.22.)

[28] Rémy Philippe, *Name Dataset*, GitHub Repository, 2025 `https://github.com/philipperemy/name-dataset`

[29] Bhatt J., *Selfie Image Detection Dataset*, Kaggle, 2025 `https://www.kaggle.com/datasets/jigrubhatt/selfieimagedetectiondataset/data`. (Utoljára ellenőrizve: 2025.04.22.)

[30] Kovács Gábor, *Kovacs_Gabor_Szakdolgozat*, GitHub Repository, 2024 `https://github.com/kovacsgabor0730/Kovacs_Gabor_Szakdolgozat` (Utoljára ellenőrizve: 2025.04.22.)

[31] Rokas Liuberskis, *01_image_to_word*, GitHub Repository, 2022 `https://github.com/pythonlessons/mltu/tree/main/Tutorials/01_image_to_word` (Utoljára ellenőrizve: 2025.04.22.)

Nyilatkozat

Alulírott *Kovács Gábor*, büntetőjogi felelősségem tudatában kijelentem, hogy az általam benyújtott, *Digitális személyazonosítás mesterséges intelligenciával* című szakdolgozat önálló szellemi termékem. Amennyiben mások munkáját felhasználtam, azokra megfelelően hivatkozom, beleértve a nyomtatott és az internetes forrásokat is.

Aláírásommal igazolom, hogy az elektronikusan feltöltött és a papíralapú szakdolgozatom formai és tartalmi szempontból mindenben megegyezik.

Eger, 2025. április 22.

Kovács Gábor

aláírás