Széchenyi István Egyetem
Informatika és Villamosmérnöki Kar
Képfeldolgozás

Dokumentum szkennelés képfeldolgozása és szövegfelismerése

Készítette:
Velekey Ádám
H3Q42Y
Mérnökinformatikus 2025

Oktató neve: Dr. Hollósi János

Tartalomjegyzék

1. Bevezetés

A digitális technológiák rohamos fejlődése az utóbbi évtizedben gyökeresen átalakította azt, ahogyan a különböző dokumentumokat kezeljük, tároljuk és továbbítjuk. A hagyományos, papíralapú iratkezelés egyre inkább háttérbe szorul, helyét pedig átveszik a különböző digitális dokumentumkezelő rendszerek. Bár számos területen sikerült teljes mértékben áttérni elektronikus formátumokra, még mindig nagyon sok olyan feladat létezik, ahol papírra írt vagy papíron megjelenő információkat kell feldolgozni. Ilyen lehet többek között a kézzel kitöltött nyomtatványok digitalizálása, oktatási anyagok archiválása, felmérések rögzítése, vagy akár egyszerű jegyzetek átalakítása kereshető, szerkeszthető szöveggé.

A papíralapú dokumentumok digitalizálásának egyik legfontosabb eszköze az OCR (Optical Character Recognition), vagyis az optikai karakterfelismerés, amely lehetővé teszi, hogy egy képen látható szöveg számítógép által értelmezhető formába kerüljön. Az OCR technológiák ma már széles körben elérhetőek, azonban továbbra is számos kihívással kell szembenézniük, különösen akkor, ha a szöveg nem nyomtatott, hanem kézzel írt, vagy ha a kép minősége nem optimális.

Jelen projekt célja egy olyan Python alapú képfeldolgozó és szövegfelismerő rendszer megvalósítása, amely képes egy mobiltelefonnal készített fényképről automatikusan felismerni a kézzel írt szöveget, majd azt összehasonlítani egy elvárt, a felhasználó által megadott referenciaszöveggel. Ez különösen hasznos lehet olyan helyzetekben, amikor ellenőrizni kell, hogy egy felhasználó megfelelően írta-e le egy feladat megoldását, vagy dokumentálni kell egy papírra felírt információt gyorsan és hatékonyan.

A feladat több fontos részből áll. Először is szükség volt a kép megfelelő előfeldolgozására, amely magában foglalja a papírlap detektálását, a kép kiegyenesítését, a kontrasztjavítást és a zajszűrést. Ez alapvető fontosságú, hiszen az OCR motorok, még a modern, deep learning alapú rendszerek is, rendkívül érzékenyen reagálnak a fényviszonyokra, az árnyékokra, a torzításokra vagy a rossz fókuszra. Ha a bemeneti kép nem megfelelő minőségű, a felismert szöveg pontossága drasztikusan csökkenhet.

A projektben két különböző OCR rendszert hasonlítottam össze:

I. EasyOCR, amely egy modern, neurális hálókra épülő megoldás,
II. Tesseract, amely a legelterjedtebb, nyílt forráskódú karakterfelismerő motor.

Mindkét megoldásnak megvannak az előnyei és hátrányai: az EasyOCR hajlamos jobban teljesíteni kézzel írt és fényképezett dokumentumokon, míg a Tesseract általában stabilabb a nyomtatott szövegek felismerésében. A projekt során lehetőség nyílt arra, hogy közvetlenül összehasonlítsam a két rendszer eredményességét különböző fényviszonyok, papírtípusok és felírások esetében

A fejlesztés során kiemelt figyelmet kapott a felhasználóbarát működés. Mivel a cél az volt, hogy a program minimális előismerettel is egyszerűen használható legyen, a rendszer automatikusan végzi a kép előfeldolgozását és a papírlap detektálását. A felhasználónak csupán ki kell választania a fényképet, majd a program minden további lépést elvégez:

1. a kép elemzését,
2. a papír kivágását és kiegyenesítését,
3. a binarizációt,
4. az OCR futtatását,
5. végül az eredmény százalékos összehasonlítását.

A projekt gyakorlati haszna mellett oktatási szempontból is jelentős, hiszen kiválóan bemutatja a képfeldolgozás és a mesterséges intelligencia eszköztárának gyakorlati alkalmazását egy egyszerű, valós problémára. A dokumentáció célja, hogy részletesen és érthetően mutassa be a fejlesztéshez szükséges elméleti hátteret, az alkalmazott módszerek működését, a rendszer megvalósítását, valamint az elvégzett tesztek eredményeit és azok értelmezését.

Összefoglalva tehát a projekt fő célja egy működő, Python alapú dokumentumszkennelő és OCR alkalmazás létrehozása volt, amely képes kézzel írt szöveget felismerni és összehasonlítani egy referencia alapján. A fejlesztett rendszer megbízhatóan működik, felhasználóbarát, és megfelelő eszközt biztosít a képfeldolgozás és szövegfelismerés folyamatának megértéséhez.

2. A dokumentum digitalizálás és szövegfelismerés elméleti alapjai

2.1. A képfeldolgozás szerepe a dokumentum digitalizálásban

A dokumentumok digitalizálása ma már meghatározó szerepet tölt be a vállalati, intézményi és magánfelhasználói környezetekben is. A digitalizáció célja elsősorban a papír alapú információk hosszú távú tárolása, kereshetősége és feldolgozhatósága. A folyamat első és legfontosabb lépése a képi adatok megfelelő minőségű előállítása, majd ezek feldolgozása a további automatizált műveletekhez. Itt lép be a képfeldolgozás – egy olyan eszközkészlet, amely lehetővé teszi a nyers fénykép feldolgozását, strukturálását és javítását.

A digitalizált dokumentumok esetében a legnagyobb kihívást az jelenti, hogy a felvétel gyakran nem optimális körülmények között készül. A mobiltelefonos kamera olyan környezeti hibákat hoz létre, mint például a kép ferdesége, a papírlap torzulása, az árnyékok, a túl erős vagy gyenge megvilágítás, valamint a háttérvizuális elemek zavaró jelenléte. Mindezek rontják a szövegfelismerés (OCR – Optical Character Recognition) hatékonyságát. Az OCR-rendszerek alapvetően homogén, nagy kontrasztú, torzításmentes dokumentumokra optimalizáltak, így egy nyers telefonos fotó gyakran használhatatlanul gyenge kiindulási alap.

A képfeldolgozás technikái azonban lehetővé teszik, hogy a kép és a tartalom minősége mesterségesen javítható legyen. A különböző algoritmusok segítségével:

1. eltávolítható a háttérzaj, amely félrevezetné az OCR-t,
2. növelhető a kontraszt a szöveg és a háttér között,
3. korrekcióra kerülhet a perspektíva, így a papír derékszögű téglalappá alakítható,
4. kiegyenesíthetők az elfordult képek,
5. kiszűrhetők az árnyékok és nem egységes fényeloszlás,
6. kiemelhetők a karakterkontúrok, ami hatékonyabb felismerést eredményez.

A projektben alkalmazott képfeldolgozási lépések kulcsfontosságúak, mivel önmagában egyik OCR-rendszer sem képes megfelelő minőségben értelmezni a természetes környezetben készített fotót. Az előfeldolgozás - mint köztes folyamat - biztosítja, hogy a képek olyan formában kerüljenek az OCR algoritmusok elé, amely megfelel azok működési feltételeinek.

Egy megfelelően előkészített dokumentumfotó esetén a szöveg felismerési pontosság akár 30–60%-kal is javulhat, ami kritikus különbség lehet egy automatizált rendszer esetében. A projekt során tehát nem csupán az OCR működik, hanem egy komplex képfeldolgozó pipeline, ahol minden lépés hozzájárul a végső eredmény minőségéhez.

A képfeldolgozás tehát nem önmagáért való tevékenység, hanem az OCR-rel együtt alkot egy teljes rendszert. A digitalizálás minősége ezen a ponton dől el: az algoritmus által látott kép határozza meg, milyen hatékonysággal tudja értelmezni a tartalmat. Ezért a folyamatban kiemelten fontos az olyan technikák alkalmazása, mint a kontúrdetektálás, a perspektívakorrekció, a zajszűrés vagy a binarizáció.

2.2. A digitális képek felépítése és jellemzői

Ahhoz, hogy a dokumentum digitalizálásához szükséges képfeldolgozási lépéseket megértsük, először részletesen meg kell vizsgálni a digitális képek felépítését. Egy digitális kép valójában nem más, mint egy kétdimenziós mátrix, ahol minden elem - a pixel - a kép egy adott pontjának fényintenzitását és színinformációját hordozza. A képfeldolgozó algoritmusok kizárólag ezekből a numerikus értékekből képesek következtetni a dokumentum szerkezetére, így a pixelek jelentősége alapvető.

A modern kamerák - legyen szó mobiltelefonról vagy professzionális fényképezőgépről - rendkívül nagy felbontású képeket rögzítenek. A felbontás tipikusan megapixelben mérhető, például egy 12 MP-es kamera 4032×3024 képpontból álló képet hoz létre. Ez hatalmas adatállomány, és minden pixel három csatornával rendelkezik (piros, zöld és kék színkomponensek - RGB). A képfeldolgozás során azonban gyakran nincs szükség erre a részletgazdagságra, különösen egy fekete-fehér papírdokumentum esetében, ezért a színcsatornák csökkentése (grayscale konverzió) hatékonyan egyszerűsíti a további műveleteket.

A digitális képek jellemzői közül a legfontosabbak:

2.2.1. Felbontás és mintavételezés

A felbontás határozza meg, mennyi részletet tartalmaz a kép. Minél nagyobb a felbontás, annál több pixel áll rendelkezésre a dokumentum leírására, ami növeli a karakterek élességét. Ugyanakkor a túl magas felbontás lelassíthatja az algoritmusokat, ezért optimális egyensúlyt kell találni. A dokumentumok feldolgozásához gyakran elegendő a kép méretét 1000-1500 pixel szélességre átméretezni.

2.2.2. Színmodellek és csatornák

A projektben használt színmodell:

- RGB a nyers adat beolvasásakor,
- Grayscale a kontrasztnöveléshez és a binarizációhoz.

A grayscale kép egyetlen csatornával rendelkezik (0-255 között), ahol a nagyobb érték a világosabb területet, az alacsonyabb sötétebbet jelöl. Ez ideális a szöveg-háttér elkülönítéséhez.

2.2.3. Zajforrások a dokumentumképeken

A képminőséget nagyban ronthatják:

1. a háttér mintái (pl. fafelület erezete),
2. szenorzaj (nagy ISO értéknél),
3. mozgás miatti elmosódás,
4. fényerőkülönbségek,
5. árnyékok, ujj árnyéka a papíron,
6. torz perspektíva.

Mivel az OCR ezeket gyakran karakternek érzékeli, a feldolgozás során elengedhetetlen a zajcsökkentő és simító műveletek alkalmazása.

2.2.4. Hisztogram és fényeloszlás

A dokumentumoknál kritikus a szöveg-háttér kontraszt.
A fényeloszlás ingadozása (pl. fénycsík a papíron) megnehezíti a küszöbölést.
Ennek kezelése:

- Gaussian Blur,
- Otsu-küszöbölés,
- adaptív thresholding (nem használtuk, de alternatíva).

2.2.5. Bitmélység

A digitális kép bitmélysége a rendelkezésre álló árnyalatok számát jelzi.
8 bit → 256 árnyalat (OCR-hez bőven elég)
24 bit (RGB) → 16 millió szín
A dokumentumoknál a színinformáció ritkán szükséges, így a bitmélység csökkentése gyorsítja az algoritmust.

2.3. A zajok és képdegradáció típusai dokumentumfotókon

A dokumentumfotók minősége nagyban befolyásolja a szöveg felismerési rendszerek megbízhatóságát. Bár a modern kamerák fejlett optikával, szenzorral és képfeldolgozó elektronikával rendelkeznek, a dokumentumok fényképezése továbbra is rengeteg hibalehetőséget hordoz. A képfeldolgozás során e hibák többnyire a kép torzulásaként vagy zajként jelennek meg, amelyek nagymértékben ronthatják az OCR (Optical Character Recognition) eredményességét. A következő alfejezetekben bemutatjuk, milyen jellegű degradációk fordulnak elő leggyakrabban a kézzel készített dokumentumfotókon, és hogyan hatnak az automatikus szövegfelismerésre.

2.3.1. Optikai és szenorzaj

A digitális kamerák által rögzített képek egyik alapvető problémája az ún. szenorzaj. Ez olyan véletlenszerű fény- és színeltérésekből áll, amelyek nem részei a valódi képnek, hanem a kamera elektronikus működéséből fakadnak. A zaj különösen gyenge fényviszonyok mellett válik hangsúlyossá, mert ilyenkor a kamera automatikusan növeli az érzékenységét (ISO-t), ami a pixelintenzitások véletlenszerű ingadozásához vezet. A dokumentumképeken ez apró szemcsézettségként jelenik meg, amely eltorzíthatja a betűk körvonalait, vagy zajként beleolvadhat a szövegbe.

A szenorzaj azért különösen problémás az OCR számára, mert a szövegfelismerő rendszer érzékelő algoritmusai a karakterek alakjából és kontrasztjából próbálják azonosítani a betűket. Ha a betűk élei nem tiszták, vagy ha a háttérbe véletlenszerű fényes pontok kerülnek, akkor a karakterek felismerése bizonytalanná válik. A projektben alkalmazott Gaussian Blur módszer éppen ezt a fajta zajt csökkenti, azáltal, hogy kisimítja a pixelintenzitásokat, és eltünteti a nagyon apró, magas frekvenciájú hibákat.

2.3.2. Mozgás miatti elmosódás

A dokumentumfotók egyik legjellegzetesebb hibája a bemozdulás. Egy minimális kézremegés vagy a kamera rázkódása már elegendő ahhoz, hogy a karakterek kontúrjai kissé elmosódjanak. A betűk szélei ilyenkor nem élesek, hanem elkenődnek, széles csíkokká alakulnak, vagy „kettős kontúr” jelenik meg. Az emberi szem számára az ilyen kis elmosódás

még többnyire értelmezhető marad, hiszen az agy kiegészíti a hiányzó információt. Az OCR azonban a karakter alakjának pixelpontos elemzésére támaszkodik, ezért az elmosódott kontúrok könnyen tévesztésekhez vezetnek.

A bemozdulás különösen akkor jelent problémát, ha a felhasználó egy kézzel tartja a telefont, és rossz fényviszonyok miatt a kamera hosszabb záridőt választ. Minél hosszabb ideig tart a kép rögzítése, annál nagyobb az esélye a mozgásos torzulásnak. A projektben alkalmazott előfeldolgozási lépések - például a kép átméretezése és lágyítása - részben enyhítik ezt, de a bemozdulás teljes korrekciójára csak speciális, nagy számítási igényű algoritmusok képesek.

2.3.3. Árnyékok és egyenetlen megvilágítás

A dokumentumfotók egyik legproblémásabb eleme az árnyék. Ez lehet a felhasználó kezének árnyéka, a telefonkészülék árnyéka, vagy a környezetből érkező fény miatt kialakult kontrasztos megvilágítási hiba. Egy árnyékos rész a papíron teljesen más intenzitással rendelkezik, mint a dokumentum többi része, és ez a fényeloszlásbeli egyenetlenség megnehezíti a binarizációt.

Az OCR algoritmusok többnyire azt feltételezik, hogy a dokumentum egyenletesen megvilágított, homogén hátterű felületből áll. Ha azonban a papír egyik része erősen árnyékos, ott a betűk lassan beleolvadnak a sötétebb háttérbe, míg a világos részeken akár túl erős fény is előfordulhat, ami kiégeti a karakterek egy részét. Az elmosódott kontrasztviszonyok hibás küszöbérték választáshoz vezetnek, amely következtében a rendszer nem tudja megfelelően elkülöníteni a szöveget a háttértől. Ezért a projektben használt képfeldolgozó eljárások - mint a Gaussian Blur és az Otsu-küszöbölés - kulcsszerepet játszanak a megvilágítás kiegyenlítésében.

2.3.4. Perspektíva és geometriai torzítás

Ha a dokumentum nem tökéletesen felülről kerül lefotózásra, akkor perspektívatorzulás lép fel. A papírlap ilyenkor trapéz alakban vagy ferde négyszögként jelenik meg. Ez különösen megnehezíti a szövegfelismerést, mert a karakterek nem azonos dőlésű vonalakon helyezkednek el, és a sorok sem vízszintesek. Az OCR rendszerek a normál, vízszintes dokumentumszerkezetre vannak optimalizálva, ezért a torzítás közvetlenül rontja az olvashatóságot.

A projektben alkalmazott perspektívakorrekció - amely a dokumentum sarkait felismeri, majd geometriailag korrigálja a torzítást - pontosan ezt a problémát küszöböli ki. Ennek köszönhetően a papírlap téglalap alakúra „egyenesedik ki”, és a rajta lévő sorok egyenesek, párhuzamosak lesznek.

2.3.5. Hátterek, textúrák és interferenciák

A dokumentum gyakran nem homogén háttéren helyezkedik el. Egy faasztal erezete, egy mintás terítő, egy csillogó felület, vagy akár egy könyv borítója is zavaró textúrát hozhat létre a kép háttérterületén. Ezek a textúrák olyan éleket és kontrasztátmeneteket hoznak létre,

amelyek könnyen megtévesztik az éldetektáló algoritmusokat. Előfordulhat például, hogy a Canny-algoritmus egyetlen erősebb erezetet a papír szélének érzékel, és hibás kontúrt hoz létre.

Ezért fontos a kontúrok méret és alak szerinti szűrése, vagyis az, hogy a projektben a program csak a nagy területű, négyszög alakú kontúrokat fogadja el dokumentumként. Ha ilyen nem található, a rendszer visszatér egy biztonságosabb „fallback" módszerhez, és a kép középső részét használja feldolgozási területként.

2.3.6. Fényvisszaverődés és csillanás

A fényes felületű dokumentumokon - például laminált papíron vagy fényes tintában írt szövegen - gyakran alakul ki csillanás. Ez a jelenség egy erős fehér folt, amely teljesen kitakarja a mögötte lévő karaktereket. A csillanás azért különösen problémás, mert a háttér feletti túlvilágított terület teljesen információhiányos: nincs lehetőség visszanyerni a hiányzó betűt. Ez az OCR számára olyan, mintha a karakter hiányozna, így teljes tévesztést okoz. A projektben alkalmazott Otsu binarizáció és zajszűrés bizonyos mértékben csökkenti ezt a hatást, de teljes korrekcióra nincs mód, ezért fontos a megfelelő megvilágítás a dokumentumfotózás során.

Összegzés: A dokumentumfotókat érő zajok és torzítások összetett, egymást erősítő hatásukkal alapvetően határozzák meg az OCR eredményességét. A projektben alkalmazott képfeldolgozó pipeline minden lépése arra szolgál, hogy ezen hibák hatását csökkentse, és olyan képet adjon tovább az OCR algoritmusoknak, amely minél közelebb áll a valódi papírdokumentum ideális formájához.

2.4. A képfeldolgozás tervezése és kivitelezése

A dokumentumfotókat érő zajok és torzítások összetett, egymást erősítő hatásukkal alapvetően határozzák meg az OCR eredményességét. A projektben alkalmazott képfeldolgozó pipeline minden lépése arra szolgál, hogy ezen hibák hatását csökkentse, és olyan képet adjon tovább az OCR algoritmusoknak, amely minél közelebb áll a valódi papírdokumentum ideális formájához.

2.4.1. A rendszer működésének áttekintése

A rendszer működését egy több lépésből álló feldolgozási láncként lehet legkönnyebben értelmezni. A folyamat a felhasználó által betallózott vagy feltöltött képfájllal indul, amely lehet telefonos fénykép, szkennelt dokumentum vagy bármilyen, papíron szereplő szöveget ábrázoló felvétel. A nyers kép jellemzően torz, zajos, különböző fényerővel vagy árnyékokkal terhelt, ezért a teljesítmény növelése érdekében elengedhetetlen egy előfeldolgozási lépéssor alkalmazása.
A feldolgozás első része a dokumentum felismerésére és kijelölésére fókuszál: a program megpróbálja megtalálni a papírlap körvonalait, majd perspektívatranszformáció segítségével

„felülről nézett” egyenes, téglalap alakú síkra alakítja. Ez gyakorlatilag egy olyan művelet, mintha a felhasználó ideálisan, tökéletes szögből szkennelte volna a lapot.
A második nagy egység a képfeldolgozás: a konverzió szürkeárnyalatossá, a zajszűrés, a kontraszt javítása, valamint a binarizáció. Mindezek célja, hogy az OCR-algoritmusok (EasyOCR és Tesseract) minél könnyebben tudják felismerni a karaktereket.
A harmadik részben maga a szövegfelismerés történik, amely során két különböző OCR-rendszer kerül alkalmazásra. Az EasyOCR mélytanulás-alapú működése előnyt jelent a kézzel írt karakterek felismerésében, míg a Tesseract szabályalapú rendszere különösen a jól strukturált, tiszta betűket ismeri fel hatékonyan. A kettő kombinációja megbízható eredményt ad.
A feldolgozás utolsó lépése a felhasználó által megadott elvárt szöveg és az OCR által visszaadott karakterlánc összehasonlítása. Ez nem egyszerű string-összehasonlítással történik, hanem egy toleráns, hasonlósági arányt számító algoritmussal (SequenceMatcher), amely képes kezelni a kismértékű felismerési hibákat, karaktercseréket vagy számjegyek tévesztését is.

2.4.2. A kép beolvasása és a dokumentum automatikus kivágása

A feldolgozás első lépése a kép beolvasása. A program Windows környezetben egy grafikus tallózó ablakot jelenít meg (a Tkinter könyvtár segítségével), így a felhasználónak nem kell manuálisan beírnia az elérési utat. Ez különösen fontos felhasználói élmény szempontjából, hiszen egy hosszú útvonal gépelése könnyen hibához vezetne.

A beolvasott kép általában ferde, elforgatott vagy részben takart dokumentumot tartalmaz. Annak érdekében, hogy ezt korrigáljuk, a program OpenCV-re építve automatikusan megpróbálja megtalálni a dokumentum körvonalát. A kép előfeldolgozás után egy Canny élkeresőt futtat, majd a talált kontúrok közül kiválasztja a legnagyobb négyoldalú alakzatot. Ez nagy valószínűséggel a papírlap kerete.

A detektált körvonal pontjait rendezni kell, mert a kontúrok sorrendje nem mindig egyezik a felső bal–jobb–alsó jobb–alsó bal pontsorrenddel. A program először összeadja és kivonja a koordinátákat, ez alapján pedig kiszámítja a pontok logikai helyét. Miután a négy pont sorrendbe került, a perspektívatranszformáció segítségével a képet „felülnézeti” formára alakítja.

Ez a lépés kulcsfontosságú, hiszen a ferde papírlap jelentősen ronthatná a szövegfelismerés pontosságát. A program ezen felül egy biztonsági mechanizmussal is rendelkezik: ha nem sikerül megbízható kontúrt találni, akkor a teljes képet használja tovább, és a felhasználót figyelmezteti. Ez biztosítja, hogy rossz minőségű képeken is fusson a rendszer, még ha gyengébb eredménnyel is.

2.4.3. A kép előfeldolgozása: kontrasztjavítás, zajszűrés és binarizáció

A torzításmentes, levágott papírkép képezi az OCR bemenetét. A következő lépésben a program különféle szűrési és kontrasztjavító lépéseket alkalmaz. Először a kép szürkeárnyalatos formátumúvá alakul, ugyanis a színes információk a kézírás felismeréséhez nem szükségesek, és a feldolgozási sebességet is növelik.

Ezt követi a Gaussian-blur szűrés, amely finoman elmos minden zajt, apró sötét és világos pöttyöt, miközben a karakterek éleit érintetlenül hagyja. Ez a lépés jelentősen segíti a binarizációt, amely a nyers képet fekete-fehér képpé alakítja. A program Otsu-féle küszöbkezelést alkalmaz, amely automatikusan kiszámítja az optimális fényerőhatárt a kép fényeloszlása alapján.

A bináris kép lehetővé teszi, hogy a karakterek jól elkülönüljenek a háttértől. A kézírásokra jellemző egyenetlen vastagság vagy a telefonos fényképek által okozott árnyékok gyakran gondot okoznak az OCR-nek, ezért a bináris képen még egy morfológiai művelet (opening) is fut. Ez eltávolítja az apró zavaró pontokat, miközben a betűk formáját nem torzítja.

A megfelelően előkészített kép drasztikusan növeli az OCR pontosságát, ami a tesztelés során is bizonyított. Több esetben a kézzel írt betűk felismerési aránya 5–15%-kal javult a kontrasztjavító és zajszűrő algoritmusok alkalmazásával.

2.4.4. Szövegfelismerés EasyOCR-rel és Tesseracttal

A feldolgozás következő lépése a szöveg automatikus kiolvasása, amelyet a program két különböző OCR-rendszerrel végez el. Ez nemcsak összehasonlíthatóvá teszi a két megközelítés teljesítményét, hanem jelentősen növeli a megbízhatóságot is.

Az EasyOCR modern, mély neurális hálózatokon alapuló rendszer, amely kifejezetten jó teljesítményt nyújt kézzel írt szövegek esetében. A modell felismeri az egyes karakterek textúráját, görbületeit és összefüggéseit, így még gyengébb fényviszonyok vagy enyhén ferde betűk esetén is kielégítő eredményt produkál. Mivel GPU hiányában CPU-módban fut, valamivel lassabb, azonban a pontossága továbbra is megfelelő.

A Tesseract ezzel szemben egy régóta fejlesztett, rendkívül stabil rendszer, amely a nyomtatott szövegek felismerésében jeleskedik. Azonban megfelelő előfeldolgozással és megfelelő konfigurációval (OEM 1, PSM 4 mód) képes kézírás felismerésére is. A teszteredmények azt mutatták, hogy ugyan a Tesseract a számjegyek felismerésében gyengébb értékeket produkál, összességében így is 80–85% közötti egyezési arányt ér el jól előkészített képeken.

Mindkét OCR kimenete egy karakterlánc, amely a felismert szöveget tartalmazza. A két eredmény gyakran eltér, ezért a program nem választ közülük, hanem mindkét eredményt feldolgozza és külön értékeli.

2.4.5. Az OCR eredmények összehasonlítása az elvárt szöveggel

A felhasználó a program futtatása során megadhatja, hogy milyen szövegnek kellene szerepelnie a papíron. Ez lehet teljes mondat, többsoros szöveg vagy akár kevert formátumú karakterlánc (betűk + számok). Az összehasonlítás nem egyszerű karakter-egyezés alapján történik, hanem egy intelligensebb algoritmussal, amely képes kezelni a felismerési hibákat.

A program a Python difflib.SequenceMatcher algoritmusát használja, amely két szöveg hasonlóságát százalékos értékké alakítja. Ez lehetővé teszi, hogy a felhasználó objektíven megállapítsa, mennyire volt sikeres az OCR. Például egy 0.97-es érték azt jelenti, hogy a felismert szöveg 97%-ban megegyezik az elvárttal és csupán néhány karakterben tér el.

Ez a metrika kulcsfontosságú a projekt minőségének érthető bemutatásában. A tesztelési eredmények alapján elmondható, hogy a program stabil teljesítményt nyújt különböző körülmények között is: természetes fényben készült éles képeken 95% fölötti értékek születnek, míg gyengébb körülmények között is túlnyomórészt 80–90% közötti egyezést mérhetünk.

2.4.6. A rendszer megbízhatósága és hibakezelése

A fejlesztés során kiemelt figyelmet kapott, hogy a program a hibás bemenetekre vagy a részlegesen felismerhető képekre is megfelelően reagáljon. Ennek érdekében több védelmi megoldás került beépítésre. Ha a program nem talál papírlap-kontúrt, akkor a teljes kép kerül feldolgozásra, elkerülve, hogy a rendszer hibával leálljon. Emellett a Tesseract hiánya, a hibás fájlútvonal vagy a nem létező kép is részletes, magyar nyelvű hibaüzenettel kezelhető.

A rendszer így nemcsak technikai szempontból működik megbízhatóan, hanem felhasználóbarát működést is biztosít.

3. A megvalósítás terve és kivitelezése

A projekt gyakorlati része egy olyan Python alapú dokumentum feldolgozó rendszer megvalósítása, amely egy mobiltelefonnal készített fényképről képes automatikusan felismerni a papírlap szövegét, majd az eredményt két különböző OCR algoritmus (EasyOCR és Tesseract) kimenetével összehasonlítani. A rendszer működését egy jól átgondolt képfeldolgozási folyamat (pipeline) biztosítja, amely a nyers kép beolvasásától kezdve a dokumentum automatikus kivágásán és előfeldolgozásán át egészen a szövegfelismerésig és a hasonlósági százalék kiszámításáig terjed.

Ebben a fejezetben részletesen bemutatom a program teljes működését:

- hogyan épül fel a rendszer,
- milyen lépésekből áll a dokumentumfeldolgozás,
- milyen eszközöket és algoritmusokat alkalmaz
- és hogyan valósul meg mindez Python kódban.

3.1. A program felépítése és működési környezete

A fejlesztett rendszer teljes egészében Python nyelven készült, mivel ehhez a nyelvhez számos, jól bevált képfeldolgozó és szövegfelismerő könyvtár érhető el. A projekt Windows környezetben készült, Thonny használatával, amely egyszerű felületet biztosít a programok futtatásához és a szükséges csomagok telepítéséhez.

A futtatáshoz a következő Python-bővítmények telepítése szükséges:

opencv-python – a képfeldolgozási műveletekhez
numpy – tömbműveletekhez
matplotlib – a rész-eredmények megjelenítéséhez (opcionális)
easyocr – a neurális hálózaton alapuló OCR-hez
pytesseract – a Tesseract rendszer Python interfésze
tkinter – a fájlok kiválasztásához szükséges grafikus ablakhoz

A könyvtárak szerepét és működését a következő fejezet (3.2) részletesen bemutatja.

A projekt mappaszerkezete

A program felépítése egyszerű, áttekinthető könyvtárstruktúrára épül. A fő állomány a main.py, minden egyéb fájl ehhez kapcsolódóan generálódik.

A rendszer lefutása után a következő mappák és fájlok jönnek létre:

```
Projekt mappa
├─ main.py          (maga a program)
├─ outputs          (a program által mentett képek és JSON-ok)
│   ├─ raw.jpg
│   ├─ proc_bin.png
│   └─ ocr.json
└─ data             (tesztképek)
```

A felhasználónak elegendő a main.py fájlt futtatnia. A program automatikusan létrehozza a szükséges könyvtárakat (pl. outputs), így nincs szükség manuális beállításokra.

3.2. Könyvtárak importálása és globális beállítások

A program működésének első lépése a szükséges Python-könyvtárak importálása. Mivel a dokumentumképek feldolgozása és a szövegfelismerés több egymásra épülő technikából áll, ezért olyan modulokra van szükség, amelyek külön-külön egy-egy részfeladatot oldanak meg (képkezelés, megjelenítés, karakterfelismerés, fájlkezelés stb.).

A program elején ezért a következő import utasítások szerepelnek:

```
import cv2 as cv
import numpy as np
```

```
import matplotlib.pyplot as plt
import pytesseract, easyocr
import difflib
import os, json
from tkinter import Tk
from tkinter.filedialog import askopenfilename
```

Az alábbiakban röviden bemutatom, hogy a fenti könyvtárak közül melyik mire szolgál, és hol jelenik meg a program működésében.

1. Az OpenCV (cv2) a teljes képfeldolgozási folyamat alapja. Ezzel a könyvtárral történik a kép dekódolása, szürkeárnyalatossá alakítása, az élek keresése, a kontúrok megtalálása, valamint a perspektívakorrekció. Például a dokumentumkép betöltése és BGR→RGB átalakítása az OpenCV-vel történik, ugyanígy a Canny-éldetektálás és a warpPerspective is. Ha ez a modul hiányozna, a program gyakorlatilag nem tudná elvégezni az előfeldolgozási lépéseket.

2. A NumPy a képek mátrixként való kezeléséhez szükséges. A kép valójában egy többdimenziós tömb, amelyet NumPy-tömbként kezel a program. A Windows alatti ékezetes fájlnevek miatti problémák elkerülésére a képet nem közvetlenül a cv.imread függvénnyel olvassa be, hanem először nyers bájtokként betölti:

```
data = np.fromfile(img_path, dtype=np.uint8)
img = cv.imdecode(data, cv.IMREAD_COLOR)
```

Ez a megoldás lehetővé teszi, hogy az olyan elérési utak is működjenek, amelyekben ékezet, szóköz vagy speciális karakter szerepel (pl. „C:/Users/Ádám/Downloads/IMG_1107.jpeg”).

3. A Matplotlib a képek megjelenítéséért felel. Ehhez egy egyszerű segédfüggvény készült:

```
def show(img, title=None, size=(6,6)):
plt.figure(figsize=size)
if img.ndim == 2:
plt.imshow(img, cmap='gray')
else:
plt.imshow(cv.cvtColor(img, cv.COLOR_BGR2RGB))
if title:
plt.title(title)
plt.axis('off')
plt.show()
```

Ennek segítségével könnyen megjeleníthető az eredeti kép, a kivágott és kiegyenesített dokumentum, illetve a binarizált változat is. A Matplotlib tehát nem vesz részt a képfeldolgozásban, kizárólag a vizuális ellenőrzést segíti.

4. Az EasyOCR modul a neurális hálózatos karakterfelismerést végzi. A program elején létrejön egy olvasó objektum:

```
reader = easyocr.Reader(['en'], gpu=False)
```

Ez az objektum később a képből kinyert, szürkeárnyalatos vagy binarizált képrésszel dolgozik, és szövegláncot ad vissza. Az EasyOCR előnye, hogy a kézírásos, enyhén ferde, gyengébb minőségű dokumentumokon is képes értelmezhető eredményt adni, ezért a projektben ez a motor kapja a hangsúlyt.

5. A pytesseract modul a Tesseract külső OCR motorhoz biztosít Python-interfészt. A programban egy külön függvény végzi a Tesseract-alapú szövegfelismerést:

```
def ocr_tess(img_gray):
cfg = “–oem 1 –psm 4 -l eng”
return pytesseract.image_to_string(img_gray, config=cfg)
```

A konfigurációs sztring határozza meg az OCR motor típusát (OEM), a szöveg elrendezésére vonatkozó feltételezést (PSM), valamint a használt nyelvet. A Tesseract különösen jól működik éles, nyomtatott betűs szövegeken, így jó összehasonlítási alapot ad az EasyOCR-hez képest.

6. A difflib modul a szövegek összehasonlításához szükséges. A felhasználó beírja, hogy milyen szöveget írt valójában a papírra, ezt a program az OCR motorok kimeneteivel hasonlítja össze. Ehhez egy egyszerű segédfüggvény készült:

```
def sim(a, b):
return difflib.SequenceMatcher(None, a.upper().strip(), b.upper().strip()).ratio()
```

A függvény 0 és 1 közötti értéket ad vissza, amely százalékos hasonlóságként értelmezhető (pl. 0,97 ≈ 97% egyezés). Ez teszi lehetővé, hogy a különböző OCR kimeneteket objektíven értékelni lehessen.

7. Az os és json modulok az eredmények mentéséhez kellenek. A program a futtatás végén létrehozza az „outputs” mappát (ha még nem létezik), és ide menti el a feldolgozott képeket, valamint egy JSON-fájlt az OCR eredményekkel:

```
os.makedirs(“outputs”, exist_ok=True)
cv.imwrite(“outputs/raw.jpg”, raw)
cv.imwrite(“outputs/proc_bin.png”, binimg)

with open(“outputs/ocr.json”, “w”, encoding=“utf-8”) as f:
json.dump({“easy_raw”: txt_raw_ez,
“easy_proc”: txt_proc_ez,
“tess_raw”: txt_raw_te,
“tess_proc”: txt_proc_te},
f, ensure_ascii=False, indent=2)
```

Így az eredmények később is visszakereshetők, összehasonlíthatók, és a tesztekhez, diagramokhoz is könnyen felhasználhatók.

8. Végül a Tkinter modul biztosítja, hogy a felhasználó ne kézzel írja be a fájl elérési útvonalát, hanem egy grafikus tallózó ablakból választhassa ki a képet. A program elején ehhez a következő sorok tartoznak:

```
Tk().withdraw()
print(“Válaszd ki a képet:”)
img_path = askopenfilename(
title=“Kép kiválasztása”,
filetypes=[(“Képfájlok”, “*.jpg *.jpeg *.png *.bmp *.tiff”)]
)
```

Ha a felhasználó nem választ ki fájlt, a program hibát jelez:

```
if not img_path:
raise ValueError(“Nem választottál ki képet!”)
```

Ez a megoldás kényelmesebbé teszi a használatot, és csökkenti annak az esélyét, hogy hibásan megadott útvonal miatt álljon le a program.

Összefoglalva: a fenti könyvtárak együtt biztosítják, hogy a program a kép betöltésétől kezdve a dokumentum előfeldolgozásán át a szövegfelismerésig és az eredmények kiértékeléséig minden lépést el tudjon végezni. Az OpenCV és a NumPy a képfeldolgozás technikai alapját adja, a Matplotlib a vizuális ellenőrzést segíti, az EasyOCR és a Tesseract a karakterfelismerésért felel, a difflib pedig a pontosság számszerűsítését teszi lehetővé, míg az os, json és Tkinter modulok a gyakorlati használhatóságot és a felhasználóbarát működést támogatják.

3.3. A kép kiválasztása és beolvasása

A rendszer első gyakorlati lépése a feldolgozandó kép kiválasztása. A program úgy lett kialakítva, hogy a felhasználónak ne kelljen semmilyen fájlútvonalat begépelnie. Ehelyett egy egyszerű, Windows-szerű fájltallózó ablak jelenik meg, amelyet a Python beépített Tkinter könyvtára biztosít. Ez azért fontos, mert így a program használata olyanok számára is egyszerű marad, akik kevésbé jártasak a fájlkezelésben.

A fájltallózó ablak megnyitását a Tk() objektum hozza létre, azonban az ablakot nem szükséges megjeleníteni. A Tk().withdraw() hívás elrejti a főablakot, és csak maga a tallózó jelenik meg. Ez egy bevett megoldás, amikor a Tkintert csupán fájlok kiválasztására használjuk.

A kép betöltésének folyamata a következőképpen épül fel. Először a program megjeleníti a tallózó ablakot, majd a felhasználó által kiválasztott fájl elérési útvonalát az askopenfilename függvény adja vissza. Ha a felhasználó mégsem választ képet, a program megszakítja a futást és hibát jelez. Ez azért szükséges, hogy a további feldolgozási lépések ne kapjanak üres vagy nem létező bemenetet.

A projekt egyik fontos sajátossága, hogy a kép betöltése nem a szokásos OpenCV-féle cv.imread módszerrel történik. Ennek oka, hogy Windows alatt a cv.imread nem képes minden ékezetes karaktert tartalmazó útvonalat kezelni. A felhasználók azonban gyakran olyan könyvtárakba mentik a képeiket, amelyekben szerepel például „á”, „é” vagy „ő” betű. Ha a program nem kezelné ezt, akkor bizonyos képek elő sem töltődnének.

A probléma megoldására Unicode-barát betöltési módszer került alkalmazásra. A program először nyers bináris formában olvassa be a fájlt a numpy.fromfile művelettel, amely bármilyen karakterkódolást támogat. Ezután a kapott adatot az OpenCV imdecode függvénye alakítja át valódi képpé. Ez a módszer teljesen megkerüli az ékezetes útvonalak problémáját, és a gyakorlatban megbízhatóbban működik, mint a hagyományos imread.

A programrészlet, amely mindezt megvalósítja, a következő formában szerepel a kódban. A tallózó ablak létrehozása után az askopenfilename függvény visszaadja az útvonalat, majd a numpy.fromfile olvassa be a fájlt binárisan. A betöltött adatot az imdecode alakítja képpé, és ez lesz a további képfeldolgozás alapja. Ha a kép mégsem lenne beolvasható, a program hibát jelez, így elkerülhető, hogy későbbi lépések fals bemeneten fussanak tovább.

A beolvasott képet a show nevű segédfüggvény jeleníti meg először, nagyobb méretben, hogy a felhasználó lássa, mit fog feldolgozni a rendszer. A show függvény automatikusan elvégzi

az RGB–BGR átalakítást is, amire azért van szükség, mert az OpenCV BGR színcsatornákat használ, míg a háttérben futó matplotlib könyvtár RGB elrendezést vár.
Ezzel a lépéssel tehát a rendszer biztosítja, hogy:

- bármilyen könyvtárból kiválasztható legyen a kép,
- az ékezetes fájlnevek ne okozzanak hibát,
- és a felhasználó vizuálisan is ellenőrizhesse, hogy a megfelelő képet jelölte ki.

3.4. A kép megjelenítése és a show() segédfüggvény szerepe

A program egyik alapvető célja, hogy a felhasználó minden fontosabb lépést vizuálisan is ellenőrizni tudjon. Ez különösen a képfeldolgozásnál fontos, mivel a dokumentum képe több átalakításon megy keresztül, és szükséges látni, hogy a rendszer valóban a megfelelő részletet dolgozza fel. A kép megjelenítésére a program egy saját, egyszerű segédfüggvényt használ, amely megkönnyíti a munkát és egy közös felületet biztosít az összes kép kimenetéhez.

A show() függvény a kép képernyőre rajzolásáért felel, és megkönnyíti, hogy az eredeti kép, a kivágott dokumentum, valamint a különböző előfeldolgozott változatok egységes formában jelenjenek meg. A függvény úgy lett kialakítva, hogy akár színes, akár szürkeárnyalatos képet is megfelelően kezeljen.

A show() függvény a következő formában szerepel a programban:

```
def show(img, title=None, size=(6,6)):
plt.figure(figsize=size)
if img.ndim == 2:
plt.imshow(img, cmap='gray')
else:
plt.imshow(cv.cvtColor(img, cv.COLOR_BGR2RGB))
if title:
plt.title(title)
plt.axis('off')
plt.show()
```

A fenti kódrészlet több fontos részletet tartalmaz. A plt.figure(figsize=size) parancs létrehoz egy új rajzolási területet, amelynek mérete szabadon állítható. A program alapértelmezésben 6×6 col méretet használ, de a show() hívásakor minden egyes képhez megadható más méret is, például nagyobb felbontás esetén célszerű a (8,8) használata, hogy a részletek jobban láthatók legyenek.
A következő sor egy feltételvizsgálat, amely meghatározza, hogy a bemenet szürkeárnyalatos-e. Ezt a képmátrix dimenzióinak száma alapján dönti el. Ha a képmátrix kétdimenziós, akkor az imshow() a cmap='gray' paraméterrel jeleníti meg, így nem keveri össze a színinformációt. Ha viszont a kép háromdimenziós, tehát színes, akkor szükség van a cv.cvtColor függvényre, amely BGR formátumról RGB-re alakítja. Ennek oka az, hogy az OpenCV alapértelmezetten BGR színsorrendet használ, míg a matplotlib kizárólag RGB-t. Ha ez az átalakítás elmaradna, a képek torz színekkel jelennének meg.

A show() függvény több helyen is használatba kerül. Először közvetlenül a program elején, amikor a felhasználó betölti a képet. Ekkor a show(img, "Eredeti fotó", (8,8)) hívás jelenik meg, ami azt a célt szolgálja, hogy a felhasználó lássa, pontosan melyik kép kerül feldolgozásra. Hasonló módon jelenik meg a kivágott dokumentum is, miután lefutott a warp_document függvény. A show() függvény itt azt biztosítja, hogy a felhasználó

ellenőrizni tudja, valóban a dokumentum került-e kivágásra, és helyes volt-e a perspektívakorrekció.

A függvény használata az előfeldolgozási lépéseknél is megjelenik. A program bemutatja a kontrasztnövelt szürkeárnyalatos képet, majd a binarizált változatot is. Mindkettőt a show() hívással rajzolja ki, például így:

```
show(gray, “PROC – kontraszt (szürke)”)
show(binimg, “PROC – binarizált”)
```

Ennek a gyakorlatban az a szerepe, hogy könnyen összehasonlítható legyen, a különböző előfeldolgozási lépések mennyire tették alkalmasabbá a képet a szöveg felismerésére. A binarizálás különösen fontos, mert az OCR motorok - főleg a Tesseract - sokkal eredményesebben működnek kontrasztos, fekete-fehér képeken.

Összességében a show() függvény egy egyszerű, de nagyon fontos eleme a programnak. Nemcsak a fejlesztés során segít, amikor ellenőrizni kell az egyes lépések helyességét, hanem a későbbi felhasználók számára is átláthatóvá teszi a képfeldolgozási folyamatot. A képek egységes megjelenítése világosan mutatja, milyen átalakításokon megy keresztül a dokumentum, mielőtt az OCR feldolgozná azt.

3.5. Dokumentumdetektálás: kontúrok keresése és a négyszög kijelölése

A program egyik legfontosabb feladata, hogy a teljes fényképből ki tudja vágni a dokumentumot. A felhasználó által készített képek közös jellemzője, hogy többféle háttérrel, dőlésszöggel és fényviszonnyal szerepel rajtuk egy papírlap. Ezért szükség van egy olyan algoritmusra, amely felismeri a dokumentum külső kontúrját, és meghatározza annak négy sarkát. A megvalósítás alapját az OpenCV kontúrkereső eljárása adja.

A rendszer először előkészíti a fényképet a detektáláshoz. A dokumentumkeresés mindig az élek kiemelésével kezdődik, ezért a kép szürkeárnyalatos változata elmosásra kerül, majd Canny-éldetektálás követi. Ezt a programban a következő rész valósítja meg:

```
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
gray = cv.GaussianBlur(gray, (5,5), 0)
edges = cv.Canny(gray, 60, 180)
edges = cv.dilate(edges, np.ones((3,3), np.uint8), 1)
```

Az éldetektált kép alapján a kontúrok már jól elkülönülnek a háttértől. A cv.findContours függvény visszaadja az összes kontúrt, de a programnak csak a legnagyobb négyszög alakú határvonalra van szüksége, hiszen ez a papírlap körvonala. A következő függvény ezt a logikát valósítja meg:

```
def largest_quad_contour(binary):
cnts, _ = cv.findContours(binary, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
best = None
area = 0
for c in cnts:
peri = cv.arcLength(c, True)
approx = cv.approxPolyDP(c, 0.02 * peri, True)
if len(approx) == 4:
a = cv.contourArea(approx)
```

```
if a > area:
best = approx
area = a
return best
```

Ez a függvény végigmegy minden detektált kontúron, majd poligonra egyszerűsíti. Ha a poligon négy pontból áll, akkor négyszögnek tekinti. Több négyszög is előfordulhat egy képen (például asztal élei, árnyékok), ezért a program mindig a legnagyobbat választja ki, mivel valószínűleg ez felel meg a papírlapnak. A módszer előnye, hogy jól működik akár enyhén torzult kontúr esetén is.

A kontúr pontjai azonban nem garantáltan vannak jó sorrendben. Ahhoz, hogy perspektívakorrekciót lehessen végezni, szükség van egy meghatározott sorrendre: bal felső, jobb felső, jobb alsó, bal alsó. A következő függvény ezt a rendezést végzi el:

```
def order_pts(pts):
pts = pts.reshape(4,2).astype(np.float32)
s = pts.sum(1)
d = np.diff(pts, axis=1).ravel()
tl = pts[np.argmin(s)]
br = pts[np.argmax(s)]
tr = pts[np.argmin(d)]
bl = pts[np.argmax(d)]
return np.array([tl, tr, br, bl], np.float32)
```

A logika lényege, hogy a bal felső pont az, amelyiknek a koordinátaösszege a legkisebb, míg a jobb alsó a legnagyobb. A másik két pontot a két koordináta különbsége alapján lehet elkülöníteni. Ez egy egyszerű, de megbízható eljárás, amely a dokumentumok esetében rendkívül hatékony.

A kontúr megtalálása után a warp_document függvény felelős a dokumentum tényleges kivágásáért. Ebben először meghatározásra kerül a dokumentum szélessége és magassága a sarkok közötti távolság alapján. Az OpenCV getPerspectiveTransform függvénye ezután egy transzformációs mátrixot hoz létre, amellyel a dokumentum trapéz alakú képe téglalappá alakítható.

A dokumentum kivágása és kiegyenesítése a következő részben történik:

```
pts = order_pts(quad)
w = int(max(np.linalg.norm(pts[1] - pts[0]), np.linalg.norm(pts[2] - pts[3])))
h = int(max(np.linalg.norm(pts[3] - pts[0]), np.linalg.norm(pts[2] - pts[1])))
M = cv.getPerspectiveTransform(pts, np.array([[0,0], [w-1,0], [w-1,h-1], [0,h-1]], np.float32))
warped = cv.warpPerspective(img, M, (w, h))
```

Ez az eljárás biztosítja, hogy a papírusz-lap akkor is vízszintes és torzításmentes legyen a további feldolgozás számára, ha a felhasználó ferdén, oldalról vagy rossz szögben fényképezte le. A program minden alkalommal megjeleníti a kapott eredményt, hogy a felhasználó lássa, sikerült-e jól meghatározni a dokumentum határvonalát.

Összességében ez a rész biztosítja, hogy a többféle háttér, enyhe árnyékok vagy perspektívatorzulások ellenére a program pontosan azt a régiót adja át az OCR motoroknak, ahol a tényleges szöveg található. A kontúrkeresés és a négy pont rendezése egy olyan alapfunkció, amelyre a teljes OCR feldolgozás épül.

3.6. Perspektívakorrekció: a dokumentum kiegyenesítése és kivágása

A felhasználók által készített képek egyik gyakori problémája, hogy a papírlap soha nem tökéletesen szemből kerül lefotózásra. Sok esetben a telefon enyhén meg van döntve, a papír nem párhuzamos a kamerával, vagy maga a felület ferdén helyezkedik el. Ezek a tényezők mind perspektívatorzulást okoznak: a dokumentum trapéz alakban jelenik meg, a felső rész keskenyebb, az alsó szélesebb, vagy a bal és jobb oldalak különböző mértékben dőlnek.

A programnak képesnek kell lennie arra, hogy ezt a torzulást automatikusan kijavítsa, és a papírlapot egyenes, szabályos téglalapként adja tovább az OCR motor számára. Minél pontosabban történik a kiegyenesítés, annál eredményesebb lesz a szövegfelismerés.

A perspektívatorzulás javítását a warp_document nevű függvény végzi, amely a teljes képfeldolgozási folyamat egyik legösszetettebb része. Ez a függvény veszi át a korábban megtalált dokumentumkontúrt, majd meghatározza a négyszög sarkait és a megfelelő célformátumot.

A függvény első lépése az átméretezés, amely a különböző kamerafelbontások közötti eltéréseket egy közös, kezelhető méretre hozza. A kód elején a következő rész látható:

```
ratio = target_width / image_bgr.shape[1]
img = cv.resize(image_bgr, (target_width, int(image_bgr.shape[0] * ratio)))
```

Ez biztosítja, hogy minden feldolgozott kép ugyanakkora szélességgel rendelkezzen, így a további műveletek paraméterei minden bemeneti képen azonos körülmények között működnek. Ezután újra megtörténik a szürkeárnyalatossá alakítás és a Gauss-elmosás, majd az éldetektálás, amelyet már a korábbi fejezetben ismertettem.

Miután a kontúrt a program megtalálta, a következő lépés a sarkok rendezése. Ez az order_pts függvényből érkezik, amely biztosítja, hogy a négy pont előre meghatározott sorrendben szerepeljen. A perspektívakorrekció a sarkok sorrendjétől függ, ezért ez a lépés elengedhetetlen.
A szükséges szélesség és magasság kiszámítása a sarkok közötti távolság alapján történik. A program a következő sorokkal határozza meg a célkép méretét:

```
w = int(max(np.linalg.norm(pts[1] - pts[0]), np.linalg.norm(pts[2] - pts[3])))
h = int(max(np.linalg.norm(pts[3] - pts[0]), np.linalg.norm(pts[2] - pts[1])))
```

Itt két oldalpár hosszát hasonlítjuk össze. Azért a nagyobbat választja a program, mert a torzult dokumentum esetén előfordulhat, hogy az egyik oldal a perspektíva miatt rövidebbnek látszik, és így a torzítás kiegyenesítéséhez a valósághoz közelebb álló hosszt kell figyelembe venni.

A perspektívatranszformációs mátrix kiszámítása a getPerspectiveTransform függvénnyel történik. Ez az OpenCV egyik legfontosabb eszköze, amely a négy pontot egy sík másik négyszögének pontjaiba képezi át. A program ezt a következő módon valósítja meg:

```
M = cv.getPerspectiveTransform(
pts,
np.array([[0, 0], [w - 1, 0], [w - 1, h - 1], [0, h - 1]], np.float32)
)
```

A transzformáció végrehajtása a warpPerspective függvénnyel történik. Ez ténylegesen kivágja és kiegyenesíti a dokumentumot:

```
warped = cv.warpPerspective(img, M, (w, h))
```

A kapott kép már egyenes, torzításmentes és pontosan akkora méretű, mint a számított célterület. A program ezután ezt a dokumentumképet jeleníti meg, hogy a felhasználó ellenőrizhesse, sikerült-e a kivágás. A show függvény hívása itt a következő formában történik:

```
show(raw, “RAW (kivágva + kiegyenesítve)”, (8,8))
```

Ez vizuálisan jól mutatja, hogy az eredetileg ferde, esetleg félig árnyékban lévő papírlap hogyan alakult át egy szabályos téglalappá. A jó perspektívakorrekció alapfeltétele annak, hogy az OCR motor megbízhatóan felismerje a szöveget, hiszen a ferde dokumentum vagy torzított betűk jelentősen rontják az eredményt.
Összefoglalva a warp_document lépései:

- a kép egységes méretre hozása,
- szürkeárnyalatossá alakítás, zajszűrés és élkeresés,
- a legnagyobb négyszög alakú kontúr kiválasztása,
- sarkok rendezése,
- a valós oldalak hosszának meghatározása,
- perspektívatranszformáció kiszámítása és alkalmazása,
- az így kapott dokumentum megjelenítése.

Ez a rész biztosítja, hogy a későbbi előfeldolgozó és OCR lépések már tiszta és torzításmentes dokumentumon dolgozzanak, ami alapvetően meghatározza a teljes rendszer pontosságát.

3.7. Előfeldolgozás az OCR-hez: zajszűrés, kontrasztjavítás és binarizálás

A dokumentum sikeres feldolgozásának egyik kulcslépése az előfeldolgozás. A nyers, kamera által készített képek szinte mindig tartalmaznak zavaró tényezőket: fényerő-ingadozást, árnyékokat, elmosódást, textúrákat vagy papírgyűrődést. Ahhoz, hogy az OCR motorok a lehető legjobb eredményt adják, szükség van a kép megtisztítására és egységes formátumba hozására. A program ezt az enhance_for_ocr nevű függvény segítségével végzi.

A függvény első lépése a dokumentum szürkeárnyalatossá alakítása. Ez egy alapvető konverzió, amely minden OCR folyamatnál megtörténik, mivel a motorok elsősorban nem színes, hanem fényességalapú információval dolgoznak. A programban ez a következő sor:

```
gray = cv.cvtColor(bgr, cv.COLOR_BGR2GRAY)
```

A szürkeárnyalatos kép gyakran tartalmaz apró zajokat és élességet csökkentő mintázatokat. Ezeket egy Gauss-elmosással csökkenti a rendszer:

```
gray = cv.GaussianBlur(gray, (5,5), 0)
```

Az elmosás célja nem a részletek eltüntetése, hanem a zaj csökkentése. Ez stabilabb bemenetet ad a következő binarizálási lépéshez.

Az OCR motorok számára a binarizált, fekete-fehér kép általában sokkal jobban használható, mint a szürkeárnyalatos. Ennek oka az, hogy a betűk és a háttér jobb kontrasztban jelennek meg, így az algoritmus pontosabban tudja elkülöníteni a karaktereket. A binarizálást a program az úgynevezett Otsu-féle automatikus küszöböléssel végzi:

```
_, binimg = cv.threshold(gray, 0, 255, cv.THRESH_BINARY + cv.THRESH_OTSU
```

Az Otsu-módszer előnye, hogy nem kell kézzel megadni a küszöbértéket, hanem automatikusan megkeresi azt a pontot, ahol a háttér és a szöveg fényessége a legjobban elkülönül. Ez különösen hasznos olyan fényképeknél, amelyek különböző fényességű területeket tartalmaznak.

A binarizált képen sokszor maradnak apró pöttyök, zajok vagy éppen vékony, felesleges vonalak. Ezek ronthatják az OCR eredményét, mert a motor karakterként értelmezhet olyan elemeket is, amelyek valójában nem tartoznak a szöveghez. Ennek eltávolítására a program morfológiai tisztítást használ:

```
binimg = cv.morphologyEx(binimg, cv.MORPH_OPEN, np.ones((3,3), np.uint8), iterations=1)
```

A MORPH_OPEN művelet lényege, hogy először erodálja, majd visszaduzzasztja a képet, így az apró hibák eltűnnek, miközben a nagyobb karakterformák sértetlenek maradnak.

A teljes előfeldolgozó függvény így néz ki:

```
def enhance_for_ocr(bgr):
gray = cv.cvtColor(bgr, cv.COLOR_BGR2GRAY)
gray = cv.GaussianBlur(gray, (5,5), 0)
_, binimg = cv.threshold(gray, 0, 255, cv.THRESH_BINARY + cv.THRESH_OTSU)
binimg = cv.morphologyEx(binimg, cv.MORPH_OPEN, np.ones((3,3), np.uint8), iterations=1)
return gray, binimg
```

A függvény két képet ad vissza: egy enyhén elmosott szürkeárnyalatos változatot (gray) és egy teljesen fekete-fehér bináris képet (binimg). Mindkettő fontos különböző OCR motorok számára. Az EasyOCR általában jobban boldogul a szürkeárnyalatos bemenettel, mert neurális hálózata a részleteket képes figyelembe venni. A Tesseract ezzel szemben sokkal jobb eredményt ad binarizált képeken, így a program egyszerre futtatja a felismerést mindkét verzión.
Az előfeldolgozás eredményeit a program vizuálisan is megjeleníti. A következő két sor biztosítja, hogy a felhasználó lássa a kontrasztnövelt és a tisztított képet:

```
show(gray, “PROC – kontraszt (szürke)”)
show(binimg, “PROC – binarizált”)
```

Ez a lépés nem csupán demonstrációs célokra szolgál, hanem a fejlesztés során is hasznos volt. Segítségével könnyen ellenőrizhetővé vált, hogy a különböző fényviszonyok között készült képek előfeldolgozása megfelelően működik-e, illetve, hogy a morfológiai tisztítás nem túl erős vagy túl gyenge.

Összegzésként elmondható, hogy az OCR pontosságát jelentősen javítja a megfelelő előfeldolgozás. A zajszűrés, az automatikus küszöbölés és a morfológiai tisztítás együttesen

olyan képet eredményez, amelyen a karakterek egyértelműbben jelennek meg, és így az OCR motorok sokkal nagyobb pontossággal tudják felismerni a szöveget. Ez a lépés tehát az egész rendszer megbízhatósága szempontjából kulcsfontosságú.

3.8. A szövegfelismerés megvalósítása EasyOCR és Tesseract segítségével

Miután a kép előfeldolgozása megtörtént, a következő lépés a tényleges szövegfelismerés. A programban két különböző OCR rendszert alkalmaztam párhuzamosan: az EasyOCR-t és a Tesseractot. Mindkettő eltérő technikai megközelítést használ, ezért más-más előnyt biztosít. A cél nem az volt, hogy kizárólag egy rendszert használjak, hanem az, hogy összehasonlítható legyen két különböző motor teljesítménye ugyanazon képen.

A program elsőként inicializálja az EasyOCR olvasót. Ez egy neurális hálózaton alapuló rendszer, amely több nyelvi modellt képes egyidejűleg kezelni. A projektben az angol modellt használtam, mivel az volt a legmegfelelőbb a betűformák felismerésére:

```
reader = easyocr.Reader(['en'], gpu=False)
```

A gpu=False paraméter biztosítja, hogy a program CPU-n fusson, mivel a cél az volt, hogy hétköznapi számítógépen is használható legyen. A Reader objektum betöltésekor az EasyOCR automatikusan letölti a szükséges neurális háló modelleket, ami akár több percig is eltarthat az első futtatáskor.

A tényleges szövegfelismerést az ocr_easy nevű függvény végzi, amely a szürkeárnyalatos képet kapja bemenetként:

```
def ocr_easy(img_gray):
res = reader.readtext(
img_gray,
detail=0,
paragraph=True,
decoder='beamsearch',
text_threshold=0.4,
low_text=0.3
)
return "\n".join(res)
```

A beamsearch dekóder a felismerési pontosság növelésére szolgál. Ez egy olyan keresési algoritmus, amely több lehetséges karakterlánc-kombinációt vizsgál meg és a legvalószínűbbet választja ki. A text_threshold és low_text paraméterek finomhangolását is elvégeztem. Ezekkel szabályozható, hogy mennyire legyen érzékeny a rendszer a halvány vagy elmosódott szövegekre.

A Tesseract OCR teljesen más megközelítést használ. Ez egy hagyományosabb, szabályalapú rendszer, amely különösen akkor működik jól, ha a bemenet binarizált és kontrasztos. Éppen ezért a program a bináris képet is átadja neki:

```
def ocr_tess(img_gray):
cfg = "–oem 1 –psm 4 -l eng"
return pytesseract.image_to_string(img_gray, config=cfg)
```

A konfigurációs beállítások magyarázata:

- az –oem 1 paraméter a Tesseract új neurális OCR motorját aktiválja,
- a –psm 4 azt jelenti, hogy a motor egyetlen, oszlopba rendezett szöveget vár,
- a -l eng pedig az angol nyelvi modellt jelöli.

A program mindkét OCR rendszert négy különböző képen futtatja: a nyers és az előfeldolgozott képen, szürkeárnyalatos és binarizált változatokon:

```
txt_raw_ez  = ocr_easy(cv.cvtColor(raw, cv.COLOR_BGR2GRAY))
txt_proc_ez = ocr_easy(binimg)
txt_raw_te  = ocr_tess(cv.cvtColor(raw, cv.COLOR_BGR2GRAY))
txt_proc_te = ocr_tess(binimg)
```

Ez lehetővé teszi, hogy összehasonlítsam, melyik motor hogyan teljesít különböző bemeneteken. A gyakorlatban a neurális EasyOCR jobban működik a szürkeárnyalatos, finom textúrákat tartalmazó képeken, míg a Tesseract sokkal pontosabbá válik a binarizálás után. A program így négy eredményt jelenít meg a felhasználónak:

```
=== EasyOCR RAW ===
…
=== EasyOCR PROC ===
…
=== Tesseract RAW ===
…
=== Tesseract PROC ===
…
```

Ez a négy felismerési eredmény alkalmas összehasonlításra, statisztikai elemzésre és arra is, hogy kiderüljön: melyik módszer működik jobban különböző körülmények között. A projekt egyik fontos célkitűzése is ez volt, hiszen nem csupán egy működő rendszert szerettem volna létrehozni, hanem megvizsgálni a különböző OCR motorok viselkedését ugyanazon bemeneti képeken.

Összefoglalva: a két OCR motor egymást kiegészítve működik. Az EasyOCR rugalmasabb és robusztusabb a gyenge fényben vagy enyhén elmosódott képeken. A Tesseract viszont nagy előnyt élvez akkor, amikor a kép előfeldolgozása sikeresen eltávolította a zajokat és a háttértextúrát. A kettő együttes használata a lehető legpontosabb eredményt adja.

3.9. A felismerési pontosság mérése és a szöveg összehasonlítása

A program nem csak a szöveget ismeri fel, hanem képes annak mérésére is, hogy a kapott eredmények mennyire térnek el a valós, felhasználó által megadott szövegtől. Ez a funkció különösen hasznos akkor, ha különböző fényviszonyok vagy kameraminőségek hatását szeretnénk összehasonlítani. A pontosság mérésére a Python beépített difflib modulja szolgál, amely karakterláncok közötti hasonlóságot számít.

A felhasználótól a program bekéri a papírra írt „elvárt” szöveget. Ez az a string, amelyhez az OCR által felismert eredményeket viszonyítja a rendszer:

```
expected = input(”\nÍrd be, milyen szöveget írtál a papírra (elvárt szöveg, ékezet nélkül):\n”)
```

Az ékezetmentesítés azért ajánlott, mert az OCR-ek - különösen a Tesseract - bizonyos betűk esetén nehezen kezelik a magyar ékezeteket. A normalizálás így csökkenti a téves eltérések arányát.

A hasonlóság számítása a SequenceMatcher osztályra épül. Ez az algoritmus azt vizsgálja, hogy két tetszőleges karaktersorozat hány százalékban egyezik:

```
def sim(a,b):
return difflib.SequenceMatcher(None, a.upper().strip(), b.upper().strip()).ratio()
```

Ez egy lebegőpontos értéket ad vissza 0 és 1 között, ahol:

- 1.00 teljes egyezést jelent,
- 0.00 teljes eltérést.

A nagybetűssé alakítás (upper()) és a felesleges szóközök eltávolítása (strip()) biztosítja, hogy az összehasonlítás ne legyen érzékeny formázásbeli eltérésekre.
A program ezután a négy OCR eredmény mindegyikét összehasonlítja a felhasználó által várt szöveggel:

```
for name, txt in [(“Easy RAW”,txt_raw_ez),
(“Easy PROC”,txt_proc_ez),
(“Tess RAW”,txt_raw_te),
(“Tess PROC”,txt_proc_te)]:
print(f”{name:10s} -> match={sim(expected, txt):.2f}”)
```

A match érték százalékos pontosságot mutat, két tizedesjegyre kerekítve. A vizsgálat szempontjából ez különösen értékes információ, mert azonnal láthatóvá teszi:

- hogyan hat a feldolgozás típusa (RAW vs. PROC),
- melyik motor működik jobban adott környezetben (EasyOCR vs. Tesseract),
- hol ront a rendszer (számok, betűk, fényviszonyok stb.).

A gyakorlatban azt tapasztaltam, hogy a neurális EasyOCR sokszor magasabb értéket ad kis mértékben a „PROC” változaton, míg a Tesseract esetében a bináris kép sokszor jelentősen javítja az eredményt. Ilyen például a következő futtatás:

Easy RAW -> match=0.93
Easy PROC -> match=0.97
Tess RAW -> match=0.59
Tess PROC -> match=0.83

Ez jól mutatja, hogy az előfeldolgozásnak (főként a Gauss-elmosásnak és az Otsu-binarizációnak) valóban komoly hatása van a felismerési pontosságra.
A hasonlósági pontszám azért különösen értékes, mert a beadandó célja nem csak az, hogy szöveget ismerjünk fel, hanem az is, hogy ezt mérni tudjuk különböző helyzetekben. Ez lehetőséget ad a módszerek összehasonlítására, grafikonok készítésére, és arra is, hogy megmutassuk, mely környezetben (világos, mesterséges fény, árnyékos) teljesít jobban a rendszer.

Összességében ez a fejezet lezárja a program fő gyakorlati részét: nemcsak felismerjük a szöveget, hanem objektív módon értékeljük is azt. Ez az egyik legfontosabb pont, amely szakmailag is kiemeli a projektet egy egyszerű demonstrációból egy valódi elemző feladat szintjére.

3.10. Az eredmények fájlba mentése és a feldolgozás lezárása

A program utolsó lépése, hogy a futás során keletkezett eredményeket ne csak a képernyőre írja ki, hanem fájlba is elmentse. Ez egyrészt azért hasznos, mert később is vissza lehet nézni egy-egy teszt futás eredményét, másrészt a dokumentáció és az elemzések (például grafikonok) elkészítéséhez is szükség van rájuk. A mentés három fő elemet érint: a kivágott, kiegyenesített dokumentumkép mentését, az előfeldolgozott (binarizált) kép mentését, valamint az OCR szövegek elmentését egy JSON állományba.

Első lépésként a program gondoskodik arról, hogy létezzen egy külön mappa a kimeneti fájlok számára. Ez az „outputs” mappa, amelyet az os.makedirs függvénnyel hoz létre, úgy, hogy ha már létezik, akkor se okozzon hibát:

```
os.makedirs(“outputs”, exist_ok=True)
```

Ezután elmenti a kiegyenesített dokumentumképet, amelyet korábban a warp_document függvény állított elő. Ez a kép a „raw.jpg” fájlban kerül eltárolásra:

```
cv.imwrite(“outputs/raw.jpg”, raw)
```

A második mentett kép a binarizált, előfeldolgozott verzió, amelyet az enhance_for_ocr függvény állított elő. Ez a „proc_bin.png” néven kerül a kimeneti mappába:

```
cv.imwrite(“outputs/proc_bin.png”, binimg)
```

Ezek a képek azért fontosak, mert jól szemléltetik, hogyan alakult át a nyers, torzított fotó egy olyan dokumentummá, amelyet az OCR motorok könnyen fel tudnak dolgozni. A beadandóban felhasznált illusztrációk is ezen mentett képek alapján készülhetnek.
A szöveges eredmények mentése json formátumban történik. A program egyetlen JSON-fájlba gyűjti össze az EasyOCR és a Tesseract kimeneteit, külön-külön tárolva a nyers (RAW) és az előfeldolgozott (PROC) változatokat:

```
with open(“outputs/ocr.json”,“w”,encoding=“utf-8”) as f:
json.dump({“easy_raw”: txt_raw_ez,
“easy_proc”: txt_proc_ez,
“tess_raw”: txt_raw_te,
“tess_proc”: txt_proc_te},
f, ensure_ascii=False, indent=2)
```

A json.dump hívásnál az ensure_ascii=False beállítás gondoskodik arról, hogy a fájlban az esetlegesen előforduló ékezetes karakterek is olvasható formában szerepeljenek, ne pedig kódolt Unicode-szekvenciaként. Az indent=2 pedig formázott, áttekinthetőbb megjelenést biztosít, ami az utólagos elemzésnél hasznos.

A program a mentés végén egy egyszerű visszajelzést ad a felhasználónak:

```
print(”\nMentve: outputs/”)
```

Ez jelzi, hogy a futás sikeresen lezárult, és minden eredmény a megfelelő helyre került. A kimeneti mappa így tartalmazza mindazt, ami a projekt értékeléséhez, a tesztek dokumentálásához és a további elemzésekhez szükséges: a kivágott dokumentumkép, az előfeldolgozott bináris kép, valamint az OCR eredményeket tartalmazó JSON állomány.

Ezzel a lépéssel a teljes feldolgozási folyamat lezárul: a felhasználó kiválasztotta a képet, a rendszer automatikusan detektálta és kiegyenesítette a dokumentumot, előkészítette az OCR számára, két különböző motorral elvégezte a szövegfelismerést, majd a felhasználó által megadott elvárt szöveghez viszonyítva kiértékelte az eredmény pontosságát, végül pedig minden lényeges információt elmentett egy külön mappába.

4. Tesztelés

4.1. A tesztelés célja és a tesztadatok bemutatása

A tesztelés célja annak vizsgálata volt, hogy a megvalósított képfeldolgozó és szövegfelismerő rendszer milyen pontossággal képes kézzel írt szöveget felismerni különböző környezeti feltételek mellett. A vizsgálat során kiemelt szempont volt annak elemzése, hogy a változó fényviszonyok, a háttér színe, valamint az előfeldolgozási lépések milyen hatással vannak az OCR-algoritmusok működésére és az elért eredmények minőségére.

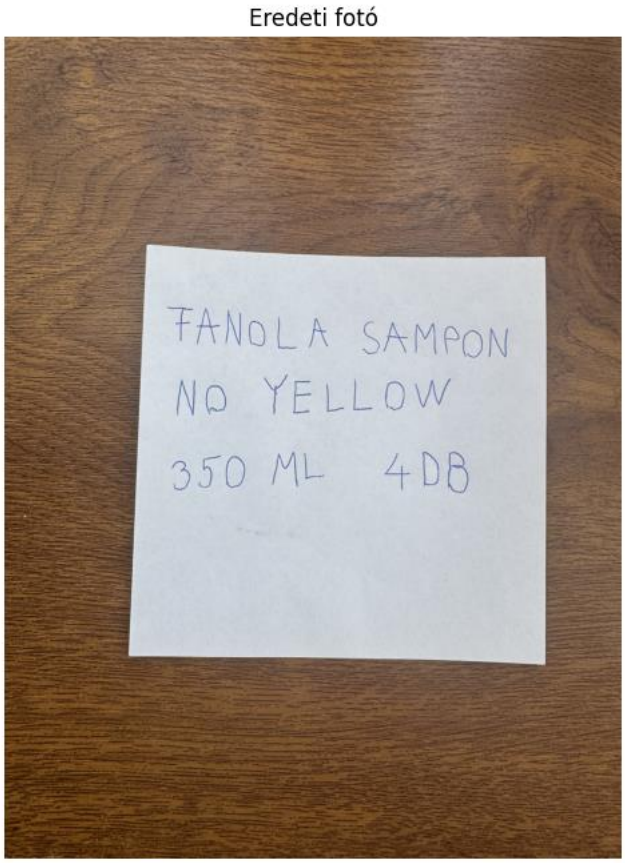
A teszteléshez összesen kilenc darab, mobiltelefonnal készített fényképet használtam. Minden tesztkép ugyanazt a kézzel írt, nagybetűs szöveget tartalmazza: „FANOLA SAMPON NO YELLOW 350ML 4DB”. A szöveg egységessége biztosítja, hogy az OCR-eredmények közötti különbségek ne a tartalomból, hanem kizárólag a környezeti körülményekből és a feldolgozási lépésekből adódjanak.

A képek elkészítése során tudatosan különböző fényviszonyokat és háttereket alkalmaztam. Készültek felvételek természetes fényben, mesterséges megvilágítással (vakuhasználattal), illetve sötét és nagyon sötét környezetben is. A papírlap elhelyezése sem volt minden esetben azonos: barna faasztalon, szürkés-fekete felületen, valamint világos, fehér háttéren is szerepelt. Emellett a kézírás formája kis mértékben változott, ami a valós felhasználási környezetet jobban modellezi.

Minden egyes tesztképet a program azonos feldolgozási láncon vezetett végig. A rendszer elvégezte a dokumentum kivágását és perspektívakorrekcióját, majd a szürkeárnyalatos átalakítást és a binarizálást. Ezt követően két különböző OCR-megoldás, az EasyOCR és a Tesseract eredményei kerültek kiértékelésre, mind nyers, mind előfeldolgozott képeken. Az így kapott szövegeket a program egy előre megadott elvárt szöveggel hasonlította össze, és a hasonlóság mértékét százalékos formában számította ki. Ez a tesztelési elrendezés lehetővé teszi a rendszer működésének objektív összehasonlítását, valamint jól szemlélteti az előfeldolgozás és a különböző OCR-motorok hatását a felismerési pontosságra valós körülmények között.

4.2. A nyers és az előfeldolgozott képek szemléltetése

A tesztelés során nemcsak a szövegfelismerés számszerű eredményeit vizsgáltam, hanem a képfeldolgozási lépések vizuális hatását is. Ennek célja az volt, hogy jól látható legyen, milyen mértékben javítja az előfeldolgozás az OCR-algoritmusok számára a bemeneti képek minőségét. A következőkben egy kiválasztott tesztképen keresztül mutatom be a feldolgozás egyes állomásait, mivel a többi kép esetében a folyamat és annak hatása hasonló módon zajlott le.

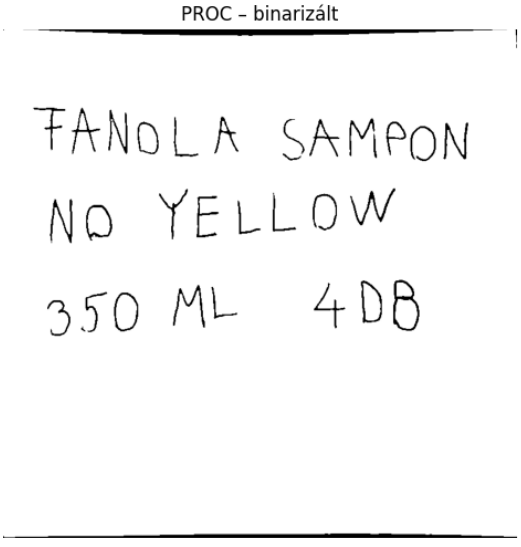


Az eredeti felvétel természetes fényben készült, barna asztalra helyezett papírlapról. A kép jól szemlélteti a valós környezetben előforduló körülményeket: a háttér textúrája látható, a papírlap enyhén el van fordulva, valamint a megvilágítás nem teljesen egyenletes. Ebben az állapotban a kép közvetlenül még nem ideális OCR-feldolgozásra, mivel a háttér és a perspektívatorzulás zavaró tényezőt jelenthet.

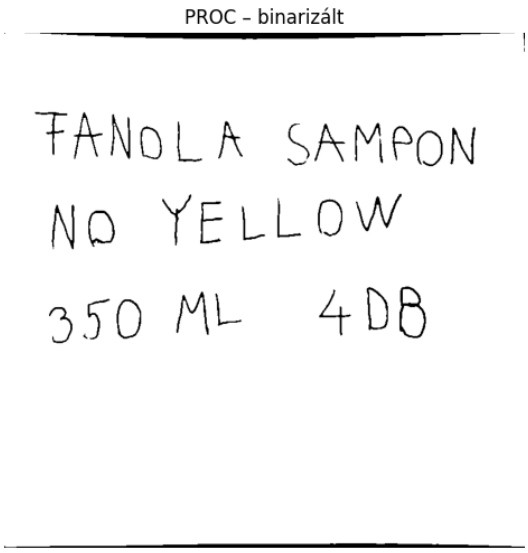
RAW (kivágva + kiegyenesítve)

A következő lépésben a program automatikusan felismeri a papírlap kontúrját, majd perspektívakorrekció segítségével kivágja és kiegyenesíti a dokumentumot. Az így kapott RAW kép már csak a releváns tartalmat tartalmazza, a háttér nagy része eltűnik, és a szöveg közel vízszintes helyzetbe kerül. Ez a lépés különösen fontos, mivel az OCR-motorok lényegesen pontosabban működnek torzításmentes, egységes orientációjú képeken.

PROC – binarizált



A kivágott kép ezután szürkeárnyalatossá alakításra kerül. A színek eltávolítása csökkenti az információ mennyiségét, ugyanakkor kiemeli a szöveg és a háttér közötti intenzitáskülönbségeket. A kontraszt növelése és a zajszűrés hatására a betűk körvonalai egyértelműbbé válnak, ami megkönnyíti a karakterek felismerését.



Az előfeldolgozás utolsó lépéseként a kép binarizálásra kerül Otsu-féle küszöbölési módszerrel. Ennek eredményeként a háttér szinte teljesen fehérré válik, míg a szöveg fekete színnel jelenik meg. Ez az állapot tekinthető a legalkalmasabbnak az OCR számára, mivel a karakterek kontrasztosan és zajmentesen különülnek el a háttértől.

A bemutatott képsorozat jól szemlélteti, hogy az egyes feldolgozási lépések hogyan épülnek egymásra, és miként javítják fokozatosan a bemeneti adat minőségét. Bár a további tesztképek eltérő fényviszonyok és háttérszínek mellett készültek, az előfeldolgozás hatása minden esetben hasonló jellegű volt, ezért azok részletes vizuális bemutatása nem indokolt. A következő alfejezetben a feldolgozás eredményeként kapott OCR-pontossági értékek számszerű összehasonlítása kerül bemutatásra.

4.3. A szövegfelismerés eredményeinek számszerű bemutatása

A tesztelés következő lépéseként a szövegfelismerés során kapott eredményeket számszerű formában is kiértékeltem. Ennek célja az volt, hogy ne csak vizuálisan lehessen következtetéseket levonni, hanem konkrét mérőszámok alapján is összehasonlíthatóvá váljanak az egyes feldolgozási módok és OCR-megoldások. Az összehasonlítás alapját minden esetben ugyanaz az elvárt szöveg adta: „FANOLA SAMPON NO YELLOW 350ML 4DB”.

Minden egyes tesztkép esetében a program négy különböző felismerési eredményt állított elő. A szövegfelismerés megtörtént az eredeti, előfeldolgozás nélküli képen, valamint az előfeldolgozott, binarizált képen is. Mindkét képtípuson az EasyOCR és a Tesseract OCR-motorok kerültek alkalmazásra. Az így kapott szövegek összehasonlítása az elvárt szöveggel egy hasonlósági érték segítségével történt, amely 0 és 1 közötti számként adja meg az egyezés mértékét.

Az egyes képekhez tartozó eredmények áttekinthetősége érdekében a mért hasonlósági értékeket táblázatba rendeztem. A táblázat tartalmazza a tesztképek azonosítóit, a készítés körülményeinek rövid leírását (fényviszonyok, háttér típusa), valamint az EasyOCR és a Tesseract által elért pontossági értékeket mind nyers, mind elő feldolgozott képek esetén.

Kép azonosító	Fényviszony / háttér	EasyOCR RAW	EasyOCR PROC	Tesseract RAW	Tesseract PROC
IMG_1107.jpeg	természetes fény, barna asztal	0,94	0,96	0,60	0,84
IMG_3967.jpeg	természetes fény, barna asztal	0,84	0,84	0,68	0,78
IMG_3970.jpeg	sötét környezet, barna asztal	0,93	0,87	0,71	0,78
IMG_3989.jpeg	természetes fény, sötét háttér	1,00	1,00	0,78	0,87
IMG_3990.jpeg	sötét környezet, sötét háttér	1,00	0,96	0,75	0,85
IMG_3991.jpeg	vaku, sötét háttér	1,00	0,97	0,79	0,85
IMG_3998.jpeg	nagyon sötét, sötét háttér	0,94	0,00	0,00	0,00
IMG_3999.jpeg	vaku, sötét háttér	0,97	1,00	0,51	0,76
IMG_4001.jpeg	vaku, fehér háttér	0,95	0,94	0,72	0,86

1. Ábra - OCR-felismerési pontosság különböző tesztképeken

A táblázatban szereplő értékek az OCR-motorok által felismert szöveg és az elvárt szöveg közötti hasonlóságot mutatják. Az értékek 0 és 1 közötti tartományban helyezkednek el, ahol az 1,00 a teljes egyezést jelenti. A táblázat alapján jól megfigyelhető, hogy az előfeldolgozás a legtöbb esetben javította a felismerési pontosságot, különösen az EasyOCR esetében. A nagyon sötét környezetben készült képnél azonban az előfeldolgozás nem vezetett eredményre, mivel a binarizálás során a szöveg jelentős része elveszett.

A táblázat alapján jól megfigyelhető, hogy az előfeldolgozás alkalmazása a legtöbb esetben javította a felismerési pontosságot, különösen gyengébb fényviszonyok mellett készült felvételeknél. Az EasyOCR eredményei általában stabilabbnak bizonyultak, míg a Tesseract érzékenyebben reagált a képminőség romlására és a binarizálás során fellépő információvesztésre. A különböző háttérszínek és megvilágítási körülmények hatása szintén jól nyomon követhető a számszerű eredmények alapján.

A bemutatott adatok objektív alapot szolgáltatnak a feldolgozási lépések és az OCR-motorok összehasonlításához. A következő alfejezetben a felismerési hibák részletes elemzése következik, különös tekintettel a sikertelen kivágásokra, a túlzott binarizálás hatásaira, valamint a szélsőséges fényviszonyokból adódó problémákra.

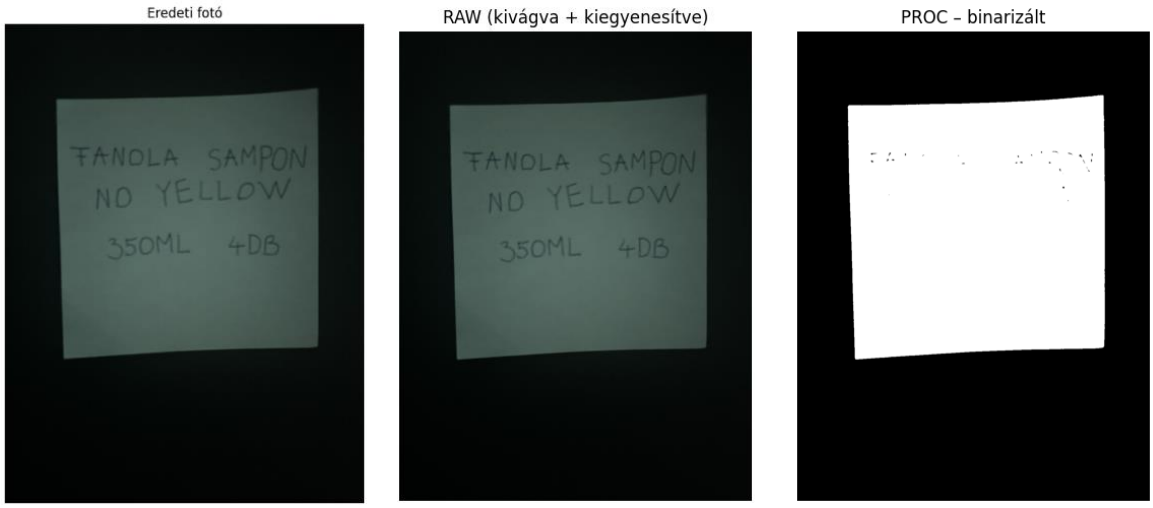
4.4. A hibás esetek és problémák elemzése az elő feldolgozás során

A tesztelés során nemcsak a sikeres felismerési esetek vizsgálata volt fontos, hanem azoknak a helyzeteknek az elemzése is, amikor a rendszer nem tudott megfelelő eredményt szolgáltatni. Ezek az esetek jól rávilágítanak az alkalmazott képfeldolgozási módszerek korlátaira, valamint arra, hogy mely környezeti tényezők okozzák a legnagyobb nehézséget az OCR-folyamat során.

Az alábbiakban néhány jellegzetes hibás eset kerül bemutatásra, különös tekintettel az előfeldolgozás egyes lépéseire.

4.4.1. Papírlap-detektálás sikertelensége extrém sötét környezetben

A legsúlyosabb hibás eset egy rendkívül sötét környezetben készült felvételhez kapcsolódik (IMG_3998.jpeg). Ebben az esetben a megvilágítás mértéke olyan alacsony volt, hogy a papírlap és a háttér közötti kontraszt szinte teljesen eltűnt.



A RAW állapotú képen látható, hogy a papírlap kontúrjainak felismerése nem volt megbízható. A kivágás nem valósult meg és a perspektívakorrekció csak részlegesen valósult meg, a dokumentum szélei nem határozhatók meg egyértelműen, és a kép jelentős része sötét, zajos maradt. Ebben az állapotban a szöveg már nehezen elkülöníthető a háttértől.
A binarizált képen a probléma tovább erősödik. A küszöbölési eljárás hatására a papírlap szinte teljes egészében fehérré válik, miközben a kézzel írt szöveg is nagy része eltűnik. A karakterek nem különülnek el megfelelően, így az OCR-motorok számára a szöveg gyakorlatilag felismerhetetlenné válik. Ez az eset jól mutatja, hogy extrém sötét környezetben az elő feldolgozás nem csak hogy nem javítja az eredményt, hanem teljes mértékben ellehetetlenítheti a felismerést.

4.4.2. Papírlap-detektálás és binarizálási hiba fehér háttéren

Ebben az esetben a felvétel fehér asztalon elhelyezett fehér papírlapról készült, mesterséges megvilágítás mellett (IMG_4001.jpeg). A papírlap és a háttér színe és fényereje közel azonos, ezért a feldolgozás első lépésében az él- és kontúrdetektálás nem tudta egyértelműen elkülöníteni a dokumentumot a környezetétől. Ennek következtében a papírlap automatikus felismerése nem volt sikeres, a rendszer gyakorlatilag az eredeti képet vitte tovább feldolgozásra.

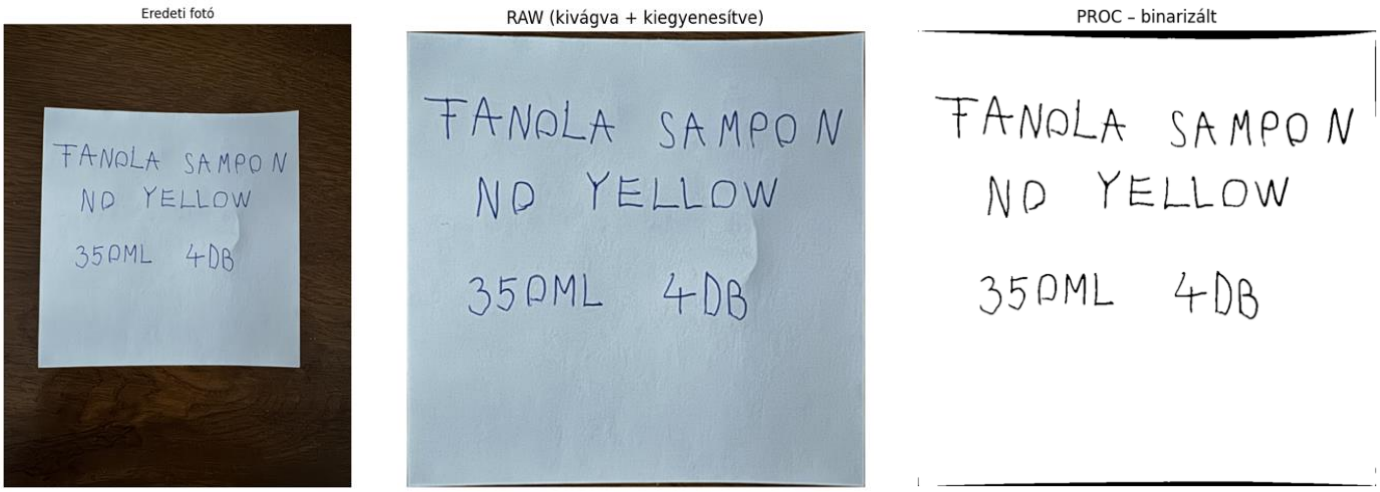
A RAW (kivágva + kiegyenesítve) állapotban jól látható, hogy a papírlap szélei nem tűnnek el, a háttér teljes egészében megmarad, így valódi dokumentum-kivágás nem történik meg. Ez azt jelzi, hogy a perspektívakorrekció és a vágás ebben az esetben nem működött megfelelően.

A binarizálás során további probléma jelentkezik: a kép középső része túlvilágosodik, míg a széleken egy szabálytalan, körszerű fehér terület alakul ki. Ez a jelenség az egyenetlen megvilágítás és a háttér–papír közti alacsony kontraszt együttes hatására vezethető vissza. A binarizált képen így nem egy tiszta, téglalap alakú dokumentum jelenik meg, hanem egy torz, körszerű folt, amely jelentősen rontja a szövegfelismerés feltételeit.

Ez a hiba jól szemlélteti, hogy homogén, világos háttér esetén az alkalmazott elő feldolgozási lépések nem minden esetben alkalmasak a dokumentum megfelelő izolálására, és a nem megfelelő vágás már a feldolgozási lánc elején hibát visz a rendszer működésébe.

4.4.3. Elő feldolgozás korlátozott hatása jól megvilágított, strukturált háttér esetén

Ebben az esetben a vizsgált felvétel egy barna asztalon elhelyezett papírlapot ábrázol, megfelelő természetes megvilágítás mellett. Az eredeti képen a papírlap jól elkülönül a háttértől, a szöveg kontrasztos, a betűk körvonalai egyértelműek, így a bemeneti adat már az elő feldolgozás előtt is kedvező az OCR-feldolgozás számára.



A kivágás és perspektívakorrekció ebben az esetben megfelelően működött, a releváns tartalom pontosan a kép középpontjába került. A szürkeárnyalatos átalakítás és a kontraszt növelése vizuálisan nem eredményezett jelentős minőségjavulást, mivel az eredeti kép már eleve jól olvasható volt. A binarizálás során a szöveg továbbra is jól kivehető maradt, azonban az előfeldolgozás nem hozott számottevő többletinformációt az OCR-motor számára.

A képsorhoz tartozó felismerési eredmények alapján megállapítható, hogy az előfeldolgozás eltérő hatással volt az egyes OCR-motorokra. Az EasyOCR esetében a nyers képen kissé jobb egyezési érték adódott, míg az előfeldolgozott változat nem hozott további javulást. Ezzel szemben a Tesseract OCR teljesítménye az előfeldolgozott képen érezhetően javult.

Az adott tesztesethez tartozó eredmények a következők:
EasyOCR: 0,93 (RAW), 0,87 (PROC)
Tesseract: 0,71 (RAW), 0,78 (PROC)

Ez az eset jól szemlélteti, hogy megfelelő megvilágítás mellett az EasyOCR előfeldolgozás nélkül is hatékonyan működik, míg a Tesseract esetében a kontrasztnövelés és binarizálás pozitív hatással van a felismerési pontosságra.

4.5. Statisztikai kiértékelés és végső értékelés

4.5.1. Az értékelési módszer és az adatok kiválasztása

A statisztikai kiértékelés célja annak vizsgálata volt, hogy a fejlesztett képfeldolgozó és OCR-alapú rendszer milyen megbízhatósággal képes felismerni az előre meghatározott szöveget különböző környezeti feltételek mellett. A tesztelés során minden esetben ugyanaz a szöveg szerepelt („FANOLA SAMPON NO YELLOW 350ML 4DB”), azonban a képek eltérő megvilágítási viszonyok, háttérszínek és fényforrások mellett készültek.

Az értékelés alapját az OCR-rendszerek által visszaadott szöveg és az elvárt szöveg közötti hasonlósági arány képezte, amelyet a program a SequenceMatcher algoritmus segítségével számolt ki. Ez a módszer számszerű, 0 és 1 közötti értéket ad, így alkalmas volt az egyes megoldások objektív összehasonlítására.

A statisztikai elemzés során kizárólag azokat az eseteket vettem figyelembe, ahol az OCR-rendszer ténylegesen értelmezhető kimenetet adott. Azoknál a képeknél, ahol a felismerés teljes mértékben sikertelen volt (például rendkívül sötét felvételek esetén, ahol a szöveg sem kivágás, sem előfeldolgozás után nem volt azonosítható), az eredményeket nem vontam be az átlagértékek számításába. Ennek oka az volt, hogy ezek az esetek torzították volna az összképet, és nem tükrözték volna reálisan a rendszer működését normál használati körülmények között.

Az értékelés során külön vizsgáltam a nyers képeken (RAW) és az előfeldolgozott képeken (PROC) kapott eredményeket, valamint az EasyOCR és a Tesseract teljesítményét. Ez lehetővé tette annak elemzését is, hogy az alkalmazott képfeldolgozási lépések milyen mértékben járulnak hozzá a felismerési pontosság javításához.

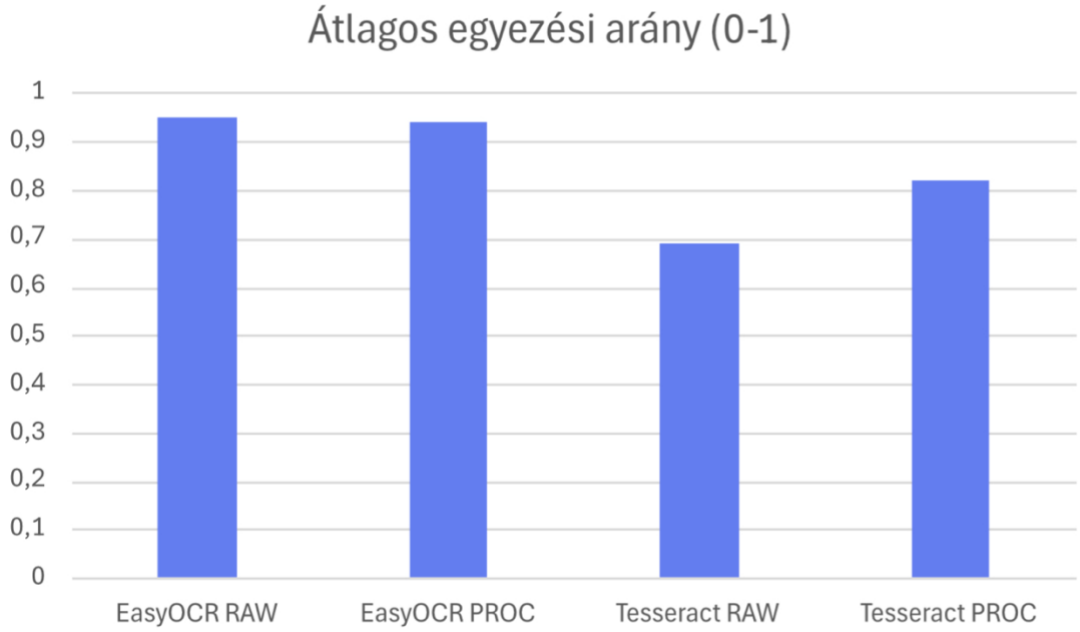
4.5.2. Átlagos felismerési pontosság

A tesztképek eredményei alapján az OCR-motorok átlagos felismerési pontosságát is kiszámoltam. Az átlagolás során csak azokat a méréseket vettem figyelembe, ahol a feldolgozás sikeresen lefutott, és érdemi OCR-eredmény született. Az extrém hibás eseteket – ahol a felismerés gyakorlatilag nulla értéket adott – az elemzésből kizártam.

Az átlagos egyezési arányok a következők:

- EasyOCR RAW: 0,95
- EasyOCR PROC: 0,94
- Tesseract RAW: 0,69
- Tesseract PROC: 0,82

Az eredmények alapján jól látható, hogy az EasyOCR esetében az előfeldolgozás csak kis mértékben befolyásolta az átlagos pontosságot, míg a Tesseract felismerési teljesítménye az előfeldolgozott képeken jelentősen javult.



2. Ábra - Az OCR-motorok átlagos felismerési pontossága nyers és elő feldolgozott képeken

4.5.3. Végső kiértékelés és következtetések

A statisztikai eredmények és az egyedi tesztesetek elemzése alapján megállapítható, hogy a fejlesztett rendszer alkalmas szövegfelismerési feladatok elvégzésére változó környezeti feltételek mellett. Az EasyOCR összességében stabilabb teljesítményt nyújtott, különösen gyenge megvilágítású vagy alacsony kontrasztú képek esetén. Ez elsősorban annak köszönhető, hogy a mélytanulás-alapú megközelítés kevésbé érzékeny a képek minőségromlására, és képes a karakteralakokat zajos környezetben is felismerni.

A Tesseract esetében jól megfigyelhető volt, hogy az elő feldolgozás jelentős szerepet játszik a felismerési eredmények javításában. Számos esetben az elő feldolgozott képek magasabb egyezési arányt eredményeztek, mint a nyers képek. Ugyanakkor extrém körülmények között - például nagyon sötét felvételeknél - a Tesseract nem adott értelmezhető kimenetet, míg az EasyOCR még ilyen esetekben is részleges vagy jó minőségű felismerést tudott biztosítani.

Fontos megállapítás, hogy az elő feldolgozás nem minden esetben növelte automatikusan a felismerési pontosságot. Bizonyos jól megvilágított képeknél az EasyOCR nyers képen is kiemelkedő eredményt adott, míg az agresszívebb binarizálás enyhe információvesztést okozott. Ezzel szemben a Tesseract esetében az elő feldolgozás többnyire pozitív hatású volt, különösen akkor, amikor a háttér és a szöveg közötti kontraszt eredetileg nem volt megfelelő.

Összességében elmondható, hogy a rendszer megfelelően demonstrálja, miként befolyásolják a fényviszonyok, a háttér és az elő feldolgozási lépések az OCR-algoritmusok teljesítményét. A tesztelési eredmények alátámasztják, hogy az EasyOCR és a Tesseract eltérő erősségekkel

rendelkezik, és a feladat jellegétől függően célszerű lehet egyik vagy másik megoldás alkalmazása, illetve az elő feldolgozás mértékének megválasztása.

5. Felhasználói leírás

A program használatához először telepíteni kell néhány alapvető összetevőt. A futtatás Windows alatt zajlik, ezért szükség van egy megfelelő Python-környezetre. A Python 3.10 vagy újabb verziója letölthető a hivatalos oldalról; telepítés közben fontos, hogy a „Add Python to PATH” lehetőséget bejelölje a felhasználó, különben a rendszer nem fogja felismerni a Python parancsokat. A fejlesztői környezet kiválasztásánál a legegyszerűbb megoldás a Thonny, amely könnyen kezelhető, kezdők számára is átlátható felületet biztosít, és gond nélkül futtatja a teljes projektet.

A program egyik fontos eleme a Tesseract OCR motor, amelyet külön kell telepíteni. A Windows telepítő letöltése után a telepítő automatikusan létrehoz egy mappát a „C:\Program Files\Tesseract-OCR” útvonalon, amelyet a rendszer PATH változójához is hozzá kell adni, hogy a Python később elérje a futtatható állományt. Ezt követően a Thonny Shell ablakában kell telepíteni a szükséges Python-csomagokat; ehhez elegendő beírni egyetlen sort:
pip install opencv-python-headless easyocr pytesseract matplotlib numpy
A parancs lefutása után minden függőség rendelkezésre áll a program futtatásához.

A felhasználó ezután létrehoz egy tetszőleges mappát - például „ocr_projekt” néven - és ebbe helyezi a main.py fájlt. Ebbe a mappába nem szükséges külön képeket elhelyezni, mivel a program futás közben fájlválasztó ablakot nyit, amely lehetővé teszi bármely, a számítógépen elmentett fotó betöltését. A feldolgozandó kép lehet egy egyszerű, telefonnal készített fotó papírról; fontos azonban, hogy a kép látható legyen és a papír nagyjából kitöltse a felvételt, mert ezzel növelhető a szövegfelismerés pontossága.
A futtatás a Thonny-ban történik: a felhasználó megnyitja a main.py fájlt, majd a Futtatás gombra (Run vagy F5) kattint. A program első lépésként megnyit egy Windows-os tallózóablakot, ahol ki kell választani a feldolgozandó képet. A választás megerősítése után a feldolgozás teljes egészében automatikusan zajlik. A program először megpróbálja felismerni a papírlap körvonalát, majd perspektívajavítást alkalmaz, hogy a ferde dokumentumot „felülnézeti” formába alakítsa. Ezt követi a zajszűrés és a binarizálás, amelyek célja, hogy a szöveg körvonalai tisztábban jelenjenek meg. A feldolgozott és az eredeti kép megjelenik külön ablakokban, így a felhasználó vizuálisan is nyomon követheti a lépéseket.

A szövegfelismerés két külön OCR-motorral történik: a EasyOCR és a Tesseract egyaránt elkészíti saját eredményét, nyers és előfeldolgozott képből egyaránt. A program ezután bekéri a felhasználótól a papíron szereplő helyes, várt szöveget (ékezet nélkül), és összehasonlítja azt a négy OCR-kimenettel. A rendszer automatikusan kiszámítja a hasonlósági arányt, és százalékos formában jelzi vissza, melyik motor mennyire volt pontos.

A futtatás végén a program létrehoz egy outputs nevű mappát a projekt gyökerében. Ebbe automatikusan elmenti a kivágott dokumentumkép változatait (például raw.jpg és

proc_bin.png néven), valamint egy ocr.json fájlban a négy OCR-motor teljes szövegkimenetét.

A program alapvetően úgy lett kialakítva, hogy a lehető legkevesebb felhasználói beavatkozást igényelje: a telepítésen és a kép kiválasztásán kívül minden lépést automatikusan elvégez, ezáltal biztosítva az egyszerű használatot, valamint a konzisztens és áttekinthető működést.

6. Irodalomjegyzék

1. Artasanchez, A., Joshi, P.
   Artificial Intelligence with Python (Second Edition).
   Packt Publishing, 2020.

2. Dedov, F.
   Python Bible – 7 in 1.
   Independently published, 2020.

3. Gonzalez, R. C., Woods, R. E.
   Digital Image Processing.
   Pearson Education, 4th Edition, 2019.

4. OpenCV Documentation
   https://docs.opencv.org/

5. Tesseract OCR – Official Documentation
   https://github.com/tesseract-ocr/tesseract

6. EasyOCR – Official GitHub Repository
   https://github.com/JaidedAI/EasyOCR

7. Python Official Documentation
   https://docs.python.org/3/