ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

SZAKDOLGOZAT

OE-NIK	Hallgató neve:	**Ruzsin Orsolya**
2021	Hallgató törzskönyvi száma:	**T/006357/FI12904/N**

Óbudai Egyetem
Neumann János Informatikai Kar
Kiberfizikai Rendszerek Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: **Ruzsin Orsolya**
Törzskönyvi száma: T/006357/FI12904/N
Neptun kódja: [redacted]

A dolgozat címe:

Kézzel írt szöveges dokumentumok digitalizálása UTF-8 formátumban
Digitalization of handwritten documents into UTF-8 format

Intézményi konzulens: Lovas István
Külső konzulens:

Beadási határidő: 2021. december 15.

A záróvizsga tárgyai: Számítógép architektúrák
IoT, beágyazott rendszerek és robotika specializáció

A feladat

Tervezzen és valósítson meg egy kézzel írt dokumentumok digitalizálását megvalósító rendszert. A cél, hogy a rendszer képes legyen egy papírra kézzel írt szöveget feldolgozni, és azt a képernyőn megjeleníteni valamilyen szövegmegjelenítő program segítségével. A folyamatot a rendszernek mesterséges intelligencia használatával kell megvalósítania. Tervezés során vizsgálja meg a lehetséges hibákat, megoldást nehezítő tényezőket.

A dolgozatnak tartalmaznia kell:

- a megoldandó feladat pontos leírását és részletes elemzését,
- meglévő rendszerek vizsgálatának elemzését,
- a rendszertervet, a döntések indoklásait,
- a megvalósítás leírását,
- a tesztelési tervet és a tesztelés eredményeit.

Ph.

.....
Dr. Fleiner Rita
intézetigazgató

A szakdolgozat elévülésének határideje: **2023. december 15.**
(OE TVSz 55.§ szerint)

A dolgozatot beadásra alkalmasnak tartom:

.....
külső konzulens

.....
intézményi konzulens

HALLGATÓI NYILATKOZAT

Alulírott hallgató kijelentem, hogy a szakdolgozat saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült szakdolgozatomban található eredményeket az egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja.

Budapest, 2021

Rózsa Orsolya

hallgató aláírása

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve: Neptun kód: Tagozat:

Ruzsin Orsolya ..... [redacted] ..... Nappali .....

Telefon: Levelezési cím (pl: lakcím):

[redacted] ..... [redacted] .....

Szakdolgozat / Diplomamunka címe magyarul:

Kézzel írt szöveges dokumentumok digitalizálása UTF-8 formátumban .....

Szakdolgozat / Diplomamunka címe angolul:

Digitalization of handwritten documents into UTF-8 format.....

Intézményi konzulens: Külső konzulens:

Lovas István..... .....

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2021.02.10	Témaválasztási konzultáció	
2.	2021.03.13	Szakirodalom áttekintése	
3.	2021.04.13	Lehetséges megvalósítás áttekintése	
4.	2021.05.04	Tartalmi, formai ellenőrzés	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplomamunka 2 / Diplomamunka 3 / Diplomamunka 4 tantárgy követelményét teljesítette, beszámolóra / védésre bocsátható.

Budapest, 2021. 05. 12.

.....
intézményi konzulens

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve: Neptun kód: Tagozat:

Ruzsin Orsolya ..... [redacted] ..... Nappali .....

Telefon: Levelezési cím (pl: lakcím):

[redacted] ..... [redacted] .....

Szakdolgozat / Diplomamunka címe magyarul:

Kézzel írt szöveges dokumentumok digitalizálása UTF-8 formátumban .....

Szakdolgozat / Diplomamunka címe angolul:

Digitalization of handwritten documents into UTF-8 format .....

Intézményi konzulens: Külső konzulens:

Lovas István ....

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2021.09.21	Specifikáció áttekintése	
2.	2021.10.14	Rendszerterv áttekintése	
3.	2021.11.18	Fejlesztés, tesztelés ellenőrzés	
4.	2021.12.08	Tartalmi, formai ellenőrzés	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplomamunka 2 / Diplomamunka 3 / Diplomamunka 4 tantárgy követelményét teljesítette, beszámolóra / védésre bocsátható.

Budapest, 2021.12.09

.....
Intézményi konzulens

TARTALOMJEGYZÉK

1 BEVEZETÉS

Szakdolgozatom célja egy olyan program létrehozása, amely egy kézzel írt szövegről készült képet képes feldolgozni, és annak tartalmát digitálisan megjeleníteni mesterséges intelligencia, betűfelismerés segítségével.

Az ötlet saját szokásaimból indult, miszerint én sokkal jobban szeretek kézzel írni, mint gépelni, viszont a gépelt szöveg sokkal könnyebben kezelhető, tanulásra alkalmasabb sokszínű felhasználhatósága és könnyű hozzáférése miatt. Rengeteg alternatíva van arra az esetre, hogy kézírásunkat már írás közben felismerje valamilyen program, de leírás után már a lapon lévő kézírás felismerésére nincsen a közéletben elterjedt szoftver. Mindig is nagy hasznát vettem volna egy olyan programnak, melynek segítségével, a kézzel írt jegyzeteimet digitalizálhatom.

A rendszer elkészítése közben az Irodalomkutatás fejezetben hasonló célú és működésű rendszereket mutatok be, azok hibát és előnyeit összevetve a saját rendszereméivel. Ebben a fejezetben foglalkozom a nyelvtani tényezőkkel, és azok hatásaival a rendszerre nézve, valamint a különböző OCR rendszerek eltérő működésével. Az egyik részfejezetben kitérek a mesterséges intelligenciák alapvető működésére is, és bemutatok néhány gyakran használt szerkezetet.

A Rendszertervezés fejezetben a program felépítését, és a részek működését mutatom be nagyvonalakban, majd a Fejlesztés cím alatt részletezem azokat.

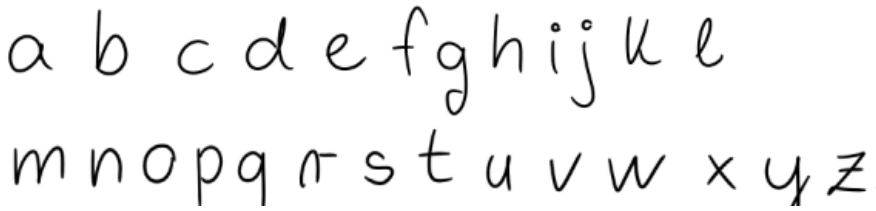
A bemenet egy kép, amin az arab abc betűivel írt szöveg olvasható. Ezen a számítógép művelteket végez, próbál összefüggéseket találni az adatbázisában tárolt adatok, és a beadott képen lévő, feldolgozott adatok között. A program kimente egy grafikus felület, ahol a felismert szöveg jelenik meg UTF-8 kódolásban.

2 IRODALOMKUTATÁS

2.1 Nyelvi kutatások

Projektemben a tervezési folyamat első lépése számomra a tanítandó abc meghatározása, és a lehetséges, betűkészlet specifikus problémák feltérképezése volt. Egyelőre nem szoftveri megvalósítás szempontjából, pusztán nyelvtani, grafológiai, és kalligráfiai szemszögből vizsgáltam az írás és az optikai karakterfelismerés kapcsolatát, valamint az ehhez kapcsolódó esetlegesen felmerülő hibákat.

„Az írás közlendőink rögzítése látható jelekkel. A helyesírás valamely nyelv írásának közmegállapodáson alapuló és közérdekből szabályozott eljárásmódja, illetőleg az ezt tükröző, rögzítő és irányító szabályrendszer.” [3] A latin abc 26 betűjét, öt magánhangzó, 21 mássalhangzó alkotja, és ebből kettő rendelkezik valamilyen írásjellel. Programom célja a magyar abc-vel írt szöveg digitalizálása lett volna, de adatbázis hiányában angol kézírást használó adatbázissal valósítottam meg terveimet. Jövőbeni fejlesztéseim része viszont, hogy magyar nyelvre is megvalósítsam programom, így a nyelvvel kapcsolatos kutatásokat elvégeztem, tanulságait az angol nyelv felismeréséhez hasznosítottam.

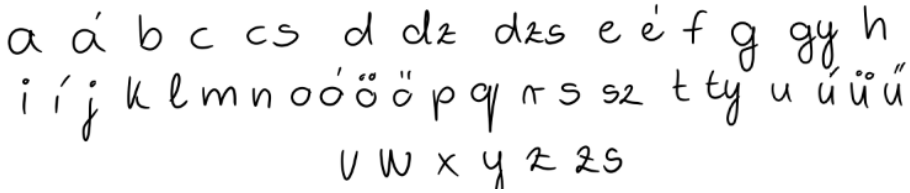


2.1. ábra A latin ABC betű, Forrás: saját kép

A magyar írás alapvetően a betűírások közé tartozik, tehát a legkisebb egységei nem szót vagy szótagot jelölnek, hanem hangokat jelölő betűket. [3] Eszerint az abc egyes betűit (betűkészlet) megfeleltetjük a hangkészlet egyes elemeivel. Ideális esetben a hang és a betűkészlet egyes elemei között kölcsönös, egyértelmű kapcsolat áll fent. [2]

„A szóalakokban általában ragaszkodunk a szóelemek feltüntetéséhez; az írásmóddal érzékeltetjük a tulajdonnevek különféle fajtáit; a különírás és az egybeírás révén megkülönböztetjük egymástól a szókapcsolatokat és az összetételeket; az összetett szavak elválasztásakor tekintettel vagyunk a szóhatárokra; stb. Ez hozzásegít a közölnivalók pontos kifejezéséhez és gyors felfogásához. Helyesírásunkat ezért nevezhetjük értelemtükrözőnek is.” [3]

A magyar abc a latint további 16 betűvel egészíti ki, 9 magán- és 7 mássalhangzóval. Az ékezetes betűk száma is nőtt, három új ékezeti fajta is megjelent. (kettőpont, két vonal, egy vonal) [2]

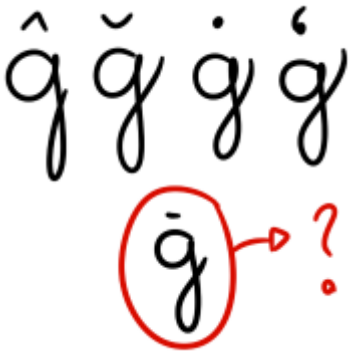


2.2. ábra A magyar abc betűi, Forrás: saját kép.

Az én dolgozatomban a betűk, és a belőlük alkotott szavak elkülönítése, felismerése a fő cél, nem a szavak értelmezése, így a továbbiakban a betűk írásmódjára fogok összpontosítani fonetikai jelentéstől függetlenül.

2.1.1 Ékezetek-Specifikáció

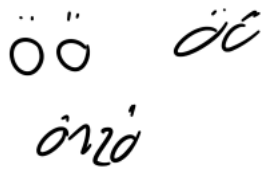
Mivel a kézírás minden ember számára bizonyos szinten egyedi, sőt, nemzetenként bizonyos betűk írása akár teljesen eltérő is lehet, a pontos eredmény elérésének érdekében fontos a nyelv meghatározása. [4] Egy olyan algoritmus előállítása, amely minden nyelv kézírását felismeri lehetséges, de rengeteg tanítási mintára lenne szüksége, és minden nyelvvel nőne a hibaszázaléka is. A többnyelvű OCR rendszer megvalósítására jó példa Premakumar [22] programja, ahol minden nyelvre azonos jellemkinyerő algoritmust és tanítómintát alkalmaztak, csak a szótár és a nyelvtani modell használatát határozták meg egy adott nyelvhez. A latin nyelvcsalád leszármazottainál maradva, már a keleteurópai nyelvek ékezetes betűin is felfedezhető a fent említett probléma.



2.3. ábra Keleteurópai nyelvek jellegzetes ékezetes betűi, alul kérdőjellel jelölt jel pedig "saját" írása az egyik fenti betűnek. Forrás: saját kép

Ha az algoritmust specifikusan az eszperantó nyelvre tanítottuk, akkor minden fajta nehézség nélkül fel fogja ismerni a fent piros körvonallal jelölt betűt (2.3. ábra) a „**ĝ**” (eszperantó „dzs”) betűként, míg ha mondjuk török nyelvre tanítottuk, akkor a „**ğ**” (török „gh”) betűt fogja megtalálni. [5] Viszont, ha minden keleteurópai nyelvet betáplálunk, a mesterséges intelligencia könnyen összezavarodhat a csúnya írás (2.4. ábra), és a sokfajta lehetséges eset kombinációjától, és hibás eredményt adhat a fenti kérdőjellel jelölt betűre.

Természetesen az ékezettel kapcsolatos probléma akkor is fennáll, ha egy adott nyelven belül gondolkozunk, ahol többféle ékezetet is használnak egy-egy betűn.

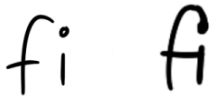


2.4. ábra Ö és ő betűk csúnya írás, Forrás: saját kép

Következtetésként levonható, hogy az AI egyik hibalehetősége az ékezetes betűk felismerése lesz. Az angol nyelvben nem sok ékezettel rendelkező betű található, és amik vannak, azok is egy féle ékezettel rendelkezhetnek összesen, így ott kisebb mértékben fog jelentkezni a hibás ékezetfelismerésből adódó hiba. A magyar nyelvre viszont erősen jellemző az ékezetes betűk használata, főképp a magánhangzók terén, így ott nagyban befolyásolhatja a felismerési pontosságot. A probléma egyértelmű megoldása a rengeteg, és még annál is több tanító minta beadása, de még ez sem biztosíthatja a megfelelő eredményt sokesetben.

2.1.2 Ligatúra - Folyóírás

A ligatúrákat (betűfűzéseket) a kézmozdulatokkal és hellyel takarékoskodó rovásírás hozta létre, és bár a nyomdászat első éveiben ezekből sokat átvettek, idővel számuk fokozatosan csökkent, míg végül napjainkban már csak a kész betűvé merevedett „ae”, „oe”, a német „ss”, és az „et” jel ligatúrái élnek hivatalosan. Az alkalmi kapcsolatban egybefűző betűfűzések közül a „fi” ligatúra a könyv formátumú igényes kiadványokban látható manapság. [7] Kéziratokban gyakran megfigyelhető a minimális, tudat alatti ligatúra. Gyakran egy ember sajátos kézírásával, vagy szimpla helyhiánnyal kapcsolatban figyelhető meg, csak nem tulajdonítunk neki nagy jelentőséget, hiszen a szó értelmezhetőségén és jelentésén nem változtat.



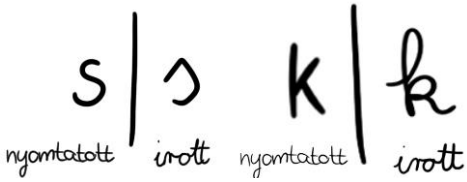
2.5. ábra fi külön- és összevont írással, Forrás: saját kép

A Word nem támogatja a ligatúrákat, csak szimbólum beszúrása paranccsal érhetőek olyan betűtípus esetén, amely ezeket tartalmazza. [7] Az írásból történő folyamatos kikopásuk miatt feltételezhetően a beolvasandó szöveg nem fog rendelkezni ligatúrákkal, legalábbis olyanokkal nem, amelyek különleges karakterek beszúrását igénylik. Viszont a ligatúrákkal kapcsolatos szoftveri megoldások implementálása a rendszer tovább fejlesztésében később felmerülhet. Amagyar abc nem rendelkezik betűvé merevedett ligatúrákkal, de köznyelvi „betűfűzés” viszont annál gyakoribb lehet, tekintve a kézírás gyakran eleve fűzött természetét. A megfelelő mennyiségű és minőségű tanítóminta bár segíthet az egybekötött betűk felismerésen, tökéletes megoldást nem nyújthat.

A folyóírás, folyóiratok digitalizációja ugyan ezt a problémát veti fel, illetve néhány betű írásával kapcsolatban is változtatásokat vezet be. A nyomtatott és folyóírással írt nagybetűk között nincs nagy különbség, így azt felesleges külön tárgyalni, viszont a kisbetűk között egyes helyeken lényeges különbségek mutatkoznak, melyekre a programot fel kell készíteni. A legtöbb helyen a különbség egy hurokban kimerül, vegyük például a nyomtatott kisbetűs „t” és a folyóírással írt „t” esetét. Ez az eset nem igényel különösebb foglalkozást, a folyóírással írt tanítómintákon kívül, hiszen a hurok ellenére is képes lesz megtalálni a program a „t”-re jellemző vonásokat mindkét esetben. De valahol a betű teljes formája és írásmódja is megváltozhat. [6] Ebben az esetben a programnak mindkét változatból megfelelő mennyiségű tanítómintát kell biztosítani, a kívánt eredmény elérésének érdekében.

2.1.3 Régi írásmódok

Programom fő célja a mai kézírással írt jegyzetek digitalizálása, viszont azt írást nem mindig ugyan úgy tanították az iskolában. Magyarországon 1906-1948 között úgynevezett „kalligrafikus szépírással” tanultak a gyerekek írni. [8] Nagyvonalakban a különbséget a betűk 50-68° szögbéli eldöntése, az árnyékolt vonalak és a díszes kezdővonalak adják a mai betűkhöz képest.

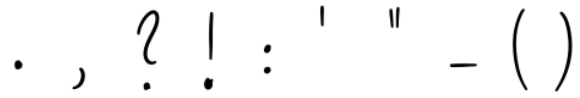


2.6. ábra Jobboldalt írott, baloldalt nyomtatott variációi az 's' és 'k' betűknek. Forrás: saját kép

A nagybetűket különösen cifrázták, mondhatni túldíszítették. Luttor Ignác kezdte kidolgozni a ma is használt írás alapjait, mely a körformára épült, és 90 fokos álló betűket kezdett használni, árnyékoltalanul. A máig használt „c”-kötéses egyszerűsített álló írás 1956-tól jelent meg Magyarországon. [8] A mesterséges intelligencia szempontjából megéri több korból (szigorúan racionális közelségből) mintákat venni, de figyelembe kell venni a kézírás folyamatos fejlődését, változását is.

2.1.4 Írásjelek és egyéb előforduló szövegbéli formázások

Egy szöveg pontos felismeréséhez elengedhetetlen az írásjelek felismerése, mivel azok a mondatok tagolásán kívül, azok értelmét is módosíthatják. „Az írásjelek szerepe kettős. Részben a mondatok szerkezetét, tagolódását, részeik-részleteik egymáshoz kapcsolódását tükrözik, részben némiképpen a beszédnek betűkkel ki nem fejezhető sajátságaira, a hanglejtésre és a beszédbéli szünetekre utalnak.” [3]



2.7. ábra Az írásgyakorlatban használt sok írásjel közül a legfontosabbak, Forrás: saját kép

Egy fogalmazásban, vagy folyóiratban leggyakrabban a fenti írásjelek fordulnak elő. A felismerésük viszont koránt sem olyan egyszerű, mint egy betű felismerése. A kérdőjel, felkiáltójel kevésbé, míg a pont és a további kisméretű írásjelek nagymértékben befolyásolhatják a program pontos működését. A következtetés levonása előtt fontos szót ejteni a matematikai képleteket tartalmazó írásokról is. Egy egyszerűbb elsőfokú egyenlettel még könnyen elbír egy mesterséges intelligenciát használó rendszer, csak pár egyéb írásjel hozzáadását igényli (+, -, *, /, =, stb.), melyeket az alap írásjelekkel együtt érdemes eleve betanítani.

Egy másodfokú egyenlet, képlet, esetleg lábjegyzet-hivatkozás megjelenítése viszont visszavezet a korábban, a pont és vessző esetében felvetett problémához. A felső- és alsó indexben lévő betűk felismerése önmagában akadályt jelent, hiszen a programnak fel kell tudni ismerni az indexeket, ehhez pedig tisztában kell lennie a sorok fogalmával, a betűk elhelyezkedésének jelentésével. Ugyan erre van szükség a pont és a többi írásjel pontos feldolgozásához is.

2.2 Technológiai kutatások

2.2.1 Múlt, Jelen és Jövő – OCR I.

Az OCR (optikai szövegfelismerés) technológia a képek, szkennelt, vagy fotózott dokumentumok szövegesen kereshető, szerkeszthető formába történő átalakítását hivatott biztosítani. Pontossága függhet a szöveget előfeldolgozó, és a szöveget a háttértől elválasztó algoritmus működésétől. Az első kereskedelmi forgalomba hozott rendszer a Reader's Digest (amerikai magazin) kiadójánál használt program volt, mely OCR technológiát használt, hogy az eladásokat felvigye a számítógép rendszerébe. [9] Azóta sokat fejlődött a technológia, viszont az OCR használata az óta is a nagy cégek automatizált folyamatinál a legelterjedtebb. Legjobb példa erre a postáknál használt önműködő címolvasó rendszer, [15] vagy a rendszámok felismerésére használt rendszerek. A technológia kezdeti idők óta elérhető a háztartások számára, viszont csak nagyon korlátozott formában.

Nagy György írt egy cikket [13] azon tézis köré fonva, miszerint az optikai szövegfelismerők folyamatos erőteljes fejlődése inkább a processzorok és a digitalizálás területén bekövetkezett robbanásszerű technológiai előre lepéseknek, mintsem a karakterfelismerő algoritmusok kikristályosodásának tudható be. A processzorok (és önmagában a számítástechnológia) robbanásszerű fejlődésgörbéje valóban jóval maga mögött hagyja a karakterfelismerő algoritmusokét, hiszen a legtöbb ma használt rendszer az évekkel ezelőtt kifejlesztett matematikai alapokon működik azóta is.

A kezdeti időkben a dokumentumok karaktereit egyesével, szekvenciálisan dolgozták fel, csak az elő. és utófeldolgozás közben foglalkoztak a nagy egésszel. Ennek paradigmája (character location-character isolation-character classification) pedig mai napig velünk van. [13]

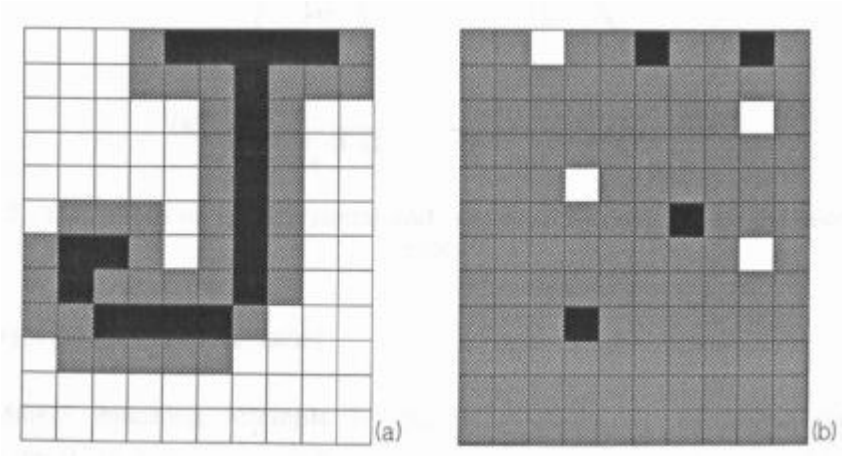
A korai időkben kiadott kutatások szinte mindegyike az egyes különálló betűk felismerésének pontosítására próbált egy jobb algoritmust találni (körvonalkövetés, alternatív megoldások vonalfelismerésre, stb.). Ezeket a kutatásokat gyakran specifikus adatokon tesztelték, melyeket újra és újra felhasználtak, így azok elvesztették értéküket. Az így betanított algoritmuson elkészülte után már nem lehetett változtatni, így az öröké ugyan azon hibákat ismételte. [13]

Magát a karakterfelismerés módszerét alapvetően két csoportba lehet sorolni. [15] Ezek a mátrix hozzárendelés, ahol a kép minden bitjét egyenesen a beadott mintákkal hasonlítjuk össze, valamint a jellem kiemelés, ahol pedig a feldolgozandó karakter valamilyen jellemzőjét kiemelve hasonítjuk össze azt a tanításként beadott mintákkal. Utóbbira példa a Tesseract, amely az egyes betűket azok poligonális jellemzői alapján ismeri fel, előbbire pedig a Wada [16] által fejlesztett rendszer.

2.2.2 Megvalósítások - Mátrix

A mátrix hozzárendelés alapja a betűk pixel szintű összehasonlítása. A rendszer minden betűt egy adott pixelszámú négyzetben tárol, melyet egy egyszerű küszöbérték számító módszerrel fekete-fehérré alakít. A karakterek egyértelmű részeit fekete pixelek jelölik, míg a hátteret fehérek. A szürke, későbbiekben fekete és fehér oldalra is sorolható pixelek a fehérek és feketék között helyezkednek el rugalmas elválasztóvonalként szolgálva. Mikor a rendszer egy ismeretlen karakterrel találkozik, minden pixelt fekete fehérré alakít(a szürkéket figyelmen kívül hagyja), és pixelenként összeveti az ismeretlen mintát a tároltakkal. [15] Természetesen a legnagyobb pixel-egyezési rátával rendelkező mintával fogja azonosítani a rendszer az ismeretlen karaktert.

Bizonyos pixelek hatásosabban segítenek megkülönböztetni egymástól karaktereket, mint mások. Ezen kulcs-pontok kiemelésével, és prioretizálásával jelentősen gyorsíthatunk a program futás idején. [15]



2.8. ábra Eredeti (a), és kiemelt pixeles (b) kép, Forrás: [15]

A módszer a legnagyobb hátránya működésének alapjában, a négyzetes feldolgozásban rejlik. Először is a mintaként használt, és a feldolgozandó betűknek is rögzített nagyságúnak kell lennie, hogy a rendszer kiemelkedő futás idejét fenntarthassa. Továbbá, a betűket tartalmazó cellák is azonos méretűek kell, hogy legyenek. [15]

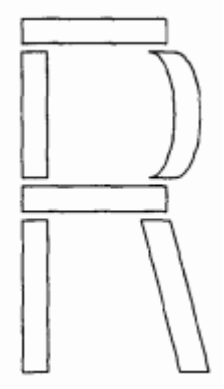
Összességében elmondható, hogy ez a megközelítés rögzített négyzetbe írt betűk, vagy jelek felismerésére kiemelkedően alkalmas. Ilyen lehet például a postai irányítószámok feldolgozása, vagy a szavazócédulák kiértékelése. Nem nyomtatott betűvel írt szövegek feldolgozására is ugyan úgy alkalmas lehet, pusztán megfelelően ki kell használni a módszer nyújtotta előnyöket.

A program egyszerűségének, és a futásidő lerövidítésének érdekében ezt a módszert választottam a szakdolgozatom elkészítéséhez.

2.2.3 Megvalósítások - Jellemkiemelés

Míg a mátrix összehasonlítás a lehető legkevesebb információ alapján próbálja felismerni az ismeretlen karaktert, a jellemkiemelés egy általános módszer, mely több, egymáshoz hasonló módszert foglal magában. Ezeknek célja az adott jellemzők kinyerése a fontfüggetlen felismerhetőség érdekében. [15]

Ilyen módszeren alapul például Grimsdale [17] OCR rendszere, mely a betűket alkotó vonalakat elemezi. Az egyes betűket az algoritmus felbontja görbe és egyenes vonalak csoportjaira, majd ezeket további ismeretekkel kiegészítve (vonalak hossza, csatlakozási pontok) egy 40 bites cellában tárolja. Ezek után a mátrix módszerhez hasonlóan az ismeretlen betű összehasonlításra kerül az eltárolt karakteradatokkal, majd a program visszaadja eredményként azt a betűt, ahol a legtöbb egyezést találta. [17]



2.9. ábra R betű részekre bontva, Forrás: [15]

A módszer egyik előnye a mátrix megoldáshoz képest, hogy az elforgatott betűket is könnyen felismeri, mivel nem rögzített pixeleket keres, hanem összefüggéseket, egész vonalakat. Ezzel elérhetővé válik a méret és pozíció független karakterfelismerés, bizonyos határok között pedig a font típusa is elhanyagolhatóvá válik.

Hátrányként egyértelműen a futásidőt lehet felhozni. Ez a megoldás sokkal összetettebb műveleteket igényel, ezáltal erősebb szoftveres háttérre is van szükség hozzá.

2.2.4 Múlt, Jelen és Jövő – OCR II.

Mostani szemlélet szerint az OCR rendszereket két részre lehet osztani. A különleges, külön az optikai felismerésmerésre épített gépek, és az otthoni számítógépre és szkennerre tervezettek csoportjára. Az elmúlt 20 évben főleg Európában és Japánban nagy lépéseket tettek meg az OCR fejlesztésének érdekében. A kutatók a gépírás helyett a kézírás digitalizálásával kezdtek foglalkozni inkább. 1980 óta a háztartásokban is elkezdtek megjelenni az OCR rendszerek, valamint egyre erőteljesebb, gyorsabb rendszereket dolgoztak ki ipari szinten is. S bár a technológia rengeteget fejlődött, még messze állunk egy tökéletesen működő optikai karakterfelismerő rendszertől, állítja Nagy György munkájában.[13]

A fő problémát a rosszul feldolgozott karakterek (rossz minőségű papír, rossz kézírás, stb.) jelentik. Nagyobb matematikai kifejezések, indexelt szövegek, betűk akár teljesen elkerülhetik a rendszer figyelmét. Nagy gondot jelent még az olvasási sorrend, bizonyos képek, kiemelt paragrafusok, táblázatok csúszhatnak el, és kerülhetnek teljesen más helyre. Még a legjobb rendszerek is függnek valamennyire a felhasználótól, hogy biztosítson információt a szöveg elhelyezkedéséről, vagy biztosítson megfelelő elválasztókat az OCR program számára.

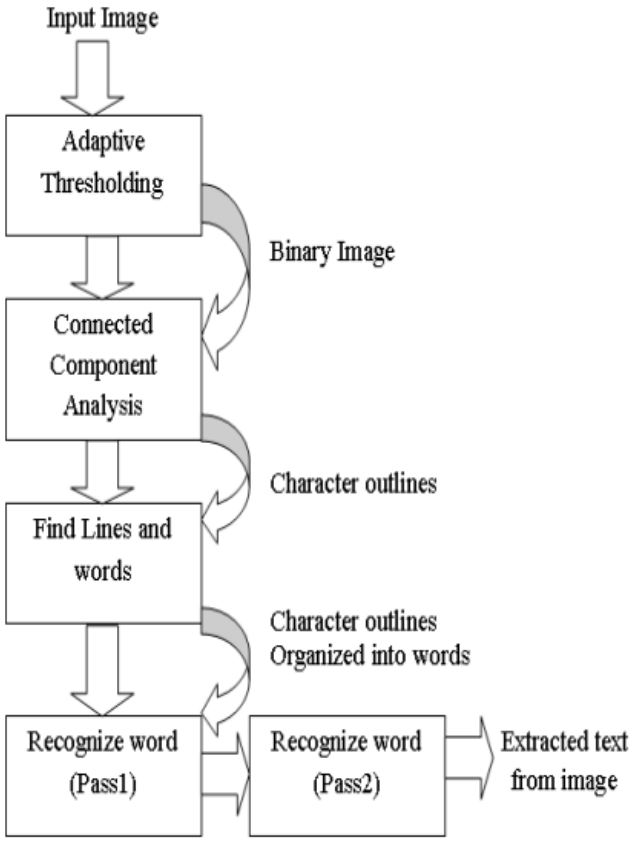
A mostani elvárások egy dokumentum megosztásánál két részre bomlanak, a dokumentum logikai, és térbeli elrendezésére. [13] Ehhez mérten két megközelítése van egy kép számítógép számára olvasható formátummá konvertálásának. Az egyik módszer a logikai felépítést semmibe véve csak a betűk felismeréséről alkotott tudásával dolgozik, a másik pedig valamilyen sémára épülő dokumentumok bizonyos részeinek kiolvasására épít. (például az OCR postai használata) Azok a módszerek, melyek csak a karakterfelismerésre támaszkodnak, legtöbbször az előfeldolgozó műveletekbe építik a dokumentum teljes egészére vonatkozó információkat. [13] Innen egyértelmű, hogy hiába veszünk egy kész programot, annak pontos képességeire nem tudunk referálni, hiszen csak bizonyos dokumentumokon volt futtatva. Holstege és Tokuda [18] munkájukban a betűtípus dokumentumértelmezésre gyakorolt hatásáról értekeznek. Bemutatják, hogy a szöveg részekre történő felbontásának segítségével fontos dokumentumrészeket emelhetünk ki, mint például a címek vagy alcímek, ezzel segítve egy komplexebb dokumentum feldolgozását. A betűszedési szabályok segítik a kisbetű nagybetű megkülönböztetését a hasonló alakú betűknél (c, C), valamint önmagából a hasonló betűk megkülönböztetését.

A publikált lapok szövegei legtöbbször kisebb tudásbázissal is felismerhetőek, ez részben az azokra vonatkozó küllemi és nyelvtani szabályoknak, részben pedig azok definiatív küllemének köszönhető. Nagyon sok specifikusan publikálásra szánt dokumentumokra írt rendszert mutattak be sikeresen. Ezek körül egy példa Dengel [19] hivatalos levelekre kifejlesztett rendszere.

2.2.5 Tesseract

A Tesseract egy nyílt forráskódú, ingyenesen elérhető optikai karakterfelismerő rendszer, melyet a HP (Hewlett-Packard Company) fejlesztett 1984 és 1994 között. [10] A C++ alapú program platform független, és akár DLL-ként implementálva más alkalmazásokban is könnyen használható. [10]

Napjainkban sok optikai szövegfelismerő szoftvert találhatunk az interneten. Ezek közül a két legelterjedtebb az Adobe Acrobat Pro DC, és az OmniPage Ultimate, a Google adatbázisa szerint legalábbis. A két program közül egyik sem ingyenes, de minkettő elég nagy pontossággal dolgozik. A Tesseract működési filozófia szempontjából a program a betűk pontos meghatározása helyett inkább arra törekszik, hogy a lehető legtöbb betűt ismerje fel. [10] Tekinthetünk erre úgyis, mint egy biztosított szerencsejátékra, hiszen ha a rendszer felismer valamilyen betűt, már a szelektáló algoritmuson múlik, hogy milyen adott betűre illeszkedő jellegzetességet talál meg benne. Míg ha eleve nem is ismer fel semmilyen betűt, annak esélye, hogy pontosan dolgozik, nullára csökken.

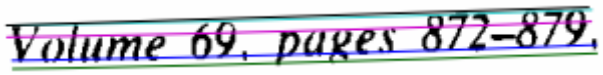


2.10. ábra Tesseract blokkdiagrammja, Forrás: [9]

Technikai szinten az első lépés a beadott kép bináris képpé történő alakítása, adaptív küszöbérték [14] segítségével. Ennek eljárása, hogy a program minden pixelt megvizsgál a képen, és ha az a pixel az adott küszöbérték felett van, akkor az előtérhez tartozik (fekete), egyéb esetben a háttérhez (fehér). Ezek után a program fő komponens analízis segítségével kiválasztja az egyes karakterek éleit. Ezen módszer kiemelkedő

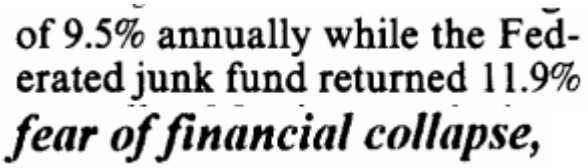
hasznosságónak bizonyult, mivel ezzel a módszerrel könnyen fel lehet dolgozni egy fekete hátterű, fehér betűs képet is. Smith, R az IEEE kilencedik nemzetközi konferenciáján [10] a Tesseract rendszeréről tartott előadásában kifejtette, hogy valószínűleg ez volt első ilyen metodikával megvalósított rendszer, amely könnyen tudta kezelni az inverz szövegeket is. [11]

A kiválasztott éleket ezután bináris adathalmazokká alakítja (Blobs), azokat pedig a szöveg vonalaiként dolgozza fel. A következő lépés a program működésében az alapvonal megtalálása. Ehhez a program az eddig összegyűjtött Blob-okat egy az elferdült kezdővonallal párhuzamosan haladó csoportokba rendezi, így kialakítva az új kezdővonalat, és az azzal párhuzamos szegmentáló vonalakat.



2.11. ábra Egy példa az elferdült kezdővonalra, Forrás: [10]

Ennek a módszernek a segítségével a Tesseract (elsőként az OCR rendszerek közül [10]) képes volt ferdített, nem egyenes vonalra írt szövegek olvasására is. A szöveget ezután betűkre bontja fel, rögzített távolság számításának segítségével. [10] Azon helyzetekre viszont, ahol nem egy bizonyos távolságra vannak egymástól a betűk, nincs bejáratott megoldás.



2.12. ábra Példa néhány nehezen elválasztható szóra, Forrás: [10]

Ezt a problémát a Tesseract a számított alapvonal, és a középvonal bizonyos korláton belüli távolságának elemzésével oldja meg, de a végső kiértékelést a szófelismerés stádiumára hagyja. [10] A program két lépésben tesz kísérletet a felismerésére. Első lépésben a rögzített távolsággal elválasztott, korábban szétbontott szavakat dolgozza fel a rendszer. [10] A második és egyben utolsó lépésben az esetleges hibákat javítja a program, majd kiírja a szöveget. [9]

A hibák javítását többek között a nehezen szétválasztható, esetlegesen egybeírt karakterek szétválasztására kifejlesztett algoritmussal kezdi meg a rendszer. Kiválasztja a legfelismertelenebb betűt, és megpróbálja blobokká alakítani. A vágási pontokat a sokszögek hasonlóságából adódó konkáv pontok csatlakozásai adják. A vágásokat prioritási sorrendbe rakja, majd megkezdi a betűk szétvágását. Azon vágás amely nem segíti a karakterfelismerést, visszavonásra kerül, de nem törlődik a rendszerből hogy

később a felhasználó kézzel újra alkothassa. A képen (2.13. ábra) remekül látszik, hogy az algoritmus megtalálta a r és az m csatlakozási pontját, és a tökéletes vágási pontot is.

2.13. ábra Vágási pontok, Forrás: [10]

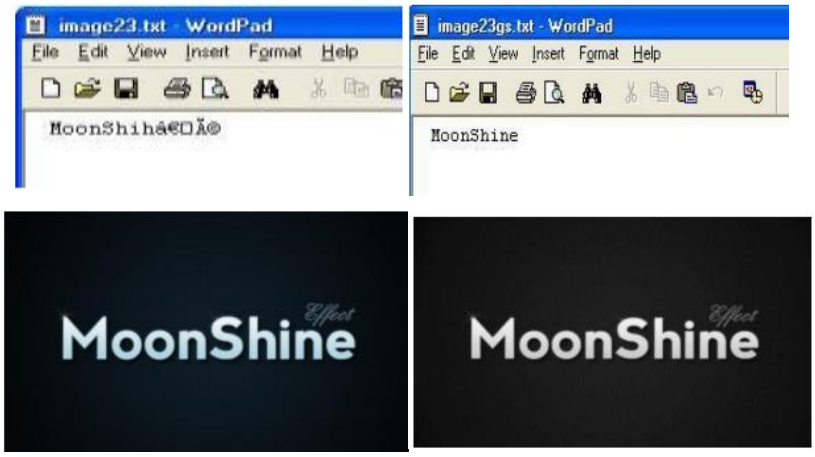
Ha a vágások után sem feldolgozható a szó a gép számára, akkor tovább küldi azt az asszociálónak. Az pedig egy első legjobb keresést futtat le az osztályozó gráfon, a szétvágott darabok minden lehetséges kombinációját megpróbálva. Útközben minden érintett gráf csúcsot elment egy hash táblába.

Maguk a betűk felismerése eleinte topológiai jellemzők alapján történt, később már a poligonok jellemzői alapján rendezett a rendszer. [10] Ezen a téren az áttörést azon felismerés jelentette, miszerint a felismerendő, és a tanításhoz használt adatok jellemzőinek nem kell feltétlen tökéletesen egyezniük. Ez a metodika a sérült, hibás karakterek felismerésében nyújt nagy segítséget, éppen ezért a Tesseract rendszerét sérült karaktereken nem tanították. [10]

Két paraméter szükséges a program lefutásához, a feldolgozandó kép valamilyen képformátumban, illetve a kimeneti állomány neve, ahol a szöveget szeretnénk viszontlátni. A kimeneti állományt a Tesseractban automatikusan „.txt” formátumban adja vissza, így nem kell azzal foglalkozni, hogy a formátumot a kimeneti fájl neve mögé írjuk. [10]

Chirag Patel és kutatótársai munkájukban [9] a Tesseract teljesítményét hasonlították össze más OCR rendszerekével. Az egyszerűbb képeket a Tesseract 100%-os pontossággal képes feldolgozni, míg bonyolultabb dokumentumok esetén nagyobb pontosságot mutat, ha a kép csak fekete-fehér. Ezt egy színes kép feldolgozásával szemléltették a tanulmányban [9], ahol is a színes háttérrel rendelkező szót a program

csak félig jól tudta feldolgozni, míg ha egy bizonyos eljárással a kép hátterét „kiszürkítették” akkor hiba nélkül képes volt kiolvasni a példaként beadott szót.



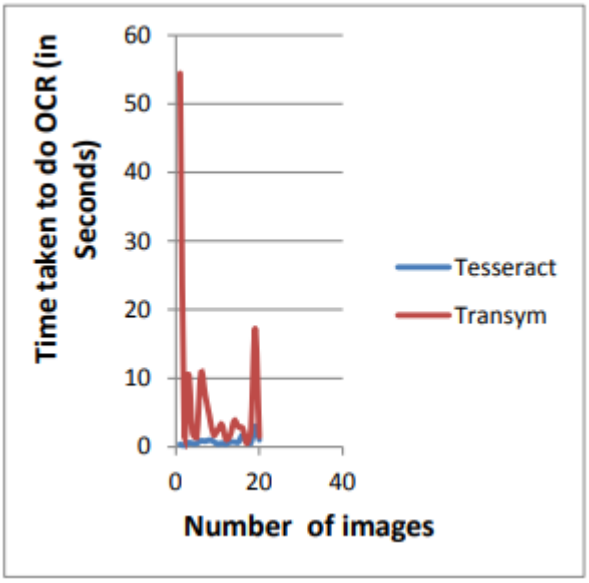
1 14. ábra Teszt-szó a tanulmányban, Forrás(ok): [10]

Chirag Patel és társai 20 különálló tesztet végeztek el, különféle autók rendszámtábláit olvastatták be az OCR eljárás segítségével. Az eredmény a fent említettekkel egybevág, a Tesseract 61%-os pontossággal dolgozott színes, és 70%-os pontossággal fekete-fehér képek esetén. Olyan színes képek esetén ahol az algoritmus közel 100%-os pontossággal dolgozott, a kép fekete-fehérré alakítása szinte alig javított az eredményen. Ahol viszont a program nem teljesített jobban 40%-os pontosságnál, a szürkeskálás eljárás vegyes eredményeket hozott, 16 és 100% között mozgott az egyes képek esetén a program pontossága. [9]

Ezek után a Tesseract eredményeit összevetették a Transym OCR nevű algoritmus eredményeivel. A Transym és a Tesseract közötti különbség elsősorban az, hogy a Transym az OCR eljárás előtt fekte-fehérré alakítja a megkapott képet, így azt nekünk külön nem kell megoldani. A kutatók ugyan azon a rendszámtábla csoporton tesztelték ezt az algoritmust is, mint a Tesseractot. Az eredményeket tekintve a Transym átlagos pontossága 47%-os volt, ezzel messze elmaradva a Tesseract 60 és 70%-os eredményétől. A kutatásban használt adatok alapján minden más szempontból is a Tesseract bizonyult hatékonyabbnak (futásidő, hibaszázalék, feldolgozási idő).

A grafikonon látható,(2.15. ábra) hogy volt olyan pont, ahol a Transymnek több mint 50 másodpercre volt szüksége az adott kép feldolgozásához, míg a Tesseract értékei végig 10 másodperc alatt maradtak.

Összefoglalásképp a kutatók egyértelműen a Tesseractot emelik ki jobbnak, az előző eredményekkel alátámasztva, viszont említést tesznek arról a tényről is, hogy ebben a kutatásban csak rendszámtáblákat alkalmaztak mintaként, így lehetséges, hogy más mintahalmazzal a Transym algoritmusa teljesítene jobban.



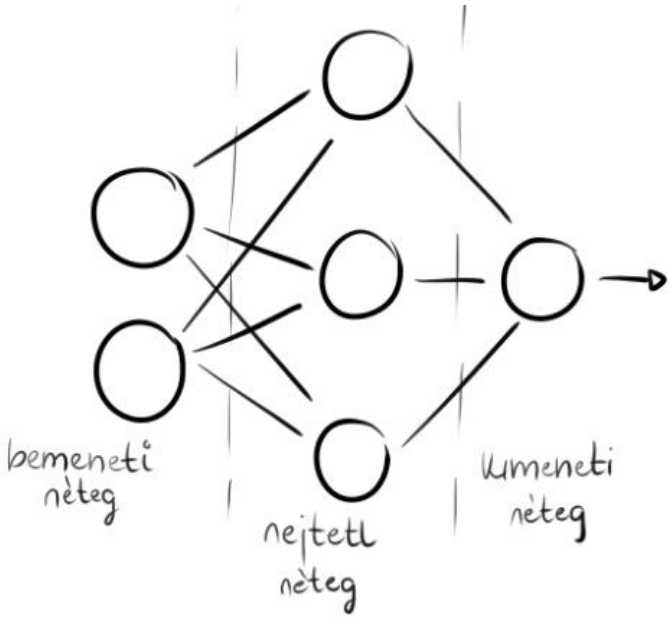
1 15. ábra Tanulmányban mért futásidő adatok grafikonja, Forrás: [11]

Összegzésképp elmondható, hogy a Tessreact egy nagy pontossággal működő, gondosan összerakott rendszer, melynek egyes részei, és megoldásai korában úttörőnek, a mai gondolkodás szerint pedig alapvető megoldásnak számítanak. A rendszer folyamatosan fejlődik, ahogy egyre több új, jobb, alternatív megoldást találnak a fejlesztők. A Tesseract legnagyobb gyengesége [10] a poligonjellemzők használata a sima körvonalak felismerése és feldolgozása helyett. Ebből adódóan Smith [10] munkájában egy jobb vágó algoritmusban, valamint a Rejtett-Markov-Model használatában látja a rendszer jövőjét.

2.2.6 Mesterséges Intelligencia

Az eddigiekben akármilyen jó és bonyolult modellt is alkottak is a tudósok, a neurális hálózat végül nem lett alkalmas az agy gondolkozásának rekonstruálására, csupán annak tanulási képességét sikerült valamennyire lemodellezni a gépi módszerek segítségével. Ebből adódóan a mai mesterséges intelligenciák egy kellően nagy adatbázison történő tanítás után képesek olyan adatokól is információt adni, amelyekkel korábban még nem találkoztak. [20]

A neurális hálózatot alkotó neuronok úgynevezett rétegekbe rendeződnek. Ezen rétegeket három csoportba soroljuk, a bemeneti, a kimeneti, és a rejtett rétegekébe. Míg bemeneti és kimeneti rétegből egy-egy lehet csak a hálózatban, a rejtett rétegek száma tetszőleges, az adott hálózat függvényében változik. A rétegeket a hálózatban súlyozott élek kötik össze. A neuronok a bementi élekről kapott bementi értékkel, és a súlyok segítségével bizonyos műveletet végeznek el (aktivációs függvény), majd az eredményt továbbítják a következő neuron felé.



2.14. ábra Egyszerű neurális hálózat, Forrás: saját kép

Az ábrán (2.16 ábra) egy olyan egyszerű neurális hálózat látható, ahol minden rétegből pontosan egy darab van.

A korábban említett tanítás során a hálózatba olyan bemeneti adatokat töltünk be, melyeknek van saját meghatározott kimeneti értékük. Minden adott bemenetre kérünk a hálózattól egy kimeneti értéket, majd összevetjük a hálózat által számolt kimeneti értéket a meglévő bemenethez rendelt értékkel. A két érték közötti eltérést hibának nevezzük, és ennek a hibának a csökkentése a célunk (a súlyértékek változtatásával), ha pontosítani szeretnék a hálózat predikcióin.

2.2.7 Mesterséges Intelligencia- Speciális hálózatok

A neurális hálózatokat sokfajta elrendezésben használhatjuk. Az alábbi fejezetben áttekintem az öt leggyakoribb elrendezést, külön kiemelve a programom számára legalkalmasabbat.

Percepton

Ez a „speciális” hálózat mindössze egyetlen neuronból áll. A bemeneti értékeket adott súlyértékek alapján összegzi, majd egy aktivációs függvénnyel elvégzett művelet után ad egy kimenetet. A perceptont lineáris elválasztási feladatoknál alkalmazzuk, tehát olyankor, amikor egy halmaz elemeit két részre szeretnénk bontani valamilyen tulajdonság alapján. [21]

Visszacsatolt neurális hálózat – RNN

A visszacsatolt neurális hálózat előnye az előre csatolthoz képest az, hogy rendelkezik memóriával, azaz képes a múltban történt eseményekre emlékezni. Az RNN esetében az információ egy hurkon halad át, így amikor a hálózat egy kimenetet ad meg, képes emlékezni a korábbi bemenetekre adott kimenetekre.

Ezen hálózat alkalmazása során két féle kérdés merülhet fel, az egyik a túlfutó, a másik az eltűnő gradiens probléma. Előbbinél az algoritmus túlzottan nagy jelentőséget tulajdonít a súlyoknak, és így nem képes összefüggést találni a bemenet változásával. Az utóbbi esetben viszont a gradiens értékünk túl alacsony, így a tanítási folyamat túl sok időt és erőforrást vesz igénybe.

A visszacsatolt neurális hálózatokat kiválóan lehet például képek automatikus feliratozására, vagy fordításra használni. [20]

Autoencoder

Az autoencoderek a neurális hálózatok egy speciális fajtái. A bemeneti adatot (legtöbbször képet) az enkóder tömörített formában eltárolja, a dekódoló rész pedig a tömörített verzió alapján megpróbálja rekonstruálni kimeneti értékként a bemenetet. Azonnal látható, hogy a módszer célja a leglényegesebb információk kinyerése, amiből a bementi adathoz hasonló kimenet előállítható. Napjainkban ezt a módszert például a képeken zaj csökkentésére, vagy teljes hiányzó részek kipótlására alkalmazzák. [20]

GAN

A GAN (Generatív Adversarial Network) hálózatok általában két, egymással szoros kapcsolatban lévő részegységből állnak. Az egyik egy előre csatolt neurális háló, melynek célja hogy adatokat generáljon, a másik pedig egy konvolúciós neurális hálózat, amely a generált adatokról dönti el, hogy azok hamisak-e vagy valódiak. A két hálózatot váltott optimalizációval tanítják be a megfelelő eredmény érdekében.

Ez a hálózat kiválóan alkalmas például arra, hogy egy kép alapján elkészítse az illető „kor képeit”, vagyis hogy hogyan néz majd ki 10 év múlva, és hogyan nézett ki 10 éve. [20]

Konvolúciós neurális hálózatok- CNN

Ezek a hálózatok elsősorban képfeldolgozási funkciókkal rendelkeznek, éppen ezért az OCR rendszeremet is ilyen struktúrával tervezem elkészíteni. Gyakori felhasználása ennek a hálózatnak, hogy a pixel szinten kinyert információt keresztülküldjük a neuronok rendszerén, majd az alapján osztályozni próbáljuk a képet.

A konvolúciós hálózat nem egészben dolgozza fel a képet, hanem annak részleteivel dolgozik. Két fő részre bonthatjuk, a jellemzők felderítésére, és az osztályozásra. A felderítés során a képet részekre bontjuk, és az egyes részeken átlagolásokat hajtunk végre, mely segítségével úgymond kiemeljük a képen lévő dolog legfőbb jellemzőit. A kiderített jellemzők alapján pedig a rendszer megpróbálja a korábban létrehozott és ismert osztályok valamelyikébe besorolni (felcímkézni) a képet.

Ezen hálózatok leggyakoribb alkalmazása az OCR területén fordul elő.

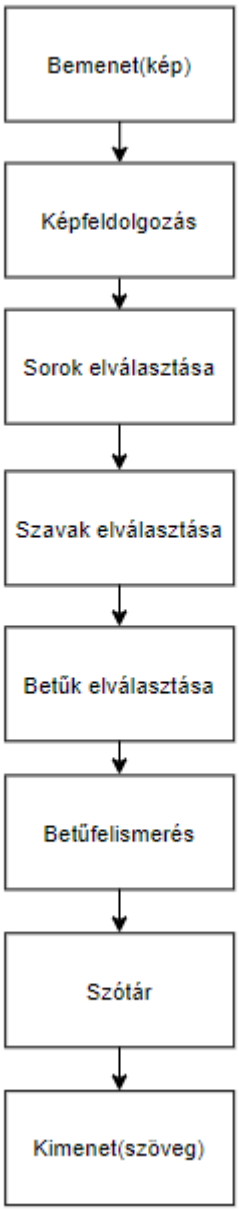
2.3 OCR korlátai

Az alapvető probléma a kézírás felismerésével kapcsolatban az AI számára az emberi léptékben vett „értelem” hiánya. Agyunk szövegértés közben nem az adott betűt próbálja feltétlenül megfejteni, hanem kitekint, és a nagy egészet nézi. Milyen szó lehet az? Abba a bizonyos szövegkörnyezetbe melyik szó illene bele? Milyen betűk vannak a szóban? Így jut el agyunk sokszor megoldáshoz, még ha egy-egy betű, vagy akár egy egész szó olvashatatlan is. Ezt a viselkedést csak egy összetett felépítésű, rengeteg tanító mintával rendelkező, folyamatosan tanuló mesterséges intelligencia tudná lemodellezni.

Tehát a „tökéletes” OCR algoritmusnak a betűk felismerésén kívül még egy fontos programrésszel kell rendelkeznie, a szövegértéssel. Mindegy mennyi mintát mutatunk a gépnek, bizonyos írásképpel írt szövegből kiemelt betűk önmagukban felismerhetetlenek. Ilyenkor kezdődne „Mi az értelme?” gondolatmenet, és a programot is ezen az úton kéne elindítani. Erre a leghatásosabb megoldás egy nyelvfelismerő és szövegértelmező algoritmus lenne, amely a jól és rosszul felismert betűkből álló szöveget teljes egészében próbálja megérteni, és az esetlegesen rosszul írt szavakat pedig helyesekre cserélni. Egy ilyen összetettségű mesterséges intelligencia előállításánál a három legnagyobb probléma az erőforrásigény mértéke, a nyelvi diverzitás, és a futásidő hosszúsága lenne. Míg futásidő és az erőforrás igény két magától értetődő probléma, melyek az évek alatt bekövetkező technológiai fejlesztésekkel kiküszöbölhetőek, a nyelvi diverzitás egy örökké fennálló probléma marad. Ha a tökéletes kézírás felismerő rendszerre úgy hivatkozunk, mint egy program, ami képes minden kézírást felismerni és digitalizálni, akkor nem elég, hogy minden ember (nagyrészt) egyedi kézírásának felismerésére kell megoldást nyújtania, de a különböző nyelvek egyedi betűinek, írásképének felismerésére is.

3 RENDSZERTERV

Ebben a fejezetben a program részletes működését, alkotó elemeinek összetételét és feladatát fejtem ki részletesen.



3.1. ábra Rendszerterv folyamatábrája, Forrás: saját kép, Visual Paradigm Online-al készítve

3.1 Részek rövid ismertetése

Bemenet

A bemeneti képet telefonnal készítem, az A4-es papírt természetes fényben, teljes nagyságában fotózom. A szöveg rajta bárhol elhelyezkedhet, több sorban, a betűméret az 5mm-es betűmérettől felfelé bármeddig nőhet, de a betűk között távolságot kell hagyni. (nyomtatott írás) A szótárprogram működésének érdekében a szöveg angol szöveg kell, hogy legyen.

Képfeldolgozás

Képfeldolgozás során a telefonnal fotózott, és számítógépről beolvasott képet meg kell tisztítani az esetleges zajszennyezéstől, ki kell emelni, és a betűket a további feldolgozásra elő kell készíteni. Ezt az OpenCV eszköztárával teszem meg.

Elválasztás

Hogy a betűket az AI számára megfelelő formátumba hozzuk, és közben ne sérüljön azok sorrendje, fokozatosan kell lebontani a szöveget. Először a sorokat, majd a szavakat kell különválasztani, legvégül pedig a betűket. A betűket ezután a különböző méretű képekből azonos, 28x28-as formátumba konvertálom.

Betűfelismerés

A képeket ezután a korábban elkészített AI modell felismeri, és hozzátársítja a megfelelő betűt, majd az eredményt kimenti.

Szótár

A szótár az AI által elkövetett felismerési hibák javítására alkalmas. Betölt egy szótárt külső forrásból, majd annak segítségével meghatározza egyes szavak mik lehettek, majd előfordulási gyakoriság alapján dönt a megoldásról.

Kimenet

A kimenet egy grafikus felületen megjelenített szöveg, ami a végső, szótárazott megoldást tartalmazza.

3.2 Adatbázis

Az tanításhoz szükséges képeket az EMNIST Balanced adatbázisából veszem, ami a NIST Special Database 19-ből származik. A NIST Special Database 19 egy több mint 3600 írótól származó, 810.000 kézzel írott betűt tartalmazó OCR fejlesztés támogatására összeállított adatbázis. Az adatbázis egyetlen hibája, ami miatt nem lehet egyenesen azt használni a fejlesztés folyamán, hogy nem egyenlő számú mintát nyújt minden betűcsoportból, hogy a formátuma nem megfelelő a fejlesztéshez. E miatt jött létre az EMNIST, ami 28x28-as formátumban tárolja a képeket, valamint csoportokra osztja azokat. A Balanced Database nevű adatbázis egyenlő számban tartalmaz minden betűt és számot, a Digits a számokból tartalmaz egyenlő mennyiséget, a Letters csak betűket tárol, de azokat egyenlő mennyiségben, a ByClass és a ByMerge pedig különféle sorrend szerint, de az egyeredeti NIST betűit tartalmazza, 28x28-as formátumban.

3.2. ábra Minták az adatbázisból, Forrás: saját kép

Az általam használt adatbázis egyetlen hátránya, hogy néhány betű esetében összevonta a kis és nagybetűs verziót. Ilyen például a „K,k” vagy a „W,w”. Tehát olyan betűk esetében, ahol amúgy sem lenne nagy különbség a kisbetűs írás, és a nagybetűs írásmód között, mindig a nagybetű ASCII kódja kerül párosításra a képpel. Ez szinte semmiben nem befolyásolja a felismerés pontosságát, hiszen a betűt felismeri, csak nagybetűként.

A projekt pontosságán viszont igenis változtat, hiszen ezen adatbázis használatával elveszik a lehetőség, hogy a nagybetűvel írott szavakat teljes pontosságukban ki tudja íratni a program. A szótár használatához muszáj, hogy kisbetűs formában érkezzenek a szavak, így még előtte kisbetűssé alakítom az összes felismert betűt, utána pedig, ha a szótár nem fogja, máshogyan nem lehet visszanyerni a nagybetűket.

4 FEJLESZTÉS

4.1 Általános információk

Programomat Google Colab, később pedig Pycharm segítségével írtam meg, Python (3.9) nyelven.

Stéfane Fermigier szavaival „a Python egy portabilis, dinamikus, bővíthető, ingyenes nyelv, ami lehetővé teszi a programozás moduláris és objektumorientált megközelítését egyaránt. 1989 óta fejleszti Guido van Rossum, és számos önkéntes.” [23]

A következő könyvtárakat használtam a fejlesztés során:

- tkinter,
- a collections-ból a Counter,
- cv2 (OpenCV),
- matplotlib.pylab,
- matplotlib.pyplot,
- numpy,
- pandas,
- tensorflow,
- keras,
- os,
- tqdm.

Ezek egy részét a mesterséges intelligencia fejlesztéséhez, másokat a képfeldolgozáshoz szükséges algoritmusok megírására.

4.2 Mesterséges Intelligencia

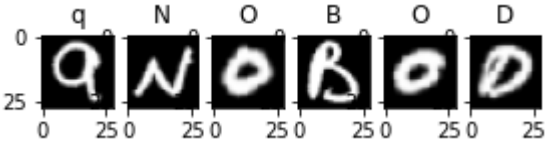
Fejlesztésemet a mesteréges intelligencia előállításával kezdtem. Célom az volt, hogy egy olyan modellt állítsak elő, ami az elmentett súlyértékei alapján később egy minimum 80%-os pontosságú felismerést tudjon végezni élesben.

Az AI elkészítését a korábban kiválasztott betű adatbázis betöltésével kezdtem. A „.csv" formátumú adatfile első oszlopában a hozzá kapcsolódó „.txt" állomány sorszáma van, így jelölve, hogy melyik sorban milyen betű van tárolva. A „.txt" soraiban található ASCII kódokat hozzátársítottam a megfelelő betűhöz vagy adott esetben számhoz. Az állományból beolvasott betűk el voltak forgatva, így visszaforgattam őket normalizálás előtt.

A Keras Szekvenciális modelljét használva felépítettem a hálómat, amely konvolúciós, maxpool, flatten és dense rétegek kombinációjából áll.

Próbálkozások után az optimális megoldásnak azt találtam, ha harmincszor fog végig a tanítás, húszas batch-size-al, és minden egyes hibaszázalék csökkenéskor elmenti az adott súlyértékekkel a modellt. Ha a tanításonkénti hibaszázalék nem csökken háromszor egymás után, a tanítás idő előtt befejeződik.

Az általam készített a modell az adatbázis saját teszt képein 94%-os pontossággal dolgozik, míg tanulás közben 95%-os pontosságot ért el.



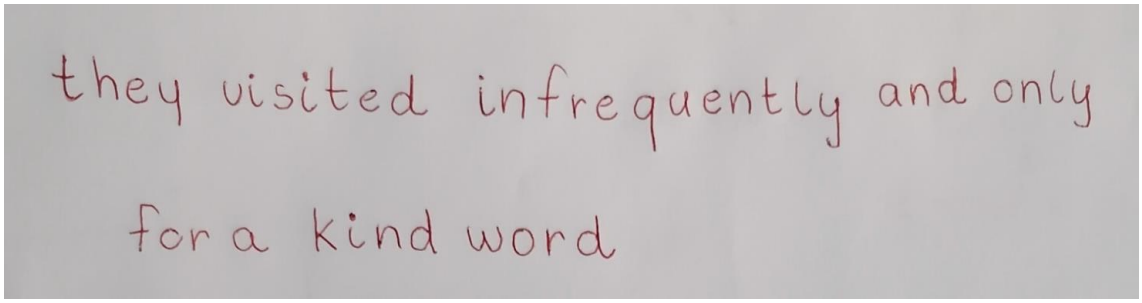
4.1. ábra Teszt mintákon futtatott betanított háló eredményei, Forrás: saját kép

4.3 Képfeldolgozás és részek szétválasztása

Az előfeldolgozás folyamán a cél a telefonnal fotózott képet a mesterséges intelligencia számára feldolgozható formátumba (binarizált 28x28 pixeles képek) átalakítani. A célra az OpenCV eszköztárát használtam.

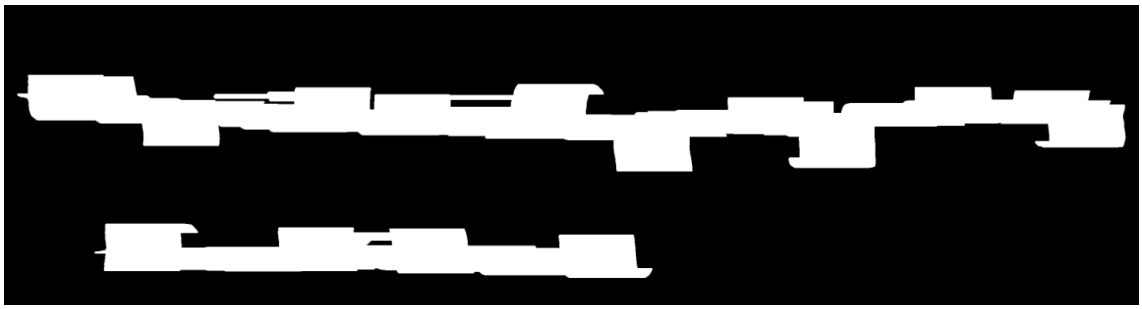
Első lépésként a színes képet, szürkévé kellett alakítanom, tehát az OpenCV egyik függvénye segítségével megváltoztattam a kép színterét. Az RGB színtérben lévő képek három, míg a szürkeskálázott képek egy csatornásak. Ez azért van, mert míg az RGB esetben egy pixel színét a három alapszín együttese adja ki (piros, zöld, kék) addig a szürke képen csak az adott pixel fényességének értékét kell jelölni. Szürke skálázás után a kép pixeleinek értékeit egy homályosító függvény segítségével közelebb hoztam egymáshoz, így később a küszöbérték alapján meghatározott fekete-fehér képen a pixelvesztés nagyban lecsökkent. Még a binarizálás előtt a megfelelő zajszűrő függvénnyel a gyakoribb kép hibákat eltüntettem.

Miután az egész kép előkészítése megtörtént, neki láttam a részekre bontani, és feldolgozni a szöveget. Ennek legfontosabb lépései sorrendben a következők: sorok meghatározása, szavak szétválasztása, betűk kiemelése a szavakból, a kiválogatott betűk átméretezése és mentése.



4.2. ábra Egy telefonnal fényképezett minta, Forrás: saját kép

A sorok szétválasztását a dilatációs eljárás mátrixának X irányú torzításával oldottam meg. Erre azért volt szükség, mert bár az emberi agy egyértelmű, hogy egy ilyen szöveget jobbról balra olvasunk, a számítógép számára nem. Míg az ember nem lép át olvasás közben a lentebbi sorba, csak ha végeztett az adott sor összes betűjével, a gép koordináták, számok, matematikai műveletek és pozíciók alapján dolgozik. Nagyon sokszor futottam bele abba a hibába, hogy bár a szavakat megtalálta a rendszer, rossz sorrendben dolgozta fel őket, így végeredményben nem kaptam vissza az eredeti szöveget. Ezen akadály kezelése találtam ki a fenti megoldást, hogy a sorokat külön szétválasztom, mielőtt megpróbálnám a szavakat megkeresni bennük. Az így kapott képen (4.3. ábra) alkalmazom aztán az OpenCV kontúrkereső és bounding box funkcióját, amely egy, az adott sort körülölelő téglalapot ad vissza, mint ROI. Ebben a szakaszban biztosítom, hogy az ékezetek a szöveggel maradnak a betűk szétválasztásáig.



4.3. ábra X irányba torzított betűk, Forrás: saját kép

4.4. ábra Képfeldolgozási lépéseken átesett kép, Forrás: saját kép

A tömbökbe mentett, sorokat tartalmazó téglalapokat aztán egy másik eljárás maszkolás segítségével szavakra bontja. Ennek során egy, a sor méreteivel egyenlő np.array tömböt (téglalapot) feltöltök nullákkal, majd az így létrehozott fekete képbe belerajzolom a korábban felismert kontúrokat, és ezt használom a zaj mentesített sor képén maszkként. A 4.5 ábrán a legfelső sorban látható az összefüggő részek észlelésére használt torzítás, alatta pedig sorra az észlelt körvonalakhoz tartozó maszkolt képek.

Ez a megoldás a szavak közti üres helyek elemzése, és bonyolult matematikai műveletek helyett nyújt egy megoldást a szavak közti szünetek megtalálására.

4.5. ábra Első sorban a szétválasztásra használt maszk, alatta pedig soronként a szétválasztott szavak, Forrás: saját kép

A betűkre bontás előtt fontos még egy Y irányú transzformációt alkalmazni a képen, hogy az ékezetek a szavakkal maradjanak. Az (4.5.ábrán) látható a torzított szó, és a kimentett betűk. Még mentés előtt a program átméretezi a betűket 28x28-as formátumba, az arányok megtartása mellett. Ennek menete, hogy egy maximum kiválasztás segítségével meghatározza, hogy melyik oldala a legnagyobb a képnek, az alapján létrehoz egy fekete négyzetet, és abba illeszti bele a korábban téglalap alapú betűt.

A felismerés pontosításának érdekében egy minimális szegély is kerül a betű legszélső pixele és a végleges négyzet oldala közé. (4.6.ábra)

4.6. ábra Y irányú torzítás, alatta pedig az így szétválasztott betűk Forrás: saját kép

Az így létrehozott, betűt tartalmazó, fekete hátterű négyzetet ezután szabadon lehet kicsinyíteni a kívánt méretre. A betűk nevei rendre: „letter{betű darabszáma}_{szó szám}.jpg", tehát például az 5. ábrán látható "t" betű átméretezett másának neve a letter1_0.jpg. A képek mellett itt kerül mentésre egy tömbbe az egyes szavak betűszáma is.

4.7. ábra A végleges betűk, Forrás: saját kép

A helytakarékosság érdekében a program végleges verziójában nem mentődnek el a képek külön, hanem végig a memóriában maradva dolgozódnak fel. Az asztali, fejlesztői verzióban megtartottam a fent említett funkciót, minták gyűjtésnek és eredmények ellenőrzésének érdekében.

4.4 Eredmények és szótár

A visszaolvasott betűk, és a kimentett mesterséges intelligencia modell segítségével a program megtippeli, milyen betű lehet a beolvasott képeken, egy szótár algoritmus segítségével pedig a tippelt eredményeket megpróbálja pontosítani. A korábban említettek miatt, a nagybetű kisbetű kérdés kiküszöbölésére, minden szót kisbetűs formában adtam meg a szótárnak.

```
A szöveg: the4 visited infreguentiy and oniy for a kind word
Javított szöveg:  they visited infrequently and only for a kind word
```

4.8. ábra Első sorban a szótározás előtti eredmény, alatta pedig a szótár utáni eredmény látható, Forrás: saját kép

A szótár működése Peter Norvig [27] munkásságát veszi alapul, a saját feladatomra készített specifikus részekkel kiegészítve. Az algoritmus beolvas egy olyan dokumentumot, ami nagy mennyiségű szót tartalmaz, optimális esetben egy szótárt, kevésbé optimális esetben egy hatalmas könyvet, vagy egy bármilyen nagyon nagy szószámú írást, „.txt” formában. Ennek alapján kiszámítja minden egyes szó előfordulásának a valószínűséget. Például angol (és magyar) nyelvben gyakori, hogy önmagában áll egy „a” betű, így egy rosszul felismert egymagában álló „e” betűt ki fog javítani „a”-ra.

Hárombetűs szavak esetében is említenék egy példát a fenti jelenségre. Legyen mondjuk a javításra váró szó a „lok”. A „sok” és a „fok” is egy betűben tér el a hibás szótól, tehát ebben a tekintetben ugyan olyan esélye van mindkettőnek. Viszont ha a szótárban a „sok” többször fordul elő, mint a „fok”, az algoritmus a „sok” szót fogja helyes megoldásnak kiadni.

Visszatérve az algoritmus működésére, miután meghatározta a szavak valószínűségét, megpróbálja megkeresni a hibás szavaknak a javított verzióját úgy, hogy elkezdi átdolgozni a betűket. Ezt négy féle képpen teszi meg. Kivon egy betűt, két egymás melletti betűt megcserél, kicserél egy betűt egy másikra, vagy hozzáad egy betűt.

```
Lehetséges jelöltek:  Ez a szó a szótárban:  set()
Egy betűs korrigálások után ilyen szavak lehetnek:  {'identify'}
Két betűs korrigálások után ilyen szavak lehetnek:  {'identity', 'identify'}
A szó önmagában: ['iaentify']
Legvalószínűbb jelölt:  {'identify'}
```

4.9. ábra A szótár által megtalált lehetséges megoldások, Forrás: saját kép

Természetesen, ha a szó eleve szerepel a szótárban, tehát tökéletesen ismerte fel a mesterséges intelligencia az összes betűt, akkor azonnal megoldásként kiadja a szót. Egyéb esetben a szó minden betűje esetlén végigmegy az ábécén, és mind a négy

metodikával átdolgozza őket, legenerálja az összes lehetséges opciót. Ha megtalálja valamelyik opciót a szótárban, azt kimenti, majd egy maximum kiválasztással kiemeli a legvalószínűbbet, és azt küldi el eredményül. Ez után megcsinálja ugyan ezt, csak most két egymás mellett lévő betűt cserél ki folyamatosan. Ezek után a két kiválasztott szót, valamint az egyéb saját javításokat tartalmazó szavakat valószínűségi sorrendbe rendezi, majd kiválasztja, hogy melyik szó a leginkább esélyes hogy helyes megoldást legyen, és ezt a szót teszi bele végül az eredményeket tartalmazó tömbbe.

A pontosság növelésének érdekében tettem bele megfigyelésen alapuló egyedi javításokat. Többszöri tesztelés után azt vettem észre, hogy vannak betűk, melyeket különösen nagy gyakoriággal kever össze a program. Ilyenek például az „i” és a „j”, vagy „l” és az „i”. Ennek az oka, az „i” és a „j” esetében, hogy a kettő közti különbség nagyon leegyszerűsítve annyi, hogy a „j” szára hosszabb. Ez a hosszúság béli különbség viszont szinte teljesen elveszik a képek 28x28-as formába alakítása közben. Éppen ezért, egy nagyon görbén írt „i” betűt bármikor összetéveszthet a program egy kicsit egyenesebb „j”-vel.

4.10. ábra A betűk hasonlósága, Forrás: saját kép

Az „i” és az „l” esetében viszont, minél kisebb a hely az „i” és az ékezete között, és az ékezet minél inkább a betű felett helyezkedik el, annál jobban lászik, hogy csak pár pixel különbség van aközött, hogy az egy pixelhibás „l” betű vagy egy nagyon szabályos „i” betű.

A képen látható (4.9. ábra) a NIST Special Database 19 adatbázisából kiszedett „l” (baloldalt) és „i” betű (jobb oldalt). Ezen adatbázisból származik az általam használt EMNIST Dataset. Ha az „l” betűt ráillesztjük az „i” betűre észrevehető, hogy mindössze a piros színnel jelölt részek különböznek. S bár emberi szemmel azonnal látjuk, hogy melyik az „i” betű, egy mátrix alapú mesterséges intelligencia számára nagyon kis különbség van a két kép között.

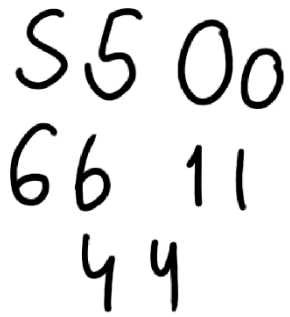
```
Eredeti szó:  ijving
Korrigált alap szó:  irving
Számok kicserélve, korrigálva:  irving
Számok kicserélve,j/i cserélve, korrigálva:  living
Számok kicserélve,i/l cserélve, korrigálva:  living
Végső tipp:  living
```

4.11. ábra A szótár által megtalált lehetséges megoldások, a megfelelő betűk cseréje után Forrás: saját kép

A másik leggyakoribb probléma a számok és betűk keveredése volt. A hasonlóan írandó betűk és számok sokszor vezették félre a mesterséges intelligenciát, így eme akadály megoldására is saját szabályrendszert állítottam fel.

A leggyakrabban összekevert betű és szám párosok a(z) „5”és”s”, 0” és „o”, 6”és”b”, „1” és „l”, valamint a „4”és”y”. Ezt a tesztelések során állapítottam meg, miután sorra rossz eredményeket adott a szótár, hiszen a számok cseréje miatt plusz egy műveletet kellett végeznie, és így már a két hibával rendelkező szavakat nem tudta kijavítani soha.

Külön feltételként hozzáadtam, ha a szóban valahol egy fent megadott szám található, akkor még minden nemű korrekció előtt cserélje ki a számot a feljebb megadott betű párjára. Természetesen, ha csak önmagukban szerepelnek a számok, a rendszer nem fogja kijavítani őket valamilyen betűre.



4.12. ábra Egymással hasonlatosan írandó betű-szám párok, Forrás: saját kép

A 4. 11.-es ábrán látható a szótár utolsó opcióinak felsorolása, valamint azok alapján a végső döntés, egy szóra nézve. Az első sorban látható a mesterséges intelligencia által felismert eredeti szó, a második sorban pedig az alapszón lefuttatott javítás eredménye. Látható példánál viszont a „0” és „o” kicserélése nélkül hibás eredményt adott volna vissza a szótár.

```
Eredeti szó:  n0om
Korrigált alap szó:  nom
Számok kicserélve, korrigálva:  room
Számok kicserélve,j/i cserélve, korrigálva:  noom
Számok kicserélve,i/l cserélve, korrigálva:  noom
Végső tipp:  room
```

4.13. ábra Szótár által helyesnek gondolt megoldások, Forrás: saját kép

4.5 Különleges fejlesztések

4.5.1 Folyóírás feldolgozása

A programomban az egy sorban található szavak szétválasztását és a betűk szétválasztását ugyan azzal a metodikával oldottam meg, X vagy éppen Y irányú dilettálással, majd maszkolással. Ez a módszer azért működhet, mert az angol kézírás nem folyóírással ír, hanem nyomtatott írással, így a betűik egyáltalán nem, vagy ritkán érnek össze. Mielőtt viszont megállapodtam volna a fenti megoldásnál, először megpróbáltam a folyóírással írt szövegek feldolgozására is kitalálni valamilyen megoldást.

4.14. ábra A folyóírással írt kezdő szó, Forrás: saját kép

Vishal Rajput és társai [27] kutatásának alapján kezdtem neki a fejlesztésnek, az eddig is használt Python környezetben. A célom az volt, hogy pixel szám alapján meghatározzam, hogy hol vannak azok részek, ahol valószínűleg el kell választani a betűket egymástól.

Az algoritmus bementként megkapja az elő feldolgozott (binarizált, levágott) szót, majd megcseréli a tengelyeit, hogy oszloponként legyenek tárolva az értékek. Az egy oszlopban található pixelek összegének számát ezután összehasonlítom egy küszöbértékkel, hogy meghatározzam hol találhatóak a betű részei, és hol van csak üres háttér. Az összehasonlítás alapjául szolgáló értéket az átlagpixelszám alapján számítottam ki. A kép szélességével megegyező egyesből és nullákból álló tömbbe lementettem, hogy melyik oszlop tartalmazza a betűk részeit, és hol van az átlagosnál kevesebb pixel.

4.15. ábra Túlszegmentált szó, Forrás: saját kép

Az ábrán piros vonalat húztam oda, ahol tömbben 0 az érték. Látszik, hogy a szavak közötti üres helyeket kívül a betűk közti elválasztó vonalak helyét is meg lehet találni ezzel a módszerrel, viszont a jó vágások mellett a program rengeteg hibás vágást is végezne. A túlszegmentálást a betűk átlagos hosszának meghatározásával próbáltam csökkenteni. A korábban meghatározott bináris tömbben az egymás után következő egyesek száma alapján lehet következtetni egy betű hosszára. (4.17. ábra).

```
hanyegyes van
[94, 72, 2, 1, 2, 66, 1, 13, 1, 16, 1, 8, 1, 2, 2, 1, 1, 42, 1, 4, 3, 3, 1, 17, 14, 87, 100]
```

4.17. ábra Az egymás után következő azonos karakterek darabszámát tartalmazó tömb

Az számsor mindig nullával kezdődik és fejeződik be, tehát az első és az utolsó érték (94,100) a szó előtti üres részt jelöli. Minden második tag az egymás után következő egyesek száma. Ahol páros számú helyen az átlagnál kisebb szám szerepel, ott túlszegmentálás történt, tehát össze kell vonni a feldarabolt részeket. Maszkolás segítségével egy új, a fenti tömb hosszával egyenlő tömböt feltöltök nullákkal és egyesekkel, ahhoz mérten, hogy az adott hely maximum 2 helyes távolságában milyen értékek találhatóak. Az így kapott tömb alapján pedig újra generálom a képet, és hogy hol kéne lennie a végső vágásoknak.

4.16. ábra Végső eredmény, Forrás: saját kép

Látható, hogy bár a túlszegmentálás problémája valóban megoldódott, maga a végső döntés helytelen. Az „l” és az „m” közti korábban megtalált elválasztás eltűnt, átcsúszott az „m” betű felső görbületére. Ennek oka az, hogy az „l” és az „m” között sok felesleges elválasztás született, amik a legutolsóig tolódtak, ami az „m” felső görbületénél van. (ábra)

Összességében elmondható, hogy bár valamilyen eredményt fel tud mutatni ez a fajta megközelítés, közel sem olyan megbízható, mint a szakdolgozatban használt megoldás. Ennek oka többek között már az elgondolásban is megtalálható. Hogyha a betűk összekötésére használt vonal mindig vékonyabb, mint maga a betűk vonala, akkor semmi gond. De abban az esetben, ha ez nem áll fenn, a fenti példában bemutatott akadállyal találhatjuk szembe magunkat. Az „l” betű összekötője, és az „m” betű felső görbülete igazából egymás tükörképei, pixelszámban megegyeznek.

Ezen programrész fejlesztése közben kezdtem el igazán megérteni, hogy az offline kézírás felismerés nagyon korlátozott, és a jövő az online felismerésben rejlik.

4.5.2 Ékezetek

Az ékezetek problémájának kezelésére nagyon sok dolgot megpróbáltam. Először megpróbáltam részekre bontani a betűket, és külön detektálni az ékezetet és betűt. Hogyha a szavaknál a betűk külön képpé vágása után, egy betű után egy bizonyos nagyságú kép jelent meg, megállapítottam, hogy az minden bizonnyal egy ékezet. Kiszedtem a tömbből, de a helyét megjegyeztem, és később, miután a mesterséges intelligencia visszaadta a betűfelismerés eredményét, a korábban megállapított helyekre reflektálva bizonyos betűket kicseréltem ékezetes párjukra. Ez egyértelműen egy rossz megoldás, nem csak feleslegesen növeli az erőforrás igényt, valamint pontatlan, akadályozza a hibakezelést, de ráadásul nem is a mesterséges intelligencia által került megvalósításra az ékezet felismerés.

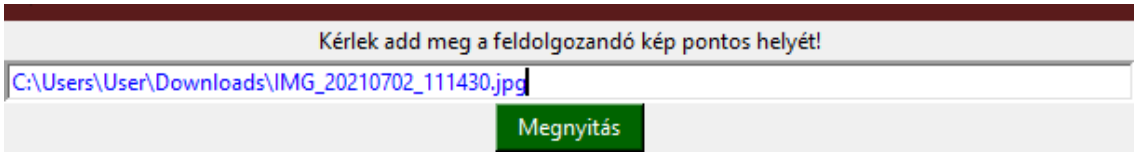
Ahhoz, hogy a dolgozatomban használt mátrixos megoldással a mesterséges intelligencia ékezetes betűket is képes legyen felismerni, egy magyar nyelvű adatbázisra lenne szükség, ahol az adott ékezetes betűkhöz elegendő tanítási mintát biztosítottak. Mivel ilyet nem találtam, a magyar nyelvre történő fejlesztést elvettem.

Végül találtam egy megoldást, amivel a betűkkel együtt ki tudtam menteni az ékezeteket egy képbe. Az X illetve Y irányba történő dilettálással a betűk fölötti ékezetek teljes egészükben megőrizhetőek, így ha a későbbiekben egy magyar nyelvű adatbázison tudom tanítani a mesterséges intelligenciát, a program képes lenne azonnal kezelni az ékezetes betűk kimentését felismeréskor.

5 TESZTELÉS

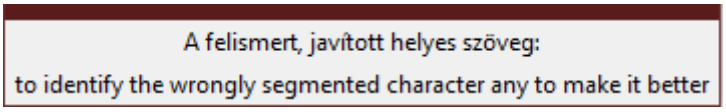
5.1 Felhasználói felület

A képen látható a program felhasználói felülete. (5.1. ábra) A Python elég limitált eszköztárral rendelkezik, ha grafikus megjelenítésről van szó, így próbáltam az egyszerű elrendezéseknél maradni.



5.1. ábra A program grafikus felülete, Forrás: saját kép

A program a feldolgozandó képet annak pontos nevével, forrás helyével jelölve kéri be. Az eredményt egy másik ablakban jeleníti meg, hogy újraindítás nélkül további képeket lehessen beadni neki.



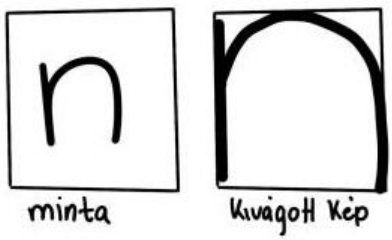
5.2. ábra A program grafikus felülete, eredmény megjelenítés Forrás: saját kép

5.2 Képfelismerés

A legtöbb tesztelést a képfelismerés során kellett végeznem. A mesterséges intelligencia fejlesztése után szinte azonnal rájöttem, hogy a beadott képek minősége befolyásolja legnagyobb részben a végeredményt. Rengeteg féle metódussal próbálkoztam, cserélgettem egymás után a különféle eljárásokat. A színes kép szürkítése egyértelmű lépés volt, valamint később a binarizálás is. De közte a kép megfelelő kezelése sok próbálkozást vett igénybe. Először csak a zajcsökkentéssel próbálkoztam, s bár valóban segített a képi hibák csökkentésében, végeredményben maguk a betűk homályosak, nehezen feldolgozhatóak lettek. Később a megfelelő erősségű homályosítás bevezetésével tudtam elérni, hogy a betűk élesek legyenek a beolvasás során.

5.3 Szétválasztás

Miután a képfeldolgozással kapcsolatos problémát kiküszöböltem, egy újabb akadállyal szembesültem. A kezdetleges szétválasztási módszereimmel a betűket maszkolás



5.3. ábra „n” betű a tanításra használt adatbázisban (balra), és az első kinyeréskor (balra) Forrás: saját kép

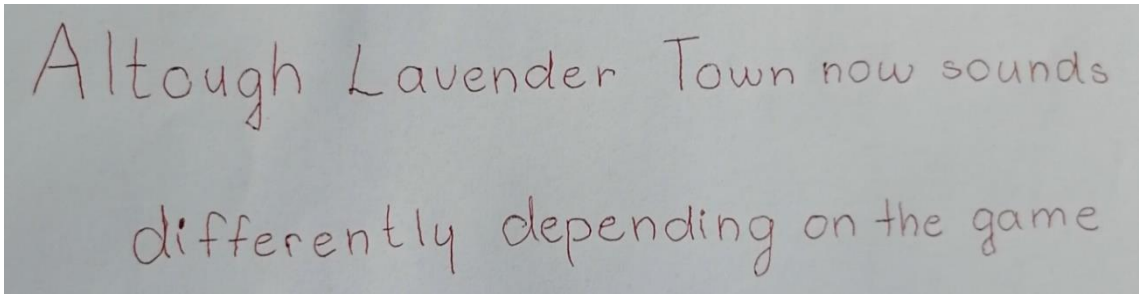
segítségével szétszedtem, összekicsinyítettem, és egy az egyben elküldtem a mesterséges intelligenciának. Ez rögtön két problémát is felvetett. Az egyik az volt, hogy a betűk eltorzultak a közvetlen kicsinyítés hatására, egy „e” és egy „l” betű azonosan nézett ki. A másik pedig, hogy a betűk a teljes területet lefedték a képen, így a mesterséges intelligenciának esélye sem volt a felismerésre.

Ezen két problémát egyszerre oldottam meg a kép arányos kicsinyítésével, és egy üres tömbbe való behelyezésével.

Először megvizsgáltam, hogy egy adott betű melyik irányban nagyobb, széltében vagy hosszában. A hosszabb oldal meghatározása után annak mentén kezdtem meg a kicsinyítést, hogy a betűk arányai megmaradjanak, megoldva így a korábban említett „i” és „j” közötti torzulás okozta hasonlóságot. Miután létrehoztam az arányosan kicsinyített képet, azt egy 28x28-as négyzetté kellett formálnom. Ha eleve nagyobb volt a betű, akkor kicsinyíteni könnyebb volt az OpenCV eszköztárával, viszont, ha kisebbre sikerült a betű, a fehér pixeleket, ami a nagyobb négyzetbe történő behelyezés után maradna, ki kellett tölteni. Éppen ezért eleve egy fekete pixelekkel (0-val) feltöltött tömbre tettem rá a képet, azonnal kitölte az amúgy fehéren maradó helyeket. Külön figyeltem arra, hogy megfelelő szünetet hagyjak a kép széle és a betű között, minél jobban hasonlítva így a tanításként használt képekre.

5.4 Használati tesztek

A tesztelés során sokféle képet adtam be a programnak, voltak, amik közel kilencvenkilenc százalékos eredménnyel dolgoztak, de volt olyan eset is, ahol nagyon alacsony lett az algoritmus pontossága. A tesztesetekként használt képeket én magam írtam, kézzel, A4-es papírra, a papírt teljes méretében, természetes fényben fotózva. A szövegek angol nyelvű irományokból lettek kiragadva. Teszteltem kis és nagybetűk felismerésére, képi hibákra, összeérő és távol lévő betűk esetére is.



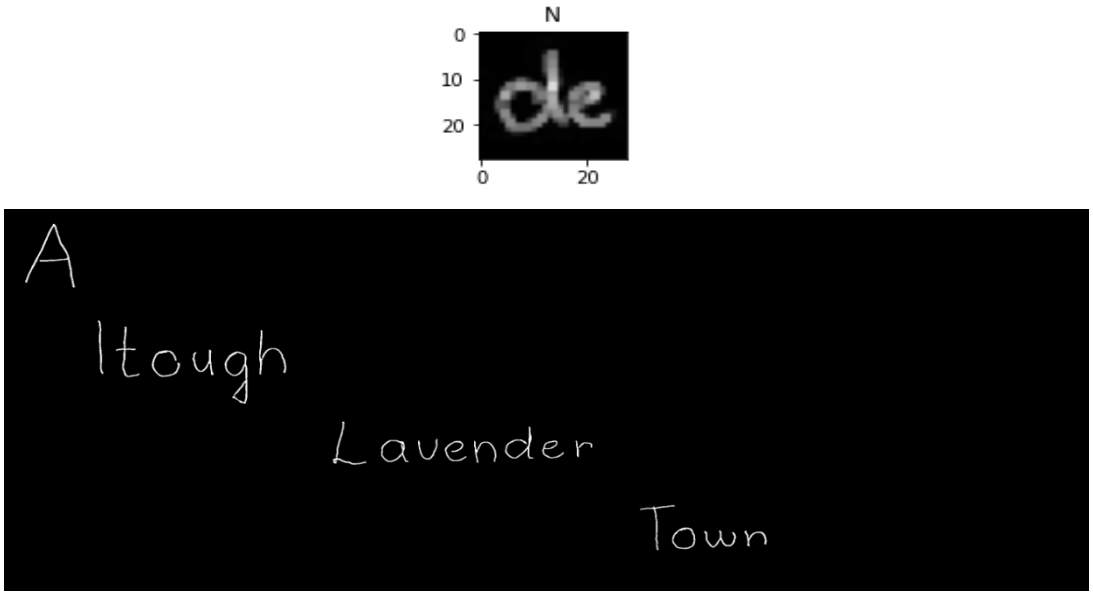
5.4. ábra Tesztkép elmosódással, Forrás: saját kép

Az 5.4. ábrán látható tesztképen fényképezés közben elmozdult a telefon, ezzel a szöveg jobb oldalán lévő betűk elmosódtak. Ilyen mértékű torzulás nem befolyásolta nagymértékben a végeredményt, minden homályos betűt pontosan ismert fel a program, kivéve a 0 és o közötti különbséget, de az a későbbiekben javításra kerül.

```
A szöveg: a itough lavender town n0w sounds diffeteniі4 npending on the game
Javított szöveg:  a tough lavender town now sounds diffeteniiy pending on the game
```

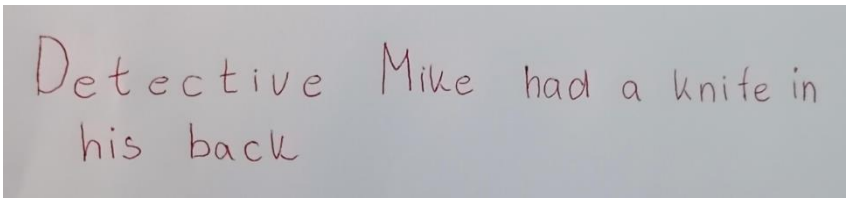
5.5. ábra A teszt végeredménye, Forrás: saját kép A szövegben előfordult hibák.

Ebben a példában három hiba miatt nem sikerült a tökéletes felismerés. A „differently” szónál kudarcot vallott a szótár algoritmus, minden bizonnyal amiatt, hogy túl sok hiba volt a szóban ahhoz, hogy meg tudja tippelni a helyes megoldást. (5.5. ábra) Az „Altough” esetében a „A” betű túl távol állt a vékony „l” betűtől, így az x irányú dilettálás nem vonta egybe a két betűt. Illetve volt még egy pont, ahol két betű túl közel került egymáshoz, emiatt nem tudta őket szétválasztani az algoritmus.



5.6. ábra Legfelül az összeérő két betű, alatta a különválasztott szó. Forrás: saját kép

Egy másik példában azt teszteltem kifejezetten, hogy milyen közel kell írni egymáshoz a betűket, hogy az algoritmus még felismerje. Az alábbi ábrán látszik az eredi kép (5.7. ábra) az alatt pedig a felismerés eredménye. Jól látszik, hogy minden betűt tökéletesen felismert a mesterséges intelligencia, de mivel túl távol vannak egymástól, így nem rakta össze őket egy szóvá.




```
A szöveg: det ec t i v e mike bad a knjte in hjs bacw
```

5.7. ábra Tesztkép, a betűk távolságának tesztelése. Forrás: saját kép

Teszteltem azt a lehetőséget is, hogy mi történik akkor, ha egy teljesen üres képet kap a program. Ebben az esetben is lefut, nem dob hibát, pusztán egy üres kimenetet ad.

5.5 Integrációs tesztek

Az integrációs teszt célja a program részeinek egymáshoz viszonyulásának vizsgálata, az egész rendszeren átívelő funkciók tesztelésének érdekében. Ez egy nagyon fontos része volt a programom helyes működésének biztosításában, hiszen nagyban függnek egymástól az elemek.

Teszt neve	Leírás	Elvárt eredmény	Eredmény
Képfeldolgozás teszt	Leteszteltem, hogy a programrész megfelelően végzi-e el a képfeldolgozási lépéseket, megfelelő formátumú kimenetet biztosít a további programrészek számára.	Sikeres	Sikeres
Körvonal teszt I.	Leteszteltem, hogy a program megtalálja-e a sorok körvonalait.	Sikeres	Sikeres
Szélsőérték (laphiba) teszt	Leteszteltem, hogy adott értéknél kisebb méretű képet nem küld tovább a programrész.	Sikertelen	Sikertelen
Körvonal teszt II.	Leteszteltem, hogy a program megtalálja-e a szavak körvonalait.	Sikeres	Sikeres
Szélsőérték (ékezet) teszt	Leteszteltem, hogy adott értéknél kisebb méretű képet nem küld tovább a programrész.	Sikertelen	Sikertelen
Méret teszt	Leteszteltem, hogy a programrész valóban megváltoztatja a kép méreteit.	Sikeres	Sikeres

6 TOVÁBB FEJLESZTÉSI ALTERNATÍVÁK

6.1 Szótár

A szótár alapvető akadálya, a szövegkörnyezet ismeretének hiánya. Hogy ha valamilyen szóra két azonos valószínűségű, vagy közel azonos valószínűségű megoldást is talál, akkor egyedül a számok alapján fog dönteni, amit pedig nem túl célszerű döntés.

A szavakhoz társított valószínűségi értéket befolyásolja, hogy éppen abban a dokumentumban hányszor szerepel az a szó. Tehát például egy szótár esetében nagyobb eséllyel gondol egy szót „noun”-nak, mintsem „noon”-nak, pusztán a szótár természete, és az abban előforduló szavak miatt.

Viszont ha a program tisztában lenne a szövegkörnyezettel, és az tudnál befolyásolni döntéseit, sokkal pontosabb eredményeket érhetnénk el. A mai telefonokban, számítógépekben található automata helyesírás javító a legtöbb esetben figyelembe veszi az általunk leggyakrabban használt kifejezéseket, valamint az azokhoz a kifejezésekhez társítható további szavakat is a javítás közben. Hogyha elkezdünk írni egy mondatot szinte azonnal, felajánlja a gyakran használt szavakat, amikre szükségünk lehet benne.

A szótár számára egyértelműen a legjobb fejlesztési lehetőség az lenne, ha mesterséges intelligenciát alkalmaznánk a szövegkörnyezet felismerésére, és az alapján próbálná javítani a szavakat a program. A megfigyelésen alapuló korrekciók megtartása mellett további alapvető hibajavító funkcióval lehetne bővíteni a rendszert, például helység és ember nevek külön felismerésével, és azok megfelelő írásmódjának betáplálásával.

6.2 Karakterfelismerés

A karakterfelismerés fejlesztéséhez egy hatékonyabb betűszétválasztó algoritmus kifejlesztése lenne a legoptimálisabb. Kezdetleges fejlesztéseimből tanulva beláttam a mátrixos karakterfelismerés határait, annak előnyeivel egyetemben. Egy olyan algoritmus, amely képes az egymással szorosan írt betűket is szétválasztani, nagyban javítaná a program kezelhetőségén. Például a kezdetleges kézírás szétválasztó részprogram mentén a mátrixos feldolgozásformát megtartva lehetne elérni jelentős eredményeket.

6.3 Felhasználói felület

A Python elég limitált grafikai megjelenítő eszköztárral rendelkezik, ezért egy stílusos, modern grafikus felhasználói felület megvalósítása komoly akadályokba ütközik. Hogyha már egy telefon kameráját használó programról van szó, optimális lenne egy Androidos alkalmazással összekötni a kódot. Vannak keretrendszerek, amelyek segítséget nyújtanak Python kód írásában Android-ra, például a BeeWare, vagy a Kivy.

7 ÉRTÉKELÉS

Az egyetemen történő tanulmányaim során nem volt része a képzésemnek a képfeldolgozás, sem pedig a mesterséges intelligenciafejlesztés Python környezetben, így értékelésem ebből a szempontból végzem.

Mivel minden képfeldolgozással kapcsolatos feladat és probléma megoldásának önmagamnak kellett utánajárnom, mindenféle alaptudás nélkül, így programom nem lett olyan kiemelkedően optimális és hibatűrő, mint egy iparban használt alkalmazás.

Ennek ellenére sikerült elérnem különféle transzformációk megfelelő tulajdonságainak kihasználásával, hogy egy telefonnal fényképezett szöveget fel tudjon dolgozni a programom, a betűk összekeverése nélkül, és az ékezetek megtartása mellett.

Így már képes voltam tovább haladni, és a mesterséges intelligencia pontosítására fókuszálni. Első sorban azért is választottam a Pythont a szakdolgozatom megírására, mivel bár nem dolgoztam még ez előtt a nyelvvel soha, ezt találtam az optimális környezetnek egy OCR program elkészítéséhez. A Tensorflow és a Keras valóban bő eszköztárat biztosítottak egy karakterfelismerő betanításához.

S bár el kellett térnem az eredeti terveimtől az adatbázis tekintetében, mivel nem tudtam elég nagy magyar betűket tartalmazó adatbázis szerezni, meg tudtam valósítani terveimet egy másik, angol kézírásból szedett mintákat tartalmazó adatbázis segítségével.

Eleinte nagyon rossz eredményeket kaptam, de folyamatos teszteléssel és fejlesztéssel sikerült egy (megfelelő feltételek mellett) 80%-os pontosságú programot létrehoznom.

8 LEZÁRÁS

A program fejlesztése során betekintést nyerhettem a képfeldolgozó eljárásokat és mesterséges intelligenciát alkalmazó programok világába, aminek köszönhetően sokat tanultam a témában. Az optikai karakterfelismerés mindig is foglalkoztatott, így jó volt elmélyülni a témában.

Leginkább a képfeldolgozást igénylő részekkel kapcsolatos nehézségek megoldását élveztem. Jó volt kigondolni és megvalósítani az ékezetek megtartására szánt módszert, kitalálni, hogyan tudnám szétválasztani a sorokat, hogy ne keveredjenek össze a szavak. Még azon gondolatok megvalósítását is élveztem, amik végül zsákutcába vezettek, ilyen például a folyóírás szétválasztására fejlesztett programrészem.

Úgy érzem jól döntöttem, hogy a Python nyelvet használtam a fejlesztésre, nagy élmény volt számomra a környezet megismerése, a komponensek kitapasztalása.

Jó volt látni, ahogy bővült és fejlődött a programom, ahogyan egyre több és több részt tudtam összevonni, a kezdeti részenkénti fejlesztésből.

További terveim azok, hogy egy újabb szintre emelem az optikai karakterfelismeréssel kapcsolatos tudásom, és az elkészített program minden részét újra vizsgálom, egy még jobb megoldás keresésének céljával. Egyik oldalról, a mátrix alapú karakterfelismerési megközelítés helyett a görbületek és matematikai műveletek segítségével tervezem újra gondolni a meglévő megoldásaim. Másfelől pedig, elkezdek foglalkozni a szótár mesterséges intelligencia alapú újra gondolásával, a szövegkörnyezet figyelembevételének céljával a szemem előtt.

Szeretném a programomat a jövőben egy magyar adatbázisra megvalósítani, ezrét kutatásokat tervezek végezni a témában, hogy hogyan tudnék akár előállítani egy magyar írók kézírásából létrejött kézzel írott betűket tartalmazó adatbázist.

9 SUMMARY

Image processing, Python, and artificial intelligence development in Python enviroment was not part of my studies at the university, so I will write my evaluation from this point of view.

Since I had to enqure the solution of every image processing problem by myself, without any basic knowledge about the topic, my program is not so outstandingly optional and fault-tolerant as the ones used in the industry.

Despite all of that, with the utilisation of the suitable characteristics of various transformations, I was able to create a program which can process an image taken by a phone without mixing the letters, and keeping all the accents.

With a reliable image preprocessing method at hand, I was able to move on and focus on correcting the artificial intelligence. I decided Python was the most suitable programming language for my thesis. Although I had never worked with it before, I found it was the optimal programming environment for Optical Character Recignition development. As I expected, Tensoflow and Keras came in really handy when I worked on the artificial intelligence part of my thesis.

Although I had to differ from my original plans in the matter of the database, because I couldn’t find a big enough database containing Hungarian letters, I was able to execute my plans with the help of an English handwritten database.

I recieved very bad result at the beginning, but the continous tesing and development I managed to create a program, which (beside suitable contidions) works with 80% accuarcy.

10 IRODALOMJEGYZÉK

[1] Halmos Ferenc főszerkesztő: Pannon enciklopédia: a magyarság kézikönyve. *Pannon Könyvkiadó,* 1993

[2] Kiefer Ferenc: Magyar nyelv. *Akadémia Kiadó,* 2011

[3] MTA Helyesírási Bizottsága: A magyar helyesírás szabályai, tizenegyedik kiadás, második, változatlan lenyomat, *Akadémia Kiadó,* 1985

[4] Manoj Sonkusare and Narendra Sahu: Survey on handwritten character recognition. 2016, *India Advances in Vision Computing: An International Journal (AVC),* Vol.3, No.1, 2016 (https://airccse.org/journal/avc/vol3.html), utoljára megtekintve 2021.11.29.

[5] Láng Attila D.: Íráskalauz, (http://mek.oszk.hu/03100/03195/html/iras1.htm), utoljára megtekintve: 2021.04.27

[6] Várkonyi-Kecsmár Szilvia: Írás kialakulása, fejlődése, *Nemzeti Szakképzési és Felnőttképzési Intézet,* 2010

[7] Hargitai Henrik: Tipográfia (írás számítógéppel),

(http://mmi.elte.hu/szabadbolcseszet/mmi.elte.hu/szabadbolcseszet/index5b50.html?option=com_tanelem&id_tanelem=959&tip=0), utoljára megtekintve: 2021.04.27

[8] Szabó Szilvia: Normaírások a grafológiában, (http://www.azirastukreben.hu/normairasok-a-grafologiaban), utoljára megtekintve: 2021.04.27

[9] Chirag Patel, Atul Patel, PhD., Dharmanedra Patel: Optical Character Recognition by Open Source OCR Tool Tesseract: A Case Study. *International Journal of Computer Applications*, Vol. 55, No.10., 2012, pp. 50-56.

[10] Ray Smith: An Overview of the Tesseract OCR Engine. In proceedings of Document analysis and Recognition. *IEEE Ninth International Conference on Document Analysis and Recognition (ICDAR 2007),* Vol. 2, 2007, pp. 629-633.

[11] Thomas Deselaers, Tobias Grass, Georg Heigold; Hermann Ney: Latent Log-Linear Models for Handwritten Digit Classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* Vol.34, No. 6., 2012, pp.1105-1117.

[12]Line Eikvil: Optical Character Recognition. (http://home.nr.no/~eikvil/OCR.pdf), utoljára megtekintve: 2021.11.30

[13] Nagy György: At the Frontiers of OCR. *Proceeding of the IEEE*, Vol. 80, No. 7., 1992, pp. 1093-1100

[14] Faisal Shafait, Daniel Keysers, Thomas M. Breuel: Efficient Implementation of Local Adaptive Thresholding Techniques Using Integral Images. *Document Recognition and Retrieval XV part of S&T/SPIE Annual Symposium on Electronic Imaging*, Vol. 6815, 2008, (https://spie.org/Publications/Proceedings/Paper/10.1117/12.767755)

[15]Smith R.W.: The extraction and recognition of text from multimedia document images, (https://research-information.bris.ac.uk/ws/portalfiles/portal/34496373/380162.pdf), utoljára megtekintve: 2021.11.30

[16] Wada H., Takahashi S., Lijima T., Okumura Y., Imoto K.: An electronic reading machine. *Information Processing*, 1959, pp. 227-232.

[17] Grimsdale R. L, Sumner F. H, Tunis C. J, Kilburn T.: A system for the automatic recognition of patterns. *Proceedings of the IEE*, Vol. 106, 1959, pp. 210-221.

[18] M. Holstege and L. Tokuda: Visual parsing: An aid to text understanding. *Intelligent Text and Image Handling*, 1991, pp. 175-193.

[19] A. Dengel: Document image analysis-Expectation-driven text recognition. *Pre-proc. IAPR Workshop on Syntactic and Structural Pattern Recognition*, 1990, pp. 78-87.

[20] Kovács Róbert: Neurális hálózatok elrendezési és funkcionális típusai, (https://mesterin.hu/neuralis-halozatok-elrendezesi-es-funkcionalis-tipusai/) utoljára megtekintve: 2021.04.27

[21]Szabó Csilla: Kézzel írt szövegek feldolgozása tanuló algoritmusokkal, (https://adoc.pub/kezzel-irt-szvegek-feldolgozasa-tanulo-algoritmusokkal.html) utoljára megtekintve: 2021.04.27

[22] Premkumar Natarajan, Zhidong Lu, Richard Schwartz, Issam Bazziand and John Makhoul: Multilingual Machine Printed OCR. *International Journal of Pattern Recognition and Artificial Intelligence*, 2001, Vol. 15, No. 1., pp. 43-63.

[23] Gérard Swinnen: Tanuljunk meg programozni Python nyelven. *O'Rielly*, 2005

[24] Patrick J. Grother Scientific Contact: NIST special database 19, (http://doi.org/10.18434/T4H01C), utoljára megtekintve: 2021.11.28

[25] Gregory Cohen, Saeed Afshar, Jonathan Tapson, André van Schaik: EMNIST: an extension of MNIST to handwritten letters, (http://arxiv.org/abs/1702.05373), utoljára megtekintve: 2021.11.30

[26] Ivo Vynckier: How OCR Works: (https://how-ocr-works.com/index.html), utoljára megtekintve: 2021.11.14

[27] Peter Norvig: How to Write a Spelling Corrector, (http://norvig.com/spell-correct.html), utoljára megtekintve: 2021.11.14

[28] R. Fisher, S. Perkins, A. Walker and E. Wolfart: Connected Components Labeling (https://homepages.inf.ed.ac.uk/rbf/HIPR2/label.htm), utoljára megtekintve: 2021.11.14

[29] R. Fisher, S. Perkins, A. Walker and E. Wolfart: Adaptive Thresholding (https://homepages.inf.ed.ac.uk/rbf/HIPR2/adpthrsh.htm), utoljára megtekintve: 2021.11.14

11 ÁBRAJEGYZÉK