

Szegedi Tudományegyetem
Informatikai Intézet

SZAKDOLGOZAT

Varga Zoltán

2025

**Szegedi Tudományegyetem
Informatikai Intézet**

**Optikai karakterfelismerés konvolúciós neurális
hálózatokkal**

Szakdolgozat

Készítette:

Varga Zoltán
programtervező
informatikus BSc szakos
hallgató

Témavezető:

Dr. Palágyi Kálmán
egyetemi tanár

Szeged
2025

Feladatkiírás

Az optikai karakterfelismerés (OCR) évtizedek óta intenzíven kutatott területe a digitális képkeldolgozás, a számítógépes látás és a mesterséges intelligencia területeknek. Esetünkben a probléma nyomtatott szövegeket tartalmazó (szkennelt) digitális képek átalakítása (.txt) szövegfájlokká.

A kitűzött feladatban hangsúlyos a hangsúly az egyes karakterek felismerése (egy megfelelő méretű publikus adatbázison tanított) konvolúciós neurális hálózattal (CNN), így a bemeneti képek lehetnek olyanok, amelyeken a lehető legegyszerűbben megoldható a szövegképek sorokra és karakterekre tördelése.

Tartalmi összefoglaló

- **A téma megnevezése:**

Optikai karakterfelismerés konvolúciós neurális hálózatokkal.

- **A megadott feladat megfogalmazása:**

Egy olyan szoftver készítése, amelyet arra lehet használni, hogy egy képen lévő szöveget átalakítsunk számítógépes szöveggé.

- **A megoldási mód:**

A bemeneti képen első lépésként előfeldolgozást hajtok végre, ez 2 lépésből tevődik össze. Az első a kép binarizálása. Ezután a képről kitörölöm a nagyon kicsi egybefüggő részeket, amelyek általában szkennelési hibából kerülnek bele. Következő lépésként a képen lévő sorokat szegmentálom, majd kivágom a képből a sorokat. Ezután a sorokban lévő karaktereket bontom szét, és méretezem át azonos méretűre, és utána egy konvolúciós neurális hálózattal felismerem az egyes karaktereket, majd összefűzöm őket az eredeti sorrendjükben, és így kapom meg a kimeneti szöveget.

- **Alkalmazott eszközök, módszerek:**

A szoftvert Python 3.12.7 programozási nyelven készítettem. Felhasználásra kerültek a következő függvénykönyvtárak: textdistance, numpy, pathlib, cv2, matplotlib, seaborn, Levenshtein, sklearn, torch, torchvision, PIL. A karakterek felismerését egy VGG16 konvolúciós neurális hálóval végzem.

- **Elért eredmények:**

A kapott képből sikeresen állítok elő szöveget. A hálózat felismeri az angol ABC kis és nagy betűit, illetve a leggyakoribb írásjeleket, összesen 58 különböző karaktert. A bemeneti kép lehet szkennelt illetve képernyőkép is, a szoftver körülbelül 97%-os pontossággal dolgozik.

- **Kulcsszavak:**

OCR, karakterfelismerés, VGG16, neurális háló, konvolúciós háló

Tartalomjegyzék

Feladatkiírás.....	1
Tartalmi összefoglaló	2
BEVEZETÉS.....	5
1. OPTIKAI KARAKTERFELISMERÉS.....	5
1.1. Optikai karakterfelismerés felhasználása	5
1.2. Optikai karakterfelismerés története	6
1.2.1. Kezdeti próbálkozások	6
1.2.2. Modern karakterfelismerő rendszerek	6
1.3. Optikai karakterfelismerés módszerei	6
2. KONVOLÚCIÓS NEURÁLIS HÁLÓZATOK.....	8
2.1. Neurális hálózatok	8
2.1.1. Neuron	8
2.1.2. Neurális hálózat felépítése	8
2.2. Konvolúció képtérben.....	9
2.3. Konvolúciós neurális hálózatok képfeldolgozásban.....	10
2.3.1. Konvolúciós hálózatok felépítése	10
2.3.2. Konvolúciós hálózatok tanítása	10
2.3.3. Konvolúciós hálózatok kiértékelése.....	10
3. FELHASZNÁLT ESZKÖZÖK BEMUTATÁSA	11
3.1. Python programozási nyelv.....	11
3.2. Felhasznált Python könyvtárak	11
3.2.1. textdistance	11
3.2.2. numpy	11
3.2.3. pathlib	12
3.2.4. cv2	12
3.2.5. matplotlib.....	12

3.2.6. seaborn.....	12
3.2.7. Levenshtein.....	12
3.2.8. pytorch.....	12
3.2.9. torchvision	13
3.2.10. PIL.....	13
3.2.11. Scikit-learn	13
4. A SZOFTVER MŰKÖDÉSI ELVE.....	14
4.1. A szoftver bemenete.....	14
4.2. A kép binarizálása	14
4.3. Zajszűrés a képen.....	16
4.4. Sorok szegmentálása.....	16
4.5. Karakterek szegmentálása	18
4.6. Kivágott karakterek feldolgozása.....	19
4.7. Szóközők észlelése	20
4.8. Karakterek felismerése.....	20
4.9. Szöveg összefűzése.....	24
5. EREDMÉNYEK.....	25
5.1. Kiértékelési módszerek.....	25
6. A SZOFTVER HASZNÁLATA	29
IRODALOMJEGYZÉK.....	31

BEVEZETÉS

Napjainkban a digitalizáció korában napról napra egyre nagyobb az igény arra, hogy a dokumentumok digitális formában elérhetőek legyenek, nem csak képként, hanem szöveges, kereshető formátumban is. Ebben a folyamatban játszik nagy szerepet az optikai karakterfelismerés (OCR), amely lehetővé teszi képen lévő szövegek átalakítását szerkeszthető formátumba. Ezen technológiákat számos területen használják, többek között könyvtárak, levéltárak, múzeumok történelmi dokumentumok digitalizálásához, cégek számlák, szerződések automatikus feldolgozásához, de még olyan szoftverek is hasznát veszik, amelyre nem is gondolnánk, mint például a Google Translate egyik funkciója [9]. Szakdolgozatom célja, hogy bemutassam ezen technológiákat, melyben ki fogok térni a történetükre, bemutatom a működésüket, illetve a legmodernebb technikákon alapuló saját megoldást nyújtsak a problémakör megoldásaként.

1. Optikai karakterfelismerés

Az optikai karakter felismerő rendszerek feladata a nyomtatott vagy kézzel írt dokumentumokon lévő szöveg felismerése, és digitális, szerkeszthető formátumba alakítása. Ez teszi lehetővé azt, hogy ezekben a dokumentumokban keressünk, vagy szükség esetén szerkesszük azokat. Napjainkban képesek felismerni különböző karakterstílusokat, különböző méretű karaktereket, de akár még különböző nyelvek speciális karaktereit is, habár a hatékonyságuk nyelvről nyelvre változó lehet [8].

1.1. Optikai karakterfelismerés felhasználása

Karakter felismerő rendszereket számos területen használnak, mivel nagyon hatékonyan tudnak nagy mennyiségű adatot feldolgozni. Múzeumokban, levéltárakban, könyvtárakban használhatják régi iratok, könyvek digitalizálására. Egy müncheni cég ahelyett, hogy kézzel vinné fel a parkoló autók rendszámábláját, csak egy fotót készít az autóról, és egy szoftver felismeri a rendszámot, és felviszi azt a rendszerbe, megkönnyítve az alkalmazottak munkáját [9]. Egy Cambridge-i cég, amely vizsga szervezéssel foglalkozik, és évente több mint egy millió vizsgát kell kijavítania, hogy minden vizsgamunkát beszkennelnek és egy optikai karakter felismerő rendszer segítségével tudják rendszerezni a vizsgákat életkor, lakóhely, vizsga típus, és vizsga szint szerint [9]. A Google Translate egyik forradalmi újítása az volt, hogy a kamera segítségét hívta segítségül. Ezt úgy teszi meg, hogy a kamerát rá kell irányítani a

szövegre, a program felismeri a szöveget a képen, és utána lefordítja a kívánt nyelvre azt [9]. Ezen példákból is kiderül, hogy a karakter felismerő rendszerek milyen fontosak lehetnek akár a mindennapi életünkben is, és miért fontos ennek a területnek a kutatása.

1.2. Optikai karakterfelismerés története

A karakterfelismerő rendszerek készítése már régóta foglalkoztatja a tudósokat, habár eleinte a lehetőségek korlátozottak voltak. A számítógépek elterjedésével megnőtt a lehetőségek száma is, megnyílt az út az optikai karakterfelismerés előtt.

1.2.1. Kezdeti próbálkozások

A legelső karakter felismerő rendszert Emanuel Goldberg készítette el, amely karaktereket olvasott és azokat alakította át szabványos táviró kódokká [7]. Vele egyidejűleg készítette el az optofon nevű eszközt Edmund Fournier d'Albe, egy kézi eszközt amely a karakterekhez különböző hangokat rendelt [7].

1.2.2. Modern karakterfelismerő rendszerek

1974-ben Ray Kurzweil sikeresen készített egy olyan rendszert, amely gyakorlatilag bármilyen karakterstílust képes volt felismerni, és úgy döntött, hogy ebből egy felolvasó gépet fog készíteni vakok számára. Ez az eszköz tartalmazott egy szkennert, egy karakter felismerőt, és egy text-to-speech modult [7]. A 2000-es években megjelentek az első WebOCR rendszerek, amelyek nem lokálisan futottak a számítógépen, hanem egy API-on keresztül továbbításra kerültek a képek a szervernek, amely szöveges formátumban adta vissza a felismerés eredményét. Ezeket a rendszereket jellemzően olyan eszközökben használják, amelyeknek nincs beépített karakterfelismerési képessége [7].

1.3. Optikai karakterfelismerés módszerei

Mióta az optikai karakterfelismerés koncepciója felmerült, a kutatók számos módszert próbáltak meg felhasználni a probléma megoldására. A korai kutatások dimenziócsökkentő eljárásokat próbáltak meg használni, azért, hogy a számítógép könnyebben felismerje a karakter jellemző pontjait, ami alapján azután a karakter felismerhető [8]. Kelner és Glauberman egy vékony résen keresztül szkennelték be a betűket fentről lefelé, ezáltal egy dimenzióssá alakítva a képet, így csak egyszerű számításokat kellett végezzenek, hogy jellemzőket nyerhessenek ki a karakterekből.

Ezeknek a módszereknek a hátránya, hogy csak egy-egy betűtípusra működnek, és abból is csak kevés számú karaktert képesek felismerni [8].

Egy másik módszer esetében minden felismerhető karakternek van egy mintája a rendszerben eltárolva, és a felismerendő karaktereket az összes mintával összehasonlítják, és pontos egyezést keresnek. Ha ilyen nincs, akkor azt a karaktert adja ki a kimenetnek a program, amivel a legnagyobb egyezés létezik. Nagy hátránya, hogy csak akkor működik hatékonyan, hogyha az eltárolt mintázat, és a felismerendő karakterek ugyanabból, vagy nagyon hasonló karakterstílusból származnak [25].

A modern rendszerek esetében már neurális hálózatokat használnak. Ezek sem tökéletesek, de a modern fejlesztésekkel jobb és jobb eredményeket képesek elérni. Ezekből a hálózatokból és többféle rendszert kipróbáltak, többek között a Support Vector Machine-t (SVM), k Nearest Neighbor (kNN), Döntési fákat [8]. A számítógépek sebességének növekedésével megjelentek a konvolúciós neurális hálózatok is [8]. Ezeknek a rendszereknek hatalmas előnye, hogy képesek általánosítani a látott tanítóadatokról, illetve a tanító példák alapján önmaguk tanulják meg a különféle karakterekre jellemző formákat [3].

2. Konvolúciós neurális hálózatok

A mély tanulás lehetővé teszi azt, hogy a neurális háló modellek elvont jellemzőket tanuljanak meg a rendelkezésre álló adatokból [1]. A konvolúciós hálózatok képfeldolgozásra tervezett technológiák, melyek konvolúciós rétegek segítségével tanulják meg a képeken a mintázatokat. Ezek a technológiák áttörést hoztak a képfeldolgozási feladatokban, többek között például az arcfelismerésben és önvezető jármű technológiákban.

2.1. Neurális hálózatok

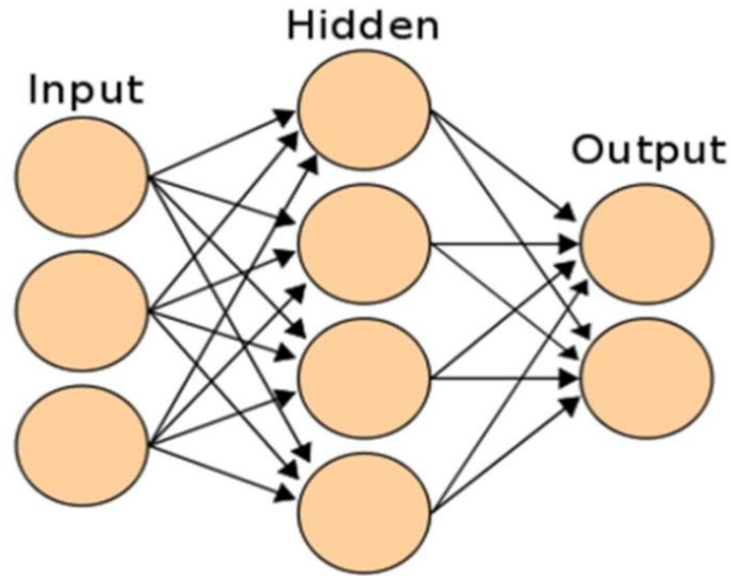
A neurális hálózatok a mesterséges intelligencia egyik ága, melynek különböző formái egyre nagyobb népszerűségnek örvendenek napjainkban [3]. Ezek a hálózatok mesterséges neuronokból álló rétegeken keresztül dolgozzák fel az adatokat. Ez teszi lehetővé, hogy a számítógépek tapasztalat alapján tanuljanak, általános viselkedést eredményezve [3]. Több dimenziós adatokra, például képekre általában mély konvolúciós hálózatokat használnak [3], ezeknek egy típusát használtam én is a munkám során.

2.1.1. Neuron

A neuron a hálózat legegyszerűbb felépítő eleme, mely a biológiai neuronokat próbálja imitálni az élőlények agyában [3]. Minden neuronnak bemenetei és kimenetei is vannak. Egy neuron bemenetei az előző réteg neuronjainak kimenetei. A neuron ezen bemenetek súlyozott összegét számolja ki, ehhez hozzáad egy eltolást, majd egy aktivációs függvény segítségével számolja ki a kimenetet [3].

2.1.2. Neurális hálózat felépítése

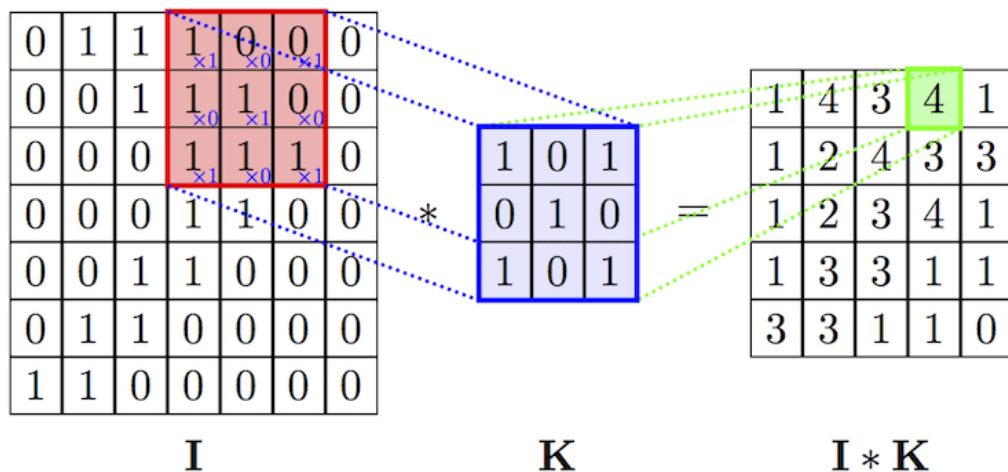
A neurális hálózatok általában egymásra épülő rétegekből épülnek fel. Tartalmaznak egy bemeneti réteget, tetszőleges számú rejtett réteget, illetve egy kimeneti réteget. A leggyakoribb neurális háló típus a teljesen összekapcsolt, azaz a fully connected hálózat, amelyben egy réteg összes neuronjának kimenete kapcsolódik a következő réteg összes neuronjának bemenetéhez [3]. Egy ilyen hálózat általános felépítése látható a 2.1-es ábrán.



2.1. ábra – Egy neurális háló általános felépítése [28]

2.2. Konvolúció képtérben

A konvolúció során egy kernelt, vagy más néven egy szűrőt csúsztatunk végig a kép összes pozícióján, és kiszámoljuk a szűrő és a képen a szűrő által lefedett terület súlyozott összegét. Ezáltal minden pozíción keletkezik egy új érték, és ezek együtt adják a kimeneti új képet, amely kiemeli a szűrő által keresett mintázatokat [4]. A 2.2-es ábrán egy konvolúciós művelet lépései láthatóak. Az 'I' mátrix a bemeneti kép, a 'K' mátrix a kernel, vagy szűrő, és az 'I * K' mátrix a végeredmény.



2.2. ábra – Konvolúció képtérben [29]

2.3. Konvolúciós neurális hálózatok képfeldolgozásban

A konvolúciós neurális hálózatokat számos képfeldolgozási feladat esetében alkalmazzák. A hálózatok képesek felismerni a képadatokban lévő jellemzőket, még akkor is ha a kép zajos. Ezen okokból kifolyólag a konvolúciós hálózatok áttörést jelentenek a képfeldolgozási feladatokban, például arcfelismerésben, vagy orvosi feladatokban [3].

2.3.1. Konvolúciós hálózatok felépítése

A konvolúciós hálózatok alapvető elemei a konvolúciós és a pooling rétegek, melyek után a hálózat végén általában teljesen összekapcsolt rétegek követnek. A konvolúciós rétegben nevéből adódóan egy konvolúció hajtódik végre a bemeneti képen, és így kimenetként egy szűrt képet kapunk [4]. A pooling réteg bemenetként megkapja ezt a képet, és lekicsinyíti a kép méretét, olyan módon, hogy közben megtartja a legfontosabb jellemzőket [1]. A hálózat végén néhány teljesen összekapcsolt réteg dönt arról, hogy a kép melyik osztályba tartozik [1].

2.3.2. Konvolúciós hálózatok tanítása

A konvolúciós hálózatokat nagy méretű adatbázisokkal tanítják, melyben a tanító példákat felcímkézik. A tanulási folyamat során a hálózat a bemeneti képhez ad egy kimeneti osztályt, majd az ismert címke alapján (az én feladatom esetében, hogy melyik karakterosztályba tartozik a kép) a hálózat számol egy hibát, és ezt visszavezeti a rétegeken, majd ennek megfelelően finomhangolja a súlyokat a neuronokban. Ennek a módszernek a segítségével iteratíván módosítja a súlyokat, és emiatt a hálózat egyre pontosabb eredményt lesz képes adni [1, 3].

2.3.3. Konvolúciós hálózatok kiértékelése

A hálózat tanítása után annak minőségét le is kell ellenőrizni, melyhez általában a tanító adattól teljesen független adatokat használnak. Az egyik legfontosabb mérőszám az accuracy, amely azt jelenti, hogy a hálózat a képek hány százalékát ismeri fel helyesen, számomra ez volt a legfontosabb mérőszám, erre támaszkodtam a leginkább [1].

3. Felhasznált eszközök bemutatása

A szoftver elkészítése, és megfelelő működése szempontjából kiemelkedő fontosságú a célnak megfelelő eszközök választása, mely lehetővé teszi a rendszerek elkészítését, ugyanakkor gyors és hatékony futást tesznek lehetővé. Ebben a fejezetben ezeket az eszközöket fogom bemutatni.

3.1. Python programozási nyelv

A szoftvert Python 3.12.7 programozási nyelven készítettem el. A Python egy felhasználóbarát, nyílt forráskódú, egyszerűen tanulható gyengén típusos szkriptnyelv, mely hatalmas közösségi támogatással rendelkezik, ezért elérhetőek benne függvénykönyvtárak rengeteg különféle problémára. A Pythont számos területen alkalmazzák, többek között webfejlesztésre, oktatásra, tudományos feladatokra, de emellett gyakran használt eszköz képfeldolgozási és mesterséges intelligencia feladatokhoz [13].

3.2. Felhasznált Python könyvtárak

A Python egyik legnagyobb előnye, hogy rengeteg különböző probléma csoportra készítettek benne függvénykönyvtárakat, melyek a csomagkezelője révén egyszerűen letölthetőek és importálás után használhatóak a szoftverünkben.

3.2.1. textdistance

A Textdistance egy listák összehasonlítására létrehozott könyvtár. Több mint 30 féle különböző algoritmust tartalmaz, melyből számomra a legfontosabb a Levenshtein távolság [14].

3.2.2. numpy

A numpy az egyik legalapabb tudományos függvénykönyvtár, mely leginkább matematikai műveletek implementációját tartalmazza. Hatékony tömb algoritmusokat tartalmaz. Lehetőség van benne többdimenziós tömbök létrehozására és hatékony kezelésére [16].

3.2.3. pathlib

Egy olyan függvénykönyvtár mely lehetőséget biztosít fájlrendszer-elérési útvonalak egyszerű és hatékony kezelésére, és ezáltal elkerülhetőek az os.path műveletek használata [17].

3.2.4. cv2

Számos CPU-only függvényt biztosít képfeldolgozási illetve gépi látási hatékony elvégzéséhez [18]. Az összes képfeldolgozási részfeladatot ezen könyvtár használatával végeztem el.

3.2.5. matplotlib

Egy adatvizualizációs eszköz, mely statikus képes képeket, animációkat és interaktív ábrákat képes létrehozni, illetve megjeleníteni publikációs minőségben [19]. Elsősorban részeredmények megjelenítésére használtam.

3.2.6. seaborn

Egy Matplotlib alapú függvénykönyvtár mely képes statisztikai adatvizualizációra [20]. A végeredmények megjelenítésére használtam

3.2.7. Levenshtein

A Levenshtein egy a nevéből adódó módon a Levenshtein-távolság és a módosítások kiszámítására létrehozott C-ben írt függvénykönyvtár. Ezzel a könyvtárral stringeket lehet összehasonlítani, és az egymásba transzformáláshoz szükséges adatokat kiszámolni [15].

3.2.8. pytorch

A pytorch egy GPU gyorsított tenzorműveleteket és mély neurális hálózatokat megvalósító csomag. Integrálható egyéb népszerű csomagokkal, hogy megnövelje. Ezeken kívül számomra a legfontosabb tulajdonsága az volt, hogy támogatja az NVIDIA videokártyákon a CUDA technológiát amely nagyban megnöveli a modell tanítási sebességét [22, 27].

3.2.9. torchvision

A pytorch-hoz készült kiegészítő csomag mely képfeldolgozáshoz adatkészleteket, neurális háló modelleket, és néhány általános képi transzformációt tartalmaz [23].

3.2.10. PIL

Egy képfeldolgozó függvénykönyvtár, mely rengeteg fájlformátumot támogat és hatékony képadat kezelést valósít meg [24].

3.2.11. Scikit-learn

A scikit-learn egy nyílt forráskódu gépi tanulási függvénykönyvtár, és algoritmusokat kínál különféle gépi tanulási feladatokhoz. Számomra a legfontosabb részei a kiértékeléshez köthetőek [21].

4. A szoftver működése

A feladat nehézsége miatt fontos, hogy részekre tudjuk bontani a problémát. Mivel egy szöveg az egy strukturált dokumentum, mely bekezdésekből és azokon belül sorokból épül fel, ezért elemi fontosságú, hogy a képet fel tudjuk bontani elemeire. A sorokat ezután karakterekre kell törni, és csak ezután lehetséges felismerni azokat, és összefűzni őket, hogy megkapjuk a végeredményt.

4.1. A szoftver bemenete

A program bemenetként egy képet vár, amellyel kapcsolatban a probléma egyszerűsítése érdekében néhány megkötést tettem, melyek a következők:

- A képen fekete nyomtatott folyó szöveg található fehér papíron
- A képen lévő szöveg vízszintes, vagy csak nagyon kicsi az eltérés attól
- A kép nem tartalmaz szkennelési hibáknál nagyobb zajt (pl. kávéfoltokat)
- A kép megvilágítása az egész dokumentumon közel azonos
- A kép megfelelő felbontással lett szkennelve, jellemzően 300 dpi fölött
- A képen lévő szöveg minimum 12 pont méretű

A bemenet egyaránt lehet képernyőkép és szkennelt kép is. Ezek a megkötések kritikusak a szoftver megfelelő működéséhez, a működési elvből adódóan.

4.2. A kép binarizálása

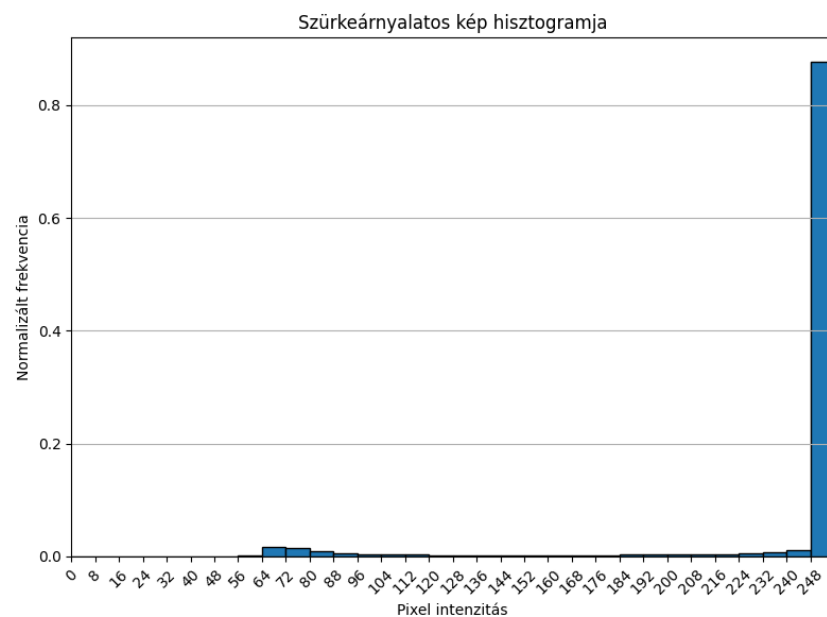
Mivel a program nem feltételezi, hogy a bemeneti kép bináris, ezért a legelső lépésként a bejövő szürkeárnyaltos képet binarizálni kell. Több módszert is kipróbáltam, melyek egy globális küszöbölés előre megadott értékkel, egy adaptív küszöbölés és egy OTSU küszöbölés voltak [6, 10].

A globális küszöbölés egy olyan eljárás, amely a kép minden pixel értékét kicseréli 255-re (fehérre) vagy 0-ra (feketére) attól függően, hogy a pixel értéke egy adott határpont felett vagy alatt van-e [6, 10]. Ennek az eredménye a 4.2-es ábrán látható.

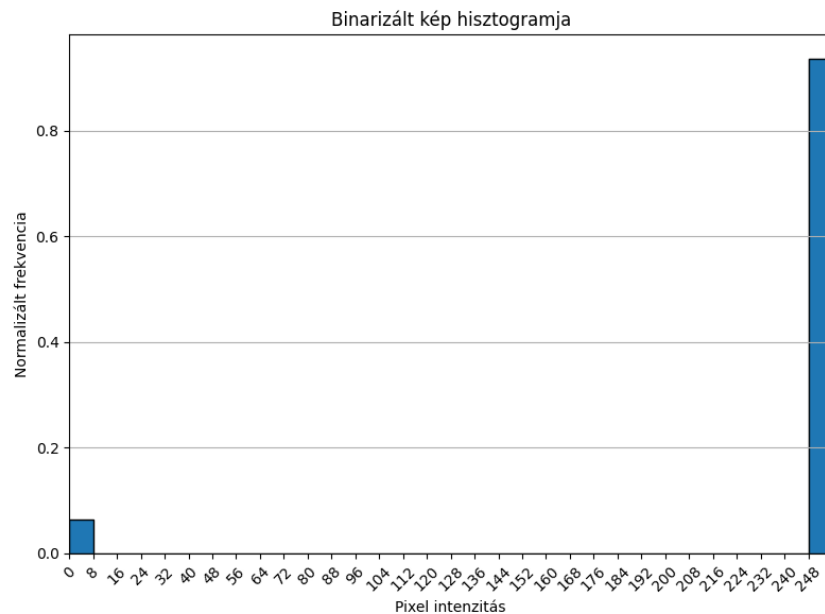
Az adaptív küszöbölés ezzel ellentétben, nevéből adódóan nem az egész képen dolgozik egyszerre, hanem egy megfelelően kicsi régió. Ehhez a régióhoz meghatározza a megfelelő küszöbértéket, és azt alkalmazza, hasonlóan a globális küszöböléshez. Ezt a folyamatot ismétli meg az egész képen [10].

Az OTSU módszer a kép hisztogramjának (4.1-es ábra) segítségével állapítja meg a legoptimálisabb küszöbértéket, mellyel utána egy globális küszöbölést hajt végre [10].

A három módszerből kettő nem adott számomra megfelelő eredményt, mivel mind az adaptív, mind az OTSU módszer esetében a küszöbölés hatására több karakter is összefolyt a képen, amely nagyban megnehezítené a sorok karakterekre bontását, úgyhogy ezt a kettő módszert elvettem. A globális küszöbölés ugyanakkor nem követte el ezt a hibát, mely abból adódhat, hogy a bemeneti kép a megkötések miatt fehér háttéren fekete karaktereket tartalmaz, ezért a kép hisztogramján kettő keskeny csúcs található, amely a 4.1-es ábrán is látható. Az egyik csúcs a karakterek képe, a másik a háttéré, és közöttük viszonylag kevés érték található, ezért középen nagyon könnyen elvágható a hisztogram.



4.1. ábra – Egy szürkeárnyaltos kép hisztogramja binarizálás előtt



4.2. ábra – Egy kép histogramja binarizálás után.

4.3. Zajsűrés a képen

Abból kifolyólag, hogy a bemenet nem csak képernyőkép lehet, hanem szkennelt dokumentum is, előfordulhat, hogy szkennelési hibák, zajok vannak a képen. Ezeket a hibákat a megfelelő működés érdekében javítani kell.

Az előző lépésben a binarizálásnál az egyszerű zajokat már kiküszöböltem, de nagyobb szkennelési hibák még mindig lehetnek a képen. Ezek a hibák úgy néznek ki, hogy fekete alapon fehér foltok, vagy fehér alapon fekete foltok jelenhetnek meg, melyek a valóságban nincsenek ott. Ezeket úgy kerülnek javításra, hogy a kép invertáltján (tehát a háttér fekete, a karakterek fehérek) megkeresem az összefüggő komponenseket, és megszámlolom azoknak a méretét. Amennyiben azok kisebbek egy előre meghatározott értéknél akkor törölöm az elemet a képről. Ez a méret tapasztalati úton 5 pixel lett, ha ennél kisebb, akkor sok hibát nem törölt a program, ha viszont ennél többre lett állítva, akkor pedig olyan elemeket is kitörölt, amelyeket nem kellett volna, például mondatvégi pont, ékezetek, stb.

4.4. Sorok szegmentálása

Következő lépésként a képet sorokra kell törni. Ehhez szintén több módszert is alkalmaztam. Az elsőnek a működési elve roppant egyszerű. A képnek kiszámolom a

vízszintes vetületét, és ebben megkeresem a lokális minimumpontokat. Ahol van egy ilyen minimum pont, ott van egy sortörés. Ennek az eredménye látható a 4.3-as ábrán.

Lorem, ipsum dolor sit amet consectetur adipisicing elit. Recusandae aut quisquam eaque aspernatur, odit iusto nulla consequuntur nihil eveniet harum, quia dicta? Adipisci, amet dicta pariatur natus dolor omnis accusantium reprehenderit minus eius sint id quod quas quos distinctio ipsam rerum ea officiis laborum corporis laudantium aliquid saepe? Consequuntur aspernatur obcaecati ipsam inventore nihil neque, odit sit recusandae culpa similique adipisci iusto in excepturi voluptatum totam ex iste harum sunt a? Sunt, iusto? Temporibus maiores pariatur molestias iste, nemo facere autem dicta eveniet est incidunt optio repellat doloremque recusandae porro nulla, at vitae minima cupiditate hic. Ea labore deleniti non nesciunt, est, eaque facere laborum enim inventore aliquid saepe vel, nobis ipsum cumque. Dignissimos quia

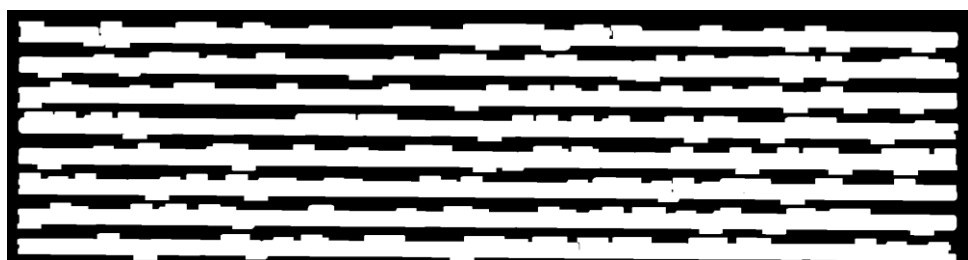
4.3. ábra – A felismert sortörések.

Ez a módszer, egyszerűségének ellenére nem bizonyult megfelelőnek, mivel feltételezi, hogy a sorok vetülete nem ér össze, azaz a sorok vízszintesek, és már nagyon kicsi eltérés esetén sem tudja megkülönböztetni egymástól a sorokat, gyakran fordult elő az, hogy többet észlelt egynek. Ez a jelenség látható a 4.4-es ábrán, melyen 4 darab sor látható, amit a program egy sornak ismert fel.

dolor non quidem illum. Consectetur in alias pariatur voluptas sed minima dignissimos asperiores ipsa recusandae commodi reiciendis cum dolore aperiam corrupti adipisci, sit illo rem, mollitia ipsum voluptates magnam provident molestiae? At expedita nesciunt modi porro provident, deserunt, nobis dolorum repudiandae magnam quas praesentium quis natus tenetur

4.4. ábra – A program nem tudta szétbontani a sorokat

Ennél egy jobb módszernek bizonyult az, hogy a kép invertáltján egy morfológiai dilatációt hajtok végre egy erősen téglalap alakú kernellel, amivel azt érem el, hogy a sorokon belül a karakterek összelógnak, és így a képen egy fehér régió az teljesen lefed egy sort. Ennek a végeredménye látható a 4.5-ös ábrán.



4.5. ábra – A sorokban lévő betűk összevonva, hogy lefedjenek egy-egy sort.

Ezután a képen megkeresi az összefüggő komponenseket, azaz a sorokat, és ezekhez a komponensekhez készít egy-egy maszkot. Egy ilyen maszk látható a 4.6-os ábrán. Ezzel a maszkkal vágja ki a sort a képből. Ezeket a kivágott képeket végül becsomagolja egy objektumba, és ezen objektumokból készült listát ad vissza a függvény.



4.6. ábra – Egy maszk, ami pontosan lefed egy sort az eredeti képen.

4.5. Karakterek szegmentálása

A képnek kiszámolom a függőleges vetületét, és ebben megkeresem a lokális minimumpontokat. Ahol van egy ilyen minimum pont, ott van egy új karakter. Ez látható a 4.7-es ábrán.

Lo|re|m, |i|p|s|u|m |d|o|l|l|o|r |s|i|t |a|m|e|t |c|o|n|s|e|c|t|e|t|u|r

4.7. ábra – Egy sorban a felismert karaktereknek a határvonalai

Ez a módszer, a sorokra tördeléssel ellentétben, habár gyakorlatilag ugyanúgy működik, itt megfelelőnek bizonyult, ugyanis egy-egy karakter között viszonylag nagy a távolság azok méretéhez képest, ezért nagyon nagy dőlés kéne ahhoz, hogy a két karakter vetülete átfedje egymást. Ezután kivágom a képből a karaktereket azok befoglaló téglalapja mentén, és egy objektumba csomagolom őket.

Ritka esetben előfordulhat, hogy két karakter vetülete átfedi egymást, és a program ilyen módon nem tudja szétválasztani őket, ezeket külön fel kell ismerni, és kezelni kell. A 4.8-as ábrán egy olyan kép látható, amelyen a program a betűket nem tudta megfelelően szétválasztani. Ez azért történt meg, mivel a vízszintes vetülete a két karakternek átfedi egymást, ezért nem találja meg a program a két karakter közötti üres részt.

Te

4.8. ábra – Egy rosszul felismert karakter

A javításához a szegmentáló modul feltételezi, hogy ha kettő karakter van a képen, akkor a vízszintes és függőleges vetület is egy darab egybefüggő terület. Ezt azért lehet megtenni, mivel a bemeneten a szöveg mindenképp vízszintes kell legyen az előzetes megkötések miatt, ezért a vízszintes vetületük egyértelmű módon átfedi egymást, a függőleges vetület esetében pedig azért lehet feltételezni, mivel ha egy karakter van a képen, akkor magától értetődő módon nem lesz több különálló rész a vetületen, ha pedig több, akkor az azért történik meg, mivel összelógott a vetületük, és emiatt nem sikerült szétbontani a két karaktert. Ha a vízszintes és függőleges vetületen is pontosan 1-1 darab összefüggő terület van a háttért nem számítva, de a képen legalább kettő összefüggő, egymástól független terület van, akkor tudja a program, hogy legalább két karakter van egy képen, és ezt szét tudja vágni, ugyanis attól, hogy a vetületük átfedi egymást, a két terület lehet egymástól teljesen független. A kivágáshoz pedig egyszerűen megkeresi az összefüggő területek befoglaló téglalapját, és kivágja azt. Ennek az eredménye látható a 4.9-es ábrán.



4.9. ábra – A rosszul felismert karakterek befoglaló téglalapja

Ez a módszer azért megfelelő, mivel a több részből álló karaktereket (például ékezetes betűk, felkiáltó jel, stb.) nem vágja szét, ugyanis azoknak a vízszintes vetületük legalább 2 különálló részből áll, ezért ez a programrész nem jelez be rá, hogy több karakter van a képen, csak akkor, ha tényleg fennáll a hiba.

4.6. Kivágott karakterek feldolgozása

Ezután, hogy megvannak a karakterek képei, ezeket következő lépésként fel kell dolgozni, mivel ezek még nem alkalmasak arra, hogy egy konvolúciós neurális háló megfelelő módon felismerje őket, ehhez ugyanis azonos méretűnek kell, hogy legyenek. A célméret ebben az esetben 64x64 ugyanis a felismerő modulban a konvolúciós háló ekkora méretű képeket fogad. Ahhoz, hogy a felismerés megfelelő eredményt adjon, úgy kell átméretezni a karaktereket, hogy az egymáshoz képest lévő méretaránybeli különbségeket megtartsák. Ezt úgy teszem meg, hogy egy előre megadott célméretre skálázom a legnagyobb karaktert, amely tapasztalati úton jelen esetben 45 pixel, és minden más karaktert ugyanekkora arányban skálázok, kivéve ha kisebb méretű

maradna 15 pixelnél, akkor minimum ekkora méretű lesz a végeredmény. Ezután az átskálázott karakternek keretet adok, hogy háttérrel együtt 64x64-es méretű legyen, és készen álljon arra, hogy a konvolúciós háló felismerje őket.

4.7. Szóközök észlelése

A szöveg struktúrája és olvashatósága szempontjából az egyik legfontosabb rész a szóköz, ezért ennek az észlelése kritikus a szoftver megfelelő működéséhez. Ehhez egy nagyon egyszerű módszert alkalmaztam. A sorszintű szegmentálás végső lépéseként az összes sorra pixel szinten átlagolom, hogy a fekete karakterek után mekkora üres rész található. A karakterszintű szegmentáláskor kivágom a sorokból a karakter képét, de mivel ezt úgy teszem meg, hogy a karakter bal szélétől vágom ki a következő karakter bal széléig, ezért garantáltan a kivágott kép bal oldalán lesz a lényegi információ, és jobb oldalt lesz egy üres fehér rész (amennyiben nem a dokumentum legszélén van). Ennek a fehér résznek kiszámolom a szélességét, és ha ez kisebb mint az egész dokumentumon a karakterek után lévő üres rész átlaga, akkor nem teszek szóközt, ha pedig nagyobb mint az átlag akkor beillesztek egy szóközt. Még abban az esetben szükséges egy szóközt beszúrni, ha a sor utolsó karakterét vizsgálom, mivel annak a jobb szélén nem található üres terület, mert azt nem vágja ki.

4.8. Karakterek felismerése

A karakterek felismerését egy konvolúciós neurális hálóval végeztem el. Első lépésként ehhez tanító adatot kellett keressek, vagy létrehozak. Mivel úgy döntöttem, hogy csak az angol ABC kis és nagybetűit próbálom felismerni, illetve néhány írásjelet, amelyek mind része az ASCII kódtáblának, ezért ehhez megfelelő tanító adathalmazt kerestem. A felhasznált adathalmaz egy 94 osztályból álló gyűjtemény, mely azonos méretű, 64x64-es szürkeárnyaltos képeket tartalmaz az összes nyomtatható ASCII karakterről, nagyságrendileg minden osztályból 200-at, összesen majdnem 24000 képet [11]. A 4.10-es ábrán látható egy részlet a tanító adatokból. Jól látható, hogy a különböző képeken a karakterek ugyanakkorák, nincs közöttük méretbeli különbség. A megfelelő tanításhoz nekem ezeket a képeket módosítani kellett. Ehhez kettő lépést hajtottam végre.



4.10. ábra – Egy részlet az eredeti tanító adatból.

Az első lépés a kép betöltése után, egy binarizálás volt. Ez a lépés azért fontos, mivel a felismerés első lépése is egy binarizálás, ezért a neurális háló nem kell, hogy tudjon felismerni szürkeárnyaltos képeket, csak fekete-fehér képekkel dolgozik.

A második lépés ennél bonyolultabb, ugyanis a tanítóhalmazban minden karakter majdnem kitölti az egész képet, méretben nem tesz különbséget két karakter között, emiatt a neurális háló sem tud méretbeli különbségeket megtanulni. Ennek kiküszöbölésére azt a megoldást találtam, hogy a tanító képről kivágom a karaktert, és átmeretezem azt, a következő módon:

- Ha a karakter az angol ABC kisbetűje és ugyanúgy néz ki kisbetűként mint nagybetűként (például 'x', 'c', 'o', stb.) akkor a karaktert 15 és 35 pixel közé méretezem 2 pixel eltéréssel, azaz készítek belőle egy 15, 17, 19...31, 33, 35 pixel méretű képet.
- Ha a karakter az angol ABC nagybetűje és ugyanúgy néz ki nagybetűként mint kisbetűként (például 'X', 'C', 'O', stb.) akkor a karaktert 36 és 50 pixel közé méretezem 2 pixel eltéréssel, azaz készítek belőle egy 36, 38, ... 48, 50 pixel méretű képet.
- Egyéb esetben hasonló módszerrel 15 és 45 pixel között méretezem át 3 pixelenként, azaz 15, 18, 21 ... 39, 42, 45 pixel méretű képet készítek belőle.

Ennek a módosításnak az eredménye látható a 4.11-es ábrán, melyen jól látható, hogy ugyanazon betűtípusok esetén a karakterek több méretben is megtalálhatóak a tanítóhalmazban.



4.11. ábra – Egy részlet a módosított tanító adatokból.

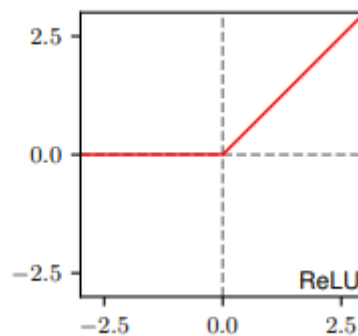
Ezeket a képeket utána egy fehér kerettel kiegészítem, hogy 64x64-esek maradjanak, mivel a neurális háló modell ekkora input képpel fog dolgozni. Ezen feldolgozási lépések után azelőtte is tekintélyes méretű ~24.000 darab képből körülbelül ~250.000 darab kép keletkezett. Ennek előnye az, hogy a háló sokkal több példából tudja megtanulni az egyes osztályokra jellemző tulajdonságokat, ugyanakkor nagyon meghosszabbítja a tanulási folyamatot, 1-2 perc helyett körülbelül 0,5-1 óra alatt sikerült betanítani a neurális hálót. Ezeket a képeket 80-20 arányban bontottam fel

tanító és teszt adathalmazra, minden 5. képet vettem ki tesztelésre, így minden karakterstílusból mindenféle méretben kerültek képek mind a tanító és tesztadathalmazba is.

A felismerő modulhoz kettő modellt teszteltem le, egy nagyon egyszerű sajátot, illetve egy state-of-the-art modellt, a VGG16-ot [5]. Mind a kettő konvolúciós neurális háló, 64x64-es fekete-fehér képekkel dolgoznak, 58 kimeneti osztállyal.

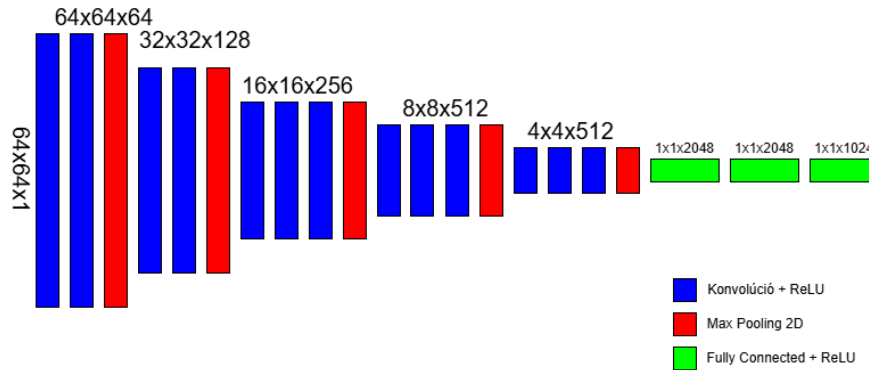
A saját modellem esetében a bemeneti kép három darab konvolúciós rétegen ment keresztül, ami után egy maxpooling végrehajtása következett és kettő darab lineáris réteg, ami után egy softmax aktivációs függvénnyel 58 osztályba sorolta a képeket. A tanítás során kipróbáltam számos beállítási lehetőséget, leginkább a learning rate és a dropout [5] értékének módosításának hatásait teszteltem. Sajnálatos módon, habár a tesztadaton remek eredményt produkált a modell, 99% felett, a saját valós életbeli példákon rendszerint sok hibát vétett, kisbetűket összekeverte a nagybetűs párjaikkal, a hasonló karaktereket rendszeresen összekeverte. Ezt többféle tanítási beállítással is kipróbáltam, de valamilyen ezekhez hasonló kirívó hibát mindig produkált.

Ennek elkerülése végett a saját készítésű modellt lecseréltem egy state-of-the-art modellre, a VGG16-ra. Ez a modell egy eredetileg ILSVRC klasszifikációhoz létrehozott modell, amely 224x224-es képeket sorol 1000 osztályba [5]. A modell 3x3-as konvolúciós rétegekből épül fel, melyek ReLU aktivációs függvényt használnak. Ez a függvény látható a 4.12-es ábrán.



4.12. ábra – ReLU aktivációs függvény [2]

A konvolúciós rétegek után 2x2-es, 2-es lépésközű MaxPooling fut le, majd a modell végén 3 darab fully-connected lineáris réteg van aminek a legvégén megszületik a kimenet [5]. Mivel én nem erre fogom felhasználni a modellt, ezért ezeken az értékeken is változtattam. Az én esetemben 64x64-es képeket sorol be 58 különböző osztályba. Ez látható a 4.13-as ábrán.



4.13. ábra – A módosított VGG16 neurális háló modell felépítése.

A tanításhoz stochastic gradient descent (SGD) algoritmust választottam keresztentropia veszteségfüggvénnyel [1]. Az SGD előnye, hogy nem az egész tanítóhalmazon dolgozik egyszerre, hanem mindig csak egy kevés adattal, ezért a rendszerben nagyobb a véletlenszerűség, nehezebben akad el lokális minimumpontokban, de pont ezért nehezebben is éri el a globális minimumot [12]. Ennek kiküszöbölése érdekében azt a módszert használom, hogyha a veszteségfüggvény értéke (amely azt mondja meg, hogy egy iterációban a modell mennyire volt pontos. Akkor jó, ha az értéke kicsi.) két iteráció között nem változik elég nagy mértékben, a learning rate-t elosztom 10-zel, mivel így a modell sokkal kisebb lépésekben tud konvergálni a globális minimumhoz, nem fog „ugrálni”. A tanítást 4 iteráción keresztül végeztem, 0.005-ös learning rate, 0.005-ös weight decay, és 0.08-as momentum mellett. A batch méret 128, és a képek sorrendje véletlenszerűen volt megkeverve [5]. Ezeket az értékeket tapasztalati úton sikerült meghatározni, mivel így már kellőképpen jó eredményt adott a neurális háló mind a teszt halmazon azaz 99% felett, mind a saját valós életbeli adataimon, ami körülbelül 97%, de még nem volt nagyon hosszú a tanítási idő. A tanítási folyamat eredménye a 4.14-es ábrán látható.

```
PS D:\egyetem\Szakdolgozat\src\network> python .\neural_network.py
cuda
08:49:21: Loading ...
08:49:46: Loading finished ...
08:49:46: Separating train/test ...
08:49:46: Separating train/test finished...
08:49:46: Creating data loaders ...
08:49:46: Start training ...
08:58:32: Epoch 1, Loss: 0.6638195686226368
09:07:05: Epoch 2, Loss: 0.01489703889887532
09:15:55: Epoch 3, Loss: 0.010910850393986303
09:24:49: Epoch 4, Loss: 0.010065067414971651
09:26:28: Accuracy: 99.94383743490248%
PS D:\egyetem\Szakdolgozat\src\network>
```

4.14. ábra – A tanítási folyamat eredménye

4.9. Szöveg összefűzése

Utolsó lépésként, miután felbontottuk a képet karakterekre, és észleltük a szóközöket, csak annyi maradt, hogy a felismert karaktereket összefűzzük egy szöveggé. Ha ezt megtesszük, akkor megkapjuk az eredményt, az eredeti bemeneti képen lévő szöveget szerkeszthető, kereshető formátumban. A szoftver az eredményt egy szöveges dokumentumba menti el a megadott elérési útvonalra.

5. Eredmények

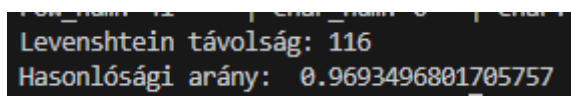
Ahhoz, hogy a szoftver működését értékelni tudjuk, az eredményeket fel kell dolgozni. Ehhez szükségünk van nem csak a bemeneti képre, hanem az azon lévő szövegre is, mint ismert adat, majd összehasonlítani különböző módszerekkel a kimeneti szöveggel, hogy megkaphassuk a szoftver pontosságát.

5.1. Kiértékelési módszerek

A kiértékeléshez több módszert is alkalmaztam, hogy a lehető legpontosabb képet kaphassak az elért eredményekről.

Az egyik módszer egy viszonylag egyszerű koncepción alapul, az eredeti ismert szöveget karakterenként összehasonlítja a kimeneti szöveggel. Ennél az a probléma merülhet fel, hogy a kimeneti szövegben esetleg 1-1 betűt nem ismert fel a program, vagy kimaradt egy szóköz, vagy esetleg egynél több van belőlük, ezért a hiba pontjától kezdve az egész szöveg el van csúszva, így nem fogja megtalálni a hasonlóságokat a program. Ennek megoldására a Levenshtein függvénykönyvtárat használtam fel [15]. A Levenshtein távolság azt jelenti, hogy az egyik stringből hány módosítással lehet eljutni a másik stringbe. Az editops függvény megadta, hogy milyen módosításokat kell végrehajtani ahhoz, hogy a kettő string a lehető legnagyobb részen illeszkedjen, és ugyanolyan hosszúak legyenek. Ezek a műveletek a replace, azaz egy karakter cseréje, a delete, azaz egy karakter törlése, és az insert, azaz egy karakter beillesztése lehet. Ezt a függvényt futtattam le a bemeneti ismert stringre, amely pontosan megegyezik a képen látható szöveggel, és a párjára, amelyet a szoftver kimenetként adott. Ennek a lépésnek hála a kimeneti string mostmár egyező hosszúságú az eredeti stringgel, ezért lehet rajta futtatni karakterenkénti összehasonlítást.

Kettő féle összehasonlítást végeztem el. Az egyik egy nagyon egyszerű karakterenkénti összehasonlítás az egész szövegen, melynek eredménye egy darab szám érték, ami azt jelzi, hogy a kettő szöveg arányaiban mennyire egyezik meg. Ez az eredmény teszt képen ~97% lett, ami az 5.1-es ábrán is látható.

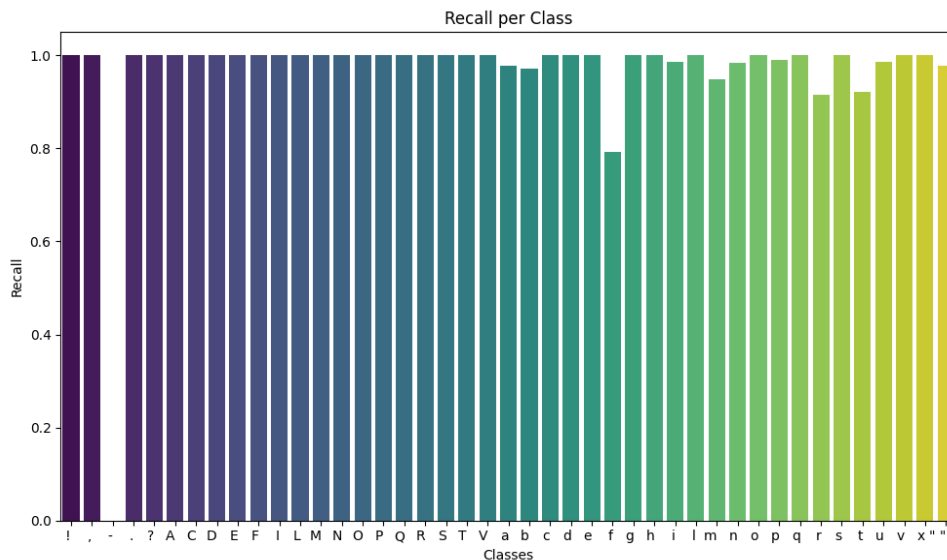


```
Levenshtein távolság: 116
Hasonlósági arány: 0.9693496801705757
```

5.1. ábra – Az összehasonlítás eredménye


$$Recall = TP / (TP + FN)$$

Ez a képlet azt adja meg, hogy ha az eredeti inputban egy karakter mondjuk az 'A' osztályba tartozott, akkor a kimenetben mekkora arányban tartozott szintén az 'A' osztályba. Az 5.3-es ábrán az osztályonkénti recall érték látható egy teszt inputon futtatva.

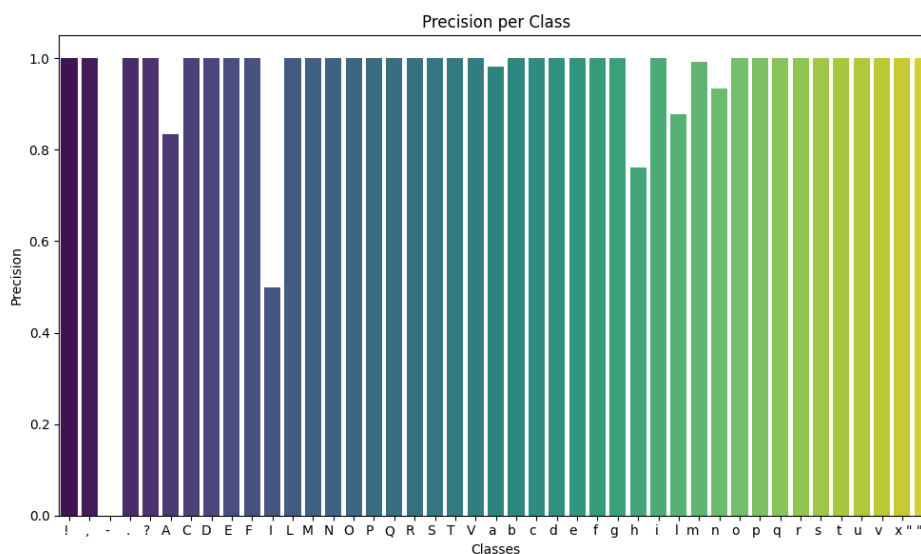


5.3. ábra – Recall érték osztályonként

Egy másik hasonló mérőszám a precision. A precisiót a következő módon lehet kiszámolni [26]:

$$Precision = TP / (TP + FP)$$

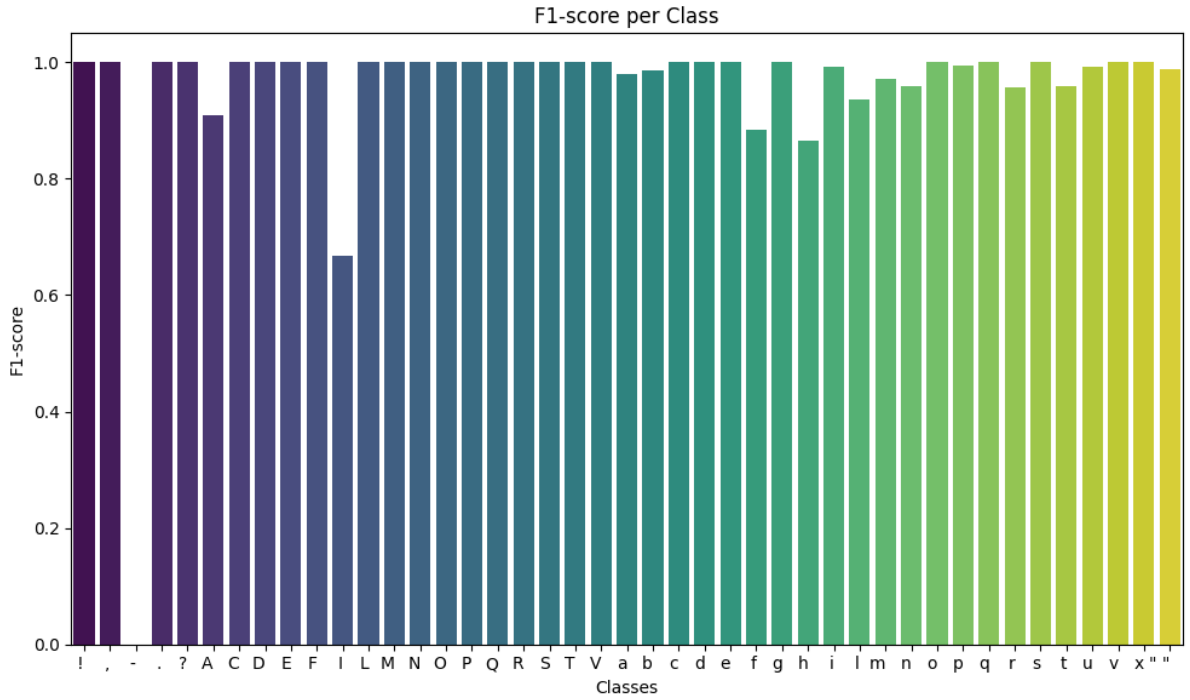
Ez a mérőszám azt mondja meg, hogy a kimenetben egy elemre azt jósolta a modell, hogy egy osztályba tartozik, az mekkora arányban tartozik ténylegesen abba az osztályba. Az 5.4-os ábrán a precision értékek láthatóak egy teszt inputon futtatva.



5.4. ábra – Precision érték osztályonként

A következő mérőszám amit használtam az az F1-score, más néven F-score, F-measure. Ez a recall és a precision hibáját hivatott kijavítani, kiegyenlíti a kettőt. Ez abban az esetben fontos, hogyha a tanítóadatok osztályai nem kiegyensúlyozottak. Az F1-értékek az 5.5-ös ábrán láthatóak. Kiszámolási módja a következő [26]:

$$F\text{-score} = (2 * Precision * Recall) / (precision + recall)$$

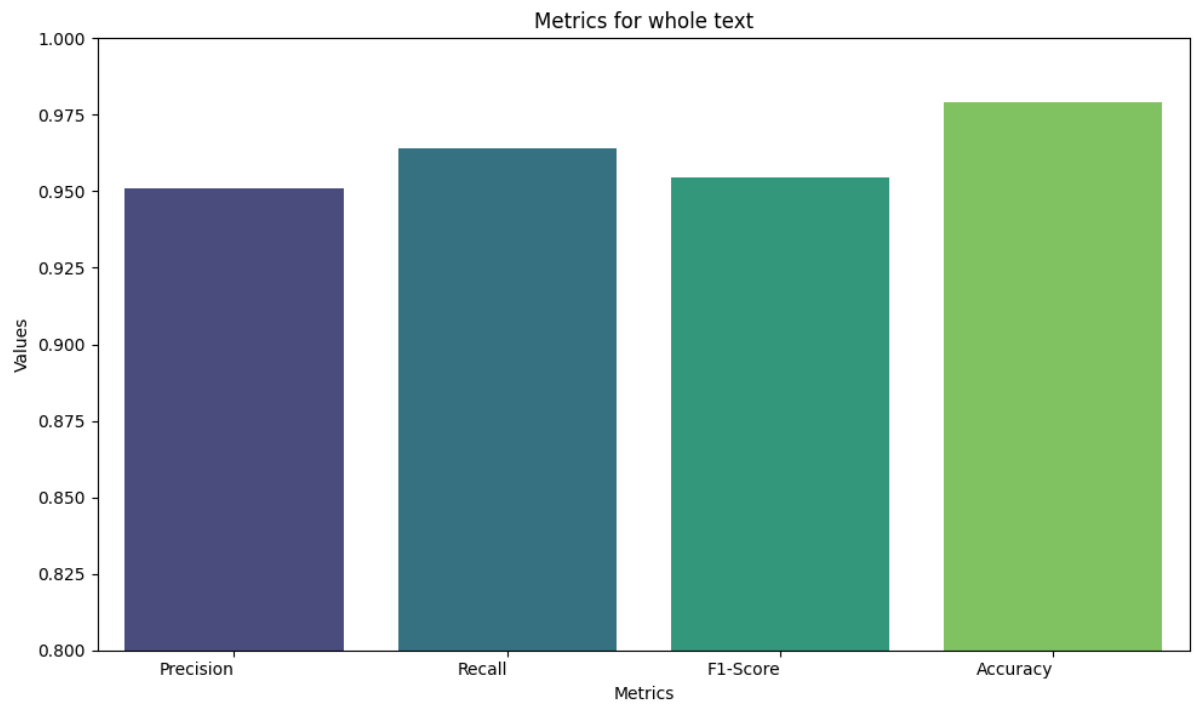


5.5. ábra – F1 értékek osztályonként.

Az utolsó mérőszám amit használtam, az az accuracy, amely azt az arányt adja meg, hogy hány esetben lett ugyanaz a *kimenet* illetve a bemenet. Accuracyt csak az egész dokumentumra számoltam. Ezt a következő képpen lehet kiszámolni [26]:

$$Accuracy = (TP + TN) / (TP + FP + FN + TN)$$

Az 5.6-os ábrán láthatóak a precision, recall, F1-score és accuracy értékek az egész dokumentumra nézve.



5.6. ábra - Precision, recall, F1-score és accuracy értékek

6. A szoftver használata

A szoftver egy konzolos alkalmazás, mely Python nyelven íródott. A használatához szükséges telepíteni a felhasznált Python verziót, illetve a felhasznált függvénykönyvtárakat. A neurális háló súlyokat Google Drive-on lehet elérni [30]. Letöltés után az `./src/network` mappába kell másolni a megfelelő működés érdekében. A program futtatásához a felhasznált csomagokat telepíteni a következő parancsok kiadásával lehet a projekt főkönyvtárában:

1. `python -m venv .`
2. `source Scripts/activate`
3. `pip install -r requirements.txt`

A program a `main.py` fájl futtatásával indítható el, mely parancssori kapcsolókat vár, melyek a következők lehetnek:

- **--path <Fájl elérési útvonala>:** A bemeneti kép elérési útvonala. Mindenképpen egy képfájl kell legyen. A \diamond jeleket nem kell megadni az elérési útvonalhoz. Alternatívája a `-p`. Ha a `--help` kapcsoló nem létezik, akkor kötelező megadni.
- **--output_path <Elérési útvonal>:** Ebbe a mappába menti a program a kimenetet, és a részeredményeket. Ha nem létezik a mappa, akkor a program létrehozza azt. A \diamond jeleket nem kell megadni az elérési útvonalhoz. Nem kötelező megadni, ha nincs megadva, akkor a kimenetet a bemeneti kép mellé menti. Alternatívája a `-op`.
- **--help:** Segítséget nyújt a program használatához. Nem kötelező megadni.
- **--debug:** Amennyiben létezik ez a kapcsoló, akkor a program elmenti a részeredményeket is. Ha meg van adva, akkor a bemeneti képpel megegyező nevű `.txt` kiterjesztésű fileban meg kell adni a képen lévő eredeti szöveget, hogy össze tudja hasonlítani vele és kiszámolni a különféle statisztikákat a program. Ha nem létezik, akkor a kimenet csak egy szöveges fájlból áll. Nem kötelező megadni.

Egy példa a programindításra:

```
python .\main.py --path D:\egyetem\Szakedolgozat\images\input\input_04.png --  
output_path C:\Users\Zoli\Desktop\teszt
```


Irodalomjegyzék

1. Yann LeCun & Yoshua Bengio & Geoffrey Hinton - Deep Learning. Nature 521. 436-44. 10.1038/nature14539. (2015)
2. Christopher M. Bishop, Hugh Bishop – Deep Learning: Foundations and Concepts. Springer. (2024).
3. Rafael C. Gonzalez - Deep Convolutional Neural Networks [Lecture Notes]. IEEE Signal Processing Magazine 35. 79-87. 10.1109/MSP.2018.2842646. (2018).
4. Juergen Schmidhuber - Deep Learning in Neural Networks: An Overview. Neural Networks 61. 10.1016/j.neunet.2014.09.003. (2014).
5. Karen Simonyan, Andrew Zisserman – Very Deep Convolutional Networks for Large-Scale Image Recognition arXiv 1409.1556. (2014).
6. Showmik Bhowmik – Document Layout Analysis, Springer (2023).
7. <https://www.pairsoft.com/blog/revolutionizing-text-processing-the-history-and-future-of-ocr-technology>
Utolsó elérés dátuma: 2025.05.11
8. Wang Junmiao - A Study of The OCR Development History and Directions of Development. Highlights in Science, Engineering and Technology 72. 409-415 (2023).
9. <https://medium.com/mysuperaai/how-is-ocr-used-in-the-real-world-e82d1354f07b>
Utolsó elérés dátuma: 2025.04.04
10. https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html
Utolsó elérés dátuma: 2025.04.05
11. <https://github.com/VeCAD-P04-UTM/94-Class-ASCII-Image-OCR-Data>
Utolsó elérés dátuma: 2025.04.15
12. Sebastian Ruder – An overview of gradient descent optimization algorithms, Insight Centre for Data Analytics, NUI Galway Aylien Ltd., Dublin (2017). <https://arxiv.org/pdf/1609.04747>
Utolsó elérés dátuma: 2025.05.11
13. <https://www.python.org>
Utolsó elérés dátuma: 2025.04.20
14. <https://pypi.org/project/textdistance>
Utolsó elérés dátuma: 2025.04.20
15. <https://rapidfuzz.github.io/Levenshtein>
Utolsó elérés dátuma: 2025.04.20
16. <https://numpy.org/doc/stable>
Utolsó elérés dátuma: 2025.04.26
17. <https://pypi.org/project/pathlib>
Utolsó elérés dátuma: 2025.04.26
18. <https://pypi.org/project/opencv-python>
Utolsó elérés dátuma: 2025.04.26

19. <https://pypi.org/project/matplotlib>
Utolsó elérés dátuma: 2025.04.26
20. <https://pypi.org/project/seaborn>
Utolsó elérés dátuma: 2025.04.26
21. <https://pypi.org/project/scikit-learn>
Utolsó elérés dátuma: 2025.04.26
22. <https://pypi.org/project/torch>
Utolsó elérés dátuma: 2025.04.26
23. <https://pypi.org/project/torchvision>
Utolsó elérés dátuma: 2025.04.26
24. <https://pypi.org/project/pillow>
Utolsó elérés dátuma: 2025.04.26
25. Nadira Muda, Nik Kamariah Nik Ismail, Siti Azami Abu Bakar, Jasni Mohamad Zain - Optical Character Recognition By Using Template Matching (Alphabet) National Conference on Software Engineering & Computer Systems 2007 (NACES 2007)
26. Sokolova, Marina & Japkowicz, Nathalie & Szpakowicz, Stan. - Beyond Accuracy, F-Score and ROC: A Family of Discriminant Measures for Performance Evaluation. AI 2006: Advances in Artificial Intelligence, Lecture Notes in Computer Science. Vol. 4304 1015-1021. 10.1007/11941439_114. (2006).
27. <https://developer.nvidia.com/cudnn>
Utolsó elérés dátuma: 2025.04.26
28. Mohaiminul Islam, Guorong Chen, Shangzhu Jin - An Overview of Neural Network. American Journal of Neural Networks and Applications Vol. 5, No. 1, pp. 7-11. doi: 10.11648/j.ajnna.20190501.12 (2019).
29. S. Mohamed, Ihab. - Detection and Tracking of Pallets using a Laser Rangefinder and Machine Learning Techniques. 10.13140/RG.2.2.30795.69926. (2017).
30. <https://drive.google.com/file/d/13w3RfT4jpOiOgm7ZAKMAVcAj56TJsCfM/view?usp=sharing>
Utolsó elérés dátuma: 2025.05.23

Nyilatkozat

Alulírott Varga Zoltán programtervező informatikus BSc szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet Képfeldolgozás és Számítógépes Grafika Tanszékén készítettem, programtervező informatikus diploma megszerzése érdekében.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy szakdolgozatomat a Szegedi Tudományegyetem Diplomamunka Repozitóriumban tárolja.

Szeged, 2025.05.11

Varga Zoltán