EÖTVÖS LORÁND TUDOMÁNYEGYETEM

INFORMATIKAI KAR

ADATTUDOMÁNYI ÉS

ADATTECHNOLÓGIA TANSZÉK

Kézzel írt szavak digitalizálása neuronhálók segítségével

Külsős témavezető:
Páros Zoltán
Mérnökinformatikus
MSc

Belsős témavezető:
Dr. Horváth Tamás
Adattudományi és
Adattechnológia Tanszék
Tanszékvezető

Szerző:
Koncz Gábor
programtervező matematikus,
nappali tagozat

Budapest, 2018

Tartalom

1 Bevezetés 6

1.1 Írásdigitalizálás 6

1.2 A diplomamunka célja 6

2 Írásfelismerés 8

2.1 Offline felismerés – Írott folyószövegek felismerése: 8

2.2 Offline felismerés – Nagybetűkkel írt különálló szavak felismerése: 8

2.3 Offline felismerés – Nyomtatott szövegek felismerése: 9

2.4 Online felismerés: 10

2.5 Írás felismerés egyéb területei: 10

3 Elméleti alapok 11

3.1 Neurális hálózat építő kövei 11

3.1.1 Neuron 11

3.1.2 Aktivációs függvény 12

3.2 Neurális hálózat felépítése 15

3.3 Tanulási folyamat 16

3.3.1 Hibafüggvény 17

3.3.2 Visszaterjesztés (Backpropagation) 17

3.3.3 Tanulási tényező (Learning rate) 19

3.3.4 Minta halmaz végig futtatása a neurális hálózaton 19

4 Tesztelés során használt algoritmusok 20

4.1 Neuronháló felépítése 20

4.2 Neuronháló bemeneti értéke 22

4.3 Neuronháló változatai 24

4.4 Különböző képmódosító algoritmusok a bemeneti érték előállításához 24

4.4.1 Medián szűrő: 25
4.4.2 Bináris képpé alakítás: 26
4.4.3 Objektumok, betűk megkeresése: 27
4.4.4 Maradék zajok, foltok kiszűrése: 28
4.4.5 Egységes betűvastagság: 28
4.4.6 Üres részek levágása és átméretezés: 28
4.4.7 Az I betű különleges kezelése 29
4.5 A neurális hálózat tanításához használt adatok 30
4.5.1 Saját magam által gyűjtött betű képekkel: 30
4.5.2 Interneten talált képek gyűjteményével 30
5 Egy szó felismerésének menete 32
5.1 Az egyes betűk felismerése 32
5.2 Szótárak használata 33
5.3 Magyar szavak kezelése 36
6 Eredmények 37
6.1 Tesztadatok 37
6.2 Összehasonlítás egy szófelismerő szoftverrel 37
6.2.1 Google képfelismerő szoftver 37
6.2.2 Összehasonlítás eredménye 38
6.3 Az algoritmus vizsgálata több szemszögből 41
6.3.1 Teszt személyek vizsgálata 41
6.3.2 A két féle tanulási adathalmaz vizsgálata 42
6.3.3 Az egyes betűk felismerésének vizsgálata 42
6.3.4 A betűk vizsgálata a hibázás szemszögéből 44
6.3.5 Különböző neurális hálózatok összehasonlítása 48
7 Összegzés 50
8 Felhasználói dokumentáció 51

8.1 Hardver követelmények ... 51
8.2 A program részei: ... 51
8.3 Telepítés ... 52
8.4 A program használata ... 52
8.4.1 Szófelismerés felület ... 52
8.4.2 Képkivágás felület ... 55
8.4.3 Tanuló algoritmus felület ... 56
8.4.4 Fájlformátumok ... 61
8.4.5 Súgó ... 62
9 Fejlesztői dokumentáció ... 63
9.1 Osztálydiagram ... 63
9.2 Algoritmusok ... 63
9.2.1 Képek előfeldolgozása: ... 63
9.3 Fontosabb függvények, adatszerkezetek, megvalósítások ... 65
9.4 Osztályok leírása ... 66
9.4.1 MainForm ... 66
9.4.2 ImageForm ... 67
9.4.3 ImageEditor ... 68
9.4.4 LearningForm ... 68
9.4.5 LearningControl : DefaultControl ... 70
9.4.6 LearningAlgorithm ... 71
9.4.7 RecognitionControl : DefaultControl ... 72
9.4.8 DefaultControl ... 73
9.4.9 Letter ... 74
9.5 AForge.Net keretrendszer ... 75
9.6 Tesztelés ... 76
9.7 Továbbfejlesztési lehetőségek ... 79

9.8 Verziójegyzék 79

10 Ábrajegyzék 80

11 Irodalomjegyzék 82

1 Bevezetés

1.1 Írásdigitalizálás

A karakterfelismerés egyre nagyobb teret hódít az informatikában, ezt az a tény is jelzi, hogy több nagy informatikai cég, pl. a Google és a Microsoft is nagymértékben támogatja az ezen a területen történő kutatásokat és fejlesztéseket. A képfeldolgozás az informatika egyik dinamikusan fejlődő területe, amely, alkalmazási lehetőségeit tekintve, sok egyéb tudományterület feladatainak megoldása, illetve megoldásának segítése céljából felhasználható.

A kézzel írt szövegek felismerését két fő csoportba szokták sorolni: offline és online. Online írásfelismerés esetén írás közben azonosítjuk a karaktereket, ilyen elven működnek például a táblagépek és okos érintőképernyős telefonok. Az offline pedig, amit én is választottam feladatnak, amikor az írást követően, a bedigitalizált képen történik a felismerés. Az offline szövegek digitalizálásának is több alcsoportja van. Ilyenek a nyomtatott szövegek és a kézzel írottak. Ez utóbbiakat még tovább lehet csoportosítani, a nagybetűkkel és a folyószöveggel írott szövegek felismerése.

Programom az írásfelismerés egy részével foglalkozik: az offline, kézzel, szeparált nagybetűkkel írt szavak digitalizálásával. Ezt az ágát használják fel a formanyomtatványok, vagy például a levelek címének a digitalizálásakor.

1.2 A diplomamunka célja

Nagyon sokféle megvalósítás látott már napvilágot. Kiemelve párat: statisztikai, geometriai megközelítés, neuronhálós, stb. Mégis a leggyakoribb a neuronhálóval történő megvalósítás. Ennek is megannyi változata ismert. International Journal of Computer Science & Information Technology - „Diagonal based feature extraction for handwritten alphabets Recognation system using neural network” [1] című publikációból indultam ki, amely egy algoritmust tár elő, mely neuronháló segítségével tanulja meg a betűk jellegzetességeit, és a betűk képének a feldolgozására – ami majd a neurális hálózat bemeneti adata lesz, - ad egy módszert.

Célom a cikkben szereplő algoritmus segítségével olyan neuronháló létrehozása, amely a digitalizált képen felismerni a kézzel, nagy betűkkel írt szavakat. Ebből kiindulva, különböző tanuló algoritmusok, neuronhálók összehasonlításával a módszerek előnyeit-hátrányait vizsgálom meg.

A diplomamunka kitér az egyes nyelvek szójegyzékének segítségével történő szófelismerésére. A tesztelés során már létező betűfelismerő szoftverrel hasonlítom össze az eredményt.

A diplomamunka során megvizsgálom a magyar nyelv ékezetes betűinek a jellegzetességeit és a megoldásommal, a programom fel tudja ismerni az ilyen betűket is.

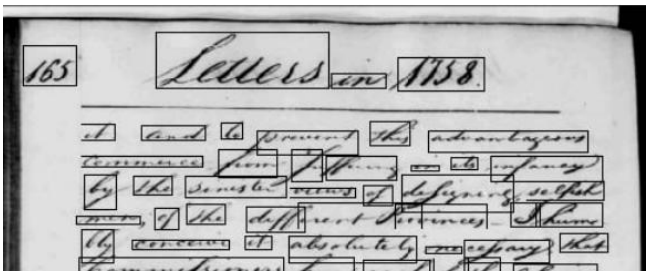
2 Írásfelismerés

Ebben a fejezetben bemutatok néhány érdekes, részletes példát az írásfelismerés bevezetésben említett változataira. A példák is jól szemléltetik, hogy a terület mennyire széleskörű és komplex.

Az írásfelismerést, a szakirodalom OCR betűszóval jelöli, mely az angol Optical Character Recognition rövidítése, de általában ma már csak a nyomtatott szövegek felismerését értik alatta. A kézzel írt szövegek karakterek felismerését ICR (Intelligent Character Recognition), míg a szavakét IWR (Intelligent Word Recognition) rövidítéssel használják.

2.1 Offline felismerés – Írott folyószövegek felismerése:

Mikor egy egész oldalt, dokumentumot szeretnénk felismertetni, akkor az egyes betűk felismerésén kívül szükséges meghatározni az egyes sorokat és szavakat is.[2]



1. ábra: Szavak megtalálása egy képen [2]

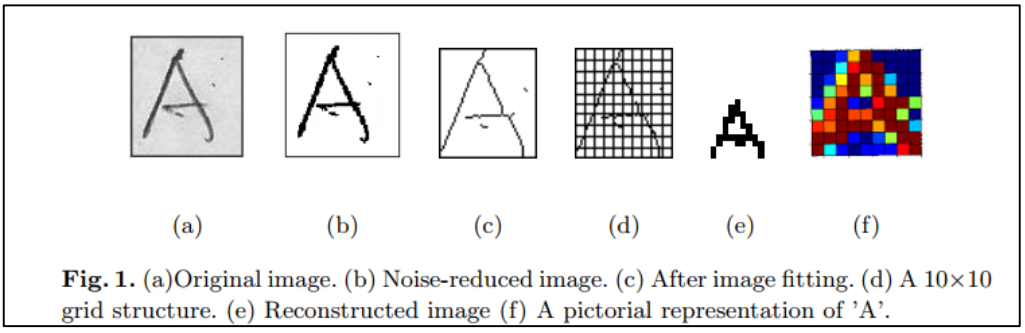
Ha megvannak a szavak, utána az egyes betűket fel kell ismerni, de vannak olyan megvalósítások, hogy szótagokat, szórészeket vagy szavakat egyben próbál felismerni egyben az algoritmus.

2.2 Offline felismerés – Nagybetűkkel írt különálló szavak felismerése:

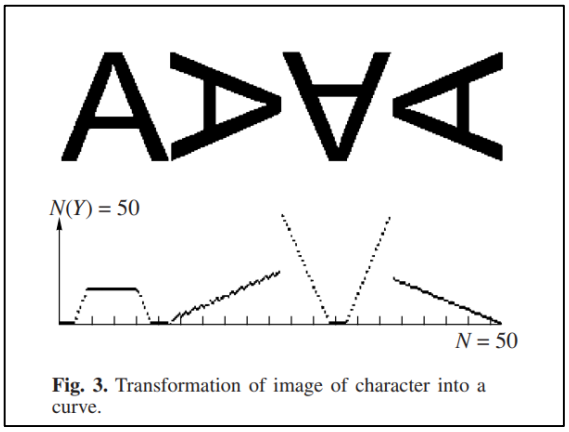
Én ezzel a témakörrel foglalkoztam, és azon belül is a neuronhálókkal való megvalósítással. Számos egyéb érdekes megvalósítást ismert, melyekből kiemelek párat.

- Fúziós eljárásokkal [3]

- Pozíció-szélesség-impulzus algoritmussal [4]
- Egyenesek és görbék vizsgálatával történő algoritmus



2. ábra: Fúziós megoldás [3]

- 

3. ábra: Pozíció-szélesség-impulzus algoritmus [4]

A neuronhálóval történő felismerésnek is több változata ismert, ahol különbféle neurális hálózat felépítést alkalmaznak, vagy ezeknek a neurális hálózatnak a bemenő paramétereire adnak ötletet, hogy hogyan lehet jobb eredményt elérni a felismerés során.

2.3 Offline felismerés – Nyomtatott szövegek felismerése:

Nyomtatott szövegek felismerése alatt azt értejük, amikor vagy PDF-ből vagy egy írott dokumentumból ismeri fel a betűket. Ez a terület talán a legtöbbet foglalkoztatott, mert ezt használják a legszéleseb körben, például a könyvek digitalizálásánál is. Ebben az esetben könnyebb a szavak felismerése, mert ugyanúgy fognak kinézni az egész dokumentumban az egyes betűk. A nehézséget és komplexitást az adja, hogy fel tudja ismerni az egyes betűtípusokat, és a különböző formázásokat megtartsa, például a

bekezdések, behúzások, vagy a betük különböző beállításait, az aláhúzott, dőlt vagy vastagított írást.

2.4 Online felismerés:

Az online felismerés esetében valós időben ismerjük fel a rajzolt betűket és szavakat. Ilyenkor segítség az, hogy tudjuk, hogy a felhasználó hogyan írta le az egyes betűket, mármint, hogy az egyes vonalak, ívek milyen sorrendben lettek lerajzolva, leírva. Ez a különböző okos telefonok és a táblagépek miatt szintén sokat foglalkoztatott terület.

2.5 Írás felismerés egyéb területei:

Az írás felismerésnek vannak olyan területei, amelyek az egyes országok jellegzetes betűivel foglalkozik, mint például a kínai írásjelek. A magyar nyelvben is vannak speciális karakterek az angolhoz képes, ilyenek az ékezetes magánhangzók. A programom és a diplomamunka is kitér ennek a megvizsgálására.

Ebbe a fejezetbe lehet sorolni azokat is, amikor valamilyen speciális jeleket tartalmazó felismerést végzünk, mint például a matematikai képletek felismerése.

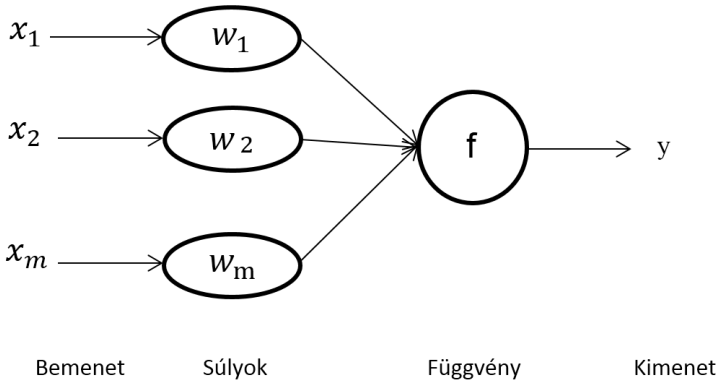
3 Elméleti alapok

3.1 Neurális hálózat építő kövei

A mesterséges neurális hálózatot sokféle területen használják tanuló algoritmusként, ilyen például a karakter, a kép és a jel felismerés, de különböző adatbányászatokhoz is használható. A neurálisháló kifejezést a biológiából vette a számítógépes tudomány, mert sok mindenben hasonlít hozzá, úgymint a felépítésében és a működésében is. A biológia idegrendszernek másnéven neurális hálózatnak három fő része van, a receptorok, melyek az ingereket veszik fel, a központi idegrendszer mely a neuronok segítségével számításokat végeznek és a hatása az ingerekre. A számítógépes neurális hálózatoknak meg vannak szintén ezek a részei, melyek a bemeneti, számító és a kimeneti részek, rétegek.

3.1.1 Neuron

Neuronok a számításokat végző alapegységek melyeket összekapcsolva hálózattá neurális hálózatot kapunk. A processzoroktól abban különbözik, hogy míg a processzorokat programozzák, addig a neuronokat tanítják.



4. ábra: Egy neuron felépítése

Az x_i (i=1,…,m) bemenetek súlyozott értékeinek a hatása lesz a kimenet, ahol a súlyok száma megegyezik a bemenő értékek számával: w_i Є R (i=0,…,m) A w_i-től

függ, hogy egy adott bemenet mennyire játszik szerepet a kimentbe, ha w_i<0 akkor gátló, ha w_i>0 akkor serkentő súlynak nevezik. A kimenet értékét a bemenet értékeinek a súlyozásával kapjuk:

$$y = f(x_1, x_2, \dots, x_m, w_1, w_2, \dots, w_m)$$

A legegyszerűbb ilyen függvény a lineáris:

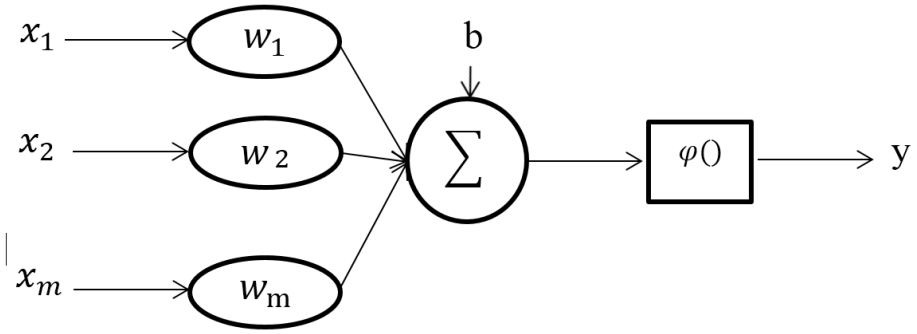
$$y = \sum_{i=1}^{m} (w_i * x_i)$$

Ez teljesen lineáris működésű és a kimenet bármekkora értéket vehet fel. Ezzel egy origón átmenő egyenes képét kaphatjuk, ezért eltoljuk az egyes neuronok kiszámolt értékét egy b torzítási értékkel:

$$y = \sum_{i=1}^{m} (w_i * x_i) + b$$

3.1.2 Aktivációs függvény

Ahhoz hogy ne lineáris függvény kapjunk, be kell vezetnünk valamilyen aktivációs függvényt. Az aktivációs függvény mondja meg, hogy a neuron mennyire érvényesül az adott bemenetek hatására.



5. ábra: Egy neuron felépítése

A következőképp alakul az egyenlet:

$$y = \varphi(\sum_{i=1}^{m}(w_i * x_i) + b)$$

Leggyakrabban olyan aktivációs függvényeket használunk, amelyek monoton növekedők, jobbról folytonosak, határértékük a -∞-ben 0, a +∞-ben pedig 1.

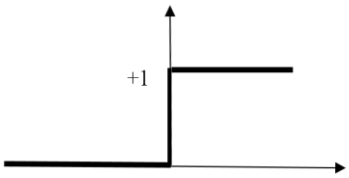
Pár elterjedtebb aktivációs függvény:

- Küszöb függvény
- Szakaszonként lineáris függvény
- Szigmoid függvény
- Hiperbolikus tangens
- ReLu

<u>Küszöb függvény:</u>

A küszöb függvény 1-et ad a kimeneten, ha a bemenete pozitív és 0-át különben.

$$f(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$$

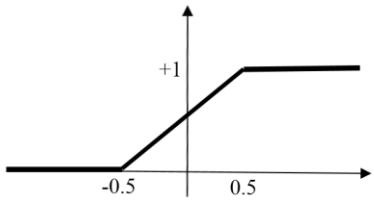


6. ábra: Küszöb függvény képe

<u>Szakaszonként lineáris függvény:</u>

A szakaszonként lineáris függvény nagyon hasonlít a küszöb függvényre:

$$f(x) = \begin{cases} 1\,, & x \geq 0.5 \\ x + 0.5, & 0.5 > x \geq -0.5 \\ 0, & x < -0.5 \end{cases}$$

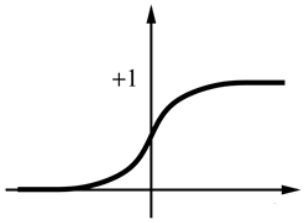


7. ábra: Szakaszonként lineáris függvény képe

Szigmoid függvény:

Más néven logisztikus függvénynek szokták nevezni.

$f(x) = \frac{1}{1+e^{-x}}$, melynek a kimenete: [0,1]



8. ábra: Szigmoid függvény képe

A programom során a szigmoid függvényt használtam és a tanulás során szükségünk lesz ennek a deriváltjára, ezért itt megjegyzem: $\frac{df(x)}{d(x)} = f(x) * (1 - f(x))$

Hiperbolikus tangens:

Szigmoidhoz hasonló aktivációs függvény, melyet jellemzően rejtett rétegekben alkalmaznak. A bemenetet -1 és +1 közé szorítja. [9]

$$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

ReLu függvény:

ReLU (rektifikált lineáris egység), a rejtett rétegek között talán leggyakrabban használt aktivációs függvény. A negatív bemeneteket nullára állítja, a pozitívokat változatlanul hagyja. Bár 0-nál nincs deriváltja, de lebegőpontos számítás esetén rejtett

rétegek között 0 bemeneti érték nagyon valószínűtlen és a gyakorlatban nem okoz problémát. Nem számításigényes és nem okoz gradiens-elhalást.

$$f(x) = \max(0, x)$$

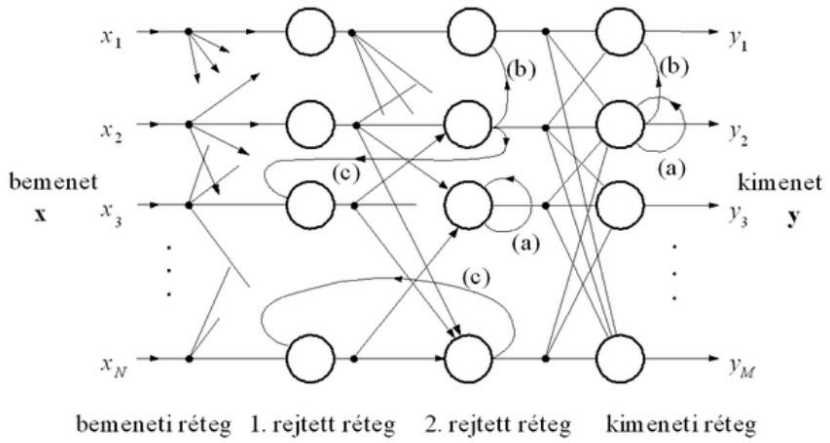
3.2 Neurális hálózat felépítése

A neurális hálózat felépítésén a neuronok rendszerét és összecsatlakozásait értjük. A neuronokat különböző kapcsolatok, vagyis biológiából vett szóval, szinapszisok kötik össze. A neuronokat rétegekben helyezzük el, az azonos rétegben elhelyezkedő neuronok vagy a kezdeti értékektől kapják a bemenetet, vagy pedig egy korábbi rétegtől, és a kimeneteik vagy egy következő réteg bemenetei, vagy pedig a teljes hálózat kimeneti értékei. Három réteg típust különböztetünk meg, bemeneti, rejtett és kimeneti réteget. A rejtett réteg több réteget is tartalmazhat, ezért rejtett rétegekről beszélünk.

Két féle csoportba lehet sorolni a neuronhálókat az összeköttetési rendszer alapján:

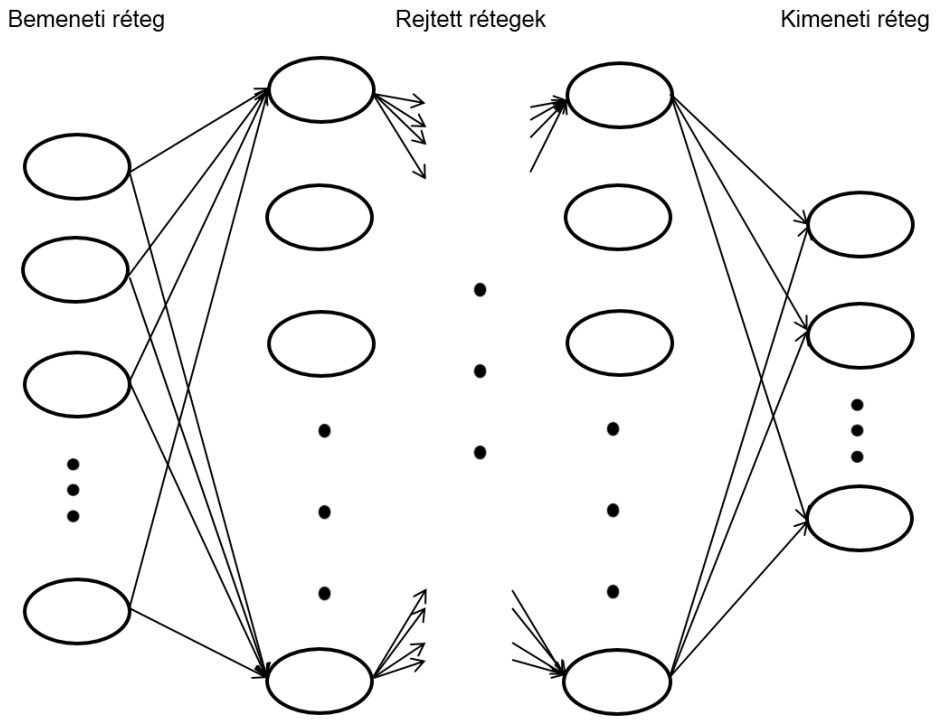
- Előre csatolt hálózat
- Visszacsatolt hálózat

A visszacsatolt réteg annyiban különbözik az előre csatolttól, hogy hurok található benne, vagyis egy réteg neuronjának a bemenete nem csak egy korábbi réteg neuronjának a kimenete lehet, hanem ugyanaz a neuron korábbi kimente, vagy másik neuronnal azonos rétegen belüli kimenet vagy egy későbbi réteg neuronjának a kimeneti értéke



.

9. ábra: Visszacsatolt neurális hálózat [5]



10. ábra: Előre csatolt neurális hálózat

Az egy rejtett réteggel rendelkező hálókat sekély, és az egynél több rejtett réteggel rendelkezőket pedig mély neurális hálózatnak nevezik.

Az előre csatolt és a visszacsatolt neurális hálózat esetén a kezdeti súlyok megállapításánál, ha a megoldandó problémáról van előzetes ismeretünk, akkor lehet célzottan beállítani a kezdeti súlyokat, de erre általában nincs lehetőség, így megfelelőbb a kezdeti súlyok véletlenszerű meghatározása.

3.3 Tanulási folyamat

A neuronháló az emberi agy működését szimulálja. A tanuló algoritmus ugyanúgy működik, mint az ember esetében. Sok-sok mintakép után már fel tudjuk ismerni a dolgokat, az egyes betűknek megvannak a jellegzetességei, amik jellemzik.

A tanulási folyamatokat két féle osztályba lehet sorolni:

- Felügyelt, ellenőrzött tanulás

- Nem felügyelt, nemellenőrzött tanulás

A fő különbség a kettő között, hogy míg a felügyelt tanulás esetén rendelkezésre állnak be- és kimeneti adatpárok és arra törekszik a tanulás során a neurális hálózat, hogy minél jobban megközelítse az elvárt kimeneteket, a nem felügyelt tanítás során nem állnak rendelkezésre kimeneti elvárt értékek, hanem általában a neurális hálózatnak kell valamilyen osztályozási feladatot megoldani, a hasonlóságokat felderíteni. A neurális hálózatnak kell a nem felügyelt tanulás esetén megtalálnia az adatok közötti kapcsolatokat, összefüggéseket.

Az előre csatolt neurális hálózat a kezdeti értékekre a neuronokban található súly értékek segítségével kiszámolja a kimenetet. A tanulási folyamat során mivel tudjuk az elvárt kimeneti értéket, össze tudjuk hasonlítani a kapott kimenettel és visszafelé haladva a neurális hálózat egyes rétegjein javítjuk a neuronok súly értékeit a hiba csökkenése érdekében.

3.3.1 Hibafüggvény

Különféle hibafüggvények használatosak melyek megmondják, hogy neurális hálózaton végigfuttatásakor kapott érték, milyen mértékben tér el az elvárt eredménytől. A legalapvetőbb hibafüggvény, amikor a tényleges különbséget számoljuk ki:

$E(x) = d(x) - f(x)$, ahol d(x) az elvárt érték, és az f(x) a neuronháló kimenete.

A másik legelterjedtebb hibafüggvény a négyzetes hibafüggvény, amit én is használtam:

$$E(x) = (d(x) - f(x))^2$$

3.3.2 Visszaterjesztés (Backpropagation)

Mivel tudjuk a hibafüggvény segítségével, hogy mennyire tértünk el az elvárt eredménytől, ezért különböző módszerekkel visszafelé haladva a rétegeken, javítani tudunk a neuronok súly értékein, hogy ha majd még egyszer végig futtatjuk a bemeneti értékekre a neuronhálót, akkor a kimeneti értékek közelítsenek az elvárt eredményhez. Különböző változatokat dolgoztak már ki, hogy a közelítést gyorsítsák, és hogy a lokális minimum helyeken való beragadást elkerüljék.

Néhányat kiemelve:

- Konjugált gradiens módszer
- Momentum módszer
- Felejtő tanulás
- Newton módszer

A konjugált gradiens módszer, amikor a gradienst kiszámoljuk, de a lépés nem a negatív gradiens irányába történik, hanem bizonyos ortogonalizációs eljárással meghatározott, módosított irányba.

Momentum módszer

Részletesebben a momentum módszert ismertetném, mert ezt használtam fel a diplomamunkám során, melynek a másik neve az általánosított delta szabály.

A momentum módszer olyan heurisztikus eljárás, ahol az egyes lépésekben a súlymódosítás meghatározása két részből áll. Először a gradiens módszernek megfelelően meghatározunk egy súlymódosító értéket, azonban nem ezzel végezzük el a korrekciót, hanem még figyelembe vesszük az előző módosítás hatását is. A súlymódosítás ennek megfelelően[6]:

$$\Delta w(k+1) = \mu\big(-\nabla(k)\big) + \rho\Delta w(k)$$

Ahol ρ az ún. momentum együttható, amelynek értéke 0 és 1 között kell, hogy legyen és a ∇ jel a gradiens jelölésére szolgál. Az új tag tehetetlenséget visz az eljárásba, hiszen az adott lépésben a módosítás irányában szerepet kap az előző irány, s így közvetve a régebbi irányok is. A momentum módszer formailag megegyezik a konjugált gradiens módszerrel, ahol szintén a pillanatnyi gradiens és az előző irány eredője irányában történik az elmozdulás. Azonban míg a konjugált gradiens módszer olyan szélsőérték-kereső eljárás, amely kvadratikus hibafelület esetén bizonyítottan N lépésben konvergál és ahol az egyes tagok együtthatói meghatározhatók, addig a momentum módszer heurisztikus eljárás, ahol mind μ, mind ρ megválasztása tapasztalati úton történik.[6]

3.3.3 Tanulási tényező (Learning rate)

A fenti egyenletben a μ változó a tanulási tényező, vagy más néven tanulási faktor, melynek célja, hogy elősegítse a gyorsabb tanulási folyamatot, ez az érték adja meg, hogy mennyire gyorsan közelítsünk a minimum hely felé, vagyis milyen mértékben javítsa a súlyokat a kisebb hibázás érdekében. Ennek az optimális kiválasztása nagyon nehéz, mert ha túl kicsit választunk ki, akkor nagyon lassú lesz a tanulási folyamat, de ha túl nagy, akkor pedig túljavíthat, vagyis átugorja a minimum helyet, és ilyenkor előfordul, hogy a hiba az nem csökkeni fog, hanem növekedni. Az egyes neurális hálóknál a tanulásifaktort tesztelés során tapasztalt eredmények alapján állítottam be, mely nagyban függött a rétegek számától és a tanulása adathalmaztól, vagyis hogy mennyi mintaképre tanítottam meg.

3.3.4 Minta halmaz végig futtatása a neurális hálózaton

Minta halmazt, vagy más néven tanító készletet használunk a tanuláshoz, mely sok bemeneti és elvárt eredmény párokból áll. Ezeknek az egyszeri végig futtatását nevezzük epochnak. Egy példán szemléltetve, ha 100 betű képére tanítjuk meg a neurális hálózatot, akkor a száz betűt egyesével végig futtatva, folyamatosan kiszámolja a kimeneti vektor értékét, majd összehasonlítja az elvárt eredménnyel, hogy melyik ABC-beli betű az aktuális tanulandó betű, és visszafele haladva a neuronokon javítja a súly értékeket. Azt nevezzük egy epochnak, amikor mind a 100 betűvel végzett. Ezeket az epochokat addig ismételjük, amíg el nem érjük a megállási feltételt, mely lehet egy hibahatár elérése vagy bizonyos számú epoch elérése. Általában akkor állunk le a tanítással, ha a tanító készlet egymásutáni végig futtatásaikor kapott hibaértékek már nagyon csekély mértékben térnek el. Az, hogy mennyire kicsi hibaértéket sikerül elérni, nagyban függ a mintahalmaz számától, és attól, hogy az osztályozandó halmaz mennyire eltérő.

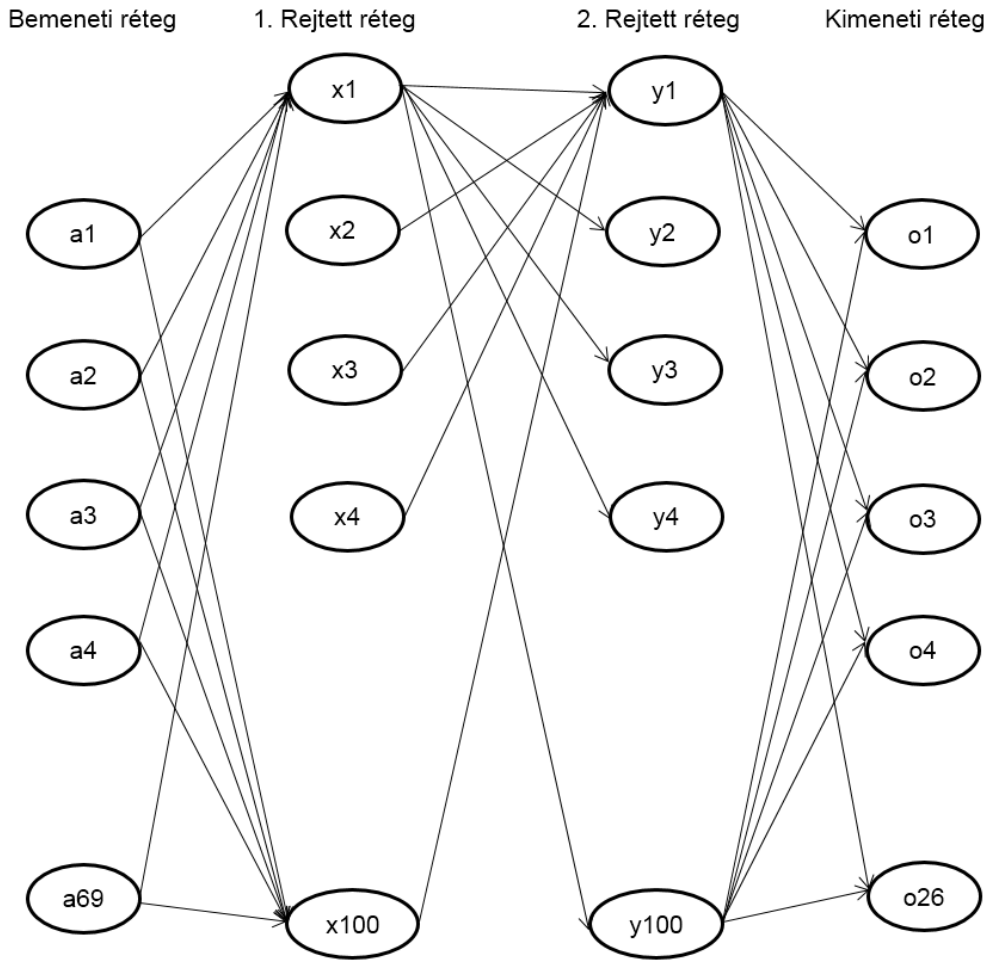
4 Tesztelés során használt algoritmusok

4.1 Neuronháló felépítése

Előre csatolt neuronhálót alkalmaztam, melyben 4 réteg van:

- Bemeneti réteg
- Két rejtett réteg
- Kimeneti réteg

A bemeneti réteg annyiban különbözik a többitől, hogy nem tartalmaz számításokat, hanem csak a bemeneti értékeket adja a következő rétegnek. A két rejtett réteg 100-100 neuront tartalmaz. A kimeneti réteg pedig 26 neuront tartalmaz, mert az elvárt eredmény az 26 különböző érték lehet az angol ABC betűiből.



11. ábra: Saját neurális hálózat

Súlyértékek:

a_i és x_j között w_{1ij} található, ahol i=1,2,…,69 és j=1,2,…,100

x_i és y_j között w_{2ij} található, ahol i=1,2,…,100 és j=1,2,…,100

y_i és o_j között w_{3ij} található, ahol i=1,2,…,100 és j=1,2,…,100

Mivel az első réteg neuronjainak 69 bejövő értéke van, és az első rétegben 100 neuron van, ezért az első réteg összesen 6.900 súly értéket tartalmaz. A második réteg az első réteg kimeneteit kapja meg, ezért ott 100 súly érték lesz egy neuronon belül, és 100 neuron van ebben a rétegben, azért 10.000 súlyt fog tartalmazni ez a réteg. Az utolsó réteg a második rétegtől kapja meg a bejövő értékeket, ami 100 súly értéket jelent az egyes neuronokban. 26 neuront tartalmaz a kimeneti réteg ezért ez 2.600 súly értéket jelent. Így a rétegekben összesen 19.500 súly érték lesz. A tanulás során ezeket a súly értékek változnak meg, hogy minél jobban az elvárt értékhez közelítsen a neurális hálózat kimenete az egyes bemeneti értékekre.

Egy új neuronhálónál kezdetben a súlyértékeket -0.5 és 0.5 közötti véletlenszerű értékekkel töltjük fel. A neuronhálózat „tudása” a neuronok közti súlyokban tárolódnak le. A tanulás nem más, mint ezeknek a súlyoknak változtatása.

A tanuláskor a mintaképekre tudjuk, hogy milyen kimenet várunk el, ezt veti össze a kapott kimenettel, az így kapott különbség a hiba. Tanulási folyamat célja ennek a hibának a minimalizálása.

Az előre csatolt neuronháló paraméterei:

Input: 69

Neuronok az egyes rétegekben: 100, 100, 26

Súly korlát: [-1,1]

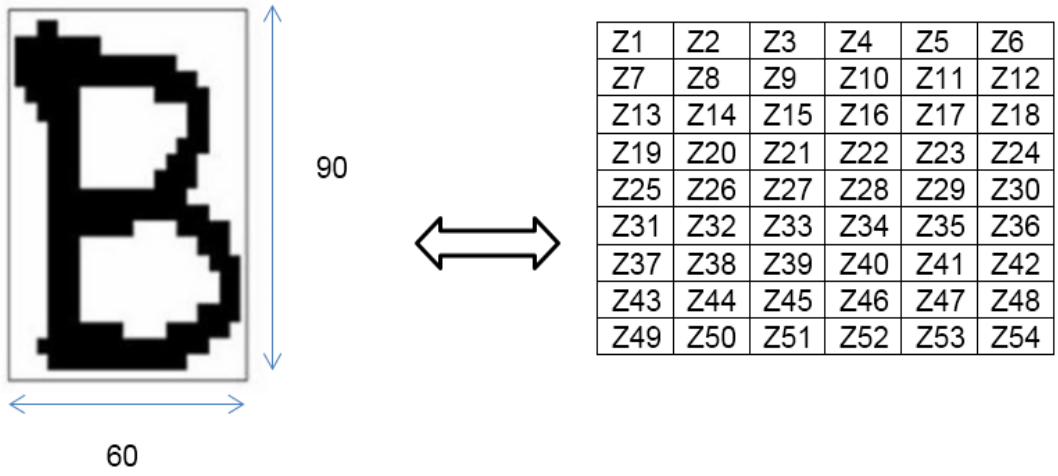
Súly inicializálás: [-0.5,0.5] véletlenszerű

Aktivációs függvény: szigmoid

Hibafüggvény: négyzetes

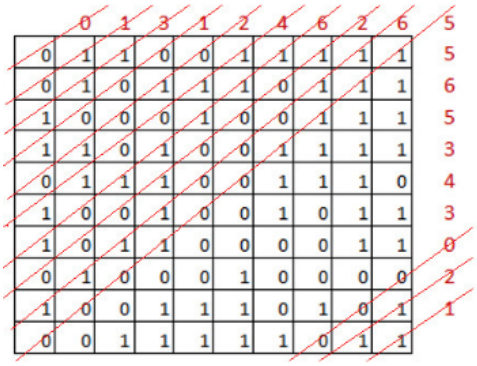
4.2 Neuronháló bemeneti értéke

A neurális hálózat bemeneti értékei a tanuló képek az egyes betűkről. A tanuláskor és a felismeréskor is átalakítja a betűket 90x60 pixeles fekete-fehér képpé. Ebből készíti el a neuronháló bemenetét: Felosztja 9x6 részre a betűt, vagyis 54 db 10x10 pixeles részképünk, cellánk lesz, ezen dolgozunk tovább. [1]



12. ábra: Kép felosztása cellákra [1]

Az egyes cellákból kiszámítunk egy értéket, mégpedig úgy, hogy a cella 10x10 es pixelből áll, 10 sor és 10 oszlop, most meghúzzuk az átlókat, amiből 19 lesz, és az átló mentén az érintett pixel értékeket összeadjuk, mivel bináris képről van szó, ezért 0 és 1-es szerepel rajta. Az összegzés után lesz 19 átlóértékünk és ezeknek vesszük az átlagát, így kapjuk meg a cella értékét.



Az átlókban található pixelek számának átlaga:

$$\frac{0+1+3+1+2+4+6+2+6+5+5+6+5+3+4+3+0+2+1}{19} =$$

$$= \frac{59}{19} = 3{,}1$$

13. ábra: A Z43 –as cella felosztása

Van most már 54 cella értékünk, ami 9 sorban és 6 oszlopban helyezkednek el. A sorokban a cellák értékeinek vesszük az átlagát, amivel lesz 9 új értékünk, és ugyanezt megcsináljuk az oszlopra, ahol 6 értéket fogunk kapni. A 69 hosszú input vektorunk következőképpen fog kinézni, a cellák sorfolytonos bejárásával az input vektorunk első 54 eleme megvan, ehhez jön a 9 sorátlag majd a 6 oszlopátlag. Ez az input vektor lesz a neuronháló bemenete és a kimenete szintén vektor, mégpedig 26 hosszú lebegőpontos számsorozat, melynek az értékei 0-tól 1-ig lehetnek. Kimeneti vektor azt mutatja meg, hogy a 26 karakteres angol ABC betűi közül melyik milyen arányban valószínű a megadott bemenetre.

A tanulás folyamán mindegyik betűre több lehetséges képet kell megadni, hogy minél jobban megtanulja az adott betű jellegzetességeit. Meg kell adni a bemenetekre az elvárt kimeneti vektort, ami ugyebár a betűt jelképezi, még pedig olyan formán, hogy az összes értéke nulla, kivéve azt a pozíciót, ahányadik az adott betű az angol ABC-ben. Például a ’D’ betű kimeneti vektora úgy néz ki, hogy a 4. pozíción 1-es szerepel, és a többi helyen 0.

A neuronhálóban több réteg van és rétegenként sok súlyérték és ezeket állítgatja a tanulás során. A bementi vektorra végigfuttatja a neuronhálót és az így kapott kimenetet hasonlítja össze a megadott elvárt kimenettel, ez alapján visszafelé haladva módosítja a súlyértékeket, hogy egyre jobban közelítsen a kimenet az elvárt kimenethez.

Ha megadunk egy C betűt, akkor az elvárt kimenet [0, 0, 1, 0, 0,…, 0] és ezt próbálja minél jobban közelíteni. A tanulás végén az várható, hogy ugyanerre a C-betűre a kimenet hasonló lesz ehhez: [0.004, 0.06, 0.97, 0.12, 0.04,…,0.01] vagyis nagyon hasonló lesz az elvárt kimenethez.

A neuronháló egyik fontos jellemzője a hibája, ami a bemenetekre kapott neuronháló kimenetek és az elvárt kimenetek közti különbségek összege. A tanulás során az egyik megadható megállási feltétel, hogy ez a hibaérték egy adott értéknél kisebb legyen.

A tanulás folyamán azért van szükség minden betűből több képre, mert mindenki másképp ír egy kicsit, és ha például több C – betű képére megtanítom, akkor egy új C – betű képét nagyobb eséllyel ismeri fel. Felmerül a kérdés, hogy akkor miért nem tanítjuk meg több százezer betűformára? Azért mert a bemeneti tanulóképek számával egyenes

arányban nő a tanulás ideje. A tanulási folyamat meglehetősen lassú, de ezt csak egyszer kell lefuttatni, és az eredmény utána bármennyiszer felhasználható.

4.3 Neuronháló változatai

Tesztelése során, kipróbáltam, hogy milyen eredményt lehet elérni, ha kevesebb vagy több réteggel rendelkezik a neurális hálózat, ezért az alkalmazásomban lehetőség van különböző neuronhálók létrehozására, paraméterezésére.

Ezek a beállítások a következők lehetnek:

- Rétegek száma
- Neuronok száma az egyes rétegekben
- Momentumváltozó értéke
- Tanulásifaktor értéke
- Epoch szám a megállási feltételhez
- Hibahatár, vagyis hibaérték a megállási feltételhez

Különböző változatokat és a belőlük levont következtetéseket az eredmények fejezetben fejtem ki részletesebben.

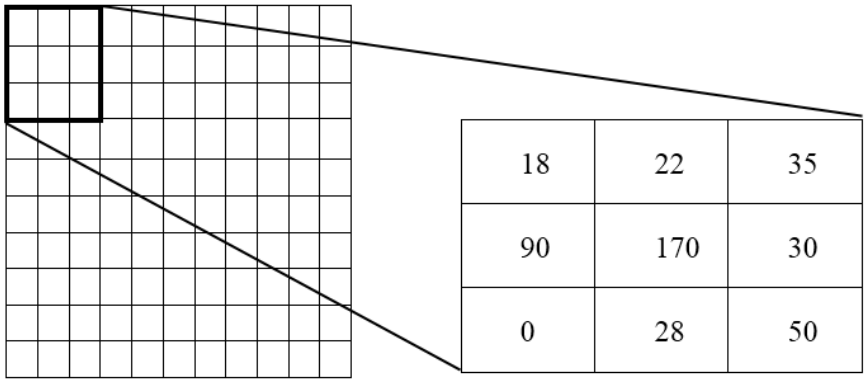
4.4 Különböző képmódosító algoritmusok a bemeneti érték előállításához

A következő fontosabb képmanipulációs algoritmusokat használtam, hogy a fent említett bemeneti vektorokat előállítsam az egyes képekből:

- Medián szűrő
- Bináris képpé alakítás
- Objektumok, betűk megkeresése
- Maradék zajok, foltok kiszűrése
- Egységes betűvastagság
- Üres részek levágása
- Átméretezés

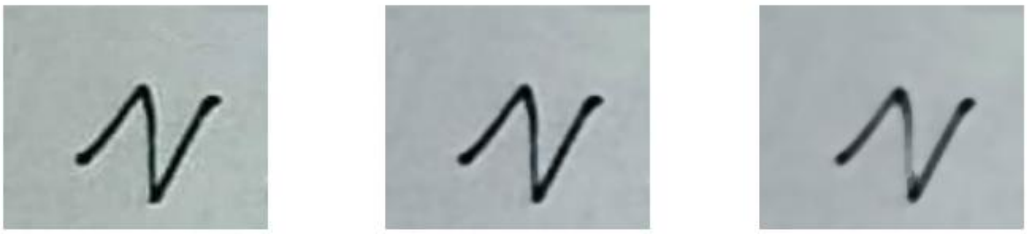
4.4.1 Medián szűrő:

Különböző zajok eltávolítására alkalmazzuk, eltünteti a kismértékű kiugró értékeket, amit a képfeldolgozási szakirodalom só, avagy bors zajnak nevez. A különböző zajszűrők közül azért ezt alkalmaztam, mert így kevésbé mosódnak el az élek, mint az átlagoló vagy a Gauss szűrő esetén. A medián szűrű neve a matematikai sorozatoknál használt medián szóból ered. A $b_1,\ b_2,\ \dots,\ b_{2n+1}$ nagyságrendbe rendezett számsorozat mediánja a középső eleme, vagyis az n+1, és ezt fogjuk a szűrőnél felhasználni a következőképp. Vesszük az első 3x3-as négyzetet:



14. ábra: Medián szűrőre egy példa

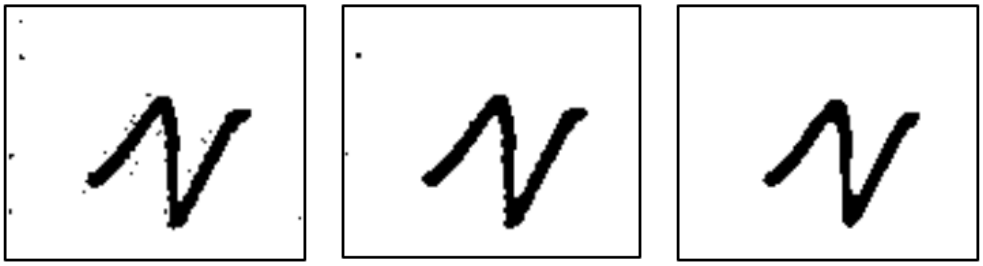
Az eredeti képen az adott 3x3-as cellában a következő értékek vannak sorban rendezve: 0,18,22,28,30,35,50,90,170 aminek a mediánja a 30 így a módosított képen a 170-es pixel helyén 30-as lesz. Az egyes képpontoknál, amikor átlagolunk, akkor mind a 3 szín értékre, RGB-re meg kell az átlagolást tenni. És ezt a folyamatot meg kell ismételni az összes 3x3-as négyzettel az eredeti képen, vagyis mindig eggyel eltoljuk, majd ha a sor végére értünk, akkor újra kezdjük egy sorral lejjebb, így megkapjuk minden egyes pixelre az új medián szűrő szerinti képet. A medián szűrőnél lehet használni a 3x3-as keret helyett, például 5x5, 7x7-es vagy még nagyobb keretet is (15. ábra).



(a) (b) (c)

15. ábra: Medián szűrők eredménye
(a) Input kép; (b) 3x3-as szűrő; (c) 7x7-es szűrő

A programom során 3x3-as szűrőt használtam, a szempont a sebesség és az eredmény összehasonlítása volt. Medián szűrő műveletigénye $O(n * m * (2k+1)^2 + \log(2k+1))$, ahol a 2k+1 a keret nagysága, vagyis a 3x3-as keret esetén a k=1. Ebből adódóan nagyságrendileg 6-szor lassabb a 7x7-es szűrő kiszámítása, mint a 3x3-as szűrőé. Az eredményt az alábbi képeken láthatjuk szemléletesebben, amikor fekete-fehér képpé alakítjuk:



(a) (b) (c)

16. ábra: Medián szűrők eredménye bináris képpé alakítás után
(a) Input kép; (b) 3x3-as szűrő; (c) 7x7-es szűrő

Innen jól látszik, hogy már a 3x3-as szűrő eltünteti a zajok 90%-t így ez megfelelőbb lesz, figyelembe véve, hogy mennyivel lassabb a 7x7-es szűrő.

4.4.2 Bináris képpé alakítás:

Ahhoz, hogy elő tudjuk állítani a neurális hálózatnak a bemenő értékeit a kép alapján, bináris képpé kell alakítanunk. Kiszámoltam a kép egyes képpontjainak az intenzitását ami alatt azt értem, hogy a 3 RGB értékét összeadtam - mely 0 és 755 közötti értéket

vehet fel, mert az egyes RGB értékek 0 és 255 közti érékek,- majd ezeknek a segítségével kiszámoltam egy határ intenzitás értéket. Ha egy adott képpont ez az érték alatt van, akkor 0 lesz az értéke, vagyis fehér lesz, különben pedig 1 lesz értéke, vagyis fekete. Ezt a határ intenzitást úgy állapítottam meg, hogy letároltam, hogy az egyes intenzitás értékeket mennyi képpontban találhatóak meg az egész képen, majd megnéztem, hogy mi az az intenzitás érték, amely feletti intenzitás értékek lefedik a kép 80%-át.

Egy sötétebb hátterű képre és egy nem feketével írt betű képre látható egy-egy példa az 17.ábrán.

(a) (b) (c) (d)

17. ábra: Bináris képpé alakítás
(a), (c) Input kép; (b), (d) Bináris kép

4.4.3 Objektumok, betűk megkeresése:

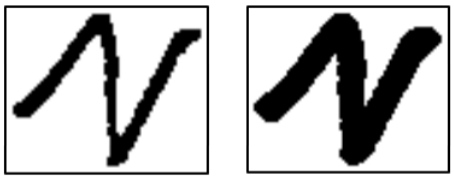
Mivel egy kép több betűt is tartalmazhat, mint például a szavak felismerésénél ezért szükségünk van arra, hogy azonosítsuk ezeket a képen. Ennek a megvalósítására rekurzív függvényt használtam, a kiinduló kép egy bináris kép, vagyis 0 és 1-eket tartalmaz. Elindulva a bal-felső sarokból egyest keresünk, ha találunk ilyet, akkor megnézzük a szomszédos pixeleket, hogy ott is egyes van-e, mert ha igen, akkor az ugyanannak az objektumnak a része. Az eljárást addig folytatjuk, amíg már nem lesz szomszédos fekete képpont, ha már nem talál szomszédos fekete pixelt, az az jelzi, hogy megtaláltuk az objektumot, majd tovább megyünk és megnézzük, hogy a képen található-e másik.

4.4.4 Maradék zajok, foltok kiszűrése:

Miután megvannak az egyes objektumok, előfordulhat, hogy a képen bizonyos zajok maradhattak, vagyis ami nem betűt ábrázol, ezeket az objektumokat lehet törölni. A hozzátartozó fekete képpontok száma alapján vizsgálom meg, hogy mik tartoznak ebbe a halmazba.

4.4.5 Egységes betűvastagság:

Különbözhet az egyes betűk vonalainak a vastagsága, így szükséges egységesíteni. Ennek eléréséhez szükséges lehet vastagítást végrehajtani, vagy másnéven bővítést, ami azt jelenti, hogy addig bővítjük fekete pixellel az objektumot, amíg el nem érünk a kívánt vastagságig. A kívánt vastagság a fekete pixelek arányát jelzi az összes pixelhez képest, ezt az értéket tapasztalati úton állítottam be..

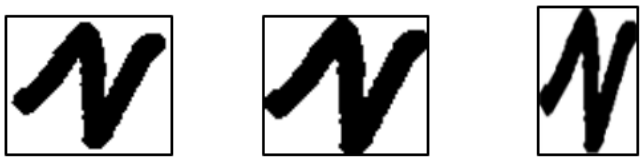


(a) (b)

18. ábra: Betűvastagság egységesítése
(a) Eredeti kép; (b) Egységesítés utáni kép

4.4.6 Üres részek levágása és átméretezés:

Az üres részek levágása után át kell méreteznünk a képet a kívánt 60x90-es méretre, hogy ki tudjuk számolni belőle a neurális hálózat bemeneti értékét a betűhöz:



(a) (b) (c)

19. ábra: Üres részek levágása és átméretezés
(a) Eredeti kép; (b) Üres részek levágása utáni; (c) Átméretezés utáni

4.4.7 Az I betű különleges kezelése

Az ’i’ betű képe nagyban különbözik a többi betű képétől, mert ha „szabályosan” van írva, akkor sokkal vékonyabb, mint a többi betű, így amikor az üres részek levágása után átméretezzük a megfelelő méretre, akkor teljesen eltorzult képet kaphatunk. Ennek orvoslására, ha az ’i’ betűt befoglaló téglalap szélessége kisebb, mint a magasságnak a 3/8 –ad része átméretezés elött átalakítjuk a képet, hogy ne torzítsa el az ’i’ betűt. A 3/8-ad értéket tapasztalati úton állapítottam meg.

(a) (b) (c)

20. ábra: ’I’ betű különleges kezelése
(a) Eredeti kép; (b) Üres részek levágása és átméretezés utáni kép;
(c) Átméretezéskor figyelve, hogy ne torzuljon el a képe

4.5 A neurális hálózat tanításához használt adatok

A tanítást két féle tanító adathalmazzal végeztem el:

- Saját magam által gyűjtött képekkel
- Interneten talált képek gyűjteményével

4.5.1 Saját magam által gyűjtött betű képekkel:

32 ismerősömtől az angol ABC betűit (26) kértem, hogy írjanak le mindegyikből 5 darabot, mert általában ugyanaz a személy sem tudja ugyanúgy leírni kétszer a betűket, így bizonyos változatokkal bővítettem a teszthalmazt egy személy esetén is. Ezen felül saját magam is írtam teszt betűket a tanuláshoz.

21. ábra: Egy személytől kért tanító kép az ABC első 3 betűjére

Így összesen az egyes angol ABC betűiből több mint 150 darabot, vagyis az összesítve **4350** betű képét olvastattam be, és tanítottam meg a neurális hálózattal.

4.5.2 Interneten talált képek gyűjteményével

A www.kaggle.com weboldalon, különböző tesztadatok találhatóak, és innen kiválasztottam az egyik megfelelő adathalmazt a betűk képeinek a tanulásához [7]. A CSV kiterjesztésű állomány 370.000 betűt tartalmaz, ahol egy sor jelöl egy betűt és a következőképp néz ki:

„3,…0,0,0,0,2,15,25,127,228,241,250,214,153,61,6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,34,127,169,183,244,222,146,134,179,246,218,93,0,0,0,0,0,0,…”

Első szám azt mondja meg, hogy az angol ABC betűi közül melyiknek a képe. A fenti példában a 3-as a D betűt jelöli, mert 0tól indult a számozás. A többi szám, ami 784 darabból áll, azt mondja meg, hogy a 28x28-as kép egyes pixeljei mennyire sötétek. 370.000 betű képének az adathalmazából véletlenszerűen kiválasztottam körülbelül 1.000 darabot minden egyes betűhöz az angol ABC-ből A programom képeket ismer fel, ezért ezeket képpé alakítottam. A szoftver a képeket mind a tanuláskor mind a szavak felismerésekor bináris képpé alakítja először, ezért a képeket már úgy mentettem le, hogy fekete fehér kép legyen, aminél azt a döntést hoztam, hogy ha a 40-es értéknél sötétebb az adott pixel, akkor azt már feketének tároltam le, majd 100x100-as méretű, „jpg" fájlformátumban mentem le a képeket.

22. ábra: Pár példa a CSV tesztadatból kinyert képekre

Összességében 24.000 betű képének a segítségével tanítottam meg a neurális hálózatot ennek az adathalmaz segítségével.

5 Egy szó felismerésének menete

Miután megtanítottam a teszt képek segítségével az angol ABC betűire a neurális hálózatot, lehetőség van letesztelni különböző szavak felismertetésével.

A szavak felismerésének négy fő lépése van:

- Szó képének feldarabolása betűkre
- A betűkhöz kiszámolni a neuronháló bemeneti vektorát
- Az egyes betük felismerése a neuronháló segítségével
- Szótár használatával megtalálni a legvalószínűbb szót

A szó képének a feldarabolása és a betük képeiből a neuronháló bemeneti vektorának a kiszámítása hasonlóképpen történik, mint a tanuló képek esetén. A neuronhálónak megadva ezt a 69 hosszú vektort, végig futtatva megkapjuk a 26 hosszú eredmény vektort, ami azt tartalmazza, hogy az egyes angol ABC betűi közül melyik milyen arányban valószínű.

5.1 Az egyes betűk felismerése

Egy „B” betű eredménye a következőképp néz ki:

[0.000038, 0.562977, 0.000001, …, 0.056012, …, 0.084199]

Ami ezt eredményezi százalékosan nézve:

A: 0 %
B: 56.3 %
C: 0 %
…
O: 5.6 %
…
Z: 8.42 %

5.2 Szótárak használata

A betűk felismerésekor a neuronháló segítségével megkapjuk minden egyes betűre, hogy az angol abc betűire mekkora a valószínűség. A továbbiakban mindegyik betűnél csak a négy legnagyobb valószínűségű abc-beli karaktert vesszük figyelembe. Végig megyünk a szótáron, és csak azokkal a szavakkal foglalkozunk, amelyek hossza megegyezik a szó betűinek a számával. Megszámoljuk, hogy az adott szó és a felismerendő szó hány karakteren azonos, mégpedig úgy, hogy a szónak az első karakterét összehasonlítjuk a felismerendő szó első betűjének a négy legvalószínűbb karakterével, és ha valamelyikkel egyezik, akkor azt mondjuk, hogy van egyezés. Tovább megyünk a második karakterre, és így végig. Ha van egyezés a szó minden karakterén, akkor ezt a szót belevesszük a lehetséges szavak listájába. Az algoritmusunk megenged kis hibákat, olyan értelemben, hogy ha egy szó esetén egy karaktert nem sikerült jól felismerni, nincs az első négy találatban, vagyis az egyezés eggyel kisebb, mint a szó hossza, akkor az ilyen szót is belevesszük a lehetséges szavak listájába, ahol 0%-al kerül be a fel nem ismert betű. Az 5 betűből álló vagy hosszabb szavaknál már két betű különbséget is megengedünk. Miután megvan a lehetséges szavak listája, már csak minden szóra átlagolni kell az egyes betűk valószínűségeit.

A tesztelés során két féle szótárt használtam:

Angol: ~350.000 szót tartalmaz

Magyar: ~30.000 szót tartalmaz

Szótárak használatának szemléltetése egy példán keresztül:

ABOVE angol szóra, ahol már csak 4 legnagyobb százalékú betűt tüntetem fel:

A betűre:

G: 1,29%

R: 0,49%

Q: 0,44%

H: 0,04%

Ami azt jelenti, hogy nem sikerült az A-betűt rendesen felismerni

B betűre:

B: 56,3%

Z: 8,42%

O: 5,6%

G: 0,01%

V betűre:

V: 99,83%

T: 0,1%

W: 0,08%

S: 0,02%

O betűre:

O: 97,83%

C: 0,42%

T: 0,08%

G: 0,07%

E betűre:

E: 99,05%

G: 1,5%

S: 1,23%

I: 0,04%

Összefoglalva az egyes betűk képeire a négy legvalószínűbb betű:

A: G, R, Q, H

B: B, Z, O, G

O: O, C, T, G

V: V, T, W, S

E: E, G, S, I

Megkeresi a lehetséges szavakat, vagyis az 5 betűsből álló szavakat, ami az angol szótáram szerint 835-öt talált:

ABASE, ABATE, ABAVE, ABOVE, … BOATS, ..., WATTS,…

Miután megvannak a lehetséges szavak, megnézi, hogy a neuronháló szerint, milyen valószínűséggel ez az adott szó, vagyis átlagolja az egyes betűk valószínűségeit. Azoknál a betűknél, melyeknél a neuronháló szerint nem volt benne az első négyben, 0%-ot vesz.

A kapott legvalószínűbb szavak:

ABOVE: (0 + 56.3 + 97.83 + 99.83 + 99.05) / 5 = 70.6 %

ROOVE: 60.56 %

GROVE: 59.6 %

…

GRAVE: (1.29 + 0 + 0 + 99.83 + 99.05) / 5 = 40.34 %

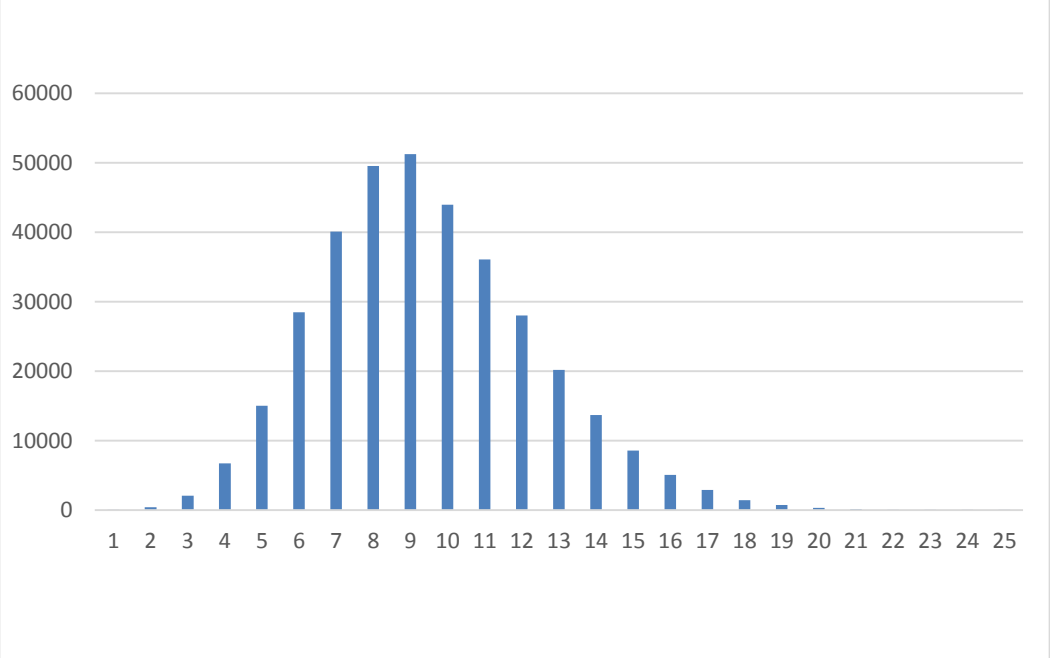
Ebből a példából jól látszik a szótár használatának az erőssége, mert így, hogy nem sikerült neki az egyik betűt felismerni, mégis megtalálta azt a szót, amit szerettünk volna felismertetni.

A több mint 350 ezer szóból álló angol szótár vizsgálata a szavak hossza alapján:

Szó hossza	Találatok száma a szótárban
1	27
2	431
3	2092
4	6758
5	15049
6	28472
7	40091
8	49528
9	51216

Szó hossza	Találatok száma a szótárban
10	43964
11	36082
12	28009
13	20174
14	13682
15	8589
16	5059
17	2905
18	1449

Szó hossza	Találatok száma a szótárban
20	355
21	167
22	73
23	30
24	12
25	8



23. ábra: Angol szótárban található szavak aránya a hosszra nézve

5.3 Magyar szavak kezelése

A magyar szavak esetében, kis bővítést kellett adni felismerés menetéhez.

A képek előkészítésekor a még bent maradt zajokat, piszkokat eltávolítottuk, azzal a logikával, hogy a betűk mérete nagyjából ugyanolyan nagy, és az ettől jelentősen kisebb foltokat kitöröljük, így az ékezetek is kitörlődnek. A továbbiakban angol abc-beli betűként kezeljük, de eltároljuk azt, hogy volt-e fölötte ékezet.

Annak ellenőrzésére, hogy a szótárban megtalálható-e a képen látható szó, a szótárban lévő szavakat is az angol ABC betűire alakítjuk az ékezetek elhagyásával. Amikor azt számoljuk, hogy milyen az adott szó valószínűsége, akkor vesszük figyelembe, hogy ékezetes volt-e. Ha a szótárban és képen is ékezetes volt a magánhangzó, akkor annak nagyobb valószínűséget adunk, körülbelül 2%-al lesz nagyobb a valószínűsége. Ez elegendő arra, hogy a találati listán a valószínűségek szerint rendezve a szavakat, előrébb szerepeljen az ékezetes szó. Ennek akkor van jelentősége, ha olyan szó fordul elő, ami csak abban különbözik, hogy ékezetes, vagy ékezet nélküli-e az egyik magánhangzó. Ilyen például a „LAP” és a „LÁP” szavunk. Az ékezet formáját nem figyeli, vagyis ugyanolyan valószínűsége lesz a „KÖR” és „KÓR” szavaknak

Példa az ilyen szavakra:

<u>LAP:</u>

LAP: 99.4%
LÁP: 97.4%
LÁT: 62.9%
PAP: 59.9%
…

<u>LÁP:</u>

LÁP: 99.4%
LAP: 97.4%
LÁT: 62.9%
PAP: 59.9%
…

<u>KOR:</u>

KOR: 99.5
KÓR: 97.5
KÖR: 97.5

Nem különbözteti meg a különböző ékezeteket, ami azt jelenti, hogy a KÓR és a KÖR esetén is ugyanazt a százalékot fogja adni.

6 Eredmények

6.1 Tesztadatok

A teszteléshez az ismerősöket megkérve írattam le szavakat, minél valósabb eredmény elérése érdekében különböztek a teszt adatot és a tanulási képeket adó személyek. Három féle teszt adattal dolgoztam:

- Angol szavak: 25 személytől ugyanazt a 20 szót leírattatva: 500 szó
- Egyéb angol szavak – figyelve, hogy minden angol betűt teszteljek: 150 szó
- Magyar ékezetes szavak

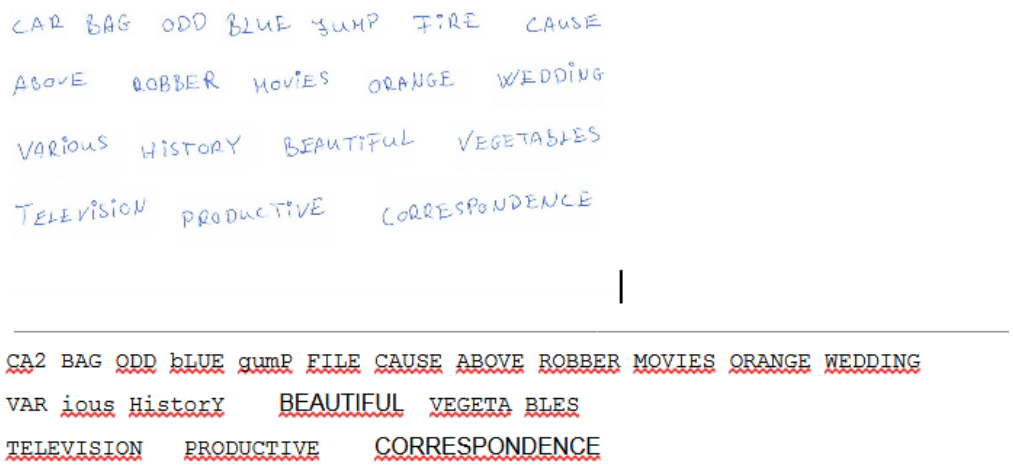
Angol szavaknál, amikor 25 személytől ugyanazt a húsz szót írattattam le, a következő szavakat használtam, figyelve arra, hogy a szavak különböző hosszúságúak legyenek:
CAR, BAG, ODD, BLUE, JUMP, FIRE, CAUSE, ABOVE, EAGLE, ROBBER, MOVIES, ORANGE, WEDDING, VARIOUS, HISTORY, BEAUTIFUL, VEGETABLES, TELEVISION, PRODUCTIVE, CORRESPONDENCE

A további angol szavak kiválasztásánál törekedtem arra, hogy minden betűje az angol ABC-nek szerepeljen megfelelő számban ahhoz, hogy statisztikai összehasonlításokat lehessen végezni velük.

6.2 Összehasonlítás egy szófelismerő szoftverrel

6.2.1 Google képfelismerő szoftver

Utána olvasva pár létező szófelismerő programoknak, a legjobb eredményt elérő ilyen ingyenes program a Google Dokumentumok felismerője. Ehhez a Google Drive-ba le kell menteni a felolvasni kívánt képet, majd Google Dokumentumként való értelmezésekor, automatikusan felismeri a képen található írásokat. [8]



24. ábra: Google Dokumentumok működése

Saját neuronhálóval történő tesztelés során ezzel a szoftverrel hasonlítottam össze az eredményeimet.

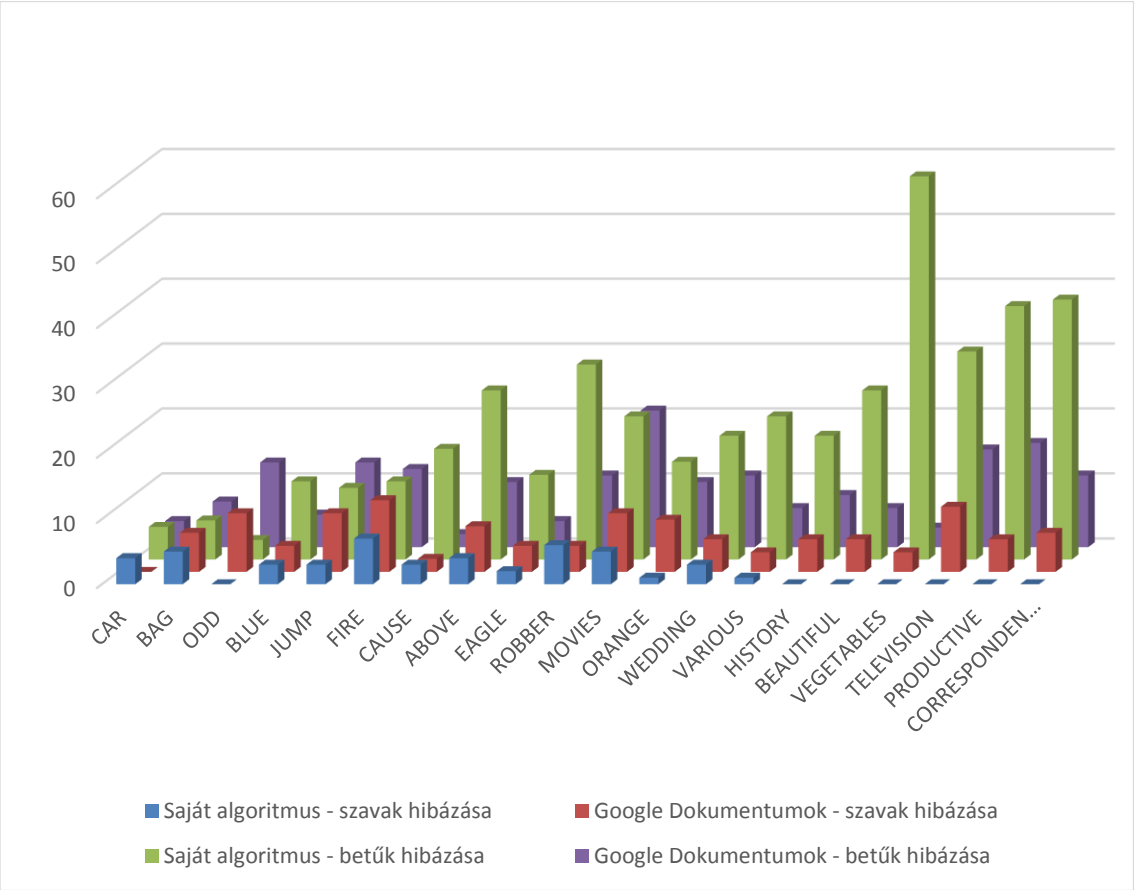
6.2.2 Összehasonlítás eredménye

Az első összehasonlításkor, 25 teszt személytől kért, ugyanazzal a húsz szóval hasonlítottam össze az a programom és a Google Dokumentumok eredményeit.

Az alkalmazásom által használt algoritmust két féle tanuló képek halmazával teszteltem le, az egyik a saját magam által gyűjtött betűk képei segítségével, mely körülbelül 4.300 betűképet tartalmaz és az interneten talált adathalmaz segítségével, melyből megközelítőleg 25.000 betű képére tanítottam meg a neurális hálózatot. (Lásd: 4.5)

A táblázatban a szavak és a betűk hibázási számát jelölöm. 25x20 szó, vagyis 500 szó és 3200 betű található a tesztben.

Szavak	**Saját neuronháló**				**Google Dokumentum**	
	4.500 betű tanításával		**25.000 betű tanításával**			
	Szó	**Betű**	**Szó**	**Betű**	**Szó**	**Betű**
CAR	4	5	14	19	4	4
BAG	5	6	13	15	6	7
ODD	0	3	4	12	9	13
BLUE	3	12	8	19	4	5
JUMP	3	11	14	35	9	13
FIRE	7	12	11	18	11	12
CAUSE	3	17	5	17	2	2
ABOVE	4	26	12	57	7	10
EAGLE	2	13	6	27	4	4
ROBBER	6	30	11	42	4	11
MOVIES	5	22	11	37	9	21
ORANGE	1	15	7	40	8	10
WEDDING	3	19	6	37	5	11
VARIOUS	1	22	10	51	3	6
HISTORY	0	19	5	40	5	8
BEAUTIFUL	0	26	1	37	5	6
VEGETABLES	0	59	3	55	3	3
TELEVISION	0	32	2	56	10	15
PRODUCTIVE	0	39	3	47	5	16
CORRESPONDENCE	0	40	1	70	6	11
Összesítve:	**47**	**428**	**147**	**731**	**119**	**188**
Arány az összesre (%)	**9.4**	**13.3**	**29.4**	**22,8**	**23.8**	**5.8**



25. ábra: Saját neuronháló összehasonlítása a Google Dokumentumokkal

A tesztelés során azt tapasztaltam, hogy a Google szoftverje 5,8 %-ot hibázott, míg a saját neurális hálózatos megvalósításom a saját tanító képekre 13,3 %-ot vétett el a betűk felismerésekor. A szavak esetében a Google Dokumentumok 23,8 %-ot, míg a programom 9,4%-ot hibázott.

Vagyis a Google Dokumentum betű felismerése 2,3-szor jobb volt, mint az enyém, de a szavak felismerése az algoritmusomnak sikerült jobban a szótár segítségével még pedig 2,5-ször teljesített jobban.

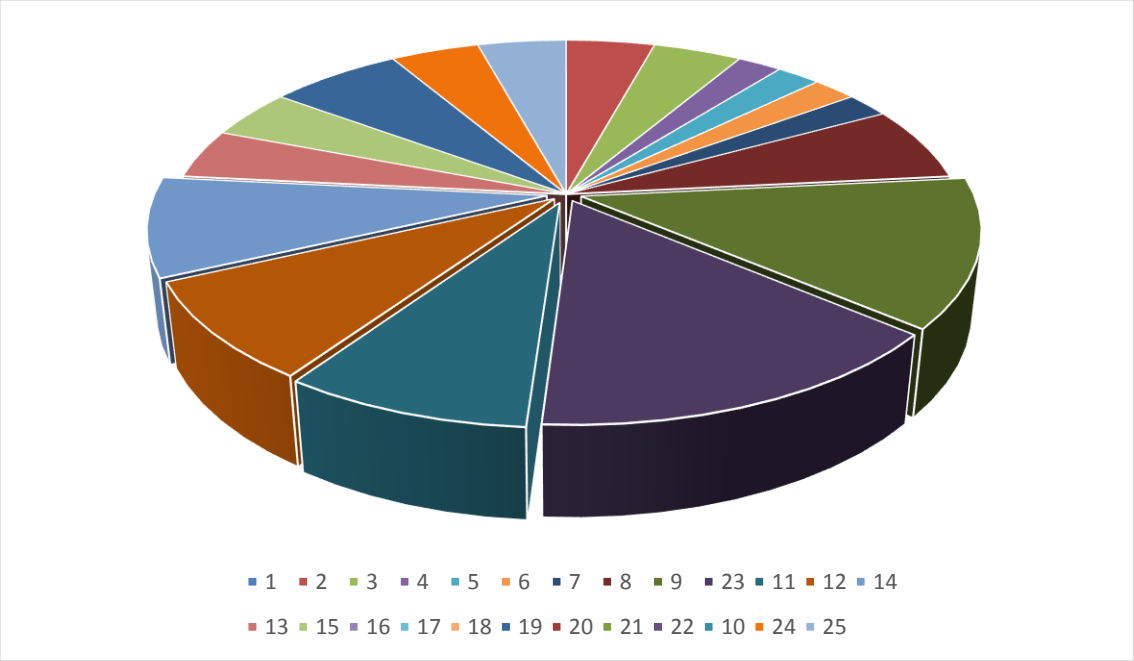
Szótár használatának az erőssége jól látható a táblázatból, mert a 6 vagy annál több betűből álló szavak esetén alig hibázott a programom. A szótár ebben az esetben ugyanazt a célt akarja elérni, mint az emberi agy, hogy nem csak az egyes betűket nézi, hanem a környezetét is, vagyis hogy milyen szóhoz tartozik. Az emberi agy működésére egy példa: „Mg gy s rtlms szvgt kpnk h nncsnk mgnhngzk”, és itt is igaz az, hogy minél hosszabb a szó, annál könnyebben ismerjük fel az adott szót.

6.3 Az algoritmus vizsgálata több szemszögből

6.3.1 Teszt személyek vizsgálata

Teszt személyeknek nevezem a továbbiakban azokat, akik a teszt szavakat írták. Mindegyik személytől ugyanazt a húsz szót kértem, hogy írják le. Az alábbi táblázatban a teszt személyek szavainak a száma látható, amiket a betanított neurálishálóm nem tudott felismerni:

Személyek	1	2	3	4	5	6	7	8	9	10	11	12	13
Szavak	0	2	2	1	1	1	1	3	**6**	0	**4**	**4**	2
Személyek	14	15	16	17	18	19	20	21	22	23	24	25	
Szavak	**4**	2	0	0	0	3	0	0	0	**7**	2	2	



26. ábra: Személyek szerinti hibázások a szavak felismerésénél

Megvizsgálva az egyes teszt személyek szavai által elért eredményeket, a táblázatból jól látszik, hogy szófelismerő algoritmus hibázásának a fele öt személyhez köthető. Ennek oka, hogy ők egy-egy betűt átlagtól eltérően írnak. Az átlag ebben az esetben, azok a személyek, akik a tanulási folyamathoz adták az egyes betűket.

Ha csak az öt személyt vizsgáljuk, akiknél a legtöbb hibázás volt a felismerés során, náluk 25%-os a rosszul felismert szavakra, míg a többit tesz személyre nézve ez összesítve csak 5%-os ez az arány.

Ha azoknak a személyeknek a szavait szeretnénk felismertetni, akik a tanuló betűket írták, akkor nagyon jó, vagyis 1% körüli hibázást kapnánk, de akkor torzított eredményeket kapnánk más szoftverekkel való összehasonlítás során, ennek elkerülése érdekében a felhasználókat két csoportba osztottam, akik a tanuláshoz adták a betűket, és akiknek a szavait ismertettem fel. Ehhez az eredményhez köthető, hogy ezzel a neuronhálós algoritmussal az egyik legjobb eredményt lehet elérni, ha az azonos felhasználók írását szeretnénk felismertetni, akik a tanuló képeket adták, és ez a neurális hálózatoknak egy nagy előnye, hogy kis eltéréseket jól tanulja meg.

6.3.2 A két féle tanulási adathalmaz vizsgálata

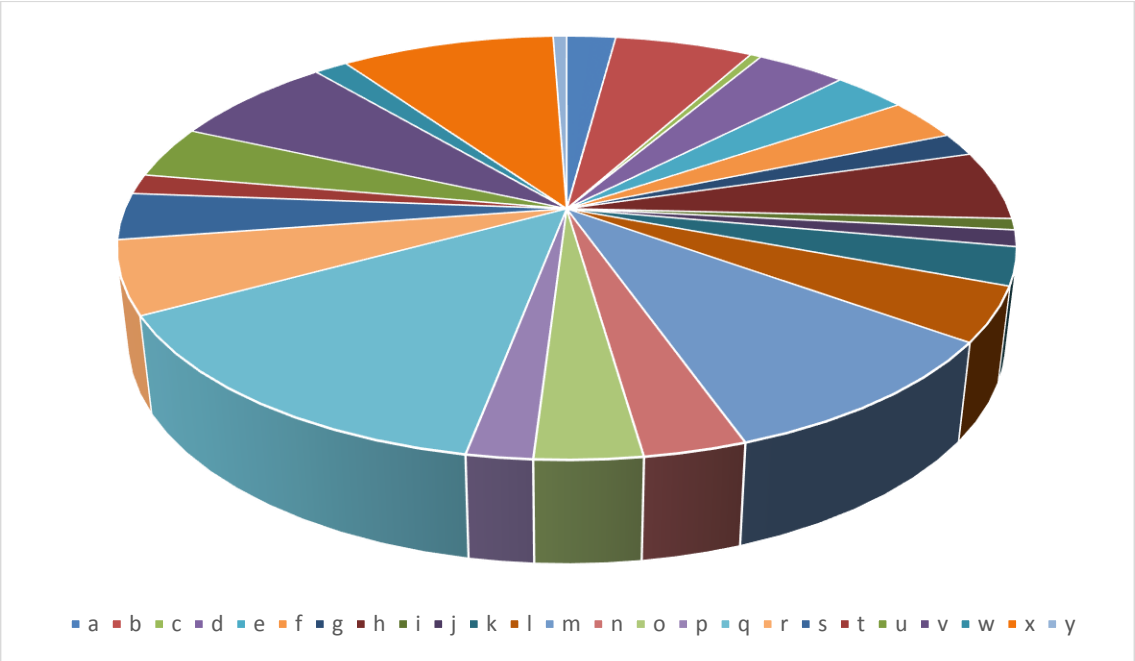
Nagyban függ a felismerés sikeressége a tanulás során adott képektől, ami jól látszik a saját és az interneten talált tanuló betűk képeire megtanított a neurális hálózat összehasonlításán. Míg a saját magam által gyűjtött képekkel 10%-os, addig a letöltött adathalmaz segítségéve 30%-os hibázás volt. Vagyis hiába volt ötször annyi tanuló kép, mégis harmad olyan eredményt tudott elérni. Jobban megvizsgálva ezeket a képet, vannak félre osztályozások benne. Volt például olyan, hogy az ’N’ betű képei között a ’Z’ betű képe szerepelt. Ha félre van osztályozva egy bemeneti kép, akkor a tanulás során rosszul fogja megtanulni a neurális hálózat az adott osztály jellegzetességeit, és így a tesztelés során a képek felismerésekor is nagyobb az esély a hibázásra.

6.3.3 Az egyes betűk felismerésének vizsgálata

A teszt szavak egyes betűinek a vizsgálatához azon kívül, hogy 25 személytől ugyanazt a húsz szót leírattattam, további szavakat kértem más személyektől, mert némely betű nem volt található ebben a húsz szóban, ilyen például a ’K’, ’Q’, ’X’, ’Z’. Ezekkel kiegészített teszt szavak száma most már több mint 750 és több mint hat ezer betűt tartalmaz. Az alábbi táblázatban a betűket, a hozzájuk tartozó hibázásokat tartalmazza. Feltüntetem, hogy az adott betű mennyiszer szerepel a teszt szavakban és a hibázások arányát is.

Betű	Hiba	Összesen	Arány(%)
A	17	442	7
B	34	327	20
C	2	238	2
D	18	273	13
E	52	934	11
F	6	101	11
G	7	230	6
H	7	62	18
I	7	444	3
J	2	70	4
K	2	21	10
L	19	248	14
M	17	97	32

Betű	Hiba	Összesen	Arány(%)
N	14	245	10
O	32	538	11
P	5	141	7
Q	7	15	47
R	45	464	18
S	20	324	12
T	7	249	5
U	25	344	13
V	32	268	24
W	2	64	5
X	7	23	30
Y	1	74	2
Z	9	40	23



27. ábra: Betűk hibázásának az aránya

A táblázatból jól látszik, hogy a betűk fele a felelős a hibázások több 80%-ért, vagyis megvannak, hogy jellegzetesen melyek azok a betűk, amelyeknél nagyobb arányban hibázik a program. Ebből arra lehet következtetni, hogy jellegzetes az, hogy melyik betűket írják máshogy az egyes emberek, és melyikek azok, amiket nagyon hasonlóan ír mindenki.

Ezek az információk nem álltak a rendelkezésemre a tanulási adatok gyűjtése során, vagyis minden embertől ugyanannyit kértem, hogy írjanak le az egyes betűkből. Ennek az információnak a segítségével nagyban lehet javítani az elért eredményen, ha további emberektől csak ezeket a betűket kérném, hogy írják le.

Azt hogy jobban megértsük a neurális hálózat tanulásának az eredményét, jobban meg kell vizsgálnunk a hibázásokat és nem csak a számok alapján levonni a következtetéseket.

6.3.4 A betűk vizsgálata a hibázás szemszögéből

Az egyes betűk felismerésekor vannak jellegzetes hibák, amiket tapasztaltam:

- Két betű hasonlósága
- Furcsán írt betük

Két betű hasonlósága:

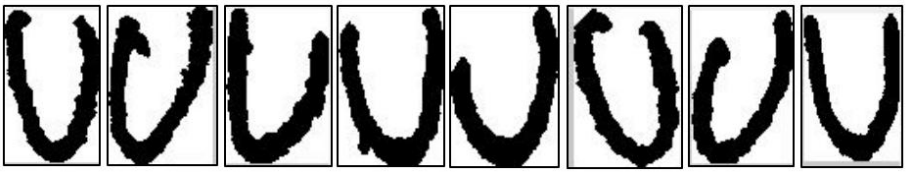
Vannak olyan betűk, melyek nagyon hasonlítanak egymáshoz, mind a két irányban, ilyenek a következő párosítások:

- O és D
- U és V

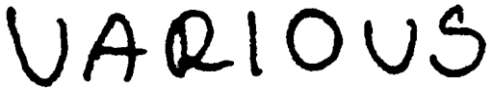
28. ábra: ’D’ betűt ’O’ betűnek ismerte fel

29. ábra: ’O’ betűt ’D’ betűnek ismerte fel

30. ábra: ’U’ betűt ’V’ betűnek ismerte fel



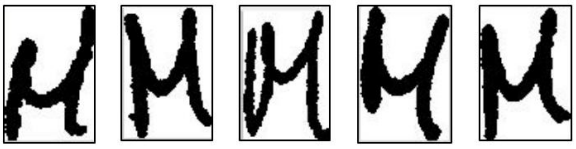
31. ábra: ’V’ betűt ’U’ betűnek ismerte fel



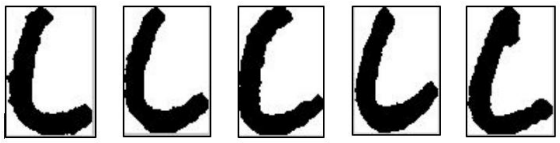
32. ábra: Egy példa, hogy egy embernél mennyire hasonló lehet az ’U’ és ’V’, egy szón belül is : „VARIOUS”

Vannak olyan betűk, melyek nagyon hasonlítanak egymáshoz, de csak egy irányban, ilyenek a következő párosítások:

- ’M’ betűt ’H’-nak ismerte fel
- ’L’ betűt ’C’-nek ismerte fel

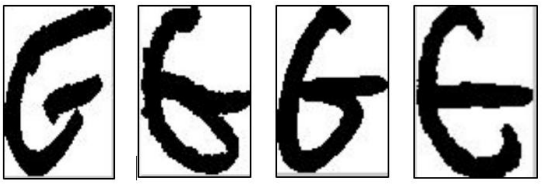


33. ábra: ’M’ betűt ’H’ betűnek ismerte fel

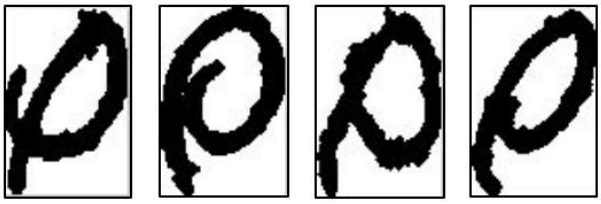


34. ábra: ’L’ betűt ’C’ betűnek ismerte fel

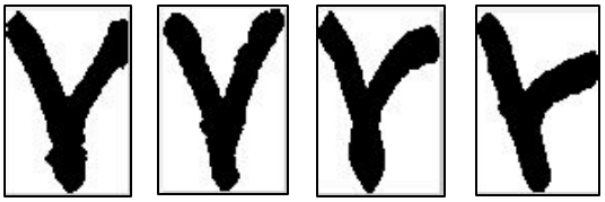
Egy-két embernél jelentkeznek jellemző esetek, amiket rosszul sikerült felismerni, de azért mert nagyon hasonlít egy másik betűre. Ezek az esetek az adott emberre jellemzőek, így több szóban is rosszul ismerte fel:



35. ábra: ’E’ betűt ’G’ betűnek ismerte fel

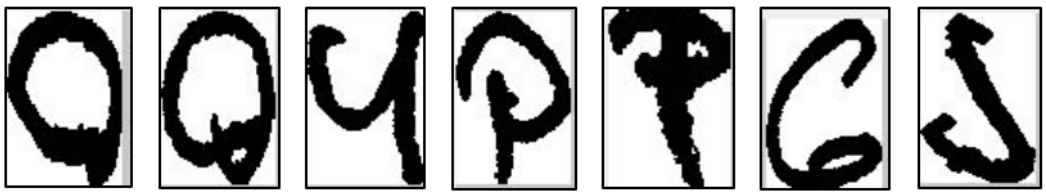


36. ábra: ’O’ betűt ’P’ betűnek ismerte fel



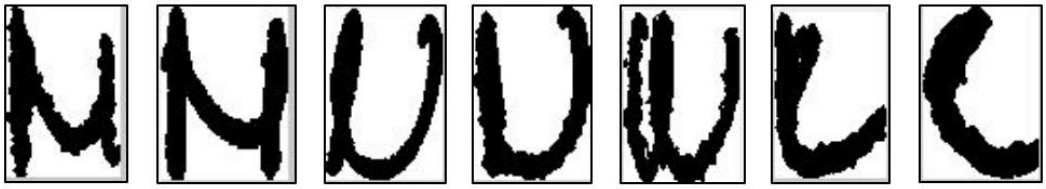
37. ábra: ’V’ betűt ’Y’ betűnek ismerte fel

Néhány esetben találtam olyan félre értelmezéseket, ahol nagyon kicsi az eredeti betű és a felismert betű között a különbség.



(a) (b) (c) (d) (e) (f) (g)

38. ábra: Hibásan felismert betűk: (a),(b) 'O' →'Q'; (c) 'U' →'Q'; (d) 'D' →'P'; (e) 'F' →'P'; (f) 'G' →'C'; (g) 'J' →'S'

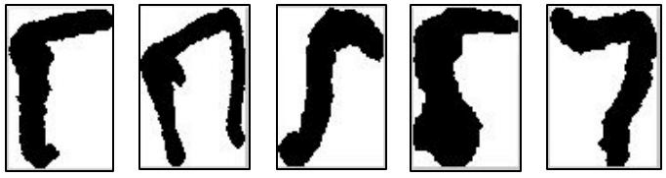


(a) (b) (c) (d) (e) (f) (g)

39. ábra: Hibásan felismert betűk: (a),(b) 'M' →'N'; (c)(d) 'N' →'U'; (e) 'N' →'W'; (f),(g) 'R' →'C'

Furcsán írt betű:

Volt pár érdekes betű képe a felismeréskor, amiket megnézve nekünk is el kell gondolkodni, hogy melyik betű is lehetett, de valószínűleg azért az emberi agy rájönne.



40. ábra: Hibásan felismert furcsa betűk

Összességében elmondható, hogy az emberi agy sem tudja megmondani bizonyos esetekben, hogy ezek milyen betűt akar jelenteni, vagy legalábbis nehezen, pedig az emberi agy bonyolultabb kép felismerő, és osztályozási „algoritmussal" rendelkezik, és a tanulás során az emberi agy több millió mintaképre tanulta meg, hogy mit jelent.

Mivel az algoritmus 10x10-es négyzetekre osztja fel a 60x90-es betű képet, vagyis 54 ilyen kis négyzet lesz, és a neurális hálózat az ezekben található értékek alapján tanulja meg, hogy milyen betűhöz milyen értékek találhatóak a betűkhöz. Ha ez az 54 kép közül

két betű között csak 3-4 négyzetben található változás, akkor nem tudja jól megkülönböztetni a két betűt, ilyen például az ’U’ és a ’V’, ahol írott formában általában csak alul középen van „értékes” információ a kettő közti különbség meghatározásához.

6.3.5 Különböző neurális hálózatok összehasonlítása

A korábbi teszteléseket háromrétegű neurális hálózattal történtek, ha a bemeneti réteget nem számoljuk bele, mert az a réteg csak a bemeneti vektort tartalmazta. A három réteg közül kettő rejtett és egy kimeneti réteg volt, és a rejtett rétegek 100-100 neuront tartalmaztak. Amikor 4.500 betű képére tanítottam meg a neuronhálót, akkor 13 volt az a hibahatár, amikor a betűket végig futtatva már csak a 6. tizedesjegyen volt változás. A 25.000 tanító halmaz esetén ez a hiba érték 48 volt, amiből látszik, hogy általában minél nagyobb a tanítandó minta halmaz, annál nagyobb lesz a hibahatár ameddig le tudunk menni. Ez azért általában igaz, mert ha ugyanazt a 4.500 betűt adjuk az tanító halmazhoz többször, akkor nem fog nőni a hiba, de nem is fogja jobban megtanulni a betűk jellegzetességeit, mert ez ilyenkor olyan, mintha többször futtatnánk le a 4.500-as tanítóbetűket. A hiba nagyban függ attól, hogy menyire különbözőek a tanító betűk és az elvárt eredményeik. Ha egy ’Z’ képe kétszer van a tanító halmazban, de más elvárt értékkel, mármint azt mondjuk, hogy az elvárt eredmény az egyik esetben ’Z’, a másikban ’N’, akkor nem fogja tudni jól meg tanulni, mert nincsenek jól szeparálva az egyes osztályok. Mint korábban említettem, ez miatt „nehéz” jól megtanulni az ’U’ és a ’V’ közti különbséget, mert sok embernél nagyon csekély az eltérés ezeknek a betűknek az írásakor.

A tesztelés során megnéztem, hogy ha a neurális hálózatnak csak egy rejtett rétege van, akkor csak 36 volt az a hibahatár, amit el tudtam érni. Ez az miatt van, mert ilyenkor kevésbé tudja megtanulni a jellegzetességeit a betűknek. A tanulás információja a neuronok súly értékeiben vannak, és ebben az esetben 100-szor kevesebb neuronunk van, így kevesebb lehetősége van megtanulni a kisebb különbségeket.

Amikor három rejtett réteggel teszteltem, akkor nagyjából ugyanolyan hibahatárt értem el, mint a két rejtett réteggel, pontosan 12.5 volt ez az érték. Hiába volt több rétegünk, százzal több neuronunk, nem tudjuk ezt a hiba határt lényegesen tovább

csökkenteni erre a tanuló adathalmazra. Elmondható, hogy 2 rejtett réteggel rendelkező neuronháló az optimális ebben az esetben, mert lényegesen nem hozott jobb eredményt a nagyobb neuronháló, elleneben a számítási idő megnőtt.

A neurális hálózatnak a bemeneti vektora 69 elemű, mely a 60x90-es betű képének a 10x10-es négyzeteiből kinyert értékeket (54), a sor átlagokat (9) és az oszlopátlagokat (6) tartalmazza. Az algoritmus nem veszi figyelembe, hogy milyen vonal vastagságú volt az adott betű, amit át akarunk alakítani a bemeneti értékekre, így különböző értékeket vett fel különböző betű vastagságok mellett. Az algoritmuson módosítottam, hogy a vektor értékeinek a meghatározásakor egységes formára hozom azáltal őket, hogy figyelembe veszem, hogy a 60x90-es képen mennyi a fekete képpontok száma:

Új vektor érték = Eredeti vektorérték/((Fekete képpontok száma/(60*90))*10)

Ekkor az ’A’ betű és a kétszer olyan vastag ’A’ betű esetén is a sor átlag és az oszlopátlag ugyanazt az értéket fogja felvenni, az egyes 10x10-es négyzetben található értékek is hasonlóak lesznek. A tesztelés során azt tapasztaltam, hogy nem lett jobb a tanulási és felismerési arány. Az oka visszavezethető arra, hogy a 60x90-es képpé alakítás előtt elvégzek egy egységesítést (Lásd 4.4.5). Ezzel az új módszerrel nem lenne szükséges egységesíteni a képeket különböző kép manipulációs módszerekkel.

7 Összegzés

A kapott statisztikai adatok alapján kijelenthető, hogy több mint 90%-osan képes olyan, a szótárban is megtalálható szavak egyértelmű felismerésére a program, melyekre teljesül, hogy a szóban található betűk nem érnek össze. Ez az érték növelhető, ha a neuronhálót tovább tanítjuk több mintaképre, figyelembe véve a kapott eredményeket, hogy melyek azok a betűk, amelyeket nagyobb százalékban ront el a program.

Google Dokumentum, mint szófelismerő programmal való összehasonlítása során, a betűk felismerése 2,3-szor jobb volt, mint az enyém, de a szavak felismerése az algoritmusomnak sikerült jobban a szótár segítségével még pedig 2,5-ször jobban teljesített.

Különböző neurális hálózatokat összehasonlítva, jól látszik, hogy meg lehet találni azt a határt, amikor a neurális hálózat felépítésének a növelése, vagyis a rétegek és a neuronok száma, már nem hoz jobb eredményt.

Ez az algoritmus jól használható lenne számfelismerésekkel kiegészítve, a leveleken található irányítószám és a hozzá tartozó település nevének digitalizálására. Tudjuk, hogy melyik irányítószám melyik településnek felel meg. Ennek segítségével a tévesztési hibát ki tudjuk küszöbölni.

Kis kiegészítésekkel formanyomtatványokon szereplő kézzel írt szövegek felismeréséhez is használható lenne. Nagyban megkönnyítené ezeknek a bedigitalizálását, mert 90%-át helyesen digitalizálja, így csak a maradék 10%-ban van emberi beavatkozásra szükség.

8 Felhasználói dokumentáció

8.1 Hardver követelmények

A program két fő részből áll, a neurális hálózat létrehozása majd tanítása és a szavak felismerése. Szavak felismerése nem igényel akkor számítás igényt, ezért az operációs rendszer és a futó programok hardverigényén felül 32MB RAM-ra van szükség.

A neurális hálózat tanítása nagy számítás és memória igényű, így minél jobb a processzor, annál gyorsabb lesz a tanulási lépések kiszámítása.

A tesztelés a következő feltételek mellett történt:

Processzor: Intel(R) Core(TM) I5-4300M CPU 2.60 GHz

Memória: 12,0 GB RAM

Operációs rendszer: Windows7

8.2 A program részei:

Handwriting recognition mappa tartalma:

Handwriting recognition.exe – A futtatandó állomány
AForge.Neuro.dll – Neuronhálóhoz szükséges segéd állomány
Aforge.dll – Neuronhálóhoz szükséges segéd állomány
dictionary mappa és tartalma – Az alapértelmezett szótárakat tartalmazza
networkWeight mappa és tartalma – Az alapértelmezett neuronhálót tartalmazza
wordRecognitionHelp.chm – Súgó fájl

images mappa – Tanuláshoz és a teszteléshez használt képeket tartalmazza

8.3 Telepítés

Először a *Handwriting recognation* mappát a teljes tartalmával együtt át kell másolni a saját meghajtóra. A program az átmásolt *Handwriting recognation* mappában levő *Handwritten recognition.exe* fájlt elindítva közvetlenül futtatható, telepítést nem igényel.

8.4 A program használata

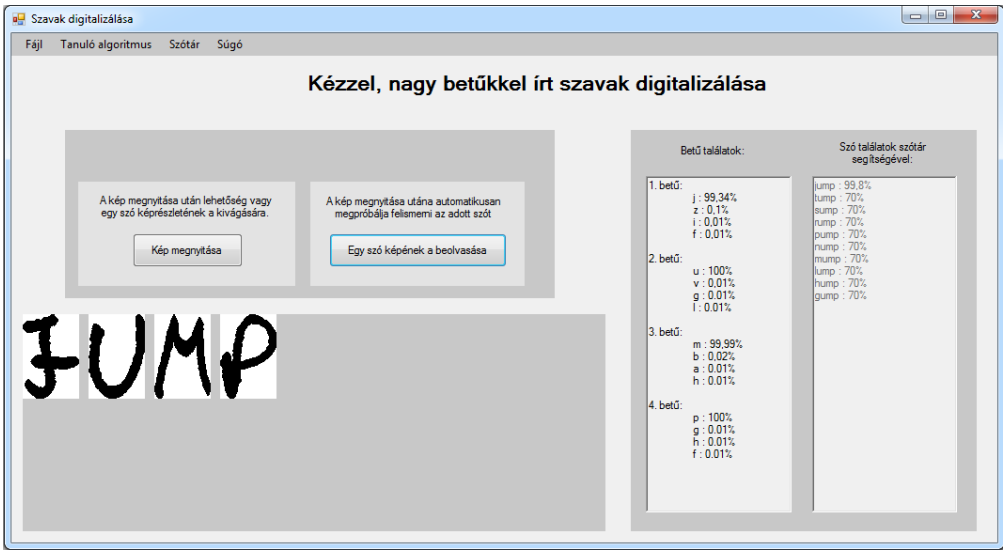
8.4.1 Szófelismerés felület

8.4.1.1 Leírás

A program fő funkciója a már bedigitalizált, szkennelt vagy fényképezett képnek a felismerése, és a találatok megjelenítése.

Először a fájlrendszerből fel kell tölteni a képet. Ennek két módja van, attól függően, hogy milyen képet szeretnénk megnyitni. Ha egy előre kivágott, és csak magát a szót tartalmazó képet szeretnénk feltölteni, akkor vagy az *Egy szó képének a beolvasása* gombra kattintva, vagy a menürendszerben a megfelelő pontot kiválasztva tudjuk megtenni, ami után egyből elkezdi felismerni a szót. A másik lehetőség, amikor egy nagyobb képnek csak egy részletét szeretnénk felismertetni. Ekkor a *Kép megnyitása* gombra kattintva, vagy a menürendszerben a megfelelő pont kiválasztása után a felnyíló ablakban lehetőségünk van a kép egy részletét kivágni, ahonnan visszatérve a program felismeri az adott képrészleten található szót. A feltöltendő fájlok mindkét esetben az általánosabb képformátumok lehetnek: BMP, PNG, JPG és JPEG.

Felismeréskor a program, különböző számítások után, megmutatja a különválasztott karaktereket, és ezeknek az abc-beli betű találati valószínűségeit. A szótár segítségével megjeleníti a legesélyesebb szavakat, és hogy ezek milyen valószínűségűek.



41. ábra: Fő felület

8.4.1.2 Menük

Fájl menü:

Kép megnyitása:

Megnyithatunk egy képet, amiből aztán kivághatjuk a számunkra szükséges, vizsgálandó szót.

Szót tartalmazó kép gyors megnyitása:

Megnyithatunk egy képet, amin csak az adott szó szerepel, és ezután egyből elkezdi felismerni. (Lásd: 9. oldal)

Kilépés:

A program bezárása.

Tanuló algoritmus menü:

Új neuronháló létrehozása:

A megnyíló felületen létrehozhatunk egy új neuronhálót, vagy egy meglévőt módosíthatunk. (Lásd: 10. oldal)

Egy neuronháló megnyitása:

Be lehet tölteni egy már létrehozott neuronhálót, hogy ennek segítségével próbálja a betűket felismerni.

Alapértelmezett neuronháló betöltése:

Az alapértelmezett neuronhálót tudjuk betölteni. Erre akkor van lehetőségünk, ha előtte beállítottunk egy másik neuronhálót.

Szótár menü:

Magyar:

Ez az egyik alapértelmezett szótár. Ezt kiválasztva egy 30 ezer szavas szótár segítségével ismeri fel a szavakat.

Angol:

Ez a másik alapértelmezett szótár. Ezt kiválasztva közel 300 ezer szavas szótár segítségével ismeri fel a szavakat. Mivel bővebb ez a szótár, a beolvasása is annál több időt vesz igénybe, eltarthat pár másodpercig.

Sajátszótár megnyitása:

Lehetőségünk van egy saját szótárt kiválasztani (Lásd: 13. oldal), ha esetleg más nyelveken akarjuk a szavakat felismerni, vagy netán egy nagyobb szókincsű szótáron. Arra figyelni kell, hogy csak az angol ABC 26 karakterét tudja felismerni.

Súgó menü:

Tartalom:

Elindítja a súgót. (Lásd: 14. oldal)

Névjegy:

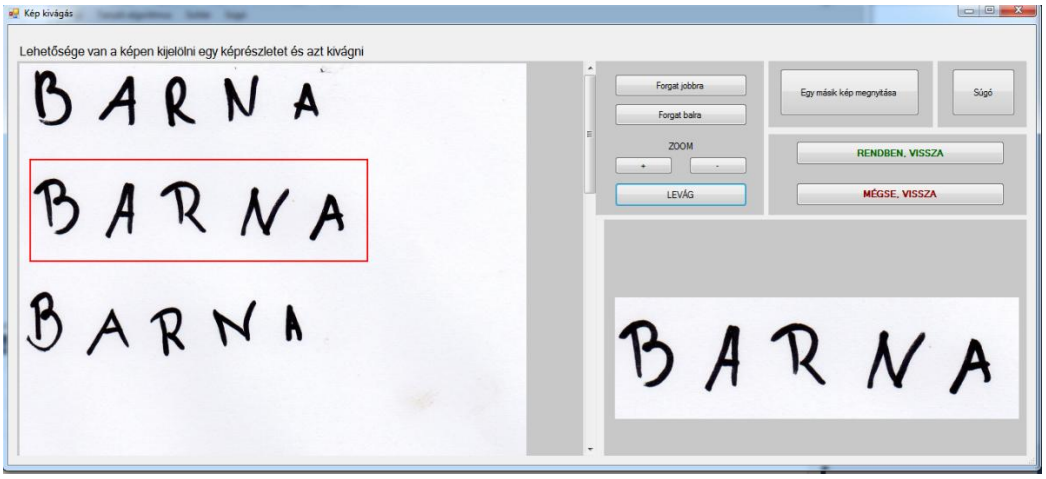
A program névjegye.

8.4.1.3 Gyorsbillentyűk

Billentyű kombináció	Funkciója
Ctrl+I	Kép megnyitása
Ctrl+W	Szót tartalmazó kép gyors megnyitása
Ctrl+L	Új neuronháló létrehozása
Ctrl+N	Egy neuronháló megnyitása
Ctrl+H	Magyar szótár betöltése
Ctrl+A	Angol szótár betöltése
Ctrl+D	Sajátszótár megnyitása
F1	Súgó megnyitása
Ctrl+Q	Kilépés

8.4.2 Képkivágás felület

Ebben az ablakban jelenik meg a kép, ha feltöltünk egy olyan képet, melynek egy részletét szeretnénk kivágni. Lehetőségünk van nagyítani, kicsinyíteni az adott képet a + és – gombokkal. Ha szükséges, akkor elfordíthatjuk. Az egérrel egy téglalapot kijelölve, majd a *LEVÁG* gombra kattintva az adott kijelölt részt levágja és megjeleníti azt. Ha ezt megismételjük, akkor az eredmény képen az aktuális kivágást látjuk, de lehetőségünk van egy másik képet is megnyitni, ha mégsem ezzel szeretnénk dolgozni. Ha a *RENDBEN, VISSZA* gombra kattintunk, akkor visszatérünk a fő ablakra, ahol egyből elkezdődik a kivágott képrészlet felismerése. Ezzel ellentétben a *MÉGSE, VISSZA* gombra kattintva a kiválasztott képet elvetjük, és visszatérünk a Fő ablakra.



42. ábra: Képkivágás felület

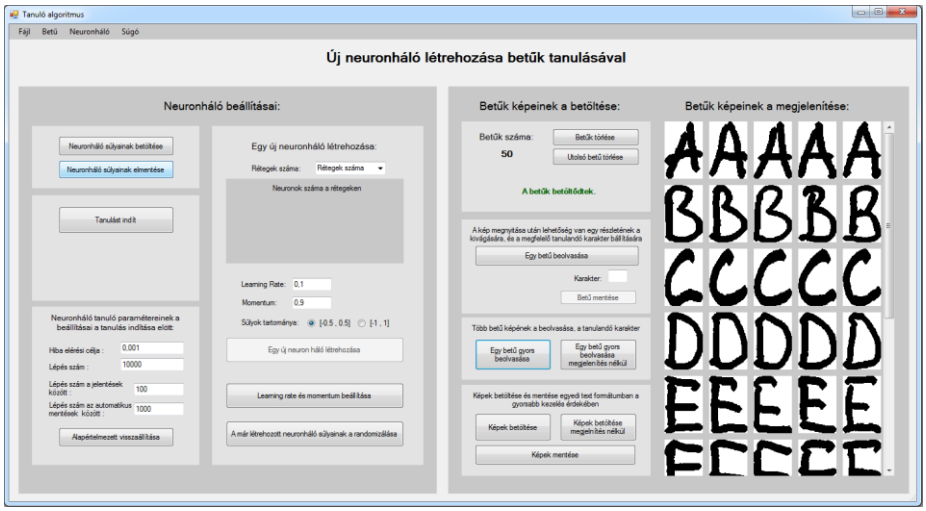
8.4.3 Tanuló algoritmus felület

8.4.3.1 Leírás

A szavak felismerése úgy történik, hogy a már előre létrehozott neuronháló az egyes bementekre megmondja, hogy az adott betű képe mely abc-beli karakterekre hasonlít a legjobban, amely után a szótár segítségével kiválasztja a betűkből alkotott legvalószínűbb szót. A neuronháló hasonlóképpen működik, mint az emberi agy. Ha például egy olyan embernek, aki soha életében nem látott mobiltelefont megmutatnánk néhányat, akkor megtanulná a jellegzetességeit. Így ha ezután egy más színű és méretű telefont mutatnánk neki, akkor nagy valószínűséggel felismerné. A neuronháló létrehozása végül is egy tanulási folyamat. Megadunk az abc betűihez mintaképeket, lehetőleg minden betűhöz többféle írásmód szerinti képet, és úgymond megtanulja, hogy az egyes betűknek mik a jellegzetességei. A tanulás után lekérdezhetjük tőle, hogy egy számára ismeretlen kép melyik betűre hasonlít a legjobban.

Ezen a felületen van lehetőség saját neuronhálót létrehozni. Ehhez nem kell mást tenni, mint feltölteni minta betűket és a neuronháló beállításait elvégezni, majd elindítani a tanulási folyamatot. A szófelismerő felületen ezt a létrehozott neuronhálót betöltve, már ezzel fogja a szavakat felismerni. Ez azért lehet hasznos, mert ha csak egy ember szavait akarjuk felismertetni, ekkor, ha az ő betűi közül töltünk fel minta képeket, akkor nagyobb valószínűséggel, pontosabban fogja felismerni a szavait. A szófelismerés felületen a *Tanuló algoritmus* menü *Új neuronháló létrehozása* menüpontjára kattintva juthatunk

erre a felületre, ahol lehetőségünk nyílik bővíteni, javítani a felismerő algoritmust, avagy egy teljesen újat létrehozni.



43. ábra: Tanuló algoritmus felület

8.4.3.2 Betűk kezelése

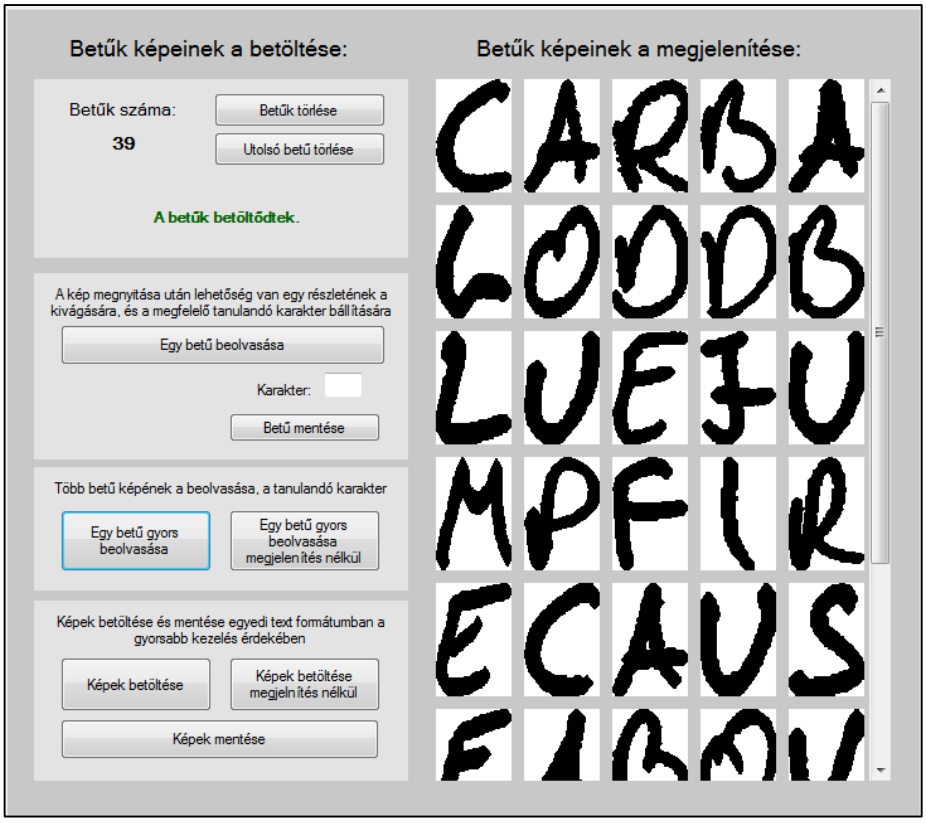
A betű képének a betöltésére hasonló lehetőségünk van, mint a szó betöltésére, egy nagyobb kép részletét kivágva töltjük be a betű képét, ilyen esetben meg kell adnunk a megfelelő karaktert is (*Egy betű beolvasása* és *Betű mentése*).

Lehetőség van egyéb módokon is betölteni a képeket, hogy megkönnyítse a felhasználók munkáját . Gyorsbetöltéskor a betöltendő állomány nevének első karaktere lesz a tanulandó kép abc–beli karaktere. Gyorsbetöltéskor lehetőség van olyan kép betöltésére is, amelyiken több, azonos karakterű betű van. Ilyen esetben egyszerre több fájlt is kilehet jelölni és betölteni a képeket.

A *betűk mentése* gombra kattintva a megnyitott, beolvasott betűket elmenthetjük a program által gyorsabban kezelhető szöveges formátumban, így a későbbiekben gyorsabb lesz a betöltése, vagyis adódik egy harmadik lehetőség is betű betöltésre: az, hogy már elmentett betűt, betűket nyissunk meg.

Az utóbbi két betöltési lehetőségkor, van rá opció, hogy anélkül töltsük be a képeket, hogy megjelenítenénk őket a felületen. Ez az miatt vált szükségessé, mert több ezer kép esetén nagymértékben nő a beolvasási idő.

A *betűk törlése* gombra kattintva kitörölhetjük a már betöltött betűket, avagy az *Utolsó betű törlése* gombra kattintva a legutolsó feltöltött betűt törölhetjük.



44. ábra: Tanuló algoritmus felület – Képek betöltése

8.4.3.3 Neuronháló paraméterezése

. A tanulás folyamán a program lépésről lépésre javítja a neuronhálót, így egyre jobban megtanulja az abc kézzel írott betűit. A javulást úgy éri el, hogy, mivel a tanulás folyamán a neuronháló tudja, hogy az egyes bemenetekre milyen kimenetet kellene, hogy kapjon, - ezt adtuk meg a betűk feltöltésével, - ezért megnézi, hogy a neuronháló az egyes betűkre milyen kimenetet ad, ezt összeveti az elvárt kimenettel, és a különbség lesz a neuronháló hibája. Ezt a hibát szeretnénk minimalizálni, így a neuronháló a kimenet függvényében módosítja a belső értékeit. Ez lesz egy tanulási lépés. Egy tanulási lépés számítás igénye egyenes arányban nő a tesztadatok számával, így a tanulás is annál lassabb lesz, minél több betűt tanítunk meg neki, de így jobb értéket kapunk majd a betűk felismerésekor. A tanulás folyamán a neuronháló hibája egyre kisebb lesz. A neuronháló tanítása előtt beállítható, hogy milyen hiba, vagy tanulási lépésszám elérése

esetén álljon le az algoritmus. Lehetőség van beállítani, hogy milyen lépés időközönként legyen az ablakban az aktuális hiba és a lépésszám frissítve, és hogy milyen sűrűn legyen automatikusan mentve.

Ez a tanulási folyamat elég sok időt vehet igénybe, de szerencsére ezt csak egyszer kell elvégezni, mert ha elkészül a neuronháló, akkor a későbbiekben a felismerés folyamán használhatjuk. Beállított automatikus mentésre azért lehet szükség, hogy ha netán egy áramkimaradás, vagy egyéb külső ok miatt bezáródik a program, akkor a több órás folyamatot ne kelljen elölről kezdeni. Ez egy biztonsági funkció.

Leállási paraméterek:

Egy adott hibaküszöb elérése:

Az az érték, ami alá, ha csökken a neuronháló hibája, akkor leáll a tanulási folyamat. Ennek nullánál nagyobb értéknek kell lennie. Ez azt jelenti, hogy a mintaképekre ekkora lesz a hibázási valószínűség.

Alapértelmezett érték: 0,001

Lépésszám elérése:

A tanulási folyamat egy lépés sorozat. Lépésről lépésre javul neuronháló, és ennek megadhatunk egy korlátot.

Alapértelmezett érték: 10.000

Megjelenési és mentési paraméterek:

Jelentések közti lépésszám:

Ennyi lépés után mindig jelez a felületen, hogy éppen mennyi a neuronháló hibája.

Alapértelmezett érték: 100

Automatikus mentések közti lépésszám:

Ennyi lépés után mindig elmenti az adott neuronhálót a program melletti *Neuralnetwork* mappába. Erre azért lehet szükség, mert a tanulási folyamat hosszú. Minél több minta betűre tanítjuk meg, annál lassabbak lesznek a lépések.

Alapértelmezett érték: 1000

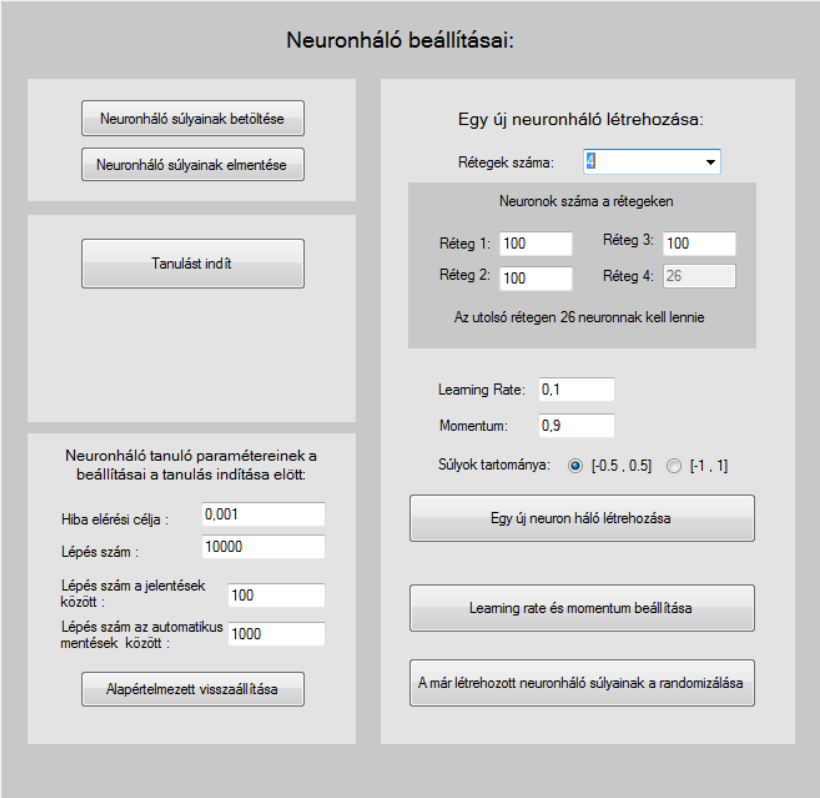
A leállási feltételek közül bármelyik teljesül, akkor abbahagyja a tanulást. Az ajánlott beállításokat alapértelmezetten betölti, így akik nem is ismeri a neuronhálók belső világát, azok is tudják a programot használni. Ezeket az alapbeállításokat később is vissza lehet állítani.

A betűk feltöltése és a beállítások elvégzése után, a *Tanulás* gombra kattintva elindul a tanuló algoritmus. A gyorsasága attól függ, hogy milyen sok betűt töltöttünk fel a tanuláshoz. A beállított megállási feltételek elérésekor kiírja, hogy milyen hiba arányt ért el. A megállási feltételek elérése után leáll a tanulás. Lehetőségünk van más paramétereket beállítva tovább tanítani, ha nem vagyunk elégedettek az eredménnyel. A tanulás elején érdemes kisebb lépésszámokat beállítani a paramétereknél, mert ha végez, akkor is folytathatjuk a tanulást. Akkor már nagyjából tudjuk, hogy egy lépés mennyi ideig tart, így számunkra kedvezőbb lépésszámokat adhatunk meg paraméterként.

Ha a neuronhálót el szeretnénk menteni, hogy a későbbiekben is használhassuk, megtehetjük a *Neuronháló súlyainak elmentése* gombra kattintva. A *Neuronháló súlyainak betöltése* gombra kattintva egy elmentett neuronhálót tudunk betölteni, és azt utána módosítani.

Lehetőség van egy új neuronhálót létrehozni különböző számú réteggel, és meglehet adni, hogy az egyes rétegek mennyi neuront tartalmazzanak. Továbbá a Tanulási faktort (Learning rate-et) és Mumentum változót is be lehet állítani a felületen, melyek azért felelősek, hogy milyen gyors legyen a tanulás. A tanulás alatt is van lehetőségünk ennek a két paraméternek a beállítására, még pedig a *Learning rate és momentum beállítása* gomb segítségével. A Neuronok tartományának a beállításánál két féle opció közül lehet választani, vagy „[-0.5,0.5]” vagy „[-1,1]”. A súly értékeket lehet menet közben is randomizálni anélkül, hogy megint létre kellene hozni a neurális hálót.

Ha bezárjuk a felületet, akkor a még el nem mentett neuronháló és képek elvesznek, és visszatérünk a Fő oldalra.



45. ábra: Tanuló algoritmus felület – Neuronháló felparaméterezése

8.4.4 Fájlformátumok

8.4.4.1 Képek

*.JPEG; *.JPG;*.BMP;*.PNG - Általános képformátumok

Neuronháló létrehozásánál a betűk feltöltésekor és felismerés felületen a szavak feltöltésekor kell használni

8.4.4.2 Szótár

*.XLS - Microsoft Excel fájl

A szavak felismerésekor, csak az adott szótárban található szavak közül keresi a legvalószínűbbet.

A szótár az adott nyelvben lehetséges szavakat tárolja az első oszlopban, de lehetnek több lapon is az Excel fájlon belül. Hozzáadható tetszőleges olyan nyelvnek a szótárja, amely az angol abc betűit használja fel. A program fel lett készítve a magyar ékezetes betűkre is, így azok is használhatóak.

Beépített szótárak:

- Angol: ~350.000 szót tartalmaz
- Magyar: ~30.000 szót tartalmaz

8.4.4.3 Neuronháló

*.NNW - (NeuronNetworkWeights)

Saját fájlformátum a neuronháló súlyainak az eltárolására. Speciális elrendezésű szöveges fájl, ezért külső módosítása elronthatja a neuronhálót.

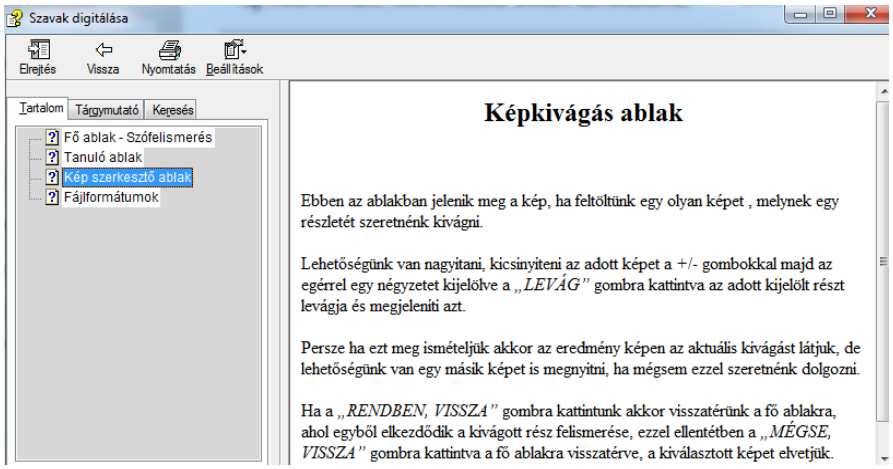
8.4.4.4 Képmátrixok

*.LTF (LeTterFile)

Betű képeinek az eltárolására, a gyorsabb beolvasás érdekében. A beolvasott betűk képei helyett a képek számunkra érdekes részeit tároljuk csak el. Ha már beolvastuk a *Tanuló algoritmus* felületen a betűket, akkor lehetőség van ezeket egy speciális elrendezésű szöveges fájlba elmenteni. A későbbiekben ennek a beolvasása már nagyon gyorsan történik.

8.4.5 Súgó

Az egyes felületeken a súgó gombra illetve menüpontra kattintva az adott felülethez tartozó segítség jelenik meg. Lehetőség van keresni egy kulcsszóra.

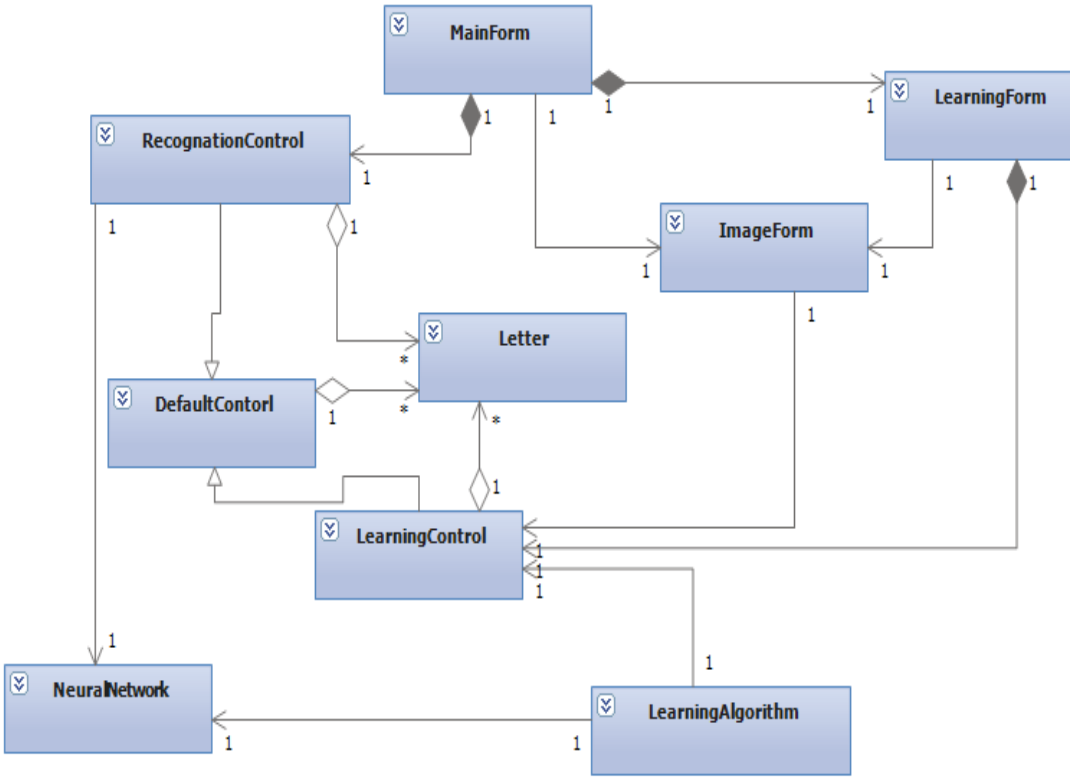


46. ábra: Súgó felülete

9 Fejlesztői dokumentáció

A Microsoft által készített, .NET keretrendszerben Framework 3.5 –ös szoftverfejlesztői platformon készült a program, C# programnyelven.

9.1 Osztálydiagram



47. ábra: Osztály diagram

9.2 Algoritmusok

9.2.1 Képek előfeldolgozása:

Lásd: ImageEditor osztály függvényei,68. oldal

- <u>Túl nagy felbontású képeket tömöríti</u>

  Ha a feltöltött kép egy adott felbontásnál nagyobb egy adott méretnél, akkor kisebbe tömöríti a gyorsabb feldolgozás érdekében.

- Medián szűrő

  Különböző zajok eltávolítására alkalmazzuk. A különböző zajszűrők közül azért ezt alkalmaztuk, mert így nem mosódnak el az élek.

- Bináris kép készítése

  A gyorsabb feldolgozás érdekében nem magával a képpel számolunk a későbbiekben, hanem a pixeleiből előállított kétdimenziós tömbbel. A betűkhöz tartozó pixelek helyén egyesek állnak, a háttér pixeleinek a helyén pedig nullák.

- Kicsinyítés. nagyítás (zoom)

  Képek beolvasásakor lehetőség van nagyítani, vagy kicsinyíteni a képet.

- Forgatás

  Képek beolvasásakor lehetőség van elforgatni 90 fokkal valamilyen irányba a képet.

- Objektumok bővítése

  A betűt egybefüggő objektumként ismeri fel, de előfordul, hogy úgy írjuk a betűt, hogy nem összefüggő, pl.: a ’T’ betű vízszintes teteje és a függőleges szára nem ér össze. Ekkor alkalmazzuk ezt, hogy bővítsük az objektumokat. Ezt úgy lehet elképzelni, mintha a betűk vonalait megvastagítanánk. Bővítések után már lehet, hogy összefüggővé válik.

- Szegmentálás, a betűk megtalálása és szeparálása

  Rekurzív függvény segítségével összefüggő fekete pixeleket keresünk, ezek az összefüggő területek a betűk.

9.3 Fontosabb függvények, adatszerkezetek, megvalósítások

Szálkezelés:

A MainFormban és a LearningFormban jelenik meg

Azokon a helyeken, ahol hosszabb ideig kell várakozni az eredményre, szálkezeléssel oldottam meg, hogy ne tűnjön úgy, mintha lefagyott volna a program.

Ilyen volt a szófelismerés, ami pár másodperctől egészen egy percig is eltarthat a kép nagyságától függően. A másik eset a tanulás folyamata, ez már lényegesen hosszabb, akár órákig is tartó folyamat, és szálkezelés nélkül nem lehetne eldönteni a folyamatról, hogy számol a háttérben vagy pedig lefagyott.

Szükség van a külön futó szálon a futás közbeni információ átadására a felhasználó felé. A felismeréskor a változtatott képek megjelennek. A tanulási folyamatnál az éppen aktuális hibaarányt írja a beállított jelentések közti lépésszámonként.

Ha egy másik szálból szeretnél elérni egy UI vezérlőt, akkor az adott vezérlőnek az Invoke() metódusát kell meghívni.

Szükséges lehet leállítani egy folyamatot, ez leginkább a tanuláskor fordulhat elő, mert az tart hosszabb ideig. Lehet, hogy meggondoljuk magunkat és mégsem szeretnénk megvárni a tanulás végét, így erre ott lehetőség van.

9.4 Osztályok leírása

9.4.1 MainForm

Feladata:

A felhasználó által a *szófelismerő* felületen kiváltott események kezelése.

Függvények és feladataik:

private void **showResult**()	Ez a függvény felelős a betű és szó találatok megjelenítéséért
private void **showImages**(List<Bitmap> letters)	Paraméterül kapott képeket jeleníti meg a felületen egy panelon
private void **startRecognition**(Bitmap image)	A paraméterül kapott képre elindítja egy külön szálon a felismerő algoritmust.
private void **recognition**()	Ez a függvény van külön szálon indítva, ez felelős a felismerő algoritmus működéséért.
private void **showAlert**()	Hibaüzenetek megjelenítése
private void **newNNToolStripMenuItem_Click** (object sender, EventArgs e)	LearningForm, neuronháló létrehozó vagy módosító felület megnyitása.
private void **NNOpenToolStripMenuItem_Click** (object sender, EventArgs e)	Egy neuronhálót lehet betölteni fájlrendszerből.
private void **baseNNLoadToolStripMenuItem_Click** (object sender, EventArgs e)	Alapértelmezett neuronháló betöltése.
private void **EnglishLoadToolStripMenuItem_Click** (object sender, EventArgs e)	Angol szótár betöltése.
private void **HungaryLoadToolStripMenuItem_Click** (object sender, EventArgs e)	Magyar szótár betöltése.
private void **dictionaryOpenToolStripMenuItem_Click** (object sender, EventArgs e)	Fájlrendszerből beállítható egy szótár Excel fájlja.
private void **wordImageOpenImageButton_Click** (object sender, EventArgs e)	Egyetlen a szót tartalmazó kép feltöltése. Ezután elindul a felismerő algoritmus
private void **openImageButton_Click** (object sender, EventArgs e)	Egy kép feltöltése után a megnyíló ImageForm-on, a kép átalakító felületen lehetőség van egy megfelelő képrész kivágására,

	miután visszatérve elkezdődik a szó felismerése.
private void **reset**()	A hiba és információ kiírások elrejtése; az eredmények törlése a felületről

9.4.2 ImageForm

Feladata:

A felhasználó számára biztosítani, hogy egy képről egy részletét kivághassa. Ez a képkivágás felület osztálya.

Függvények és feladataik:

private void **openPictureBox_MouseDown**, **openPictureBox_MouseMove**, **openPictureBox_MouseUp** (object sender, EventArgs e)	Egérrel való kijelölés kezelése
private void **largerButton_Click**, **smallerButton_Click** (object sender, EventArgs e) private void **refreshImage**()	A kép nagyítása és kicsinyítése
private void **rotateLeftButton_Click, rotateRightButton_Click** (object sender, EventArgs e)	A kép forgatása jobbra és balra
private void **cutImageButton_Click** (object sender, EventArgs e)	A kép egy részének a levágása
private void **openButton_Click** (object sender, EventArgs e)	Egy új képfájl megnyitása
private void **helpButton_Click** (object sender, EventArgs e)	Súgó megjelenítése
private void **returnButton_Click** (object sender, EventArgs e)	Kilépés és a kivágott kép használata
private void **cancelButton_Click** (object sender, EventArgs e)	Kilépés és a kép elvetése

9.4.3 ImageEditor

Feladata:

Különböző képmanipulációs eljárások gyűjteménye.

Függvények és feladataik:

public Bitmap **MedianFilter** (Bitmap Image, int Size)	Átlagoló szűrő a zajok eltávolítására
public Bitmap **expansion** (Bitmap Image, int size)	Az objektumok bővítése, vastagítása képen
public int[,] **expansion** (int[,] Image, int size)	Az objektumok bővítése, vastagítása egy bináris mátrixon
public int[,] **bitArrayFromImage** (Bitmap picture)	Bit mátrix készítése egy képből
public Bitmap **resizeImage** (Bitmap imgToResize, Size size)	Kép átméretezése egy bizonyos méretre
public Bitmap **ImageFromBitarray** (int[,] imgMtx)	Fekete-fehér kép készítése egy mátrixból
public Bitmap **compress** (Bitmap modImage,int height)	Kép kicsinyítése egy adott magasságra
public Bitmap **cutWhite** (Bitmap modImage)	Alulról, felülről és oldalakról levágja a fehér részeket a képen
public int[,] **cutWhite** (int[,] tempMtx)	Alulról, felülről és oldalakról levágja a fehér részeket a bináris mátrixon
public Bitmap **rotation**(Bitmap bmp, int degree)	Kép forgatása jobbra és balra
public int[,] **biggerMatrix**(int[,] inputMtx, int size)	Mátrix bővítése mind a négy oldalról egy bizonyos értékkel
public int[,] **biggerMatrixForI**(int[,] inputMtx)	Mátrix bővítése az ’i’ betű miatt, a torzítás elkerülése érdekében
public int[,] **setCommonSize**(int[,] inputMtx,int min, int max)	Egységesíti a betűket a fekete pixelek arányának a segítségével
private float **pixelRate**(int[,] inputMtx)	Vissza adja a fekete pixelek arányát a képen

9.4.4 LearningForm

Feladata:

A felhasználó számára biztosítani új neuronháló létrehozását, módosítását és a hozzá tartozó képek feltöltését. Ez a tanuló felület osztálya.

Függvények és feladataik:

private void **openLetterButton_Click** (object sender, EventArgs e)	Kép megnyitása egy betű feltöltésére
private void **multiOpenImageButton_Click** (object sender, EventArgs e)	Csak a betűket tartalmazó képek betöltése, a fájlnak az első karaktere a tanulandó abc-beli betű
private void **openLettersNoLoadImg Button_Click** (object sender, EventArgs e)	Csak a betűket tartalmazó képek betöltése megjelenítés nélkül
private void **openLettersButton_Click** (object sender, EventArgs e)	Betűk gyors beolvasása egy korábbi mentésből
private void **saveLettersButton_Click** (object sender, EventArgs e)	Betűk képeinek a mentése szöveges formátumban későbbi felhasználás miatt
private void **deleteLettersButton_Click** (object sender, EventArgs e)	Kitörli a már beolvasott betűket és képeiket
private void **deleteLastLetterButton_Click** (object sender, EventArgs e)	Az utoljára beolvasott betűt törli
private void **showLetters**()	A már beolvasott betűk képeinek a megjelenítése
private void **showErrorNetwork**(string error)	A neuronhálóhoz tartozó hibák kiírása
private void **showErrorLetter**(string error)	Betűk feltöltésével kapcsolatos hibák kiírása
private void **showInfoNetwork**(string info)	A neuronhálóhoz tartozó információ kiírása
private void **showInfoLetter**(string info)	Betűk feltöltésével kapcsolatos információk kiírása
private void **defaultButton_Click** (object sender, EventArgs e)	Az alapértelmezett neuronháló súlyainak a betöltése
private void **networkLoadButton_Click** (object sender, EventArgs e)	Egy neuronháló súlyainak a betöltése
private void **networkSaveButton_Click** (object sender, EventArgs e)	A neuronháló súlyainak az elmentése
private void **neuronStartButton_Click** (object sender, EventArgs e)	A tanulási folyamat elindítása egy új szálon. A tanulási folyamat megállítása.
private void **startLearningThread**()	A tanulási folyamat elindítása egy külön szálon
public void **processReport**(double error)	Tanulás közben az aktuális lépésszám és hiba kiírása
public void **processEnd**(double error)	Jelzés a tanulási folyamat sikeres befejezéséről a felhasználó felé
private void **randNeuronWeightButton Click** (object sender, EventArgs e)	Neuronháló neuronjainak a súlyértékeit random értékekkel tölti fel
private void **setLearningrateAnd MomentumButton_Click** (object sender, EventArgs e)	Neuronháló két paraméterét beállítja: Learning rate és momentum

9.4.5 LearningControl : DefaultControl

Feladata:

Kapcsolatot teremt a LearningForm és a LearningAlgorithm között. Elvégzi a képek feldolgozását, a betűk mentését és a neuronháló módosítását.

double[][] **input**;	Betűkből készített neuronháló bemenet
double[][] **output**;	Betűkhöz tartozó neuronháló kimenet
double **goal**;	Megállási hiba arány
int **report**;	Jelentések közti lépésszám
int **save**;	Mentések közti lépésszám
List<Letter> **letterList**;	Betűket tartalmazó lista

Függvények és feladataik:

public void **loadNeuralNetwork** (string filePath)	Neuronháló betöltése egy fájlból
public void **saveNeuralNetrwork** (string filePath)	Neuronháló mentése egy fájlba
public void **addLetterFromFiles** (string[] fileNames)	Több kép fájl beolvasása és betűk mentése
public void **addLetter** (Image img,string inputCharacter)	Egy betű mentése a paraméterül kapott képpel és karakterével.
public void **loadLetters**(string filePath)	Betűk gyors betöltése szöveges fájlból
public void **saveLetters**(string filePath)	Betűk elmentése szöveges fájlba
public List<Bitmap> **getImages**()	A már feltöltött betűk képeinek a lekérdezése
public void **deleteLetters**()	Betűk törlése
public void **deleteLastLetter**()	Az utoljára feltöltött betű törlése
private int **getOutputIndex**(string fileName)	A fájlnévből meghatározza a betű tanulandó karakterét (a fájlnév első betűje).
public double **learning**()	Tanulási folyamat elindítása
public void **init** (double goal,int report,int save)	Tanulási folyamat elindítása előtti paraméter beállítások
protected List<Letter> **getAndSave LettersFromImages**(int[,] binImage, bool learning)	Beolvassa a betűket egy képekről
private void **search**(int i, int j, int number)	Megkeresi az obejektumokat a képről
protected double[] **getInput**(int[,] letterMtx)	Elkészíti a neurális hálózat bemeneti vektorát egy kép mátrixából

9.4.6 LearningAlgorithm

Feladata:

AForge.Neuro.dll segítségével fölépíti a neuronhálót, és a visszacsatolásos tanulási folyamatot elindítja. Lekérdezhető egy inputra a neuronháló kimenete.

private ActivationNetwork **network**;	Neuronháló
private BackPropagationLearning **teacher**;	Tanuló algoritmus

const int **Layer1** = 100;
const int **Layer2** = 100;
const int **Layer3** = 26;
	A rétegekben található neuronok száma

const double **defaultMomentum** = 0.9;	Momentum állandó
const double **defaultLearningRate** = 0.1;	Tanulási sebesség állandó
int **stepCount**;	Lépés szám
int **saveNN**;	Mentések közti szám

Függvények és feladataik:

public **LearningAlgorithm**(int input_count)	Konstruktorban definiáljuk a tanulási algoritmust. A neuronok súly értékeit véletlenszerű értékekkel feltöltjük
public void **newLearningAlgorithm** (int[] layers, double learningRate, double momentum, bool randomWeight)	Létrehoz egy új neurális hálózatot a megadott paraméterekkel
public void **updateLearningAlgorithm** (double learningRate, double momentum)	Beállítja a neuronhálóhoz a learning rate-et és a momentum-ot
public void **randWeight**(bool weight)	Random értékekkel állítja be a neuronháló súlyainak az értékeit
private double **RandomNumber**()	double random számgenerátor
public void **save**(string filePath)	Neuronháló súlyainak mentése szöveges fájlba
public void **load**(string filePath)	A neuronháló súlyainak a betöltése szöveges fájlból
public double **learn**(double[][] x,double[][] y, double goal, int epoch)	A tanulási folyamat elindítása
public double[] **computeLetter**(double[] inp)	Egy bemenetre kiszámítja a neuronháló kimenetét

9.4.7 RecognitionControl : DefaultControl

Feladata:

A szó felismeréshez szükséges függvények biztosítása.

SortedList<double, string> **wordsHit**;	Végső találat a szavakra
List<Letter> **word**	Betűk listája
List<DataSet> **dataList**	A szótár használatához
double[] **maxHit**;	Lépések legnagyobb valószínűségi találatai
SortedList<double, string>[] **wordsHits**;	Lépések találatai a szavakra
List<Letter>[] **words**;	Lépéseknél felismert betűk

Függvények és feladataik:

public Bitmap **initRecognition**(Bitmap image)	A felismeréshez szükséges előkészületeket elvégzi
public bool **recognition**()	A szófelismerő eljárás váza, mely több lépésben dolgozik, és akkor tér vissza igazzal, ha már végzett, különben hamissal
public List<Bitmap> **getLetters**()	Az aktuális szóhoz tartozó betűk képei listájának a visszaadása
private double **getMaxHit**(Bitmap bmp)	Kiszámítja a lehetséges szavakra a valószínűségeiket és visszaadja a legnagyobbat
public void **recognition**()	Kiszámítja a neuronháló segítségével az egyes betűk képeire az abc-beli karakter valószínűségét
private void **searchWord**()	Kiszámítja a lehetséges szavakra a valószínűségeiket
private **List**<string> getWordsFromDictionary()	Megkeresi a szótár(Excel fájl) segítségével a lehetséges szavakat
private char **getEnglishMatch**(char c)	Magyar ékezetes karaktereknek megadja az angol ékezetmentes megfelelőit.
public char[][] **getCharHitOrder**()	A képen található betűk mindegyikére visszaadja a 4 legvalószínűbbet
public double[][] **gethitsPercentage**()	A képen található betűk mindegyikére visszaadja a 4 legvalószínűbbnek a valószínűségi értékét
public **SortedList**<double, string> getWords()	Megadja a legvalószínűbb szavakat a valószínűségeikkel együtt.
public void **loadNeuralNetwork**(string fileName)	Egy neuronháló betöltése, vagy alapértelmezett, vagy egy külső fájlból

public void **setDictionary**(int dictionary)	Az alapértelmezett szótár beállítása
public void **setDictionary**(string dictionaryFile)	A szótár beállítása egy külső fájlból
private void **loadDictionary**()	A szótár beolvasása egy külső szöveges fájlból

9.4.8 DefaultControl

Feladata:

RecognitionContorl és a LearningControlban is szükséges függvények, változók implementálása.

protected ImageEditor **imgEditor**;	Szerkesztő osztály
protected const string **abc** = "abcdefghijklmnopqrstuvwxyz";	Az angol abc betűinek az elmentése
protected const int **width** = 60;	Algoritmusban szereplő betűk magassága
protected const int **height** = 90;	Algoritmusban szereplő betűk szélessége
protected const string **defaultNNWeights** = "./networkWeight/default.nnw";	Az alapértelmezett neuronháló helye
protected Bitmap **inputImage**;	Input kép eltárolására
protected int[,] **binInputImage**;	Az input képből készített mátrix eltárolására.

Függvények és feladataik:

protected List<Letter> **getAndSaveLetters** (int[,] binImage)	Egy szó képének a bináris mátrixából megkeresi a betűket és eltárolja.
private void **search**(int i, int j, int number)	Rekurzív függvény, mely a képen összefüggő objektumokat keres.
private Size **getLetterSizeAvarage** (List<Letter> letters)	Megadja, hogy mennyi betűk átlag mérete, szélessége és magassága.
protected double[] **getInput**(int[,] letterMtx)	Egy betű mátrixából elkészíti a neuronháló bemenetét.

9.4.9 Letter

Feladata:

Betűkhöz tartozó információk elmentése

const string **abc** = "abcdefghijklmnopqrstuvwxyz"	Az angol abc betűinek az elmentése
bool **accent**	Az, hogy ékezetes-e
Rectangle **rectangle**	Az eredeti képen található helye
int[,] **matrix**	A kép mátrixa
int **outputInd**	Tanuláskor a hozzátartozó karakter sorszáma az abc stringben
double[] **result**	A neuronháló output vektorja erre a képre
char[] **hits**	A 4 legvalószínűbb abc-beli karakter erre a betűre
double[] **hitPercentage**	A 4 legvalószínűbb abc-beli karakter valószínűségei erre a betűre

Függvények és feladataik:

public void **calculateMaxs**()	Kiszámítja a neuronháló kimeneti vektorjából a 4 legvalószínűbbet

9.5 AForge.Net keretrendszer

AForge.NET egy nyílt forráskódú C# keretrendszer fejlesztőknek és kutatóknak. Számítógépes Látás és Mesterséges Intelligencia területivel foglalkozik: képfeldolgozás, neurális hálózatok, genetikus algoritmusok, fuzzy logika, gépi tanulás, robotika, stb. [10]

A következő internetes oldalról letölthető: http://www.aforgenet.com/framework/

A program az AForge.NET keretrendszer segítségével készíti el a neuronhálót és tanulási folyamat is ennek segítségével történik, az *AForge.Neuro.dll* könyvtár függvényeit használja. Emiatt van szükség az *AForge.Neuro.dll* és *Aforge.dll* állományokra a *Handwriting recognition* mappában.

BackPropagationLearning **teacher**	A tanuló algoritmus
ActivationNetwork **network**	Maga a neuronháló felépítése
network.Layers[i]	Neuronháló i. rétege
network.Layers[i].Neurons[j]	Neuronháló i. rétegének j. neuronja
network.Layers[i].Neurons[j].Weights[k]	Neuronháló i. rétegének j. neuronjának k. súlyértéke

Függvények és feladatai:

public double[] **Compute(**double[] inp**)**	Kiszámítja az input vektor kimenetét
public double RunEpoch (double[][] x, double[][] y)	Egy tanulási lépés a neuronhálón. Két paramétere van, a neuronháló inputja és az erre elvárt kimeneti vektorok.

9.6 Tesztelés

Fekete doboz tesztelés tesztesetei:

- A szó és betű feltöltésnél a képek tesztelése:
  - Hibás fájlformátum megadása
  - Sötét képek
  - Nagyon világos képek
  - Nagy felbontású képek
  - Szó feltöltésekor nagyon sok betűt tartalmazó kép
- Neuronháló paraméterei: (nullánál nagyobb számokat vár)
  - Számtól különböző karakter megadása
  - Negatív szám megadása
  - 0 megadása
  - Hiányos paraméter megadása
  - Lépés számoknál csak egész számot lehet megadni, tört szám megadása
- Betű képének a feltöltésekor, amikor a hozzá tartozó betűt meg kell adni:
  - Üres mező
  - [a-z] –től különböző karakter
  - Több karakter
- Betű képének a gyorsbetöltésekor, amikor a kép nevének első karaktere lesz a tanulandó betű:
  - [a-z] –től különböző karakter
- Neuronháló betöltésekor külsőleg módosított neuronháló fájl megadása a szófelismerő és a tanuló felületen:
  - Üres fájl
  - Hiányos fájl
  - Fájlban található értékek átírása
- A betű elmentett képeinek a betöltésekor külsőleg módosított fájl megadása a tanuló felületen.
  - Üres fájl
  - hiányos fájl
  - Fájlban található értékek átírása
- Szótár

    - Üres szótár megadása
    - Nagy szótár megadása
- Hiányzó alapértelmezett neuronháló és szótár fájlok

Tesztesetek:

- Mind a szó, mind a betű feltöltésekor nagy felbontású képek használata.
- Betű feltöltésekor a karakter megadásánál hibás karakter megadása.
- Neuronháló paramétereinél hibás paraméterek megadása.
- Neuronháló betöltésekor külsőleg módosított neuronháló fájl megadása a szófelismerő és a tanuló felületen.
- A betűk elmentett képeinek a betöltésekor külsőleg módosított fájl megadása a tanuló felületen.

Sebesség teszt:

A neuronháló tanításának az ideje attól függ, hogy mennyi képet töltünk fel a tanuláshoz. Jobb tanulási eredményt érhetünk el, ha egy abc-beli betűhöz több hozzá tartozó képet töltünk fel.

Tanulási idő tesztelésekor azt néztem, hogy X ábécényi képre, vagyis ha minden betűhöz X db hozzátartozó képet töltök fel, akkor 100 lépést mennyi másodperc alatt ér el.

- 1 ábécényi betűre (26 input): 4 mp
- 2 ábécényi betűre (52 input): 7 mp
- 3 ábécényi betűre (78 input): 10 mp
- 4 ábécényi betűre (104 input): 12 mp
- 5 ábécényi betűre (130 input): 15 mp
- 10 ábécényi betűre (260 input): 29 mp
- 20 ábécényi betűre (520 input): 64 mp

Az eredményekből jól látszik, hogy a tanulási folyamat nagyon időigényes folyamat.

Eredmények alapján elmondható, hogy az input számával egyenes arányban nő a tanulás ideje. Általában több tízezer lépés is szükséges lehet, hogy a neuronháló hibáját megfelelően kicsire csökkentsük.

9.7 Továbbfejlesztési lehetőségek

Eredményeim alapján a következő tovább fejlesztési lehetőségeket látom a rendszerben.

Ha speciálisan magyar nyelvre szeretnénk használni, akkor egy bővebb szótár lehet, hogy pontosabb eredményt tudna adni, hogyha pedig más nyelvekre is szeretnénk használni, akkor a felismerendő karaktereket az angol abc betűiről kilehetne bővíteni, saját szimbólumokat lehetne felvinni, mint például különböző nyelvek karaktereit, számokat, matematikai és nyelvtani jeleket.

Egy neuronhálótól független fejlesztési lehetőség lenne egyszerre több szavas, egész oldalas szövegek digitalizálása. Az összeért betűket különböző vágási algoritmusokkal szét lehetne vágni, így használhatóbbá téve a programot.

Egy nagy tovább fejlesztés lenne a folyószöveggel írott szavak digitalizálása, mert még kevésbé érvényesülnek az írás szabályosságai, még jobban előjönnek az egyes emberek sajátos írásformája. A másik nehézség, hogy a betűk a szavakban nincsenek szeparálva, és nehéz meghatározni az egyes betűket.

9.8 Verziójegyzék

A következő iterációs lépéseken keresztül érte el a program a jelenlegi formáját.

1, Képekhez tartozó átalakító függvények megírása

2, Neuronháló felépítése és a képekből neuronháló bemenet elkészítése

3, Betűk felismerése neuronhálóval

4, Szótár alkalmazása

5, Ékezetek kezelése

6, Neurális hálózat paraméterezhetősége

6, Felületek felhasználóbaráttá tétele, súgó készítése

10 Ábrajegyzék

1. ábra: Szavak megtalálása egy képen [2] 8
2. ábra: Fúziós megoldás [3] 9
3. ábra: Pocizió-szélesség-impulzus algoritmus [4] 9
4. ábra: Egy neuron felépítése 11
5. ábra: Egy neuron felépítése 12
6. ábra: Küszöb függvény képe 13
7. ábra: Szakaszonként lineáris függvény képe 14
8. ábra: Szigmoid függvény képe 14
9. ábra: Visszacsatolt neurális hálózat [5] 15
10. ábra: Előre csatolt neurális hálózat 16
11. ábra: Saját neurális hálózat 20
12. ábra: Kép felosztása cellákra [1] 22
13. ábra: A Z43 –as cella felosztása 22
14. ábra: Medián szűrőre egy példa 25
15. ábra: Medián szűrők eredménye (a) Input kép; (b) 3x3-as szűrő; (c) 7x7-es szűrő 26
16. ábra: Medián szűrők eredménye bináris képpé alakítás után (a) Input kép; (b) 3x3-as szűrő; (c) 7x7-es szűrő 26
17. ábra: Bináris képpé alakítás (a), (c) Input kép; (b), (d) Bináris kép 27
18. ábra: Betűvastagság egységesítése (a) Eredeti kép; (b) Egységesítés utáni kép 28
19. ábra: Üres részek levágása és átméretezés (a) Eredeti kép; (b) Üres részek levágása utáni; (c) Átméretezés utáni 28
20. ábra: ’I’ betű különleges kezelése (a) Eredeti kép; (b) Üres részek levágása és átméretezés utáni kép; (c) Átméretezéskor figyelve, hogy ne torzuljon el a képe 29
21. ábra: Egy személytől kért tanító kép az ABC első 3 betűjére 30
22. ábra: Pár példa a CSV tesztadatból kinyert képekre 31
23. ábra: Angol szótárban található szavak aránya a hosszra nézve 35
24. ábra: Google Dokumentumok működése 38
25. ábra: Saját neuronháló összehasonlítása a Google Dokumentumokkal 40
26. ábra: Személyek szerinti hibázások a szavak felismerésénél 41

27. ábra: Betűk hibázásának az aránya 43
28. ábra: ’D’ betűt ’O’ betűnek ismerte fel 44
29. ábra: ’O’ betűt ’D’ betűnek ismerte fel 45
30. ábra: ’U’ betűt ’V’ betűnek ismerte fel 45
31. ábra: ’V’ betűt ’U’ betűnek ismerte fel 45
32. ábra: Egy példa, hogy egy embernél mennyire hasonló lehet az ’U’ és ’V’, egy szón belül is : „VARIOUS” 45
33. ábra: ’M’ betűt ’H’ betűnek ismerte fel 45
34. ábra: ’L’ betűt ’C’ betűnek ismerte fel 46
35. ábra: ’E’ betűt ’G’ betűnek ismerte fel 46
36. ábra: ’O’ betűt ’P’ betűnek ismerte fel 46
37. ábra: ’V’ betűt ’Y’ betűnek ismerte fel 46
38. ábra: Hibásan felismert betűk: (a),(b) ’O’ →’Q’; (c) ’U’ →’Q’; (d) ’D’ →’P’; (e) ’F’ →’P’; (f) ’G’ →’C’; (g) ’J’ →’S’ 47
39. ábra: Hibásan felismert betűk: (a),(b) ’M’ →’N’; (c)(d) ’N’ →’U’; (e) ’N’ →’W’; (f),(g) ’R’ →’C’ 47
40. ábra: Hibásan felismert furcsa betűk 47
41. ábra: Fő felület 53
42. ábra: Képkivágás felület 56
43. ábra: Tanuló algoritmus felület 57
44. ábra: Tanuló algoritmus felület – Képek betöltése 58
45. ábra: Tanuló algoritmus felület – Neuronháló felparaméterezése 61
46. ábra: Súgó felülete 62
47. ábra: Osztály diagram 63

11 Irodalomjegyzék

[1] J.Pradeep, E.Srinivasan és S.Himavathi : Diagonal based feature extraction for handwritten alphabets Recognation system using neural network, International Journal of Computer Science & Information Technology (IJCSIT), Vol 3, No 1, Feb 2011

[2] Manmatha R., Srimal N. (1999) Scale Space Technique for Word Segmentation in Handwritten Documents. In: Nielsen M., Johansen P., Olsen O.F., Weickert J. (eds) Scale-Space Theories in Computer Vision. Scale-Space 1999. Lecture Notes in Computer Science, vol 1682. Springer, Berlin, Heidelberg

[3] Chakraborty R., Sil J. (2005) Handwritten Character Recognition Systems Using Image-Fusion and Fuzzy Logic. In: Pal S.K., Bandyopadhyay S., Biswas S. (eds) Pattern Recognition and Machine Intelligence. PReMI 2005. Lecture Notes in Computer Science, vol 3776. Springer, Berlin, Heidelberg

[4] Nusratov, O.K. & Geydarov, P.S. : Position-Width-Pulse Algorithm for Recognition of Handwritten Characters, Automatic Control and Computer Sciences, 2007, Vol. 41,

[5] http://project.mit.bme.hu/mi_almanach/books/neuralis/ch08s02, 2018.04.01

[6] Altrichter Márta, Horváth Gábor, Pataki Béla, Strausz György, Takács Gábor, Valyon József: Neurális hálózatok, Hungarian Edition Panem Könyvkiadó Kft, 2006 [433], ISBN-10 9-635454-64-3, ISBN-13 978-9-635454-64-8

[7] https://www.kaggle.com/ashishguptajiit/handwritten-az, 2018.03.12

[8] https://www.google.hu/intl/hu/docs/about/, 2018.04.15

[9] https://hu.wikipedia.org/wiki/Mesters%C3%A9ges_neur%C3%A1lis_h%C3%A1l%C3%B3zat, 2018.05.01

[10] http://www.aforgenet.com/framework/, 2017.08.02