

BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND INFORMATICS
SPECIALIZATION: COMPUTER SCIENCE

License Thesis

Road sign recognition

Abstract

There are numerous methods for road sign recognition but a robust and cost-effective solution is still an active research subject. There are several factors hindering the effective recognition of traffic signs, like luminosity of the image, partial coverage, shading, and other obstacles. We aim for a system capable of recognising road signs using machine learning techniques taking in consideration the above mentioned factors. The most important is accuracy and speed. The application displays the examined image and analyses it on background threads. The program extracts random rectangles from the original picture and decides if it contains a traffic sign or not, and if it does, it categorises them.

This work is the result of my own activity. I have neither given nor received unauthorized assistance on this work.

JULY 2015

SZABÓ ÁGNES-TERÉZ

ADVISOR:
CSATÓ LEHEL BABEŞ-BOLYAI UNIVERSITY,
FACULTY OF MATHEMATICS AND INFORMATICS

BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND INFORMATICS
SPECIALIZATION: COMPUTER SCIENCE

License Thesis

Road sign recognition



SCIENTIFIC SUPERVISOR:

CSATÓ LEHEL BABEŞ-BOLYAI UNIVERSITY,
FACULTY OF MATHEMATICS AND INFORMATICS

STUDENT:

SZABÓ ÁGNES-TERÉZ

JULY 2015

UNIVERSITATEA BABEŞ-BOLYAI, CLUJ-NAPOCA
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
SPECIALIZAREA INFORMATICĂ

Lucrare de licență

Recunoașterea semnelor de circulație



CONDUCĂTOR ȘTIINȚIFIC:

CSATÓ LEHEL
UNIVERSITATEA BABEŞ-BOLYAI,
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ

ABSOLVENT:

SZABÓ ÁGNES-TERÉZ

IULIE 2015

BABEŞ-BOLYAI TUDOMÁNYEGYETEM KOLOZSVÁR
MATEMATIKA ÉS INFORMATIKA KAR
INFORMATIKA SZAK

Licensz-dolgozat

Forgalmi táblák felismerése



TÉMAVEZETŐ:

CSATÓ LEHEL
BABEŞ-BOLYAI TUDOMÁNYEGYETEM,
MATEMATIKA ÉS INFORMATIKA KAR

SZERZŐ:

SZABÓ ÁGNES-TERÉZ

2015 JÚLIUS

Tartalomjegyzék

1. Bevezető	3
1.1. A dolgozat célja	3
1.2. Megvalósítás	3
2. Forgalmi táblák	4
2.1. Mik a forgalmi táblák?	4
2.2. Adathalmaz	4
2.2.1. Tanulási adatok	4
2.2.2. Tesztelési adatok	5
3. Darabolás	6
4. Képek előfeldolgozása	8
4.1. Színterek	8
4.1.1. RGB színtér	8
4.1.2. HSV színtér	8
4.2. Megvalósítás	9
4.3. Átméretezés	10
5. Neurális hálózatok	11
5.1. Bevezető	11
5.1.1. Biológiai modell	11
5.2. Mesterséges neuronok	11
5.2.1. Sigmoid neuron	12
5.3. Neurális hálók struktúrája	12
5.4. Sztochasztikus gradiens csökkentés	14
5.5. Backpropagation	15
6. Az algoritmus	16
7. A kimenet	17
8. Eredmények és következtetések	18
8.1. Rétegek számának meghatározása	18
8.2. Tanulási ráta meghatározása	18
8.3. Batch méretének meghatározása	19
8.4. Ciklusok számának meghatározása	21
A. Fontosabb programkódok listája	22
A.1. A neurális háló	22
A.2. Az adatok előfeldolgozása	24
A.3. A <i>main</i> függvény	25

1. fejezet

Bevezető

1.1. A dolgozat célja

Számos olyan rendszer létezik ami megtalál és felismer forgalmi táblákat valós környezetből vett képeken, de a robusztus és költséghatékony megoldás a mai napig aktív kutatási téma. Mivel több különböző típusú forgalmi tábla létezik és ezek között sok a hasonlóság, a megkülönböztetésük a mai napig kihívásnak bizonyul. A táblák felismerhetőségét befolyásolja több tényező, mint a fényerősség, árnyékolás, parciális fedés és egyéb akadályok.

Egyes autóvezetők számára nehézséget jelenthet a figyelmük megosztása, ami bizonytalansághoz, balesethez vezethet. Ha rendelkezésükre állna egy rendszer, ami képes figyelmeztetni őket a forgalmi táblákról és az általuk megszabott korlátról vagy információról, magabiztosabbá válhatnának.

A céлом egy gyors, hatékony és megbízható rendszer készítése forgalmi táblák felismerésére valós környezetben készített képekről.

1.2. Megvalósítás

A rendszer először előfeldolgozást végez a képeken. Mivel a tesztadatok nem azonos méretűek, átméretezi őket, majd szín szerinti szűrést végez, hogy egyszerűsítse a feldolgozandó adatokat. Ezt követően neurális háló segítségével elkezdődik a tanulási folyamat. Miután a rendszer megtanulta a különböző osztályok jellemzőit, a rendszer használható forgalmi táblák felismerésére tetszőlegesen kiválasztott képeken.

A felhasználó által kiválasztott képen elsősorban meg kell találni a forgalmi tábla helyét. Ennek érdekében a képet kisebb darabokra vágja és ezeken is végrehajtja a fentebb említett előfeldolgozást. Az így kapott képeken próbálja azonosítani a forgalmi táblákat.

Mivel a forgalmi táblák idővel változhatnak, lényeges szempont, hogy a tervezett rendszer könnyen alkalmazkodjon a változásokhoz, anélkül, hogy nagymértékben módosítani kelljen azt. A neurális hálók tökéletesek erre a feladatra, hiszen, ha az adatok változnak is, elég a tanulási folyamatot megismételni; nincs szükség a kód módosítására [L.D.o.T., 2010].

2. fejezet

Forgalmi táblák

2.1. Mik a forgalmi táblák?

A forgalmi táblák a forgalom irányításáért felelősek; figyelmeztetik, irányítják és informálják a gépkocsivezetőket. Három fő típusú forgalmi tábla van és mindegyik különböző formával rendelkezik:

1. utasító táblák (kör)
2. figyelmeztető táblák (háromszög)
3. informáló táblák (téglalap)

Ez alól kivétel a STOP tábla, ami nyolcszög alakú.

További információval szolgál a forgalmi tábláról a színe. A forgalmi táblák figyelemfelkeltőek kell legyenek, ennek megfelelően alkalmazzák a színeket is. A megjelenő színek: piros, fehér, fekete, sárga és kék [L.D.o.T., 2010].

A forgalmi táblák többsége két részből tevődik össze: egy külső telített sáv és egy belső szimbólum vagy szám. A belső szimbólum általában fekete.

2.2. Adathalmaz

Az adathalmazhoz egy forgalmi táblák felismerésére kitűzött németországi verseny adatait használtam fel [9].

2.2.1. Tanulási adatok

Az adathalmaz 1213 tanulási adatot tartalmaz. A képek mérete 16x16 pixeltől 128x128 pixelig változik, különböző szemszögből és változatos fényhatások mellett. A képek 43 kategóriába vannak sorolva, a tartalmazott forgalmi tábla alapján.

Habár nagy mennyiségű tanulási adat áll rendelkezésre, ezek nincsenek egyenlően elosztva az osztályok között. Ez megnehezíti a tanulást a neurális háló számára. Azok az osztályok amelyekben kevés tanulási adat van, kevésbé vannak befolyással a neurális háló kimenetére.

A 2.1 ábrán látható példa négy tanítási adatra. Az első három képen levő táblák könnyedén felismerhetőek, míg a az utolsó tábla nehezen azonosítható. Nehéz megmondani, hogy 30-as, 70-es vagy 80-as

2. FEJEZET: FORGALMI TÁBLÁK

sebességkorlát. Az adathalmazban kis számban ugyan, de előfordulnak ehhez hasonló táblák. Ezek megnehezítik a tanulási folyamatot.



2.1. ábra. Példa tanulási adatokra

forrás:http://benchmark.ini.rub.de/?section=gt_sdb&subsection=dataset

2.2.2. Tesztelési adatok

900 teszt adat állt a rendelkezésemre, amiken nullától hatig akárhány forgalmi tábla megjelenhetett. Ezek jóval nagyobb méretű képek a tanításra szánt képeknél, 1360x800 pixel nagyságúak. A 2.2 ábrán látható egy példa ilyen képre. Ezen a képen két forgalmi tábla található.



2.2. ábra. Példa teszt adatra

forrás:http://benchmark.ini.rub.de/?section=gt_sdb&subsection=dataset

3. fejezet

Darabolás

Annak érdekében, hogy megtaláljunk egy forgalmi táblát egy képen, szegmentációt alkalmaztam. A szegmentáció az a folyamat, amikor kisebb képeket vágunk ki egy nagyobb képből [Ohlander et al., 1978].

Több különböző módszer létezik a szegmentálás végrehajtására. Az általam alkalmazott véletlenszerű kiválasztáson alapszik. Véletlenszerű téglalapokat generálok és vágom ezeket ki az eredeti képből.

Mivel a valós környezetből vett képek több forgalmi táblát is tartalmazhatnak, fontos hogy az egész képet bejárják ezek a darabok. Azáltal, hogy nagy számban generálok ilyen darabokat, biztosítom, hogy nagy valószínűséggel megtaláljam a képen levő forgalmi táblákat. A táblák mérete is változik attól függően, hogy milyen távolságra van a fényképezőtől, ezért arra is figyelnem kellett, hogy a kivágott darabok mérete széles tartományban mozogjon.

Azáltal, hogy véletlenszerűen választom ki a vizsgálandó részeket az eredeti képből, ezeknek nagy része olyan kép lesz, ami nem tartalmaz forgalmi táblát vagy csak részlegesen fogja azt tartalmazni. Ez az oka annak, hogy a neurális háló akkor is megfelelően kell osztályozza a táblát ha csak 70-80% látszik belőle.



3.1. ábra. Szegmentálás előtti kép

forrás:http://benchmark.ini.rub.de/?section=gt_sdb&subsection=dataset

3. FEJEZET: DARABOLÁS



3.2. ábra. Példa szegmentálás utáni képekre

forrás:<http://benchmark.ini.rub.de/?section=gtSDB&subsection=dataset>

A 3.1 ábrán látható kép darabolásából nyert képekre látunk példákat a 3.2 ábrán. Az így létrejött képek között megtalálható a két forgalmi tábla (első és negyedik kép) de ezeken kívül olyan darabok is létrejönnek, amik nem tartalmaznak forgalmi táblát. Azok a darabok amik nem tartalmaznak forgalmi táblát más információt tartalmaznak, mint például emberek, fák, járművek, esetleg hirdető táblák.

4. fejezet

Képek előfeldolgozása

Összefoglaló: Annak érdekében, hogy egyszerűsítse a bevitt adatokat, előfeldolgozást végzek a képeken a színük alapján. A neurális háló a pixelek szürke árnyalatával dolgozik. A képek különböző fényviszonyok mellett készültek, ennek köszönhetően némelyik nagyon világos, mások pedig nagyon sötétek.

Megjegyzés: A képek valós környezetből származnak, így más objektumokat is tartalmazhatnak, mint például faágak.

4.1. Színterek

A szín a módja annak, ahogyan az emberi látórendszer a bejövő elektromágneses sugárzást értelmezi. Az a tartomány, ahol a rendszer színként érzékeli ezt a sugárzást körülbelül 400 és 830 nm-es hullámhosszak között jelentkezik (személytől függően változhat némi leg) [Tkalcic and Tasic].

A színterek pontosan meghatároznak színeket az érzékelhető hullámhossz tartományból. Arra szolgálnak, hogy könnyebben tudjuk meghatározni, létrehozni és felfogni a színeket. Egy színt általában három paraméter határoz meg. Ezek fejezik ki az adott szín helyét az illető színtérben. Különböző színtereket ismerünk, attól függően válasszuk ki a megfelelőt, hogy mire akarjuk alkalmazni [Ford and Roberts, 1998].

4.1.1. RGB színtér

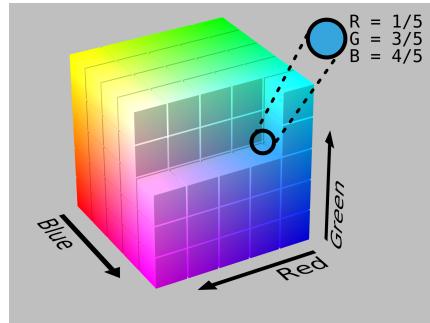
Az RGB színtér (4.1) egy additív színrendszer, mely a tri-kromatikus elméletben alapszik. A tri-kromatikus elmélet lényege, hogy 3 különböző szín segítségével bármilyen szín kikeverhető. Az RGB színtér könnyen implementálható, de nem párhuzamos az emberi látással. Az RGB színteret gyakran felhasználják számítógépes alkalmazásokhoz, mert nincs szükség transzformációra ahhoz, hogy megjelenítsük az információt. A színtér legegyszerűbb ábrázolása egy kocka, ahol a 3 tengely a piros, zöld és kék színekkel egyenértékű.

4.1.2. HSV színtér

Ellentétben az RGB ábrázolással, a HSV színtér (4.2) henger-koordinátákat alkalmaz. Ez az egyik leggyakoribb ábrázolási módszer. Az árnyalat(hue) az az emberi érzet, amely szerint egy szín egy másik színre vagy több szín keverékére hasonlít a piros, sárga, zöld és kék színek közül. A szín telítettsége

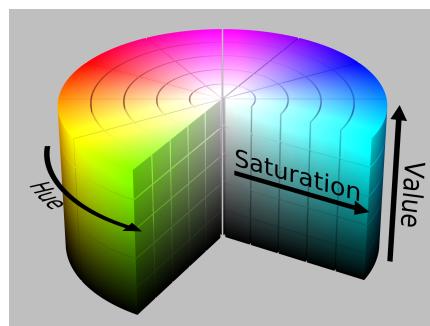
4. FEJEZET: KÉPEK ELŐFELDOLGOZÁSA

(saturation) egy szín színessége és a fényessége közti arány. A fényesség (value-brightness) az az emberi érzet, hogy egy terület több vagy kevesebb fényt áraszt magából.



4.1. ábra. RGB színtér

forrás:http://upload.wikimedia.org/wikipedia/commons/8/83/RGB_Cube_Show_lowgamma_cutout_b.png



4.2. ábra. HSV színtér

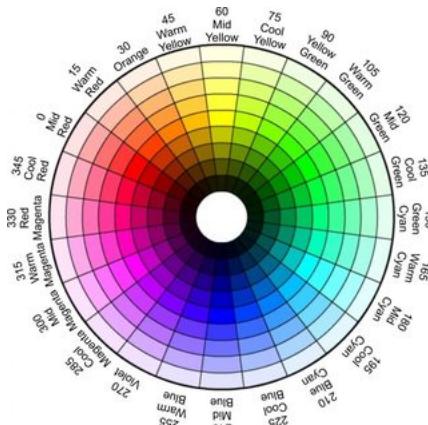
forrás:http://upload.wikimedia.org/wikipedia/commons/0/0d/HSV_color_solid_cylinder_alpha_lowgamma.png

4.2. Megvalósítás

A forgalmi táblák figyelemfelkeltőek kell legyenek, ezért jól látható színeket használnak, mint például piros, kék, sárga, fekete és fehér.

A bemeneti képeket szín alapján szűröm, ezáltal felépíték egy új, fekete-fehér képet. Ha az árnyalat, a telítettség és a fényesség egy bizonyos intervallumban helyezkedik el, az új kép megfelelő pixelje fekete lesz, különben fehér. Ezeket az intervallumokat kísérletezéssel és a 4.3 ábra segítségével határoztam meg.

4. FEJEZET: KÉPEK ELŐFELDOLGOZÁSA



4.3. ábra. Színkör

forrás:<http://stackoverflow.com/questions/21737613/image-of-hsv-color-wheel-for-opencv>

Szín	Árnyalat	Telítettség	Fényesség
Piros	0-20, 340-360	>0.3	>0.2
Sárga	25-60	>0.3	>0.2
Kék	180-270	>0.3	>0.2
Fekete	-	-	<0.3

4.1. táblázat. Elfogadási tartomány

4.3. Átméretezés

Mivel a tanulási adatok és a teszt adatból kivágott képek mérete is változó, a képeket átméretezem 25x25 pixelre és ezzel dolgozom tovább. Ennek következtében adatvesztés és torzulás léphet fel, amikor más méretű képet méretezek át.

5. fejezet

Neurális hálózatok

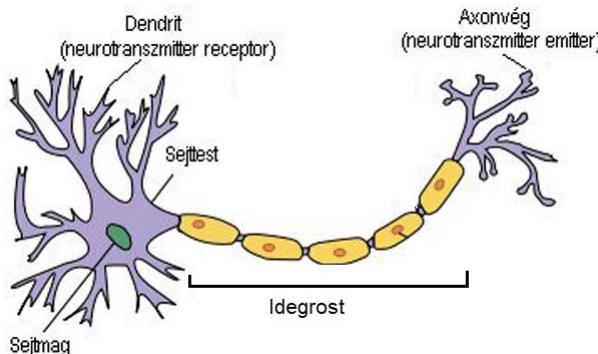
Összefoglaló: Ebben a fejezetben megnézzük a biológiai modellt aminek a mintájára a mesterséges neuronokat létrehozták, megismerkedünk a mesterséges neuronokkal és a neurális hálókkal majd pedig megnézzük hogyan is tanulnak ezek.

5.1. Bevezető

5.1.1. Biológiai modell

Az idegsejtek (neuronok) az idegrendszer alkotóelemei. A neuron jeleket fogad és ezekre megfelelő választ generál. Az emberi idegrendszer nagyon komplex műveletek elvégzésére képes rövid idő alatt. Mivel ennek analógjára hozták létre a mesterséges neurális hálókat, ezért fontos a biológiai alapok megismerése.

A 5.1 ábrán látható egy idegsejt és annak részei. A dendritek más neuronokkal állnak kapcsolatban és a bejövő információt szállítják. A beérkezett információ feldolgozását a sejtmag hajtja végre. Az idegrost egy szigetelt vezető aminek az a feladata, hogy továbbszállítsa az információt. Az axonvégek más neuronok dendritjeivel vannak kapcsolatban, a választ szállítják tovább a velük kapcsolatban levő idegsejteknek.



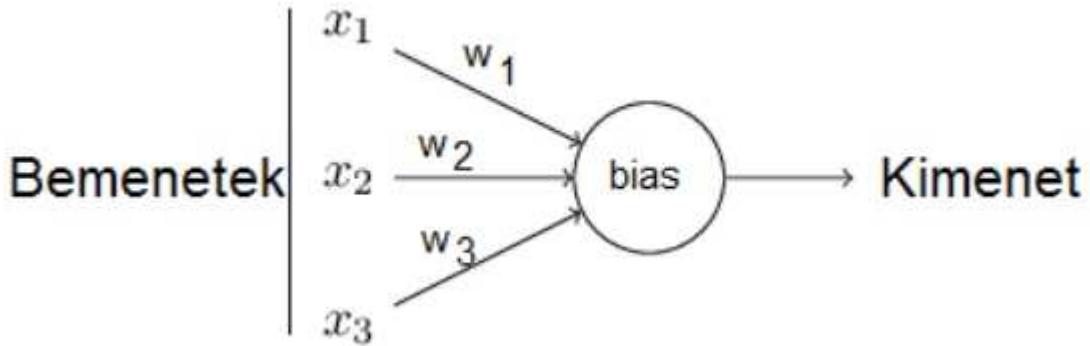
5.1. ábra. Idegsejt
forrás:<http://www.clker.com/clipart-neuron.html>

5.2. Mesterséges neuronok

A neuronok a neurális hálók építőkövei.

5.2.1. Sigmoid neuron

A szigmoid neuron 0 és 1 közötti értékeket kap bemeneti paraméterként és létrehoz egy kimenetet ugyanbben az intervallumban.



5.2. ábra. Sigmoid neuron

forrás:<http://neuralnetworksanddeeplearning.com/chap1.html>

Mindegyik bemenethez súlyok vannak rendelve, ami azt jelképezi, hogy az illető bemenet mennyire játszik fontos szerepet a neuron kimenetében. minden neuronhoz tartozik továbbá egy küszöbérték, ami a kisebb rendellenességek javítására szolgál.

A szigmoid neuron kimenete a 5.1 képlettel számítható ki.

$$\sigma\left(\sum_{i=1}^{n-1} w_i x_i + b\right) \quad (5.1)$$

$$\sigma(z) = \frac{1}{1 + e^{-z}} \quad (5.2)$$

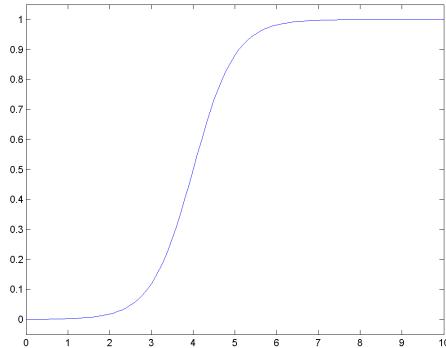
A 5.2 függvényt aktiválási függvénynek nevezik. Az aktiválási függvény szerepe, hogy a kimenetet egy adott intervallumba szorítsa. Ahogy ez az 5.3 ábrán is látszik, a szigmoid neuron esetében ez az intervallum 0 és 1 között van.

A szigmoid neuron egyik tulajdonsága, hogy kis változás a súlyokban kis változást eredményez a kimenetben. Ez egyszerre előny és hátrány is. Előny, mert amikor a kimenet közel van az elvárt kimenethez, a módsítások nem eredményezik azt, hogy a kimenet túlságosan eltávolodjon az elvárttól. Hátrány, mert amikor a kimenet nagyon helytelen, mint például a tanulás elején, hosszú időbe telik a javítása [Nielsen, 2014].

5.3. Neurális hálók struktúrája

A neuronokból álló hálózatokat neurális hálóknak nevezzük. A hálózat neuronjai hasonló típusú műveletet végeznek a többi neurontól függetlenül. Egy neuron sok más neuronnal áll kapcsolatban, ahogy ez

5. FEJEZET: NEURÁLIS HÁLÓZATOK



5.3. ábra. Sigmoid függvény grafikus képe

a biológiai analógiából kikövetkeztethető. Tehát a neurális hálózatok olyan információfeldolgozó eszközök, amelyek párhuzamos, elosztott működésre képesek, lokális feldolgozást végző neuronokból állnak, képesek tanulni, és a megtanult információt felhasználni [Csató].

A neuronok egy hálózaton belül általában csak meghatározott számú neuronnal vannak összekötve, és ez a kapcsolat általában egyirányú, ezért a hálózatokat különböző rétegekre szoktuk bontani az összekötések szerint. Az egyes rétegekhez tartozó neuronok az előző réteg neuronjainak kimenetével, vagy a bemenettel, illetve a következő réteg bemenetével vannak összekötve [Csató].

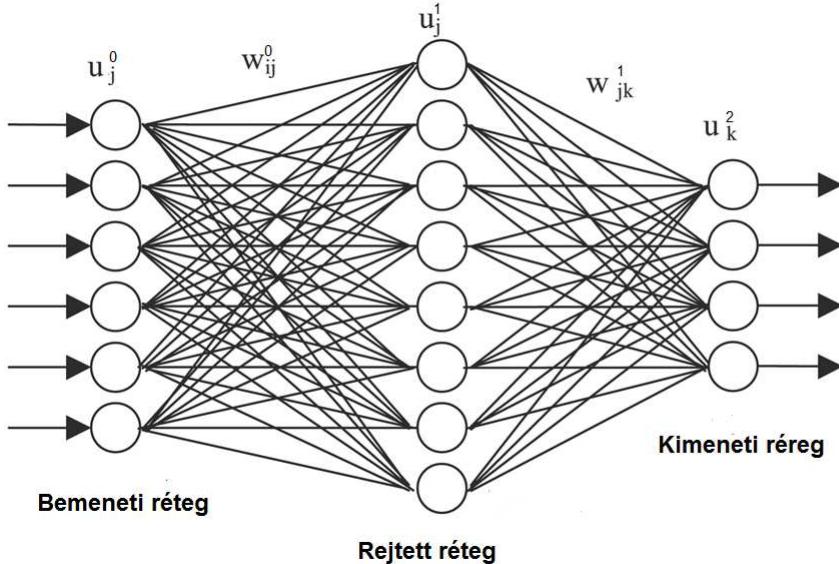
1. Bemeneti réteg: azok a neuronok találhatók itt, amelyek a bemeneti jel továbbítását végzik a hálózat felé. A legtöbb esetben nem jelöljük őket külön
2. Rejtett réteg: a tulajdonképpen feldolgozást végző neuronok tartoznak ide. Egy hálózaton belül több rejtett réteg is lehet
3. Kimeneti réteg: az itt található neuronok a külvilág felé továbbítják az információt. A feladatuk ugyanaz, mint a rejtett rétegbeli neuronoké [Csató].

Ennek függvényében a neuronok elhelyezkedésük alapján három csoportba sorolhatók:

1. Bemeneti neuronok
2. Rejtett neuronok
3. Kimeneti neuronok

A legtöbb hálózat esetében az egyes rétegek teljesen össze vannak kötve, vagyis egy réteg egy neuronjának kimenete a következő réteg összes bemenetével össze van kötve. A hálózatoknál előforduló számítások elvégzéséhez célszerűbb a hálózatokat különböző mátrixokkal jellemzni. A bemeneteket elegendő egy darab vektorba rendezve ábrázolni, ez N bemenet esetén egy N-es vektort jelent. Az egyes rétegek közötti súlyokat hasonlóan a bemenetekhez mátrixokba szoktuk rendezni. Ez a következőképpen néz ki: ha az 1. rétegen n darab neuron van az 1+1. rétegen pedig m, akkor a köztük lévő súlyok száma $n \times m$. Ez mátrixba rendezve egy $m \times n$ -es mátrixot jelent, amelynek oszlopai az 1. réteg neuronjait, sorai az 1+1. réteg neuronjait reprezentálják [Csató].

A 5.4 ábrán egy egyszerű sigmoid neuronkból álló neurális háló látható. A súlyvektorok mérete már ennél a kis példánál is viszonylag nagy, ezért egy nagy hálónál ezek az értékek jelentősen megnőnek.



5.4. ábra. Egyszerű neurális háló

Olyan neurális hálót használtam, amelynek 625 bemenete és 43 kimenete van. A bemeneti neuronok a kép egy pixelének szürke árnyalatát tartalmazzák. A kimenetek száma megegyezik a forgalmi táblák osztályainak a számával.

5.4. Sztochasztikus gradiens csökkentés

Annak érdekében, hogy a súlyokat és küszöbértékeket a kimenetnek az elvárt eredménytől való eltérése függvényében tudjam módosítani, a sztochasztikus gradiens csökkentést alkalmaztam. Ez a gradiens csökkentés egyszerűsített változata. Gradiens csökkentéssel a teljes tanítási adathalmazra számolja a hiba értékét, míg a sztochasztikus gradiens csökkentés közelíti a hibát egy bizonyos számú véletlenszerűen kiválasztott adat alapján. Minél nagyobb ez a szám, a közelítés pontossága annyival jobb lesz, de mivel nem a teljes adathalmazzal dolgozik, ezért gyorsabb lesz.

A 5.3 egyenlet a négyzetes hibafüggvény. n a bemeneti tanítási adatok száma, $y(x)$ az x bemenetre a háló által generált kimenet és a az elvárt kimenet az x bemenetre. Amikor az elvárt és a kiszámított kimenet közel van egymáshoz, a hiba kicsi lesz. Mivel a célom, hogy a kiszámított kimenet az elvárt kimenetet közelítse meg minél jobban, ezt a hibát kell minimalizálnom. Ennek érdekében bevezetem a ΔC -vel jelölt gradiensét a hibafüggvénynek. A gradiens értékét a 5.4 képlettel lehet kiszámítani.

$$C = \frac{1}{2n} \sum_x \|y(x) - a(x)\|^2 \quad (5.3)$$

$$\Delta C = \left(\frac{\partial C}{\partial w_i}, \frac{\partial C}{\partial b_j} \right)^T \quad (5.4)$$

A hiba csökkentése érdekében ismételtem alkalmaztam a 5.5 és a 5.6 szabályokat a súlyokra és

5. FEJEZET: NEURÁLIS HÁLÓZATOK

küszöbértékekre. η a tanítási ráta. Ez egy kicsi érték kell legyen, de nem túl kicsi, mert akkor a tanulási folyamat túl lassú lenne.

$$w_i \rightarrow w_i - \eta \frac{\partial C}{\partial w_i} \quad (5.5)$$

$$b_j \rightarrow b_j - \eta \frac{\partial C}{\partial b_j} \quad (5.6)$$

5.5. Backpropagation

A backpropagation algoritmus arra szolgál, hogy a háló súlyait és küszöbértékeit úgy állítsa be, hogy a kimenet megközelítse minél jobban az elvárt kimenetet.

A 5.7 egyenletben z^l a súlyozott bementi vektor az l -edik rétegben. Ahogy a 5.8 egyenletben látható, a kimeneti vektor egy rétegen az előző réteg kimenetének használatával számolható ki, vagyis az információ továbbításával. A háló kimenetének kiszámítása után a 5.9-es egyenlet segítségével lehet kiszámolni az utolsó réteg hibáját, ahol L a rétegek számát jelöli, $\Delta_a C$ pedig a $\frac{\partial C}{\partial a_j^L}$ parciális deriváltakat tartalmazó vektor.

Maga a backpropagation a 5.10 képlet segítségével történik minden l rétegre, $L-1$ -től 2-ig.

$$z^l = (w^l a^{l-1} + b^l) \quad (5.7)$$

$$a^l = \sigma(z^l) \quad (5.8)$$

$$\delta^L = \Delta_a C \circ \sigma'(z^L) \quad (5.9)$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \circ \sigma'(z^l) \quad (5.10)$$

6. fejezet

Az algoritmus

Algorithm 1 Backpropagation

```
1: Súlyok és küszöbértékek inicializálása véletlenszerű adatokkal
2: for minden ciklus  $i$  do
3:   Tanulási adatok összekeverése
4:   for minden batch  $j$  do
5:     for minden  $d$  tanulási adatra a  $j$  batch-ból do
6:       minden réteg kimenetének kiszámítása a 5.2 egyenlet segítségével a bemenettől a kimenet felé
7:       Kimenet hibájának kiszámítása a 5.9 egyenlet segítségével
8:       minden neuron hibájának kiszámítása a kimenettől a bemenet felé haladva a 5.10 egyenlet segítségével
9:     end for
10:     $d$  tanulási adatra számított hibák összegzése
11:    Súlyok és küszöbértékek módosítása felhasználva a hibák összegét a 5.5 és 5.6 egyenlet segítségével
12:  end for
13: end for
```

Ahhoz, hogy a háló tanulni tudjon, fontos az, hogy a súlyokat és küszöbértékeket megfelelően kicsi számokkal inicializáljuk. Ezt úgy valósítottam meg, hogy a 0 és 1 között véletlenszerűen generált számot elosztottam 3000-el.

A batch-ben egyidőben jelen levő adatok minél változatosabbak kell legyenek, annak érdekében hogy tanulás közben egy időben figyelembe vegye a különböző osztályok tulajdonságait. Ezért van szükség az adatok összekeverésére. A beolvasás sorban történik, vagyis az azonos osztályba tartozók egymás után fognak szerepelni. Ha ebben a sorrendben kerülnének be a batch-be, akkor a háló először megpróbálná megtanulni az egyik osztály tulajdonságait, utána pedig sorban a következőkét. Az adatok keverésével ezt kerülöm el [Nielsen, 2014].

7. fejezet

A kimenet

A neurális háló kimenete 43 neuront tartalmaz. Ez megegyezik a forgalmi táblák kategóriáinak számával.

Azt, hogy a háló melyik kategóriába sorolta az adott táblát a legnagyobb kimenetű neuron indexe dönti el. E mellett bevezettem egy olyan feltételt, hogy a legnagyobb és a második legnagyobb érték közötti különbség nagyobb kell legyen mint 0,2. Erre azért volt szükség, mert megtörténhet olyan, hogy a háló nem képes eldönteni, hogy melyik osztályba tartozik egy kép. Ekkor a kimenet több neuronjának is hasonló értéke lesz.

A kimenet alapján tehát három különböző lehetőség van: helyesen osztályoz egy képet, helytelenül osztályoz egy képet, nem képes osztályozni egy képet. Amikor nem képes osztályozni egy képet, azt úgy tekintem, hogy a kép nem tartalmaz forgalmi táblát.

8. fejezet

Eredmények és következtetések

A háló teszteléséhez a tesztelési adatokból kivágott forgalmi táblákat használtam.

Különböző paraméterekre (rétegek száma, rejtett réteg/rétegek mérete, batch méret, tanulási ráta) folytattam tesztelést az ideális körülmények megtalálása érdekében. Eleinte nem a teljes adathalmazra, hanem csak pár osztályra futtattam, a gyors eredmény érdekében. Ezek a tesztek gyorsan lefutottak és viszonylagos eredményt adtak a hálóról.

Amikor kis adathalmazzal teszteltem, kevés osztállyal, azt tapasztaltam, hogy ha az osztályok olyan táblákat tartalmaztak amik jelentősen különböznek egymástól, akkor 100%-ban felismeri a tesztadatokat. Ilyen osztályokra példa a STOP tábla és a 30-as sebességkorlátozó tábla. Abban az esetben, ha az osztályokba tartozó táblák hasonlítanak egymásra, mint például a 30-as és 50-es sebességkorlátozó tábla, a felismert táblák száma lecsökken, kb. 85%-ra.

8.1. Rétegek számának meghatározása

A rendszer a legjobb eredményt akkor adta, ha nem volt jelen rejtett réteg. Az által, hogy nem volt rejtett réteg a számítások egyszerűsödtek és a folyamat felgyorsult.

Abban az esetben, ha egy rejtett réteg volt, a tanulás kb. 100 ciklus után kezdődött el. A tanulási folyamat lassú volt és a végeredmény nem mutatott jobb eredményt mint amikor hiányzott a rejtett réteg.

Az elkövetkezendő teszteredmények olyan rendszerben készültek, ahol nem volt jelen rejtett réteg.

8.2. Tanulási ráta meghatározása

A tanulási ráta behatárolására először három-négy osztályra futtattam teszteket. Ez által gyorsan kaptam egy viszonyítási pontot, amit majd alkalmazni tudtam a több osztályra történő futáskor.

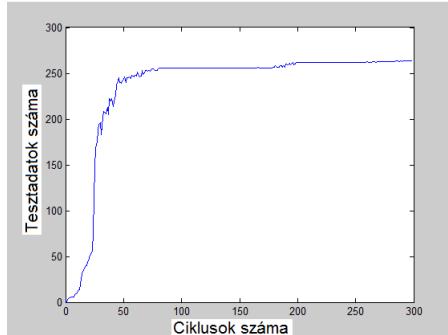
A tanulási ráta általában 0 és 1 között vehet fel értékeket. Én nagyobb tanulási rátával is próbálkoztam [Csató].

Elsőnek 1-es tanulási rátával próbálkoztam. Az eredmények pozitívak voltak, a 8.1 ábrán látható a helyesen kategorizált képek száma minden ciklusban.

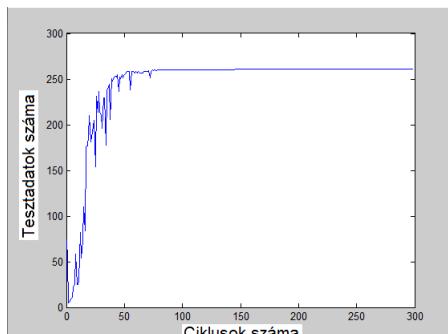
Ezek után a tanulási ráta növelésével próbálkoztam. Habár a helyesen kategorizált képek száma minimálisan csökkent, ebben az esetben a háló sokkal hamarabb kezdte megtanulni az osztályok jellemzőit.

8. FEJEZET: EREDMÉNYEK ÉS KÖVETKEZTETÉSEK

A 8.2 ábrán jól látszik, hogy a görbe sokkal meredekebb eleinte, vagyis gyorsabban tanul az elején és utána normalizálódik. A maximum elérése után a kezdeti oszcilláció is megszűnik.



8.1. ábra. Helyesen felismert képek száma 1-es tanulási ráta mellett ciklusonként



8.2. ábra. Helyesen felismert képek száma 3-as tanulási ráta mellett ciklusonként

Végeztem teszteket más értékű tanulási rátával is, de ha nem változtattam nagy mértékben az értéket, akkor hasonló eredményeket kaptam.

Ha túlságosan megnövelte az értékét, az oszcilláció ami a 8.2 ábra elején látható megnőtt és nagy mértékben lecsökkent a felismert táblák száma. Ha túl kicsi értéket választottam (0,0001), a tanulási folyamat 300 ciklus után sem indult be, a felismert képek száma 0 maradt. Ugyan az 1-es tanulási ráta is hasonló pontosságot eredményezett mint a 3-as, a 3-as tanulási ráta mellett sokkal kevesebb ciklusra van szükség a pontosság maximalizálásához. Emiatt a többi paraméter meghatározásakor a 3-as tanulási rátával dolgoztam. A paraméterek meghatározása után a nagy adatmennyiség gyors feldolgozása érdekében használtam 3-as tanulási rátát, a pontosság növeléséhez viszont 0,8-as tanulási rátát használtam több cikluson keresztül. Ebből kiindulva a további tesztekhez a tanulási rátát háromnak választottam meg.

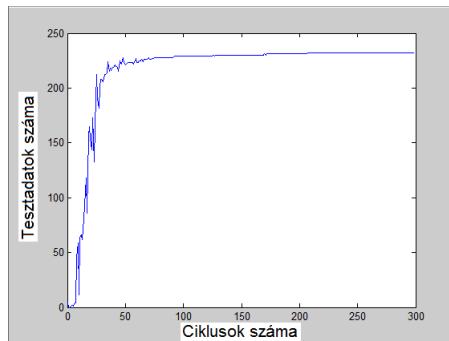
Lehetséges adaptív megoldásokat alkalmazni, ahol a tanulási ráta lépésenként változik, ezt én nem próbáltam ki, mert a tanulás beindulása után nem lépett fel oszcilláció [Csató].

8.3. Batch méretének meghatározása

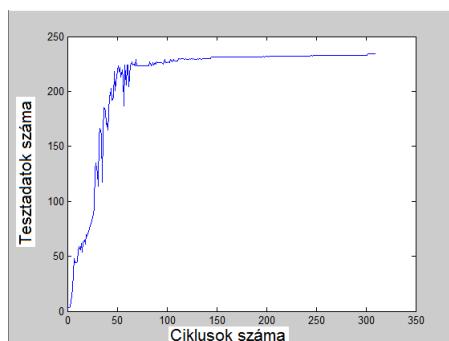
A batch méretének meghatározásához is több tesztet futtattam különböző értékekre. A behatárolást ebben az esetben is kis méretű adathalmazra végeztem.

8. FEJEZET: EREDMÉNYEK ÉS KÖVETKEZTETÉSEK

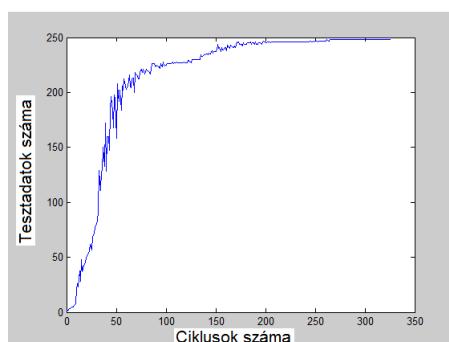
A batch méretének változtatása nem volt nagy hatással a helyesen kategorizált képek számára. A 8.3, 8.4 és 8.5 ábrákon látható a 10-re, 20-ra, illetve 30-ra kapott eredmények. Nincs nagy különbség közöttük, de a 30-as értékre lett a legnagyobb a felismert táblák száma, ezért a továbbiakban ezzel az értékkal dolgoztam. Nagyobb mennyiségi tesztadatra megpróbáltam később a bath méretét 10-re állítani, de a tanulás jóval nehezebben indult be, ezért nem végeztem vele több tesztet.



8.3. ábra. Helyesen felismert képek száma 10-es bath méret mellett ciklusonként



8.4. ábra. Helyesen felismert képek száma 20-as bath méret mellett ciklusonként



8.5. ábra. Helyesen felismert képek száma 30-as bath méret mellett ciklusonként

8. FEJEZET: EREDMÉNYEK ÉS KÖVETKEZTETÉSEK

8.4. Ciklusok számának meghatározása

A ciklusok száma a tanulási folyamatban nem játszik olyan fontos szerepet mint a tanulási ráta, viszont ha túl kicsinek választjuk meg a hálónak nincs elég ideje tanulni. A nagy ciklusszám nincs negatív hatással a felismert táblák számára, de feleslegesen növeli a futási időt. Miután a háló elérte azt a pontot, amikor már nem képes tovább fejlődni, felesleges további ciklusokat futtatni. Lehetne egy olyan feltételt beiktatni, hogy ha egy bizonyos százalékban felismeri a háló a táblákat akkor állítsuk le a tanulást még mielőtt az összes ciklust végrehajtanánk, de ha ezt a százaléket túl nagyra állítjuk és ez által elérhetetlené válik, akkor felesleges, ha pedig túl kicsire, a futási időt ugyan lecsökkent, de a háló még képes lett volna tanulni tovább.

100 ciklus ez esetek többségében eléggesnek bizonyul, de sok esetben 50 ciklus is ugyanazt eredményezte.

A. függelék

Fontosabb programkódok lista

A.1. A neurális háló

A lenti osztály felelős a háló karbantartásáért.

```
1 #include "network.h"
2
3 //konstruktor
4 Network::Network(std::vector<int> sizes)
5 {
6     //retegek beallitasa
7     _num_layers = sizes.size();
8
9     _sizes = sizes;
10
11    //biasok inicializalasa
12    _biases = Matrix2D(_sizes);
13    _biases.randInit();
14
15    //sulyok inicializalasa
16    _weights = Matrix3D(_sizes);
17    _weights.randInit();
18 }
19
20 //az elozo reteg kimenete segitsegevel kiszamolja a kovetkezo reteg kimenetet
21 Vector Network::feedforward(Vector a)
22 {
23     for(unsigned int i = 0; i < _weights.GetMatrix2DCount(); i++) {
24         Matrix2D w_i = _weights.GetMatrix2D(i);
25         int s = _biases.GetVector(i).GetCount();
26         Vector new_a(s);
27         for (int j = 0; j < s; j++)
28             new_a.SetItem(j, w_i.dot(a).GetItem(j) + _biases.GetData() [i].GetData() [j]);
29         a = sigmoid(new_a);
30     }
31     return a;
32 }
33
34 //a helyesen kategorizat kepek szama
35 unsigned int Network::evaluate(DataSet testDataSet)
36 {
37     unsigned int correct = 0;
38     for(unsigned int i = 0; i < testDataSet.GetDataCount(); i++) {
39         DataSet testData = testDataSet.GetData() [i];
40         Vector tv = feedforward(testData.GetImageData());
41         double imgClass = tv.GetMax();
42         if (imgClass == testData.GetImageClass())
43             correct++;
44     }
45     return correct;
46 }
47
48 //szochasztikus gradiens csokkentes
49 void Network::SGD(DataSet dataset, int epochs, int mini_batch_size, double eta, DataSet
50 testDataSet)
51 {
52     unsigned int n = dataset.GetDataCount();
53
54     for (unsigned int j = 0; j < epochs; j++) {
55         //az adatok osszekeverese
56         dataset.Shuffle();
57         for (int i = 0; i <= n-mini_batch_size; i+=mini_batch_size) {
58             DataSet mini_batch = DataSet(dataset.GenerateMiniBatch(i,mini_batch_size));
59             updateMiniBatch(eta,mini_batch);
60         }
61     }
62 }
```

A. FÜGGELÉK: FONTOSABB PROGRAMKÓDOK LISTÁJA

```

61    }
62    //a vegso biasok kiirasa hogy felhasznalhato legyen kesobb
63    struct stat buffer;
64    std::string name = "biases.txt";
65    if (stat(name.c_str(), &buffer) == 0) {
66        remove(name.c_str());
67    }
68    _biases.writeData(name);

69    //a vegso sulyok kiirasa hogy felhasznalhato legyen kesobb
70    struct stat buffer2;
71    std::string name2 = "weights.txt";
72    if (stat(name2.c_str(), &buffer2) == 0) {
73        remove(name2.c_str());
74    }
75    _weights.writeData(name2);
76 }

77 //backpropagation alkalmazasa a batch-ekre
78 void Network::updateMiniBatch(double eta, DataSet mini_batch)
79 {
80     Matrix2D nabla_b(_sizes);
81     nabla_b.zeros();
82     Matrix3D nabla_w(_sizes);
83     nabla_w.zeros();
84     for(unsigned int i = 0; i < mini_batch.GetDataCount(); i++) {
85         Data mini_batch_row = mini_batch.GetData()[i];

86         Matrix2D nabla_b2(_sizes);
87         nabla_b2.zeros();
88         Matrix3D nabla_w2(_sizes);
89         nabla_w2.zeros();

90         Vector activation = mini_batch_row.GetImageData();
91         Matrix2D activations(_sizes,0);
92         activations.SetDataRow(0,activation);
93         Matrix2D zs(_sizes);

94         //a kimenetek kiszamitása előre haladva
95         for(unsigned int j = 0; j< _biases.GetVectorCount(); j++) {
96             Vector z(_biases.GetVector(j).GetCount());
97             z.SetData(_weights.GetMatrix2D(j).dot(activation).add(_biases.GetVector(j)));

98             zs.SetDataRow(j,z);
99             activation = sigmoid_prime(z);
100            activations.SetDataRow(j+1,activation);
101        }

102        Vector outActivation = activations.GetVector(activations.GetVectorCount()-1);
103        Vector preferedOutput(outActivation.GetCount());
104        //elvárt kimenet felelítése
105        preferedOutput.setItem(mini_batch_row.GetImageClass(), 1.0);

106        //a kimeneti reteg hibajának kiszamitása
107        Vector delta = cost_derivative(outActivation,preferedOutput).HadamardProd(
108            sigmoid_prime(zs.GetVector(zs.GetVectorCount()-1)));
109        nabla_b2.SetDataRow(nabla_b2.GetVectorCount()-1,delta);

110        nabla_w2.SetData(nabla_w2.GetMatrix2DCount()-1,dot(delta,activations.GetVector(
111            activations.GetVectorCount()-2)));

112        //backpropagation alkalmazva visszafele a retegek hibainak kiszamitasara
113        for (unsigned int j = 2; j<_num_layers; j++) {
114            Vector z = zs.GetVector(zs.GetVectorCount()-j);
115            Vector spv = sigmoid_prime(z);
116            delta = dot(_weights.GetMatrix2D(_weights.GetMatrix2DCount()-j+1),delta).
117                HadamardProd(spv);

118            nabla_b2.SetDataRow(nabla_b2.GetVectorCount()-j,delta);
119            nabla_w2.SetData(nabla_w2.GetMatrix2DCount()-j,dot(delta,activations.GetVector(
120                activations.GetVectorCount()-j-1)));
121        }

122        Nablas ns(nabla_b2,nabla_w2);

123        Matrix2D delta_nabla_b = ns._nabla_b;
124        Matrix3D delta_nabla_w = ns._nabla_w;
125        nabla_b = nabla_b.add(delta_nabla_b);
126        nabla_w = nabla_w.add(delta_nabla_w);
127    }

128    //sulyok és biasok megfelelo modosítása
129    _weights = _weights.sub(nabla_w.multiplyByScalar(eta/(double)mini_batch.GetDataCount()));
130    _biases = _biases.sub(nabla_b.multiplyByScalar(eta/(double)mini_batch.GetDataCount()));
131
132
133
134
135
136
137
138
139
140
141

```

A. FÜGGELÉK: FONTOSABB PROGRAMKÓDOK LISTÁJA

}

A.2. Az adatok előfeldolgozása

```

1 #include "data.h"
2
3 #include <QImage>
4 #include<QStringList>
5
6 Data::Data()
7 {
8 }
9
10 //kep adatainak inicializalasa
11 Data::Data(Vector imageData, unsigned int imageClass)
12 {
13     _imageData.SetData(imageData.GetData());
14     _imageClass = imageClass;
15 }
16
17 //kep adatainak lekerdezese
18 Vector Data::GetImageData()
19 {
20     return _imageData;
21 }
22
23 //kep osztalyanak lekerdezese
24 unsigned int Data::GetImageClass()
25 {
26     return _imageClass;
27 }
28
29 //adatok betoltese
30 void Data::LoadData(QString path,unsigned int dim)
31 {
32     QStringList pieces = path.split( "/" );
33     //kep osztalyanak meghatarozasa
34     _imageClass = pieces.value( pieces.length() - 2 ).toInt();
35     _imageData=Vector(dim*dim);
36     QImage * image = new QImage();
37     if(image->load(path))
38     {
39         //kep adatainak feldolgozasa
40         for(unsigned int i = 0; i< image->width(); i++) {
41             for(unsigned int j = 0; j< image->height(); j++) {
42                 QRgb rgba = image->pixel(i,j);
43
44                 //kep pixeleinek konvertalasa HSV szinterbe
45                 double r = (double) qRed(rgba)/255;
46                 double g = (double) qGreen(rgba)/255;
47                 double b = (double) qBlue(rgba)/255;
48                 double c_max = std::max(std::max(r,g),b);
49                 double c_min = std::min(std::min(r,g),b);
50                 double D = c_max - c_min;
51                 double H = 0;
52                 double L = 0;
53
54                 if (D == 0)
55                     H = 0;
56                 else if (c_max==r)
57                     H = 60*((g-b)/D);
58                 else if (c_max==g)
59                     H = 60*(2+(b-r) /D);
60                 else if (c_max==b)
61                     H = 60*(4+(r-g) /D);
62
63                 if (H<0)
64                     H+=360;
65
66                 if (c_max==0)
67                     L = 0;
68                 else
69                     L = D/c_max;
70
71                 //szín szerinti szüres
72                 if (((H >=0 && H <=20) || (H>=340)) && c_max >0.2 && L>0.3)
73                 {
74                     image->setPixel(i,j,0);
75                 }
76             }
77         }
78     }
79 }
```

A. FÜGGELÉK: FONTOSABB PROGRAMKÓDOK LISTÁJA

```

81         else if ((H >=25 && H <=60) && c_max >0.2 && L>0.3)
82         {
83             image->setPixel(i,j,0);
84         }
85         else if ((H >=180 && H <=270) && c_max >0.2 && L>0.3)
86         {
87             image->setPixel(i,j,0);
88         }
89         else if (c_max <0.3)
90             image->setPixel(i,j,0);
91         else
92             image->setPixel(i,j,qRgb(255, 255, 255));
93     }
94
95     //kep atmeretezese
96     QImage scaledImage = image->scaled(dim,dim,Qt::IgnoreAspectRatio,Qt::
97     FastTransformation);
98     for (int i = 0;i< dim;i++)
99     {
100         for (int j = 0; j< dim;j++){
101             //szurkearnyalat elmentese
102             _imageData.SetItem(i*dim+j, qGray(scaledImage.pixel(i,j))/255);
103         }
104     }
105 }
```

A.3. A *main* függvény

```

2 #include <QtGui/QApplication>
#include "mainwindow.h"
#include "network.h"
#include <QDirIterator>
#include <iostream>
7 int main(int argc, char *argv[])
{
    srand (time(NULL));
12     //kepek egyszeges meretenek beallitasa
    int size = 25;
13
14     //a halo retegeinek es ezek mereteinek beallitasa
15     std::vector<int> array2(2);
16     array2[0] = size*size;
17     array2[1] = 43;
18
19     //az adatok betoltese
20     DataSet d = DataSet();
21     QString path = ".../..../TestLearning";
22     d.LoadData(size,path);
23
24     std::cout << "Learning_started" << std::endl;
25     //a halo letrehozasa
26     Network net(array2);
27     //tanulas megkezdese
28     net.SGD(d,300,30,3,d);
29     std::cout << "Learning_finished" << std::endl;
}
```

Irodalomjegyzék

L.D.o.T., *Know your traffic signs official edition.* London Department for Transport, 2010.

<http://benchmark.ini.rub.de/?section=gtsdb&subsection=dataset>.

R. Ohlander, K. Price, and D. R. Reddy, „Picture segmentation using a recursive region splitting method,” *Computer graphics and image processing*, vol. 8, pp. 313–333, 1978.

A. Ford and A. Roberts, „Colour space conversions,” 1998.

M. Tkalcic and J. F. Tasic, „Colour spaces - perceptual, historical and applicational background.”

M. A. Nielsen, „Neural networks and deep learning,” 2014.

L. Csató, „A neurális hálózatok alapjai.”

X. W. Gao, L. Padladchikova, D. Shaposhnikov, K. Hong, and N. Shevtsova, „Recognition of traffic signs based on their colour and shape features extracted using human vision models,” *Journal of Visual Communication and Image Representation*, vol. 17, no. 4, pp. 675–685, 2006.

S. Escalera, X. Baró, O. Pujon, J. Vitria, and P. Radeva, „Traffic-sign recognition systems,” *Journal of Visual Communication and Image Representation*, vol. VI, 2011.

K. Brkic, „An overview of traffic sign detection methods,” 2010.

J. Hatzidimos, „Automatic traffic sign recognition in digital images,” 2004.

S. Haykin, *Neural Networks And Learning Machines Third Edition.* Pearson Education Inc., 2009.

_____, *Neural Networks A Comprehensive Foundation Second Edition.* Pearson Education Inc., 1999.

IRODALOMJEGYZÉK

C. M. Bishop, *Information Science and Statistics.* Springer, 2006.

S. Russell and P. Norvig, *Artificial Intelligence A modern Approach Third Edition.* Pearson Education Inc., 2010.

R. Szeliski, *Computer Vision: Algorithms and Applications.* Springer, 2010.

T. Mainer, „Genetic algorithm for traffic sign detection.”