

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 the robust and cost-effective solution is still an active research subject. The recognisability of traffic signs is affected by many variables, like luminous intensity, shading, partial coverage and other obstacles. The developed system is 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 more background threads. Extracts random rectangles from the original picture and decides if it contains a traffic sign or not, and if it does categorises it. In case of a hit saves the coordinates and encloses it on the picture.

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	4
1.1. Cél	4
1.2. Megvalósítás	4
2. Eddigi eredmények	5
2.1. Szín és formai jellemzők kivonásával	5
3. Forgalmi táblák	6
3.1. Mik a forgalmi táblák?	6
3.2. Adathalmaz	6
3.2.1. Tanulási adatok	6
3.2.2. Tesztelési adatok	7
4. Darabolás	8
5. Képek előfeldolgozása	10
5.1. Színterek	10
5.1.1. RGB színtér	10
5.1.2. HSV színtér	11
5.2. Megvalósítás	11
5.3. Átméretezés	12
6. Neurális hálók	13
6.1. Bevezető	13
6.1.1. Biológiai modell	13
6.2. Mesterséges neuronok	13
6.2.1. Sigmoid neuron	13
6.3. Szerkezetek	15
6.4. Sztochasztikus gradiens csökkentés	15
6.5. Backpropagation	16
7. Az algoritmus	17
8. A kimenet	18
9. Előzetes tesztelés	19
10. Eredmények és következtetések	20
10.1. Rétegek számának meghatározása	20
10.2. Tanulási ráta meghatározása	20
10.3. Batch méretének meghatározása	22
10.4. Ciklusok számának meghatározása	22
11. Továbbfejlesztési lehetőségek	25

TARTALOMJEGYZÉK

12. Fontosabb programkódok listája

26

1. fejezet

Bevezető

1.1. Cél

Számos olyan rendszer létezik ami megtalál és felismer forgalmi táblákat valós környezetből vett képeken, de a robusztus és költséghatékony megoldás mai napig aktív kutatási téma. Mivel több különböző típusú forgalmi tábla létezik és ezek között sok a hasonlóság, a megkülönböztetésük 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 rendszert készítése.

1.2. Megvalósítás

A rendszer a gépi tanulás segítségével ismeri fel a forgalmi táblákat. Ezen belül neurális hálókat használtam.

Amikor egy alkalmazás valós környezetbe kerül, fontos, hogy megbízható legyen. Egy tábla hibás azonosítása végzetes következményekkel járhat.

Mivel a forgalmi táblák idővel változhatnak, lényeges szempont, hogy a tervezett rendszer könnyen alkalmazkodjon a változásokhoz, anélkül, hogy nagy mértékben 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.

2. fejezet

Eddigi eredmények

Forgalmi táblák felismerésére több módszert fejlesztettek ki, ennek egyike a neurális hálókkal történő tanulás. Ebben a fejezetben bemutatok néhány alternatív módszert.

2.1. Szín és formai jellemzők kivonásával

A forgalmi táblákat színük és formájuk segítségével lehet kategorizálni, teljes felismerést viszont nem lehet biztosítani. Az [Gao et al., 2006] hivatkozásban a forgalmi táblák belsejében levő szimbólum figyelmen kívül hagyásával végeztek kategorizálást, pusztán a forma és szín alapján.

Az [Hatzidimos, 2004] hivatkozásban két részre osztható a javasolt megoldás

1. A tábla megtalálása a képen
2. A tábla felismerése

A tábla behatárolásához szín szerinti szűrést használ. Az így létrehozott bináris képből kiválasztja a fekete pixelek sűrűsége alapján azt a területet ami potenciálisan tartalmazza a táblát. Ezután forma alapján végez egy szűrést a képen. Amennyiben úgy véli, hogy megtalálta a forgalmi tábla helyét, elkezdi a felismerést. Pixelenként összehasonlítja a piros, zöld és kék komponensét a kinyert képnek egy mintaképpel mindegyik osztályból és ezekből számol egy átlagot. Abba az osztályba sorolja a képet, amelyik mintaképre a legnagyobb átlag jön ki.

3. fejezet

Forgalmi táblák

3.1. Mik a forgalmi táblák?

A forgalmi táblák a forgalom irányításáért felelősek; figyelmeztetik, irányítják és informálják a gépkereskedőket. Három fő típusú forgalmi tábla van és mindegyiknek 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 figyelemfelkeltők kell legyenek, ennek megfelelően alkalmazzák a színeket is. A megjelenő színek: piros, fehér, fekete, sárga és kék. [8, 2010]

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

3.2. Adathalmaz

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

3.2.1. 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 3.1 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, de 70-es vagy

3. FEJEZET: FORGALMI TÁBLÁK



3.1. ábra. Példa tanulási adatokra

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

3.2.2. 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 3.2 ábrán látható egy példa ilyen képre. Ezen a képen két forgalmi tábla található.



3.2. ábra. Példa teszt adatra

4. 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. Az által 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.

Az által, 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.



4.1. ábra. Szegmentálás előtti kép

4. FEJEZET: DARABOLÁS



4.2. ábra. Példa szegmentálás utáni képekre

5. fejezet

Képek előfeldolgozása

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

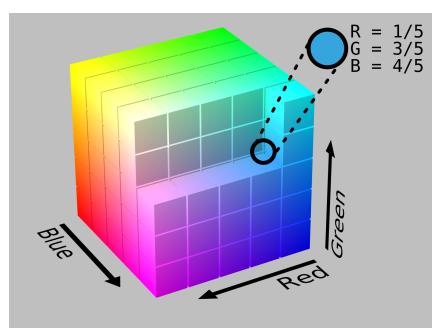
Megjegyzés: A képek valós környezetből származnak, így más objektumokat is tartalmazhatnak, mint például faágak.

5.1. Színterek

A színterek arra szolgálnak, hogy könnyebben tudjuk meghatározni, létrehozni és felfogni a színeket. Egy színt általában három paraméter határozza 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]

5.1.1. RGB színtér

Az RGB színtér 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.

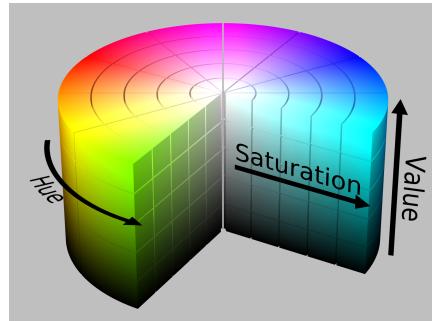


5.1. ábra. RGB színtér

5. FEJEZET: KÉPEK ELŐFELDOLGOZÁSA

5.1.2. HSV színtér

Ellentétben az RGB ábrázolással, a HSV színtér 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(saturation) egy szín színessége és a fényessége közti arány. A fényesség (value) az az emberi érzet, hogy egy terület több vagy kevesebb fényt áraszt magából.

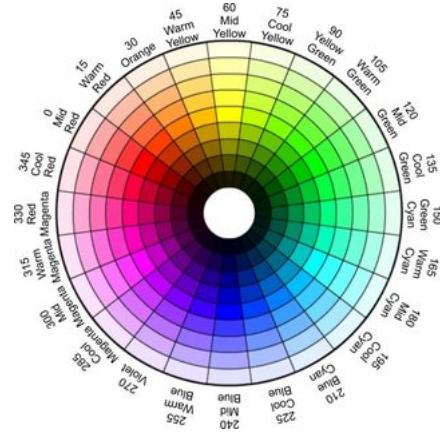


5.2. ábra. HSV színtér

5.2. 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 5.3 ábra segítségével határoztam meg.



5.3. ábra. Színkör

5. FEJEZET: KÉPEK ELŐFELDOLGOZÁSA

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
Fehér	-	-	>0.85

5.1. táblázat. Elfogadási tartomány

5.3. Átméretezés

Mivel a tanulási adatok és a teszt adatból kivágott képek mérete is változik, a képeket átméretezem 25x25 pixelre és ezzel dolgozom tovább.

6. fejezet

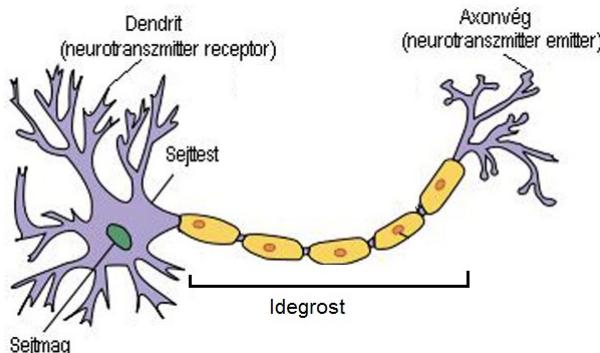
Neurális hálók

6.1. Bevezető

6.1.1. Biológiai modell

Az idegsejtek (neuronok) az idegrendszer alkotóelemei. A neuron jeleket fogad és ezekre megfelelő választ generál. A 6.1 ábrán látható egy idegsejt és annak részei. A dendritek a bejövő információt szállítják, az idegrost egy szigetelt vezető, az axonvégek a választ szállítják tovább a velük kapcsolatban levő idegsejteknek.

Az emberi idegrendszer nagyon komplex műveletek elvégzésére képes rövid idő alatt, ennek analógjára hozták létre a mesterséges neurális hálókat.



6.1. ábra. Idegsejt

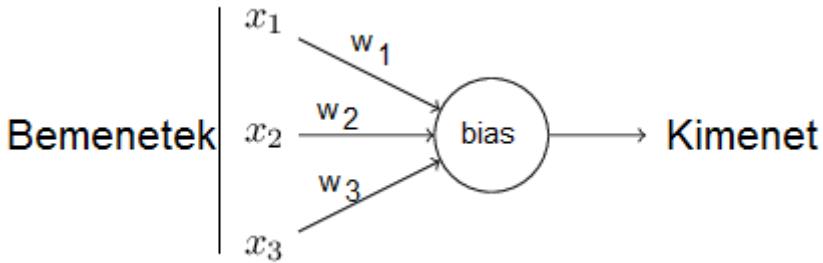
6.2. Mesterséges neuronok

A neuronok a neurális hálók építőkövei.

6.2.1. Sigmoid neuron

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

6. FEJEZET: NEURÁLIS HÁLÓK



6.2. ábra. Sigmoid neuron

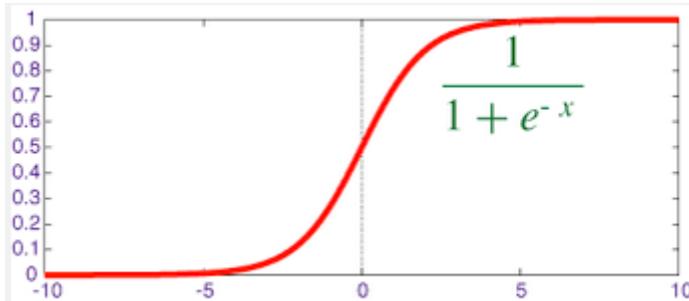
Mindegyik bemenethez súlyok vannak rendelve, ami azt jelképezi, hogy az illető bemenet mennyire játszik fontos szerepet a neuron kimenetében. minden neuronhoz tartozik továbbá egy "bias" ami a kisebb rendellenességek javítására szolgál.

A sigmoid neuron kimenete a 6.1 képlettel számítható ki.

$$\sigma\left(\sum_{i=1}^{n-1} w_i * x_i + b\right) \quad (6.1)$$

$$\sigma(z) = \frac{1}{1 + e^{-z}} \quad (6.2)$$

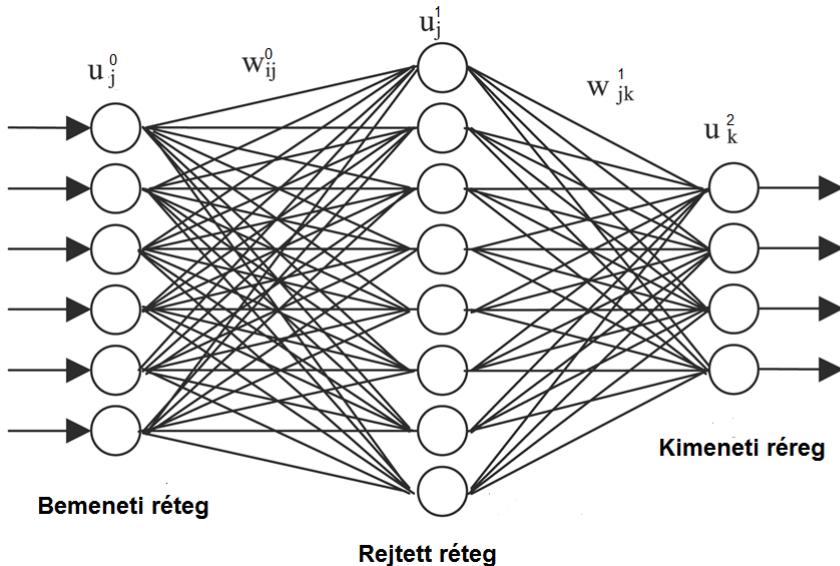
A 6.2 függvényt aktiválási függvénynek nevezik. Az aktiválási függvény szerepe, hogy a kimenetet egy adott intervallumba szorítsa. Ahogy ez a 6.3 ábrán is látszik, a sigmoid neuron esetében ez az intervallum 0 és 1 között van.



6.3. ábra. Szigmoid függvény grafikus képe

A szigmoid neuron egyik tulajdonsága, hogy kis változás a súlyokban, kis változást eredményez a kimenetben. Ez legalább annyira előny is, mint hátrány. 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.

A 6.4 ábrán egy egyszerű szigmoid neuronokból álló neurális háló látható. A súlyvektorok mérete már ennél a kis példánál is viszonylag nagy, ezért egy nagy hálónál ezek az értékek jelentősen megnőnek.



6.4. ábra. Egyszerű neurális háló

6.3. Szerkezetek

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. Különböző számú rejtett réteggel kísérleteztem, és ezeken belül is változó számú neuronnal.

6.4. Sztochasztikus gradiens csökkentés

Annak érdekében, hogy a súlyokat és bias-okat 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 6.3 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 ∇C -vel jelölt gradiensét a hibafüggvénynek. A gradiens értékét a 6.4 képlettel lehet kiszámítani.

$$C = \frac{1}{2n} \sum_x \|y(x) - a(x)\|^2 \quad (6.3)$$

$$\nabla C = \left(\frac{\partial C}{\partial w_i}, \frac{\partial C}{\partial b_j} \right)^T \quad (6.4)$$

A hiba csökkentése érdekében ismételtem alkalmaztam a 6.5 és a 6.6 szabályokat a súlyokra és biasokra. η a tanítási ráta. Ez egy kicsi érték kell legyen, de nem túl kicsi, mert akkor a tanulási folyamat túl lassú lenne.

$$w_i \rightarrow w_i - \eta \frac{\partial C}{\partial w_i} \quad (6.5)$$

$$b_j \rightarrow b_j - \eta \frac{\partial C}{\partial b_j} \quad (6.6)$$

6.5. Backpropagation

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

A 6.7 egyenletben z^l a súlyozott bemeneti vektor az l -edik rétegben. Ahogy a 6.8 egyenletben látható, a kimeneti vektor egy rétegen az előző réteg kimenetének használatával számolható ki, vagyis az információ továbbításával. A háló kimenetének kiszámítása után a 6.9-es egyenlet segítségével lehet kiszámolni az utolsó réteg hibáját, ahol L a rétegek számát jelöli, $\nabla_a C$ pedig a $\frac{\partial C}{\partial a_j^L}$ parciális deriváltakat tartalmazó vektor.

Maga a backpropagation a 6.10 képlet segítségével történik minden l rétegre, $L-1$ -től 2-ig.

$$z^l = (w^l * a^{l-1} + b^l) \quad (6.7)$$

$$a^l = \sigma(z^l) \quad (6.8)$$

$$\delta^L = \nabla_a C \circ \sigma'(z^L) \quad (6.9)$$

$$\delta^l = ((w^{l+1})^T * \delta^{l+1}) \circ \sigma'(z^l) \quad (6.10)$$

7. fejezet

Az algoritmus

1. Súlyok és bias-ok inicializálása véletlenszerű adatokkal
2. minden ciklusra végezd:
 - (a) Tanulási adatok összekeverése
 - (b) minden batch-re végezd:
 - i. minden tanulási adatra a batch-ból végezd:
 - A. Mindegyik neuron kimenetének a kiszámítása a bemenettől a kimenet felé
 - B. Kimenet hibájának kiszámítása
 - C. minden neuron hibájának kiszámítása a kimenettől a bemenet felé haladva
 - ii. Gradiens csökkentés: súlyok és bias-ok újraszámolása

Ahhoz, hogy a háló tanulni tudjon, fontos az, hogy a súlyokat és bias-okat 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]

8. 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.

9. fejezet

Előzetes tesztelés

Le akartam ellenőrizni, hogy mennyire helyes és hatékony az általam kódolt tanítási program. Ehhez tanítási és teszt adatokat generáltam.

Két kategóriát hoztam létre: az egyik osztályban a képek felső része többnyire fekete volt és az alsó többnyire fekete, a másikban pedig fordítva. A 9.1 és 9.2 ábrákon láthatóak példák a létrehozott képekre.

A fekete oldalon a pixelek 20%-os valószínűsséggel fehérek, 80% valószínűsséggel fekete; a fehér oldalon a pixelek 20%-os valószínűsséggel feketék, 80% valószínűsséggel fehérek. A tesztelés lényege az volt, hogy a program felismerje, hogy a fentebb említett két kategóriát helyesen megkülönböztesse egymástól.

A generált képek 16x16 pixelesek voltak.

A neurális háló az összes hasonlóan létrehozott tesztelési képet megfelelően kategorizálta.



9.1. ábra. Példa első kategóriájú képekre



9.2. ábra. Példa második kategóriájú képekre

10. 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.

10.1. 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.

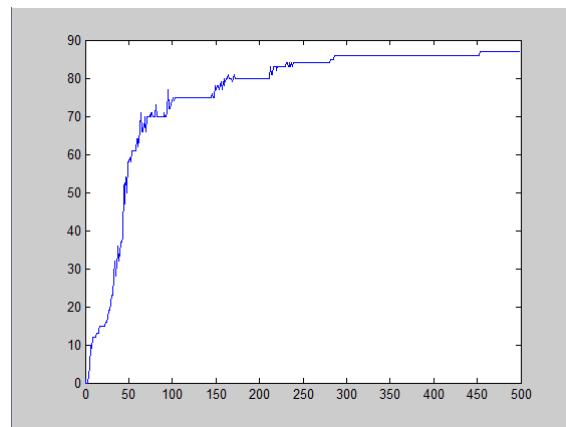
10.2. Tanulási ráta meghatározása

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.

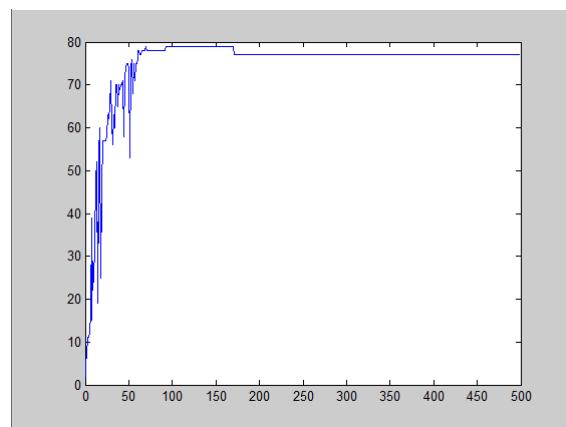
Elsőnek 1-es tanulási rátával próbálkoztam. Az eredmények elég pozitívak voltak, a 10.1 ábrán látható a helyesen kategorizált képek száma minden ciklusban.

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. A 10.2 ábrán jól látszik, hogy sokkal meredekebb az elején, vagyis gyorsabban tanul az elején és utána normalizálódik. A maximum elérése után a kezdeti oszcilláció is megszűnik.

10. FEJEZET: EREDMÉNYEK ÉS KÖVETKEZTETÉSEK

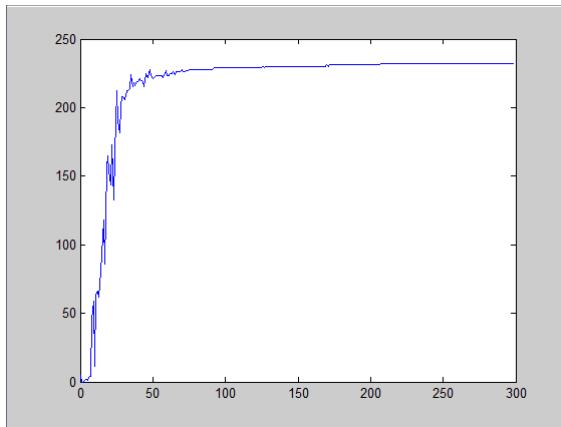


10.1. ábra. 1-es tanulási ráta



10.2. ábra. 3-as tanulási ráta

10. FEJEZET: EREDMÉNYEK ÉS KÖVETKEZTETÉSEK



10.3. ábra. 10-es bath méret eredményei

Végeztem teszteket más értékű tanulási rátával is, de ha nem változtattam nagy mértékben az értéket, akkor hasonló eredményt adtak. Ha túlságosan megnövelte az értékét, az oszcilláció ami a 10.2 ábra elején látható megnőtt és nagy mértékben lecsökkent a felismert táblák száma. Ha túl kicsi értéket választottam, a tanulási folyamat nagyon lassan indult be. Ebből kiindulva a további tesztekhez a tanulási rátát háromnak választottam meg.

10.3. Batch méretének meghatározása

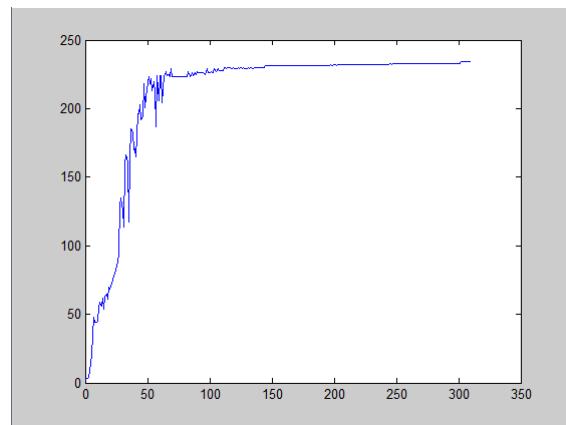
A batch méretének meghatározásához is több tesztet futtattam különböző értékekre. A behatárolást ebben az esetben is kis méretű adathalmazra végeztem.

A batch méretének változtatása nem volt nagy hatással a helyesen kategorizált képek számára. A 10.3, 10.4 és 10.5 ábrákon látható a 10-re, 20-ra, illetve 30-ra kapott eredmények. Nincs nagy különbség közöttük, de a 30-as értékre let legnagyobb a felismert táblák száma, ezért a továbbiakban ezzel az értékkel dolgoztam. Nagyobb mennyiségi tesztadatra megpróbáltam később a bath méretét 10-re állítani, de a tanulás jelentősen nehezebben indult be, ezért nem végeztem vele több tesztet.

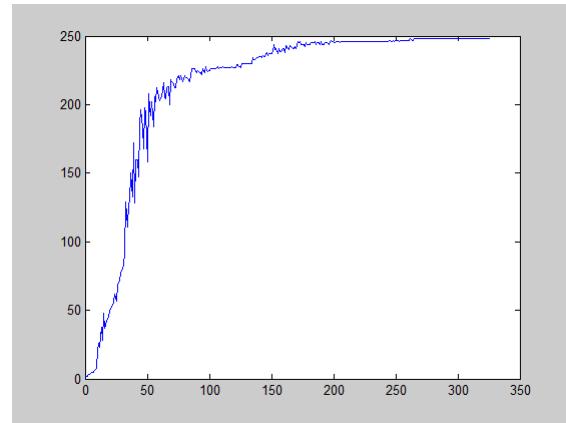
10.4. Ciklusok számának meghatározása

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, hogy 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 meg mielőtt az összes ciklust végrehajtanánk, de ha ezt a százalékot túl nagyra állítjuk és ez által elérhetetlenné válik, akkor felesleges, ha pedig túl kicsire, a futási időt ugyan lecsökkent, de a háló még képes lett volna tanulni tovább.

10. FEJEZET: EREDMÉNYEK ÉS KÖVETKEZTETÉSEK



10.4. ábra. 20-as bath méret eredményei



10.5. ábra. 30-as bath méret eredményei

10. FEJEZET: EREDMÉNYEK ÉS KÖVETKEZTETÉSEK

100 ciklus ez esetek többségében elégségesnek bizonyul, de néha 50 ciklus is ugyanazt eredményezte.

11. fejezet

Továbbfejlesztési lehetőségek

Az alkalmazás továbbfejlesztése szempontjából a pontosság és hatékonyság az elsődleges szempontok amiket szem előtt kell tartani. Ennek érdekében lehetne a képek előfeldolgozásán módosítani, a szín szerinti szűrést javítani, forma szerinti szűrést bevezetni.

A neurális háló szempontjából is több fejlesztési lehetőség áll rendelkezésre: más típusú neuronokat használni(nem sigmoid, hanem perceptron vagy más), konvolúciós neurális hálót alkalmazni, más hibafüggvényt használni, különböző típusú neuronokat beépíteni ugyanabba a hálóba.

12. fejezet

Fontosabb programkódok listája

```
1 #include "network.h"
2
3 Network::Network(std::vector<int> sizes)
4 {
5     _num_layers = sizes.size();
6     _sizes = sizes;
7     _biases = Matrix2D(_sizes);
8     _biases.randInit();
9
10    _weights = Matrix3D(_sizes);
11    _weights.randInit();
12}
13
14 Vector Network::sigmoid(Vector z)
15 {
16     Vector ret(z.GetCount());
17     for(unsigned int i = 0; i < z.GetCount(); i++)
18         ret.SetItem(i, 1.0 / (1.0 + exp(-z.GetItem(i))));
19
20    return ret;
21}
22
23 Vector Network::sigmoid_prime(Vector z)
24 {
25     Vector ret(z.GetCount());
26     Vector sigm(z.GetCount());
27     sigm = sigmoid(z);
28     for(unsigned int i = 0; i < z.GetCount(); i++)
29         ret.SetItem(i, sigm.GetItem(i) * (1 - sigm.GetItem(i)));
30
31    return ret;
32}
33
34 Vector Network::feedforward(Vector a)
35 {
36     for(unsigned int i = 0; i < _weights.GetMatrix2DCount(); i++) {
37         Matrix2D w_i = _weights.GetMatrix2D(i);
38         int s = _biases.GetVector(i).GetCount();
39         Vector new_a(s);
40         for (int j = 0; j < s; j++)
41             new_a.SetItem(j, w_i.dot(a).GetItem(j) + _biases.GetData() [i].GetData() [j]);
42         a = sigmoid(new_a);
43     }
44     return a;
45}
46
47 unsigned int Network::evaluate(DataSet testDataSet)
48 {
49     unsigned int correct = 0;
50     for(unsigned int i = 0; i < testDataSet.GetDataCount(); i++) {
51         Data testData = testDataSet.GetData() [i];
52         Vector tv = feedforward(testData.GetImageData());
53         double imgClass = tv.GetMax();
54         if (imgClass == testData.GetImageClass())
55             correct++;
56     }
57     return correct;
58}
59
60 void Network::SGD(DataSet dataset, int epochs, int mini_batch_size, double eta, DataSet
61 testDataSet)
62 {
63     unsigned int n = dataset.GetDataCount();
64
65     for (unsigned int j = 0; j < epochs; j++) {
66         dataset.Shuffle();
67         for (int i = 0; i <= n-mini_batch_size; i+=mini_batch_size) {
```

12. FEJEZET: FONTOSABB PROGRAMKÓDOK LISTÁJA

```

        DataSet mini_batch = DataSet(dataset.GenerateMiniBatch(i,mini_batch_size));
        updateMiniBatch(eta,mini_batch);
    }

71
    struct stat buffer;
    std::string name = "biases.txt";
    if (stat(name.c_str(), &buffer) == 0) {
        remove(name.c_str());
    }
    _biases.writeData(name);

    struct stat buffer2;
    std::string name2 = "weights.txt";
    if (stat(name2.c_str(), &buffer2) == 0) {
        remove(name2.c_str());
    }
    _weights.writeData(name2);
}

86 void Network::updateMiniBatch(double eta, DataSet mini_batch)
{
    Matrix2D nabla_b(_sizes);
    nabla_b.zeros();
    Matrix3D nabla_w(_sizes);
    nabla_w.zeros();
    for(unsigned int i = 0; i < mini_batch.GetDataCount(); i++) {
        Data mini_batch_row = mini_batch.GetData()[i];

96        Matrix2D nabla_b2(_sizes);
        nabla_b2.zeros();
        Matrix3D nabla_w2(_sizes);
        nabla_w2.zeros();

101       Vector activation = mini_batch_row.GetImageData();
        Matrix2D activations(_sizes,0);
        activations.SetDataRow(0,activation);
        Matrix2D zs(_sizes);

106       for(unsigned int j = 0; j< _biases.GetVectorCount(); j++) {
            Vector z(_biases.GetVector(j).GetCount());
            z.SetData(_weights.GetMatrix2D(j).dot(activation).add(_biases.GetVector(j)));

111           zs.SetDataRow(j,z);
            activation = sigmoid_prime(z);
            activations.SetDataRow(j+1,activation);
        }

116       Vector outActivation = activations.GetVector(activations.GetVectorCount()-1);
        Vector preferedOutput(outActivation.GetCount());
        preferedOutput.setItem(mini_batch_row.GetImageClass(),1.0);

        Vector delta = cost_derivative(outActivation,preferedOutput).HadamardProd(
            sigmoid_prime(zs.GetVector(zs.GetVectorCount()-1)));
        nabla_b2.SetDataRow(nabla_b2.GetVectorCount()-1,delta);

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

126       for (unsigned int j = 2; j<_num_layers; j++) {
        Vector z = zs.GetVector(zs.GetVectorCount()-j);
        Vector spv = sigmoid_prime(z);
        delta = dot(_weights.GetMatrix2D(_weights.GetMatrix2DCount()-j+1),delta).
            HadamardProd(spv);

131           nabla_b2.SetDataRow(nabla_b2.GetVectorCount()-j,delta);
        nabla_w2.SetData(nabla_w2.GetMatrix2DCount()-j,dot(delta,activations.GetVector(
            activations.GetVectorCount()-j-1)));
    }

136       Nablas ns(nabla_b2,nabla_w2);

        Matrix2D delta_nabla_b = ns._nabla_b;
        Matrix3D delta_nabla_w = ns._nabla_w;
        nabla_b = nabla_b.add(delta_nabla_b);
        nabla_w = nabla_w.add(delta_nabla_w);
    }

141       _weights = _weights.sub(nabla_w.multiplyByScalar(eta/(double)mini_batch.GetDataCount()));
        _biases = _biases.sub(nabla_b.multiplyByScalar(eta/(double)mini_batch.GetDataCount()));
    }

146 Vector Network::cost_derivative(Vector output_activations,Vector y)
{
    Vector delta = output_activations.sub(y);
    return delta;
}

```

12. FEJEZET: FONTOSABB PROGRAMKÓDOK LISTÁJA

```
151 }
152 Matrix2D Network::dot (Vector v1,Vector v2)
153 {
154     Matrix2D ret (v1.GetCount (),v2.GetCount ());
155     Vector rowVector(v2.GetCount ());
156     for(unsigned int i = 0; i < v1.GetCount (); i++) {
157         for(unsigned int j = 0; j < v2.GetCount (); j++) {
158             rowVector.SetItem(j,v1.GetItem(i)*v2.GetItem(j));
159         }
160         ret.SetDataRow(i,rowVector);
161     }
162     return ret;
163 }
164
165 Vector Network::dot (Matrix2D m,Vector v)
166 {
167     Vector row(m.GetColumnCount ());
168     double value;
169     for(unsigned int i = 0; i < m.GetColumnCount (); i++) {
170         value = 0;
171         for(unsigned int j = 0; j < v.GetCount (); j++) {
172             value+= m.getItem(j,i) * v.GetItem(j);
173         }
174         row.SetItem(i,value);
175     }
176     return row;
177 }
```

Irodalomjegyzék

- 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.
- J. Hatzidimos, „Automatic traffic sign recognition in digital images,” 2004.
- Know your traffic signs official edition.* London Department for Transport, 2010.
- 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. A. Nielsen, „Neural networks and deep learning,” 2014.
- 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.