

Szegedi Tudományegyetem
Informatikai Intézet

SZAKDOLGOZAT

Vass Róbert

2020

**Szegedi Tudományegyetem
Informatikai Intézet**

Japán írásjelek felismerése neurális hálózatokkal

Szakdolgozat

Készítette:

Vass Róbert

programtervező informatika

BSc szakos hallgató

Témavezető:

Dr. Tóth László

egyetemi docens

**Szeged
2020**

Feladatkiírás

Tetszőleges mély neuronhálós fejlesztőkörnyezetben (Keras, Tensorflow, PyTorch) tervezzen meg és implementáljon egy mély neuronhálót, amely képes japán írásjelek felismerésére azok képi reprezentációja alapján.

A fejlesztés során elemezze ki, hogy a különböző népszerű hálóimplementációs finomítások (konvolúciós hálók, batch normalization, dropout, data augmentation, stb.) milyen mértékben javítanak a kiindulás rendszerén.

Vizsgálja meg, hogy standard, publikus hálóimplementációkat (pl. alexnet, vgg19) használva vajon tud-e javítani az eredményein. Készítsen egy egyszerű grafikus felületet, amellyel a háló működése szemléltethető, kipróbálható.

Tartalmi összefoglaló

- **A téma megnevezése:**

Japán írásjelek felismerése neurális hálózatokkal.

- **A megadott feladat megfogalmazása:**

A feladat egy 80-as években létrehozott japán karakter adatbázis példáin keresztül betanítani egy neurális hálózatot a karakterek felismerésére és általánosítására. A hálózat működését grafikus felületen bemutatni.

- **A megoldási mód:**

Az első lépés az adatok alakjának felmérése, továbbá annak normalizálása, majd annak többféle hálózat architektúrával való megközelítése. Például teljesen csatolt hálózatokkal vagy konvolúciós hálózatokkal megpróbálni minél nagyobb pontosságot elérni. A cél minél több konfiguráció kipróbálása, ezek között regularizációs módszerek mint a Dropout és L1,L2 regularizáció, továbbá az adatok augmentációja ahol új tanítóadat nyerhető az eredeti megváltoztatásával, majd a kívánt hálózatot felcsatolni egy weboldalra és felületet nyújtani annak érdekében hogy a hálózat kipróbálható legyen.

- **Alkalmazott eszközök, módszerek:**

Python: Egyszerű szkriptnyelv ami több operációs rendszeren is elérhető, nagy előnye hogy rengeteg külső könyvtár és kiegészítő van hozzá továbbá hogy egyszerűen futtatható.

Tensorflow: Egy nyílt forráskódú gépi tanulási platform amit a Google fejleszt, többféle beépített eszközt tartalmaz.

Google Colab: Egy online Python jegyzetfüzet felület amit a Google biztosít, megkönnyíti a neurális hálózatok felállítását, tanítását és lehetővé teszi hardver „kölcsonnését” is.

PHP: Webprogramozáshoz használt szkriptnyelv, a háló működését bemutató grafikus felület elkészítéséhez volt szükséges.

- **Elért eredmények:**

A hálózatokkal elért eredmény fokozatosan javult, míg az elején 40-50 százalék körül mozgott a pontosság, addig a bonyolultabb konvolúciós hálókkal már a 98 százalékos eredményt is elért a tesztadaton.

A hálózat jól becsül meg olyan karaktereket is amelyek az adatbázison kívülről származnak, ha nem is találja el elsőre, általában akkor is benne van az elkövetkezendő tíz becslésében.

A weboldal szemlélteti a működést mind felöltött példákon, mind rajzolt karaktereken.

- **Kulcsszavak:**

Gépi tanulás, Python, Tensorflow, Keras, Mesterséges neurális hálók, Google Colab

Tartalomjegyzék

Feladatkiírás	2
Tartalmi összefoglaló	3
Tartalomjegyzék	4
BEVEZETÉS ÉS ÖSSZEFOGLALÁS	6
1. GÉPI TANULÁS	7
1.1. Osztályozási problémák	7
1.2. Az osztályozási problémák megközelítése.....	8
1.2.1. Perceptron modell.....	8
1.2.2. Neurális hálózatok	9
1.3. Mesterséges neurális hálózatok (ANN)	9
1.3.1. Előnyei	10
1.3.2. Hátrányai	10
1.3.3. Fajtái	11
1.4. Mély neuronhálók (DNN).....	12
1.5. ANN az osztályozási feladatokhoz.....	13
2. KARAKTERFELISMERÉS (OCR)	13
2.1. Neurális hálók OCR-hez	14
2.1.1. Példák	14
3. A JAPÁN KARAKTEREK.....	15
3.1. Hiragana.....	15
3.2. Katakana	15
3.3. Kanji.....	16
3.4. Japán karakterek osztályozásban	17
4. A TANÍTÓADATOK	17
4.1. Szürkeárnyaltos:	18
4.2. Bináris:	19
4.3. Problémák	19
4.3.1. Adatok felosztása.....	20
4.3.2. Címkék normalizálása.....	20
5. ADAT AUGMENTÁCIÓ.....	21

5.1. Adat augmentációs technikák írott karakterekre.....	21
5.1.1. Forgatás.....	22
5.1.2. Eltolás	23
5.1.3. Mesterséges zajterhelés.....	24
5.2. Címkék feldolgozása.....	24
5.2.1. Címkék japán karakterekké alakítása	24
5.2.2. Kódolás	25
5.2.3. Példa kódolásokra.....	25
 6. HASZNÁLT HÁLÓK JELLEMZŐI.....	 25
6.1. Az adat előkészítése a hálózat tanítása előtt:	25
6.2. Első hálózat	27
6.2.1. Rétegek	27
6.3. Rectifier Activation(ReLU)	28
6.4. Softmax aktiváció.....	28
6.5. Hibafüggvény	29
6.6. Regularizációs módszerek.....	29
6.7. Második hálózat(konvolúciós).....	30
6.7.1. Rétegek	31
6.8. Harmadik hálózat.....	31
6.8.1. Rétegek	32
 7. HÁLÓKKAL ELÉRT EREDMÉNYEK.....	 32
7.1. Első hálózat	32
7.1.1. Hálózat teljesítménye egyetlen nagy batch-el 32x32-es példákon.....	33
7.1.2. Teljesítmény batch felosztással 32x32-es példákon.....	36
7.1.3. Teljesítmény 63x64-es példákon	39
7.2. Második hálóval elért eredmények	40
7.3. Harmadik hálóval elért eredmények.....	41
7.4. Hálózatok elért eredménye összesítve	42
 8. A FELHASZNÁLÓI FELÜLET	 42
8.1. Saját kép feltöltése.....	42
8.2. Saját kép rajzolása	43
8.3. Rajzolt vagy feltöltött kép felismerése	44
8.4. A megfelelő karakter kiválasztása	45
Irodalomjegyzék	48
Nyilatkozat	49

Bevezetés és összefoglalás

A gépi tanulás napjainkban mindenhol jelen van, legyen az önvezető autó vagy a közösségi hálók adatelemzése. Ennek egy fontos része a képi alakfelismerés, ahol a bemenet egy kép, és a kimenet változó a feladat függvényében. Lehet például hogy mit ábrázol egy kép, vagy hol helyezkedik el a képen egy adott tárgy. A következőkben részletezek egy lehetséges megoldást japán karakterek felismerésére, gépi tanulás, pontosabban mély neurális hálók segítségével.

A kutatást, illetve feladatvégzést többféle irodalmat felhasználva végeztem. Az olvasott irodalom többsége a neurális hálók felépítéséről és a velük kapcsolatos legjobb praktikákról szólt. Az alább felsorolt és jellemzett hálózatok teljesen saját készítésűek, ami gyakorlati tapasztalatot nyújtott a neurális hálózatok tanítása és kiértékelése terén.

A kitűzött feladatot az alábbiak szerint valósítottam meg:

1. Megterveztem a tanítóadat feldolgozását és kidolgoztam hogy milyen hálózatokat szeretnék majd tanítani.
2. A tanítóadatot kigyűjtöttem a megfelelő formában és úgy módosítottam Python-ban az OpenCV könyvtár használatával hogy a hálózat a legkönnyebben tudja feldolgozni.
3. A tervezett hálózatstruktúrákat Keras segítségével valósítottam meg Google Colab környezetben. Az adatot többféleképpen adtam át a hálóknak és a modellek felépítését módosítottam a legjobb eredmény megtalálása érdekében. Többek között a tanítóadatot felosztottam több részre, regularizációs módszereket alkalmaztam a tanuláskor és augmentáltam az adatot a jobb általánosítás céljával.
4. A modellek eredményeit összevetettem tesztadaton elért pontosság alapján és rangsoroltam őket, a legjobb háló 98 százalékos pontosságot ért el a tesztadaton továbbá az adatbázison kívüli karaktereket is kellő pontossággal felismerte.
5. Megterveztem egy weboldalt, ahol a hálózat kipróbálható feltöltött illetve rajzolt karakterek becslésére.
6. A tervezett oldalt megvalósítottam HTML, PHP és Javascript segítségével. Lehetővé tettem hogy a weboldal futtasson egy Python szkriptet a háttérben hogy a legjobban teljesítő neurális háló modelljét betöltse, majd becsléseket végezzen vele a bevitt karakteren.

A kutatásom során pont ezen az adatbázison más által elért eredményt nem találtam, ezért hasonlítási alapom nincsen. Viszont Yuhao Zhang által folytatott kutatásban[5], ahol japán helyett kínai karaktereken volt több hálózat betanítva, a legjobb eredmény 98 százalék volt a validációs halmazon. Zhang által végzett kutatás hasonlóan 200 példa tartozott minden osztályhoz, viszont az osztályok száma 3755 volt. Tehát elmondható hogy a saját hálók által produkált 97 százalékos eredmény az elvárásoknak megfelelt.

1. Gépi tanulás

1.1. Osztályozási problémák

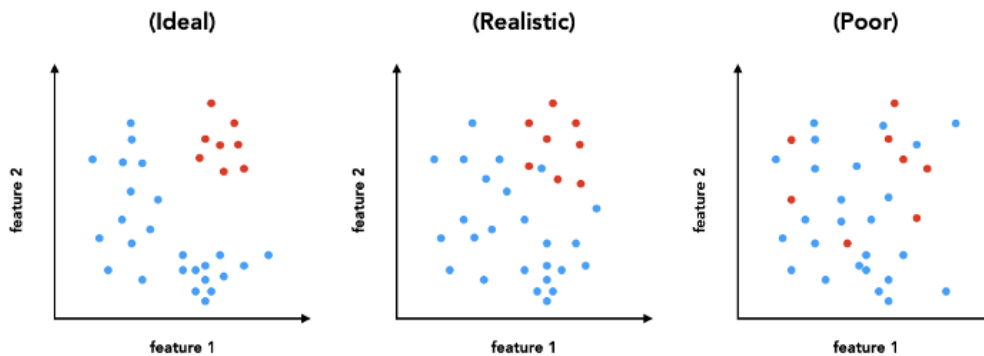
Az osztályozási probléma a gépi tanulás egyik legmeghatározóbb és leggyakoribb fajtája. A cél az objektum-példányok besorolása előre megadott osztályokba. Az objektum-példányok bizonyos tulajdonságokkal rendelkeznek, és ezek alapján kell besorolni őket. Az osztályozási probléma bemenetekből és kimenetekből áll.

- a. Bemenet: Valamilyen mérési adatokból álló (fix méretű) vektor.
 - i. Jellemzővektor, feature vector, attribútumvektor
- b. Tanítópéldák halmaza:
 - i. Jellemzővektorokból és hozzájuk tartozó osztálycímkekből álló párosok egy halmaza.
- c. Példa: A következő adatok alapján elemezni kell hogy a személynek jár-e biztosítás.

Életkor	Jövedelem(E Ft/hó)	Alapbetegség	Biztosítás jár
70	80	Érrendszeri	Nem
26	250	Nincs	Igen
33	130	Tüdő	Nem

Itt az *Életkor*, *Jövedelem* és az *Alapbetegség* oszlopok a jellemzővektor részei, míg a *Biztosítás jár* oszlop az osztálycímke. Ezen sorok összességét hívjuk tanítópéldányoknak.

d. A jellemzőtér



Ábra 1 A jellemzőtér minősége¹

Ha a jellemzővektor N komponensből áll, akkor a tanítópélda egy N -dimenziós tér pontjaként jeleníthető meg.

Itt a két jellemző az X és az Y tengelyek (x_1, x_2) .

A tanulási feladat tehát: Becslést adni az

$$(x_1, x_2) \rightarrow c$$

függvényre a tanítópéldák alapján.

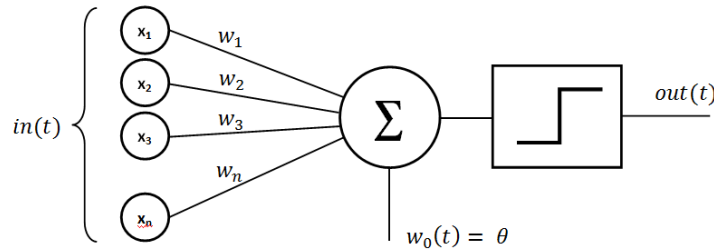
1.2. Az osztályozási problémák megközelítése

Az osztályozási problémát több szemszögből meg lehet közelíteni.

1.2.1. Perceptron modell

A legismertebb neurális modell, amit a biológiai neuronokhoz való hasonlósága miatt tartanak jónak. Több bemenete lehet, de csak egy kimenete.

¹ Forrás: Tom Grigg (2019): Concept Learning and Feature Spaces, <https://towardsdatascience.com/concept-learning-and-feature-spaces-45cee19e49db>, Letöltés időpontja: 2020.11.01



Ábra 2 Rosenblatt féle perceptron modell 1957-ből²

Több $x_1 \dots x_n$ bemenetből áll, melyekhez $w_1 \dots w_n$ súlyok tartoznak, továbbá van egy w_0 bias paraméter is. A neuront érő ingerek összességét aktivációnak hívjuk.

$$a = \sum_{i=0}^n w_i x_i + w_0$$

Az aktivációból a kimenetet az $o(a)$ aktivációs függvény állítja elő.

$$o = \left\{ 1 \text{ ha } \sum_{i=0}^n w_i x_i + w_0 > 0 \text{ különben } -1 \right\}$$

1.2.2. Neurális hálózatok

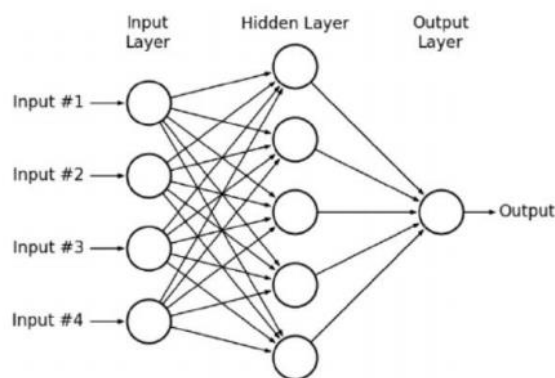
Neuron helyett neuronok hálózata szükséges. A klasszikus felépítés egy olyan többrétegű hálózat, ahol minden réteg az alatta lévő rétegtől kapja a bemenetet és minden neuron össze van kötve az utána lévő összes többi neuronnal.

1.3. Mesterséges neurális hálózatok (ANN)

Egy neuron csak egy egyenes behúzására képes a hipersíkban, emiatt az ÉS illetve VAGY operátorok megtanulhatók, viszont egy XOR operátor már nem.

Ezért a reprezentációs erő növelése érdekében jön be a képbe a neuronháló.

² Forrás: Jean-Christophe B. Loiseau (2019): Rosenblatt's perceptron, the first modern neural network, <https://towardsdatascience.com/rosenblatts-perceptron-the-very-first-neural-network-37a3ec09038a>, Letöltés időpontja: 2020.11.02



Ábra 3 Hagyományos neurális hálózat³

- Minden réteg az előtte lévő rétegtől kapja a bemenetet.
- Rétegek között teljes összeköttetés létezik, tehát minden neuron minden neuronnal össze van kapcsolva, ez az úgynevezett “fully-connected” hálózat.
- Minden kapcsolat előreutató.

A háló 3 fő részből áll: a bemeneti réteg, rejtett réteg és a kimeneti réteg.

Lehetséges még ritka „sparse” hálózatokat is készíteni, ahol nem minden neuron van összekötve minden neuronnal, továbbá rétegek átugrása is megoldható.

1.3.1. Előnyei

A hálózatok komplex modelleket és kapcsolatokat tudnak mind tanulni, mind modellezni. Remekül általánosítanak, a bemeneti példák alapján olyan összefüggéseket tudnak felfedezni az adatok között ami nem feltétlen triviális, majd ezeket még nem látott példákra tudják alkalmazni. További előnyük hogy sokféle adatot tudnak kezelni olyan dolgokat is figyelembe véve mint például időbeli egymásutánosság vagy elhelyezkedés egy képen.

1.3.2. Hátrányai

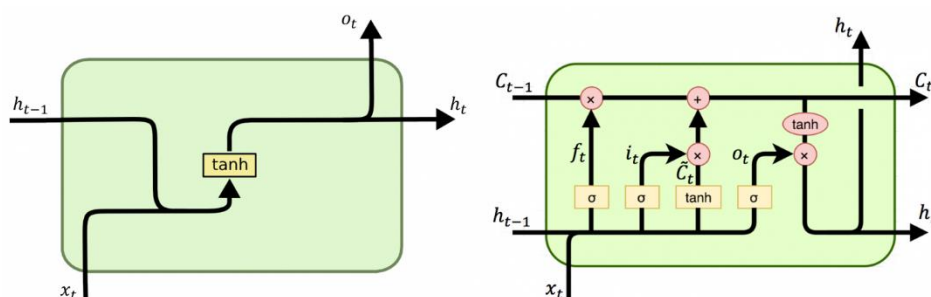
A hálózatok hátránya hogy igen adatigényesek, tehát sok adatra van szükség hogy jól tudjon általánosítani. Egy másik nagy hátránya hogy az egész folyamat black box, tehát egy nagyobb neurális hálózatnál szinte lehetetlen megmondani hogy mit hogyan tanul meg, éppen ezért maga a hálózat építése is nehéz lehet. Egy problémára sok különböző hálózatot érdemes építeni, majd összevetni az eredményeket, hiszen nincs csak egy jó megoldás.

³ Forrás: Ajit Jaokar(2019): The Mathematics of Data Science: Understanding the foundations of Deep Learning through Linear Regression, <https://www.datasciencecentral.com/profiles/blogs/the-mathematics-of-data-science-understanding-the-foundations-of>, Letöltés dátuma: 2020. 11. 05

1.3.3. Fajtái

Hálózatból is van többféle, van amelyik hasonló funkciót lát el, de többségük különféle feladatokra specializált.

- **RNN:** Visszacsatolt (rekurrens) neuronhálók figyelembe veszik az egymásutániságot a bemeneti adatban. Ez főleg az időbeli sorozatok modellezésénél fontos. Például beszédfelismerés, kézírás felismerése, árfolyamok becslése. A lényeg hogy egy bemenet környezetét is figyelembe kell venni, ezért tanuláskor a háló az adat körül lévő többi bemenetet is megfigyeli.
- **Konvolúció:** Alapvetően egy normális előrekapcsolt hálózatról van szó, viszont egy hagyományos teljes összeköttetésű hálózattal szemben itt a bemenet hierarchikusan van feldolgozva. Ez azt eredményezi, hogy a háló előrehaladtával kisebb részleteket tanul meg. A magasabb rétegek nagyobb részeket fednek le, viszont ezzel együtt a felbontás is romlik ezért kevésbé veszi figyelembe az apróbb részleteket. A konvolúciós hálók fő alkalmazási területei közé tartozik az képi alakfelismerés, de használják beszédfelismeréshez is.
- **Memory networks:** Alapvetően ezeknek a hálózatoknak van egy hosszútávú memóriája, amibe írni és amiből olvasni tudnak. Erre egy remek példa az LSTM (Long short-term memory) hálók, ahol azt kell tanítani hogy a háló mit felejtson, és mit jegyezzen meg a korábbi bemenetekből. Az információ alapvetően több párhuzamos kapun keresztül áramlik.
 - Forget gate: Szabályozza mit töröljön a memória.
 - Input gate: Szabályozza mire kell emlékezni a bemenetből.
 - Output gate: Szabályozza hogyan álljon össze a kimenet.



Ábra 4 RNN és LSTM⁴

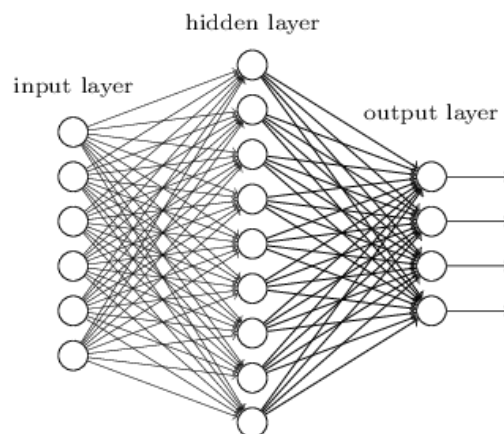
⁴ Forrás: Dprogrammer (2019): RNN, LSTM & GRU, <http://dprogrammer.org/rnn-lstm-gru>, Letöltés dátuma:2020.11.07

1.4. Mély neuronhálók (DNN)

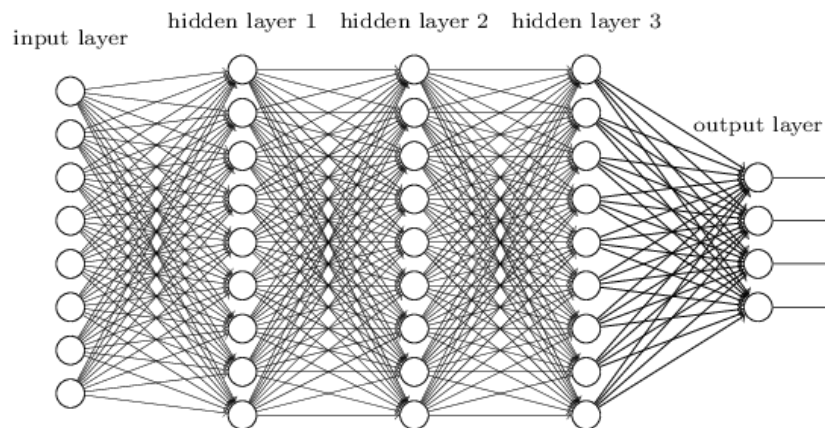
A mély neuronhálók abban különböznek a hagyományos hálóktól hogy a mély hálóknban több rejtett réteg van, ellenben egy hagyományos hálóban csak egy. Ha már egynél több rejtett réteget tartalmaz egy háló, akkor mélyhálóról van szó. Gyakorlatban nem csak két rejtett réteg lehetséges, leggyakoribbak az 5-10 de léteznek 100-150 rejtett rétegű hálók is.

Az alapötlet egyszerű, mégis csak az utóbbi időkben lett megvalósítható. Kezdetben nem voltak meg az algoritmusok amik lehetővé tették volna a több rejtett réteg használatát. Az első ilyen algoritmus a deep belief network előtanítása volt 2006-ban. Egy másik ok a késői megvalósításra a tanítóadatok hiánya. Egy mélyháló nagyon sok tanítóadatot igényel, ami régebben nem volt elérhető. Végül az egyik legnyomósabb ok a számításigény, a hardverek limitáltsága miatt nem volt opció a mélyháló, viszont a mostani videokártyák és processzorok már gyorsan tanulnak egy komplexebb hálózattal is.

De miért jobb a mélyháló? Két rejtett réteggel minden feladat megoldható, viszont ez csak akkor igaz ha minden körülmény optimális. Sok tanítóadat is kell, rengeteg neuron és a globális optimumot adó tanító algoritmus. Ezért jobb gyakorlat az hogy 1-2 “széles” réteg helyett sok “keskeny” rétegű a hálózat. Ez azt is lehetővé teszi hogy a háló hierarchikusan dolgozza fel a kapott adatokat. Hátránya viszont, hogy tanítása jóval nehezebb mint a hagyományos hálóké.



Ábra 5 Példa hagyományos hálóra



Ábra 6 Példa mélyhálóra⁵

1.5. ANN az osztályozási feladatokhoz

A mesterséges neurális hálózat remek általánosítóképessége miatt tökéletes osztályozási feladatokhoz, remek példa erre a CIFAR-10 adathalmaz⁶. Ami 60000 32x32-es képből áll különféle tárgyakról és élőlényekről, összesen 10 különböző osztály van, és minden osztályhoz 6000 példa tartozik. A CIFAR-10 látványos szemléltetője annak hogy mennyire is jól tud általánosítani egy neurális háló, a 10000-es teszhalmazon akár 96 százalékos pontosság is elérhető⁷.

2. Karakterfelismerés (OCR)

OCR(Optical character recognition) vagy optikai karakterfelismerés az egyik legrégebbi képfeldolgozási probléma, ahol nyomtatott illetve kézzel írt karaktereket szeretnének feldolgozni elektronikusan, vagy mechanikusan. Főként adatok beolvasására és nyomtatott irodalom digitalizálására használják. Mostanra már nagyon kifinomult módszerek vannak arra hogy nagy pontossággal karaktereket lehessen felismerni, akár 95-99 százalékos pontossággal nyomtatott betűknél. A probléma folyóírásnál jön elő, ahol nehéz jól általánosítani. Fontos az OCR-nél az egységesítés, tehát megpróbálni a karaktereket hasonló alakra hozni, legyen az felbontás vagy kinézet (például bináris fekete-fehér).

⁵ Forrás: Michael Nielsen (2019): Why are deep neural networks hard to train?, <http://neuralnetworksanddeeplearning.com/chap5.html>, letöltés dátuma:2020.11.07

⁶ <https://www.cs.toronto.edu/~kriz/cifar.html>

⁷ <https://paperswithcode.com/paper/wide-residual-networks>

2.1. Neurális hálók OCR-hez

Mivel az OCR régi probléma ezért nem feltétlen szükséges neurális háló a megoldásához, viszont ezzel jobb eredmény érhető el kevesebb erőfeszítéssel. Több példa is mutatja hogy neurális hálókkal kiváló karakterfelismerő készíthető.

2.1.1. Példák

SVHN(Street View House Numbers): Egy házszám adatbázis 600 000 különböző példával.



Ábra 7 SVHN példák⁸

Ez az adatbázis bemutatja hogy még ha sok különböző adat is van, a hálózat remekül képes általánosítani, ha megvan a kellő mennyiségű tanítópélda. Megfelelő hálózati struktúrával akár 98 százalékos pontosság is teljesíthető. A példák Google Street View-ből lettek kinyerve és azóta is egy népszerű példája a neuronhálók tanításának.

MNIST:⁹



Ábra 8 MNIST példák

⁸ Forrás: <http://ufldl.stanford.edu/housenumbers/>

⁹ Forrás: <http://yann.lecun.com/exdb/mnist/>

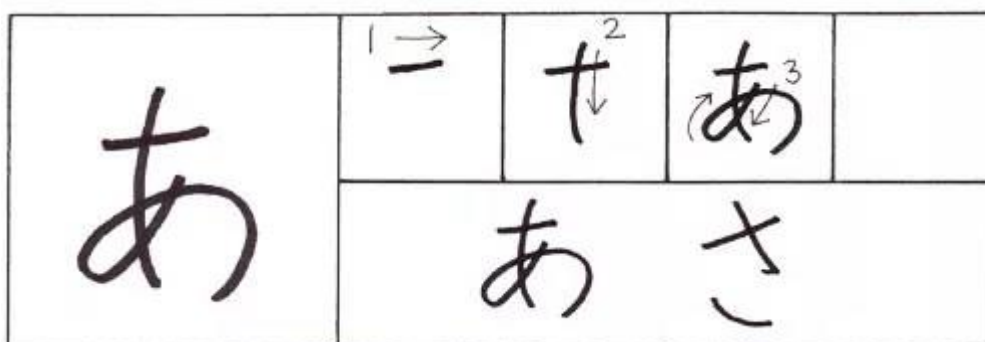
Az MNIST egy kézzel írt számokból álló adatbázis 70 000 példával, a NIST adatbázis egy részhalmaza. Rengeteg háló illetve osztályozó létezik hozzá, már szinte viszonyítási alapként használják. Konvolúciós hálókkal 99,67 százalékos eredményt értek el, de egy viszonylag egyszerű háló is könnyen meg tudja ütni a 94 százalékos szintet.

3. A japán karakterek

A japán karakterek alapvetően 3 csoportba sorolhatók.

3.1. Hiragana

Szótagírás ami a 800-as években jelent meg. A karaktereket kínai írásból származtatták. Így jött létre a mára 46 karakterből álló hiragana. A karakterek egyszerűek, a legbonyolultabb is maximum csak 4 vonásból áll.

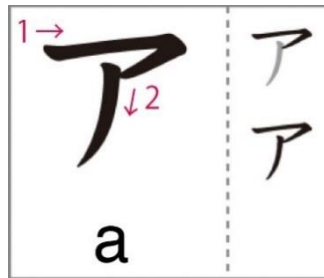


Ábra 9 Hiragana "a"¹⁰

3.2. Katakana

Hasonlóan a hiraganához a katakana is szótagírás. Minden hiragának van egy katakana megfelelője, ezen felül van még kettő, ami hiraganában nem létezik, így van összesen 48 írásjel. Ezek a karakterek alapvetően a külföldi eredetű szavak írásához használatosak. Például külföldi nevek, városok és jövevényszavak.

¹⁰ Forrás: Abe, Namiko(2020): A Hiragana Stroke Guide to あ、い、う、え、お (A, I, U, E, O). ThoughtCo. <https://www.thoughtco.com/how-to-write-hiragana-a-i-u-e-o-2027939> , Letöltés dátuma: 2020.11.11

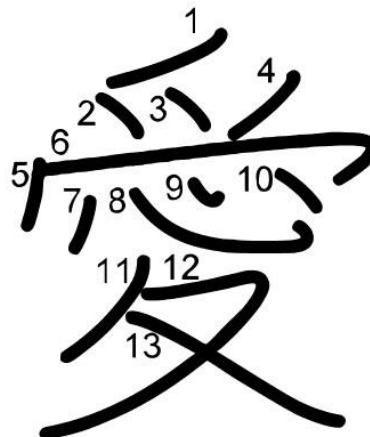


Ábra 10 Katakana "a"¹¹

3.3. Kanji

A kanji a kínaiaktól az ötödik században átvett logografikus írásmód. A kanji-k többsége mára egyszerűsödött, kevesebb vonásból állnak és nem annyira bonyolultak. A mindennapi életben használt kanji-k száma 2 136, ezek az úgynevezett jouyoukanji-k (amelynek jelentése gyakran használt kanji-k), de több mint 50 000 karakter létezik, többségük csak szinoníma amiket nem használnak.

Egy kanji-nak több olvasata is lehet, többségük 3-5 olvasattal rendelkezik, néhány karakternek azonban lehet akár 20-30 is. Az olvasatoknak két csoportja van, kínai olvasat és japán olvasat. Az első értelemszerűen a kínai nyelvből lett átvéve, az utóbbit pedig később adták hozzá.



Ábra 11 "ai" kanji¹²

¹¹ Forrás: (2015): Drawing Katakana (Part 1), <https://thewholenewlive.wordpress.com/2015/02/25/> , Letöltés dátuma: 2020.11.11

¹² Forrás: <http://kanji.nihongo.cz/kanjipix/12326/l/en>, Letöltés dátuma: 2020.11.11

3.4. Japán karakterek osztályozásban

A fő problémát a karakterek száma, illetve hasonlósága jelenti. Csak a mindennap használatos kanji-k csoportját nézve is 2136 különféle karakter ami ugyanennyi osztályt jelent. A különbség pedig néhány esetben csak pár vonás.



Ábra 12 Hasonló karakterek

A kanji-k összetétele nagyon hasonló, ezért ugyanazok az elemek jelennek meg több karakterben is. Ez főleg akkor probléma ha a karaktereket ábrázoló képeket skálázni kell helymegtakarítás, vagy gyorsítás céljából.

A japán karakterek osztályozási nehézsége igazán csak a kanji-knál jelenik meg. Hiraganából és katakanából nincs olyan sok, ezért nem kell olyan sok osztállyal foglalkozni, éppen ezért hiraganára és katakanára is viszonylag egyszerű jól teljesítő modellt készíteni.

4. A tanítóadatok

A kiválasztott tanítóadat az ETLCDDB, ami egy 1980-ban írt és 1984-ben adatbázisba szedett karakterek gyűjteménye. Megkértek 4000 embert hogy fejenként írjanak le körülbelül 152 előre megadott karaktert, így jött létre a végül 607 200 karaktert tartalmazó adatbázis. Az adatbázis összesen 2965 féle kanjit és 71 féle hiraganát tartalmaz, utóbbiból azért van ennyi mert majdnem az összes karakterhez van egy kis és egy nagy méretű verzió. Minden karakterből 200 példa van, így kapjuk meg a 607 200 darab tanítóadatot.

A tanítóadatnak kétféle verziója érhető el.

4.1. Szürkeárnyaltos:

Az eredeti képek szkennelt változatánál minden egyes példa 8199 bájt, viszont ebben a formában több olyan adat is van ami nem releváns. A 8199 bájt a következőképpen épül fel:

Bájt pozíció	Bájtok száma	Típus	Tartalom
1 - 2	2	Integer	Sorozatszám (≥ 1)
3 - 4	2	Binary	JIS kanji kód (JIS X 0208)
5 - 12	8	ASCII	Tipikus JIS olvasat (pl. „AI.MEDER”)
13 - 16	4	Integer	Adatszám (≥ 1)
17	1	Integer	Karakter értékelése (≥ 0)
18	1	Integer	Karakter csoport értékelése (≥ 0)
19	1	Integer	Karaktert író neve (JIS X 0303)
20	1	Integer	Karakert író életkora
21 - 22	2	Integer	Ipar besorolás kódja (JIS X 0403)
23 - 24	2	Integer	Foglalkozás besorolás kódja (JISX0404)
25 - 26	2	Integer	Írás időpontja (19)ÉÉHH
27 - 28	2	Integer	Szkennelés időpontja (19)ÉÉHH
29	1	Integer	Karakter X pozíciója a lapon (≥ 0)
30	1	Integer	Karakter Y pozíciója a lapon (≥ 0)
31 - 64	34		Nincs meghatározva
65 - 8192	8128	Packed	Szürkeárnyaltos kép adatai (4bit/pixel) 128 x 127 = 16256 pixel
8193-8199	7		Nincs meghatározva

Jól látszik hogy itt rengeteg felesleges adat van az osztályozás szempontjából, a tanításhoz csak a *JIS kanji kód* a *sorozatszám* és a *kép adatai* fontosak. Egyszerűen ki lehet nyerni ezeket az adatokat az összesített adatbázisból, viszont célszerű bináris képen dolgozni, a mérete is kisebb és a tárolása is egyszerűbb. A 13-as ábrán néhány példa látható az adatbázisból.



Ábra 13 Példák a szürkeárnyaltos képekre

4.2. Bináris:

A bináris verzió a szürkeárnyaltosból lett kinyerve küszöböléssel.

A szürkeárnyaltos verzióval szemben, a bináris verzió sokkal kevesebb felesleges adatot tartalmaz és a képek mérete sem túl nagy. A bináris verziónál egy példa 576 bájt. Egy példa felépítése a következő:

Bájt pozíció	Bájtok száma	Típus	Tartalom
1 - 2	2	Integer	Sorozatszám (≥ 1)
3 - 4	2	Binary	JIS kanji kód (JIS X 0208)
5 - 8	4	ASCII	Tipikus JIS olvasat (pl. „A.I.M”)
9 – 512	504	Packed	Bináris kép $63 \times 64 = 4032$ pixel
513 – 576	64	-	Nincs meghatározva

Sokkal egyszerűbb feldolgozni a bináris alakot, csupán az utolsó 64 bájtot és az olvasatot kell elhagyni hogy a kívánt formára jusson az adat.

4.3. Problémák

Az adatot megfelelő alakra kell hozni mert az eredeti formával több probléma is van.

- Csak öt fájlra van felosztva, ami túl nagy fájlokat eredményez.
- A JIS kanji kódok túlságosan terjedelmesek, a kódok 0-tól körülbelül 21000-ig terjednek, viszont rengeteg helyen rés van köztük (például 900 után egyből 6000 következik) a cél ezt normalizálni hogy az a szám megegyezzen az osztályok számával(3037).

A túl sok adatra a legjobb megoldás a *batch-ek* létrehozása, tehát felosztani az öt nagy fájlt, sok kisebb fájlra. Több mérettel is érdemes próbálkozni mert a tanulást is nagyban

befolyásolja. Az hogy mennyi adat legyen egy batch-ben preferencia kérdése, viszont érdemes kettőnek a hatványaival próbálkozni.

4.3.1. Adatok felosztása

Az adatok felosztása a következőképpen történik:

1. Az összes adat betöltése és eltárolása.
2. Keverés elvégzése az adaton hogy ne legyenek sorban.
3. Az adatok/címkék elmentése új .csv formátumú fájlba.
4. Az új nagy .csv-ket szétválasztjuk több kisebbre (batch számnak megfelelően)

4.3.2. Címkék normalizálása

A kódolás problémája megoldható normalizálással. Ennek a lépései a következők:

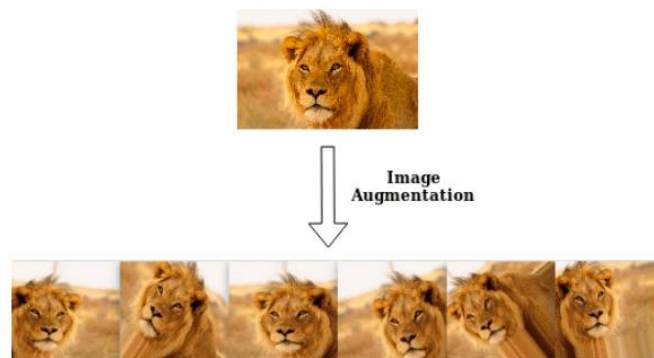
1. Eltárolni az összes címkét egy *dictionary*-ben és mindegyikhez számot hozzárendelni növekvő sorrendben.
2. A *dictionary*-t elmenteni egy *dictionary.csv* fájlba, majd ezt használva normalizálni a már meglévő címkéket.
3. Ez később lesz fontos, mikor one-hot kódolásra kell hozni a címkéket, a one-hot körülbelül 20000 osztályba sorolta volna a címkéket, de így normalizálva már helyesen csak 3037 osztályt hoz létre.

Ez az adatbázis remek alapot ad egy jó osztályozási feladathoz, hiszen az adat teljesen egyenletes. Minden karakterből pontosan 200 példát tartalmaz, ezért nem nagyon kell foglalkozni azzal hogy extra adat legyen létrehozva, viszont az általánosítás eléréséhez jó gyakorlat lehet az adat augmentáció elvégzése a már meglévő példákon.

5. Adat augmentáció

Az adat augmentáció egy olyan módszer ahol a már alpból meglévő adatot valamilyen módosítás elvégzése után új adat kerül létrehozásra, ezáltal növelve a tanítóadatbázis méretét. Segít csökkenteni a túltanulás esélyét, tehát regularizációs módszerként alkalmazható.

Mivel a példák karakterek, ezért az adat augmentáció egy kicsit limitáltabb. Ha tárgyakat vagy élőlényeket kellene osztályozni, akkor több opció lenne, viszont írott karaktereknél a lehetőségek közül kiesik a tükrözés és a legtöbb forgatás.



Ábra 14 Példa adat augmentációra¹³

Ha az osztályozási feladat nem írott karakterekre vonatkozik akkor több opció is lehetséges, forgatás, tükrözés, nyújtás és eltolás, ezt szemlélteti a 14-es ábra.

5.1. Adat augmentációs technikák írott karakterekre

A következőkben négy különféle adat augmentációs módszer van részletezve írott karakterekre. Elforgatás, x illetve y tengelyeken való eltolás és só-bors zaj hozzáadása. Egyszerű módja annak hogy több tanítópélda legyen, viszont ügyelni kell az egyenletes eloszlásra az új és a régi között, például az ETLCDDB esetében 200 példány van egy karakterre, akkor az augmentált karakterekből is érdemes hasonló számú variációt létrehozni.



Ábra 15 Eredeti példák

¹³ Forrás: Shubham Goyal(2019): MachineX: Image Data Augmentation Using Keras , <https://towardsdatascience.com/machinex-image-data-augmentation-using-keras-b459ef87cd22> , Letöltés dátuma: 2020.11.15

5.1.1. Forgatás

A következő Python-ban írt rövid függvény megkeresi a kép középpontját majd elforgatja valamennyi fokkal, ha az *angle* változó pozitív akkor óramutató járásával ellentétesen, ha pedig negatív akkor az óramutató járásával megegyezően.

```
def rotate_image(image, angle):
    image_center = tuple(np.array(image.shape[1::-1]) / 2)
    rot_mat = cv.getRotationMatrix2D(image_center, angle, 1.0)
    result = cv.warpAffine(image, rot_mat,
        image.shape[1::-1], flags=cv.INTER_LINEAR)
    return result
```

Szükséges az OpenCV-ben található warpAffine függvény az affin transzformációhoz. Egy kép forgatása θ fokkal a következő transzformációs mátrixsal történik:

$$M = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$$

Viszont OpenCV-ben van egy skálázott forgatás is, ahol be lehet állítani a forgáspontot, ez a következő mátrixban van megadva:

$$\begin{bmatrix} \alpha & \beta & (1 - \alpha) * \text{középpont}.x - \beta * \text{középpont}.y \\ -\beta & \alpha & \beta * \text{középpont}.x + (1 - \alpha) * \text{középpont}.y \end{bmatrix}$$

ahol:

$$\alpha = \text{skálázás} * \cos \theta$$

$$\beta = \text{skálázás} * \sin \theta$$

Ehhez a transzformálási mátrixhoz kell az OpenCV getRotationMatrix2D függvénye. A 16-os ábrán látható példák -20 fokos elforgatás hatására ilyenek.



Ábra 16 Elforgatott példák

5.1.2. Eltolás

Python-ban írt függvény képes az x és az y tengelyen eltolni a képet. Az *image* változó a bemeneti kép, az *amount* a mérték ami minél nagyobb annál kisebb értékkel tolja el a képet, és a *direction* ami ha 0 akkor y tengely, ha 1 akkor x tengely mentén tolja el a képet.

```
def translate_image(image, amount, direction):
    height, width = image.shape[:2]
    quarter_height, quarter_width = height / amount, width / amount
    if(direction == 0):
        T = np.float32([[1, 0, 0], [0, 1, quarter_height]])
    else:
        T = np.float32([[1, 0, quarter_width], [0, 1, 0]])
    img_translation = cv.warpAffine(image, T, (width, height))
    return img_translation
```

Az eltolás a következőképpen működik: ha tudjuk az (x, y) koordinátákat amerre el szeretnénk tolni a képet, akkor legyen (t_x, t_y) és ebből hozzuk létre a következő transzformációs mátrixot.

$$T = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$$

A 17-es ábrán látható a 10 pixeles eltolás az x tengelyen pozitív irányban, míg a 18-as ábrán az y tengelyen való negatív eltolás látszik 10 pixellel.



Ábra 17 x tengelyen eltolt példák



Ábra 18 y tengelyen eltolt példák

5.1.3. Mesterséges zajterhelés

A következő Python függvény mesterségesen ad hozzá só-bors zajt a képekhez.

```
def add_noise(img):  
    noise = np.zeros(img.shape,np.int16)  
    cv.randu(noise,-100.0,100.0)  
    imnoise = cv.add(img,noise,dtype=cv.CV_8UC1)  
    return imnoise
```

A függvény egyetlen paramétere az *img* változó, ami a kép, amihez a zajt hozzáadja. Létrehoz egy *noise* nevű változót, aminek mérete megegyezik a bemeneti kép felbontásával, majd az OpenCV beépített *randu* függvény segítségével egyenletes eloszlású zaj képet generál ugyanebbe a változóba. A zajt hozzáadja a képhez mátrix összeadás használatával. Az eredmény a 19-es ábrán látható mesterséges zajjal ellátott képek.



Ábra 19 Mesterséges zajjal ellátott példák

5.2. Címkék feldolgozása

A címkék feldolgozása jóval egyszerűbb mint a képeké, mivel alapból JIS kód formátumban található az adatbázisban. Az hogy melyik JIS kód melyik karakternek felel meg könnyen kinyerhető egy Python függvény segítségével.

5.2.1. Címkék japán karakterekké alakítása

A címkék decimális formában érhetőek el, az első lépés hogy hexadecimálisra kell konvertálni a kívánt karakter címkéjét.

```
format(codes[i], 'x')
```

A *format* függvény megadja a decimális karakter hexadecimális alakját a 0x prefix nélkül.

```
bytes.fromhex( '1b2442' + str(char[i]) + '1b2842').decode('iso2022_jp')
```

Hexadecimális alakból kinyeri a bájtokat majd az egészet dekódolja iso2022 kódolás szerint, ezek után az érték maga a japán karakter lesz.

5.2.2. Kódolás

A címkék 0-tól 3036-ig terjednek. Jobb alakra lehet hozni őket one-hot kódolás segítségével. A one-hot kódolás lényege hogy kategorikus adatot számadattá alakít át, így a hálózat is egyszerűbben tudja kezelni. Néhány algoritmus jól bánik a kategorikus adattal is, ilyenek például a döntési fák.

5.2.3. Példa kódolásokra

A példában szereplő adatokat három külön osztályba lehet sorolni.

- Kategorikusan: piros , zöld , kék
- Egész számmal: piros lesz az 1 , zöld lesz a 2 és kék lesz a 3
- One-hot kódolással:

piros	zöld	kék
1	0	0
0	1	0
0	0	1

Könnyen reprezentálható egy tömbben hogy egy adat melyik osztályhoz tartozik, például {1,0,0} megfelel egy piros osztályba tartozó adatnak.

6. Használt hálók jellemzői

6.1. Az adat előkészítése a hálózat tanítása előtt:

A korábban tárgyalt .csv fájlokat használja a háló az adat beviteléhez és feldolgozásához. Fontos az adatok megkeverése minden tanításkor mert ezzel biztosítja hogy nem ugyanazokat a példákat tanulja a háló. Ezt a következő függvény végzi:

```
def unison_shuffled_copies(a, b):
    assert len(a) == len(b)
    p = np.random.permutation(len(a))
    return a[p], b[p]
```

A függvény két paramétert vár, *a* ami egy adatokat tartalmazó tömb és *b* ami az adatokhoz tartozó címkék tömbje. Az assert kulcsszó biztosítja hogy *a* és *b* hosszúsága megegyezik. Majd egy numpy függvény segítségével véletlenszerű sorrendet generál a tömböknek és visszatér az *a* és *b* átrendeztet alakjával.

```
def load_data(idx, batch_size):
    df = pd.read_csv("/content/drive/My Drive/data.csv",
                     skiprows=idx * batch_size, nrows=batch_size)
    lb = pd.read_csv("/content/drive/My Drive/labels.csv",
                     skiprows=idx * batch_size, nrows=batch_size)

    return (np.asarray(df, dtype="uint8") .
            reshape(batch_size, 63, 64, 1),
            np.array(tf.keras.utils.to_categorical(lb, num_classes=3037),
                     dtype="uint8"))
```

A load_data függvény tölti be az adatokat a .csv fájlkból. Két paramétere az *idx* és a *batch_size*. Az *idx* adja meg a batch sorszámát, míg a *batch_size* a batch méretét és hogy hány adattal térjen vissza. A visszatérési érték az adat és a hozzá tartozó címke one-hot kódolásban.

```
def batch_generator(batch_size, steps, starting):
    idx = starting
    while True:
        yield load_data(idx, batch_size)
        if idx < steps:
            idx+=1
        else:
            idx = starting
```

A load_data függvényre a batch generátorban volt szükség. Ez a függvény adja át a hálózathoz az adatokat.

Három paramétere van, *batch_size*, *steps* és *starting*. A *batch_size* adja meg hogy hány adat van egy batch-ben, a *steps* hogy hány batch-et adjon át a generátor a hálózathoz, a *starting* pedig megadja a kezdő batch sorszámát.

Egy generátor a következőképpen hozható létre:

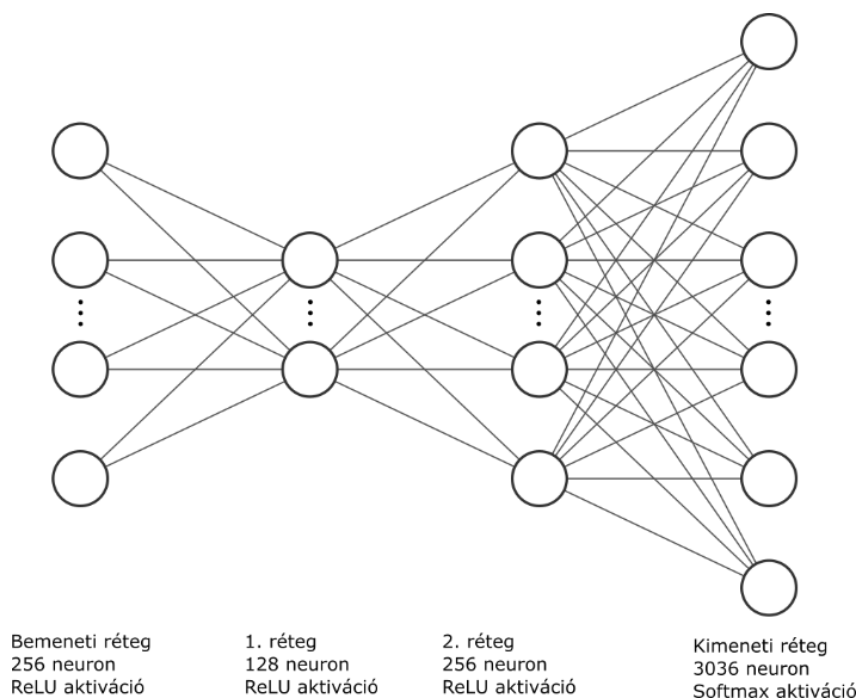
```
training = batch_generator(759, 485760/batch_size, starting=0)
```

A fenti kódrészlet létrehozza a training batch generátort, ahol egy batch mérete 759, a tanítóadat száma 485 760 , ez leosztva a batch méretével megadja a szükséges lépések számát és a *starting* pedig megadja hogy az elejétől kezdje a számlálást.

A hálónál fontos a „trial and error” avagy próbálkozás minél több felépítéssel, topológiával és különböző paraméterekkel hogy kiderüljön melyik az adott problémához legjobban illő.

6.2. Első hálózat

A legelső hálózat felépítése egy egyszerű négy rétegből álló előrekapcsolt struktúra, fully-connected tehát minden neuron össze van kötve az összes őt követő másik neuronnal.



Ábra 20 Első modell felépítése

6.2.1. Rétegek

- A bemeneti réteg: bemenetként megkapja a képek pixelét, 256 neuronból áll és ReLU aktivációt használ.

- Első(rejtett) réteg: 128 neuronból áll, ReLU aktivációt használ.
- Második(rejtett) réteg: 256 neuron úgyszintén ReLU aktivációval.
- Kimeneti réteg: a 3037 osztálynak megfelelő számú neuron, Softmax aktivációval.

A használt aktivációs függvények a köztes rétegeken rectifier (ReLU) míg az utolsó rétegen softmax volt.

```
model = Sequential()
model.add(Dense(256, input_dim=input_dim, activation='relu',))
model.add(Dense(128, input_dim=256, activation='relu'))
model.add(Dense(256, input_dim=128, activation='relu'))
model.add(Dense(output_dim, input_dim=256, activation='softmax'))
```

6.3. Rectifier Activation(ReLU)

A ReLU az egyik leghasználtabb aktivációs függvény a sigmoid és a tanh mellett. Előnyei közé tartozik hogy csökkenti az eltűnő/felrobbanó gradiensek esélyét, viszont a felrobbanó aktiváció még így is lehetséges.

$$h = \max(0, a)$$

ahol

$$a = Wx + b$$

A függvényen látható hogy nullát is felvehet. Ennek a hátulütője az úgynevezett „halott ReLU” ahol túl sok nulla következtében egyszerűen nem fog tanulni, vagy hátráltatja a háló tanulását.

6.4. Softmax aktiváció

A kimeneti réteghez ajánlott a softmax aktivációs függvény használata:

$$\sigma(a)_j = \frac{e^{a_j}}{\sum_{k=1}^M e^{a_k}} \text{ for } j = 1, \dots, M$$

Tehát a kimenetek értéke így biztosan 0 és 1 közé esik, továbbá összegük 1 lesz ezért alkalmas valószínűségi becslésekhez.

6.5. Hibafüggvény

Egy háló kiértékelésekor az elért pontosságot vagy hiba arányát és a hibafüggvény értékét érdemes figyelembe venni. Mivel osztályozási feladatról van szó ezért célszerű a keresztentrópia hibafüggvényt alkalmazni:

$$E(w) = -\frac{1}{N} \sum_{d \in D} \sum_{k \in \text{outputs}} t_{kd} \ln(o_{kd})$$

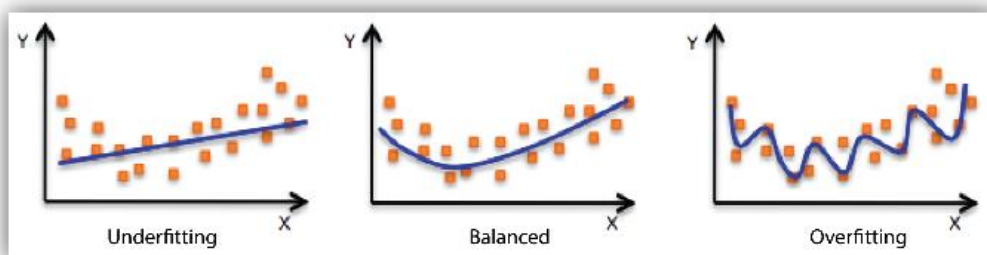
Eredetileg diszkrét eloszlások eltérésének mérésére találták ki. A one-hot kódolás miatt a t értéke csak 1 vagy 0 lehet ezért a célfüggvény következőképpen egyszerűsödik:

$$E(w) = -\frac{1}{N} \sum_{d \in D} \ln(o_{\text{correct},d})$$

„Ez akkor lesz minimális, ha a helyes osztályhoz tartozó kimenet minél inkább 1-hez közelít, mivel a softmax függvény révén a kimenetek össze vannak kötve, ez csak úgy lehetséges, ha az összes többi osztályhoz tartozó értéke 0-hoz közelít.”[1]

6.6. Regularizációs módszerek

Regularizációt a túltanulás legküzdésére szokás alkalmazni. Túltanulás az amikor a modell nem általánosít, hanem csak megjegyzi a példákat. Ennek ellentéte az alultanulás, ahol túlságosan is általánosít a modell.



Ábra 21 Alultanulás, Kiegyensúlyozott, Túltanulás¹⁴

¹⁴ Forrás: Amazon Web Services: <https://docs.aws.amazon.com/machine-learning/latest/dg/model-fit-underfitting-vs-overfitting.html>, letöltés dátuma: 2020.12.05

Többféle regularizációs módszer létezik, ezek közül az egyik a már fentebb tárgyalt adat augmentáció, ahol új adat van létrehozva mesterségesen a már meglévőből, ezzel is csökkentve a túltanulás esélyét.

A dropout olyan regularizációs módszer ahol van egy „dropout réteg” ami valamennyi valószínűséggel kiejt kapcsolatokat az egyébként teljesen összekapcsolt hálózatból. Ez arra készíti a neuronokat hogy magukra hagyatkozzanak többet, és ne a többi neuronra.

Léteznek még L1 és L2 regularizációs módszerek. L2 az egyik legegyszerűbb regularizációs módszer, ahol a hálót kis súlyok preferálására készíti azzal hogy bünteti a nagy súlyokat. L1 már egy kicsit agresszívbabban regularizál, itt a súlyok nem egynél kisebb konstanssal vannak szorozva, hanem egy kis konstans van hozzáadva illetve kivonva, ez akár azt is eredményezheti hogy a súlyokat kinullázza.

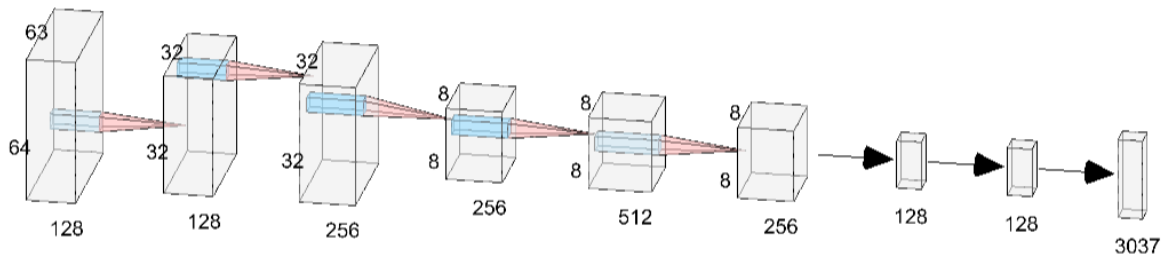
6.7. Második hálózat(konvolúciós)

A második struktúra egy konvolúciós hálózat. A fully-connected hálózattal szemben itt a jellemzők elrendezése szerepet játszik a háló tanulásában. Ez főleg képi alakfelismerésben jelentkezik, ahol a pixelek összekeverése más képet eredményez, ezt viszont egy fully-connected háló ugyanúgy tanulná meg.

Tehát a konvolúciós háló egy fontos jellemzőt használ fel a képeknél, a pixelek egymáshoz való viszonyát. A képek alapvetően hierarchikusan épülnek fel és a korai hálók ezt nem tudták rendesen megtanulni. Ennek hatására született meg a konvolúciós neuronháló, ami az inputokat hierarchikusan dolgozza fel. Az elején csak a kép kisebb részleteire koncentrál, de ahogy halad rakja össze a részeket. Ahogy növekszik a kép úgy romlik a felbontás is, így az apró részletek elveszhetnek.

A konvolúciós neuronháló felépítése a következő:

- konvolúciós neuronok (szűrők)
- pooling (egyesítés)
- ezek ismétlése
- a háló tetején néhány fully connected réteg, majd a kimenet



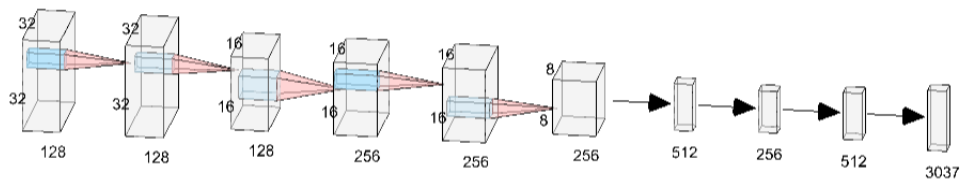
Ábra 22 Második modell felépítése

6.7.1. Rétegek

```
model=Sequential()
model.add(Conv2D(filters=128,kernel_size=(3,3),input_shape=(63,64,1),
, strides=(1,1),padding='same',activation='relu'))
model.add(MaxPool2D(pool_size=(3,3),strides=(2,2)))
model.add(Dense(128,activation='relu'))
model.add(Conv2D(256,kernel_size=(3,3),input_shape=(32,32,1),
, strides=(1,1),padding='same',activation='relu'))
model.add(MaxPool2D(pool_size=(3,3),strides=(2,2)))
model.add(Dense(128,activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(128,activation='relu'))
model.add(Conv2D(512,kernel_size=(3,3),input_shape=(16,16,1),
, strides=1,padding='same',activation='relu'))
model.add(MaxPool2D(pool_size=(3,3),strides=(2,2)))
model.add(Dense(128,activation='relu'))
model.add(Flatten())
model.add(Dense(128,activation='relu'))
model.add(Dense(units=output_dim,activation="softmax"))
```

6.8. Harmadik hálózat

A harmadik hálózat szintén egy konvolúciós háló 12 réteggel, ahol csak az utolsó rétegek teljes összeköttetésűek.



Ábra 23 Harmadik modell felépítése

6.8.1. Rétegek

```
model=Sequential()

model.add(Conv2D(filters=128,kernel_size=(3,3),input_shape=(32,32,1),
, strides=(1,1),padding='same',activation='relu'))
model.add(Conv2D(filters=128,kernel_size=(3,3),strides=(1,1),
padding='same',activation='relu'))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(256,kernel_size=(3,3),strides=(1,1),padding='same',
activation='relu'))
model.add(Conv2D(256,kernel_size=(3,3),strides=(1,1),padding='same',
activation='relu'))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Flatten())
model.add(Dense(512,activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(256,activation='relu'))
model.add(Dense(512,activation='relu'))
model.add(Dense(units=output_dim,activation="softmax"))
```

7. Hálókkal elért eredmények

7.1. Első hálózat

Már az első háló is jó eredményeket ért el annak ellenére hogy egy igen egyszerű modelltől van szó. A hálózat egyik rejtett rétegében kevesebb neuron van hogy segítse az általánosítást. Az eredmények több részre vannak osztva a könnyebb összehasonlítás érdekében.

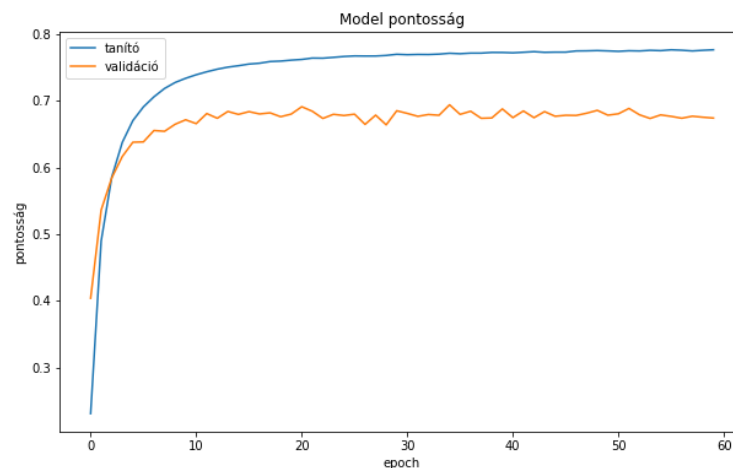
- az eredeti 63x64-as képek
 - regularizáció nélkül
 - regularizációval

- csökkentett méretű 32x32-es képek
 - regularizáció és adat augmentáció nélkül
 - regularizáció nélkül de adat augmentációval
 - regularizációval de adat augmentáció nélkül
 - regularizációval és adat augmentációval együtt

A tanítóadat 8:1:1 arányban van részekre osztva, a 607 200 példából 80 százalék a tanító, 10 százalék a validációs és 10 százalék a teszt adat. A két fő szempont a teljesítmény felmérésekor a validációs és a teszt adaton elért pontosság. A tanítóadaton elért pontosság is fontos viszont a teljesítmény mérésénél nem lesz mérvadó.

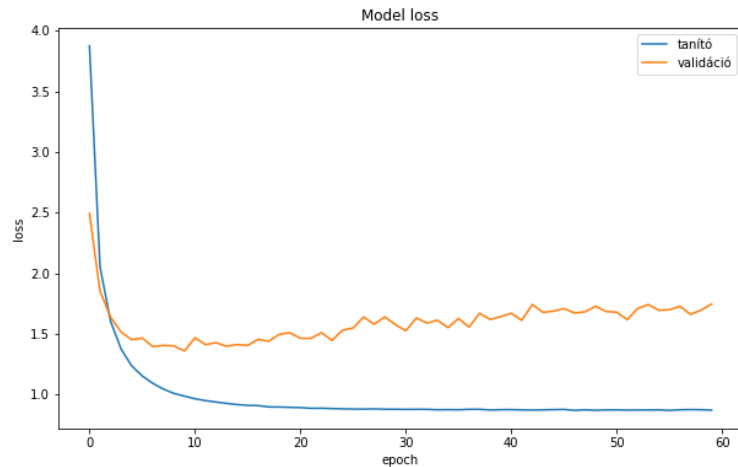
7.1.1. Hálózat teljesítménye egyetlen nagy batch-el 32x32-es példákon

A hálózat egy nagy batch-el, továbbá adat augmentáció nélkül tanult 32x32-es bináris képeken 60 epochon keresztül. Egyben kapta meg az összes tanító és validációs adatot. Megfigyelhető hogy a pontosság a validációs adaton nagyon hamar megtorpant és stagnált, míg a tanító adaton javult a pontosság. Ez egy remek példája a túltanulásnak, ahol a modell csak a tanító példákat jegyzi meg és nem általánosít rendesen.



Ábra 24 Első modell pontossága egy batch-ben

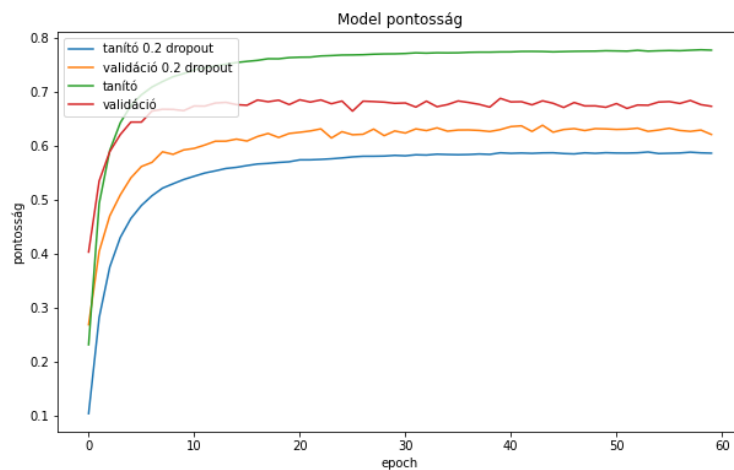
Látható a 24-es ábrán ahogy a tanítóadaton növekszik a pontosság, viszont a validációs adaton stagnál, helyenként csökken is. Ezt tovább erősíti a 25-ös ábra amin a veszteségfüggvény értéke látható. A validációs adaton a veszteségfüggvény értéke növekszik ami egyértelműen a túltanulásra utal.



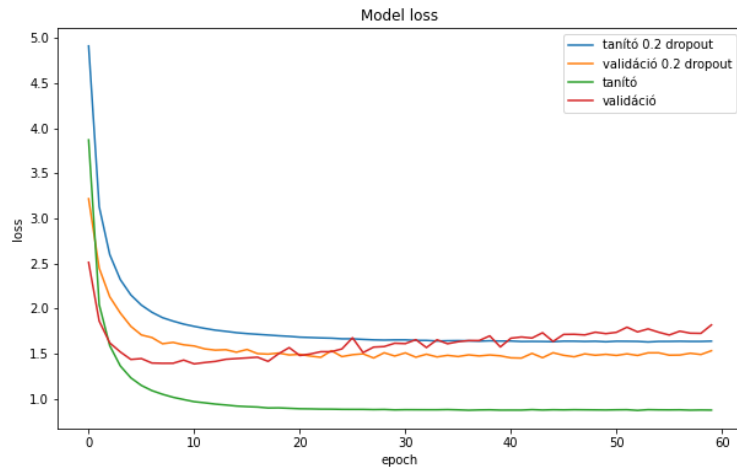
Ábra 25 Első modell vesztesége egy batch-ben

A következő lépésben regularizációval lesz szabályozva a tanulás. Ebben az esetben dropout segítségével, továbbra is csak egy batch-ből áll a tanítóadat, tehát nincs részekre bontva.

Célszerű itt is több eltérő dropout értékkel próbálkozni. A két ábra(26)(27) szemlélteti milyen hatással van a dropout esélyének növelése a tanulásra.



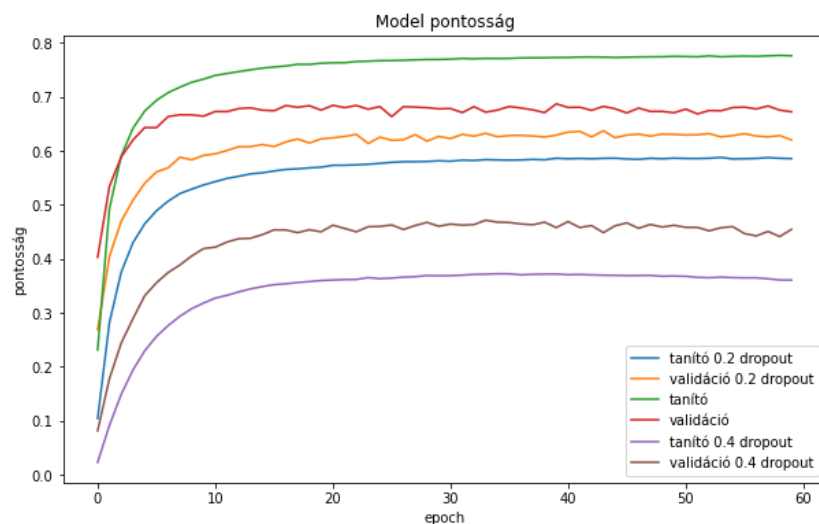
Ábra 26 Első modell pontossága 0,2 dropout-tal



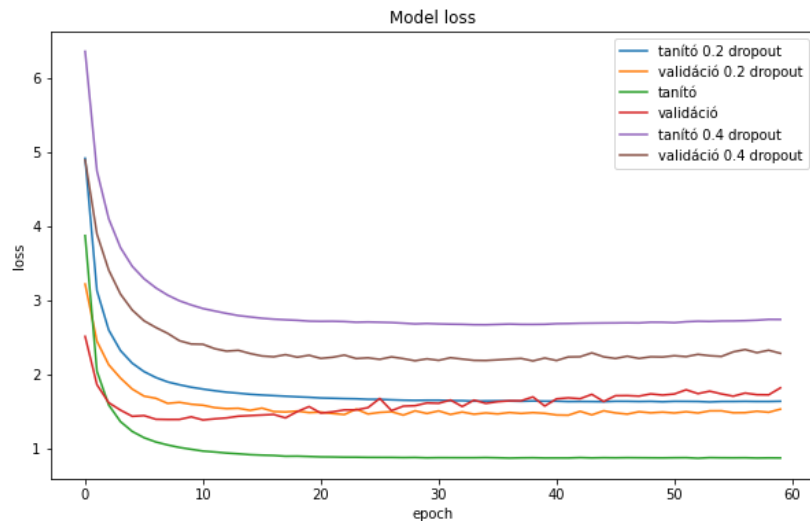
Ábra 27 Első modell vesztesége 0,2 dropout-tal

Mind a tanítóadathoz, mind a validációs adathoz tartozó pontosság csökkent, viszont a validációs adat pontossága a tanítóadatot túlszárnyalta. Ennek számos oka lehet. A modell egy viszonylag egyszerű 4 rétegű struktúra ezért lehetséges hogy 0,2 dropout túl nagy hozzá.

Az is észrevehető hogy a veszteségfüggvény ugyanúgy pozitív irányba tart még dropout használatával is. A következő ábrán látszik mi az eredménye ha a dropout 0,4-re van állítva.



Ábra 28 Első modell pontossága 0,4 dropout-tal



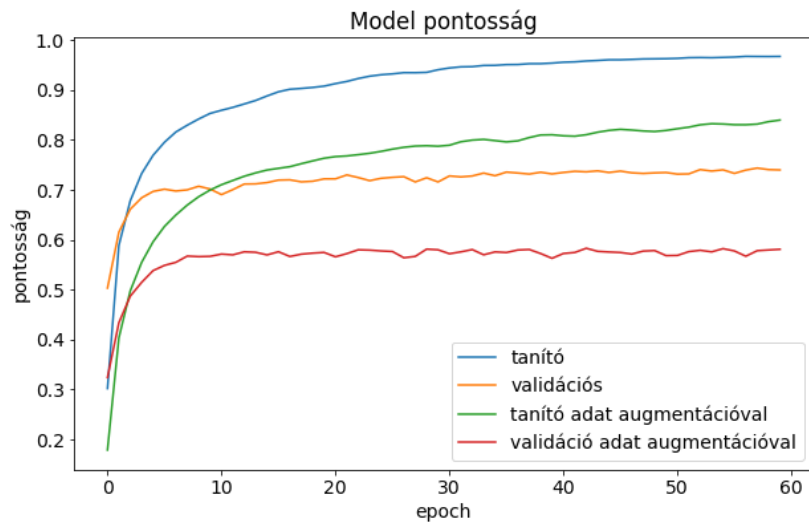
Ábra 29 Első modell vesztesége 0,4 dropout-tal

Szemlélteti a 28 és 29-es ábra hogy a dropout nem sokat segít ebben az esetben.

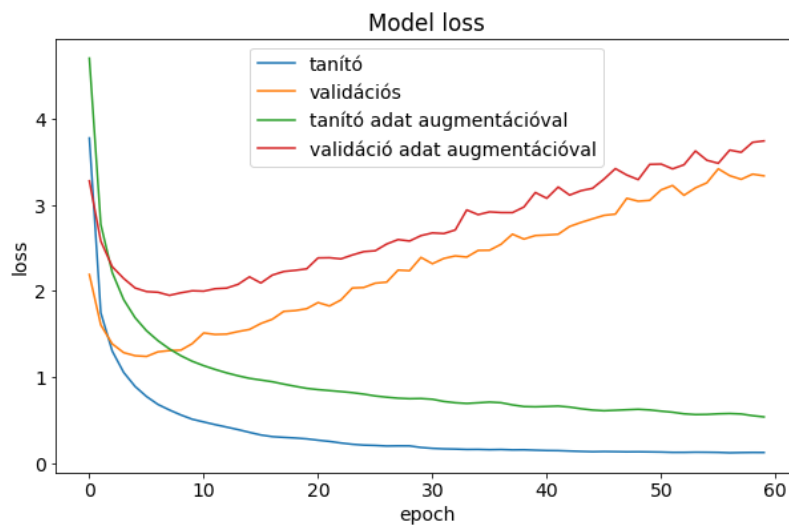
A következőkben fel lesz osztva a tanítóadat kisebb részekre, tehát a hálózat nem kapja meg az egész adatot egyben. Az elvárt eredmény az lenne hogy ez javítson a pontosságon és a túltanulást is csökkentse. Ha a batch-ek mérete elég kicsi akkor ez is regularizációs módszerként hathat a tanulás során.

7.1.2. Teljesítmény batch felosztással 32x32-es példákön

Az alábbi alfejezetben megfigyelhető hogy mi történik ha a tanítóadat fel van osztva kisebb részekre és úgy van átadva a hálónak. A 607 200 példa 800 részre van szétbontva, minden batch 759 példát tartalmaz. A tanító, validációs és teszt adat továbbra is 8:1:1 arányban van felosztva. Az első tanítás adat augmentáció nélkül futott. A batch felosztásos próba sokkal lassabb tanulást eredményezett, viszont kevesebb epoch elég volt jobb eredmény eléréséhez. A hálózat itt 32x32 felbontású képeken tanult.

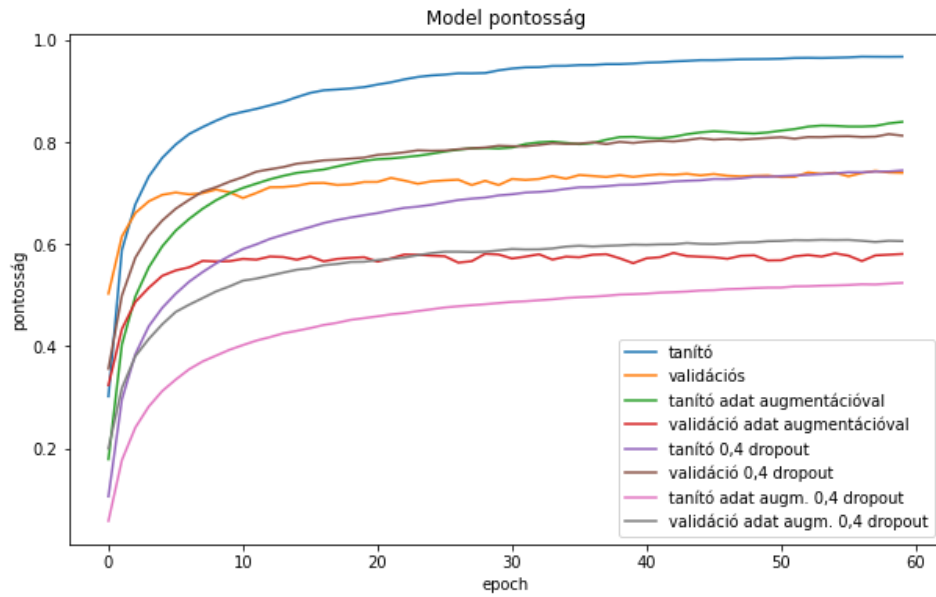


Ábra 30 Első modell batch és adat augmentációval: pontosság

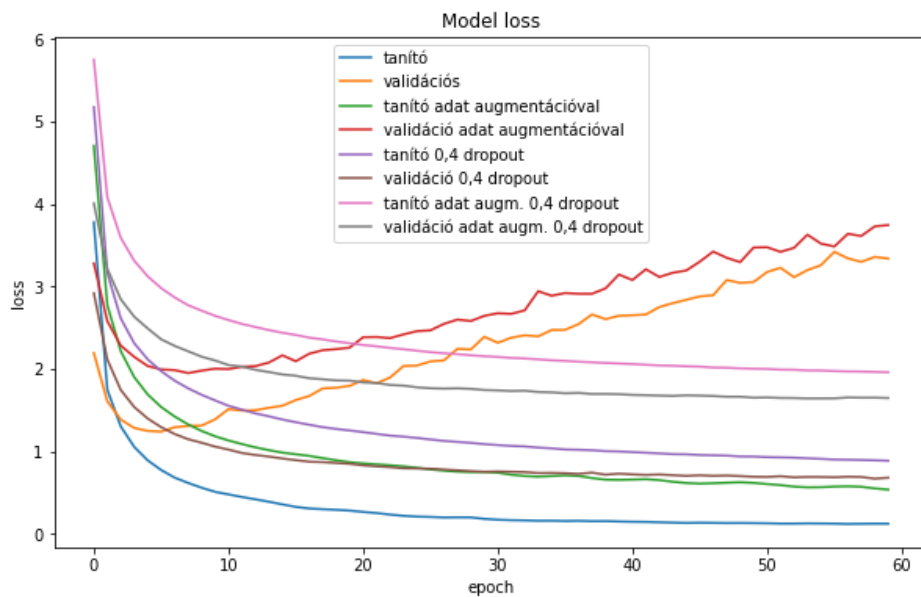


Ábra 31 Első modell batch és adat augmentációval: veszteségfüggvény

A modell 96 százalékot ért el a tanítóadaton és 74 százalékot a validációs adaton, adat augmentációval csökkent a pontosság mindkét esetben és látható a túltanulás jele a veszteségfüggvényen. A batch felosztás hozott egy kis javulást, csupán 5 százalékot de regularizációs módszerek nélkül ez sem számít rossznak.



Ábra 32 Összesített modell pontosság

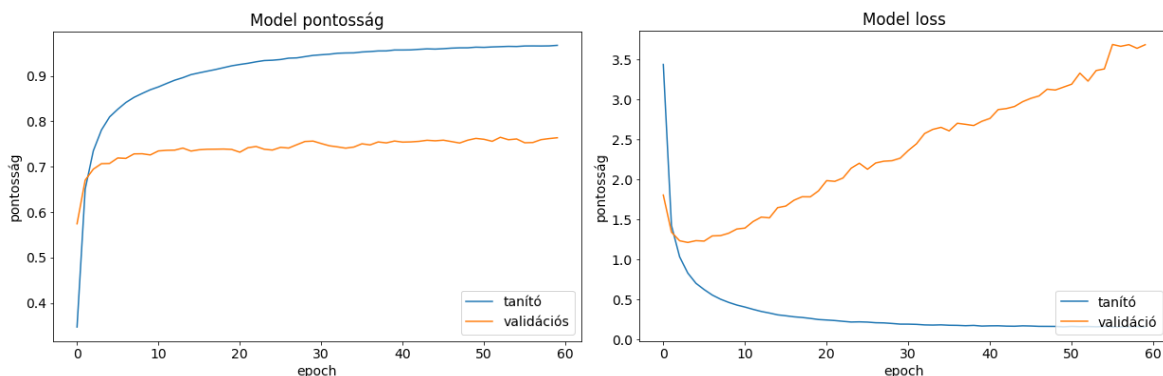


Ábra 33 Összesített modell veszteségfüggvénye

Az összesítő ábrák (32) (33) szemléltetik a különbséget különféle konfigurációk között. Ennél a modellnél a 0,4 dropout túl nagynak bizonyult ezért inkább ront a tanuláson mintsem javít rajta. Az adat augmentáció is rosszabb eredményt produkál, viszont ez betudható annak hogy egy karakternek többféle alakja lesz így, ami rövidtávon ronthatja de hosszútávon javítja mind a pontosságot mind pedig általánosítást. Összegezve ez a modell túl egyszerű, de ahhoz képest hogy négyrétegű, jól teljesített.

7.1.3. Teljesítmény 63x64-es példákon

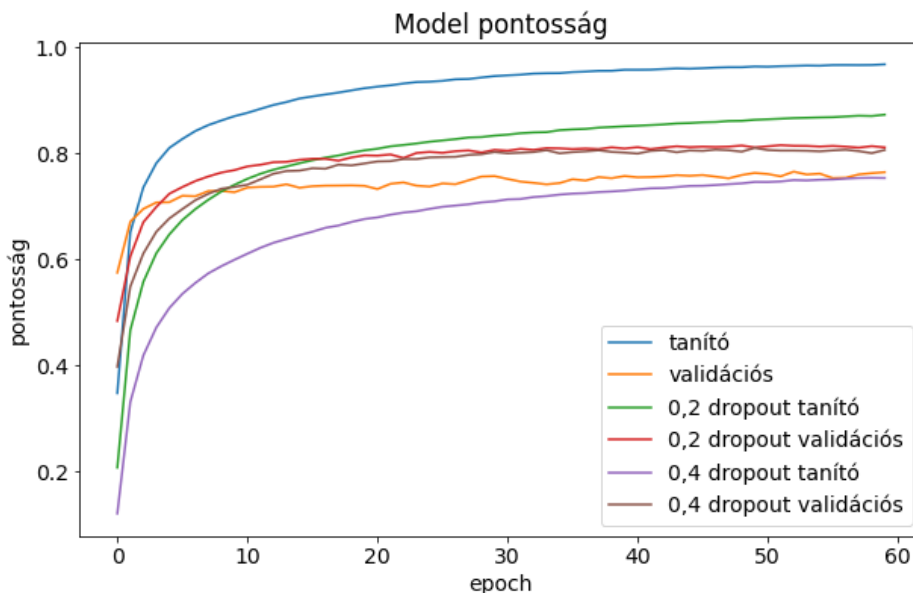
Itt az adat eredeti formájában(63x64) került a hálózathoz. Nem volt sem adat augmentáció sem regularizáció. A hálózat egyszerűsége ellenére szép eredményeket ért el. Láthatóak a tútanulás jelei de ez regularizáció nélkül nem meglepő. A pontosság növekedett a tanítóhalmazon, viszont validáción a 74 százalékon nem tudott túljutni. Továbbra is látszik a tútanulás, főleg a veszteségfüggvény grafikonján (33b). A felosztás továbbra is 8:1:1 arányban történt.



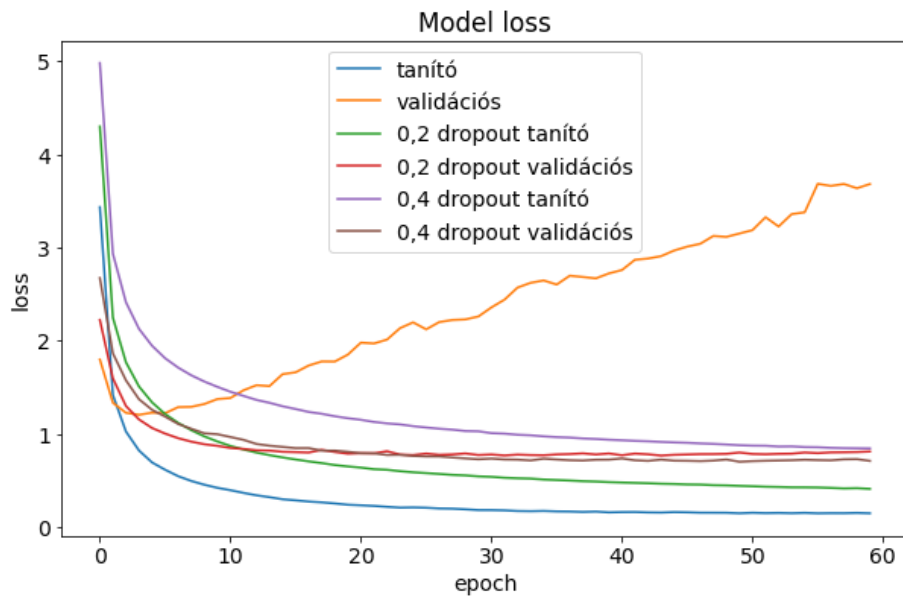
Ábra 34 (a) Első modell pontosság

(b) Első modell veszteség

A következő grafikon megmutatja a regularizáció hatását a hálón.



Ábra 35 Pontosság összehasonlítása

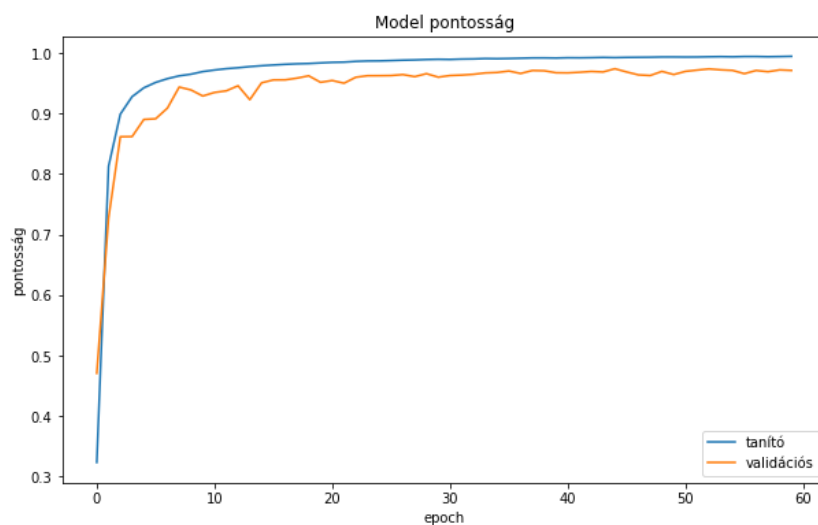


Ábra 36 Veszteségfüggvény értékének összehasonlítása

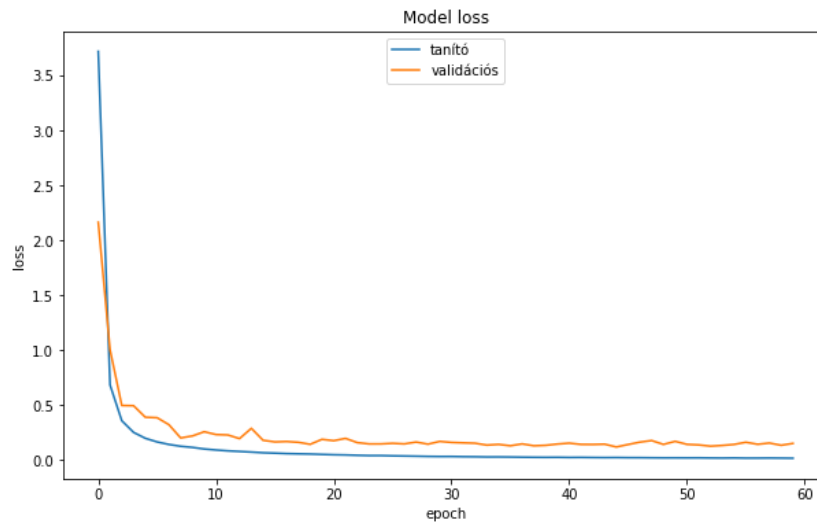
Itt már megfigyelhető hogy a regularizáció ténylegesen segített, mind 0,4, mind 0,2 dropout-tal javult az eredmény a validációs halmazon és a túltanulás is csökkent. A legjobb eredmény 81 százalék volt a validációs halmazon 0,2 dropout-tal és 80 százalék 0,4 dropout-tal. Tehát egy ponton már a dropout nem hogy nem segít, még ronthatja is a teljesítményt.

7.2. Második hálóval elért eredmények

A második hálózat sokkal jobb eredményeket produkált. Elérte 60 epoch alatt a 97 százalékos eredményt mind tanítóadaton, mind validációs adaton.



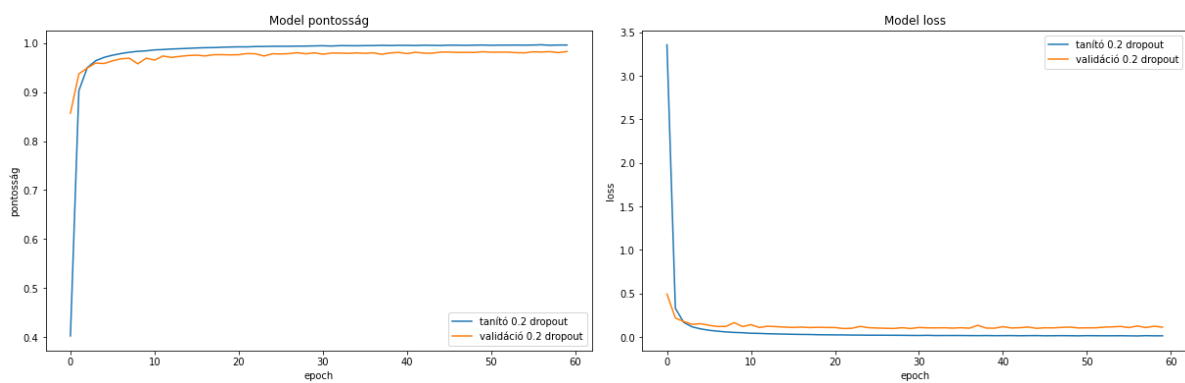
Ábra 37 CNN háló pontosság



Ábra 38 CNN háló veszteségfüggvényének értéke

7.3. Harmadik hálóval elért eredmények

A harmadik háló is jó eredményekkel zárt. A tanítóadaton 99 százalékot ért el, míg a validációs halmazon 98-at. Itt egy kisebb 0,2-es dropout értékkel tanult a háló, továbbá nem voltak köztes teljes összeköttetésű rétegek, csak a legvégén.



7.4. Hálózatok elért eredménye összesítve

A táblázat összegyűjti az eddig tárgyalt hálóknak a legjobb eredményét.

Név, Leírás	Tr. Acc.	Tr. Loss	Val. Acc.	Val. Loss	Epoch
63x64 teljesen összekötött	96,74%	0.1498	76,50%	1.2066	60
63x64 teljesen összekötött 0,2 dropout	87,24%	0.4121	81,49%	0.7693	60
63x64 teljesen összekötött 0,4 dropout	75,34%	0.8457	80,98%	0.7029	60
32x32 teljesen összekötött	96,74%	0.1176	74,36%	1.2378	60
32x32 teljesen összekötött 0,4 dropout	74,46%	0.8846	81,57%	0.6673	60
32x32 teljesen összekötött adat augmentáció	83,98%	0.5347	58,31%	1.9478	60
32x32 telj. Összekötött 0,4 dropout adat augm.	52,44%	1.9575	60,86%	1.6402	60
63x64 Konvolúciós 0,5 dropout (Második modell)	99,44%	0.01680	97,38%	0.1193	60
63x64 Konvolúciós 0,2 dropout (Harmadik modell)	99,61%	0.0132	98,30%	0.1123	60
32x32 Konvolúciós 0,2 dropout (Harmadik modell) adat augmentációval	96,40%	0.1103	94,89%	0.1986	20

A legjobb eredményt a harmadik modell produkálta, ahol 98,3 százalékot ért el a validációs halmazon. Általánosságban elmondható hogy azok a modellek ahol adat augmentáció volt jobban teljesítettek olyan példákon, amik teljesen külön adatbázisból származtak. Tehát sokat segített az általánosításon az hogy több, illetve többféle példából tanult a hálózat.

8. A felhasználói felület

A felhasználói felület alapvetően PHP és HTML használatával készült, a következő funkciókat tartalmazza:

- saját kép feltöltése
- saját kép rajzolása
- rajzolt vagy feltöltött kép felismerése
- karakterek kiválasztása

8.1. Saját kép feltöltése

A saját kép felöltése PHP-ban van megvalósítva egy HTML form-ban. Használathoz a kívánt képet ki kell választani, majd a kép feltöltése gombra kattintani. Ilyenkor az oldal megjeleníti a feltöltött képet.

```
<form action="index.php?upload=true" method="post" enctype="multipart/form-data">
    <p>Kép kiválasztása a feltöltéshez</p>
    <p><input type="file" name="fileToUpload" id="fileToUpload"></p>
    <p><input type="submit" value="Kép feltöltése" name="submit"></p>
</form>
```

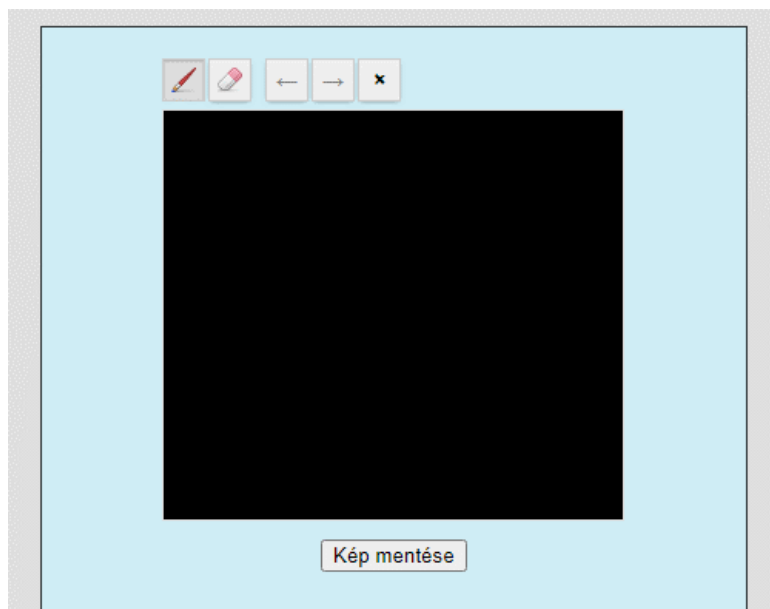
Form elküldése az *upload* változót igaz értékre állítja és frissíti az oldalt, hogy megkezdje a feltöltést. A következő kódrészlet végzi a feltöltést:

```
$target_dir = "images/";
$target_file = $target_dir . "image.png";
$uploadOk = 1;
$imageFileType = strtolower(pathinfo($target_file,PATHINFO_EXTENSION));
if (!move_uploaded_file($_FILES["fileToUpload"]["tmp_name"], $target_file))
{
    echo "Fájl feltöltése sikertelen.";
}
```

A feltöltött képet az *images* mappába mozgatja *image.png* néven. Ha valami miatt nem sikerül a feltöltés, akkor kiírja hogy a fájl feltöltése sikertelen volt.

8.2. Saját kép rajzolása

A saját kép rajzolása egy nyílt-forráskódú javascript könyvtárral lett az oldalhoz adva.



Ábra 39 Rajzfelület

Funkciói között lehet radírozni, vissza-előre lépni és az egész rajfelületet nullázni, továbbá feltölteni. Ez az egész drawingboard.js¹⁵ segítségével lett az oldalra integrálva. Az oldalhoz illesztése egy egyszerű <div> tag-gel megoldható majd a következő szkript meghívásával:

```
var myBoard = new DrawingBoard.Board('board', {
  background: "#000000",
  color: "#ffffff",
  size: 7,
  controls: [
    { DrawingMode: { filler: false } },
    'Navigation'
  ],
});
```

8.3. Rajzolt vagy feltöltött kép felismerése

Az oldal megnyit egy parancssort a háttérben, majd meghívja a python szkriptet ami betölti a betanított modellt és átadja neki a feltöltött képet.

```
<?php
$arg = "images/image.png";
$pyscript = 'predict.py';
$python = 'C:\Users\Robi\AppData\Local\Programs\Python\Python38\python.exe';
try{
    exec("$python $pyscript $arg",$output);
    ...
}
```

Az *arg* változó a python szkriptnek átadott argumentum. A *pyscript* változó a python szkriptfájl neve és a *python* változó az abszolút elérési útvonala a python.exe fájlnek. Ezeket használja bemementnek a parancssor megnyitásakor hogy lefuttassa a szkriptet majd az *output* változóba mentse a visszatérési értékeket.

A python szkript a következőképpen kapja meg a PHP parancssori argumentumát:

```
php_param = sys.argv[1]
```

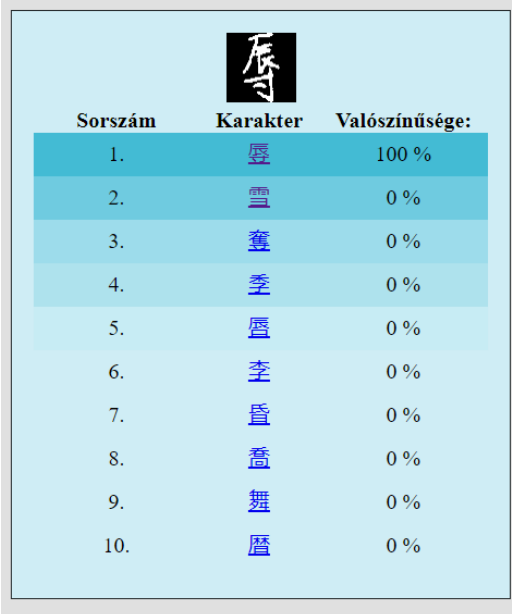
A kép útvonalát paraméterként hívja meg a predict nevű függvényt.

¹⁵ <https://github.com/Leimi/drawingboard.js>, letöltés dátuma: 2020.12.01

```
def predict(image):
    model = load_model('cnn.h5')
    img = cv.imread(image,cv.IMREAD_GRAYSCALE)
    img = np.asarray(img,dtype="uint8").reshape(1,63,64,1) / 255
    d = pd.DataFrame(np.loadtxt("dict.csv",delimiter=',', dtype="uint16"))
    predictions = model.predict(img)
    ...
```

Betölti a legjobb eredményű modellt majd beolvassa az *img* paraméterben kapott képet. Ezek után normalizálja, leosztja 255-el.

8.4. A megfelelő karakter kiválasztása



Sorszám	Karakter	Valószínűsége:
1.	辰	100 %
2.	雪	0 %
3.	奎	0 %
4.	季	0 %
5.	厶	0 %
6.	李	0 %
7.	昏	0 %
8.	喬	0 %
9.	舞	0 %
10.	厶	0 %

Ábra 40 Kiválasztható karakterek és valószínűségük

Miután a neurális háló megbecsüli a bevitt karaktert, a szkript készít egy 10 karakterből álló listát ami valószínűségek szerint sorba van rendezve, legnagyobbtól a legkisebbig.

```

...
top10 = np.argsort(predictions,axis=-1, kind='quicksort', order=None)[::-1]
codes = np.zeros(10).astype(int)
char = ["", "", "", "", "", "", "", "", "", ""]
chances = np.zeros(10)
for i in range(0,10):
    codes[i] = d.iloc[top10[0][3036 - i]][0]
    chances[i] = predictions[0][top10[0][3036 - i]]
    char[i] = format(codes[i], 'x')
    char[i] =
        bytes.fromhex( '1b2442' + str(char[i]) + '1b2842').decode('iso2022_jp')
return (char,chances)

```

A fenti kódrészlet a predict függvény folytatása. Először a *predictions* változót értékek szerint rendezi és a sorbarendezett indexeket a *top10* változóba menti. Ezután létrehoz három változót, *codes* ami a decimális hivatkozása a karakternek egy dictionary-ben, *char* amiben maga a japán karakter van, és a *chances*, ami a valószínűségi értékeket tartalmazza.

A végén visszatér a karakterekkel és a hozzájuk tartozó valószínűségi értékekkel, amit a PHP egy *output* nevű változóba ment.

A karakterek kilistázása PHP segítségével történik egy táblázatba.

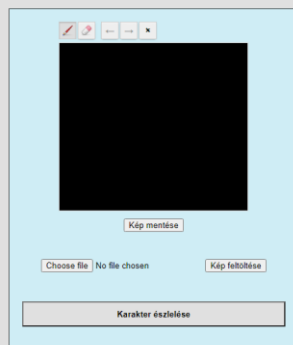
```

$k = 1;
for ($i = 18; $i >= 0; $i -= 2) {
    echo '<tr class="tall" id="grad" . $k . ">';
    echo "<td class='spacious'>" . $k . "</td>";
    echo "<td class='spacious'> <a href='https://jisho.org/search/" .
        toUTF8($output[$i]) . ">" . toUTF8($output[$i]) . "</td>";
    echo "<td class='spacious'>" . $output[$i + 1] * 100 . " % </td>";
    echo "</tr>";
    $k++;
}

```

Kilistázza a karakter sorszámát, magát a japán karaktert és azt hogy milyen valószínűséggel ítélte úgy a modell, hogy a bevitt karakter az általa megadottnak felel-e meg. Minden karakter egy hivatkozásra mutat egy kanji keresőoldalon¹⁶.

¹⁶ <https://jisho.org/> , Látogatás dátuma: 2020.12.05



Sorszám	Karakter	Valószínűsége:
1.	𠂇	32.55 %
2.	二	22.77 %
3.	厶	7.49 %
4.	𠂆	6.81 %
5.	𠂇	6.6 %
6.	𠂇	6.37 %
7.	𠂇	3.81 %
8.	𠂇	2.58 %
9.	𠂇	2.09 %
10.	𠂇	1.39 %

Ábra 41 A weboldal

Irodalomjegyzék

- [1]Dr. Tóth László: “Mesterséges neuronhálók és alkalmazásaik”, <http://www.inf.u-szeged.hu/~tothl/ann/Neuronhalok-egyben.pdf>, 49-es diaszám
- [2]Rose Holley(2009): “How Good Can It Get? Analysing and Improving OCR Accuracy in Large Scale Historic Newspaper Digitisation Programs”,
<http://www.dlib.org/dlib/march09/holley/03holley.html>
- [3]Alexander Mordvintsev & Abid K.(2013): “Geometric Transformations of Images”,
https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_geometric_transformations/py_geometric_transformations.html
- [4] <http://alexlenail.me/NN-SVG/LeNet.html>
- [5] Y. Zhang, “Deep convolutional network for handwritten Chinese character recognition,”
[Online]. Available:http://yuhao.im/files/Zhang_CNNChar.pdf.

Nyilatkozat

Alulírott Vass Róbert programtervező informatikus BSc szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet Szoftverfejlesztés Tanszékén készítettem, programtervező informatikus BSc diploma megszerzése érdekében.

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

Tudomásul veszem, hogy szakdolgozatomat / diplomamunkámat a Szegedi Tudományegyetem Informatikai Intézet könyvtárában, a helyben olvasható könyvek között helyezik el.

Dátum

2020. december 11.

Aláírás