SZÉCHENYI ISTVÁN EGYETEM

GÉPI LÁTÁS
(GKNB_INTM038)

Kézzel írt karakterek felismerése

Tömördi Tamás
(Z8GD9X)

Mérnök informatikus BSc

Győr, 2022

TARTALOMJEGYZÉK

1. BEVEZETÉS

1.1. A megoldandó feladat kifejtése

A beadandó téma javaslatok közül a kézzel írt karakterek felismerését választottam. Bár napjainkban szövegeinket főként digitális formában tároljuk és szerkesztjük, a kézírás továbbra is kulcsfontosságú szerepet játszik életünkben. A kézírás gépi felismerése fontos lehet például a meglévő, kézzel írt dokumentumaink (akár történelmi korok szövegeinek) digitalizálása során. Sőt, a kézírásra akár művészetként is tekinthetünk, hiszen találkozhatunk szebbnél szebb kalligrafikus írásokkal is.

A valóságban azonban a kézírás felismerése nem annyira egyszerű, hiszen a sok írott szöveg még emberi szemmel is nehezen értelmezhető – gondoljunk csak a korábbi, papír alapú orvosi recepteken látható kézírásokra. Tovább bonyolítja a helyzetet az is, ha tekintetbe vesszük a különböző nyelvek eltérő írásrendszereit. Az orosz cirill kézírás például rögtön sokkal nehezebben értelmezhető, mint a mi latin betűs írásaink, a keleti (arab, kínai, japán) nyelvekről nem is beszélve, hiszen a kínai nyelv esetében például a karakterek nagy száma nehezítheti a felismerést.

Feladatom tehát nem teljes szövegek, hanem egyes kézzel írt karakterek – számok és az angol ábécé betűinek – felismerése volt, homogén háttér előtt, amely nagyban csökkenti a fentebb említett probléma komplexitását.

1.2. Az általam választott megközelítési mód

A beadandóm elkészítésének kezdeti szakaszában rögtön arra a megállapításra jutottam, hogy a napjainkban használt neurális hálózatoknak köszönhetően a karakterek egyesével történő felismerése egyeltalán nem is olyan nehéz feladat. Ebből kifolyólag túlságosan is rövid és egyszerű lett volna egy modell megvalósítása és bemutatása, ezért úgy döntöttem, hogy pár CNN (*Convolutional Neural Network*) modell megvalósítása, bemutatása és értékelése egy sokkal szemléletesebb beadandót eredményezne.

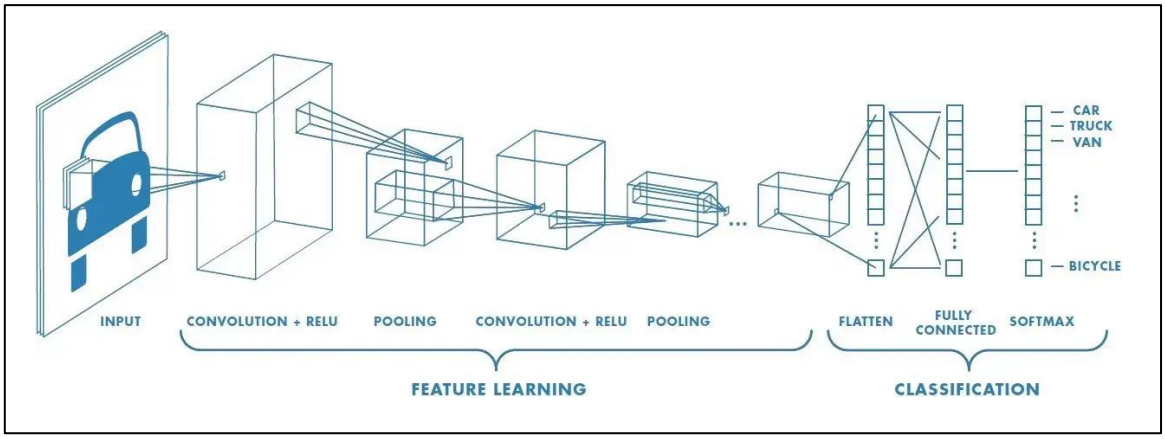
A konvolúciós neurális hálózatok elméletének rövid áttekintése után öt különböző általam megismert háló felépítését mutatom be, amelyek közül egy széles körben elterjedt és ismert. Utána bemutatok egyes részleteket, szemelvényeket az általam írt kódból, majd kiértékelem az egyes modellek teljesítményét az EMNIST teszt adatait használva a modellek validálására (validation data), három különböző személy általam rögzített kézzel írt karaktereit pedig a modell „éles” tesztelésére. A beadandó végén a felhasználói dokumentációban pedig ismertetem a GitHub repository-ban megtalálható fájlok használatát.

2. A KONVOLÚCIÓS NEURÁLIS HÁLÓK ELMÉLETE

A konvolúciós neurális hálózatokat az különbözteti meg a többi neurális hálózattól, hogy kiváló teljesítményt nyújtanak a kép-, beszéd- vagy hangjel bemenetekkel. Három fő rétegtípusuk van, amelyek a következők:

- Konvolúciós réteg
- Összevonó (Pooling) réteg
- Teljesen összekapcsolt (FC) réteg

A konvolúciós réteg a konvolúciós hálózat első rétege. Míg a konvolúciós rétegeket további konvolúciós rétegek vagy pooling rétegek követhetik, a teljesen összekapcsolt réteg az utolsó réteg. Minden egyes réteggel a CNN összetettsége növekszik, és a kép nagyobb részeit azonosítja. A korábbi rétegek az egyszerű tulajdonságokra, például a színekre és az élekre összpontosítanak. Ahogy a képadatok haladnak előre a CNN rétegeiben, a CNN elkezdi felismerni a tárgy nagyobb elemeit vagy formáit, míg végül azonosítja a kívánt tárgyat [6].



1. ábra. A CNN hálózatok általános felépítése [1]

2.1. Konvolúciós réteg

A konvolúciós réteg a CNN központi építőköve, és itt történik a számítások nagy része. Néhány összetevőre van szükség, amelyek a bemeneti adatok, egy szűrő és egy jellemzőtérkép. Tegyük fel, hogy a bemenet egy színes kép lesz, amely egy 3D-s pixelmátrixból áll. Ez azt jelenti, hogy a bemenetnek három dimenziója lesz - magasság, szélesség és mélység -, amelyek megfelelnek a kép RGB-értékének. Van egy jellemződetektorunk is, más néven kernel vagy szűrő, amely végigmegy a kép receptív mezőin, és ellenőrzi, hogy a jellemző jelen van-e a képen. Ezt a folyamatot konvolúciónak nevezzük.

A jellemződetektor egy kétdimenziós (2D) súlyokból álló tömb, amely a kép egy részét reprezentálja. Bár ezek mérete változhat, a szűrő mérete jellemzően egy 3x3-as mátrix; ez határozza meg a receptív mező méretét is. A szűrőt ezután a kép egy területére alkalmazzuk, és a bemeneti képpontok és a szűrő között pontszorzatot számolunk. Ez a pontproduktum ezután egy kimeneti mátrixba kerül. Ezután a szűrő egy lépéssel eltolódik, és a folyamat addig ismétlődik, amíg a kernel át nem söpörte a teljes képet. A bemenet és a szűrő pontproduktumainak sorozatából származó végső kimenetet feature map, aktivációs térkép vagy konvolvált feature néven ismerjük [6].

2.2. Összevonó (Pooling) réteg

A rétegek összevonása, más néven downsampling, dimenziócsökkentést végez, csökkentve a bemeneti paraméterek számát. A konvolúciós réteghez hasonlóan a pooling művelet is egy szűrőt söpör végig a teljes bemeneten, de a különbség az, hogy ennek a szűrőnek nincsenek súlyai. Ehelyett a kernel egy aggregációs függvényt alkalmaz a receptív mezőn belüli értékekre, feltöltve a kimeneti tömböt. A poolingnak két fő típusa van:

a) MaxPooling: Ahogy a szűrő áthalad a bemeneten, kiválasztja a maximális értékkel rendelkező pixelt, amelyet a kimeneti tömbbe küld.
b) AveragePooling: Ahogy a szűrő áthalad a bemeneten, kiszámítja a befogadó mezőn belüli átlagos értéket, amelyet a kimeneti tömbre küld.

Három olyan hiperparaméter van, amelyek befolyásolják a kimenet térfogatának méretét, és amelyeket a neurális hálózat tanításának megkezdése előtt kell beállítani:

1. A szűrők száma befolyásolja a kimenet mélységét. Például három különböző szűrő három különböző jellemzőtérképet eredményezne, ami három mélységet hozna létre.
2. A stride az a távolság, vagyis a pixelek száma, amelyet a kernel a bemeneti mátrixon áthalad. Míg a kettő vagy annál nagyobb stride értékek ritkák, a nagyobb stride kisebb kimenetet eredményez.
3. A nulla kitöltést általában akkor használják, ha a szűrők nem illeszkednek a bemeneti képhez. Ez a bemeneti mátrixon kívül eső összes elemet nullára állítja, ami nagyobb vagy azonos méretű kimenetet eredményez.

2.3. Teljesen kapcsolt (Fully Connected) réteg

A részlegesen kapcsolt rétegekben a bemeneti kép pixelértékei nem kapcsolódnak közvetlenül a kimeneti réteghez. A teljesen összekapcsolt rétegben azonban a kimeneti réteg minden egyes csomópontja közvetlenül kapcsolódik az előző réteg egy csomópontjához. Ez a réteg az osztályozás feladatát az előző rétegeken és azok különböző szűrőin keresztül kinyert jellemzők alapján végzi. Míg a konvolúciós és a pooling rétegek általában ReLu függvényeket, az FC rétegek általában softmax aktiválási függvényt használnak a bemenetek megfelelő osztályozásához, amely 0 és 1 közötti valószínűséget eredményez [6].

3. FEJLESZTÉSI DOKUMENTÁCIÓ

3.1. A választott modellek ismertetése

3.1.1. Az első modell

Az első modellben háromszor váltakozik egymás után egy konvolúciós és egy Pooling réteg. Mindhárom konvolúciós réteg 64 filtert, 3x3-as kernelméretet és ReLU aktivációs-függvényt használ. A Pooling rétegek pedig 2x2-esek és MaxPooling-ot végeznek. Ezt követően egy 64-es Fully Connected réteg következik szintén ReLU aktivációs függvénnyel, legvégül pedig Softmax segítségével jön létre az a 36 osztály (10 számjegy és 26 angol betű), amelyeket fel szeretnénk ismerni a képeken.

3.1.2. A második modell

A második modell az elsőhöz képest kevesebb konvolúciós és Pooling réteggel dolgozik, kevesebb filtert használ, azonban ugyanúgy 3x3-as kernelekkel és szintén ReLU aktivációs függvénnyel. Egy 32 filteres konvolúciós réteget egy 2x2-es MaxPooling követ, majd ez ismétlődik újra még egyszer. A Fully Connected réteg 128 unittal dolgozik, az utolsó rétegben pedig a SoftMax segítségével létrejön a 36 osztály.

3.1.3. A harmadik modell

A harmadik modell az első kettőtől jobban különbözik. Itt három különböző konvolúciós réteget egy MaxPooling követ, ami kissé szokatlanbb felépítés, hiszen a legtöbb neurális hálóban igyekszünk a konvolúciókat és a pooling-okat felváltva használni.

Az első konvolúciós réteg 32 filterrel, 3x3-as kernelmérettel és ReLU aktivációs függvénnyel dolgozik. A második réteg 64 filterrel, de szintén 3x3-as kernellel, míg a harmadik konvolúció már 128 filtert és 4x4-es kernelméretet használ. Ezt követi egy 2x2-es MaxPooling, majd egy Dropout, amely a neuronok 25%-át véletlenszerűen eldobja.

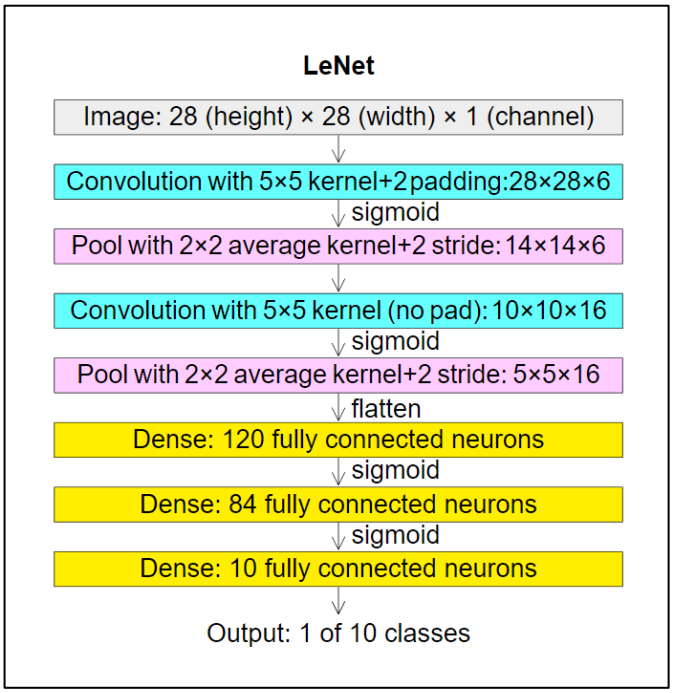
A Fully Connected réteg egy 128-as réteg, és ReLU-t használ. Ezt követi egy újabb 0.25-ös Dropout, végül pedig a Softmax segítségével a 36 osztály.

3.1.4. A negyedik modell (a harmadik módosítása)

A negyedik modell a harmadik egy apróbb módosításával jött létre. Valójában a kettő közötti különbség csak annyi, hogy egy plusz Fully Connected réteg került beépítésre, illetve ezt követi egy újabb 0.25-ös Dropout. Szimplán kíváncsi voltam arra, hogy hoz-e érdemi változást ez a módosítás a hálózat hatékonyságát illetően. (Amint az a következő fejezetben, a kiértékelésben kiderül – nem).

3.1.5. Az ötödik modell, a LeNet5

Yann LeCun, Leon Bottou, Yosuha Bengio és Patrick Haffner az 1990-es években javasoltak egy neurális hálózati architektúrát kézírásos és gépi karakterfelismerésre, amelyet LeNet-5-nek neveztek el. Az eredeti publikációban számos más verziója is szerepel, ilyenek a Le-Net-1, LeNet-4 vagy a Boosted LeNet-4. A LeNet-5 architektúra egyszerű és könnyen érthető, ezért gyakran használják a konvolúciós neurális hálózatok oktatására is. A modell 98,48%-os pontosságok képes elérni abban az esetben, ha az MNIST adatkészlettel kerül tanításra, és az MNIST tesztkészletén történik a tesztelése [5]. Természetesen ez a 10 számjegyre használva igaz, mivel én ezt a modellt kissé átalakítva 36 osztályra használtam a következő fejezetben majd láthatjuk, hogy így már egyeltalán nem ér el olyan jó eredményt.



2. ábra. Az eredeti LeNet felépítése [2]

A modell 28x28-as méretű képek feldolgozására alkalmazható, amely számomra az EMNIST képeinek felbontásából kifolyólag pont megfelelő. A két konvolúciós réteg 5x5-ös kernelt akalmaz, első alkalommal 2-es padding-al, a második alkalommal pedig padding nélkül. Mindkét konvolúciós réteget követi egy 2x2-es Average Pooling, amely a modell készítésének idejében még általánosan használt volt. Bár manapság a legtöbb esetben Max-Poolingot alkalmazunk, én az eredeti LeNet architektúrához kívántam alkalmazkodni. Az első teljesen összekapcsolt réteg tekinthető konvolúciós rétegnek is, abban az esetben, ha nagyobb a kép felbontása, jelen esetben azonban ez a réteg már teljesen összekapcsolt rétegnek számít, amely 120 neuront használt, a következő réteg 84, majd az utolsó, az eredeti LeNet-ben 10-et, hiszen ez felel meg a 10 számjegy osztálynak. Mivel én a 26 angol betűt is szerettem volna felismerni, ezért ezt az utolsó réteget 36 neuronra módosítottam.

3.2. Felhasznált Python csomagok

A projekt elkészítéséhez a következő python csomagokat használtam, amelyek a fájlok futtatásához is szükségesek:

- Numpy
- OpenCV
- Matplotlib
- Tensorflow, Keras
- EMNIST

Az OpenCV-t képfájlok beimportálásához és különböző képfeldolgozási műveletekhez használtam, mint például a képek átméretezése vagy szürkeárnyalatossá konvertálása. A Numpy az importált képek tárolásában és egyszerűbb kezelhetőségében volt segítségemre. A Matplotlib-et a képek és grafikonok megjelenítésére használtam a fejlesztés során. A statisztikákról készült grafikonok megtalálhatóak képformátumban a projekt `stats/` mappájában. A Tensorflow és a Keras a modellek elkészítéséhez volt szükséges, amelyeket az EMNIST dataset segítségével tanítottam be.

3.3. A tervezés és fejlesztés folyamata

Első lépésként a karakterfelismerés (OCR – Optical Character Regcognition) elméletének olvastam utána, és megnéztem, hogy milyen módszerekkel lehet és érdemes karaktereket felismerni. Mivel a leghatékonyabban konvolúciós hálózatokkal lehet a problémát megoldani, ezért én is ezeknek a megismerésével kezdtem. Áttekintettem a CNN-ek általános működését, amelyről korábban érintőlegesen a Mesterséges intelligencia tantárgy keretében már tanultam.

Ezután néhány tutorial videó segítségével megtanultam a Tensorflow és a Keras használatát, azt, hogy pontosan milyen módon lehet felépíteni Pythonban konvolúciós hálózatokat. Ezt rövid kísérletezgetés követte, amely során különböző hálózatokat próbáltam felépíteni az interneten fellelhető más CNN-ek mintájára.

Az első konzultáció alkalmával bemutattam három, akkor még kezdetlegesen megvalósított modellt, amelyek a 26 angol betűre működtek eléggé alacsony pontossággal. A modelleket később átalakítottam, hogy a 10 számjeggyel együtt már együtt 36 osztály felismerésére legyenek alkalmasak. Ettől fogva tehát az MNIST helyett az EMNIST adatait használtam.

A második konzultáción már négy modellem volt, amelyeket azonban – mint megtudtam – nagyon kevés epoch számmal tanítottam csak, így rendkívül rossz Accuracy értékeket produkáltak. A Google Colab szolgáltatását használva azonban GPU segítségével sokkal gyorsabban taníthattam a modelleket, így mindegyik modellen 50 epoch-ot futtattam, és a Pyplot segítségével grafikonon is megjeleníttettem a Loss és Accuracy értékek alakulását. Valamint egy ötödik modellt is kipróbáltam, a LeNet5-öt, amelynek felépítése a 2. ábrán látható.

3.4. Kódrészletek

A modellek tanításához először is az EMNIST képein kellett átalakításokat végezni. A képek normalizálást egy nagyon egyszerű módon végeztem el, a Numpy segítségével a tömböket egyszerűen elosztottam a maximum intenzitásértékkel, azaz 255-el. A következő lépésben a tömböket át kellett alakítani, hogy megfeleljenek a Keras által várt shape-nek, ami (N, 28, 28, 1) egy 28x28-as képek esetében, ahol az N dimenzió a tanításhoz rendelkezésre álló képek számát jelöli. Az idetartozó kódrészletek a 3. ábrán láthatók:

```
# Normalization:
x_train = x_train / 255
x_val = x_val / 255
# Reshape:
IMG_SIZE = 28
x_train = x_train.reshape(-1, IMG_SIZE, IMG_SIZE, 1)
x_val = x_val.reshape(-1, IMG_SIZE, IMG_SIZE, 1)
```

3. ábra. Az EMNIST adatok normalizálása és alakítása

A 4. ábrán látható kódrészlet mutatja a harmadik modell felépítését. Mint látható, a modellhez rendkívül átláthatóan és könnyen adhatók hozzá a különböző rétegek, és látható az is, hogy milyen módszerrel adhatók meg a filter- és kernelméretek, valamint az aktivációs függvények. Csupán a kód segítségével is már ránézésre jól átlátható egy modell felépítése.

Az 5. ábra a recognition.py fájlban található kód egy részletét mutatja, amellyel egy tetszőleges képfájlon lévő karaktert lehet felismerni a mentett modellek egyikével. Az OpenCV segítségével először megtörténik a fájl beolvasása, majd pedig az átalakítás: a kép átalakítása 28x28-as méretre lineáris interpoláció segítségével, a bitwise_not függvény pedig az EMNIST adatkészletében található képekhez alakítja az általam rajzolt képet. Ezután a már fentebb említett Numpy műveletek következnek, végül pedig a prediction futtatása a lementett és beimportált modellel.

```
model.add(Conv2D(filters = 32, kernel_size = (3, 3), activation = 'relu',
input_shape = x_train.shape[1:]))
model.add(Conv2D(filters = 64, kernel_size = (3, 3), activation = 'relu'))
model.add(Conv2D(filters = 128, kernel_size = (4, 4), activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2, 2)))
model.add(Dropout(.25))
model.add(Flatten())
model.add(Dense(128, activation = 'relu'))
model.add(Dropout(.25))
model.add(Dense(36, activation = 'softmax'))

model.compile(loss = "sparse_categorical_crossentropy", optimizer = "adam",
metrics = ["accuracy"])
model.fit(x_train, y_train, epochs = 50, validation_data = (x_val, y_val))
model.save("saved_models/model_3/")
```

4. ábra. A harmadik modell megvalósítása

```
model = keras.models.load_model('./saved_models/model_lenet5')
print("Type the name of the file: ")
file_name = input()

image = cv2.imread('./letters/' + file_name, cv2.IMREAD_GRAYSCALE)
image = cv2.resize(image, (28, 28), interpolation = cv2.INTER_LINEAR)
image = cv2.bitwise_not(image)
img = np.array(image)
img = img / 255
img = img.reshape(-1, 28, 28, 1)

predictions = model.predict(img)
```

5. ábra. Mentett modell használata egy képfájl felismerésére

4. A MODELLEK ÖSSZEHASONLÍTÁSA

4.1. A mérőszámokról

4.1.1. Accuracy

A Keras-ban megtalálható különböző Accuracy mérőszámok (Accuracy, Binary Accuracy, Categorical Accuracy, Sparse Categorical Accuracy, Top K Categorical Accuracy, Sparse Top K Categorical Accuracy) közül a legegyszerűbb, sima Accuracy segítségével hasonlítottam össze a modellek teljesítményét, amely a következő formulával írható le:

$$Accuracy = \frac{NCP}{TNP}$$

$$NCP - Number\ of\ Correct\ Predictions$$
$$TNP - Total\ Number\ of\ Predictions$$

Tehát az Accuracy értéket egyszerűen úgy kapjuk meg, hogy a helyes felismerések számát (NCP) leosztjuk a felismerni kívánt képek mennyiségével (TNP). Mint majd láthatjuk, a tanítás során a validálási adatokon, tehát az EMNIST teszt adatain nagyon hasonló Accuracy értékeket hoznak a különböző modellek, azonban az általam írt betűkön már nagyobb különbségek láthatók. A beadandó végén ennek lehetséges miértjeire is megpróbálok rávilágítani, és kitérek arra is, hogy mit lehetne másképp csinálni ahhoz, hogy jobb Accuracy értékeket hozzanak a modellek.

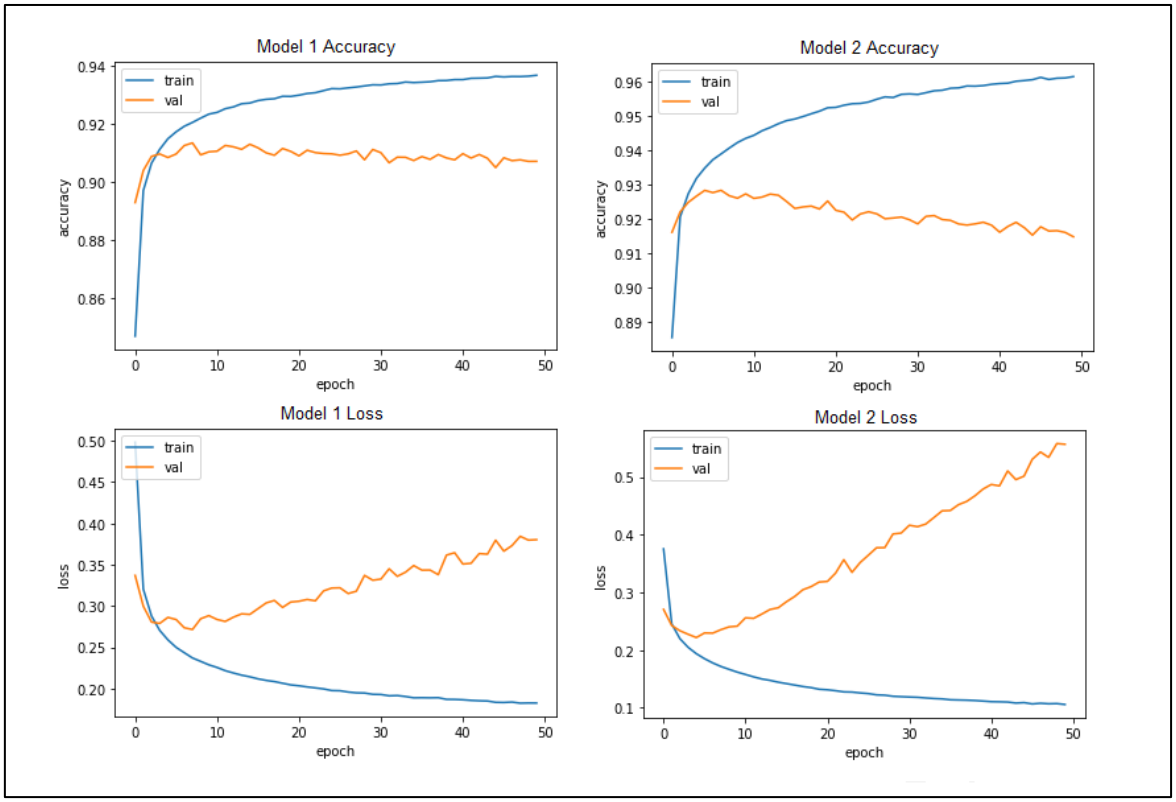
4.1.2. Loss

A Keras különböző Loss értékei közül vannak, amelyek bináris osztályozás esetén használhatók (Binary Crossentropy), illetve vannak, amelyek több osztállyal dolgozó felismerések esetén (Categorical Crossentropy, Sparse Categorical Crossentropy, Poison Loss, Kullback-Leibler Divergence Loss). Abban az esetben – ami a miénk is –, amikor kettő vagy több osztállyal dolgozunk és a label-ek integer típusúak a Sparse Categorical Crossentropy-t érdemes használni, így én is ezt teszem.

4.2. A modellek betanítása és validálása az EMNIST-el

A modellek tanítását az EMNIST adatkészlet segítségével végeztem, amely angol kézírásos karaktereket tartalmaz. A EMNIST tanításra szánt adatait használtam a tanításhoz, míg a teszt adatokat a validáláshoz. Minden modell tanítását követően a Pyplot segítségével megjelenítettem, hogy a lefuttatott 50 epoch alatt, hogy változott a Train Accuracy, a Train Loss, illetve a Validation Accuracy és a Validation Loss. A következő oldalakon ezek a statisztikák láthatók.

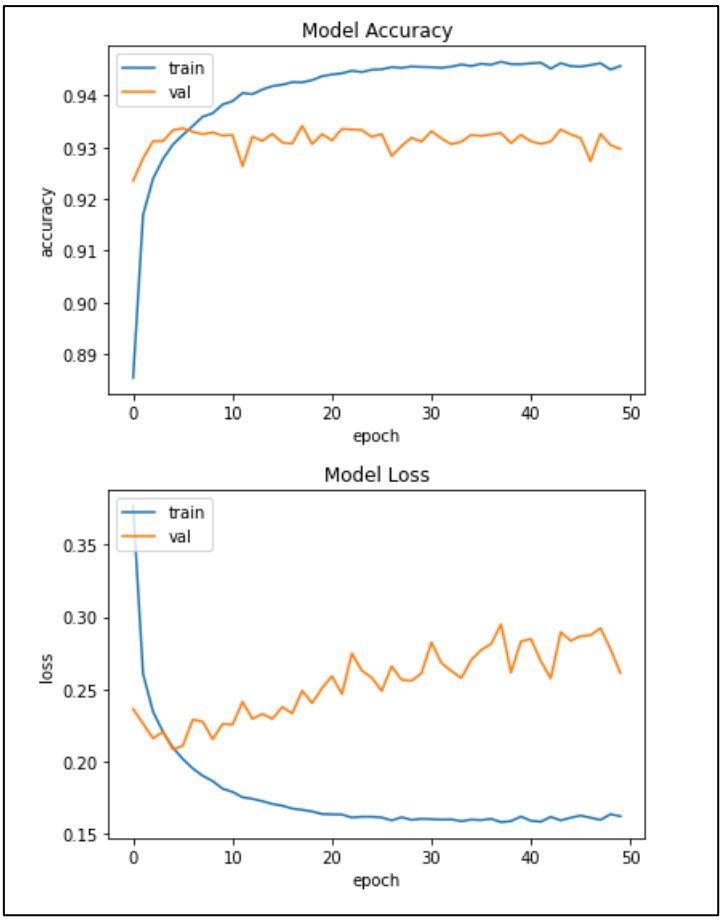
Az első modell esetében azt láthatjuk (6. ábra), hogy a Validation Accuracy már 8 epoch környékén eléri a maximumát 0,91-es érték körül. Az epoch-ok növekedésével először ingadozni kezd, majd szép lassan csökkenésnek indul. A Validation Loss esetében a modell szintén a 8. epoch körül elére a legalacsonyabb 0.28 körüli értéket, majd innen meredeken emelkedni kezd.



6. ábra. Az első és második modell statisztikái

A második modell (szintén 6. ábra) a kezdeti epoch-okban nagyon hasonló eredményeket produkál, annyi különbséggel, hogy az epoch szám növekedésével a modell pontossága egyre romlik. A Validation Accuracy az 5. epoch körül eléri maximumát, csak kicsivel marad 0.93 alatt, majd csökkenésbe kezd. A Loss esetében azonban az 5-ös epoch után egy nagyon meredek emelkedés kezdődik, tehát mondhatjuk, hogy a Validation Data esetében ez a modell úgy tűnik az elsőnél rosszabbul teljesít. Hogy ez a tesztadatokra is igaz lesz-e, a következő alfejezetben kiderül.

A harmadik modell (7. ábra) már jobban teljesít, szintén az ötödik epoch körül eléri maximumát az Accuracy és minimumát a Loss. Az Accuracy itt már 0.93 fölé is emelkedik és a Loss is alacsonyabb picit a korábbi modellekhez képest. Az epoch szám növekedésével azonban mindkét érték eléggé gyorsan romlani kezd.

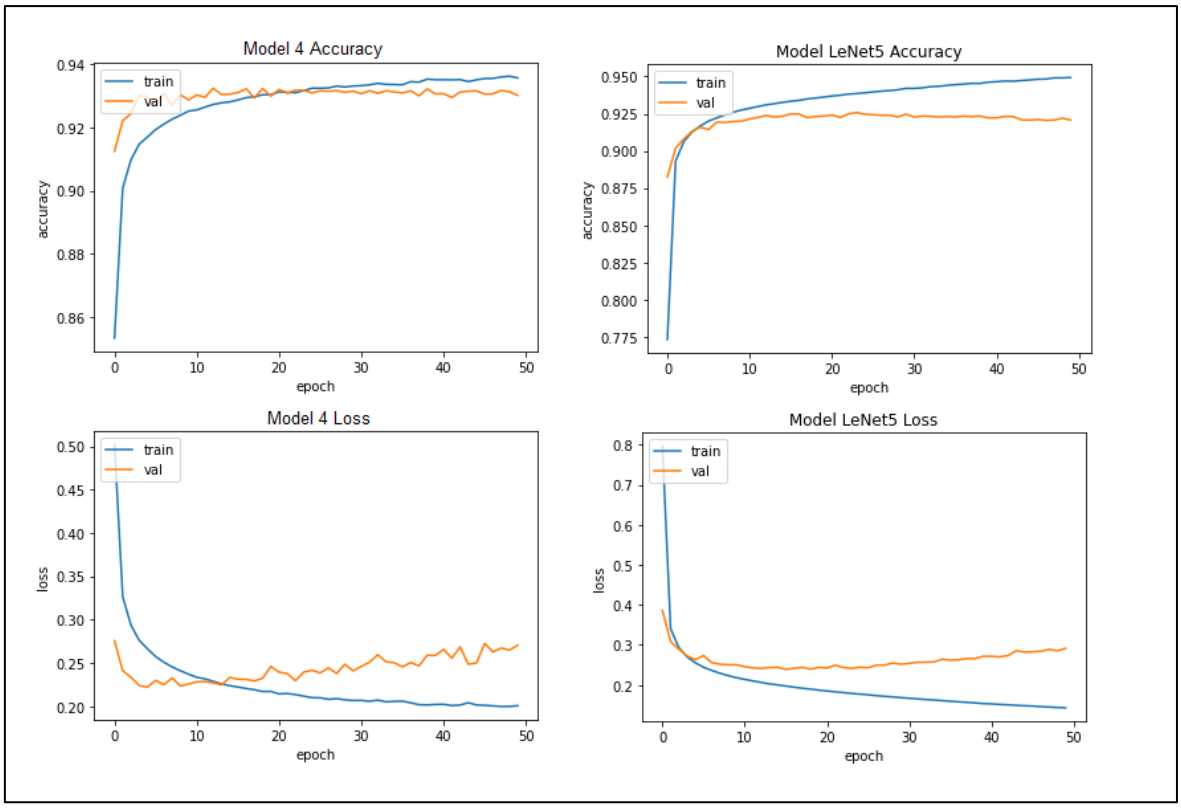


7. ábra. A harmadik modell statisztikái

A negyedik modell (8. ábra) esetében a Validation Accuracy szintén 0.93 fölé emelkedik, az epoch szám növekedésével azonban nem kezd el olyan gyorsan csökkeni, elég hosszú ideig stagnál a 0,92-es érték felett. A Loss azonban itt is emelkedésnek indul a kezdeti pár epoch után.

Az ötödik modell, a LeNet-5 (8. ábra) hozza a legjobbnak tűnő eredményt a Validation Data esetében. A Validation Accuracy a 0.93-as értéket elérve hosszan és biztosan tartja ezt a szintent, míga Loss is relatívan alacsony marad az epoch szám növekedésével, bár ennek a lementett modell hatékonyságára nézve nincs nagyobb jelentősége. Az azonban megmutatkozik, hogy a LeNet-5 esetében egy népszerűbb, biztosabb modellről beszélünk. Az általam rajzolt karakterek esetében, vagyis a tesztelésnél azonban meglepő módon mégsem ez hozza a legjobb eredményt.

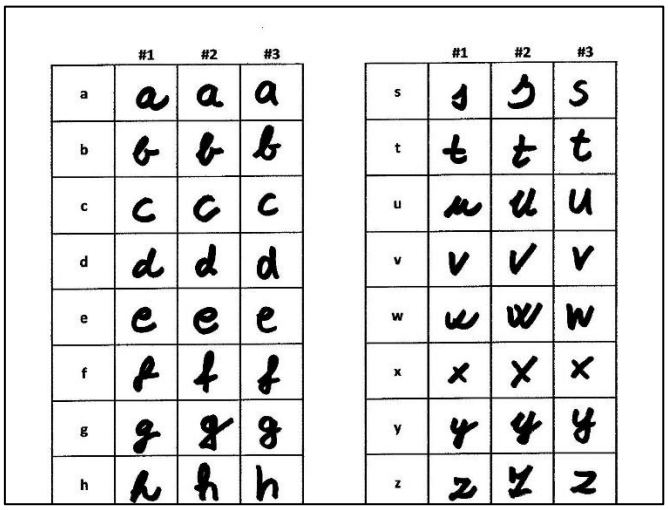
Összességében tehát elmondhatjuk, hogy a tanítás és validálás tekintetében (tehát az EMNIST adatain!) a modellek nagyon hasonlóan teljesítenek. Az Accuracy tekintetében pedig egészen magas értékeket érnek el. Az igazán mérvadónak természetesen a következő részfejezetben látható eredmények tekinthetők, hiszen itt magyar írással írt, saját karakteren futtatom majd a modelleket, azaz ezt nevezhetjük a valódi tesztelésnek.



8. ábra. A negyedik és ötödik (LeNet-5) modell statisztikái

4.3. A modellek tesztelése saját képfájlokon

A modellek teszteléséhez saját karaktereket rajzoltam, illetve megkértem három mások személyt arra, hogy kézzel is írjanak karaktereket, papíron. Ezeket utána beszkenneltem és az egyes karaktereket külön fájlokba mentettem le. Összesen tehát 280 képen teszteltem a modelleket, ezekből 80 ábrázol számjegyet, 100 kisbetűket, 100 pedig nagybetűket. A 9. ábrán látható pár minta a karakterekből:



9. ábra. Pár kézzel írt karakter a tesztadatokból

Az Az eredményeket az alábbi táblázatban összesítettem. Mint látható, ezeken a teszt adatokon a modellek már nem értek el olyan jó eredményeket. Piros színnel kiemeltem a legrosszabb Accuracy és Loss értékeket, zölddel pedig a legjobbakat. Összességében tehát elmondhatjuk, hogy a saját adataimon a legjobbnak a harmadik modell bizonyult, amely 0.82-es Accuracy értéket ért el:

Model	Accuracy	Loss
Model_1	0.56	5.33
Model_2	0.71	5.81
Model_3	0.82	1.53
Model_4	0.77	1.71
Model_LeNet5	0.75	1.05

A tesztelés során megfigyeltem azt is, hogy a modellek sokkal könnyebben ismerik fel a számjegyeket és a nagybetűket, a problémát leginkább a kisbetűk jelentik, amelyeknek írása sokszor tömörebb, a ’g’, ’k’, ’f’ betűknél például a kunkorok nem mindig kivehetőek, amely tovább nehezíti ezen betűk felismerését. Szintén problémát jelent a modelleknek azon karakterek megkülönböztetése, amelyek sokszor emberi szemmel és nehezen megítélhetőek, ilyenek például a ’0’, az ’o’ és a ’O’ közötti eltérések, valamint például az angol és magyar írásban eltérően ábrázolt ’1’-es és ’7’-es is könnyen összetéveszthető, hiszen az angolok gyakran nem húzzák át a hetes szárát középen, míg az egyesről lehagyják a második, visszatérő kampós vonalat.

Természetesen nagyon sok olyan betű is van, amelyek osztályozása rendkívül egyszerű feladat, ezeken a karaktereken sokkal hatékonyabban működnek a modellek, ilyen karakterek leginkább például a nagybetűk, hiszen ezek sokkal szellőssebek és könnyebbel felismerhetők rajtuk bizonyos jellegzetességek, amelyeket a modell konvolúciós rétegei hamarabb megtalálnak.

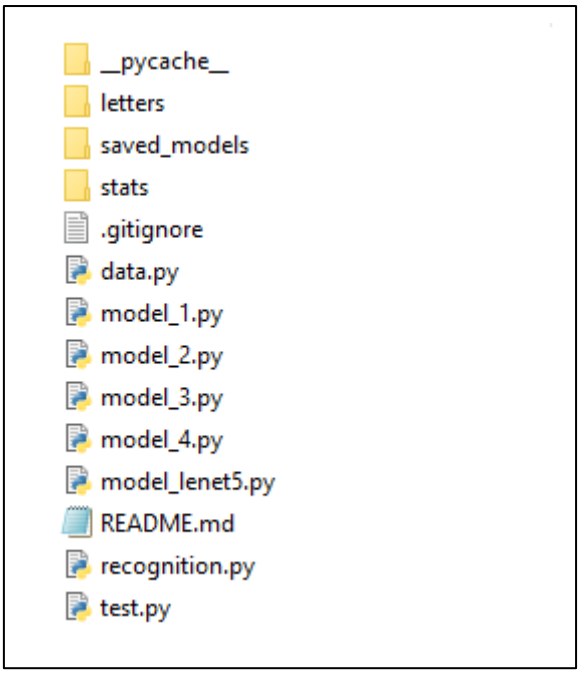
Érdemes röviden kitérnünk arra, hogy miért szerepelhetnek rosszabbul a modellek ezeken a tesztadatokon, mint az EMNIST tesztadatain. Erre véleményem szerint a következő magyarázatok lehetségesek:

a) a magyar kézírás kisebb-nagyobb mértékben eltér az angoltól,
b) a modellek túlságosan jól megtanulják az EMNIST adatok különböző jellegzetességeit, és ettől valamilyen mértékben eltérő adatokon már rosszabbul szerepelnek,
c) a saját képeim nem eléggé kerületek átalakításra ahhoz, hogy az EMNIST-hez hasonlóan ábrázolják az adott karaktereket, így valójában további képátalakításokra lenne szükség.

5. FELHASZNÁLÓI DOKUMENTÁCIÓ

A projekt „kipróbálásához” lényegében a két legfontosabb fájl a `recognition.py` és a `test.py`. Az előbbi segítségével egy megadott modellt futtathatunk le egy megadott betűre. A modell nevét a modell importálásának sorában (8. kódsor) kell megadni, az elmentett modellek pedig a `saved_models/` mappában találhatók, innen kerülnek beimportálásra. Ezt követően a python fájl futtatása után megadhatjuk a képfájl nevét (kiterjesztéssel együtt), amelyre szeretnénk kipróbálni, hogy milyen eredményt hoz az adott modell. A fájlt a kód a `letters/` mappában keresi, onnan importálja be. Ezt követően a console kimeneten láthatjuk, hogy a modell milyen betűt lát a képen, illetve maga a kép is megjelenítésre kerül a pyplot segítségével.

A `test.py` segítségével futtathatjuk le a mentett modelleket az összes képfájlon, ami a `letters/` mappában található. Itt történik a modell kiértékelése, a futtatás után a consoleon megjelennek a számított Loss és Accuarcy értékek. Azt, hogy melyik mentett modellre szeretnénk futtatni a tesztelést, itt is a 8. kódsorban adhatjuk meg.



7. ábra. A projekt fájlstruktúrája

A data.py segítségével került importálásra az EMNIST dataset, amely segítségével a `model_1.py`, `model_2.py`, `model_3.py`, `model_4.py` és a `model_lenet5.py` fájlokban a modelleket megvalósítottam és betanítottam. A `stats/` mappa képformátumban tartalmaz pár olyan statisztikát, amelyeket a tanítás és a tesztelés során készítettem, és amelyeket ezen dokumentációhoz felhasználtam.

6. ÖSSZEGZÉS

Összességében elmondhatom, hogy a tárgy beadandójának keretében sikerült megismerkednem azokkal az alapokkal, amelyek egy neurális háló elméletét adják és egy modell gyakorlati megvalósításához szükségesek. Az általam megírt modellek ugyan nem túl magam Accuracy értékekkel működnek, az alapok elsajátításához mégis nagyon hasznosak voltak. További feladat lehet a modellek még széleskörűbb tesztelés különböző betűkön, amely a modellek tanításakor elsősorban időt emészt fel, hiszen a tanítás a Google Colab szolgáltatásainak segítségével könnyebb ugyan, de hasonlóan időigényes, illetve a Colab által is korlátozott az ingyenesen rendelkezésre álló memória és GPU használat.

Érdemesebb lehet a modellek betanítása saját karakterek segítségével, törekedve arra, hogy a magyar karakterírásnak megfelelően tanítsunk, ehhez azonban nagyon sok kézzel írt karakter szükséges, amelynek előállítása szintén elsősorban időt vesz igénybe, és sokkal inkább repetitív feladat, mintsem technikai jellegű kérdés.

A beadandómhoz készült prezentációban igyekeztem majd kiemelni a fontosabb részleteket, és még több képet mutatok majd a különböző modellekről, illetve a tesztadatokról, így vizuálisan is jobban szemléltetve a modellek működését, erősségeit és hiányosságait. Bemutatom a modellek vázlatát, a statisztikai adatok, és röviden kitérek majd a fejlesztés folyamatára is.

7. IRODALOMJEGYZÉK

[1] Saha, S. (2018, December 15). *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way.* Medium.
https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[2] *Comparison of the LeNet and AlexNet*. Wikipedia.
https://en.wikipedia.org/wiki/LeNet#/media/File:Comparison_image_neural_networks.svg

[3] Benyó, B., *Konvolúciós hálózatok.*
https://semmelweis.hu/kepalkotas/files/Convolut_network_Benyo.pdf

[4] *Konvolúciós neurális hálózatok (CNN)*.
http://home.mit.bme.hu/~engedy/NN/NN-CNN.pdf

[5] Rizwan, M. (2018, October 16). *LeNet-5-A Classic CNN Architecture*. Data Science Central.
https://www.datasciencecentral.com/lenet-5-a-classic-cnn-architecture/

[6] *How do convolutional neural networks work?*, IBM.
https://www.ibm.com/cloud/learn/convolutional-neural-networks

[7] *Accuracy metrics*. Keras.
https://keras.io/api/metrics/accuracy_metrics/

[8] *Keras Loss Functions: Everything You Need to Know*. MLOps Blog.
https://neptune.ai/blog/keras-loss-functions