Adatelemzés Mélytanulási Módszerekkel  
Karakterfelismerési feladat

E1IK75 - Tarsoly Levente

TFB4FY - Czotter Benedek

T83K5I - Hermann Máté Béla

Bevezetés

Az adatelemzés és gépi tanulás módszertanai az elmúlt időben kiemelt szerepet kaptak az adatvezérelt döntéshozatal és az automatizáció fejlődésével. A félév során megismert módszerek egy jó alapot biztosítanak az adatelemzés problémáinak orvoslására.

A karakterfelismerési feladat jelentősége

A karakterfelismerés a gépi tanulás egyik klasszikus problémája, amely az optikai karakterfelismerésen keresztül számos gyakorlati alkalmazást kínál, például a dokumentumok digitalizálása, automatikus szövegfeldolgozás vagy járműrendszám-felismerés terén. A modern mélytanulási módszerek, különösen a konvolúciós neurális hálózatok (CNN), igen hasznosnak bizonyultak ezen a téren.

Ez a feladat nemcsak az adatelemzés és a gépi tanulás elméleti ismereteinek gyakorlati alkalmazását teszi lehetővé, hanem hozzájárul az adatelemzési készségek fejlesztéséhez is. A karakterfelismerés során alkalmazott modellek megkívánják a félév során tanult módszerek, például az előfeldolgozás, az osztályozási algoritmusok és a mélytanulási modellek használatát.

Összegzésül, a félév során szerzett tudás jó alapot nyújt a karakterfelismerési feladat megoldásához, amely egy fontos lépés az intelligens rendszerek fejlesztése felé. Ez a projekt egy jó lehetőség arra, hogy amit elméletet tanultunk, átültetve a gyakorlatba egy gyakorlati példán keresztül alkalmazhassuk.

Feladat és az adathalmaz ismertetése

A feladat bemutatása

A projekt célja egy mélytanulási osztályozó modell megvalósítása, amely képes osztálycímkék előrejelzésére a tanuló adathalmaz alapján, majd ezek helyességének mérése a teszt adathalmazon. A megoldás során különösen fontos, hogy az algoritmus manuális beavatkozás nélkül végezze el az osztályozást, így a projekt teljes mértékben automatizált legyen. Az emberi szem általi ellenőrzés kerülése a tiszta algoritmikus megközelítés elősegítését, valamint az eredmények objektivitását és reprodukálhatóságát biztosítja.

A projekt keretében végzendő feladatok:

1. **Az adatok megfelelő módon történő beolvasása:**Mivel az adathalmaz nem egy előre elkészített, netről elérhető és importálható adathalmaz, ezért az első fontos lépés az adatok/fájlok felépítésének megismerése, majd beolvasása.
2. **Mélytanulási modell fejlesztése és betanítása:**A modellnek képesnek kell lennie az egyes karakterek osztályozására a tanuló adathalmaz alapján. Ehhez a mély neurális hálózatok (például CNN) használata ajánlott, mivel ezek nagyon jó eredményeket érnek el képfeldolgozási feladatokban. Az osztályozó teljesítménye a tanuló adatokon történő betanítás során nyert jósági mutatók alapján mérhető.
3. **Paraméteroptimalizálás:**A modell paramétereinek (például a tanulási ráta, rétegek száma, aktivációs függvények) optimalizálása kulcsfontosságú a jó teljesítmény elérése érdekében.
4. **Eredmények rögzítése és dokumentálása:**A projekt során elért eredményeket (pontosság, konvergencia, jósági mutatók) rögzíteni kell. Emellett részletesen le kell írni a használt módszereket, modelleket és a feladat megoldása során tett megfontolásokat.

Adathalmaz bemutatása

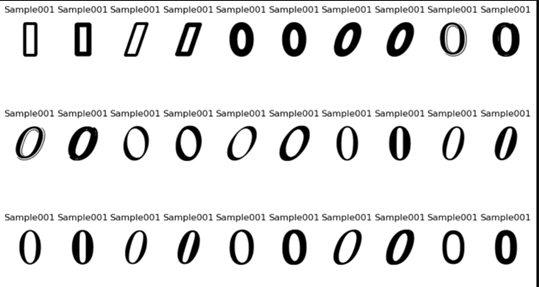
Az adathalmaz három fő részből áll: **Train1**, **Train2** és **Test** mappák. Ezeknek a mappáknak a szerkezete és tartalma kulcsfontosságú a projekt szempontjából:

1. **Train mappák (Train1 és Train2):**Ezek tartalmazzák a tanításhoz szükséges képeket. A képek karakterenként rendezettek a Sample mappákban, így minden mappa egy adott osztályhoz tartozó példákat tartalmaz.
   * **Adatok sokszínűsége:**A mappákban lévő képek egységes méretűek, azonban ez nem jelenti azt, hogy automatikusan az a jó méret, ugyanis lehetséges, hogy szükségeltetik átméretezés a számítási problémák elkerülése végett. Emellett más orientációjúak (pl. egy kicsir döntött) vagy eltérő stílusúak lehetnek, ami kihívást jelent a modell számára. Az előfeldolgozás (például normalizálás, méretezés) különösen fontos a modellezési folyamatban.
   * **Címkeinformációk:**Az adathalmaz mappaszerkezete implicit módon tartalmazza az osztálycímkéket, mivel minden mappa egy-egy címkének felel meg. Ezért a fájlok betöltése és a címkék hozzárendelése során figyelembe kell venni a mappák hierarchiáját.
2. **Test mappa:**Ez a mappa tartalmazza azokat a képeket, amelyekre a modell előrejelzéseket fog végezni. A címkék itt nincsenek explicit módon megadva, és ezeket a képeket a modell természetesen nem ismeri. Ez garantálja a modell független validációját és az automatizált osztályozás tesztelését.
   * **Kihívások:**A tesztadatok eltérhetnek a tanító adatoktól, például különböző zajszintek, stílusok, vagy egyéb vizuális torzítások miatt. Ezért a modell általánosító képessége kulcsfontosságú.

Megvalósítás lépései

*Adatok feldolgozása*

A feladat elvégzésének első lépésében az adatok megfelelő előfeldolgozására volt szükség. Ehhez beolvastuk a Train1 és Train2 mappákban található adatokat almappánként, így könnyen el tudtuk tárolni az egyes almappákban lévő képekhez a címkéket, amely jelen esetben az almappák neve lett. A képeket szürkeárnyalatos módban töltöttük be, és megfelelő átméreteztük megfelelő méretre. Ezen lépésekhez az OpenCV könyvtár imread() és resize() metódusait hívtuk segítségül. Az adatokat annak érdekében, hogy könnyebben el tudjuk képzelni, hogy hogyan is néznek ki, illetve, hogy teszteljük a beolvasás sikerességét ki is rajzoltuk.



A beolvasott képeket és a címkéket két külön tömbbe (image\_data és labels) mentettük. Majd a képek pixelértékeit egy egyszerű 255-tel való osztással 0-1 tartományba hoztuk a későbbi könnyebb feldolgozás érdekében.

Ezután a címkéket kellett olyan formára hozni, hogy az gépi tanulási feladatokban is használható legyen. Ehhez string műveletekkel kinyertük a címkék nevének a végén szereplő számokat. Majd LabelEncoder segítségével transzformáltuk őket. Végül pedig a to\_categorical() függvénnyel előállítottuk a címkék one-hot encoded formáját. Utóbbira azért volt szükség, hogy a későbbiekben kategórikus keresztentrópiát tudjunk használni a modellek veszteség függvényeként.

Tanító -és teszthalmaz meghatározása

Következő lépésben fontos volt az adathalmazt tanító -és teszthalmazra bontani, hogy nyomon tudjuk követni az egyes modellek hogyan teljesítenek. Fontos volt szem előtt tartani, hogy minden osztályból megfelelő mennyiségben legyen adat mind a tanító mind a teszt halmazban is. Ehhez az sklearn beépített keretrendszerét a StratifiedShuffleSplit-et használtuk, amellyel pontosan azt lehetett elérni, hogy az egyes osztályok példányai ugyanolyan arányban szerepeljenek a tanító -és teszt adatok között. Kezdetben 80-20 százalékban volt a tanító -és teszthalmaz aránya, majd ezt később 90-10-re módosítottuk, hogy több tanító adatunk legyen, és pontosabb eredményeket érjünk el.

*Baseline modell*

Annak érdekében, hogy lássuk hol tartunk létrehoztunk egy baseline modellt. Ehhez az ötletet a hatodik gyakorlaton megismert modellből merítettük. A modell több rétegből állt: konvolúciós rétegek (32 és 64 szűrővel), ReLU aktivációk, max pooling a rétegek dimenziójának csökkentéséhez, dropout rétegek az overfitting elkerülésére, valamint egy teljesen összekapcsolt réteg a klasszifikációhoz. A végén egy softmax aktivációs réteg található, amely a több osztályos kimenetet generálta. A modellt 28x28-as képeken tanítottuk. Érezhető volt, hogy van mit javítanunk, és ennél jóval nagyobb pontosság is elérhető lesz, akár a képek méretének növelésével, akár új vagy módosított konvolúciós rétegek hozzáadásával.

*Képek augmentációja*

Ezt követően az ImageDataGenerator osztály segítségével a képek augmentálását végeztük el. A beállítások között szerepelt a képek vízszintes és függőleges eltolása, nyírása, valamint a transzformációk során keletkező üres területek kitöltése a szomszédos pixel értékek alapján. A vízszintes tükrözést itt nem engedélyeztük, mivel a képeken szereplő karakterek irányultsága fontos információt hordozhat. Kezdetben alkalmaztunk enyhe középpont körüli forgatást is a képeken. Azonban miután jobban megvizsgáltuk a tanító adatban található képeket úgy gondoltuk, hogy ezt nem alkalmazzunk, mivel sok esetben már kellően elforgatottnak vagy nyújtottnak találtuk a képeket. Az augmentáció célja a modell általánosíthatóságának javítása volt változatosabb bemeneti adatok biztosításával.

*GPU használat tanításhoz*

Annak érdekében, hogy robosztusabb modellekkel és nagyobb képmérettel tudjunk dolgozni szükséges volt lehetővé tenni a GPU használatát a tanítási folyamat felgyorsításához, így már a végleges modellünket 128x128-as kép méretre készítettük.

Több platformra is megvalósítottunk GPU használatot. Először mac-en, ahol a tensorflow-macos és tensorflow-metal csomagok letöltésével automatikusan GPU-t használva tudtuk a modelleket tanítani. Azonban ez nem hozta meg a várt gyorsulást nagyobb képméretekre, így windows-on is létre kellett hozni a megfelelő környezetet, ahol nagyobb GPU és memória kapacitás volt elérhető számunkra. Windows rendszeren két gépünk volt, amin képesek voltunk GPU acceleration segítségével futtatni a számításainkat. Az egyik egy 32 GB memóriával rendelkező és egy 1660 Ti 6 GB VRAM-mal működő Desktop gép volt. Ezen végeztük végső számításainkat, mert bár a másik gép videókártyájának jobb számítási kapacitása volt (3050), viszont csak 4 GB volt a VRAM és 16 GB a belső memória. Ezekhez a számításokhoz az NVIDIA CUDA API cuDNN könyvtárát használtuk TensorFlow-ba integrálva. Ennek telepítése és a megfelelő verziók használata kifejezetten nehézkes volt, végül Windows rendszeren nem is volt megoldható (mivel Windowson a TensorFlow 2.10es verzióig támogatja a GPU supportot, amivel gyakorlatilag mindent downgradelni kellett volna). Ehelyett a Windows rendszerbe már integrált WSL (Windows Subsystem for Linux) szolgáltatást használtuk, így nem volt szükség külön virtuális gép telepítésére, vagy dual-bootra. A Linux (Ubuntu) telepítése és megnyitása után telepítettük a megfelelő csomagokat (Python, pip, venv, tensorflow, cuda, cudnn, …) és indítottunk egy jupyter szervert localhoston, így böngészőből tudtuk futtatni és módosítani a jupyter notebook-unkat. Mikor a képméretet növeltük 128x128 pixelesre, észrevettük, hogy a virtuális környezet nem használja ki a rendszer memóriáját és emiatt megakad már az adathalmaz beolvasásánál. Ennek növelése és swap memória hozzáadása után gördülékenyen képesek voltunk tanítani.

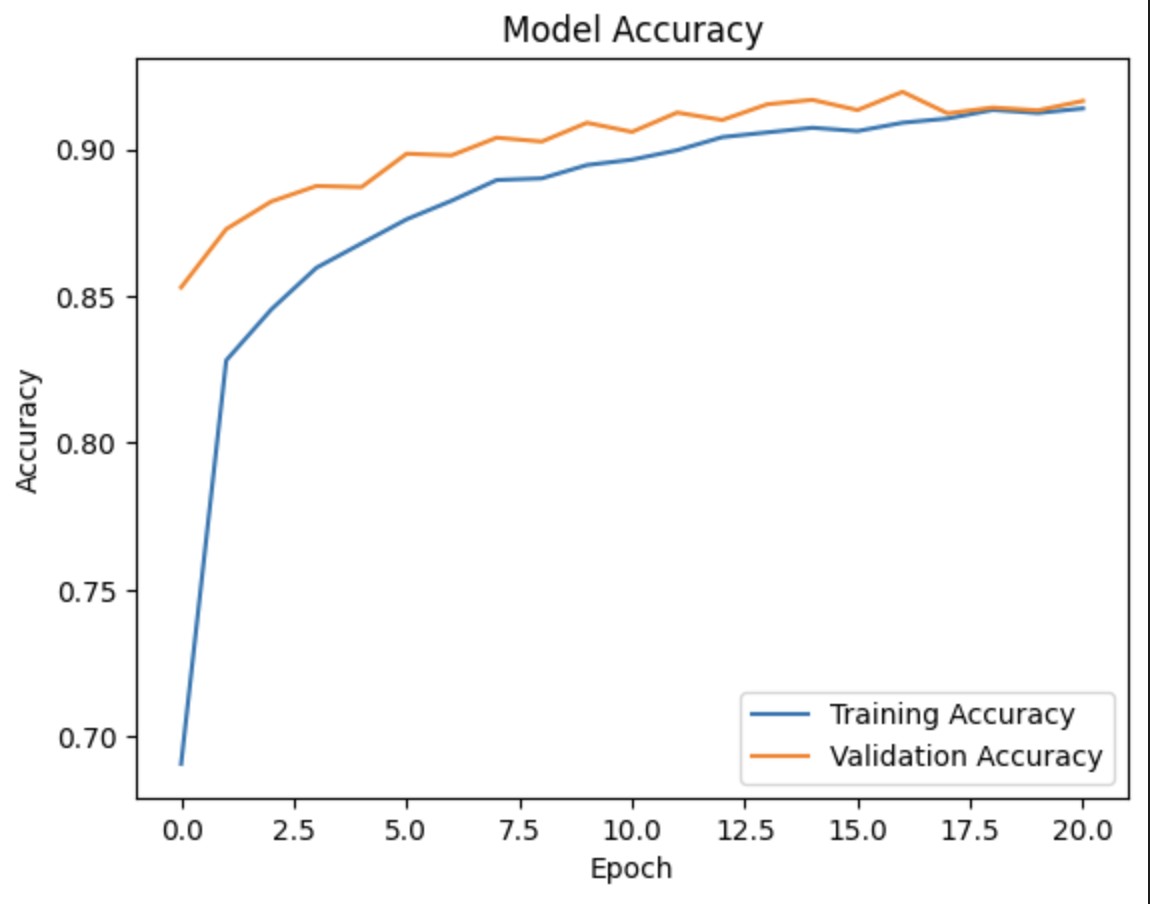
*Végleges modell*

Modell felépítése

A végleges modellünk egy általunk készített 4 konvolúciós réteget tartalmazó mély neurális háló volt. Az első és második konvolúciós rétegben 5x5-ös, a harmadikban 4x4-es, a negyedikben pedig 3x3-as ablakmérettel mintavételeztünk. Mindegyik konvolúciós réteget ReLu aktivációs függvény, kötegelt normalizálás majd egy 2x2-es MaxPooling réteg követte. A szűrők száma a konvolúciós rétegekben 32, 64, 128 az utolsóban pedig 256 lett. Ezután következett a Flatten réteg, amit egy 512 neuronból álló fully connected réteg követett szintén ReLu aktivációs függvénnyel. A dense réteg után 50 százalékos dropout-ot helyeztünk. Végül pedig a többosztályos osztályozás miatt softmax aktivációs függvényt alkalmaztunk. Végezetül több mint 3.5 millió tanítható paraméterrel rendelkezett a modellünk.

Regularizáció

A túltanulás ellen többféle regularizációs módszert is alkalmaztunk, kezdve a már említett minden konvolúciós réteget követő kötegelt normalizálással és a teljesen összekapcsolt réteget követő dropout-tal. Ezenkívül korai megállást is alkalmaztunk, ahol a teszthalmazon mért pontosságot figyeltük, és amennyiben 5 epoch-on keresztül nem történt javulás megszakítottuk a tanulás menetét, és a legjobb eredményt hozó súlyokat állítottuk vissza a modellben, így nagyon magas epoch számot is megadhattunk a tanítás kezdetén, hiszen a korai megállás miatt mindig a megfelelő pillanatban állt meg a folyamat. Eleinte a val\_loss mutatót használtuk, munkánk későbbi szakaszában áttértünk a val\_accuracy értékhez, mert ezt véltük számunkra a legfontosabbnak.



Optimalizáció

Optimalizációs eljárásként az ADAM mellett döntöttünk, mivel ezzel kaptuk a legjobb eredményeket. Illetve mini köteges tanítást alkalmaztunk, ami jelen esetben 32-es batch size-t jelentett. Ez volt az a batch méret, amelynek futtatása 128x128-as képek esetén is 32GB RAM mellett problémamentes volt.

Mind a modell paramétereinek megválasztása, mind a regularizációs és optimalizációs lépések végső meghatározása hosszas tesztelést és tervezést igényelt. Számos más kombinációt is kipróbáltunk, mire végül eljutottunk a végleges modellünkig, amellyel a legjobb teljesítményt értük el a teszthalmazon. Kipróbáltunk 3-7-es ablakméretig minden kombinációt a konvolúciós rétegekben, MaxPooling helyett AveragePooling-ot, különböző számú szűrőket és a Dropout rátáját is többször módosítottuk. Optimalizációs eljárásként az ADAM mellett teszteltünk RMSprop-ot is. Ezenkívül a félév során terveztünk egy másik konvolúciós modellt is a végső modellen kívül. Ebben több konvolúciós réteg volt (6), és csak minden második konvolúciós réteg után helyeztünk MaxPooling-ot. Itt is BatchNormalization-t alkalmaztunk, de itt Dropout-ot nem. Mindvégig a tensorflow keras könyvtárát használtuk, és az ebben található metódusokat és osztályokat alkalmaztuk, többek között a Sequential API-t is.

Egyéb tervezési ötletek

Hasonló képek szűrése

Mivel kezdettől fogva látható volt, hogy a modellnek azon karakterek megkülönböztetésével lesz a legnagyobb nehézsége, ahol a kis -és nagybetűk nagyon hasonlóak, ezért felmerült az ötlet, hogy szabaduljunk meg azon képektől, amelyek nagyon hasonlóak, és nem ugyanabba az osztályba tartoznak. Ennek a megvalósításához ezeket a képeket strukturális hasonlósági indexük (SSIM) alapján hasonlítottuk össze. Abban az esetben, ha két kép hasonlósági pontszáma meghaladt egy küszöbértéket, megjelöltük őket, hogy később ki tudjuk őket törölni. Ezzel az eljárással azonban nem sikerült javítanunk a modell tanulási teljesítményét, így végül elvetettük ennek az ötletnek az alkalmazását.

Pretrained modellek használata

Próbálkoztunk előre tanított modellek használatával is, amelyek esetén a ResNet50-et, VGG-16-ot és Xception-t érdemes megemlíteni. Itt nehézséget okozott a kezdetben egy csatornás képek három csatornásra konvertálása, amely szükséges volt ezen modellek használatához. Azonban sem a ResNet50, sem a VGG-16, sem az Xception nem hozott olyan jó pontosságot mint az általunk készített modell, így végül nem ezen modellek használata mellett döntöttünk.

#### 

Dimenziócsökkentés

Annak érdekében, hogy pontosabb eredményeket érjünk el együttes tanulás megvalósítására is gondoltunk, így a konvolúciós neurális hálós modellünket más egyszerűbb modellekkel akartuk megtámogatni. Ehhez első lépésben dimenziócsökkentést végeztünk a képeken, amit UMAP-pal hajtottunk végre. Ezután pedig KNN-t és RandomForestClassifier-t is kipróbáltunk a kisebb dimenziós tanító adathalmazon. Mindkét modell lényegesen rosszabbul teljesített, mint a neurális hálónk, így végül a végső megoldásunkban nem alkalmaztuk ezt a megközelítést.

Eredmények

Baseline modell

Első futtatáskor a gyakorlaton tanult modell segítségével 10 epochon közel 90%os eredményt értünk el a validációs adathalmazon, viszont 11 osztály esetében nem volt képes 0.8 felé hozni a pontosságot. Ennek a modellnek a 10 epochon való futtatása nagyjából 10 percet vett igénybe.

Validation Accuracy: 0.8954

|  |  |
| --- | --- |
| Class 12: 0.64  Class 24: 0.51  Class 28: 0.68  Class 32: 0.79  Class 35: 0.77  Class 50: 0.60  Class 54: 0.66  Class 57: 0.56  Class 58: 0.68  Class 59: 0.63  Class 61: 0.66  Összesen: 11 db |  |

Baseline - Early Stopping

Második futtatásra, mivel egy epoch nagyjából 1 percet vett igénybe, nem adtunk konkrét epoch számot a modellnek, hanem Early Stopping regularizációs módszerrel határoztuk meg a maximális epoch számot, hogy minél jobb lehessen az eredmény. Ekkor a futás 19 epochnál állt meg, de nem kaptunk szignifikánsan jobb eredményt.

Validation Accuracy: 0.8992

|  |  |
| --- | --- |
| Class 0: 0.78  Class 12: 0.62  Class 24: 0.57  Class 28: 0.70  Class 32: 0.72  Class 33: 0.68  Class 38: 0.79  Class 50: 0.66  Class 54: 0.69  Class 57: 0.48  Class 58: 0.79  Class 61: 0.53  Összesen: 12 db |  |

Képek augmentációja

A képek augmentációja után sikeresen elértük a 90%os határ feletti pontosságot, viszont még mindig nagyjából ugyanannyi osztályra értékelt ki 80% alatt.

Validation Accuracy: 0.9206

|  |  |
| --- | --- |
| Class 24: 0.44  Class 28: 0.71  Class 33: 0.69  Class 35: 0.78  Class 38: 0.78  Class 47: 0.78  Class 50: 0.74  Class 54: 0.78  Class 57: 0.72  Class 58: 0.74  Class 61: 0.74  Összesen: 11 db |  |

Végleges modell

A végleges modell implementálásával nem csak a pontosságot tudtuk növelni, hanem a hibás osztályok számán is javultak az eredmények.

Validation Accuracy: 0.9254

|  |  |
| --- | --- |
| Class 18: 0.77  Class 24: 0.70  Class 28: 0.70  Class 32: 0.73  Class 38: 0.64  Class 50: 0.52  Class 57: 0.59  Class 59: 0.68  Class 61: 0.75  Összesen: 9 db |  |

Képméret növelése

GPU támogatás implementálása után növelni tudtuk a képméretet anélkül, hogy a batch méretet csökkenteni kelljen, és minden konvolúciós réteg után használtunk egy Max Pooling réteget, így várhatóan még jobban teljesített az új modell, viszont a futásidő még gyorsabb is volt, mint CPU-n 28-as képmérettel (GPU-n nagyjából 6-7 másodperc volt egy epoch 28-as képmérettel, 128-al 30-40 másodperc).

Validation Accuracy: 0.9318

|  |  |
| --- | --- |
| Class 24: 0.60  Class 28: 0.78  Class 33: 0.63  Class 35: 0.79  Class 38: 0.78  Class 50: 0.54  Class 54: 0.78  Class 57: 0.76  Összesen: 8 db |  |

Early stopping val\_accuracy metrikával

Ezen a ponton jöttünk rá, hogy az Early stopping a val\_loss-t monitorozza a futáskor, így a hosszabb futás árán (20-22-ről 28-32-re ugrott az átlagos epoch szám), de minimálisan jobb pontosságot értünk el.

Validation Accuracy: 0.9342

|  |  |
| --- | --- |
| Class 24: 0.59  Class 28: 0.71  Class 31: 0.77  Class 38: 0.79  Class 50: 0.74  Class 59: 0.79  Class 61: 0.77  Összesen: 7 db |  |

Optimalizáció

A konvolúciós rétegek számának és méreteinek kioptimalizálása után és a forgatás kikapcsolásával még tudtunk javítani az eredményen. Ezekkel az ablakméret konstrukciókkal próbálkoztunk, a mellette levő eredmény pedig a validációs halmazon mért pontosság:

* (3,3) (3,3) (3,3) (3,3) - 93.96
* (4,4) (4,4) (4,4) (4,)4 - 93.38
* (5,5) (5,5) (5,5) (5,5) - 94.14
* (5,5) (5,5) (4,4) (3,3) - 94.23
* (5,5) (5,5) (3,3) (3,3) - 93.79
* (5,5) (5,5) (4,4) (3,3) (3,3) - 93.17
* (5,5) (5,5) (5,5) (3,3) (3,3) - 93.38
* (5,5) (5,5) (5,5) (4,4) (3,3) - 93.78
* (5,5) (3,3) (5,5) (3,3) (4,4) - 93.59
* (6,6) (6,6) (6,6) (6,6) - 93.72
* (7,7) (6,6) (5,5) (4,4) (3,3) - 93.96
* (5,5) (3,3) (5,5) (3,3) - 93.72

Próbálkozás másik modellel

6 konvolúciós rétegű, csak minden második konvolúció utáni Max Poolinggal rendelkező neutális hálóval is próbálkoztunk.

Validation Accuracy: 0.9382

|  |  |
| --- | --- |
| Class 24: 0.69  Class 28: 0.74  Class 35: 0.79  Class 50: 0.64  Összesen: 4 db |  |

Pretrained modellek

**VGG16**

Validation Accuracy: 0.9039

|  |  |
| --- | --- |
| Class 18: 0.23  Class 24: 0.49  Class 28: 0.71  Class 31: 0.66  Class 32: 0.68  Class 33: 0.60  Class 35: 0.71  Class 50: 0.70  Összesen: 8 db |  |

**ResNet50**

Validation Accuracy: 0.8897

|  |  |
| --- | --- |
| Class 12: 0.73  Class 18: 0.34  Class 24: 0.47  Class 28: 0.74  Class 31: 0.62  Class 32: 0.78  Class 33: 0.63  Class 35: 0.56  Class 47: 0.75  Class 50: 0.62  Class 54: 0.69  Class 58: 0.73  Összesen: 12 db |  |

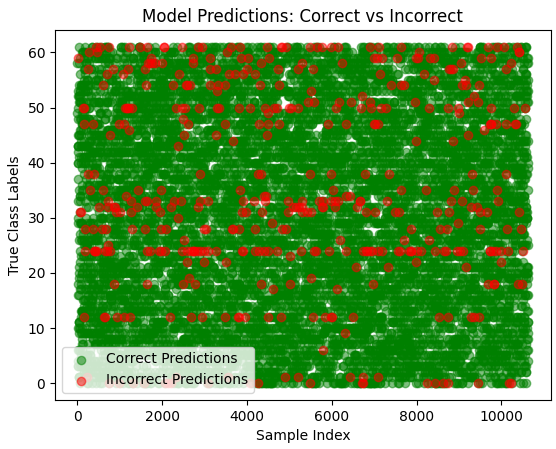
**Xception**

Validation Accuracy: 0.8591

|  |  |
| --- | --- |
| Class 0: 0.76  Class 12: 0.63  Class 18: 0.32  Class 24: 0.57  Class 28: 0.64  Class 31: 0.64  Class 32: 0.61  Class 33: 0.67  Class 35: 0.54  Class 47: 0.79  Class 50: 0.65  Class 54: 0.77  Class 59: 0.74  Összesen: 13 db |  |

Forgatás nélkül, az 5,5,4,3 konvolúciós réteggel ellátott model eredményei:

|  |  |
| --- | --- |
| Class 24: 0.53  Class 28: 0.78  Class 61: 0.78  Összesen: 3 db |  |



Konklúzió

A projekt során sikeresen megvalósítottuk a karakterosztályozási feladatot egy mélytanulási modell segítségével. A fejlesztés során számos módszert és technikát alkalmaztunk, például adataugmentációt, paraméteroptimalizálást, valamint több alternatív modellt és megközelítést is teszteltünk.

Az általunk készített modell, egy többrétegű konvolúciós neurális háló, kiemelkedő teljesítményt mutatott a tanító adathalmazon, és felülmúlta a baseline megoldásokat, valamint az előre tanított modellek eredményeit is. A rendszer hatékonyságát tovább növeltük az augmentációval, dropout rétegekkel, batch normalizációval és korai megállással, amelyek segítettek a túlzott túltanulás elkerülésében és a modell általánosítási képességeinek javításában.

A végső tesztelés során, azonban kiderült, hogy a modellünk valószínűleg kissé túltanult és nem tudott kellő mértékben általánosítani a nehezen felismerhető karakterekre. Ez annak következtében alakult ki, hogy az early stopping patience paraméterét túl magasra állítottuk, ami tovább hozzájárult a túltanuláshoz és a rosszabb eredményhez. Ezen felül, a megfelelő keresztvalidálás hiánya miatt a modellt nem tudtuk kellő alapossággal kiértékelni. Így az általunk kapott 94.23-as validation accuracy helyett 84.5-öset kaptunk a teszthalmazon. Végezetül, ha úgy döntünk, hogy egy pretrained modellt használunk, akkor lehetséges, hogy bár a saját teszteléskor kisebb accuracyt kaptunk volna, de mivel az a modell több adaton tanult, a végső teszt során kevésbé tért volna el az általunk kapott eredménytől.

Kontribúció eloszlása

* **Czotter Benedek**: Modellek készítése, valamint folyamatos tesztelése
* **Tarsoly Levente:** Pretrained modellek tesztelése, valamint GPU acceleration megvalósítása.
* **Hermann Máté:** Beolvasás megvalósítása, és optimális paraméterek keresése.

Emellett sokat foglalkoztunk együtt is a projekttel, valamint nagyobb “checkpoint”-oknál megbeszéltük épp mit lehetne tesztelni, minek az optimalizálását lehetne kipróbálni. Miután nagyjából kész volt a modellünk mindannyian külön külön próbáltuk optimalizálni és Levente gépén tesztelni a legjobb eredményeket hozó paramétereket.

Irodalomjegyzék

<https://medium.com/@anushreedas.2710/image-dataset-analysis-using-python-libraries-pandas-and-matplotlib-a640e5f59805>

<https://medium.com/@sssspppp/image-classification-using-cnn-0fad8367acfd>

<https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html>

<https://chatgpt.com/>

<https://towardsdatascience.com/batch-norm-explained-visually-how-it-works-and-why-neural-networks-need-it-b18919692739>

<https://developer.nvidia.com/cuda-downloads?target_os=Linux&target_arch=x86_64&Distribution=WSL-Ubuntu&target_version=2.0&target_type=deb_local>

<https://developer.nvidia.com/cudnn-downloads?target_os=Linux&target_arch=x86_64&Distribution=Ubuntu&target_version=24.04&target_type=deb_local>

<https://www.tensorflow.org/install/pip>

<https://medium.com/@ajaytshaju/how-to-use-tensorflow-with-gpu-on-windows-for-heavy-tasks-2024-a1536448b0fe>

<https://keras.io/api/applications/>