Gépi látás

GKNB_INTM038

Kézzel írt karakterek felismerése

Joós Tibor

G04H4D

Győr, 2020/2021/1

Tartalomjegyzék

1 Bevezetés

A beadandó dolgozat címe is hűen tükrözi a témáját, azaz a kézzel írt karakterek felismerését. A beadandó dolgozat célja egy olyan program fejlesztése, amely képes kézzel írt karaktereket felismerni. Ezek a karakterek minden esetben homogén háttéren helyezkednek el, illetve ezek számok, illetve az angol abc kis-, és nagybetűi lehetnek.

A feladat a digitalizáció témakörében egy rendkívül érdekes feladat. Napjainkban egyre nagyobb szerepet kap a digitalizáció, ez pedig magával vonzza például a régi, kézzel írt dokumentumok digitalizálását is. Ehhez pedig elengedhetetlen a karakterfelismerés. A digitalizáció ezen iránya azonban már a 90-es években is létezett [1], ám mára ez a témakör rendkívül széles körűvé vált, mivel elérhetőek már számos nyelvhez ilyen karakterfelismerők, például: kínai betűkhöz, arab betűkhöz, latin betűkhöz és lehetne még sokáig folytatni a sort. A számos módszert sok esetben összemérik egymással mennyire hatékonyak, ezekhez pedig újabb és újabb adatbázisokat használnak a folyamatos fejlődés érkében. [2]

A feladatban először is szükség lesz az elméleti háttér áttekintésére. A következő bekezdés során bővebben ki fogok térni ezekre a fontos elméleti tudásokra. Mivel a megoldás mesterséges intelligencia alapokra fog épülni, ezért szükség lesz a neurális hálózatok bővebb megismerésére. Az elméleti hátteret követően a szoftvertervezés fázisának bemutatása fog következni. Itt a legfontosabb döntés, hogy mennyire legyen a megoldás modern és okos. Én a modernizáció végett egy mesterséges intelligencia által támogatott megoldást fogok megvalósítani, és részletezni. A tervezést követően a konkrét szoftver megvalósítását fogom részletezni. Itt szót ejtek majd bővebben a programozási nyelv nyújtotta előnyökről, a szükséges csomagokról, illetve a szoftver felépítéséről, illetve fejlődéséről. Ezt követően a kész szoftver tesztelése fog következni. Ez egy viszonylag nagyobb adathalmazon fog történni, hogy minél valósághűbb és pontosabb legyen az eredmény. Végezetül egy rövid felhasználói leírást fogok adni a szoftver használatához.

2 Elméleti háttér áttekintése

2.1 Írásfelismerésről általánosan

Az írásfelismerés az első elektronikus tabletekkel egyidőben jelent meg, az 1950-es években. Kis visszaesést követően az 1980-as években azonban újra fejlődésnek indult a technológiával karöltve. Napjainkban már számos különféle technológia létezik, többféle karakterkészletre.

2.2 Írásfelismerési technológiák

Az írásfelismerésen belül a két legnagyobb csoportot az online és offline írásfelismerés képezi. Offline írásfelismerésről akkor beszélünk, amikor az írásfelismerés csak az írás befejezése után hajtódik végre. Itt természetesen évek is eltelhetnek. Ehhez a technológiához szükség van egy optikai szkennerre, a digitalizálandó írás beolvasáshoz.

Online írásfelismerésről akkor beszélünk, amikor az írás felismerése már az írás közben megtörténik. Természetesen ez a technológia már rendelkezik gépigénnyel. Így az eszközön múlik, hogy mennyivel, de kisebb-nagyobb mértékben mindig le lesz maradva az írásfelismerés a valós írási gyorsaságtól. Az átlagos írási sebesség például angol karakterek esetén 1.5-2.5 karakter/másodperc. A csúcsérték speciális karaktersorozat esetén 5-10 karakter/másodperc is lehet. Természetesen az egyre mikroprocesszoroknak köszönhetően ma már sokszor előbb ajánlatot kapunk arra, hogy mit szeretnénk beírni, mielőtt beírtuk volna az adott karakterünk. Az online írásfelismeréshez szükség van egy transzducerre is, ami rögzíti az írást. [3]

2.3 Egy, illetve többnyelvű írásfelismerés

Természetesen minden írásfelismerőnek rendelkeznie kell egy adatbázissal, melyből kiválasztja a leírt karaktert. Mivel több nyelv írásában is találhatóak egyedi karakterek, így egy technológia nem biztos, hogy több nyelven is működik. Azonban a nagy tárolási kapacitásnak köszönhetően napjaink írásfelismerő szoftverei általában többnyelvűek. Természetesen, ha a szoftver készítői lehetőséget adnak a nyelv beállításaira a lehetséges karakterek számát lehet dinamikusan csökkenteni.

2.4 Gépi tanulás

A gépi tanulás a modern informatikai megoldásokkal együtt jelent meg. Röviden összefoglalva a fogalmat azt mondhatjuk, hogy a gépi tanulás célja, az emberi gondolkodás és felismerés lemodellezése számítógépek segítségével. A gépi tanulás szorosan kapcsolódik a robotika tudományához is. Természetesen mivel emberi tulajdonságokkal próbálunk felruházni gépeket, ezért is ők is képesek tévedésre, mellélövésekre. A gépi tanulás fő iránya, célja az lehet, hogy az embereket segítő, saját, önálló gondolkodással rendelkező gépeket hozzunk létre. A témán belül amúgy is fontos kérdés a gép és ember együttműködése. [5]

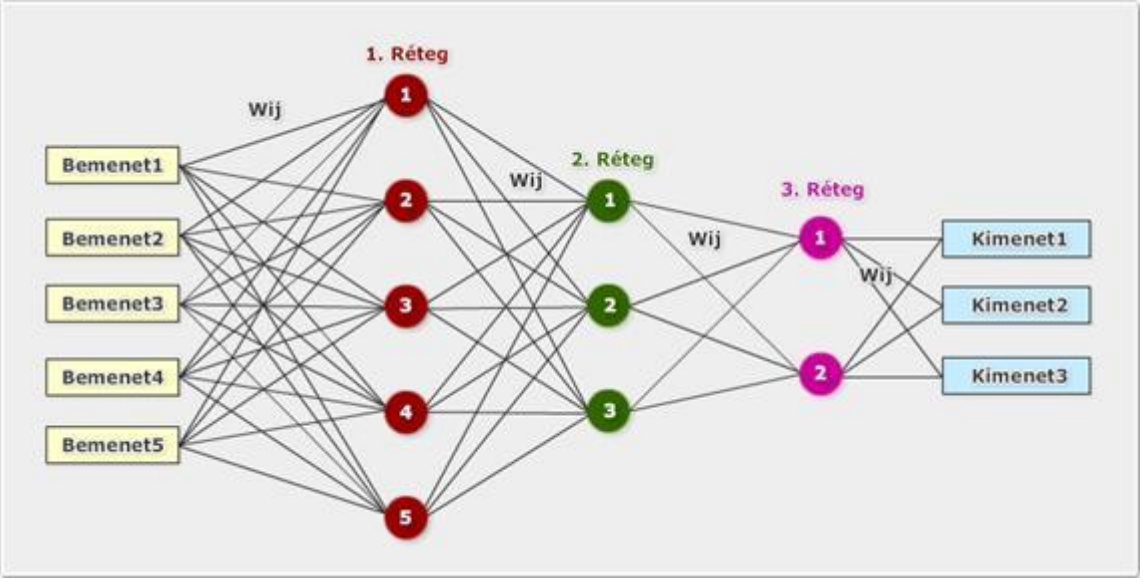
A gépi tanulás úgy gondolom az informatika egy jó iránya lehet, mindaddig amíg az ember ellen nem szeretne fordulni a gép. A feladat megoldása során például egy jó és hasznos megoldásra használom, és úgy vélem a jövőben is ennek kellene világméretűleg is a fő iránynak lennie.

2.5 Neurális hálózatok

A neurális hálózat a mesterséges intelligencia megoldásokhoz szorosan kapcsolódó fogalom. A gépi tanulás során elengedhetetlenné vált napjainkban. A mesterséges neurális hálózat az idegrendszer felépítése és működése analógiájára kialakított számítási mechanizmus. Hiszen a fő cél nem elvi, hanem ténylegesen működő modell létrehozása. Ezt pedig tipikusan valamilyen elektronikai eszközzel és valamilyen tudományos eljárással lehet elérni. Tehát a biológiai elvek alapján megalkottak bizonyos matematikai jellegű modelleket. Ezeket elméleti matematikai módszerekkel pontosították, alkalmazott matematikai módszerekkel számításokra alkalmassá tették, majd számítógépen realizálták. Azonban a

matematikai módszerek mellett sokszor heurisztikus meggondolásokra és számítógépes kísérletezésre is szükség van. Egy neurális hálózatot érdemes úgy felfogni, hogy nem kívánja a jelenséget modellezni, arra törvényszerűségeket megállapítani, hanem a jelenséget fekete dobozként kezeli, csak a bemenő (input) és a kimenő (output) adatokat tekinti. A jó neurális hálózat olyan, hogy ugyanarra az inputra hasonló outputot ad, mint a vizsgált jelenség. De a fekete dobozban működő mechanizmust nem tárja fel, maga a neurális hálózat pedig nem hasonlít a jelenségre. A jelenségek fekete dobozként való kezelése a neurális hálózatoknak részben hátrányuk, de részben előnyük is. Hiszen működésükhöz csak adatokra van szükség. [4]

A neurális hálózatok lehetnek az egészen egyszerűek, ám akár számos rejtett réteggel rendelkezőek is, melyek fontosak lehetnek a gépi tanulásban.



1. ábra: Több rejtett réteggel rendelkező neurális háló
Forrás: https://regi.tankonyvtar.hu/en/tartalom/tamop412A/2010-0017_08_meres_es_iranyitastechnika/ch06s03.html (Megtekintés napja: 2020-11-23)

Egy neurális hálózat felépítéséről általánosságban elmondható, hogy rendelkezik bementi adatokkal, melyek a közbeeső rétegen(rétegeken) áthaladva fognak kimeneti eredményt adni. Minél pontosabb eredményt szeretnénk elérni, annál több rejtett rétegre lehet szükségünk.

2.6 Konvolúciós neurális hálózatok [6]

A klasszikus képfeldolgozásnál alapművelet a konvolúció. Alkalmazható zajszűrésre, alacsony képi jellemzők kiemelésére, összetett objektumok kiemelésére. A konvolúciós neurális hálózatok előnye a teljesen összekötött hálókhoz képest, hogy jóval kevesebb a szabad paraméter. A konvolúciós rétegben a bementi kép egy megadott nagyságú részét kivesszük és számítást végzünk rajt. Ebben a rétegben egy neuron érzékenységi mezőjének a bemeneti kép azon részét nevezzük, melytől függ a kimeneti értéke.

A pooling réteg lényegében mitavételezi a képet. Az *avarage pooling* a gyakori minták kiemelését segíti elő. A *max pooling* olyan jellemzők kiemelését segíti, melyek csak kis számban fordulnak elő. Az ilyen hálózatok rendelkeznek egy *sorosító* (angolul: *flatten*) réteggel is. Ennek a rétegnek a feladata a többdimenziós jelek egy dimenzióssá alakítása. Ez a réteg nem tartalmaz tanítható paramétereket. Ilyen neurális hálózat például a *DenseNet(2017).* A módszer lényege, hogy legyen rövid hiba-visszaterjesztési utak. További jellemzője, hogy egy blokkon belül minden réteg kimenete közvetlen bemenete minden azt követő rétegnek.

A konvolúciós neurális hálózati technológia egyik fő alkalmazási területe az objektumdetektálás. Ez a detektálás történhet csúszó ablakosan, régión alapúan. A csúszó ablakos megoldás különböző méretű téglalapokat tol végig a vizsgálandó képen, és az így kivágott részeket osztályozza. Az osztályokat a felismerendő objektum típusok és a háttér alkotja. Előnye, hogy kis munkaigényű, azonban hátránya, hogy vagy gyors és pontatlan, vagy lassú a pontosság növelése érdekében. A régión alapuló megoldás bemenete a kivágott és átméretezett képrészlet, míg a kimenete pedig a képrészlet osztályozása és a pozíciójának korrekciója. Gyakorlati alkalmazását tekintve egy rendkívül lassú megoldás, így létezik gyorsított változata is. A gyorsítás alapgondolata, hogy a teljes képet vizsgáljuk egy mély, teljesen összekapcsolt konvolúciós neurális hálóval, majd az abból kinyert jellemzőket vizsgáljuk egy sekélyebb neurális hálóval.

3 A feladat megvalósítási terve és kivitelezése

3.1 Ötlet, alaptervek

A téma kiválasztását követően elkezdtem különféle megoldásokat keresni, ezeket átgondolni. Lehetséges lett volna egyszerű, tisztán képfeldolgozásos megoldás, ám úgy éreztem mivel mai informatikai ismeretek alapján nem nehéz elkészíteni, így egy gépi tanulásos megoldást választok. Miután az alapötlet megvolt elkezdtem gondolkodni a konkrét megvalósításon. A feladathoz szükség van megfelelő adatbázisra, megfelelő neurális hálózatra, illetve a tesztekhez egy megfelelő tesztkészletre.

3.2 Adatbázis

Az adatbázis kiválasztásánál fontos szempont volt számomra, hogy minél több tanító, illetve validáló képet tartalmazzon. Emellett a kiválasztásnál a másik fontos szempont az volt, hogy legyen könnyen tanítható. Mindezek mellett fontos volt, hogy ingyenesen elérhető legyen. Ezen szempontok alapján két megfelelő adatbázist találtam. Az egyik az ***MNIST adatbázis*** [7], mely kézzel írott számokat tartalmaz kizárólag.

Az adatbázis 60 ezer tanító képet és 10 ezer tesztképet tartalmaz. Ez a gyűjtemény a kezdeti tesztekre, illetve alapmodell építésre kiválóan alkalmas volt. Emellett kiváló alapot szolgál összemérni a csak számokat felismerő modell, illetve a betűket is felismerő modell közti különbségeket.

A másik általam választott adatbázis a kibővített MNIST, vagyis az ***EMNIST adatbázis*** [8] volt. Ez az adatbázis már tartalmaz kézzel írt karaktereket is. Megtalálhatóak az adatbázisban az angol ABC nagybetűi, illetve kisbetűinek egy része, valamint tartalmazza a kézzel írt számokat is.

Az általam használt EMNIST adatbázis a *Balanced* verzió. Ez a verzió 131600 tanító karaktert tartalmaz 47 osztályhoz, vagyis összesen 47-féle karaktert képes felismerni ez alapján a modell. Ugyan ez az adatbázis nem tartalmazza az összes angol kisbetűt, a megbízhatóság miatt úgy gondoltam inkább ezt használom, mint a kevésbé kiforrott, bővebb adatbázisokat.

3.3 Fejlesztői környezet, programnyelv, programcsomagok

A projekthez fejlesztő környezetként a ***Visual Studio Code*** [9] alkalmazást használtam. A környezet használatának az előnye, hogy többféle programozási nyelvet támogat, emellett alacsony helyigényű és könnyen, jól személyre szabható. További előnye még többek között, hogy platformfüggetlen.

A projekt programozási nyelvének a ***Pythont*** választottam. Azért esett a választásom erre a programnyelvre, mivel a tárgy keretein belül is ezzel a programozási nyelvvel ismerkedhettem meg jobban, emellett kiválóan alkalmas gépi tanulásos projektekhez. Emellett a nyelv előnye, hogy könnyen tanulható, szintaktikája és adatkezelése egyszerű.

A fejlesztés során többféle programcsomag használata volt szükség. A gépi tanulás és a neurális hálózatok végett elengedhetetlen volt a ***tensorflow*** [10] programcsomag használata. Ez a programcsomag szolgál a neurális hálózatok felépítésre. Egy egyszerű neurális tanító hálózat felépítéséhez elegendő lehet akár csupán ezen programcsomag használata. Az adatok kezeléséhez, beolvasásához, feldolgozásához a ***numpy [11]***, illetve a ***pandas*** [12] programcsomagokat használtam. Előbbi csomag előnye, hogy a tömbökkel való munkát rendkívül egyszerűvé teszi. Utóbbi programcsomag felfőbb előnye pedig, hogy bonyolult adatszerkezetek is képes hamar és hatékonyan beolvasni, például egy csv fájlt. A gépi tanulásos programrészeknél ezen kívül csupán két kisebb csomagösszetevőt használtam, melyek az ***sklearn*** [13] és a ***keras*** [14] programcsomagból származnak.

A tesztek automatizálásakor szükségem volt még az ***OpenCV*** [15] programcsomag használatára, mely a képek beolvasását segítette. A grafikus felület programozásakor még szükségem volt alapvető, a grafikus felületek programozásával kapcsolatos csomagra is. Ilyen volt a ***Pillow [16]*** és a ***tkinter*** [17].

3.4 Egyszerű neurális hálózat programozása írott számok felismeréséhez

Elsőként egy egyszerű neurális hálózatot tanítottam be. Ez jó volt számomra, mivel ezen tudtam tanulni a gépi tanulással kapcsolatos módszereket, lépéseket. A végeredményben kapott modell számok felismerését teszi lehetővé. Ezen bekezdés programkódját a ***make_num_model.py*** fájl tartalmazza.

A modell készítéséhez az *MNIST adatbázist* használtam, mely a *tensorflow* programcsomagnak köszönhetően egyszerűen betölthető, hiszen eleve tartalmazza az adatbázist. Ezt követően a tanító és a teszt adatokat normalizálni kell, majd a normalizálást követően kezdődhet a neurális hálózat kialakítása.

Az általam választott neurális hálózat egy szekvenciális, másnéven időfüggő hálózat. Ezen modelltípus segítségével könnyen programozhatóak dinamikus rendszerek, ahol fontos az adatok időrendje is. Emellett az ilyen szekvenciális modellek alkalmasak jóslási problémák megoldására. Ez a fő oka a választásomnak.

A modell inicializálását követően hozzáadtam a szükséges rétegeket. A bementi réteg egy *Flatten* réteg lett, mivel a bemenetek egységes méretűek és az ilyen problémákhoz kiválóan alkalmazható ez a réteg. A bementi méretet a programban be is állítottam az adatbázisnak megfelelő 28x28 pixelre. A következő, első köztes réteg 128 elemű *Dense* réteg, melynek aktiválása *relu,* vagyis *javított lineáris aktiválási függvény.* Ez az eljárás közvetlenül adja ki a bementet ha pozitív, ellenkező esetben nullát ad vissza. Számos neurális hálózatban alapértelmezett aktiválási funkcióvá vált, mivel az ezt használó modellek könnyebben taníthatóak és pontosabbak. [19] A pontosság növelése érdekében egy újabb, teljesen megegyező *Dense* réteget is hozzáadtam a hálózathoz. A végső, kimeneti réteg is egy újabb Dense réteg, ám ez már csak 10 elemmel rendelkezik, mivel 10 féle szám felismerésére alkalmas. Ennek a rétegnek az aktiválása *softmax*, vagyis a 10 lehetséges megoldás közül azt adja eredményül, amelyiknek a legnagyobb a valószínűsége.

Miután a modell ezzel elkészült, következik a lefordítása. A fordítás során az ***adam optimalizálót*** használtam. Ez az optimalizáló algoritmus szolgál a hálózati súlyok iteratív frissítésére, a tréning adatok figyelembe vételével. A veszteség számításához a ***ritka kategorikus keresztcentrikusságot*** használatam. Ez az algoritmus arra szolgál, hogy kiszámítja a címkék és a jóslatok közötti keresztcentrikus veszteséget. Ezt a funkciót akkor célszerű használni, mikor kettő vagy több kimenet van, így ez jelen esetben kiválóan alkalmas a feladatra. [14] Metrikáknak a pontosságot használtam. Ez a módszer kiszámolja, hogy a jóslat milyen gyakran egyezik meg azonos kimenetekkel. Emellett két helyi változót hoz létre, a *totalt* és a *countot*. Ezek segítségével az egyezések gyakorisága könnyen kiszámolható. Az algoritmus a végén egyszerűen csak elosztja egymással a két adatot. [14]

A fordítást követően a modell tanítása következik. A tanítás során a megfelelő adatok használata mellett beállítottam, hogy a hálózaton egyszerre 16 minta továbbítódjon. Emellett beállításra került, hogy a tanító folyamat 5-ször fusson végig. Ez ugyan nem nagy szám, de a projekt méretét tekintve így is magas, 90% feletti pontosságot sikerült elérni. A tanítást követően a modell mentésre került további felhasználás céljából.

3.5 Konvolúciós neurális hálózat programozása írott számok felismeréséhez

Ezen bekezdés programkódját a ***make_num_model_conv.py*** fájl tartalmazza.

Az előbbi modell továbbfejlesztéseképpen a neurális hálózatot konvolúcióssá alakítottam. Hasonlóan az előbbihez az adatok betöltésével kezdtem. Annyival volt nehezebb itt a feladat, hogy nem közvetlenül a tensorflow programcsomagból olvastam be az adatokat, hanem előre letöltött fájlból, így szükség volt a pandas programcsomag használatára is. A beolvasást követően a normalizálás során a képeket megfelelően forgattam és a megfelelő típusúra állítottam. Ezen modell esetén az osztályok számát külön be kellett állítani. Ezt a tanító-, és tesztadatok alapján megtettem. Mivel ez a modell más alakot vár el, ezért elvégeztem a szükséges újraformálást.

Miután ezek a szükséges lépések megvoltak kezdődhetett a háló építése. Hasonlóan az előzőhöz egy szekvenciális modell mellett döntöttem. Első rétegként egy kétdimenziós konvolúciós réteg került a modellbe. Ez a réteg 6 szűrővel és 3x3-as szűrő mérettel rendelkezik. Az aktiválása relu. A bemeneti méretet itt is beállítottam az MNIST adatoknak megfelelően 28x28-as méretre. Következő rétegként bekerült egy átlagos térbeli adatok összesítésére szolgáló réteg. Ezt követően az előbbi két réteg került be ismét a modellbe, annyi változtatással, hogy a konvolúciós réteg 16 szűrővel rendelkezik másodjára. A konvolúció után az egyszerű modell elemei kerültek be a modellbe; a Flatten, és a három Dense réteg. A fordítás teljesen megegyezik az egyszerű modellben alkalmazottal. A tanításban az ismétlések számát 5-re állítottam, mivel ennyivel már elérhető volt a kellő pontosság. A hálózaton ebben a modellben egyszerre 16 minta továbbítódik, illetve a tanítás során megtörténik az adatok validálása is a tesztadatok alapján. Végül a modell mentésre kerül.

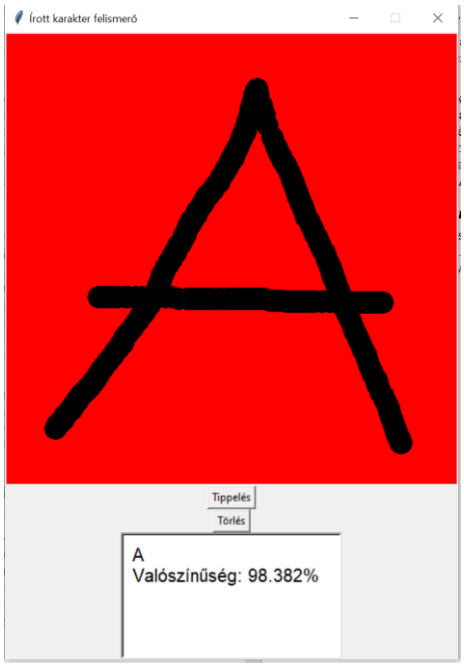
3.6 Konvolúciós neurális hálózat programozása írott karakterek felismeréséhez

Ez a hálózat szolgálja azt a célt, hogy az EMINST adatbázis alapján képes legyen nem csak számok, hanem betűk felismerésére is. Ezen bekezdés programkódját a ***make_char_model_conv.py*** fájl tartalmazza.

Maga a modell rendkívül hasonlít a számfelismerésre alkalmas konvolúciós neurális hálózathoz, csupán az adatok miatt adódnak különbségek. A különbségek a modell belsejében figyelhetőek meg. A három belső Dense réteg esetében az elemek száma 120, 84, illetve 47. Utóbbi szám arra utal, hogy 47 lehetséges kimenetünk van. A modell fordításakor a veszteség számításához a simán ***kategorikus keresztcentrikusságot*** alkalmaztam. A betanításnál az ismétlések számát 5-re állítottam, hogy elérjem a kellő pontosságot. A végén ez a modell is mentésre került későbbi felhasználás céljából.

3.7 Grafikus felhasználói felület

A feladat végrehajtás utolsó lépéseként egy grafikus felületet készítettem. Ez a felület szolgál a modellek tesztelésére, illetve arra, hogy a felhasználó önmaga is ki tudja próbálni. A felhasználói élmény ugyan nem a legnagyobb, de úgy gondolom a célnak teljes mértékben megfelel.



2. ábra: A program felhasználói felülete
Forrás: saját készítés

A grafikus felület mögötti szoftver elsőként betölt egy megadott modellt, illetve betölti az osztályzáshoz szükséges szöveges filet. Ezt követően beállításra kerülnek a globális szinten szükséges változók. Ezen szakasz után a különböző segédfüggvények kerülnek definiálásra. Elsőként a ***testing*** függvény, melynek feladata, hogy a képet, melyen a rajzolt karakter van beolvassa, majd ez alapján jóslatot tegyen. Eredményként visszaküldi a jóslatokat tartalmazó listát. A ***paint*** metódus feladat, hogy a felhasználó egérmutatójának mozgása alapján kirajzolja a karaktert. A ***tipp*** metódus elmenti a rajzoló felület aktuális állapotát egy átmeneti fájlban. Ezt követően meghívja a testing függvényt. Végül kiírja a képernyőre, hogy mennyire valószínű, hogy jó a program által adott tipp, illetve megjeleníti a tippelt karaktert is. A ***clear*** metódus törli a képernyő pillanatnyi állapotát és alaphelyzetbe helyezi a program felületét.

A kódban a függvények utáni részen történik a felület inicializálása, alaphelyzet beállítása. Itt kerül beállításra a rajzolási háttér színe, a szövegdoboz és a gombok megjelenése. A gombokhoz természetesen a szükséges esemény is hozzárendelésre kerül. A beállítást követően indul a főprogram, mely egészen addig fut, amíg a felhasználó be nem zárja a programot.

A grafikus programrész eléggé alapszintű, számos továbbfejlesztése lehetséges. Ezek között elsőként kiemelendő talán, hogy implementálható lenne egy fájlbeolvasó rész, ahol a számítógépről a felhasználó a saját befotózott képére is tudjon jóslatot kérni. Emellett a dinamikus, felhasználó által egyszerűbb személyre szabást is meg lehetne valósítani. Például a háttérszín jelenleg csak a programkódban módosítható. Összességében úgy gondolom viszont a felhasználói felület tesztelése és alap felhasználási célokra teljesen mértékben megfelel és kellően hasznos.

4 A szoftver tesztelése

A témából adódóan a tesztelés a program működésének ellenőrzése végett fontos feladat. Könnyű általa visszajelzéseket kapni, hogy hol érdemes esetleg javítani, mi az, ami jól működik és még számos haszna lehet. A tesztelés megkezdéséhez elsőként egy teszttervezést végeztem.

4.1 Teszttervezés

A teszttervezés folyamán átgondoltam hogyan lehet sokrétűen, alaposan letesztelni az elkészített szoftvert. A tervezés során elsőként szerettem volna a szoftvert viszonylag könnyebben kezelhető bemenetekre letesztelni. Erre a végső megoldásom az lett, hogy készítettem egy számítógépen rajzolt tesztkészletet fehér háttérrel, fekete betűvel. Ez a pixelek könnyű megkülönböztetése miatt egyszerű felismerést segít elő. Mivel úgy gondoltam, hogy egyre inkább terjed a digitális eszközön (*például: tableten*) történő jegyzetelés, ezért készítek többféle számítógépes szoftverrel készített tesztkaraktereket tartalmazó tesztkészletet is. Ezt követően szerettem volna valódi, kézzel írt karaktereket tartalmazó tesztkészleteket is készíteni. Itt még a tervezéssel kapcsolatban az volt az elképzelésem, hogy kicsit nehezítek a program számára, így próbáltam kicsit a fényviszonyokkal is játszani.

4.2 Tesztkészletek

Végső összegzésképpen a számfelismeréshez négy különböző tesztkészlet készült. Ezen mindegyike számítógépes szoftveren rajzolt. Azonban a különböző tesztkészletek különböző nehezítéseket tartalmaznak. A négy tesztkészlet sajátosságai a következőek:

- *„hirtelen írásmódú”* tesztkészlet, mely arra a való életből merített szituációkra igyekszik példát adni, amikor a felhasználó csak úgymond hirtelen felskiccel valamit a lapra, képernyőre,
- *„odafigyelt írásmódú”* tesztkészlet, mely során igyekeztem odafigyelni, hogy a számok alakját szépen kerekítsem annak érdekében, hogy a szoftver nagy pontossággal felismerje az adott számot,
- *„vastag írásmódú”* tesztkészlet esetén a számokat vastagabb „tollal” írtam, ugyanakkora képméter mellett,
- *„színes hátterű”* tesztkészletben pedig a számok valamilyen homogén, de színes háttér előtt lettek írva.

A karakterfelismerő modell teszteléséhez több, hat tesztkészlet készült. Ezek közül kettő számítógépes szoftver segítségével íródott, míg négy valódi kézírással. A számítógépes tesztek közül az egyik az angol ABC nagy-, a másik a kisbetűit tartalmazza. Igaz, hogy a tanító algoritmus nem készíti fel a szoftvert az összes kisbetű felismerésére, de úgy gondoltam, hogy kíváncsiságból készítek azokra is példát, hogy mit mondd rá a program. A számítógépes tesztkészletekben még az a plusz nehezítés, hogy némelyik karakter színes háttérre lett írva.

A négy kézzel írott tesztkészlet a következő:

- *ColorBG* színes háttér előtt, árnyékmentesen fotózott,
- *FarLight* messzebbről fotózott, ezáltal árnyékos képek, melyek fehér háttéren feketén írt karaktereket tartalmaznak,
- *FarLightColor* az előzőhöz teljesen hasonló, csak itt a karakterek színes háttérre lettek írva,
- *LightWhite* az elsőként említett tesztkészlettel nagyban megegyező, csak itt a karakterek fehér háttérre lettek írva.

Ezen mennyiségű teszt úgy gondolom, hogy a projekt méreteinek megfelel, így hát következhetett a valódi tesztelés.

4.3 Tesztek előkészítése, automatizálása

Úgy vélem, a mai modern, informatikai világban már illik nem egyesével tesztelni egy ilyen kis tesz-halmazt sem. Így hát valamilyen formában szerettem volna a tesztjeimet automatizálni. Ehhez úgy gondoltam, hogy egy strukturált dokumentumszerkezetet hozok létre. Ennek értelmében a különböző modellekhez tartozó tesztkészletek külön mappába kerültek. Ezen mappákon belül is igyekeztem logikailag jól elkülöníteni a készleteket. Maguknak a rajzolt, fotózott képeknek a nevét is igyekeztem rámutatóan megadni. Így a fájlnevek első betűi adott, leírt karaktert tartalmazzák. A fájlnév többi része a tesztkészlettől, ismétlődéstől függően alakult. Természetesen ahhoz, hogy a tesztek automatikus lefussanak megfelelő programkód elkészítése is szükségessé vált.

4.4 Tesztelő algoritmusok

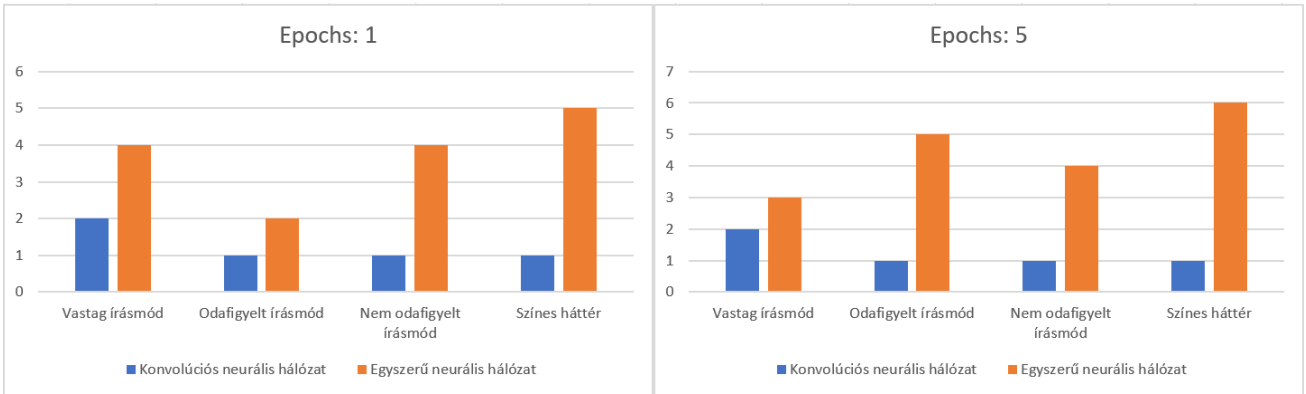
A kézzel, illetve számítógépen írt karakterek tesztjeihez külön programkód készült. A számítógépen írott karakterek tesztjeit a ***test_paint.py*** fájl tartalmazza, míg a kézzel írt karakterek tesztjeit a ***test.py***.

A két program közös vonása, hogy a kód elején betöltenek egy-egy modellt, illetve a statisztikához szükséges változók alapértékeit beállítják. A különbség a feldolgozásban van. A számítógépes képek esetében egy saját algoritmust alkalmazok a kép pixeleinek szétválasztására. Ez a homogén háttérnek köszönhetően könnyen megtehető. A befotózott, kézzel írt karakterek esetében a pixelek heterogénebbek, emiatt ott az OpenCV programcsomag beépített *threshold* függvényét alkalmaztam, mely szétválogatja az összetartozó értékeket.

A kiértékelés mindkét programkód esetében azonos. Miután a szoftver jóslatot tesz, ezt a jóslatot összehasonlít a program a beolvasott fájl első karakterével, mely az abban található karaktert tartalmazza. Az algoritmus a végén kiírja a statisztikai adatokat és ezzel készen van a tesztek kiértékelése.

4.5 Teszteredmények

Elsőként a számok felismerésére szolgáló két modellt teszteltem. Ezek egymáshoz való hasonlítása külön izgalmas, emellett a modellek helyes, vagy épp helytelen működése is megfigyelhető. Elsőként mindkét modell egyszer ment végig a tanító adatokon. Ezt követően megismételtem a tanítást, csak immár ötszöri ismétléssel. A tesztkészletek minden esetben tíz mintát tartalmaznak. Ezek alapján az eredményeket a következő diagramok szemléltetik:

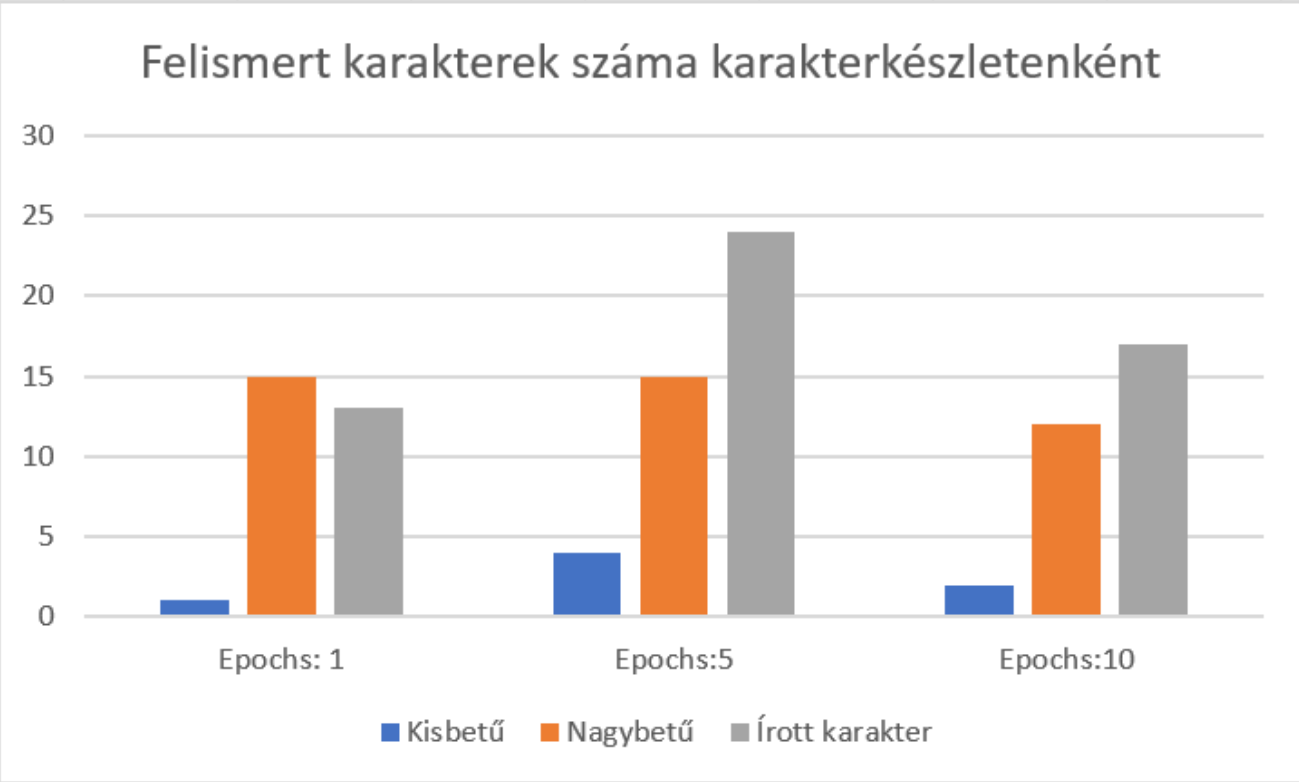


3. ábra: Számok felismerésére szolgáló modell tesztelése
Forrás: saját készítés

Az eredmények kissé meglepően alakultak. Megfigyelhető, hogy némely esetben az egyszerű modell jobban működik bonyolultabb társánál. Összességében a számok nem mutatnak túl jó felismerési

százalékot, ám ez javában köszönhető az írásmódnak. Az egyszerű modell mellett szól pedig, hogy erre egyszerű problémára az egyszerűbb megoldás a jobb. A tanítás során megfigyeltem, hogy a többszöri tanítás már csak alig javít az algoritmusokon, így akár már három betanítás után is elérhető a kívánt pontosság.

A karakterek tesztelése esetén a számítógépes tesztekre 26 karakteres tesztkészleteket hoztam létre, míg a fotózott karakterek esetén összesen 64 karakteren teszteltem a modellt. Elsőként egyszer futtattam végig a tanítást, másodszor pedig ötször. Az eredmények a következőképpen alakultak:



4. ábra: Karakterfelismerő modell teszteredményei
Forrás: saját készítés

Ezeknek a teszteknek az eredményét befolyásolta az is, hogy nem minden betűt képes a betanított modell felismerni. Emellett általánosságban elmondható, hogy az azonos vonású betűket nehezen különbözteti meg a program. Összességében úgy gondolom az eredmények a tesztkészletek függvényében megfelelőek.

4.6 Összegzés

Összességében megállapítható, hogy nem ez a leghatékonyabban működő írásfelismerő szoftver, ám ha megfelelő bemeneti adatokat kap, akkor képes a megfelelő, elvárt kimenetet adni. Ezen állítást támasztja alá, hogy a validáló képek alapján a validált pontosság 95 feletti szinte minden modell esetén. Továbbá ezt a megállapítást támasztja alá az is, hogy amikor teszteltem a grafikus felületet és odafigyeltem a karakter minél szebb írására, akkor egy idő után felismerte az adott karaktert.
A modellek, illetve a szoftver még fejleszthető, azonban szerintem az alapok jól le vannak fektetve. Esetleges jövőbeli javításként elképzelhető lenne saját adatbázis alkalmazása is, ami sokat javíthatna úgy érzem a hatékonyságon, mivel a mostani adatkészlet elég speciális, egyedi írásmódot igényel.

5 Felhasználói leírás

A program használata rendkívül egyszerű. Telepítéséhez nincs másra szükség, mint a Github repository leklónozására. Ezt követően telepíteni kell a dokumentációban szereplő Python programcsomagokat, betanítani a modelleket és használható is.

A programcsomagok telepítésénél érdemes figyelni, hogy a programcsomagok különböző verzió együttműködnek-e egymással. Különös tekintettel érdemes odafigyelni arra, hogy a *tensorflow* programcsomag melyik *numpy* programcsomagot támogatja.

A program grafikus felületének használata rendkívül egyszerű. A felhasználó rajzol az egér, érintőpad segítségével egy karaktert a piros felületre, majd megnyomja a *Tippelés* gombot. Ekkor a program a háttérben megteszi a jóslatát, melynek eredményéről a későbbiekben tájékoztatja is a felhasználót. Ezt követően a *Törlés* gomb segítségével lehetséges a képernyő alaphelyzetbe állítása. Ezek a lépések mindaddig ismételhetőek, amíg a felhasználó be nem zárja a programot.

A program személyre szabásáról egyelőre sajnos csak a programkódban lehet gondoskodni. A modellépítős programkódokban lehetőség van akár a teljes modell újragondolására is. Emellett az egyszerűbb változtatások, mint például a tanítások száma könnyedén, egy-egy érték átírásával módosíthatóak. A grafikus felület személyre szabása szintén programkódból lehetséges.

Amennyiben a felhasználó saját tesztkészletet szeretne készíteni, illetve ezeken teszteket végrehajtani szükség lesz a tesztprogram változtatására. Itt is elengedő a megfelelő sorban a bemeneti fájlok könyvtárát átírni, ezt követően a program a képek listáját már magától legenerálja. A teszteredmények könnyű kiértékelhetőségének érdekében érdemes a fentebb leírtaknak megfelelően kialakítani a könyvtárstruktúrát.

A projekt még számos helyen felhasználóbarátabbá tehető. Az alapok rendelkezésre állnak, innentől úgy gondolom már minden fejlesztés csak szebbé, jobbá teheti a projektet.

6 Jótanácsok a program használatára

Grafikus felület használata esetén célszerű a rendelkezésre álló területet minél jobban kihasználni. Erre azért van szükség, hogy a lekicsinyített kép se legyen túlságosan pici és felismerhetetlen. Saját tesztkészlet készítése esetén érdemes több dologra is odafigyelni. Elsőként talán azt említeném meg, hogy nem érdemes kis méretben elkészíteni az írást, érdemes inkább a program segítségével kicsinyíteni. Emellett amennyiben digitálisan írt karaktereket szeretne a felhasználó tesztelni érdemes közepes méretű vonalvastagságot alkalmazni. Erre azért lehet szükség, mivel túl vékony vonal esetén a kicsinyítés során szinte eltűnik a vonal, túl vastag vonal esetén pedig az egész kép csak egy nagy paca lesz a transzformációk után. Kézzel írt és fotózott karakterek esetén érdemes odafigyelni, hogy a lehető legkevesebb árnyék kerüljön a háttérre; legyen a háttér minél homogénebb. Emellett a többi fent jótanács igaz a fotózott karakterekre is.

7 Felhasznált irodalom

[1] L. Bottou; C. Cortes; J.S. Denker; H. Drucker; I. Guyon; L.D. Jackel; Y. LeCun; U.A. Muller; E. Sackinger; P. Simard; V. Vapnik 1994. Comparison of classifier methods: a case study in handwritten digit recognition

[2] Cheng-Lin Liu, Fe iYin, Da-Han Wang, Qiu-Feng Wang 2013. Online and offline handwritten Chinese character recognition: Benchmarking on new databases 155-162

[3] Charles C. Tappert, Ching Y. Suen, Toru Wakahara (1990.) The State of the Art in On-Line Handwriting Recognition

[4] Fazekas István 2013. Neurális Hálózatok

[5] Gyarmati Péter (2019): Gondolatok a mesterséges intelligencia, gépi tanulás kapcsán. Mesterséges intelligencia – interdiszciplináris folyóirat, I. évf. 2019/1. szám. 31–39.

[6] Hadházi Dániel (2018): Mély konvolúciós neurális hálózatok

[7] MNIST database Retrieved 16. Dec. 2020, from https://en.wikipedia.org/wiki/MNIST_database

[8] EMINST database Retrieved 16. Dec. 2020, from https://www.nist.gov/itl/products-and-services/emnist-dataset

[9] Visual Studio Code weboldala Retrieved 16. Dec. 2020, from https://code.visualstudio.com/

[10] Tensorflow programcsomag Pythonhoz Retrieved 16. Dec. 2020, from https://pypi.org/project/tensorflow/

[11] Numpy programcsomag Retrieved 16. Dec. 2020, from https://numpy.org/

[12] Pandas programcsomag Retrieved 16. Dec. 2020, from https://pandas.pydata.org/

[13] Scikit-learn programcsomag Retrieved 16. Dec. 2020, from https://scikit-learn.org/stable/

[14] Keras programcsomag Retrieved 16. Dec. 2020, from https://keras.io/

[15] OpenCV programcsomag Retrieved 16. Dec. 2020. from https://pypi.org/project/opencv-python/

[16] Pillow programcsomag Retrieved 16. Dec. 2020, from https://pillow.readthedocs.io/en/stable/

[17] Tkinter programcsomag Retrieved 16. Dec. 2020, from https://docs.python.org/3/library/tkinter.html

[18] Adam optimalizáló algoritmus Retrieved 16. Dec. 2020, from https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

[19] ReLU Retrieved 16. Dec. 2020, from https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/