Kézzel írt karakterek felismerése fejlesztési dokumentáció

Tartalom

Bevezetés

A beadandó programom egy olyan modell készítése volt, amely képes felismerni a kézzel írott karaktereket. Egy rövid kutatás után arra jutottam, hogy ehhez a feladathoz konvolúciós neurális hálózatokat fogok használni. A célom az volt, hogy végül képes legyen felismerni a saját kézzel írt karaktereimet.

A megoldáshoz szükséges elméleti háttér rövid ismertetése

NumPy

A nagy mennyiségű adatom hatékony tárolása és feldolgozása érdekében a NumPy könyvtárat használtam. Ennek a könyvtárnak az a szerepe, hogy támogatja a nagy méretű, több dimenziós tömböket. Arra tervezték, hogy gyorsan és hatékonyan lehessen műveleteket végezni nagy méretű tömbökön, ezért C-ben írták, hogy ki tudják használni a nyelv adta teljesítmény előnyeit.

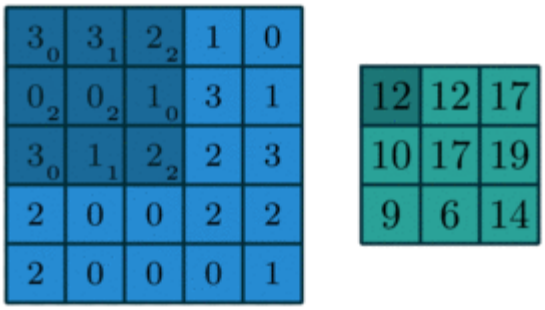
Keras

A Keras egy magas szintű, felhasználóbarát neurális hálózat könyvtár Pythonban írva. Azért használtam a Kerast, mert ezzel volt már némi tapasztalatom és nagyon egyszerűvé teszi a neurális hálók felépítését komolyabb és mélyebb tudás nélkül is.

Neurális hálózat

Mint már említettem a bevezetésben, kutatásom során arra jutottam, hogy ennek a problémának a megoldására egy konvolúciós neurális hálózatot fogok használni. Ezeket elsősorban a képfeldolgozáshoz hozták létre. Ennek a hálózatnak az előnye a képfeldolgozásban az, hogy a képre gyakorlatilag egy szűrőt rak, amit manuálisan

állíthatunk be. Ezeknek a szűrőknek az a szerepe, hogy megkönnyíti például az alakzatok felismerését.



1. ábra

Forrás: https://calvinfeng.gitbook.io/machine-learning-notebook/supervised-learning/convolutional-neural-network/convolution_operation
(2022.12.17)

Ezekben a hálózatokban általában használnak pooling layer-t is (összevonó réteg). Hasonlóan a konvolúcióhoz, ez is egy szűrőt csúsztat végig a képen, de itt a mátrixból a legnagyobb értéket viszi tovább a következő rétegbe, ezzel segítve a zaj kiszűrését.

Ezek után a rétegek után egy Dense réteg következik. Ez tulajdonképpen egy hagyományos neurális réteg. Minden neuronja bemenetként megkapja az előző réteg kimenetét. Az utolsó rétegem is egy Dense layer lesz annyi neuronnal, amennyi osztály közül kell választanom. Kimeneti függvénye a softmax lesz, ennek lényege, hogy a kimeneteket 0 – 1 –es tartományba hozza, a kimenetek összege pedig 1 legyen. Ez gyakorlatilag az én esetemben azt jelenti, hogy a 62 elemű vektorom az előre meghatározott 62 kategória közül melyikbe mennyire tartozik bele.

Dropout réteget a túltanulás kiküszöbölése miatt használtam. Ez a réteg véletlenszerűen az inputokat 0-ra állítja a tanítás minden lépése során.

A kimenet előtt található még egy Flatten réteg. Ennek szerepe, hogy „kihajtogatja” a három dimenziós bemenetemet egy egy dimenziós vektorrá.

OpenCV

Ezen kívül még az OpenCV nevű könyvtárat használtam ahhoz, hogy a saját képeimet olyan formára tudjam hozni, mint a training set-em.

Az OpenCV(Open Source Computer Vision) egy nyílt forráskodú gépi látás és gépi tanulás könyvtár C++ nyelven írva. Ennek a könyvtárnak az előnye, hogy rengeteg kép és videó formátummal képes dolgozni és bonyolult képfeldolgozási algoritmusok elérhetőek benne, használata nem igényel mély tudást.

CUDA

A CUDA egy NVIDIA által fejlesztett modell. Ez gyakran használt a gépi tanulás terén, mert lehetővé teszi, hogy a grafikus kártyánk segítségével párhuzamosan tanítsuk a neurális hálóinkat.

Matplotlib

Végül a képeim megjelenítésére és az eredményeim kiértékelésére a Matplotlib Python könyvtárat használtam. Ez a leggyakrabban használt könyvtár adatok vizualizálására. Magas szintű interfacet kínál a felhasználók számára, így nagyon egyszerűen tudunk vele mindenféle adatelemzéshez szükséges ábrát készíteni.

A megvalósítás terve és kivitelezése

Kutatás kezdete

Első lépésként egy rövid kutatást végeztem, hogy megtudjam merre érdemes elindulni a probléma megoldása érdekében.

Miután ez megtörtént, keresnem kellett nagy mennyiségű adatot amin betaníthattam a hálózatomat. Egy olyan datasetet találtam, amelyben több, mint 60000 kézzel írott

karakter és szám van ASCII kóddal csoportosítva. Kiválogattam belőle a számokat és a kis- és nagybetűket.

2. ábra

Feladat megoldása

Tesztelés miatt 2 fájlt hoztam létre, mindkettő ugyanazon az adathalmazon tanul, csak a cnn1.py training set-je és test-setje egy adathalmazból készül el. A cnn2.py ki van egészítve egy függvénnyel, ami elkészíti a test-set-et a saját képeimből, így anélkül tudom futtatni külön- külön a két megoldást, hogy bele kéne nyúlnom a kódba.

Következő lépésben be kellett olvasnom az adataim. Erre írtam egy függvényt, ami végigmegy mappánként az összes elemen, és ASCII- kód szerint fel is címkézem őket, hogy melyik milyen betű vagy szám.

```
def read_folder(start_code, end_code, dict, train_x, train_y, correction):
    for code in range(start_code, end_code+1):
        files = os.listdir("training_set_full\\" + str(code))
        training_hist[code] = len(files)
        for file in files:
            pic_array = read_picture("training_set_full\\" + str(code) + "\\" + file)
            train_x.append(pic_array)
            train_y.append(code-correction)
```

3. ábra

A függvény úgy működik, hogy külön csoportokban olvasom be a számokat, a kis betűket és a nagy betűket. Meghívom az első elem ASCII-kódjával, az utolsó elem ASCII-kódjával, egy tömbbel amit tanításra használok, egy másik tömbbel, amiben a beolvasott karakterek osztálya szerepel, végül egy korrekciós számmal, aminek segítségével 0-tól 62-ig tudom sorba rendezni az osztályaimat.

```
read_folder(48, 57, training_hist,x_train, y_train, 48)
read_folder(97, 122, training_hist,x_train, y_train, 87)
read_folder(65, 90, training_hist,x_train, y_train, 39)
```

4. ábra

A read_picture függvény beolvassa képeket, majd átkonvertálja szürkeárnyalatossá azokat.

```
def read_picture(picture):
    pic = Image.open(picture).convert('L')
    pix = np.array(pic)
    return pix
```

5. ábra

Most, hogy már el van tárolva az adatom, véletlenszerűen összekeverem őket, hogy amikor elkészítem a training set-et és a test set-et minden karakterből szerepeljen bennük. Az első 50000 adatot tanításra használom, a maradékot tesztelésre.

```
x_train_shuffled,y_train_shuffled = shuffle(x_train,y_train)

x_train = x_train_shuffled[:50000]
x_test = x_train_shuffled[50001:]

y_train = y_train_shuffled[:50000]
y_test = y_train_shuffled[50001:]
```

6. ábra

A következő lépés az adataim újraformázása. Mivel a képeim 64*64 pixelből állnak, ezért az adataim is ilyen formára hozom.

```
x_train = x_train.reshape(x_train.shape[0], 64, 64, 1)
x_test = x_test.reshape(x_test.shape[0], 64, 64, 1)
```

7. ábra

A következő lépésben az adataimhoz tartozó osztályokat kategorizálom. Mivel 62 osztályom van, így 62 kategóriára bontom őket.

```
y_train = keras.utils.to_categorical(y_train, num_classes=62, dtype='int')
y_test = keras.utils.to_categorical(y_test, num_classes=62, dtype='int')
```

8. ábra

Ezek után normalizálom az adataimat, így könnyebben fog tanulni a hálózatom. Mivel a pixelek intenzitás értéke 0-tól 255-ig terjed, így az adataimat elosztom 255-el, hogy az értékek 0-tól 1-ig terjedő skálán legyen.

```
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

x_train /= 255
x_test /= 255
```

9. ábra

Így az adataim készen is állnak, oda lehet már adni a neurális hálónak tanítani.

```
batch_size = 32
num_classes = 62
epochs = 20

model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform',
input_shape=(64, 64, 1)))
model.add(MaxPooling2D((2, 2)))
model.add(Dense(32))
model.add(Dropout(0.5))
model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))

opt = SGD(learning_rate=0.02, momentum=0.5)
model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])

hist = model.fit(x_train,
y_train,batch_size=batch_size,epochs=epochs,verbose=1,validation_data=(x_test, y_test))
```

10. ábra

A cnn2.py két dologban tér el az eddig leírtaktól. Először is, hogy ne bonyolítsam túl a tanítást ezt a hálót csak számokra és kis betűkre használom. Másodszor pedig ehhez készítettem egy függvényt, aminek segítségével el tudom készíteni a test set-emet, ha már feldolgoztam a képeimet előtte.

A create_test függvény segítségével el tudom készíteni a test-set-emet, ha le akarom tesztelni a hálómat saját kezűleg készített képekkel.

```
def create_test(test_x, test_y, test_dict):
    files = os.listdir("test_set3\\")
    for file in files:
        pic = Image.open( "test_set3\\" + file).convert('L')
        pix = np.array(pic)
        print(pix)
        df = pd.DataFrame(pix)
        df.to_csv("pix\\" + file, index=False, header=False)
        test_x.append(pix)

        if  48 <= ord(file[0]) <= 57:
            code = ord(file[0])-48
        elif 97 <= ord(file[0]) <= 122:
            code = ord(file[0]) - 87
        elif 65 <= ord(file[0]) <= 90:
            code = ord(file[0]) - 55

        test_y.append(code)

        if code in test_dict.keys():
            test_dict[code] += 1
        else:
            test_dict[code] = 1
```

11. ábra

Ahhoz, hogy saját képekkel tudjak dolgozni, készítettem egy kódot, ami a tanítóhalmazzal egyenlő formára hozza a saját képeim. Ennek a fájlnak preprocess.py a neve.

```
files = os.listdir("test_set\\")
for file in files:
    img_resized = cv2.imread("test_set\\" + file, cv2.IMREAD_GRAYSCALE)

    se = cv2.getStructuringElement(cv2.MORPH_RECT, (8, 8))
    bg = cv2.morphologyEx(img_resized, cv2.MORPH_DILATE, se)

    out_gray = cv2.divide(img_resized, bg, scale=255)
    img_resized = cv2.threshold(out_gray, 0, 255, cv2.THRESH_OTSU)[1]
    img_resized = cv2.bitwise_not(img_resized)
    img_resized = cv2.cvtColor(img_resized, cv2.COLOR_GRAY2BGR)
    img_resized = cv2.fastNlMeansDenoising(img_resized)
    img_resized = cv2.cvtColor(img_resized, cv2.COLOR_BGR2GRAY)

    result = cv2.imwrite("test_set3\\" + file, img_resized)
```

12. ábra

A script úgy működik, hogy megadjuk neki a mappa nevét amiben szerepelnek a képeink (64*64 pixelnek kell lennie, fekete karakterekkel, fehér háttéren), egyesével végigmegy minden képen, és minden pixelnek az inverzét veszi, így elkészül a fekete hátterű képünk fehér karakterekkel, majd a megadott mappába kimenti mindet.

13. ábra

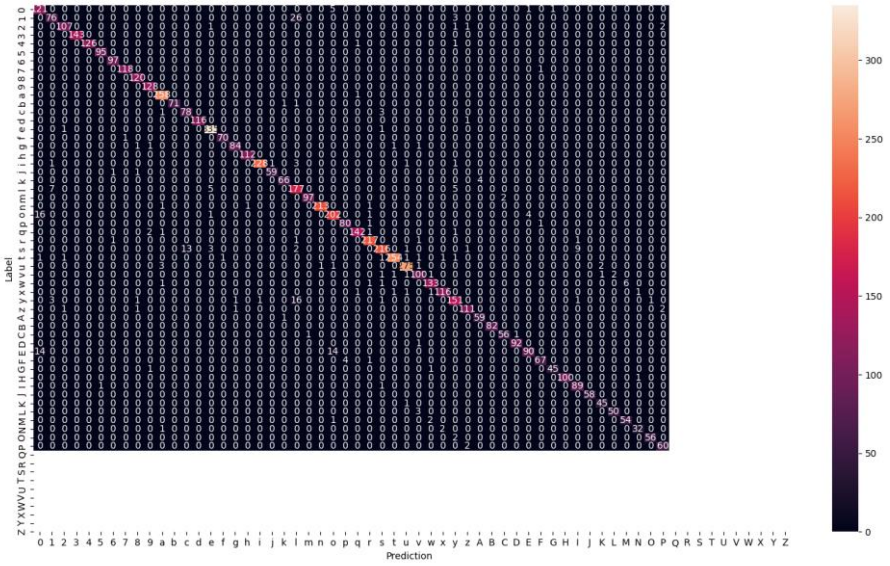
Továbbá fontos, hogy képeink neve az osztály-sorszám.png formátumban kell lennie, mert az algoritmus az első karakter alapján osztályozza be a képeinket. (a1.png, b5.png, d2.png, 41.png, 82.png)

Tesztelés

cnn1 – Az adathalmazból készített test set

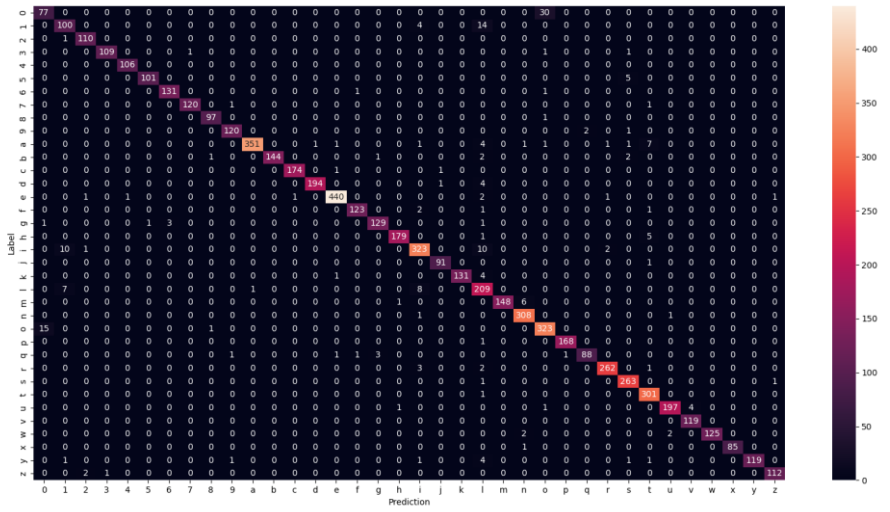
A kiértékelést a cnn1_evaluate.py fájlban lehet lefuttatni.

Abban az esetben, ha a kis- és nagybetűket külön osztálynak tekintem, 30 epochig történő tanítás után 95%-os pontosságot sikerült így elérni.



14. ábra

Ugyan ezekkel a paraméterekkel újra tanítottam a hálózatot, de ebben az esetben a kis-, és nagybetűket egy osztályba vettem, így 96%-os pontosságot sikerült elérni.



15. ábra

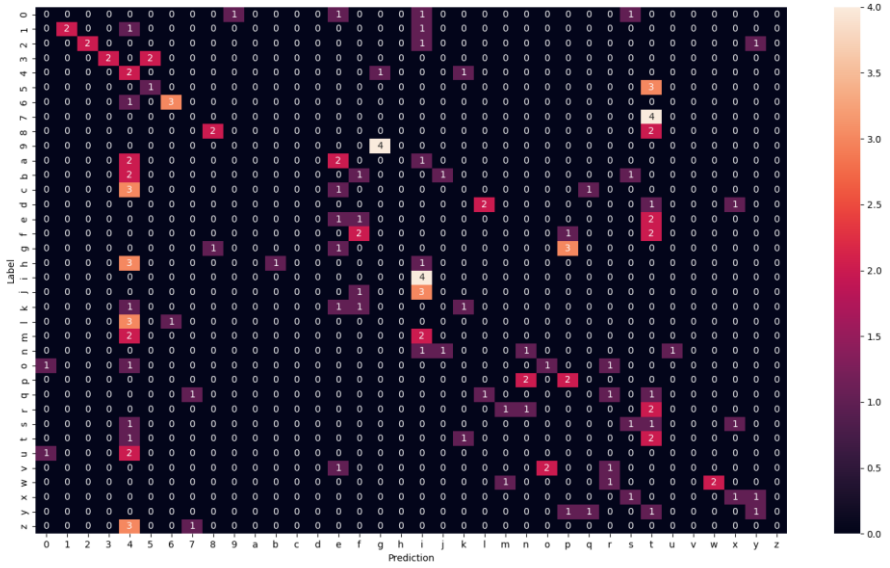
Látható a tévesztési mátrixból, hogy a legtöbb hibát a 0 és az o betű, valamint az 1 és az i betű hibás felismerése okozza. Ebből kifolyólag tovább lehetne növelni a háló pontosságát, ha a 0-át és az o betűt egy osztályba sorolnánk.

cnn2- Saját kézírás

A kiértékelést a cnn2_evaluate.py fájlban lehet lefuttatni.

Az eredményeim a saját képekkel tesztelve elég szegényesek lettek. Legjobb esetben is 30%-os pontosságot tudtam csak elérni.

Próbáltam mélyebb neurális hálókat betanítani, csökkenteni a méretüket, újabb rétegekkel kiegészíteni, illetve elvenni belőle, sajnos nem jártam túl sok sikerrel.



16. ábra

Az látszik ebben a tévesztési mátrixban, hogy bizonyos karaktereket egész jó pontossággal felismer, viszont úgy tűnik, hogy a legtöbbre egy 4-es számjegyet vagy egy t betűt predictel. Valószínűnek tartom, hogy a test halmazom mérete is rossz hatással van az eredményekre.

Úgy gondolom, hogy ezzel a konvolúciós neurális hálózattal nem lesz egyszerű pontosan felismerni a saját kézírásomat.

Ha úgy szeretnénk futtatni a programot, hogy az adathalmazból készített training setet és testet használjuk, a cnn1.py fájlt kell lefuttatni. Ha változtatni szeretnénk azon, hogy a kis- és nagybetűk egy illetve külön osztályba tartozzanak, akkor a program elején létrehozott num_classes nevű változót kell átírni 62, illetve 36-ra és a beolvasásnál használt függvény nagybetűkre vonatkozó részét kell módosítani.

Ennek a modellnek a kiértékelése a cnn1_evaluate.py fájlban található. Ha az előző fájlban módosítottuk a változónkat, akkor ugyan erre az értékre kell átírni ebben a fájlban található kategorizáláshoz használt függvény paraméterét.

Ha saját képeinkkel szeretnénk tesztelni a programot, akkor a cnn2.py fájlt kell futtatni. Ha új képeket szeretnénk használni, akkor a 64*64 pixeles képeinket a test_set nevű mappába kell tárolni, majd a preprocess.py fájlt kell lefuttatni .

Ehhez a modellhez tartozó kiértékelés a cnn2_evaluate.py nevű fájlban található.

Ábrajegyzék

Irodalomjegyzék

https://numpy.org/doc/stable/ (2022.12.17)

https://keras.io/about/ (2022.12.17)

https://opencv.org/about/ (2022.12.17)

https://docs.nvidia.com/deeplearning/cudnn/developer-guide/index.html (2022.12.17)

https://github.com/sueiras/handwritting_characters_database (2022.12.17)