Dokumentáció

Bevezetés

A megoldandó feladat egy olyan program elkészítése, amely képes felismerni előre meghatározott kézzel írt karaktereket homogén háttér előtt. Ilyen karakterek az angol ábécé betűi, illetve a számok.

Terv

A feladat megoldásához a NIST Special Database 19 adta az alapot. Ebben

az adatbázisban megtalálható több százezer kézzel írt karakter, amiket felhasználhatunk a feladat teszteléséhez és megoldásához egyaránt. A megoldáshoz egy egyszerű neurális hálózatot használunk. Ehhez szükséges számunkra a felismerni kívánt karakterekből osztályok készítése és egy közös fájlba foglalása, melyet a későbbiekben újra fel lehet használni. Az így elkészített fájlunkat pedig felhasználjuk egy külön programban adatok felismeréséhez.

Megvalósítás

A probléma megoldását elsősorban egy neurális hálózat létrehozásával és az adatok megfelelő csoportosításával kell kezdeni. Az adat halmazunkat, amit a NIST adatbázisból megszereztünk rendeznünk kell. Az adatok az ASCII tábla szerint hexadecimális számuk alapján vannak mappákba rendezve, ez a rendezési módszer sajnos nem veszi egymás után a betűket és számokat, ezért ezt rendeznünk kell úgy, hogy a számok, a kisebtűk és a nagybetűk egymás után legyenek. Az adatok sorrendje fontos, erre a későbbiekben kitérünk miért.
Az adataink tehát csoportosítva vannak így következhet a neurális hálózat

megalkotása, ehhez a Keras nyíltforráskódú szoftverkönyvtárat használjuk, ez megfelelő alapot biztosít számunkra a neurális hálózat létrehozásához. A Models API-t felhasználva egy egyszerű layerek sorozatából álló modell kerül kialakításra a feladathoz. Ennél bonyolultabb modellre az alapfeladat megoldásánál nincsen szükségünk. Az adatok beolvasása, ezután felbontása a modell betanításához és validálásához, végül a betanított modellünk mentése a cél. Ezután ezt egy másik programba betöltve tesztelni tudjuk.
A feladat megoldásához így két programot készítünk. Az egyikkel létrehozzuk és betanítjuk a modellünket, a másikkal pedig a betanított modell tesztelését végezzük el. A modellt létrehozó programot seq_mod.py-nak nevezzük el. Az adatok beolvasásához a tf.keras.utils.image_dataset_from_directory-t használjuk. Ez a metódus egyszerűen beolvassa és osztályozza a képeket, ezért fontos a sorrend amikor az adathalmazunkat elrendezzük. Ez a metódus beolvassa az előre kijelölt elérési úton lévő adatokat, ehhez az alábbi formában szükséges szerepelnie az adatoknak és a main_directory elérési útját kell megadni, ezt a DATA_DIR-ben tároljuk.
main_directory/
...class_a/
......a_image_1.jpg
......a_image_2.jpg
...class_b/
......b_image_1.jpg
......b_image_2.jpg
A metódus az os.walk alapján megy végig a megadott könyvtáron és annak megfelelő sorrendben gyűjti össze az adatokat. Az így összegyűjtött adatokat a metódussal össze is lehet kevertetni, illetve fel is lehet osztani. A class_a illetve class_b a megfelelő képekhez rendelt értékek lesznek. Emiatt fontos a sorrendiség, mert ha össze van keverve az adatkészletünk, akkor a felismerés folyamán amikor már kész a modellünk fontos lesz megállapítani azt, hogy a visszaadott értékünk melyik betanított osztályhoz tartozik. Ezután létrehozzuk a Sequential model-t, amihez hozzáadjuk a szükséges layereket. Elsőként egy Rescaling layert adunk a modellhez, ami a kapott adatot minden színcsatornán 0 és 1 közé kicsinyíti, így egyszerűbben feldolgozhatjuk a képeinket, ezután egy Conv2D layer kap a modellünk, amiben meghatározzuk a bemeneti kép méretét is. Az ehhez használt

méretet az IMG_WIDTH és az IMG_HEIGHT-ben határozzuk meg. A Conv2D réteg egy konvolúciós kernelt hoz létre, amelyet a réteg bemenetével konvolálnak, hogy a kimenetek tenzorát állítsák elő. A következő rétegünk lemintázza a bemenetet annak térbeli méretei (magasság és szélesség) mentén úgy, hogy a bemenet minden egyes csatornájához a maximális értéket veszi át egy bemeneti ablakon. Ezeket a layereket megismételjük egymás után többször, majd a Flatten() réteggel kisimítjuk a bemenetet. A Dropout réteg véletlenszerűen 0-ra állítja a bemeneti egységeket az edzési idő minden lépésében x gyakorisággal, ami segít megelőzni a túlillesztést. Ezután a végső réteg a Dense ami egy sűrűn kapcsolt neurális hálózat réteg. Ez adja a kimenetet az osztályok alapján, amit a NUM_CLASSES-ban tárolunk. Ezután egy optimalizáló segítségével összeszerkesztjük a modellünket. Ehhez az Adam-et használjuk, ami egy sztochasztikus gradiens süllyedési módszer, amely az első és másodrendű momentumok adaptív becslésén alapul. Ezután a fit() és a beolvasott adatkészlet segítségével betanítjuk a modellt meghatározott számú epoch-al (adatkészlet iteráció). Ezek mentést végzünk a modellről egy .h5 típusú fájlba ezek után kiíratjuk a pontosságát és a betanítás közbeni iterációk egyéb adatait.

seq_mod.py forráskódja:

```
import keras
from keras import layers
import tensorflow as tf
from tensorflow.keras.optimizers import Adam

IMG_WIDTH=24
IMG_HEIGHT=24
DATA_DIR=r"path to the dataset directory"
NUM_CLASSES = 62

trainset=tf.keras.utils.image_dataset_from_directory(
    DATA_DIR, labels="inferred",
    label_mode="categorical", class_names=None,
    color_mode="grayscale", batch_size=64,
    image_size=(IMG_WIDTH, IMG_HEIGHT),
    shuffle=True, seed=3, validation_split=0.2, subset="training",
    interpolation="bilinear", follow_links=False,
    crop_to_aspect_ratio=False
)
valset=tf.keras.utils.image_dataset_from_directory(
    DATA_DIR, labels="inferred", label_mode="categorical",
    class_names=None, color_mode="grayscale",
    batch_size=64, image_size=(IMG_WIDTH, IMG_HEIGHT),
    shuffle=True, seed=3, validation_split=0.2, subset="validation",
    interpolation="bilinear", follow_links=False,
crop_to_aspect_ratio=False
)
```

A szükséges könyvtárak importálása

Főbb konstansok kiemelése

Adatok beolvasása és felbontása a rendezett könyvtárstruktúrából

```
model = keras.Sequential(
    [
        keras.Input(shape=(IMG_HEIGHT,IMG_WIDTH,1)),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(NUM_CLASSES, activation="softmax"),
    ]
)
model.compile(optimizer = Adam(learning_rate=0.001),
loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(trainset, batch_size=64, epochs=3,
                            validation_data = valset)
model.summary()
score=model.evaluate(valset,verbose=0)
print(score)
model.save(r'modelname.h5')
print("The validation accuracy is :", history.history['val_accuracy'])
print("The training accuracy is :", history.history['accuracy'])
print("The validation loss is :", history.history['val_loss'])
print("The training loss is :", history.history['loss'])
```

Modell és a rétegek létrehozása

Modell összeállítása, betanítása és mentése

Adatok kiírása a képernyőre

Az elmentett modellünket ezekután betöltjük a másik programunkba, amiben tesztelni tudjuk a pontosságát és a működő képességét. Ebben a programban a legfontosabb dolgunk a szótár létrehozása, amely meghatározza a megtippelt értékhez tartozó osztályt. Ezt a programot Command Prompt-ból/PowerShell-ből kell elindítani, mivel át kell adni a programnak egy bemeneti fájlhoz tartozó elérési utat. Ezután a kapott képet előfeldolgozzuk, majd pedig átadjuk a modellünknek, hogy megtippelhesse milyen karakter található a képen. A modell egy vektort ad vissza, amiben 0-1 közötti értékek szerepelnek ebből kiválasztva a legnagyobbat kapjuk meg azt, melyik az amire a modell szerint a legjobban hasonlít a képen látható karakter.
Alphabet_recogn.py forráskódja:

```
import cv2
import numpy as np
from keras.models import load_model
import sys,getopt
```

A szükséges könyvtárak importálása

```
argv=sys.argv[1:]
inputfile=''
try:
   opts, args = getopt.getopt(argv,"hi:",["infile="])
except getopt.GetoptError:
   print('Alphabet_recogn.py -i <inputfile>')
   sys.exit(2)
for opt, arg in opts:
   if opt == '-h':
      print('Alphabet_recogn.py -i <inputfile>')
      sys.exit()
   elif opt in ("-i", "--infile"):
      inputfile = arg
```

Command Prompt/Powershell-ről indított program argumentum ellenőrzése

```
model = load_model('modelname.h5')
model.summary()
alph_dict = {0:'0',1:'1',2:'2',3:'3',4:'4',5:'5',6:'6',7:'7',8:'8',9:'9',
          10:'a',11:'b',12:'c',13:'d',14:'e',15:'f',16:'g',17:'h',18:'i',19:'j',
          20:'k',21:'l',22:'m',23:'n',24:'o',25:'p',26:'q',27:'r',28:'s',29:'t',
          30:'u',31:'v',32:'w',33:'x',34:'y',35:'z',36:'A',37:'B',38:'C',39:'D',
          40:'E',41:'F',42:'G',43:'H',44:'I',45:'J',46:'K',47:'L',48:'M',49:'N',
          50:'O',51:'P',52:'Q',53:'R',54:'S',55:'T',
          56:'U',57:'V',58:'W',59:'X',60:'Y',61:'Z'}
```

Modell betöltése, és a hozzá tartozó szótár létrehozása

```
img= cv2.imread(inputfile)
img_copy =img.copy()
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img = cv2.resize(img, (400,400))
img_copy = cv2.GaussianBlur(img_copy, (3,3), 0)
img_gray = cv2.cvtColor(img_copy, cv2.COLOR_BGR2GRAY)
_, img_thresh = cv2.threshold(img_gray, 100, 255,
                              cv2.THRESH_BINARY_INV)
img_final = cv2.resize(img_thresh, (24,24))
img_final =np.reshape(img_final, (1,24,24,1))
img_pred = alph_dict[np.argmax(model.predict(img_final))]
tipp=np.argmax(model.predict(img_final))
print(tipp)
print('Number of arguments: {}'.format(len(sys.argv)))
print('Argument(s) passed: {}'.format(str(sys.argv)))
```

Kép betöltése argumentumból, előfeldolgozás és tipp meghatározása

```
cv2.putText(img, "Tipp: " + img_pred, (20,370),
        cv2.FONT_HERSHEY_TRIPLEX, 1.3, color = (255,0,0))
cv2.imshow('Handwritten character recognition', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

A megjósolt tippünk ráhelyezése a képre, ablak bezárása gombnyomásra

Tesztelés

A tesztelést a második programunkkal végezzük el, ehhez felhasználhatunk adatokat a NIST adatbázisból is, de saját magunk is készíthetünk teszt adatokat, illetve az internetről is lehet szerezni egyéb teszt adatokat. A modell pontosságát már a betanítás folyamán is kijelzi a programunk így, ezzel előre meghatározható mekkora a pontossága. Azonban elengedhetetlen a tesztelés utólag is, hogy kiderüljön miben kéne még fejlődnie a modellnek.

Felhasználói leírás

seq_mod.py

Függőségek:

Python 3.9.7 futtatására alkalmas operációs rendszer. TensorFlow, Keras telepítése szükséges a program működéséhez.

Program indítása:

Command Prompt/PowerShell segítségével a program indítása

python [program neve]

Egyszerűség kedvéért a programkódban kell megváltoztatni a betanítandó adatok helyét, nem paraméteres az átadás. (DATA_DIR). Ugyan ez vonatkozik az osztályok számára (NUM_CLASSES), valamint a képek méretére is. (IMG_WIDTH, IMG_HEIGHT)

Program kimenete

A kimeneti fájl egy .h5 típusú fájl lesz, aminek a nevét a model.save() metódusban kell megváltoztatni, illetve átnevezni. A program a futása közben mutatja az előrehaladását. Miután végzett a program a tanulással, egy összegzést készít, amelyet alább láthatunk.

Mintafutás:

```
Found 731668 files belonging to 62 classes.
Using 585335 files for training.
Found 731668 files belonging to 62 classes.
Using 146333 files for validation.
Epoch 1/3
9146/9146 [==============================] - 1226s 134ms/step - loss: 1.8480 - accuracy: 0.5169 - val_loss: 0.7374 - val_accuracy: 0.7737
Epoch 2/3
9146/9146 [==============================] - 1226s 134ms/step - loss: 0.9245 - accuracy: 0.7236 - val_loss: 0.6629 - val_accuracy: 0.7925
Epoch 3/3
9146/9146 [==============================] - 1234s 135ms/step - loss: 0.8366 - accuracy: 0.7468 - val_loss: 0.6291 - val_accuracy: 0.8022
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 22, 22, 32)        320

 max_pooling2d (MaxPooling2D  (None, 11, 11, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 9, 9, 64)          18496

 max_pooling2d_1 (MaxPooling  (None, 4, 4, 64)         0
 2D)

 flatten (Flatten)           (None, 1024)              0

 dropout (Dropout)           (None, 1024)              0

 dense (Dense)               (None, 62)                63550

=================================================================
Total params: 82,366
Trainable params: 82,366
Non-trainable params: 0
_________________________________________________________________
```

A program lefutása közben és után kiírt adatok

Alphabet_recogn.py

Függőségek:

Python 3.9.7 futtatására alkalmas operációs rendszer. OpenCV, NumPy, Keras telepítése szükséges a program működéséhez

Program indítása

Command Prompt/PowerShell segítségével a program indítása

python [program neve] -i [elérési útvonal]

alternatív

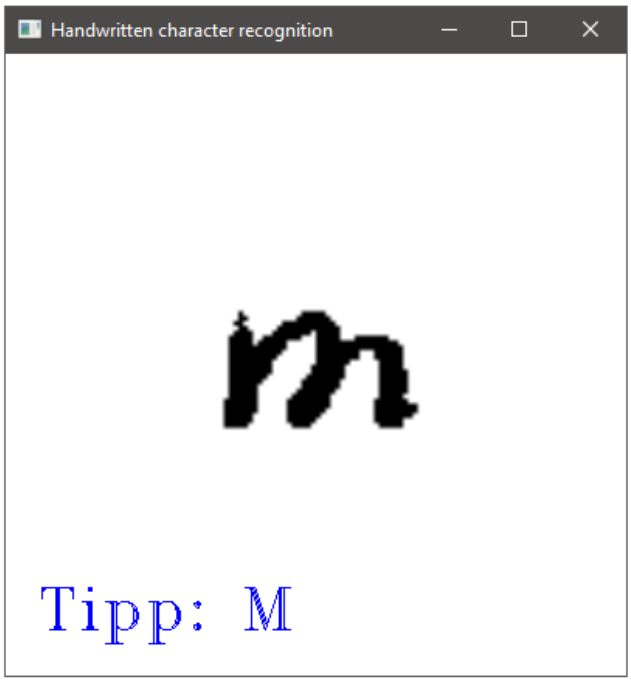
python [program neve] --input [elérési útvonal]

-h paraméterrel indítva a program megmondja a megfelelő bemeneti szintaxist a képernyőre íratva.

Program kimenete

Megnyílik a bemeneti kép fájl és azon színes felirattal a tipp, hogy melyik betű/szám található a képen.

Mintafutás:



A megnyitott kép és a tippelt adat

```
48
Number of arguments: 3
Argument(s) passed: ['Alphabet_recogn.py', '-i', 'D:\\           \\           \\       \\nagy M\\train_4d_00011.png']
```

A képernyőre kiírt adatok

Irodalomjegyzék

- https://keras.io/
- https://www.tensorflow.org/