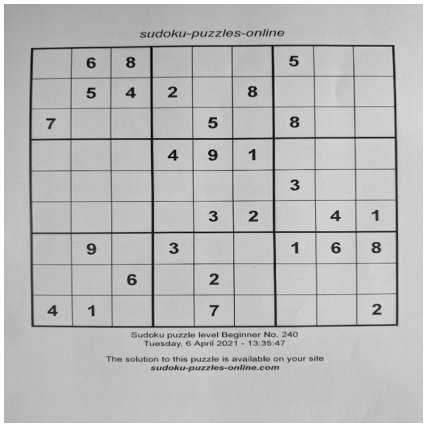
Nume: Sociu Daniel
Grupa: 342

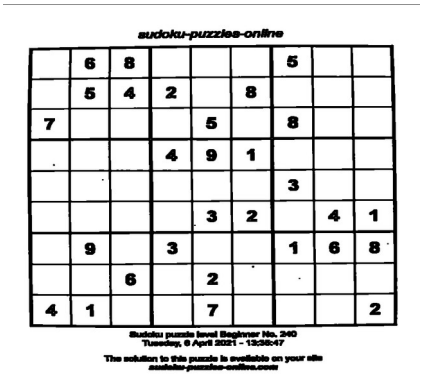
Documentatie Sudoku Image Recognition

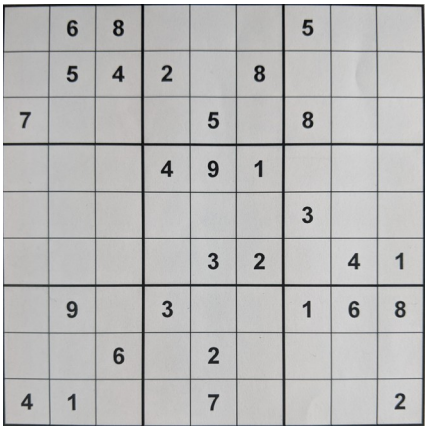
Acest proiect primeste ca input niste imagini cu jocul sudoku, clasic si jigsaw, si returneaza o mapa virtuala a hartii sudoku.

In primul rand dupa ce obtin imaginile, pentru ambele cazuri obtin zona patratica care contine doar jocul de sudoku, ignorand restul imaginii.

De exemplu imaginea initiala este transformata astfel:







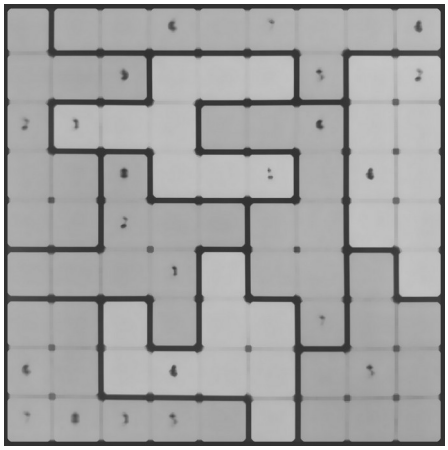
Initial dau sharpen la imagine si aplic un threshold, iar dupa se pot obtine contururile din imagine destul de usor si astfel cu un algoritm obtin colturile careului sudoku. In final se aplica un crop si resize.

Pentru imaginea de tip clasic obtin patch-urile (cele mai mici patratele) impartind imaginea aproximativ in 9 pe vertical si orizontal, adica simulez teoretic cam unde ar fi liniile din imagine. Dupa aceasta iau fiecare patch si verific daca in acesta exista o cifra (aplic un threshold binar, iar dupa fac o medie de continut al pixerilor din patch, iar daca acesta este peste un anumit threshold, consider ca contine o cifra).

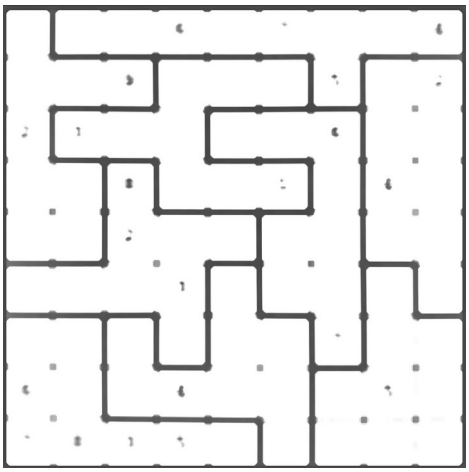
Prin verificarea fiecarui patch, daca continue sau nu o cifra, obtinem mapa virtuala care reprezinta rezolvarea task-ului 1 fara bonus.

Pentru imaginea de tip jigsaw, pentru detectarea cifrelor aplic aceeasi pasi, doar ca tipul acesta aduce o noua problema, si anume detectare zonelor corespunzatoare. Desi avem doua tipuri de imagini jigsaw, colore si grayscale, eu am folosit acelasi algoritm pentru imagini grayscale la ambele (am convertit cele colore la grayscale)

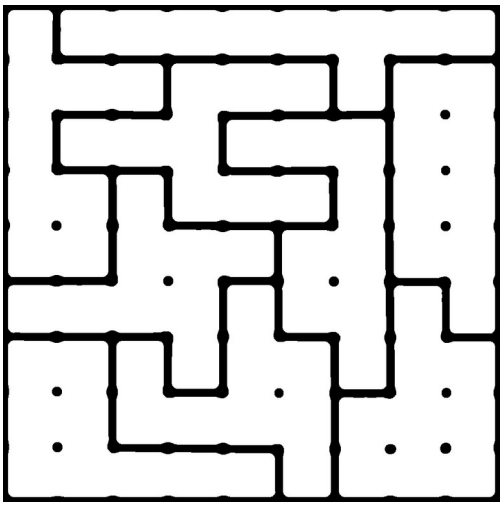
In primul rand obtin liniile care definesc zonele diferite ale imaginii, deoarece liniise care separa zonele sunt mai groase ca liniile normale, daca aplic un simplu medianBlur, obtin urmatoarea imagine:



Dupa asta scad din aceasta un gaussianBlur pentru a accentua imaginea si obtin urmatoarea imagine:



Iar daca pe aceasta imagine aplicam un threshold (eu intai am decis sa mai aplic un medianBlur pentru a mai scapa din puntele random):



Acum avand aceasta imagine, trebuie sa decidem zonele. Pentru a reduce complexitatea timp, am redus aceasta imagine de la 1000x1000 la 100x100, doar pentru partea de obtinere a zonelor si a deciderii acesteia avand un patch dat (adica interogarea).

Avand o linie neagra bine definita care separa zonele, as putea sa fac un algoritm de fill pe baza acestei proprietati:

In primul rand definesc o mapa la fel de mare ca imaginea care ia urmatoarele valori: -1 (255 pentru ca folosesc uint8) pentru culoarea negra, iar fiecare zona de alb din imagine ia valoarea zonei respective care este intre 1 si 9. Pentru a obtine asta parcurg pe linii, de la stanga la dreapta matricea mea, iar cand gaseste o zona fara o valoare atribuita si este alba in imagine, fac fill (peste tot pe unde e alb in imagine) pe mapa mea cu indicele mapei corespunzator, si continui pe linii pana cand ajung la final.

Iar dupa pe fiecare patch fac o interogare despre zona, bazat pe mapa descrisa mai sus. Pe scurt fac un frequency de valori de la 1 la 9, si dupa un argmax pe frecventa, care ne da indicele zonei. Adaugand acest detaliu la raspuns obtinem rezolvarea pentru jigsaw fara punct bonus.

Puntru digit recognition folosesc fisierul digit_recognition.ipynb pentru a antrena un CNN model. Ca imput am generat niste imagini cu fonturi din fonts/ .

```
def get_digits(number):
    digitList = ['1','2','3','4','5','6','7','8','9']
    fonts = []
    digits = []
    digit_images = []

    for font in glob.glob("fonts/*"): # Create Image for every font
        fonts.append(os.path.abspath(font))

    for _ in range(number):
        rand_digit = random.randint(0, len(digitList) - 1)
        rand_font = random.randint(0, len(fonts) - 1)
        font = ImageFont.truetype(fonts[rand_font], random.randint(40, 50))
        width, height = font.getsize(digitList[rand_digit])
        img_width, img_height = 64, 64
        img = Image.new('L', (img_width, img_height), color='white')
        d = ImageDraw.Draw(img)
        d.text(((img_width - width) / 2, (img_height - height) / 2), digitList[rand_digit], font=font, fill=0, align="center")
        np_img = np.array(img)
        digits.append(digitList[rand_digit])
        digit_images.append(np_img)
    return np.array(digit_images), np.array(digits)

X, y = get_digits(10000)
```

Am folosit un CNN destul de simplu, initial a fost un leNet-5 si dupa am mai adaugat niste layere

```
neural_model = Sequential([
    layers.Input(shape=(64,64,1)),
    layers.Conv2D(32, kernel_size=5, padding="same", activation="relu"),
    layers.MaxPooling2D(pool_size=2, strides=2),
    layers.Dropout(0.2),
    layers.Conv2D(64, kernel_size=3, activation="relu"),
    layers.MaxPooling2D(pool_size=2, strides=2),
    layers.Dropout(0.3),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(64, activation='relu'),
    layers.Dense(9, activation="softmax")
])
```

Pentru a avea o diversitate mai mare de imagini, am folosit un image generator cu mici modificari asupra imaginilor:

```
datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.05,
    zoom_range=0.1,
    fill_mode='nearest'
)
datagen.fit(train_data_CNN)

#%%

result = neural_model.fit(
    datagen.flow(train_data_CNN, train_label_CNN, batch_size=64),
    validation_data=(test_data_CNN, test_label_CNN),
    steps_per_epoch= len(train_data_CNN) // 64,
    epochs = 5,
    #shuffle=True,
    #verbose=False,
    use_multiprocessing=True,
    # callbacks=[early_stopping]
)
```

Am exportat acest model in digit_CNN/digit_cnn.h5

Folosind acest model am prezis digit-urile din patch-uri si le-am dat replace la respectivele x-uri din fisierele fara digit-recognition.