

## 8. MELLÉKLET

### *8.1. Dolgozatban felhasznált programok forráskódja*

A melléklet ezen fejezetében megtekinthető az által készített algoritmusok Python programozási nyelvben készített forráskódja. A kód fejlesztése során a Jupyter Notebook szoftver környezetét, Python 3.10.4-es verzióját alkalmaztam, illetve a programban is megtekinthető nyílt forráskóddal rendelkező egyéb könyvtárakat (pl. NumPy, OpenCV, pandas, torch stb.).

### *8.2. Manuális módszerrel készült képek előkészítése*

```
from PIL import Image, ImageChops

img1 = Image.open(r'C:\cmg\MISORJA1\manual fibre detection_original\
JPEG\SORJAPRED3_manual_output_1.jpg')

img1.size

left=41
right=img1.size[0]-42
top=503
bottom=img1.size[1]-503

im1 = img1.crop((left, top, right, bottom))

im1.size

import os

# folder path
DIR_PATH = r'C:\cmg\MISORJA1\manual fibre detection_original\JPEG'

# list to store files
PATHS=[]

# Iterate directory
for path in os.listdir(DIR_PATH):
    # check if current path is a file
    if os.path.isfile(os.path.join(DIR_PATH, path)):
        PATHS.append(DIR_PATH+r'\\'+path)

print('Number of files in dictionary: ',len(PATHS),'\n\nFirst files:
\n',PATHS[:5])
```

```

# new_dir=r'C:\cmg\MISORJA1\manual_fibre_detection_resized'

# for i in range(len(PATHS)):
#     img1 = Image.open(PATHS[i])
#     im1 = img1.crop((left, top, right, bottom))

#     im1.save(new_dir+PATHS[i][PATHS[i].rfind('\\'):])

```

### 8.3. Vizsgálathoz használt képek beolvasása megfelelő méretben

```

import os

manual_dir=r'C:\cmg\MISORJA1\manual_fibre_detection_resized'

# list to store files
MANUAL_PATHS=[]

# Iterate directory
for path in os.listdir(manual_dir):
    # check if current path is a file
    if os.path.isfile(os.path.join(manual_dir, path)):
        MANUAL_PATHS.append(DIR_PATH+r'\\'+path)

print('Number of files in dictionary: ',len(MANUAL_PATHS),'\n\nFirst
files:\n',MANUAL_PATHS[:5])

#read direactory with test files

test_dir=r'C:\cmg\SORJAPRED3\input for tests'

TEST_NAMES=[]

# Iterate directory
for path in os.listdir(test_dir):
    # check if current path is a file
    if os.path.isfile(os.path.join(test_dir, path)):
        TEST_NAMES.append(path)

print('Number of names in dictionary: ',len(TEST_NAMES),'\n\nFirst f
iles:\n',TEST_NAMES[:5])

from natsort import os_sorted
os_sorted(TEST_NAMES)

TEST_NAMES

#get captured images and save them

image_dir=r'C:\cmg\SORJAPRED3\04_gyuritok adatok frisebb'

```

```

REAL_IMAGES=[]

TEST_NAMES[0][:7]

for i in range(len(TEST_NAMES)):
    REAL_IMAGES.append(image_dir + '\\'+ TEST_NAMES[i][:7]+'.jpg')

print(REAL_IMAGES[:5])

import cv2

real_images_for_test_dir=r'C:\cmg\MISORJA1\real_images_for_test'

REAL_IMAGES=[]

# Iterate directory
for path in os.listdir(real_images_for_test_dir):
    # check if current path is a file
    if os.path.isfile(os.path.join(real_images_for_test_dir, path)):
        REAL_IMAGES.append(real_images_for_test_dir+'\\'+path)

from natsort import os_sorted
REAL_IMAGES=os_sorted(REAL_IMAGES)

REAL_IMAGES[:5]

```

#### 8.4. Képek létrehozása a DIP algoritmus segítségével

```

import cv2
import numpy as np
import imutils
import numpy as np
from PIL import Image as im
from PIL import ImageFilter

def is_contour_bad(c):
    # approximate the contour
    peri = cv2.arcLength(c, True)
    return not peri < 100

def DIP_method(file,save_1,save_2,save_3,save_4):
    img = cv2.imread(file)

    top = int(0.01 * img.shape[0]) # shape[0] = rows
    bottom = top
    left = int(0.01 * img.shape[1]) # shape[1] = cols
    right = left

```

```

borderType = cv2.BORDER_CONSTANT
value = [0,0,0]

th, im_th = cv2.threshold(img, int(np.mean(img)/20 + (np.std(img)
) * 3)), 255, cv2.THRESH_BINARY)
blur = cv2.GaussianBlur(im_th,(13,13),0)
thresh = cv2.threshold(blur, 50, 255, cv2.THRESH_BINARY)[1]
edges = cv2.Canny(thresh, threshold1=0, threshold2=0,L2gradient=
True)
dst = cv2.copyMakeBorder(thresh, top, bottom, left, right, borde
rType, None, value)
img_border = cv2.copyMakeBorder(img, top, bottom, left, right, b
orderType, None, value)
edges2 = cv2.Canny(dst, threshold1=1000, threshold2=200,L2gradie
nt=True)
cnts = cv2.findContours(edges2.copy(), cv2.RETR_LIST, cv2.CHAIN_
APPROX_NONE)
cnts = imutils.grab_contours(cnts)
mask = np.ones(dst.shape[:2], dtype="uint8") * 255
cv2.drawContours(mask, cnts, -1, 0, -1)
cnts = cv2.findContours(edges2.copy(), cv2.RETR_LIST, cv2.CHAIN_
APPROX_NONE)
cnts = imutils.grab_contours(cnts)
mask = np.ones(dst.shape[:2], dtype="uint8") * 255
# loop over the contours
for c in cnts:
    # if the contour is bad, draw it on the mask
    if float(cv2.contourArea(cv2.convexHull(c))) !=0:
        if cv2.contourArea(c) / float(cv2.contourArea(cv2.convex
Hull(c)))<0.8 and cv2.arclength(c, True)>20:
            cv2.drawContours(mask, [c], -1, 1, 2)
cnts_2 = cv2.findContours(mask.copy(), cv2.RETR_LIST, cv2.CHAIN_
APPROX_NONE)
cnts_2 = imutils.grab_contours(cnts_2)
mask_4 = im.fromarray(mask)
mask_4 = mask_4.filter(ImageFilter.ModeFilter(size=5))

mask_5=np.asarray(mask_4)

edges_5=cv2.Canny(mask_5, threshold1=0, threshold2=200,L2gradien
t=True)
cnts_5 = cv2.findContours(edges_5.copy(), cv2.RETR_LIST, cv2.CHA
IN_APPROX_NONE)
cnts_5 = imutils.grab_contours(cnts_5)
mask_6 = np.ones(dst.shape[:2], dtype="uint8") * 255

for c in cnts_5:
    cv2.drawContours(mask_6, [c], -1, 1, -1)

```

```

    contours = cv2.findContours(edges_5.copy(), cv2.RETR_LIST, cv2.C
HAIN_APPROX_SIMPLE)
    contours = imutils.grab_contours(contours)

    # Find the convex hull object for each contour
    hull_list = []
    for i in range(len(contours)):

        hull = cv2.convexHull(contours[i])
        hull_list.append(hull)

    hull_img = np.ones(mask.shape[:2], dtype="uint8") * 255
    for i in range(len(contours)):
        # cv2.drawContours(hull_img, contours, i, (0,0,0))
        if is_contour_bad(contours[i]):
            cv2.drawContours(hull_img, hull_list, i, (0,0,0), -1)

    new_img = cv2.addWeighted(img_border, 1, cv2.cvtColor(hull_img,
cv2.COLOR_GRAY2RGB), 1, 0)

    cv2.imwrite(save_1, im_th)

    cv2.imwrite(save_2, thresh)

    cv2.imwrite(save_3, mask_6)

    cv2.imwrite(save_4, hull_img)

for i in range(len(REAL_IMAGES)):
    DIP_method(REAL_IMAGES[i],
                save_dir_1+REAL_IMAGES[i][REAL_IMAGES[i].rfind('\\'):
]+'.jpg',
                save_dir_2+REAL_IMAGES[i][REAL_IMAGES[i].rfind('\\'):
]+'.jpg',
                save_dir_3+REAL_IMAGES[i][REAL_IMAGES[i].rfind('\\'):
]+'.jpg',
                save_dir_4+REAL_IMAGES[i][REAL_IMAGES[i].rfind('\\'):
]+'.jpg'
                )

manual=cv2.imread(r'C:\cmg\MISORJA1\manual_fibre_detection_resized\S
ORJAPRED3_manual_output_1.jpg', cv2.IMREAD_GRAYSCALE)
dip=cv2.imread(r'C:\cmg\MISORJA1\DIP3_images\11_elott_KI_0.jpg.jpg',
cv2.IMREAD_GRAYSCALE)

manual = cv2.resize(manual, (dip.shape[1],dip.shape[0]), interpolati
on = cv2.INTER_AREA)

```

```

#cv2.imshow('overlay', manual)
#cv2.imshow('overlay', dip)

#cv2.waitKey(0)

def DIP_extra(file, save_1, save_2, save_3, save_4):
    img = cv2.imread(file)

    top = int(0.01 * img.shape[0]) # shape[0] = rows
    bottom = top
    left = int(0.01 * img.shape[1]) # shape[1] = cols
    right = left
    borderType = cv2.BORDER_CONSTANT
    value = [0,0,0]

    best=0
    best_found=0
    best_score=-10000000
    for num in range(25):

        th, im_th = cv2.threshold(img, 2*num+60, 255, cv2.THRESH_BIN
ARY)
        blur = cv2.GaussianBlur(im_th, (13,13), 0)
        thresh = cv2.threshold(blur, 50, 255, cv2.THRESH_BINARY)[1]
        edges = cv2.Canny(thresh, threshold1=0, threshold2=0, L2gradi
ent=True)
        dst = cv2.copyMakeBorder(thresh, top, bottom, left, right, b
orderType, None, value)
        img_border = cv2.copyMakeBorder(img, top, bottom, left, righ
t, borderType, None, value)
        edges2 = cv2.Canny(dst, threshold1=1000, threshold2=200, L2gr
adient=True)
        cnts = cv2.findContours(edges2.copy(), cv2.RETR_LIST, cv2.CH
AIN_APPROX_NONE)
        cnts = imutils.grab_contours(cnts)
        mask = np.ones(dst.shape[:2], dtype="uint8") * 255
        cv2.drawContours(mask, cnts, -1, 0, -1)
        cnts = cv2.findContours(edges2.copy(), cv2.RETR_LIST, cv2.CH
AIN_APPROX_NONE)
        cnts = imutils.grab_contours(cnts)
        mask = np.ones(dst.shape[:2], dtype="uint8") * 255
        # Loop over the contours
        for c in cnts:
            # if the contour is bad, draw it on the mask
            if float(cv2.contourArea(cv2.convexHull(c))) !=0:
                if cv2.contourArea(c) / float(cv2.contourArea(cv2.co
nvexHull(c)))<0.8 and cv2.arcLength(c, True)>20:
                    cv2.drawContours(mask, [c], -1, 1, 2)

```

```

    cnts_2 = cv2.findContours(mask.copy(), cv2.RETR_LIST, cv2.CHAIN_APPROX_NONE)
    cnts_2 = imutils.grab_contours(cnts_2)
    mask_4 = im.fromarray(mask)
    mask_4 = mask_4.filter(ImageFilter.ModeFilter(size=5))

    mask_5=np.asarray(mask_4)

    edges_5=cv2.Canny(mask_5, threshold1=0, threshold2=200,L2gradient=True)
    cnts_5 = cv2.findContours(edges_5.copy(), cv2.RETR_LIST, cv2.CHAIN_APPROX_NONE)
    cnts_5 = imutils.grab_contours(cnts_5)
    mask_6 = np.ones(dst.shape[:2], dtype="uint8") * 255
    for c in cnts_5:
        cv2.drawContours(mask_6, [c], -1, 0, -1)

    contours = cv2.findContours(edges_5.copy(), cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
    contours = imutils.grab_contours(contours)

    # Find the convex hull object for each contour
    hull_list = []
    for i in range(len(contours)):

        hull = cv2.convexHull(contours[i])
        hull_list.append(hull)

    hull_img = np.ones(mask.shape[:2], dtype="uint8") * 255
    for i in range(len(contours)):
        # cv2.drawContours(hull_img, contours, i, (0,0,0))
        if is_contour_bad(contours[i]):
            cv2.drawContours(hull_img, hull_list, i,(0,0,0),-1)

    found=0
    not_found=0
    found_bad=0
    black_count=0
    for i in range(len(manual)):
        if 0 in manual[i]:
            black_count =black_count + list(manual[i]).count(0)
            for j in range(len(manual[i])):

                if manual[i][j]==0 and mask_6[i][j]==0:
                    found=found+1
                elif manual[i][j]==0 and mask_6[i][j]!=0:
                    not_found=not_found+1
                elif manual[i][j]!=0 and mask_6[i][j]==0:

```

```

        found_bad=found_bad+1
score=found-found_bad/1.5

if score>best_score:
    best_score=score
    best_found=found
    best_not_found=not_found
    best_found_bad=found_bad
    best=2*num+60

    best_im_th=im_th
    best_thresh=thresh
    best_hull_img=hull_img
    best_mask_6=mask_6

cv2.imwrite(save_1,best_im_th)
cv2.imwrite(save_2,best_thresh)
cv2.imwrite(save_3,best_mask_6)
cv2.imwrite(save_4,best_hull_img)

log={'score':best_score,'finding accuracy':(100*found/black_count),
'good brightness thresh':best, 'black pixels found': best_found,
'black pixels not found':best_not_found,'pseudo findings':best_found_bad}
return log

%%time

useful_data={}
save_dir_5=r'C:\cmg\MISORJA1\ADV_DIP1_images'
save_dir_6=r'C:\cmg\MISORJA1\ADV_DIP2_images'
save_dir_7=r'C:\cmg\MISORJA1\ADV_DIP3_images'
save_dir_8=r'C:\cmg\MISORJA1\ADV_DIP4_images'

count_files=0
for i in range(len(REAL_IMAGES)):

    useful_data[REAL_IMAGES[i]]=DIP_extra(REAL_IMAGES[i],
        save_dir_5+REAL_IMAGES[i][REAL_IMAGES[i]
].rfind('\\')+'.jpg',
        save_dir_6+REAL_IMAGES[i][REAL_IMAGES[i]
].rfind('\\')+'.jpg',
        save_dir_7+REAL_IMAGES[i][REAL_IMAGES[i]
].rfind('\\')+'.jpg',
        save_dir_8+REAL_IMAGES[i][REAL_IMAGES[i]
].rfind('\\')+'.jpg'
    )

    count_files=count_files+1

```

```
print(count_files, ' files are done out of', len(REAL_IMAGES), ' files')

```

### 8.5. Hibrid algoritmus mesterséges intelligenciát felhasználó része

```
import os
os.environ["KMP_DUPLICATE_LIB_OK"]="TRUE"
import anodet
import numpy as np
import torch
import cv2
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt

DATASET_PATH = os.path.realpath(r"C:\cmg\SORJAPRED3\to_anomalib\\")
MODEL_DATA_PATH = os.path.realpath(r"C:\cmg\SORJAPRED3\to_anomalib\please_please\distriutions\\")

dataset = anodet.AnodetDataset(os.path.join(DATASET_PATH, r"please_please\train\no_fiber"))
dataloader = DataLoader(dataset, batch_size=32)
print("Number of images in dataset:", len(dataloader.dataset))

padim = anodet.Padim(backbone='resnet18')

padim.fit(dataloader)

distributions_path = r"C:\cmg\SORJAPRED3\to_anomalib\please_please\distriutions\\"
torch.save(padim.mean, os.path.join(MODEL_DATA_PATH, "mean.pt"))
torch.save(padim.cov_inv, os.path.join(MODEL_DATA_PATH, "cov_inv.pt"))

# folder path

dirdir=r"C:\cmg\MISORJA1\to_padim"

new_path=[]

# Iterate directory
for path in os.listdir(dirdir):
    # check if current path is a file
    if os.path.isfile(os.path.join(dirdir, path)):
        new_path.append(dirdir+'\\'+path)

images = []
for path in new_path:
    #print(path)
    image = cv2.imread(path)

```

```

        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        images.append(image)

batch = anodet.to_batch(images, anodet.standard_image_transform, torch.device('cpu'))

mean = torch.load(os.path.join(MODEL_DATA_PATH, 'mean.pt'))
cov_inv = torch.load(os.path.join(MODEL_DATA_PATH, 'cov_inv.pt'))

padim = anodet.Padim(backbone='resnet18', mean=mean, cov_inv=cov_inv, device=torch.device('cpu'))

image_scores, score_maps = padim.predict(batch)

THRESH = 22
score_map_classifications = anodet.classification(score_maps, THRESH)
image_classifications = anodet.classification(image_scores, THRESH)
print("Image scores:", image_scores)
print("Image classifications:", image_classifications)

test_images = np.array(images).copy()

%%time

boundary_images = anodet.visualization.framed_boundary_images(test_images, score_map_classifications, image_classifications, padding=40)
heatmap_images = anodet.visualization.heatmap_images(test_images, score_maps, alpha=0.5)
highlighted_images = anodet.visualization.highlighted_images(test_images, score_map_classifications, color=(255, 165, 0))

for idx in range(60):
    fig, axs = plt.subplots(1, 4, figsize=(24, 12))
    fig.suptitle('Image: ' + new_path[idx][new_path[idx].rfind('\\')+1:], y=0.75, fontsize=14)
    axs[0].imshow(images[idx])
    axs[1].imshow(boundary_images[idx])
    axs[2].imshow(heatmap_images[idx])
    axs[2].axis('off')
    axs[3].imshow(highlighted_images[idx])
    axs[3].axis('off')
    #plt.savefig('C:\cmg\MISORJA1\PaDiM\mix'+new_path[idx][new_path[idx].rfind('\\')+1:])
    plt.show()

print('Done')

%%time

from PIL import Image

```

```

boundary_images = anodet.visualization.framed_boundary_images(test_images, score_map_classifications, image_classifications, padding=40)
heatmap_images = anodet.visualization.heatmap_images(test_images, score_maps, alpha=0.5)
highlighted_images = anodet.visualization.highlighted_images(test_images, score_map_classifications, color=(255, 165, 0))

for i in range(len(images)):
    img = Image.fromarray(highlighted_images[i], 'RGB')
    img.save(r'C:\cmg\MISORJA1\PaDiM2\purple'+new_path[i][new_path[i].rfind('\'):]])

for i in range(len(images)):
    img = Image.fromarray(heatmap_images[i], 'RGB')
    img.save(r'C:\cmg\MISORJA1\PaDiM2\heatmap'+new_path[i][new_path[i].rfind('\'):]])

print('done')

blend= np.zeros([images[0].shape[0],images[0].shape[1],3],dtype=np.uint8)

blend.shape

highlighted_images[0].shape

blend_list=[]

for i in range(len(images)):
    blend_list.append(blend)

highlighted_images2 = anodet.visualization.highlighted_images(blend_list, score_map_classifications, color=(255, 255, 255))

for i in range(len(images)):
    img = Image.fromarray(highlighted_images2[i], 'RGB')
    img.save(r'C:\cmg\MISORJA1\PaDiM2\final'+new_path[i][new_path[i].rfind('\'):]])

```

## 8.6. Mérészámok meghatározása a DIP és a hibrid algoritmus képeinek összehasonlításával a manuálisan készített képekkel

```
import os

# folder path
MANUAL_PATH = r'C:\cmg\MISORJA1\manual_fibre_detection'
DIP1_PATH=r'C:\cmg\MISORJA1\DIP1_images'
DIP2_PATH=r'C:\cmg\MISORJA1\DIP2_images'
DIP3_PATH=r'C:\cmg\MISORJA1\DIP3_images'
DIP4_PATH=r'C:\cmg\MISORJA1\DIP4_images'
PADIM_PATH=r'C:\cmg\MISORJA1\PaDiM\real_final'
PADIM2_PATH=r'C:\cmg\MISORJA1\PaDiM2\real_final'

# list to store files
MANUAL_PATHS=[]
DIP1_PATHS=[]
DIP2_PATHS=[]
DIP3_PATHS=[]
DIP4_PATHS=[]
PADIM_PATHS=[]
PADIM2_PATHS=[]

# Iterate directory
for path in os.listdir(DIP4_PATH):
    # check if current path is a file
    if os.path.isfile(os.path.join(DIP4_PATH, path)):

        MANUAL_PATHS.append(MANUAL_PATH+'\\'+path)
        PADIM_PATHS.append(PADIM_PATH+'\\'+path[:-4]+'_v3.jpg')
        PADIM2_PATHS.append(PADIM2_PATH+'\\'+path[:-4]+'_v3.jpg')
        DIP1_PATHS.append(DIP1_PATH+'\\'+path)
        DIP2_PATHS.append(DIP2_PATH+'\\'+path)
        DIP3_PATHS.append(DIP3_PATH+'\\'+path)
        DIP4_PATHS.append(DIP4_PATH+'\\'+path)

manual_0=cv2.imread(MANUAL_PATHS[0],cv2.IMREAD_GRAYSCALE)
padim_0=cv2.imread(PADIM_PATHS[0],cv2.IMREAD_GRAYSCALE)

resized = cv2.resize(manual_0, (padim_0.shape[1],padim_0.shape[1]),
interpolation = cv2.INTER_AREA)

(thresh, manual) = cv2.threshold(resized, 128, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)
(thresh, padim) = cv2.threshold(padim_0, 128, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)
```

```

all=manual.shape[0]*manual.shape[1]

found=0
not_found=0
found_bad=0
black_count=0
for i in range(len(manual)):
    if 0 in manual[i]:
        black_count =black_count + list(manual[i]).count(0)
        for j in range(len(manual[i])):

            if manual[i][j]==0 and padim[i][j]==0:
                found=found+1
            elif manual[i][j]==0 and padim[i][j]!=0:
                not_found=not_found+1
            elif manual[i][j]!=0 and padim[i][j]==0:
                found_bad=found_bad+1

print('There were ',black_count, 'black pixels in manual detected ob
ject')
print(found,' pixels were found')
print(not_found,' pixels were not found')
print(found_bad,' pixels were found on bad place')

print('That is ',(100*found/black_count),'% accuracy')

def metrics_generation(MANUAL_IN,DIP1_IN,DIP2_IN,DIP3_IN,DIP4_IN,PAD
IM_IN,PADIM2_IN):

    manual_0=cv2.imread(MANUAL_IN,cv2.IMREAD_GRAYSCALE)
    dip1_0=cv2.imread(DIP1_IN,cv2.IMREAD_GRAYSCALE)
    dip2_0=cv2.imread(DIP2_IN,cv2.IMREAD_GRAYSCALE)
    dip3_0=cv2.imread(DIP3_IN,cv2.IMREAD_GRAYSCALE)
    dip4_0=cv2.imread(DIP4_IN,cv2.IMREAD_GRAYSCALE)
    padim_0=cv2.imread(PADIM_IN,cv2.IMREAD_GRAYSCALE)
    padim2_0=cv2.imread(PADIM2_IN,cv2.IMREAD_GRAYSCALE)

    resized = cv2.resize(manual_0, (padim_0.shape[1],padim_0.shape[1
]), interpolation = cv2.INTER_AREA)

    (thresh, manual) = cv2.threshold(resized, 128, 255, cv2.THRESH_B
INARY | cv2.THRESH_OTSU)
    (thresh, padim) = cv2.threshold(padim_0, 128, 255, cv2.THRESH_BI
NARY | cv2.THRESH_OTSU)
    (thresh, padim2) = cv2.threshold(padim2_0, 128, 255, cv2.THRESH_
BINARY | cv2.THRESH_OTSU)

```

```

    (thresh, dip1) = cv2.threshold(dip1_0, 128, 255, cv2.THRESH_BINA
RY | cv2.THRESH_OTSU)
    (thresh, dip2) = cv2.threshold(dip2_0, 128, 255, cv2.THRESH_BINA
RY | cv2.THRESH_OTSU)
    (thresh, dip3) = cv2.threshold(dip3_0, 128, 255, cv2.THRESH_BINA
RY | cv2.THRESH_OTSU)
    (thresh, dip4) = cv2.threshold(dip4_0, 128, 255, cv2.THRESH_BINA
RY | cv2.THRESH_OTSU)

```

```

all=manual.shape[0]*manual.shape[1]

```

```

dip1_found=0
dip1_not_found=0
dip1_found_bad=0
dip2_found=0
dip2_not_found=0
dip2_found_bad=0
dip3_found=0
dip3_not_found=0
dip3_found_bad=0
dip4_found=0
dip4_not_found=0
dip4_found_bad=0
padim_found=0
padim_not_found=0
padim_found_bad=0
padim2_found=0
padim2_not_found=0
padim2_found_bad=0
black_count=0

```

```

for i in range(len(manual)):
    black_count =black_count + list(manual[i]).count(0)
    for j in range(len(manual[i])):

        if manual[i][j]==0 and padim[i][j]==0:
            padim_found=padim_found+1
        elif manual[i][j]==0 and padim[i][j]!=0:
            padim_not_found=padim_not_found+1
        elif manual[i][j]!=0 and padim[i][j]==0:
            padim_found_bad=padim_found_bad+1

        if manual[i][j]==0 and padim2[i][j]==0:
            padim2_found=padim2_found+1
        elif manual[i][j]==0 and padim2[i][j]!=0:
            padim2_not_found=padim2_not_found+1
        elif manual[i][j]!=0 and padim2[i][j]==0:
            padim2_found_bad=padim2_found_bad+1

```

```

        if manual[i][j]==0 and dip1[i][j]==0:
            dip1_found=dip1_found+1
        elif manual[i][j]!=0 and dip1[i][j]==0:
            dip1_found_bad=dip1_found_bad+1
        elif manual[i][j]==0 and dip1[i][j]!=0:
            dip1_not_found=dip1_not_found+1

        if manual[i][j]==0 and dip2[i][j]==0:
            dip2_found=dip2_found+1
        elif manual[i][j]!=0 and dip2[i][j]==0:
            dip2_found_bad=dip2_found_bad+1
        elif manual[i][j]==0 and dip2[i][j]!=0:
            dip2_not_found=dip2_not_found+1

        if manual[i][j]==0 and dip3[i][j]==0:
            dip3_found=dip3_found+1
        elif manual[i][j]!=0 and dip3[i][j]==0:
            dip3_found_bad=dip3_found_bad+1
        elif manual[i][j]==0 and dip3[i][j]!=0:
            dip3_not_found=dip3_not_found+1

        if manual[i][j]==0 and dip4[i][j]==0:
            dip4_found=dip4_found+1
        elif manual[i][j]!=0 and dip4[i][j]==0:
            dip4_found_bad=dip4_found_bad+1
        elif manual[i][j]==0 and dip4[i][j]!=0:
            dip4_not_found=dip4_not_found+1

log={'Total_black_pixels_in_manual':black_count,
'DIP1_black_pixels_found':dip1_found,
'DIP1_black_pixels_found_on_bad_pos':dip1_found_bad,
'DIP1_black_pixels_not_found':dip1_not_found,
'DIP2_black_pixels_found':dip2_found,
'DIP2_black_pixels_found_on_bad_pos':dip2_found_bad,
'DIP2_black_pixels_not_found':dip2_not_found,
'DIP3_black_pixels_found':dip3_found,
'DIP3_black_pixels_found_on_bad_pos':dip3_found_bad,
'DIP3_black_pixels_not_found':dip3_not_found,
'DIP4_black_pixels_found':dip4_found,
'DIP4_black_pixels_found_on_bad_pos':dip4_found_bad,
'DIP4_black_pixels_not_found':dip4_not_found,
'PADIM_black_pixels_found':padim_found,
'PADIM_black_pixels_found_on_bad_pos':padim_found_bad,
'PADIM_black_pixels_not_found':padim_not_found,
'PADIM2_black_pixels_found':padim2_found,
'PADIM2_black_pixels_found_on_bad_pos':padim2_found_bad,
'PADIM2_black_pixels_not_found':padim2_not_found,
}

```

```

    return log

%%time

metrics_data={}

for i in range(len(MANUAL_PATHS)):

    metrics_data[MANUAL_PATHS[i]][MANUAL_PATHS[i].rfind('\\')+1:]=metrics_generation(MANUAL_PATHS[i],DIP1_PATHS[i],DIP2_PATHS[i],DIP3_PATHS[i],DIP4_PATHS[i],PADIM_PATHS[i],PADIM2_PATHS[i])

    print(i+1, ' files done')

metrics_data

files=[]
tot=[]
DIP1_f=[]
DIP1_nf=[]
DIP1_bf=[]
DIP2_f=[]
DIP2_nf=[]
DIP2_bf=[]
DIP3_f=[]
DIP3_nf=[]
DIP3_bf=[]
DIP4_f=[]
DIP4_nf=[]
DIP4_bf=[]
padim_f=[]
padim_nf=[]
padim_bf=[]
padim2_f=[]
padim2_nf=[]
padim2_bf=[]

for i in metrics_data:
    files.append(i)
    tot.append(metrics_data[i]['Total_black_pixels_in_manual'])
    DIP1_f.append(metrics_data[i]['DIP1_black_pixels_found'])
    DIP1_nf.append(metrics_data[i]['DIP1_black_pixels_not_found'])
    DIP1_bf.append(metrics_data[i]['DIP1_black_pixels_found_on_bad_pos'])
    DIP2_f.append(metrics_data[i]['DIP2_black_pixels_found'])
    DIP2_nf.append(metrics_data[i]['DIP2_black_pixels_not_found'])

```

```

        DIP2_bf.append(metrics_data[i]['DIP2_black_pixels_found_on_bad_p
os'])
        DIP3_f.append(metrics_data[i]['DIP3_black_pixels_found'])
        DIP3_nf.append(metrics_data[i]['DIP3_black_pixels_not_found'])
        DIP3_bf.append(metrics_data[i]['DIP3_black_pixels_found_on_bad_p
os'])
        DIP4_f.append(metrics_data[i]['DIP4_black_pixels_found'])
        DIP4_nf.append(metrics_data[i]['DIP4_black_pixels_not_found'])
        DIP4_bf.append(metrics_data[i]['DIP4_black_pixels_found_on_bad_p
os'])
        padim_f.append(metrics_data[i]['PADIM_black_pixels_found'])
        padim_nf.append(metrics_data[i]['PADIM_black_pixels_not_found'])
        padim_bf.append(metrics_data[i]['PADIM_black_pixels_found_on_bad
_pos'])
        padim2_f.append(metrics_data[i]['PADIM2_black_pixels_found'])
        padim2_nf.append(metrics_data[i]['PADIM2_black_pixels_not_found'
])
        padim2_bf.append(metrics_data[i]['PADIM2_black_pixels_found_on_b
ad_pos'])

```

```

import pandas as pd

```

```

col=[]

```

```

for i in metrics_data[files[0]]:
    col.append(i)

```

```

col.insert(0,'Files')

```

```

df = pd.DataFrame(list(zip(files,
                             tot,
                             DIP1_f,
                             DIP1_bf,
                             DIP1_nf,
                             DIP2_f,
                             DIP2_bf,
                             DIP2_nf,
                             DIP3_f,
                             DIP3_bf,
                             DIP3_nf,
                             DIP4_f,
                             DIP4_bf,
                             DIP4_nf,
                             padim_f,
                             padim_bf,
                             padim_nf,
                             padim2_f,
                             padim2_bf,

```

```
                padim2_nf
            ))
        ,columns = col)

df.to_excel(r'C:\cmg\MISORJA1\results1030.xlsx', engine='xlsxwriter'
)
```