TL;DR

The goal is to detecting ink from 3D X-ray scans and reading the contents.
Due to the heat of the volcano, the scrolls were carbonized, and are now impossible to open without breaking them.
We will try to find the best model to get the maximum accuracy.

Uriel Malka - www.kaggle.com/urielmalka
Dori Rosen - www.kaggle.com/dorirosen

Imports

In [1]:
```
import csv
from torch import nn
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader
import numpy as np
import random
from random import randrange, shuffle
from matplotlib import pyplot as plt
import matplotlib.patches as patches
from torchvision import transforms
from torchvision.transforms.functional import rotate
import torchvision
import cv2
import time, glob
import PIL
import os
import gc
import pandas as pd
import torch.utils.data as data
from tqdm import tqdm
import albumentations as A
from albumentations.pytorch import ToTensorV2

ROOT_DIR = f"/kaggle/input/vesuvius-challenge-ink-detection"
```

```
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected versio
n 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

In [2]:
```
import sys
sys.path.append('/kaggle/input/pretrainedmodels/pretrainedmodels-0.7.4')
sys.path.append('/kaggle/input/efficientnet-pytorch/EfficientNet-PyTorch-master')
sys.path.append('/kaggle/input/timm-pytorch-image-models/pytorch-image-models-maste
sys.path.append('/kaggle/input/segmentation-models-pytorch/segmentation_models.pyto
```

The scrolls

- Show data from train (images and shape).

In [3]:
```
""" Show data from train folder """

def show_image(path):
    fig, axs = plt.subplots(1, 3,figsize=(12, 6))
    count = 0
    for img in glob.glob(f"{path}/*"):

        if ".png" not in img:
            continue
        image = PIL.Image.open(img)
        label = img.split("/")[-1]
        image_array = np.array(image)
        axs[count].imshow(image_array, cmap="gray")
        axs[count].set_xlabel(f"File = {label}")
        count += 1
    for ax in axs.flat:
      ax.set(xticks=[], yticks=[], ylabel='')


show_image(f"{ROOT_DIR}/train/1")
show_image(f"{ROOT_DIR}/train/2")
show_image(f"{ROOT_DIR}/train/3")
```

```
/opt/conda/lib/python3.10/site-packages/PIL/Image.py:3176: DecompressionBombWarnin
g: Image size (140973980 pixels) exceeds limit of 89478485 pixels, could be decompr
ession bomb DOS attack.
  warnings.warn(
```

Layers

To ensure accurate testing, it is advisable to focus on the inner layers of the sample rather than the outer layers.
The outer layers may contain volcanic ash, which could potentially interfere with the test results.
Therefore, it is recommended to exclude the outer layers from the testing process to maintain the reliability and validity of the tests.

We will take for training the layers from X_START to (X_END - 1)

In [4]:
```
X_START = 20
X_END = 40
LAYERS_SIZE = X_END - X_START
```

DataSet

In [5]:
```
class VesuviusDataset(data.Dataset):

    def __init__(self,
                 slice_imgs:list[str],
                 mask_path:str,
                 label_path:str = None,
                 validation_rectangle: tuple[4] = None,
                 validation:bool =False,
                 voxels:tuple[3]= (224,224,LAYERS_SIZE)):
        super(VesuviusDataset).__init__()
        self.slice_imgs = slice_imgs
        self.mask_path = mask_path
        self.label_path = label_path
        self.validation = validation
        self.voxels = voxels
        self.validation_rectangle = validation_rectangle

        self.z = voxels[2]

        self.load_label()
        self.load_slices()
        self.crate_pixels(validation_rectangle)

    def load_label(self):
        self.mask = torch.from_numpy(np.array(PIL.Image.open(self.mask_path))).gt(0
        if self.label_path:
            """ If True that mean this DataSet for train or validation because we d
            self.label = torch.from_numpy(np.array(PIL.Image.open(self.label_path))


        self.shape = self.mask.shape

        self.x = self.shape[0]
        self.y = self.shape[1]



    def crate_pixels(self,validation_rectangle: tuple[4]):

        if not validation_rectangle:
            """ If True that mean this DataSet for test """
            self.pixels = [(x,y) for x in range(self.x) for y in range(self.y) if s
            return

        """
            The next line in this function is for train and validation.
        """

        x0,y0,x1,y1 = validation_rectangle

        def check_area(x,y):
            """
                self.mask[x][y]] is False mean we out of the correct area
                self.mask[x][y] is True  mean we in the correct area
            """


            if self.validation:
                """ This return us all pixles in the rectangle area """
                return x0 < x < x1 and y0 < y < y1 and self.mask[x][y]

            """ This return us all pixles out of the rectangle area """
            return not (x0 < x < x1 and y0 < y < y1) and self.mask[x][y]

        """ (x,y) = [z0,z1, .. , z64] """
        self.pixels = [(x,y) for x in range(self.x) for y in range(self.y) if check

    def get_3D_voxels(self,x,y):

        new_image_3d = np.zeros(self.voxels)

        x_range = self.voxels[0] // 2
        y_range = self.voxels[1] // 2

        x0 = x - x_range
        y0 = y - y_range
        x1 = x0 + self.voxels[0]
        y1 = y0 + self.voxels[1]

        x_count = 0
        y_count = 0

        for xd in range(x0,x1):
            y_count = 0
            for yd in range(y0,y1):
                if yd >= 0 and xd >= 0:
                    z = self.image_stack[xd][yd]
                    new_image_3d[x_count][y_count] = z

                y_count+= 1
            x_count += 1

        return new_image_3d


    def get_2D_mask(self,x,y):

        new_mask_2d = np.zeros((self.voxels[0],self.voxels[1]))
        x_range = self.voxels[0] // 2
        y_range = self.voxels[1] // 2

        x0 = x - x_range
        y0 = y - y_range
        x1 = x + x_range
        y1 = y + y_range

        x_count = 0
        y_count = 0

        for xd in range(x0,x1):
            y_count = 0
            for yd in range(y0,y1):
                if yd >= 0 and xd >= 0:
                    new_mask_2d[x_count][y_count] = self.label[xd][yd]
                y_count+= 1
            x_count += 1

        return new_mask_2d


    def load_slices(self):
        images = [np.array(PIL.Image.open(filename), dtype=np.float32)/65535.0 for
        self.image_stack = torch.stack([torch.from_numpy(image) for image in images

    def __len__(self):
        return len(self.pixels)

    def get_z_slide(self):
        x, y = self.pixels[randrange(self.__len__())]
        z = self.image_stack[x][y]

        if self.label_path:
            l = self.label[x][y]
        else:
            """ l = None only for test """
            l = None

        return z , l ,(x,y)



    def __getitem__(self, index):
        x, y = self.pixels[index]
        img3d = self.get_3D_voxels(x,y)

        if self.label_path:
            l = self.get_2D_mask(x,y)
        else:
            """ l = None only for test """
            l = None

        """
        return values:
            img3d - list of z dimention
            l - label if 1 is ink if 0 is noink
            (x,y) - location from the image
        """
        return img3d , l

    def getranditem(self,):
        x, y = self.pixels[randrange(self.__len__())]
        img3d = self.get_3D_voxels(x,y)

        if self.label_path:
            l = self.get_2D_mask(x,y)
        else:
            """ l = None only for test """
            l = None

        """
        return values:
            img3d - list of z dimention
            l - label if 1 is ink if 0 is noink
            (x,y) - location from the image
        """
        return img3d , l ,(x,y)


    """ Show validation rectangle """
    def show_validation_rectangle(self):
        if self.validation_rectangle:
            array_2d = self.label
            fig, ax = plt.subplots()
            ax.imshow(array_2d, cmap='gray')
            rectangle = self.validation_rectangle
            rectangle_x = self.validation_rectangle[0]
            rectangle_y = self.validation_rectangle[1]
            rectangle_width = self.validation_rectangle[2]
            rectangle_height = self.validation_rectangle[3]
            rectangle = patches.Rectangle((rectangle_x, rectangle_y), rectangle_wid
            ax.add_patch(rectangle)
            plt.show()
```

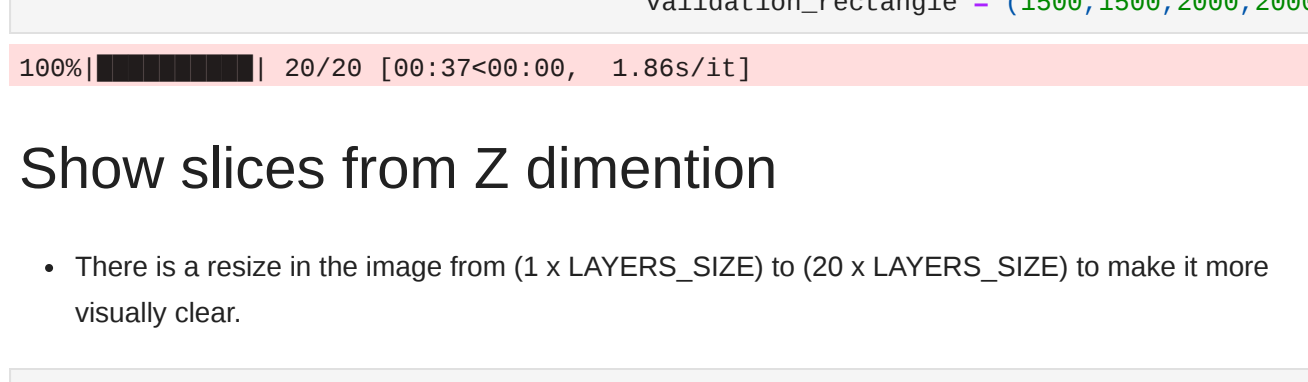
In [6]:
```
val1 = list(glob.glob(f"{ROOT_DIR}/train/1/surface_volume/*"))[X_START:X_END]
label = f"{ROOT_DIR}/train/1/inklabels.png"
mask = f"{ROOT_DIR}/train/1/mask.png"

train_VesuviusDataset = VesuviusDataset(slice_imgs=val1,
                                        mask_path=mask,
                                       label_path=label,
                                        validation_rectangle = (1500,1500,2000,2000
```

```
100%|██████████| 20/20 [00:37<00:00,  1.86s/it]
```

Show slices from Z dimention

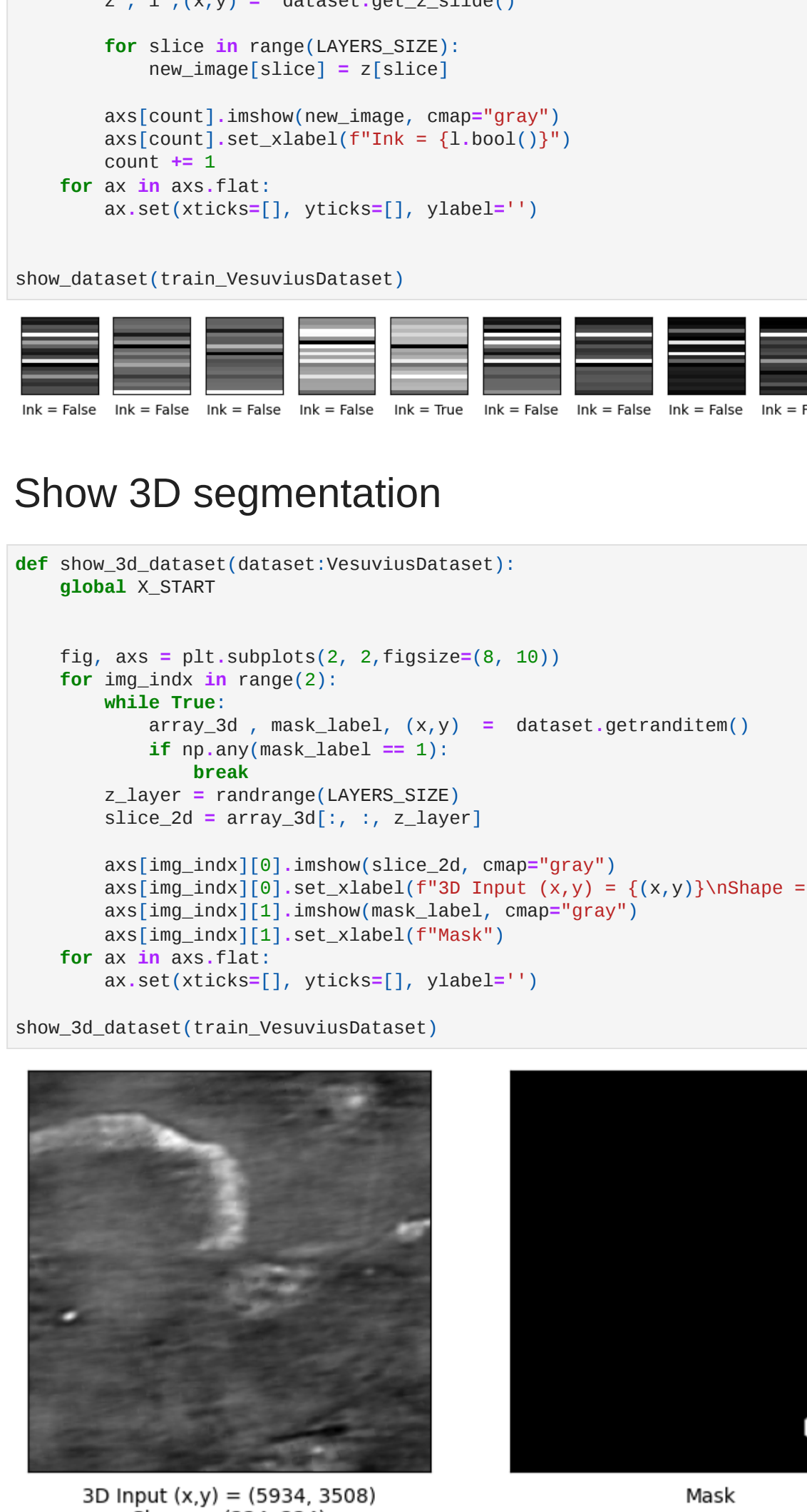
- There is a resize in the image from (1 x LAYERS_SIZE) to (20 x LAYERS_SIZE) to make it more visually clear.

In [7]:
```
def show_dataset(dataset:VesuviusDataset):
    fig, axs = plt.subplots(1, 10,figsize=(12, 6))

    for count in range(10):
        new_image = np.zeros((LAYERS_SIZE,20))
        z , l ,(x,y) =  dataset.get_z_slide()

        for slice in range(LAYERS_SIZE):
            new_image[slice] = z[slice]

        axs[count].imshow(new_image, cmap="gray")
        axs[count].set_xlabel(f"Ink = {l.bool()}")
        count += 1
    for ax in axs.flat:
        ax.set(xticks=[], yticks=[], ylabel='')


show_dataset(train_VesuviusDataset)
```



Show 3D segmentation

In [8]:
```
def show_3d_dataset(dataset:VesuviusDataset):
    global X_START


    fig, axs = plt.subplots(2, 2,figsize=(8, 10))
    for img_indx in range(2):
        while True:
            array_3d , mask_label, (x,y)  =  dataset.getranditem()
            if np.any(mask_label == 1):
                break
        z_layer = randrange(LAYERS_SIZE)
        slice_2d = array_3d[:, :, z_layer]

        axs[img_indx][0].imshow(slice_2d, cmap="gray")
        axs[img_indx][0].set_xlabel(f"3D Input (x,y) = {(x,y)}\nShape = {slice_2d.s
        axs[img_indx][1].imshow(mask_label, cmap="gray")
        axs[img_indx][1].set_xlabel(f"Mask")
    for ax in axs.flat:
        ax.set(xticks=[], yticks=[], ylabel='')

show_3d_dataset(train_VesuviusDataset)
```



DataSet and fucntions for test

In [9]:
```
class CFG:
    # ============== comp exp name =============
    comp_name = 'vesuvius'
    comp_dir_path = '/kaggle/input/'
    comp_folder_name = 'vesuvius-challenge-ink-detection'
    comp_dataset_path = f'{comp_dir_path}{comp_folder_name}/'

    # ============== pred target =============
    target_size = 1

    # ============== model cfg =============
    model_name = 'Unet'
    backbone= 'densenet161'

    in_chans = 13
    # ============== training cfg =============
    size = 224
    tile_size = 224
    stride = 224 // 4

    train_batch_size = 16
    valid_batch_size = train_batch_size * 2

    epochs = 15 # 30

    warmup_factor = 10
    lr = 1e-4 / warmup_factor
    # ============== fold =============
    valid_id = 1
    metric_direction = 'maximize'

    # ============== fixed =============
    pretrained = True
    inf_weight = 'best'  # 'best'

    min_lr = 1e-6
    weight_decay = 1e-6
    max_grad_norm = 1000

    print_freq = 50
    num_workers = 0

    seed = 42
    #'''
    # ============== set dataset path =============
    print('set dataset path')
    exp_name = 'exp_name'
    outputs_path = f'/kaggle/working/outputs/{comp_name}/{exp_name}/'

    submission_dir = outputs_path + 'submissions/'
    submission_path = submission_dir + f'submission_{exp_name}.csv'

    model_dir = outputs_path + \
        f'{comp_name}-models/'

    figures_dir = outputs_path + 'figures/'

    log_dir = outputs_path + 'logs/'
    log_path = log_dir + f'{exp_name}.txt'
    #'''
    # ============== augmentation =============

    train_aug_list = [
        A.RandomBrightnessContrast(p=0.25),
        A.RandomGridShuffle(grid=(3, 3), always_apply=False, p=0.4),
        A.OneOf([
                A.GaussNoise(var_limit=[10, 50]),
                A.GaussianBlur(),
                A.MotionBlur(),
                ], p=0.4),
        A.GridDistortion(num_steps=5, distort_limit=0.3, p=0.5),
        A.CoarseDropout(max_holes=1, max_width=int(size * 0.3), max_height=int(size
                        mask_fill_value=0, p=0.5),
        A.Normalize(mean= [0] * in_chans, std= [1] * in_chans),
        ToTensorV2(transpose_mask=True)
    ]

    valid_aug_list = [
        A.OneOf([
                A.GaussNoise(var_limit=[10, 50]),
                A.GaussianBlur(),
                A.MotionBlur(),
                ], p=0.4),
        A.RandomGridShuffle(grid=(3, 3), always_apply=False, p=0.4),
        A.Normalize(mean= [0] * in_chans, std= [1] * in_chans),
        ToTensorV2(transpose_mask=True)
    ]


class InkDetectionDatasetTest(Dataset):
    def __init__(self, images, cfg, labels=None, transform=None):
        self.images = images
        self.cfg = cfg
        self.labels = labels
        self.transform = transform

    def __len__(self):
        return len(self.images)

    def __getitem__(self, idx):
        image = self.images[idx]

        label = np.zeros_like(image)

        if self.transform:
            data = self.transform(image=image, mask=label)
            image = data['image']
            label = data['mask']

        return image

def read_image_mask(fragment_id):

    images = []
    start = 22
    end = 35
    idxs = range(start, end)

    for i in tqdm(idxs):
        ForStringNumber = str("{:02d}".format(i))
        tpath = CFG.comp_dataset_path + f"test/{fragment_id}/surface_volume/{ForStr
        image = cv2.imread(tpath,0)

        pad0 = (CFG.tile_size - image.shape[0] % CFG.tile_size)
        pad1 = (CFG.tile_size - image.shape[1] % CFG.tile_size)

        image = np.pad(image, [(0, pad0), (0, pad1)], constant_values=0)

        images.append(image)
    images = np.stack(images, axis=2)

    return images


def rle(img):
    '''
    img: numpy array, 1 - mask, 0 - background
    Returns run length as string formated
    '''
    pixels = img.flatten()
    # pixels = (pixels >= thr).astype(int)

    pixels = np.concatenate([[0], pixels, [0]])
    runs = np.where(pixels[1:] != pixels[:-1])[0] + 1
    runs[1::2] -= runs[::2]
    return ' '.join(str(x) for x in runs)

def make_test_dataset(fragment_id):
    test_images = read_image_mask(fragment_id)

    x1_list = list(range(0, test_images.shape[1]-CFG.tile_size+1,CFG.stride//4))
    y1_list = list(range(0, test_images.shape[0]-CFG.tile_size+1, CFG.stride//4))

    test_images_list = []
    xyxys = []
    for y1 in y1_list:
        for x1 in x1_list:
            y2 = y1 + CFG.tile_size
            x2 = x1 + CFG.tile_size

            test_images_list.append(test_images[y1:y2, x1:x2])
            xyxys.append((x1, y1, x2, y2))
    xyxys = np.stack(xyxys)

    test_dataset = InkDetectionDatasetTest(test_images_list, CFG, transform=A.Compo

    test_loader = DataLoader(test_dataset,
                          batch_size=CFG.train_batch_size,
                          shuffle=False,
                          num_workers=CFG.num_workers, pin_memory=True, drop_last=F

    return test_loader, xyxys
```

```
set dataset path
```

Test: Unet - ResNet18

- Tanh

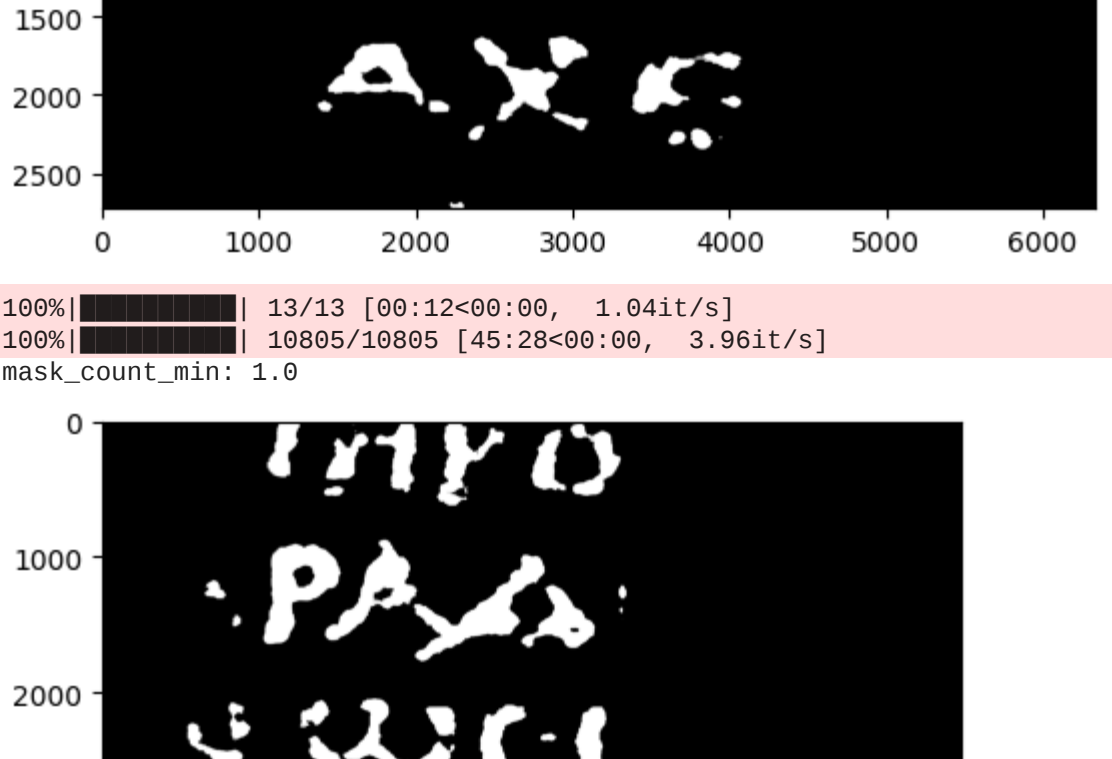
In [10]:
```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

model = torch.load("/kaggle/input/resnet18-tanh-x/u-resnet18_tanhadamW_best_model_l
```

```
/kaggle/input/pretrainedmodels/pretrainedmodels-0.7.4/pretrainedmodels/models/dpn.p
y:255: SyntaxWarning: "is" with a literal. Did you mean "=="?
  if block_type is 'proj':
/kaggle/input/pretrainedmodels/pretrainedmodels-0.7.4/pretrainedmodels/models/dpn.p
y:258: SyntaxWarning: "is" with a literal. Did you mean "=="?
  elif block_type is 'down':
/kaggle/input/pretrainedmodels/pretrainedmodels-0.7.4/pretrainedmodels/models/dpn.p
y:262: SyntaxWarning: "is" with a literal. Did you mean "=="?
  assert block_type is 'normal'
```

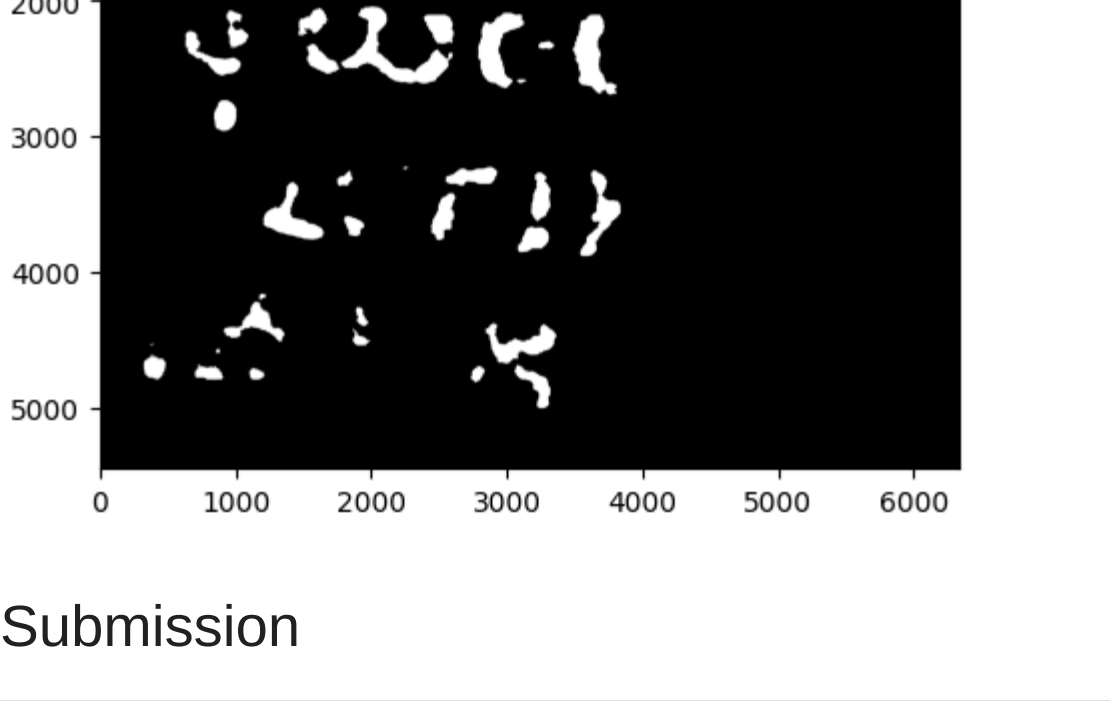
In [11]:
```
TH = 0.4
fragment_ids = sorted(os.listdir(CFG.comp_dataset_path + "test"))
```

In [12]:
```
results = []
for fragment_id in fragment_ids:

    test_loader, xyxys = make_test_dataset(fragment_id)


    binary_mask = cv2.imread(CFG.comp_dataset_path + f"test/{fragment_id}/mask.png"
    binary_mask = (binary_mask / 255).astype(int)

    ori_h = binary_mask.shape[0]
    ori_w = binary_mask.shape[1]
    # mask = mask / 255

    pad0 = (CFG.tile_size - binary_mask.shape[0] % CFG.tile_size)
    pad1 = (CFG.tile_size - binary_mask.shape[1] % CFG.tile_size)

    binary_mask = np.pad(binary_mask, [(0, pad0), (0, pad1)], constant_values=0)

    mask_pred = np.zeros(binary_mask.shape)
    mask_count = np.zeros(binary_mask.shape)

    for step, (images) in tqdm(enumerate(test_loader), total=len(test_loader)):
        images = images.to(device)
        batch_size = images.size(0)

        with torch.no_grad():
            y_preds = model(images)

        start_idx = step*CFG.train_batch_size
        end_idx = start_idx + batch_size
        for i, (x1, y1, x2, y2) in enumerate(xyxys[start_idx:end_idx]):
            mask_pred[y1:y2, x1:x2] += y_preds[i].squeeze(0).cpu().detach().numpy()
            mask_count[y1:y2, x1:x2] += np.ones((CFG.tile_size, CFG.tile_size))

    #plt.imshow(mask_count)
    #plt.show()

    print(f'mask_count_min: {mask_count.min()}')
    mask_pred /= mask_count

    mask_pred = mask_pred[:ori_h, :ori_w]
    binary_mask = binary_mask[:ori_h, :ori_w]

    mask_pred = (mask_pred >= TH).astype(int)
    mask_pred *= binary_mask

    plt.imshow(mask_pred,cmap="gray")
    plt.show()

    inklabels_rle = rle(mask_pred)

    results.append((fragment_id, inklabels_rle))


    del mask_pred, mask_count
    del test_loader

    gc.collect()
    torch.cuda.empty_cache()
```

```
100%|██████████| 13/13 [00:06<00:00,  1.99it/s]
100%|██████████| 5417/5417 [23:27<00:00,  3.85it/s]
mask_count_min: 1.0
```



```
100%|██████████| 13/13 [00:12<00:00,  1.04it/s]
100%|██████████| 10805/10805 [45:28<00:00,  3.96it/s]
mask_count_min: 1.0
```



Submission

In [13]:
```
print("Start sunmission")
sub = pd.DataFrame(results, columns=['Id', 'Predicted'])
sample_sub = pd.read_csv(CFG.comp_dataset_path + 'sample_submission.csv')
sample_sub = pd.merge(sample_sub[['Id']], sub, on='Id', how='left')
sample_sub.to_csv("submission.csv", index=False)
```

```
Start sunmission
```