Supplementary materials

Contents

1. Video descriptions

Supplementary Movie 1
Simulation results (obtained by the software COMSOL Multiphysics) showing the dynamic process of utilizing the contraction-expansion microchannel to sort a mixed solution of two types of microspheres with different sizes (the diameters are 20 μm and 10 μm respectively). In the video, the top panel shows the flow velocity distribution inside the channel under stable flow conditions, the middle panel shows the dynamic movements of the two types of microspheres in the channel, and the bottom panel is the fusion of the top panel and the middle panel.

Supplementary Movie 2
Experimental results showing the process of utilizing the contraction-expansion microchannel to sort a mixed solution of two types of polystyrene microspheres with different sizes. For visual discrimination, the large microspheres (the diameter is 20 μm) were coated with red fluorescein and the small microspheres (the diameter is 10 μm) were coated with blue fluorescein.

Supplementary Movie 3
Simulation results of utilizing the contraction-expansion microchannel with shortened contraction structures to sort a mixed solution of two types of microspheres with different sizes (the diameters are 25 μm and 12 μm respectively).

Supplementary Movie 4
Experimental results of utilizing the contraction-expansion microchannel with shortened contraction structures to sort a mixed solution of U87 cells and Raji cells. The U87 cells were stained with red fluorescein and the Raji cells were stained with blue fluorescein.

Supplementary Movie 5
Automated AFM indentation assay performed on the isolated living U87 cells to measure cellular Young's modulus. Cells and the AFM probe tip (the microsphere-modified probe was used) were automatically identified from the optical bright-field image (cells were recognized by deep learning model, and the AFM probe tip was recognized by image template matching algorithm), and then the AFM probe was precisely and sequentially moved to the cells near the AFM probe to perform indentation experiments on each cell. After the detection, the AFM sample stage was moved so that the AFM probe was positioned at a new area to detect other cells.

Supplementary Movie 6
Automated AFM indentation assay performed on the isolated living Raji cells to measure cellular Young's modulus.

Supplementary Movie 7
Experimental results of sorting U87 cells from blood by the contraction-expansion microchannel. Living U87 cells (labeled with red fluorescein for visual verification) were mixed with the blood, which were then driven to pass through the contraction-expansion microchannel.

Supplementary Movie 8
Automated AFM-based single-cell force spectroscopy (SCFS) experiments performed on the isolated living U87 cells to measure cellular adhesion force. The single-cell probe (a living MGC-803 cell was attached to the tipless cantilever) was used.

2. Supplementary figures

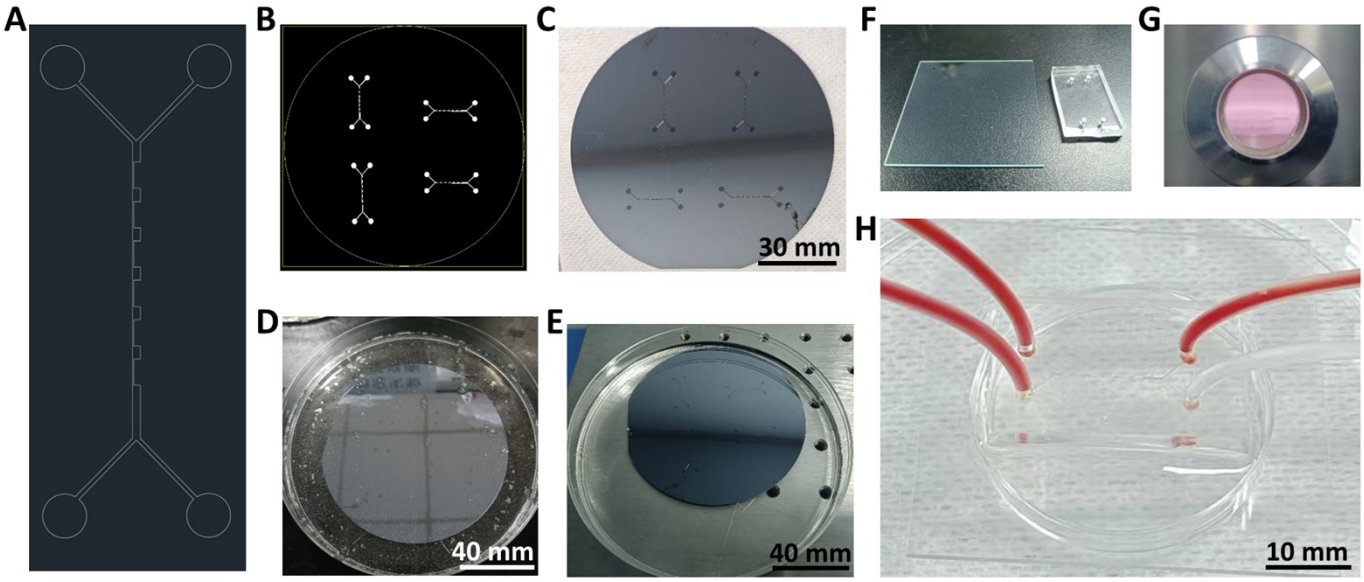


Figure S1 Process of fabricating the contraction-expansion microfluidics. (A) Drawing of the microfluidics containing contraction-expansion microchannels obtained by using the software AutoCAD. (B) Importing the AutoCAD drawing into the photolithography software according to the size the silicon wafer. (C) The mold obtained after photolithography of the silicon wafer. (D) Pouring the mixture of PDMS monomer and curing agent into the mold. (E) The hardened PDMS film on the mold. (F) Peeling the microchannel part from the mold by using a knife to cut the hardened PDMS film and preparing a fresh glass sheet. (G) Bonding the PDMS microfluidic chip onto the glass sheet by using a plasma cleaner (Diener, Germany). (H) The prepared contraction-expansion microfluidics.

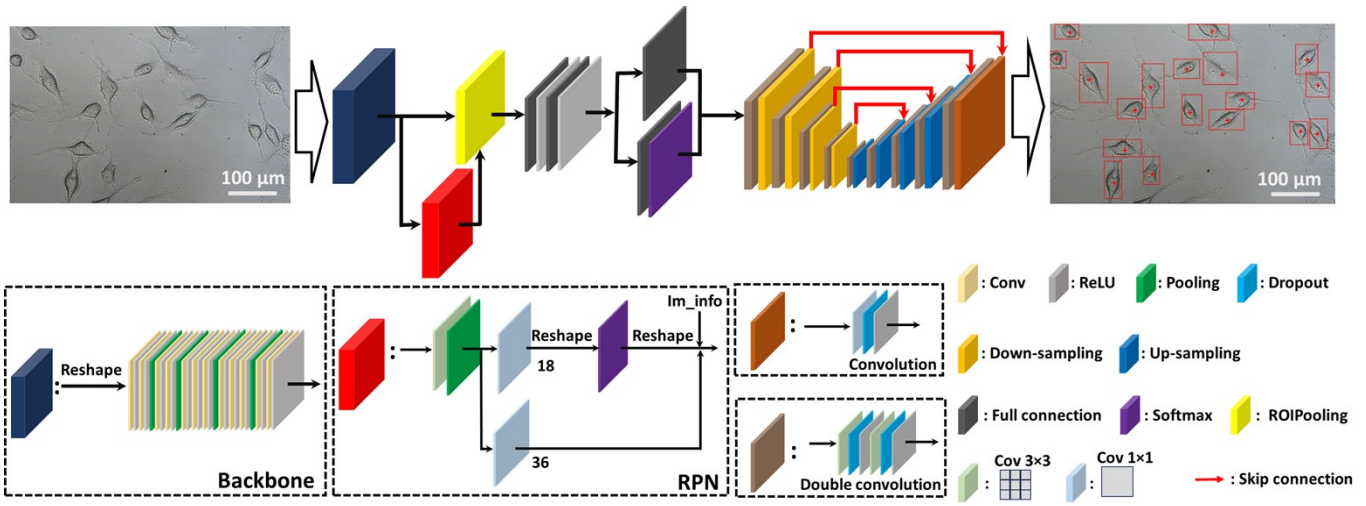


Figure S2 Structure of the deep learning optical image recognition model which is constructed based on the combination of the object detection neural network Faster R-CNN and the image segmentation neural network U-Net. With the use of the model, cells can be recognized from the optical bright-field images.

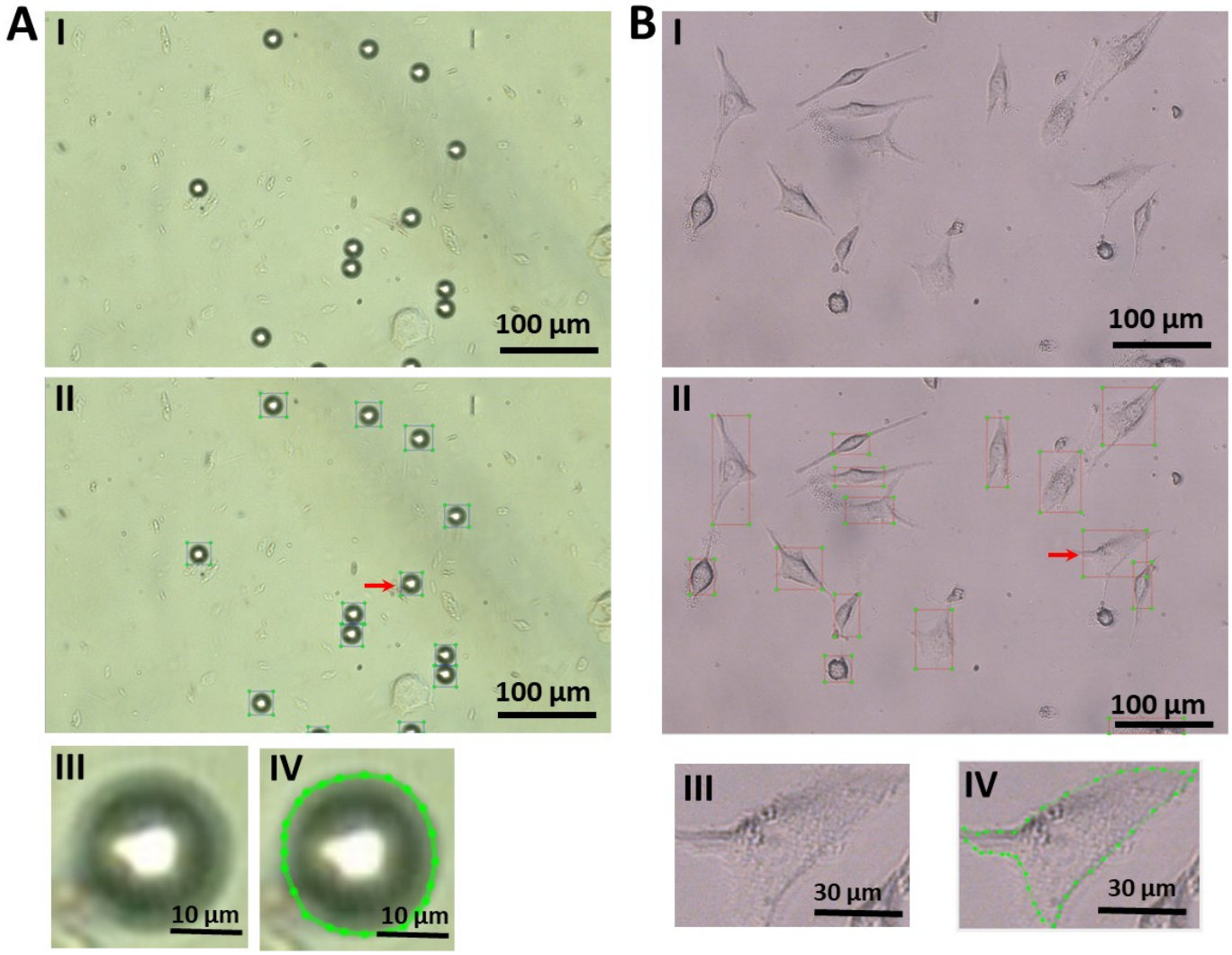
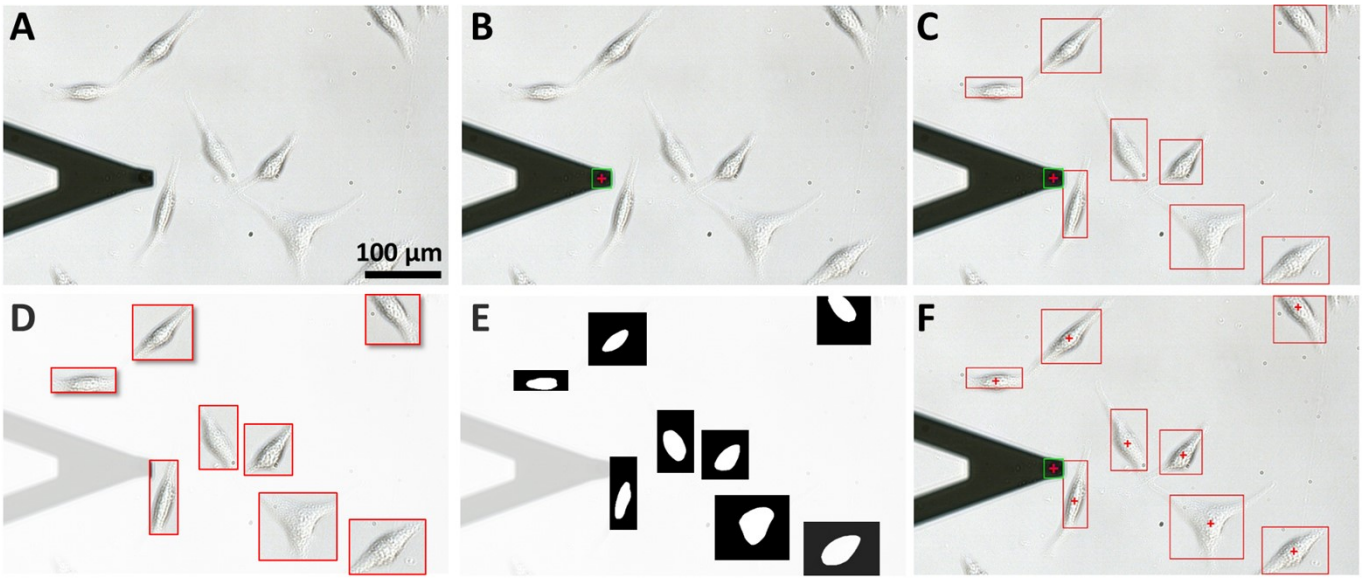


Figure S3 Preparing the annotated data for training of the deep learning image recognition model. (A) Preparing the annotated data of microspheres. (B) Preparing the annotated data of U87 cells. (I) Original optical bright-field images. (II) Creating label images using the annotation software LabelImg for Faster R-CNN-based object detection. (III, IV) Creating label images using the annotation software LabelMe for U-Net-based segmentation. An example (denoted by the red arrow) is shown. (III) Optical bright-field images of the example. (IV) Labeling the example to create label data.



G

	Parameters	GFLOPs	Recall	F1 score	Precision	Accuracy	mAP
Faster R-CNN	28.286 M	940.953	97.13%	0.88	79.61%	77.78%	96.63%

	Parameters	GFLOPs	Recall	F1 score	Precision	Accuracy	mIoU
U-Net	24.891 M	451.472	85.1%	0.94	98.5%	99.27%	87.75%

H

Method	Capturing optical images and recognition	Automatically performing detection and record force curves	Manually moving AFM probe to cellular center	Manually clicking AFM software interface buttons to obtain force curves	Total time
Manual operation	NA	NA	1.3 h	0.22 h	1.52 h
Automated measurement	0.07 h	0.43 h	NA	NA	0.5 h
Note: For manual operation, the experimental time is closely dependent on the operator's experience and can vary for different operators.					

Figure S4 The recognition process of the presented deep learning model during AFM mechanical measurements of U87 cells. (A) Original bright-field image showing the U87 cells as well as the AFM probe (microsphere-modified probe). (B) Recognizing the tip of the AFM spherical probe (denoted by the red box and the red cross indicates the center of the tip) by using the template matching algorithm. (C) Detecting the cell areas (denoted by the red boxes) by applying the Faster R-CNN. (D) Cropping the whole image based on the Faster R-CNN detection results to obtain the small image of each cell. (E) Applying the U-Net to segment the cell contour area (denoted by the white area in the small image) for each small image of the cell. (F) Obtaining the center of each cell (marked by the red cross) based on the identified cell contour. (G) Performance metrics of the deep learning model based on the combination of Faster R-CNN and U-Net. (H) Comparison of the time required for manual mechanical measurement and automated mechanical measurement of 50 cells (here 5 force curves were obtained on each cell).

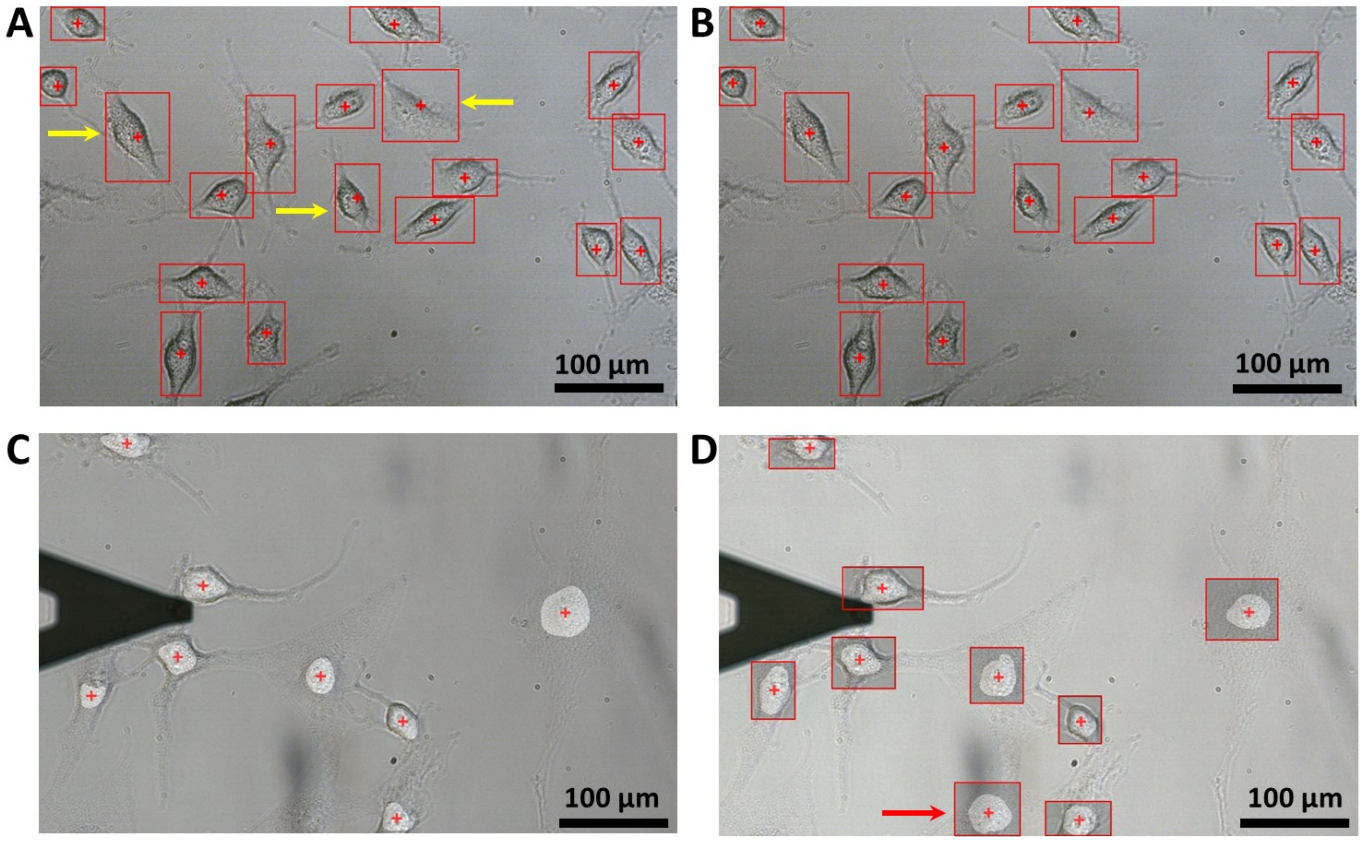


Figure S5 Comparison between the presented combined deep learning neural network and the single deep learning neural network. (A, B) Recognizing U87 cells from the optical bright-field image by Faster R-CNN (A) and by the combined Faster R-CNN and U-Net (B). (C, D) Recognizing U87 cells from the optical bright-field image by the U-Net (C) and by the combined Faster R-CNN and U-Net (D). The combined neural network performs better than the Faster R-CNN neural network in detecting the center of some cells (typically denoted by yellow arrows), and some cells (denoted by the red arrow) can be recognized by the combined network but not by the U-Net neural network, showing that the combined Faster R-CNN and U-Net has improved cell recognition performance.

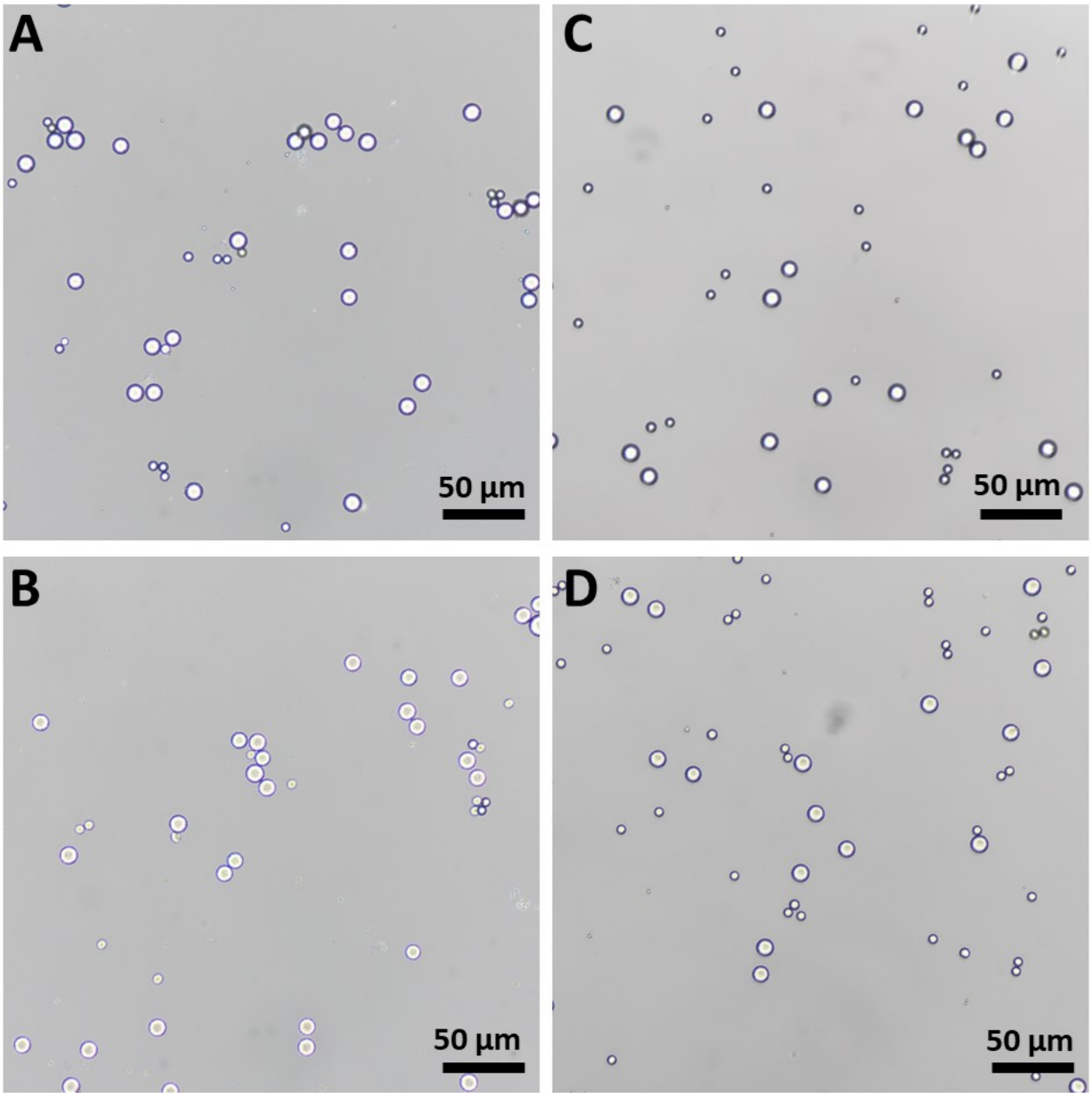


Figure S6 Performing surface modification on microspheres to disperse microspheres. (A, B) Optical images of the microsphere solution without surface modification. (C, D) Optical images of the microsphere solution after using sodium lauryl sulfonate solution to modify the microspheres.

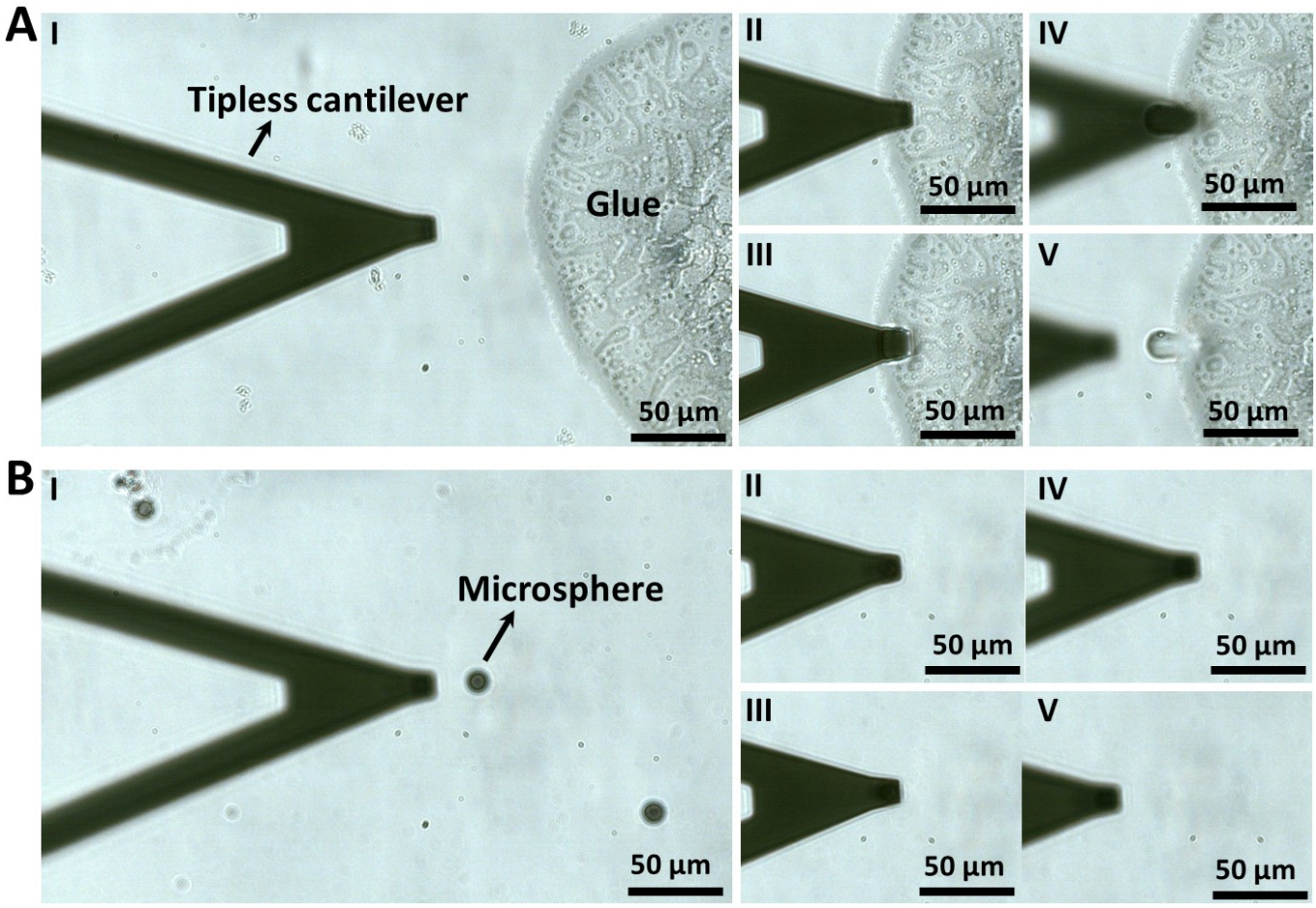


Figure S7 Process of preparing the microsphere-modified AFM probe based on AFM micromanipulations. (A) Coating the AFM tipless cantilever with glue. (I) Moving the cantilever toward the glue drop. After the cantilever reaches the glue area (II), the cantilever is controlled to contact the glue (III). The cantilever is then lifted (IV) and the sample stage is moved (V) to make the cantilever reach the microsphere area for subsequent manipulations. (B) Controlling the glue-coated cantilever to adsorb a microsphere. (I) Moving the glue-coated cantilever toward a microsphere (denoted by the black arrow). After the cantilever reaches the microsphere (II), the cantilever is controlled to touch the microsphere (III) for a period of time. (IV, V) The cantilever is then retracted and the microsphere-modified probe is obtained.

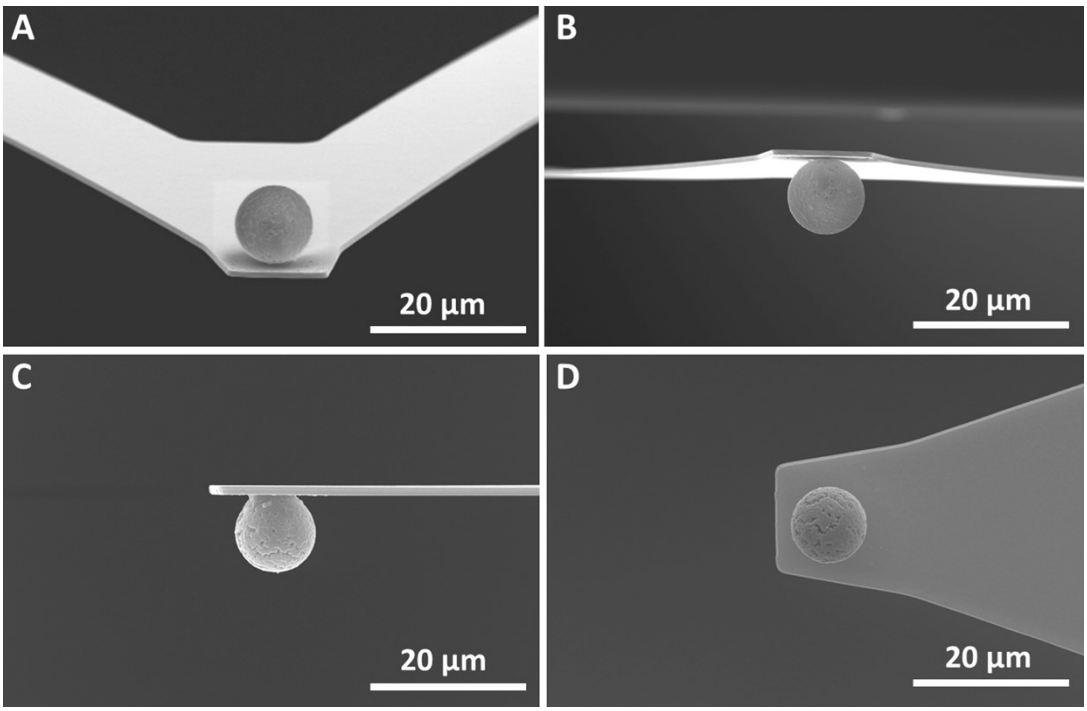
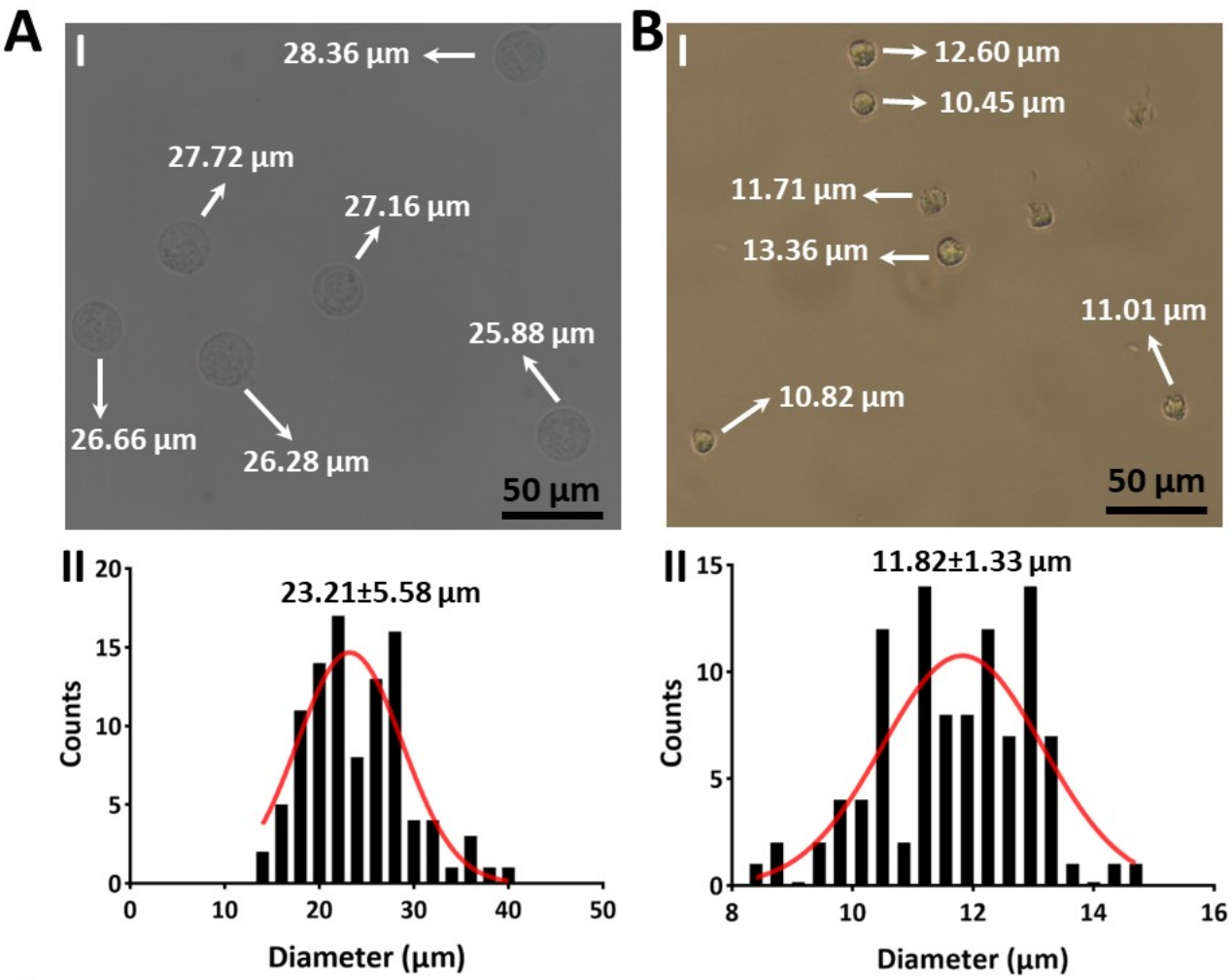


Figure S8 SEM images of a prepared microsphere-modified probe taken from different perspectives. (A) Oblique view image. (B, C) Side view images. (D) Top view image.



C

Sorting performances of a mixed solution containing U87 cells and Raji cells by the contraction-expansion microfluidics					
	Number of each cell type in the mixed solution before sorting (per 4 mL)	Number of each cell type collected from the upper outlet (per 4 mL)	Number of each cell type collected from the lower outlet (per 4 mL)	Purity	Recovery
U87 cells	11.2×10^5	6.9×10^5	1.6×10^5	77.53%	61.60%
Raji cells	7.7×10^5	2.0×10^5	5.2×10^5	76.47%	67.53%

D

Sorting performances of U87 or Raji cells from blood by the contraction-expansion microfluidics			
	Number of each cell type in the mixed solution before sorting (per 4 mL)	Number of each cell type collected after sorting (per 4 mL)	Recovery
U87 cells	6.4×10^5	4.3×10^5	67.19%
Raji cells	5.8×10^5	3.7×10^5	63.79%

Figure S9 Calculating the sizes of U87 cells and Raji cells as well as the sorting performances of the microfluidics. (A) U87 cells. (B) Raji cells. (I) Typical optical bright-field images of cells (the measured cell diameter is shown). For diameter measurements, U87 cells were digested to obtain round cells. Raji cells are naturally suspended cells and do not require digestion. (II) Statistical results of cell diameters (the red curves represent the Gaussian distribution fitting results of the histograms). (C) Performances of the contraction-expansion microfluidics in sorting a mixture of U87 cells and Raji cells. (D) Performances of the contraction-expansion microfluidics in sorting a mixture of blood and U87 (or Raji) cells.

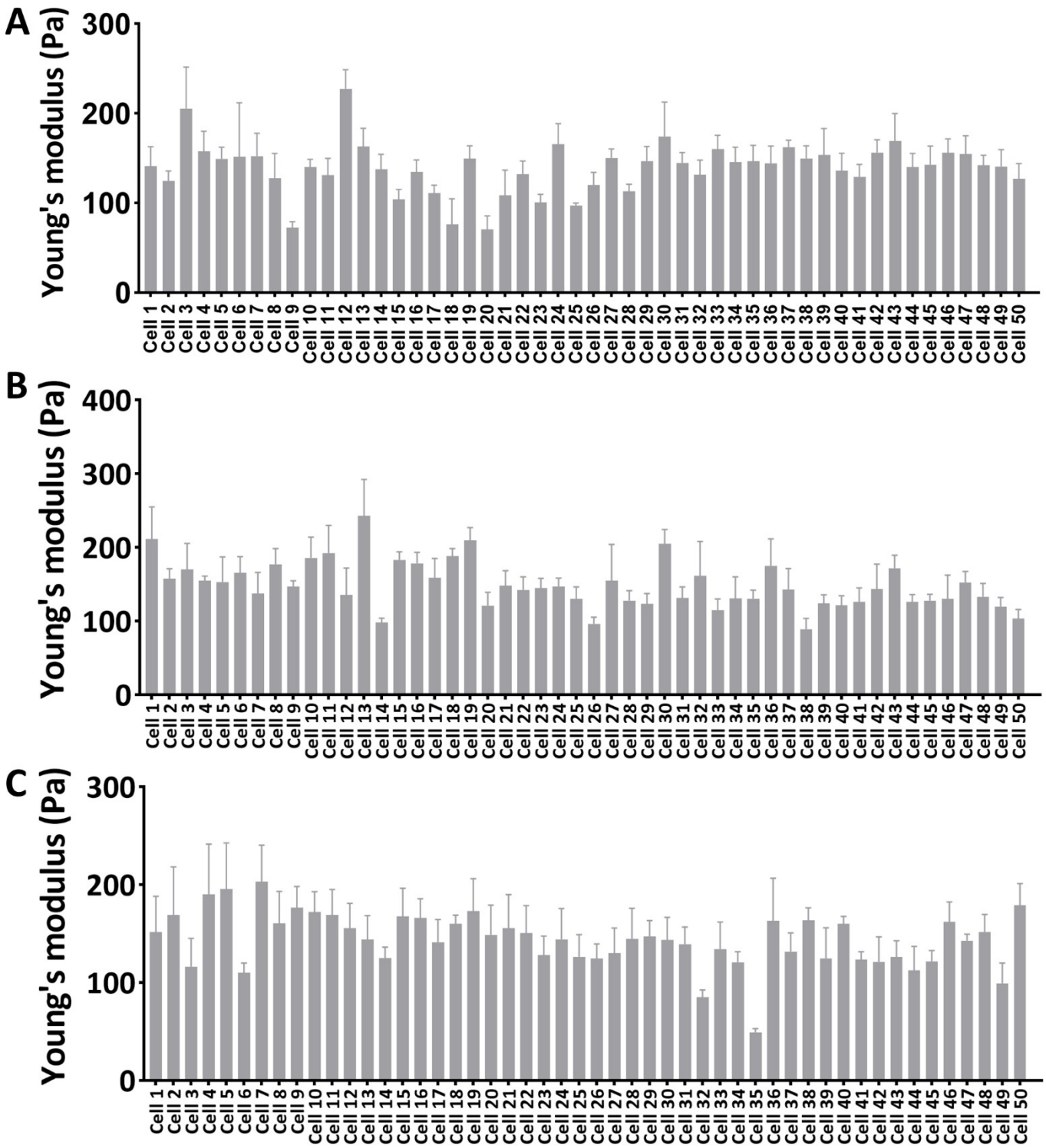


Figure S10 Histograms of the Young's modulus of U87 cells measured under three different conditions. In each condition, 50 U87 cells were measured. (A) U87 cells before passing through the contraction-expansion microchannels. (B) U87 cells for control (not passed through the contraction-expansion microchannels). (C) U87 cells after passing through the contraction-expansion microchannels.

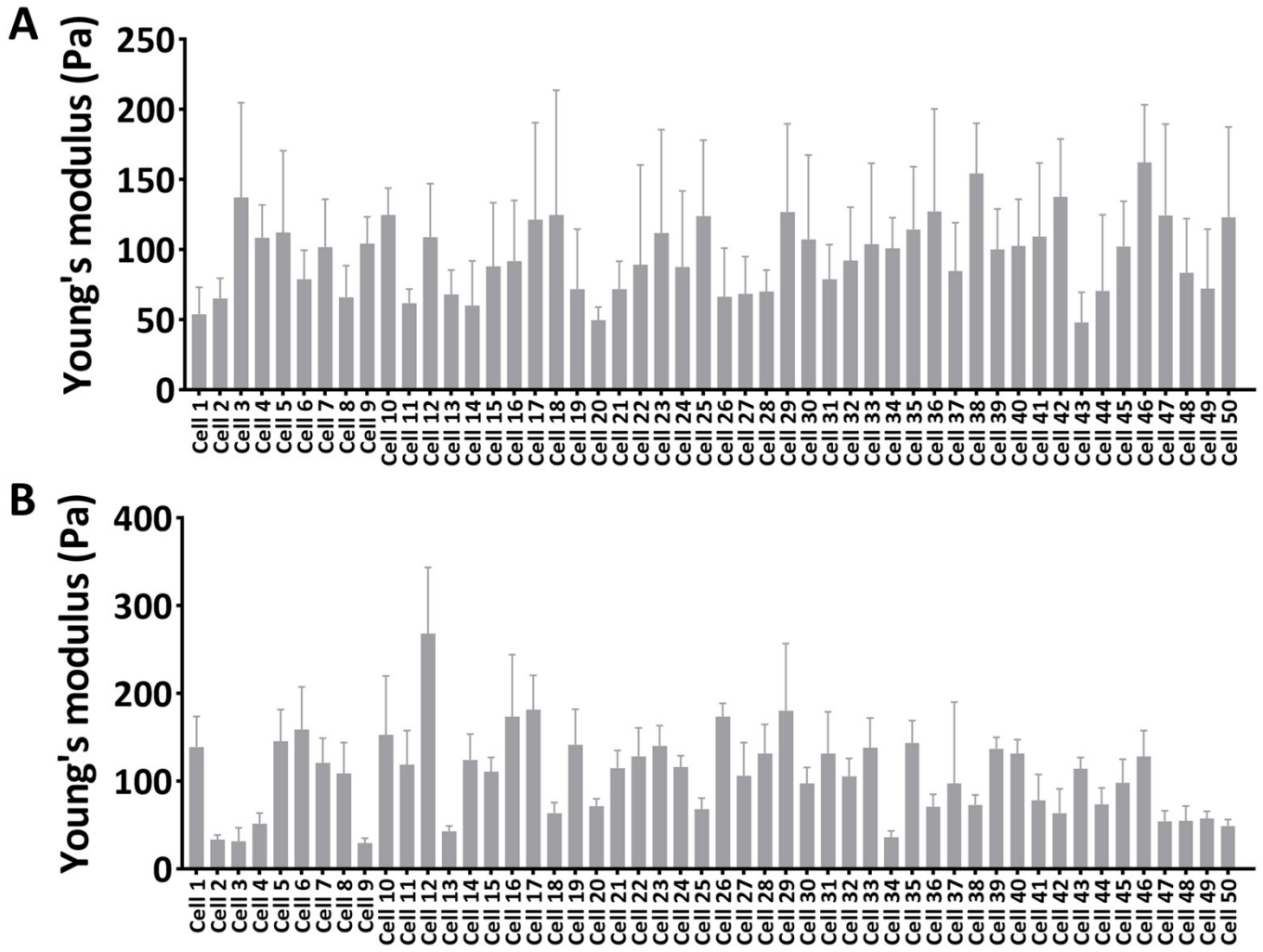


Figure S11 Histograms of the Young's modulus of Raji cells measured under two different conditions. In each condition, 50 Raji cells were measured. (A) Raji cells isolated from a mixture of U87 cells and Raji cells. (B) Raji cells isolated from the blood.

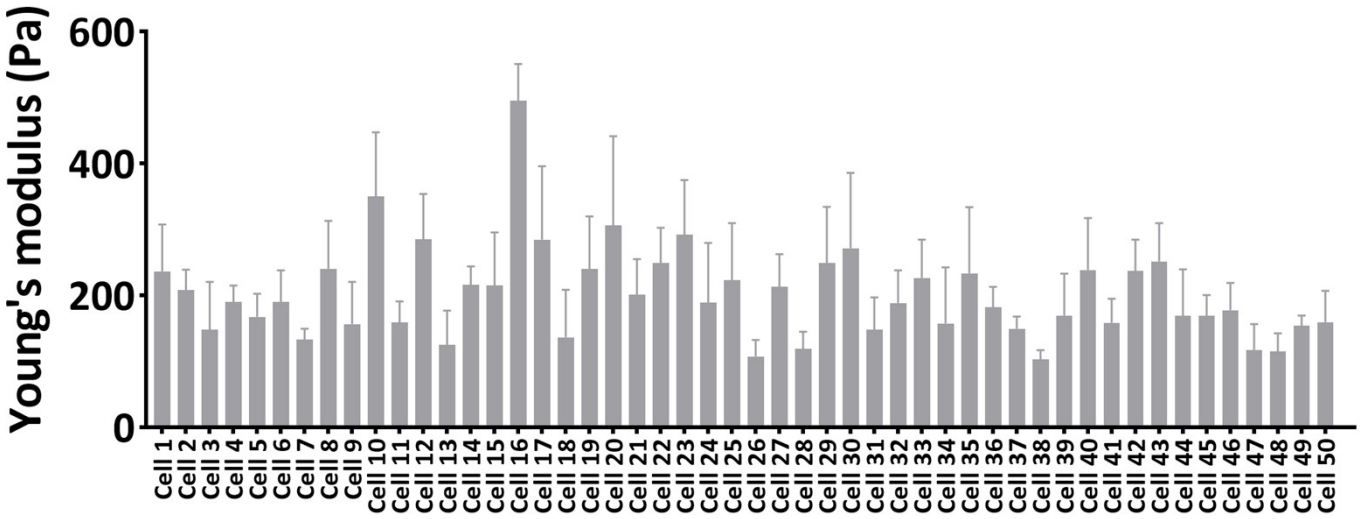


Figure S12 Histograms of the measured Young's modulus of U87 cells isolated from the blood.

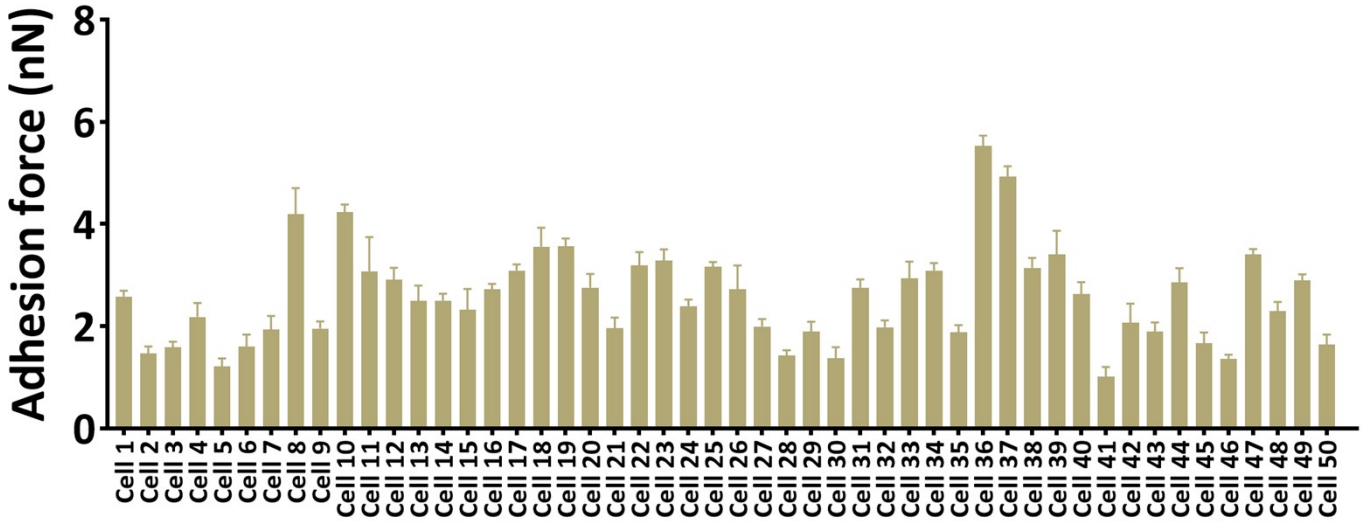


Figure S13 Histograms of the measured adhesion forces of U87 cells isolated from the blood.

3. Source codes

The source codes have been uploaded to the GitHub website and are publicly available (https://github.com/cnncell/Combining-Faster-R-CNN-with-U-Net.git). The main codes are following.

3.1 Environmental requirements

torch == 1.2.0

torchvision

tensorboard

scipy==1.2.1

numpy==1.17.0

matplotlib==3.1.2

opencv_python==4.1.2.30

tqdm==4.60.0

Pillow==8.2.0

h5py==2.10.0

3.2 Documentation about the source codes

Training and testing

1. Training datasets.

(1) Make your own dataset by labelimg, put the original image in VOCdevkitVOC2007JPEGImages, and put the label file in VOCdevkitVOC2007Annotations, run the voc_annotation.py to distinguish between the training set and the test set, and finally run the train_faster-rcnn.py training.

(2) Run crop.py and select Batch Crop Images in the frnn.py, and use faster-rcnn to crop out all the cells in one folder in batches.

(3)Use labelme to label the cropped cells, then put them in the datasetsbefore folder, run json_to_dataset.py to change the labels in jion format to png format, and store the original image in datasetsJPEGImages, and store the label file in datasetsSegmentationClass. Transfer JPEGImages and SegmentationClass to VOCdevkit_unetVOC2007, run voc_annotation_u-net.py divide the training set and dataset, and run the train_u-net.py to train the U-Net model.

2. Perform Cell image detection

predict.py run, enter img/1.jpg

Additional Notes

1.The trained file will be saved in logs/, remember to change the file name to the corresponding file name during the test

2.The part where u-net is embedded in faster-rcnn is reflected in the frcnn.py

3.The combination of template matching and combined networks is reflected in the frcnn.py

Reference

https://github.com/chenyuntc/simple-faster-rcnn-pytorch

https://github.com/ggyyzm/pytorch_segmentation

https://github.com/bubbliiiing/faster-rcnn-pytorch/tree/bilibili

https://github.com/bubbliiiing/unet-pytorch/tree/bilibili

3.3 The Faster R-CNN neural network and the codes to recognize cells by combining Faster R-CNN and U-Net

import colorsys

```
import os
import time
import math
import cv2
import numpy as np
import torch
import torch.nn as nn
from PIL import Image, ImageDraw, ImageFont, ImageEnhance

from nets.frcnn import FasterRCNN
from utils.utils import (cvtColor, get_classes, get_new_img_size, resize_image,
                         preprocess_input, show_config)
from utils.utils_bbox import DecodeBox
import csv
from unet import Unet
import pandas as pd
import random
unet = Unet()
name_classes        = ["background","u87"]

#--------------------------------------------------------------------------------------#
#To predict with your own trained model, you need to modify 2 parameters:
#model_path and classes_path both need to be changed!
#--------------------------------------------------------------------------------------#
class FRCNN(object):
    _defaults = {
        #----------------------------------------------------------------------------------------------------------------------------#
        #   Be sure to modify the model_path and classes_path when using your own trained model for prediction!
        #   The model_path should point to the weight file in the logs folder, and the classes_path should point to the txt file in the model_data folder.
        #----------------------------------------------------------------------------------------------------------------------------#
        "model_path"        : 'logs/Faster_rcnn.pth.pth',
        "classes_path"    : 'model_data/new_classes.txt',
        #---------------------------------------------------------------------#
        #   The main body of the network for feature extraction, ResNet50 or VGG.
        #---------------------------------------------------------------------#
        "backbone"          : "resnet50",
        #---------------------------------------------------------------------#
        #   Prediction boxes with scores higher than the confidence threshold will be retained.
```

```
        #---------------------------------------------------------------------------------------------------------------#
        "confidence"        : 0.9,
        #---------------------------------------------------------------------------------------------------------------#
        #       The size of the nms_iou used in Non-Maximum Suppression (NMS).
        #---------------------------------------------------------------------------------------------------------------#
        "nms_iou"           : 0.03,
        #---------------------------------------------------------------------------------------------------------------#
        #       Used to specify the size of the anchor boxes.
        #---------------------------------------------------------------------------------------------------------------#
        'anchors_size'    : [8, 16, 32],
        #-------------------------------------------------------------------------#
        #       Whether to use Cuda for translation.
        #-------------------------------------------------------------------------#
        "cuda"              : True,
        #-------------------------------#
        #    Crop and save in bulk : 0
        #    Save only one         : 1
        #-------------------------------#
        "Batch_cropping"    : 0,
}

@classmethod
def get_defaults(cls, n):
        if n in cls._defaults:
                return cls._defaults[n]
        else:
                return "Unrecognized attribute name '" + n + "'"

#---------------------------------------------------#
#       initialization faster RCNN
#---------------------------------------------------#
def __init__(self, **kwargs):
        self.__dict__.update(self._defaults)
        for name, value in kwargs.items():
                setattr(self, name, value)
                self._defaults[name] = value

        self.class_names, self.num_classes    = get_classes(self.classes_path)

        self.std        = torch.Tensor([0.1, 0.1, 0.2, 0.2]).repeat(self.num_classes + 1)[None]
        if self.cuda:
```

```
            self.std        = self.std.cuda()
        self.bbox_util    = DecodeBox(self.std, self.num_classes)

        #---------------------------------------------------#
        #     Set different colors for drawing boxes
        #---------------------------------------------------#
        hsv_tuples = [(x / self.num_classes, 1., 1.) for x in range(self.num_classes)]
        self.colors = list(map(lambda x: colorsys.hsv_to_rgb(*x), hsv_tuples))
        self.colors = list(map(lambda x: (int(x[0] * 255), int(x[1] * 255), int(x[2] * 255)), self.colors))
        self.generate()

        show_config(**self._defaults)

    #---------------------------------------------------#
    #   Loading a model
    #---------------------------------------------------#
    def generate(self):

        self.net       = FasterRCNN(self.num_classes, "predict", anchor_scales = self.anchors_size, backbone = self.backbone)
        device           = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
        self.net.load_state_dict(torch.load(self.model_path, map_location=device))
        self.net       = self.net.eval()
        print('{} model, anchors, and classes loaded.'.format(self.model_path))

        if self.cuda:
            self.net = nn.DataParallel(self.net)
            self.net = self.net.cuda()
    #---------------------------------------------------#
    #     Detect images
    #---------------------------------------------------#
    def detect_image(self, image, crop = False, count = False):

        image_shape = np.array(np.shape(image)[0:2])

        input_shape = get_new_img_size(image_shape[0], image_shape[1])

        image           = cvtColor(image)

        image_data    = resize_image(image, [input_shape[1], input_shape[0]])
#
        image_data    = np.expand_dims(np.transpose(preprocess_input(np.array(image_data, dtype='float32')), (2, 0, 1)), 0)
```

```
        with torch.no_grad():
            images = torch.from_numpy(image_data)
            if self.cuda:
                images = images.cuda()



            roi_cls_locs, roi_scores, rois, _ = self.net(images)

            results = self.bbox_util.forward(roi_cls_locs, roi_scores, rois, image_shape, input_shape,
                                                        nms_iou = self.nms_iou, confidence = self.confidence)

            if len(results[0]) <= 0:
                return image

            top_label   = np.array(results[0][:, 5], dtype = 'int32')
            top_conf    = results[0][:, 4]
            top_boxes   = results[0][:, :4]

        #---------------------------------------------------------#
        #   Set font and border thickness
        #---------------------------------------------------------#

        font        = ImageFont.truetype(font='model_data/simhei.ttf', size=np.floor(2.0e-2 * image.size[1] +
0.5).astype('int32'))
        thickness   = int(max((image.size[0] + image.size[1]) // np.mean(input_shape), 0.5))
        #---------------------------------------------------------#
        # count
        #---------------------------------------------------------#
        if count:
            print("top_label:", top_label)
            classes_nums    = np.zeros([self.num_classes])
            for i in range(self.num_classes):
                num = np.sum(top_label == i)
                if num > 0:
                    print(self.class_names[i], " : ", num)
                classes_nums[i] = num
            print("classes_nums:", classes_nums)
        #---------------------------------------------------------#
        #   Whether to crop the target
        #---------------------------------------------------------#
```

```
        if crop:
            mean_x = []
            mean_y = []
            for i, c in list(enumerate(top_label)):
                predicted_class = self.class_names[int(c)]
                if predicted_class == 'Sphere_Probe':
                    continue
                top, left, bottom, right = top_boxes[i]
                top = max(0, np.floor(top).astype('int32'))
                left = max(0, np.floor(left).astype('int32'))
                bottom = min(image.size[1], np.floor(bottom).astype('int32'))
                right = min(image.size[0], np.floor(right).astype('int32'))


                img_origin_crop = "img_origin_crop"
                if not os.path.exists(img_origin_crop):
                    os.makedirs(img_origin_crop)

                dir_uimage_save_path = "dir_uimage_save_path"
                if not os.path.exists(dir_uimage_save_path):
                    os.makedirs(dir_uimage_save_path)



                if predicted_class == 'u87':
                    #--------------------------------------------------------------------------------------------------------#
                    # Whether to crop the dataset in batches
                    #--------------------------------------------------------------------------------------------------------#
                    if self.Batch_cropping == 0:
                        random_integer = random.randint(1, 100000000000)
                        a=random_integer
                        crop_image_origin = image.crop([left, top, right, bottom])
                        crop_image_origin.save(os.path.join(img_origin_crop, "crop_" +str(a)+ "_"+str(i) + ".jpg"),
quality=95, subsampling=0)
                    elif self.Batch_cropping ==1:
                        crop_image_origin = image.crop([left, top, right, bottom])
                        crop_image_origin.save(os.path.join(img_origin_crop, "crop_"+ "_"+str(i) + ".jpg"), quality=95,
subsampling=0)
#--------------------------------------------------------------------------------------------------------#
#                                       Embed U-Net network
#--------------------------------------------------------------------------------------------------------#
```

```
                    uimage = unet.detect_image(crop_image_origin,name_classes=name_classes)
                    uimage_32 = np.float32(uimage)
                    uimage = np.uint8(uimage_32)
                    cv2.imwrite(os.path.join(dir_uimage_save_path, "crop_" + str(i) + ".png"), uimage)
                    print("save uimage crop_" + str(i) + ".png to " + dir_uimage_save_path)
#-------------------------------------------------------------------------------------------------------#
#                                       Calculate the centroid of the cell
#-------------------------------------------------------------------------------------------------------#

                    crop_image = cv2.cvtColor(uimage, cv2.COLOR_BGR2GRAY)
                    _, binary_img = cv2.threshold(crop_image, 200, 255, cv2.THRESH_BINARY)
                    contours,_=cv2.findContours(binary_img,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)

                    img_pil = Image.fromarray(cv2.cvtColor(binary_img, cv2.COLOR_BGR2RGB))
                    draw = ImageDraw.Draw(img_pil)

                    for i, cnt in enumerate(contours):

                        M = cv2.moments(cnt)

                        if M["m00"] != 0:
                            mean_xs = int(M["m10"] / M["m00"])
                            mean_ys = int(M["m01"] / M["m00"])

                    mean_x.append(mean_xs)
                    mean_y.append(mean_ys)
                    print('mean_x',mean_x)
                    print('mean_y',mean_y)

                else:
                    center_x_ball = (right - left) / 2
                    center_y_ball = (bottom - top) / 2
                    mean_y.append(center_y_ball)
                    mean_x.append(center_x_ball)
                    continue


            # ----------------------------------------------------------#
            #     Image drawing
            # ----------------------------------------------------------#
```

```
            list_class=[]
            list_x = []
            list_y = []
            list_class.clear()
            list_x.clear()
            list_y.clear()


            for i, c in list(enumerate(top_label)):
                predicted_class = self.class_names[int(c)]
                box = top_boxes[i]
                score = top_conf[i]

                top, left, bottom, right = box

                top = max(0, np.floor(top).astype('int32'))
                left = max(0, np.floor(left).astype('int32'))
                bottom = min(image.size[1], np.floor(bottom).astype('int32'))
                right = min(image.size[0], np.floor(right).astype('int32'))



            # ---------------------------------------------------------#
            #     The position of the cell center in the original image
            # ---------------------------------------------------------#

                center_x = mean_x[i]+left#
                center_y = mean_y[i]+top#
                print('center_x',center_x)
                print('center_y',center_y)
#-------------------------------------------------------------------------------------------------------------#
#           Obtain relative coordinates from the coordinates acquired through template matching
#-------------------------------------------------------------------------------------------------------------#
                df = pd.read_csv('center_coordinates.csv')
                tx = df['x'].astype(int)
                ty = df['y'].astype(int)
                x1 = -(center_x-tx)
                y1 = -(center_y-ty)

                x1 = np.around(x1, decimals=1).item()
```

```
                y1 = np.around(y1, decimals=1).item()
#                 label = f'{predicted_class}:{score:.2f}    {"num:"} {i} \n {"Relative:"} {(y1,x1)}'
                label = f''

                label_number = '{}'.format(i)
                class_detect = '{} '.format(predicted_class)
                draw = ImageDraw.Draw(image)
                label_size = draw.textsize(label, font)
                number_size = draw.textsize(label_number, font)

                label = label.encode('utf-8')

                if top - label_size[1] >= 0:
                        text_origin = np.array([left, top - label_size[1]])
                else:
                        text_origin = np.array([left, top + 1])

                if top - number_size[0] >= 0:
                        number_s = np.array([left, top - number_size[0]-16])#4
                else:
                        number_s = np.array([left, top + 1])

                for i in range(thickness):
                        draw.rectangle([left + i, top + i, right - i, bottom - i], outline=self.colors[c])



                class_data = 'Cell class：{}'.format(class_detect)
                list_class.append(class_data)

                for i in range(thickness):
                        draw.rectangle([center_x + 3, center_y + 15, center_x - 3, center_y - 15], fill=self.colors[c],
outline=self.colors[c])
                        draw.rectangle([center_x + 15, center_y + 3, center_x - 15, center_y - 3],
                                        fill=self.colors[c], outline=self.colors[c])
                        draw.rectangle([left + i, top + i, right - i, bottom - i], outline=self.colors[c])
#
                blue_color = (0, 0, 255)
                green_color = (0, 255, 0)
                draw.rectangle([tx + 5, ty + 20, tx - 5, ty - 20], fill=self.colors[c], outline=blue_color)
                draw.rectangle([tx + 20, ty+ 5, tx - 20, ty- 5],
```

```
                                        fill=self.colors[c], outline=blue_color)
            draw.rectangle([tx + 40, ty+ 40, tx - 40, ty- 40], outline=green_color)
            draw.rectangle([tx + 41, ty+ 41, tx - 41, ty- 41], outline=green_color)
            draw.rectangle([tx + 42, ty+ 42, tx - 42, ty- 42], outline=green_color)
            draw.rectangle([tx + 43, ty+ 43, tx - 43, ty- 43], outline=green_color)

            draw.rectangle([tuple(text_origin), tuple(text_origin + label_size)], fill=self.colors[c])
            draw.text(text_origin, str(label,'UTF-8'), fill=(0, 0, 0), font=font)
            del draw

        return image, top_label, count


def get_FPS(self, image, test_interval):

        image_shape = np.array(np.shape(image)[0:2])
        input_shape = get_new_img_size(image_shape[0], image_shape[1])

        image               = cvtColor(image)

        image_data    = resize_image(image, [input_shape[1], input_shape[0]])

        image_data    = np.expand_dims(np.transpose(preprocess_input(np.array(image_data, dtype='float32')), (2, 0, 1)), 0)

        with torch.no_grad():
            images = torch.from_numpy(image_data)
            if self.cuda:
                images = images.cuda()

            roi_cls_locs, roi_scores, rois, _ = self.net(images)

            results = self.bbox_util.forward(roi_cls_locs, roi_scores, rois, image_shape, input_shape,
                                                        nms_iou = self.nms_iou, confidence = self.confidence)
        t1 = time.time()
        for _ in range(test_interval):
            with torch.no_grad():
                roi_cls_locs, roi_scores, rois, _ = self.net(images)

                results = self.bbox_util.forward(roi_cls_locs, roi_scores, rois, image_shape, input_shape,
                                                            nms_iou = self.nms_iou, confidence = self.confidence)
```

```
        t2 = time.time()
        tact_time = (t2 - t1) / test_interval
        return tact_time

    #---------------------------------------------------#
    #   Detect the picture
    #---------------------------------------------------#
    def get_map_txt(self, image_id, image, class_names, map_out_path):
        f = open(os.path.join(map_out_path, "detection-results/"+image_id+".txt"),"w")

        image_shape = np.array(np.shape(image)[0:2])
        input_shape = get_new_img_size(image_shape[0], image_shape[1])

        image       = cvtColor(image)

        image_data  = resize_image(image, [input_shape[1], input_shape[0]])

        image_data  = np.expand_dims(np.transpose(preprocess_input(np.array(image_data, dtype='float32')), (2, 0, 1)), 0)

        with torch.no_grad():
            images = torch.from_numpy(image_data)
            if self.cuda:
                images = images.cuda()

            roi_cls_locs, roi_scores, rois, _ = self.net(images)

            results = self.bbox_util.forward(roi_cls_locs, roi_scores, rois, image_shape, input_shape,
                                             nms_iou = self.nms_iou, confidence = self.confidence)

            if len(results[0]) <= 0:
                return

            top_label   = np.array(results[0][:, 5], dtype = 'int32')
            top_conf    = results[0][:, 4]
            top_boxes   = results[0][:, :4]

        for i, c in list(enumerate(top_label)):
            predicted_class = self.class_names[int(c)]
            box             = top_boxes[i]
            score           = str(top_conf[i])
```

```
            top, left, bottom, right = box
            if predicted_class not in class_names:
                continue

            f.write("%s %s %s %s %s %s\n" % (predicted_class, score[:6], str(int(left)), str(int(top)),
str(int(right)),str(int(bottom))))

        f.close()
        return
```

3.4 The U-Net neural network

```
import colorsys
import copy
import time

import cv2
import numpy as np
import torch
import torch.nn.functional as F

from torch import nn
from PIL import Image, ImageDraw
from nets.unet import Unet as unet
from utils.utils_unet import cvtColor, preprocess_input, resize_image, show_config



class Unet(object):
    _defaults = {

        #--------------------------------------------------------------------#
        # model_path point to the metric file in the logs folder
        #--------------------------------------------------------------------#
        "model_path"        : 'logs/u-net.pth',
        #--------------------------------------------------------------------#
        # The number of classes to be distinguished +1
        #--------------------------------------------------------------------#
```

```
        "num_classes"      : 2,
        #--------------------------------------------------------------------#
        # The backbone network used: VGG
        #--------------------------------------------------------------------#
        "backbone"          : "vgg",
        #---------------------------------------------------------------------- #
        # The backbone network used: VGG and resnet50
        #---------------------------------------------------------------------- #
        "input_shape"      : [96, 96],
        #-------------------------------------------------------------------------------------------------------------#
        # mix_type parameters are used to control how the inspection results are visualized
        # mix_type = 0 means that the original image is mixed with the generated image
        # mix_type = 1 means that only the generated graph is retained
        # mix_type = 2 means that only the background is removed, and only the target in the original image is retained
        #-------------------------------------------------------------------------------------------------------------#
        "mix_type"          : 1,
        #-------------------------------------------------------------------------#
        # Whether to use Cuda
        # If you don't have a GPU, you can set it to False
        #-------------------------------------------------------------------------#
        "cuda"                  : True,
}

#---------------------------------------------------#
# Initialize UNET
#---------------------------------------------------#
def __init__(self, **kwargs):
        self.__dict__.update(self._defaults)
        for name, value in kwargs.items():
                setattr(self, name, value)
        #---------------------------------------------------#
        # Set the frame to different colors
        #---------------------------------------------------#
        if self.num_classes <= 21:
                self.colors = [ (0, 0, 0), (255, 255, 255), (0, 128, 0), (128, 128, 0), (0, 0, 128), (128, 0, 128), (0, 128, 128),
                                        (128, 128, 128), (64, 0, 0), (192, 0, 0), (64, 128, 0), (192, 128, 0), (64, 0, 128), (192, 0, 128),
                                        (64, 128, 128), (192, 128, 128), (0, 64, 0), (128, 64, 0), (0, 192, 0), (128, 192, 0), (0, 64, 128),

                                        (128, 64, 12)]
        else:
                hsv_tuples = [(x / self.num_classes, 1., 1.) for x in range(self.num_classes)]
```

```
            self.colors = list(map(lambda x: colorsys.hsv_to_rgb(*x), hsv_tuples))
            self.colors = list(map(lambda x: (int(x[0] * 255), int(x[1] * 255), int(x[2] * 255)), self.colors))

        self.generate()

        show_config(**self._defaults)

#---------------------------------------------------#
#   Get all the classifications
#---------------------------------------------------#
def generate(self, onnx=False):
        self.net = unet(num_classes = self.num_classes, backbone=self.backbone)

        device          = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
        self.net.load_state_dict(torch.load(self.model_path, map_location=device))
        self.net        = self.net.eval()
        print('{} model, and classes loaded.'.format(self.model_path))
        if not onnx:
            if self.cuda:
                self.net = nn.DataParallel(self.net)
                self.net = self.net.cuda()

#---------------------------------------------------#
#   Detect the picture
#---------------------------------------------------#
def detect_image(self, image, count=False, name_classes=None):

        image           = cvtColor(image)

        old_img         = copy.deepcopy(image)
        orininal_h    = np.array(image).shape[0]
        orininal_w    = np.array(image).shape[1]

        image_data, nw, nh    = resize_image(image, (self.input_shape[1],self.input_shape[0]))

        image_data    = np.expand_dims(np.transpose(preprocess_input(np.array(image_data, np.float32)), (2, 0, 1)), 0)

        with torch.no_grad():
            images = torch.from_numpy(image_data)
            if self.cuda:
                images = images.cuda()
```

```
        pr = self.net(images)[0]

        pr = F.softmax(pr.permute(1,2,0),dim = -1).cpu().numpy()

        pr = pr[int((self.input_shape[0] - nh) // 2) : int((self.input_shape[0] - nh) // 2 + nh), \
                int((self.input_shape[1] - nw) // 2) : int((self.input_shape[1] - nw) // 2 + nw)]

        pr = cv2.resize(pr, (orininal_w, orininal_h), interpolation = cv2.INTER_LINEAR)

        pr = pr.argmax(axis=-1)

    #---------------------------------------------------------#
    #   count
    #---------------------------------------------------------#
    if count:
        classes_nums        = np.zeros([self.num_classes])
        total_points_num    = orininal_h * orininal_w
        print('-' * 63)
        print("|%25s | %15s | %15s|"%("Key", "Value", "Ratio"))
        print('-' * 63)
        for i in range(self.num_classes):
            num     = np.sum(pr == i)
            ratio   = num / total_points_num * 100
            if num > 0:
                print("|%25s | %15s | %14.2f%%|"%(str(name_classes[i]), str(num), ratio))
                print('-' * 63)
            classes_nums[i] = num
        print("classes_nums:", classes_nums)

    if self.mix_type == 0:
        seg_img = np.reshape(np.array(self.colors, np.uint8)[np.reshape(pr, [-1])], [orininal_h, orininal_w, -1])
        #------------------------------------------#
        # Calculate the centroid of the cell
        #------------------------------------------#
        image   = Image.fromarray(np.uint8(seg_img))
        uimage_32 = np.float32(image)
        uimage = np.uint8(uimage_32)
        crop_image = cv2.cvtColor(uimage, cv2.COLOR_BGR2GRAY)
        _, binary_img = cv2.threshold(crop_image, 127, 255, cv2.THRESH_BINARY)
```

```
            contours, _ = cv2.findContours(binary_img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
            img_pil = Image.fromarray(cv2.cvtColor(binary_img, cv2.COLOR_BGR2RGB))
            draw = ImageDraw.Draw(old_img)
            for cnt in contours:
                M = cv2.moments(cnt)
                if M["m00"] != 0:
                    cX = int(M["m10"] / M["m00"])
                    cY = int(M["m01"] / M["m00"])
                    color = (255, 0, 0)
                    draw.ellipse([cX-2, cY-2, cX+2, cY+2], outline='red')
                    draw.rectangle([cX + 3, cY + 15, cX - 3, cY - 15], fill=color, outline=color)
                    draw.rectangle([cX + 15, cY + 3, cX - 15, cY - 3],
                                   fill=color, outline=color)
            image     = Image.blend(old_img, img_pil, 0.2)
        elif self.mix_type == 1:
            seg_img = np.reshape(np.array(self.colors, np.uint8)[np.reshape(pr, [-1])], [orininal_h, orininal_w, -1])
            image     = Image.fromarray(np.uint8(seg_img))

        elif self.mix_type == 2:
            seg_img = (np.expand_dims(pr != 0, -1) * np.array(old_img, np.float32)).astype('uint8')
            image = Image.fromarray(np.uint8(seg_img))




        return image

    def get_FPS(self, image, test_interval):

        image           = cvtColor(image)

        image_data, nw, nh  = resize_image(image, (self.input_shape[1],self.input_shape[0]))

        image_data  = np.expand_dims(np.transpose(preprocess_input(np.array(image_data, np.float32)), (2, 0, 1)), 0)

        with torch.no_grad():
            images = torch.from_numpy(image_data)
            if self.cuda:
                images = images.cuda()
```

```
            pr = self.net(images)[0]

            pr = F.softmax(pr.permute(1,2,0),dim = -1).cpu().numpy().argmax(axis=-1)

            pr = pr[int((self.input_shape[0] - nh) // 2) : int((self.input_shape[0] - nh) // 2 + nh), \
                    int((self.input_shape[1] - nw) // 2) : int((self.input_shape[1] - nw) // 2 + nw)]

        t1 = time.time()
        for _ in range(test_interval):
            with torch.no_grad():

                pr = self.net(images)[0]

                pr = F.softmax(pr.permute(1,2,0),dim = -1).cpu().numpy().argmax(axis=-1)

                pr = pr[int((self.input_shape[0] - nh) // 2) : int((self.input_shape[0] - nh) // 2 + nh), \
                        int((self.input_shape[1] - nw) // 2) : int((self.input_shape[1] - nw) // 2 + nw)]
        t2 = time.time()
        tact_time = (t2 - t1) / test_interval
        return tact_time

    def convert_to_onnx(self, simplify, model_path):
        import onnx
        self.generate(onnx=True)

        im                  = torch.zeros(1, 3, *self.input_shape).to('cpu')  # image size(1, 3, 512, 512) BCHW
        input_layer_names   = ["images"]
        output_layer_names  = ["output"]

        # Export the model
        print(f'Starting export with onnx {onnx.__version__}.')
        torch.onnx.export(self.net,
                          im,
                          f                 = model_path,
                          verbose           = False,
                          opset_version     = 12,
                          training          = torch.onnx.TrainingMode.EVAL,
                          do_constant_folding = True,
                          input_names       = input_layer_names,
                          output_names      = output_layer_names,
                          dynamic_axes      = None)
```

```
        # Checks
        model_onnx = onnx.load(model_path)  # load onnx model
        onnx.checker.check_model(model_onnx)  # check onnx model

        # Simplify onnx
        if simplify:
            import onnxsim
            print(f'Simplifying with onnx-simplifier {onnxsim.__version__}.')
            model_onnx, check = onnxsim.simplify(
                model_onnx,
                dynamic_input_shape=False,
                input_shapes=None)
            assert check, 'assert check failed'
            onnx.save(model_onnx, model_path)

        print('Onnx model save as {}'.format(model_path))

    def get_miou_png(self, image):

        image           = cvtColor(image)
        orininal_h  = np.array(image).shape[0]
        orininal_w  = np.array(image).shape[1]

        image_data, nw, nh  = resize_image(image, (self.input_shape[1],self.input_shape[0]))

        image_data  = np.expand_dims(np.transpose(preprocess_input(np.array(image_data, np.float32)), (2, 0, 1)), 0)

        with torch.no_grad():
            images = torch.from_numpy(image_data)
            if self.cuda:
                images = images.cuda()


            pr = self.net(images)[0]

            pr = F.softmax(pr.permute(1,2,0),dim = -1).cpu().numpy()

            pr = pr[int((self.input_shape[0] - nh) // 2) : int((self.input_shape[0] - nh) // 2 + nh), \
                    int((self.input_shape[1] - nw) // 2) : int((self.input_shape[1] - nw) // 2 + nw)]
```

```
            pr = cv2.resize(pr, (orininal_w, orininal_h), interpolation = cv2.INTER_LINEAR)

            pr = pr.argmax(axis=-1)

        image = Image.fromarray(np.uint8(pr))
        return image
```

3.5 AFM script program of automated mechanics

```
# Python code for ExperimentPlanner
import time
import os
import csv
checkVersion('SPM', 7, 0, 178)

file_path = '/media/jpkuser/mediator/output_table.csv'
filename = '/media/jpkuser/mediator/1.tif'

def wait_for_file_path(file_path):
    while not os.path.exists(file_path):
        print("waiting for document.....")


n = 0
while n < 41:
            #Coordinates of the next area
            nextAreaX = 0
            nextAreaY = 0
            temp = 0

            #Number of detectable cells in the area
            coordinate_count = 0
            file_path = '/media/jpkuser/mediator/output_table.csv'
            wait_for_file_path(file_path)


            #Read coordinate points within probe detection range into script
            def addposition(x,y):
                ForceSpectroscopy.addPosition(x, y)

            #Disabled platform moves
            MotorizedStage.disengage()
```

```
#Clear coordinates, read new coordinates
ForceSpectroscopy.clearPositions()
#init

#Add initial position to software table, set to origin, index 0
ForceSpectroscopy.addPosition(0, 0)

#Read the coordinates of the detectable point and the center point of the next area into the script
with open(file_path, mode='r') as file:
    reader = csv.reader(file)
    next(reader)
    for row in reader:
        x = float(row[0])
        y = float(row[1])
        if abs(x) < 5e-5 and abs(y) < 5e-5 :
            addposition(x, y)
            coordinate_count += 1
        #else :
            nextAreaX = x
            nextAreaY = y

#Probe tip moves back to initial position
ForceSpectroscopy.moveToForcePositionIndex(0)

#Get force curves for all detectable cells and save them automatically
i = 0
for j in xrange(coordinate_count):
    i+=1
    ForceSpectroscopy.moveToForcePositionIndex(i)

    Scanner.approach()
    ForceSpectroscopy.startScanning(25)
    Scanner.retractPiezo()
    Scanner.moveMotorsUp(2e-5)
    time.sleep(1.0)

#After detecting each area, the probe must return to its original position
ForceSpectroscopy.moveToForcePositionIndex(0)
temp += 1
#Enabling the platform
```

```
MotorizedStage.engage()
#Move the platform to the next regional center
MotorizedStage.moveToRelativePosition(nextAreaX, nextAreaY)
nextAreaX = 0
nextAreaY = 0

#Disabled platform
MotorizedStage.disengage()

#Probe down and up
Scanner.approach()
time.sleep(1.0)
Snapshooter.saveOpticalSnapshot(filename)

Scanner.retract()
os.remove(file_path)
```