

Supporting Information for

A Deep-Learning Framework for the Automated Recognition of Molecules in Scanning-Probe-Microscopy Images

Zhiwen Zhu¹, Jiayi Lu¹, Fengru Zheng¹, Cheng Chen², Yang Lv², Hao Jiang¹, Yuyi Yan¹, Akimitsu Narita³, Klaus Müllen³, Xiao-Ye Wang², Qiang Sun^{1*}

¹Materials Genome Institute, Shanghai University, 200444 Shanghai, China

²State Key Laboratory of Elemento-Organic Chemistry, College of Chemistry, Nankai University, 300071 Tianjin, China

³Max Planck Institute for Polymer Research, 55128 Mainz, Germany

*E-mail: qiangsun@shu.edu.cn

Table of Contents

- 1. t-SNE**
- 2. Image augmentation techniques.**
- 3. Resnet-50 and FPN.**
- 4. More recognition cases.**
- 5. Model performances on testing images with different resolutions**
- 6. Non-maximum suppression.**
- 7. Precision, recall and mAP.**
- 8. User guideline.**
- 9. Materials and methods**
- 10. References**

1. t-SNE

On one hand, t-SNE in this project can be used to judge whether the clarity and separability of two molecules meet the standards of machine learning models. On the other hand, if outliers were found in the t-SNE results (possibly due to manual faults or poor image quality), they can be excluded when selecting the training set.

Before the t-SNE analysis, we manually crop the molecules from the STM image and rotate the image to have all the molecules with the same orientation, such that the distinction between the images of two molecules only comes from their STM appearances. Given that there may be slight molecular orientation deviations after manually rotating the images, a random rotation of 10° is added to all the molecular images to average the deviations. All the molecular images are subsequently resized to a uniform size and their image colors are converted to a single-channel grayscale representation and finally flattened into a data stream. The data stream is well adapted to the t-SNE algorithm.

In this project, we use the t-SNE algorithm built into the scikit-learn library. The code of our t-SNE analysis can be found on GitHub:

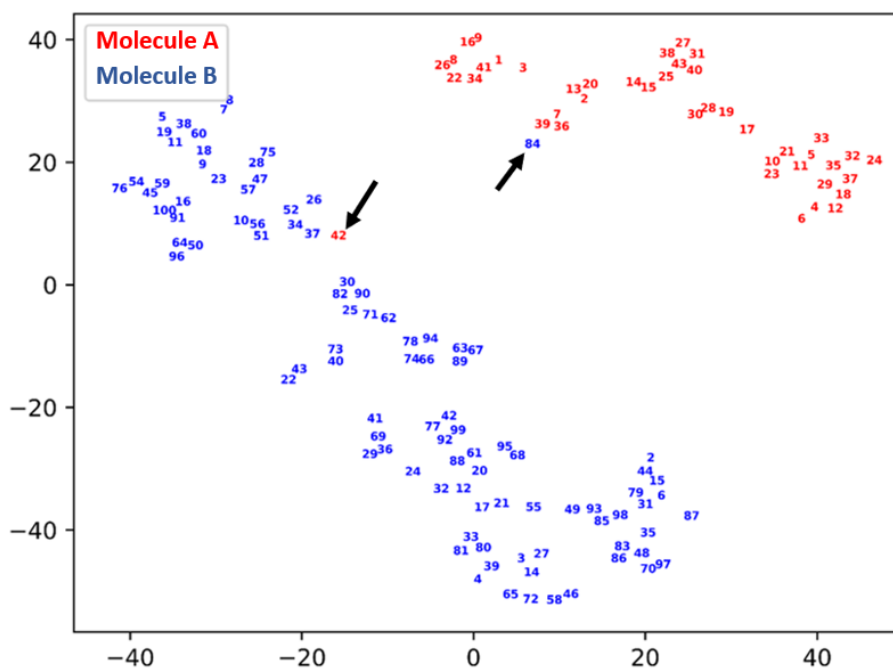
https://github.com/gggg0034/SPM_image_Mask-R-CNN/tree/master.

Table S1. Hyperparameters and values used in the t-SNE algorithm.

Hyperparameters	values
sklearn manifold.TSNE	N/A
N_components	2
Perplexity	10
Learning_rate	100
N_iter	2000
Angle	0.1
Others by default	N/A

In the process of the molecule assessment, we first choose a small area of the STM image and annotate the molecules manually. This requires human efforts to draw closed contours on the templating molecules and label them accordingly. t-SNE is used to identify the separability between two molecules in the STM images, but not used for molecular labeling. In other words, we rely on human classification to the training dataset.

Nevertheless, t-SNE provides important signs when the human classification contradicts the t-SNE results. For example, the t-SNE result shown below was acquired in the early stage of our experiment. The red and blue points represent the molecules A and molecules B, respectively. From the t-SNE plot, we clearly see two outliers as indicated by black arrows. There are two possibilities to lead to these outliers.



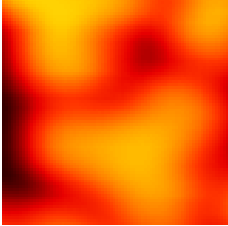
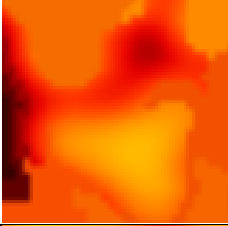
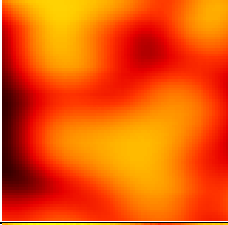
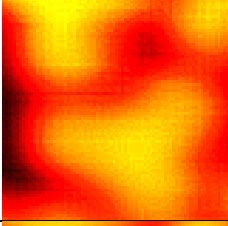
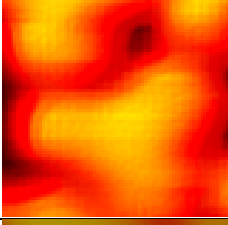
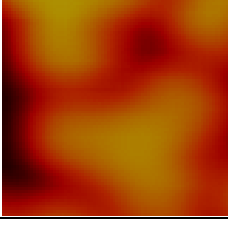
1. The STM probe was in a bad condition (tip shaking, tip touch, tip drift or noise) when scanning the molecules of RED NO.42 and BLUE NO.84, so that t-SNE algorithm failed on clustering the molecules.
2. Human eyes made mistakes, and we gave the algorithm wrong label data in the first place.

Usually, such ambiguous molecules will be avoided when we select images to generate the training dataset. Therefore, the t-SNE algorithm is efficient and indispensable, but the supervision by the domain knowledge of the human is also essential for dataset preparations.

2. Image augmentation techniques.

We use the Imgaug library for data augmentations. Some image augmentation techniques are difficult for human eyes to distinguish before and after the image augmentations. However, they are brand new image data for computers and contain useful information for machine learning models. In Table S2, we show the effects of the different image augmentation techniques we have applied in the work.

Table S2. Effects and descriptions of different image augmentation techniques.

Augmentation techniques	Image example	Description
None		The original image
Superpixels		Completely or partially transform images to their superpixel representation.
Blur		Blur an image by computing simple means over neighbourhoods or by computing median values over neighbourhoods.
Sharpen		Sharpen images and overlay the result with the original image.
Emboss		Emboss images and overlay the result with the original image.
EdgeDetect		Detect all edges in the images, mark them in a black-and-white image and then overlay the result with the original image.

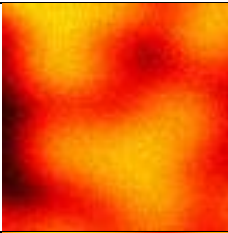
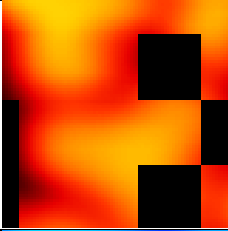
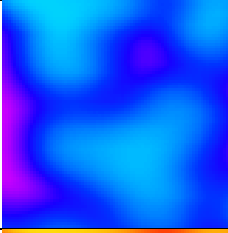
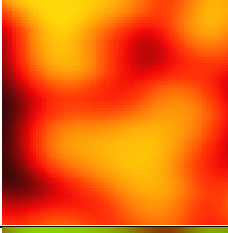
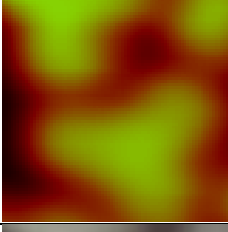
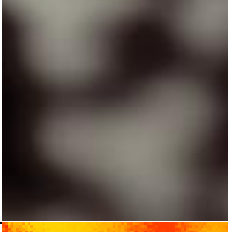
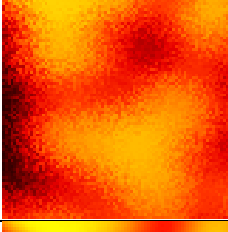
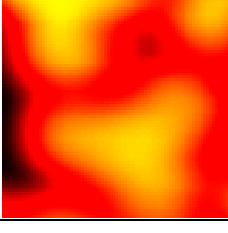
GaussianNoise		Add noises sampled from gaussian distributions elementwise to images.
Dropout		Set a certain fraction of pixels in images to zero.
Invert		Invert all values in images, i.e. sets a pixel from value x to $255 - x$.
Add		Add a certain value to all pixels in images.
Multiply		Multiply all pixels in an image with a specific value, thereby making the image darker or brighter.
Grayscale		Convert images to their grayscale versions.
Elastic		Transform images by moving pixels locally around using displacement fields.
Contrast		Adjust contrasts by scaling each pixel to $127 + \alpha * (x - 127)$.

Table S3. different augmentation techniques on the model performance. The following data augmentation methods are based on the completion of the L1 data augmentation technology. mAP is calculated with an IoU threshold of 0.5.

Augmentation techniques	recognition rates	mean average precision (mAP)
Add&Mutupiy	0.582	0.335
Blur	0.768	0.625
Elastic	0.806	0.767
EdgeDetect	0.778	0.716
Emboss	0.838	0.737
Sharpen	0.740	0.681
Contrast&Grayscale	0.693	0.514
Dropout&GaussianNoise&Emboss	0.860	0.842
Blur&Sharpen&Dropout&Elastic	0.851	0.825
Dropout&GaussianNoise&Emboss&Blur	0.848	0.827

3.Resnet-50 and FPN

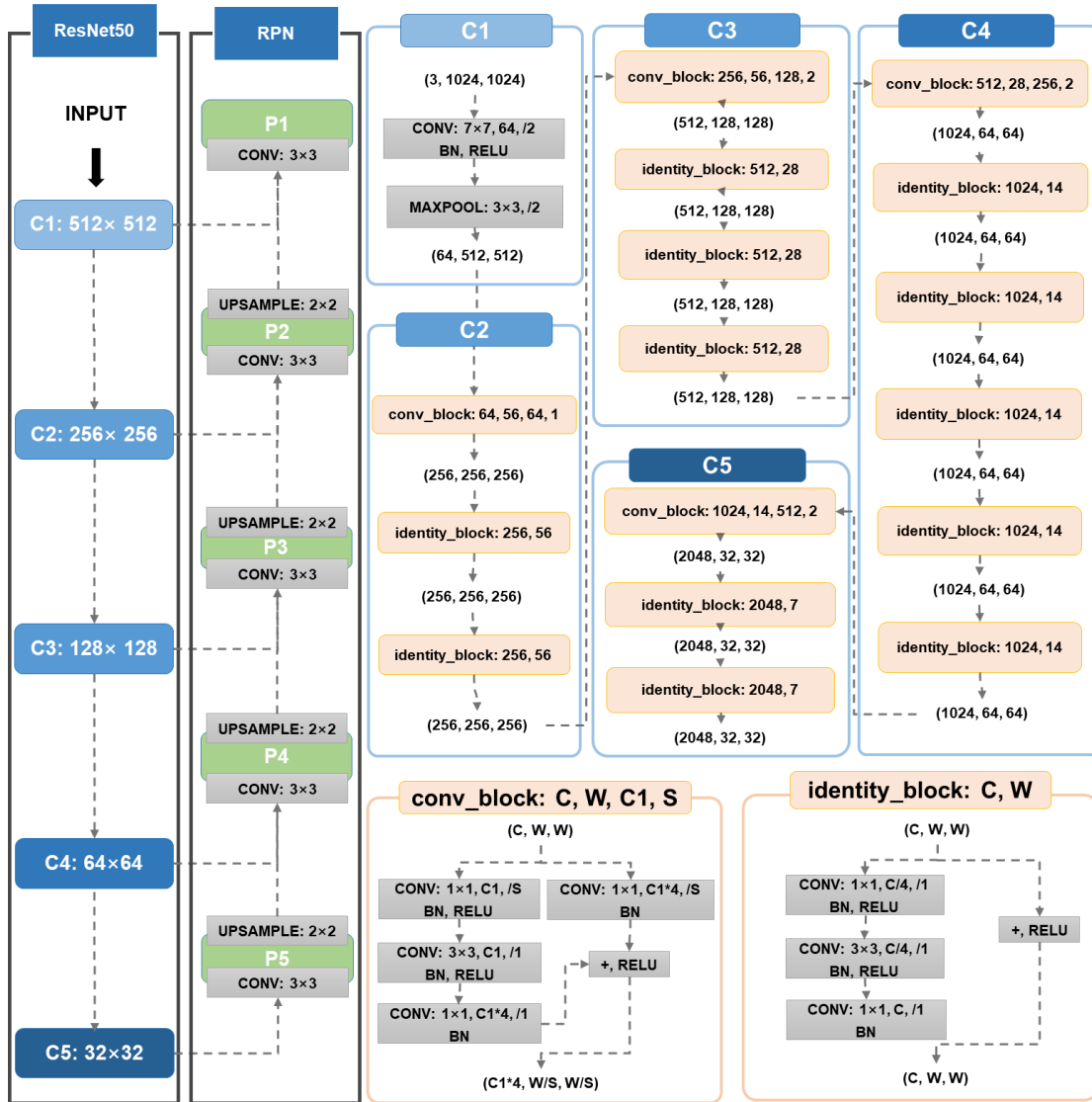
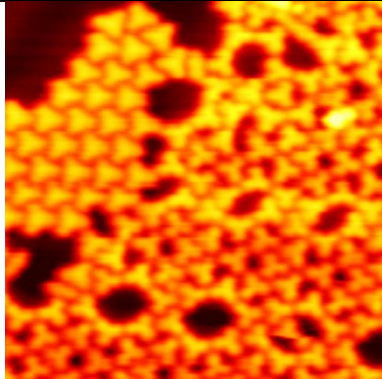
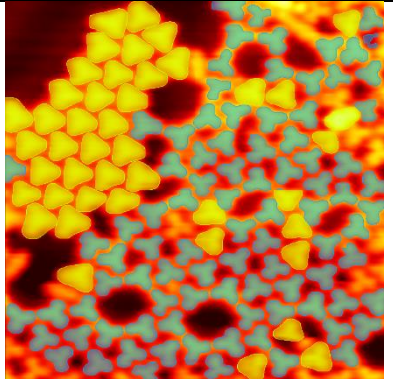
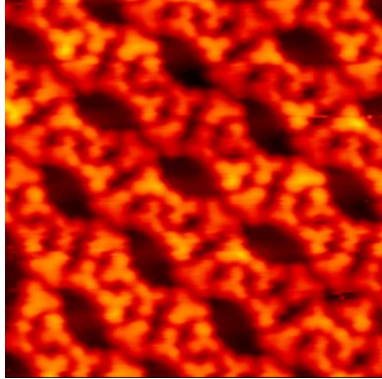
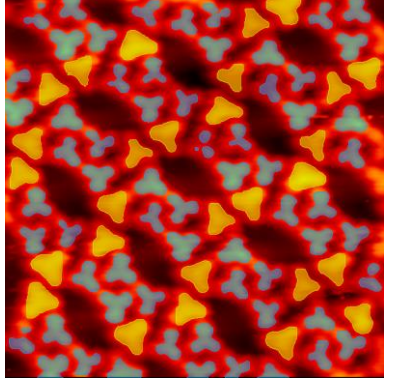
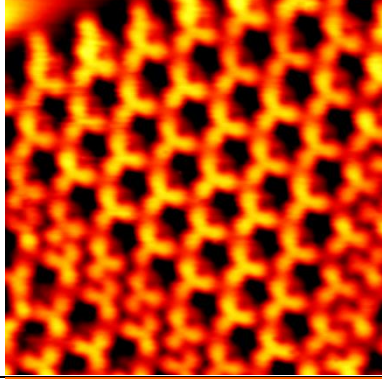
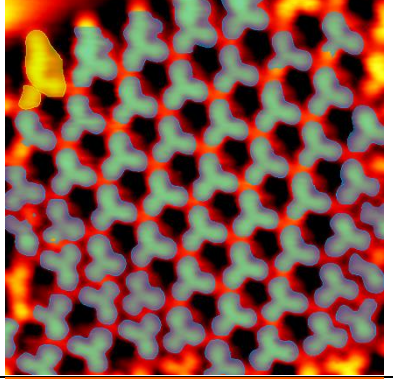
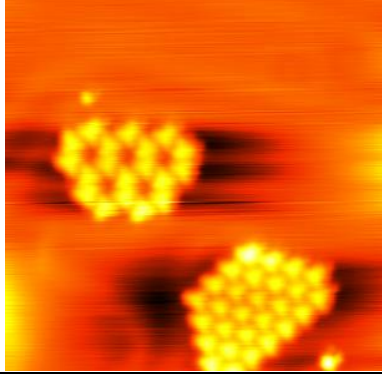
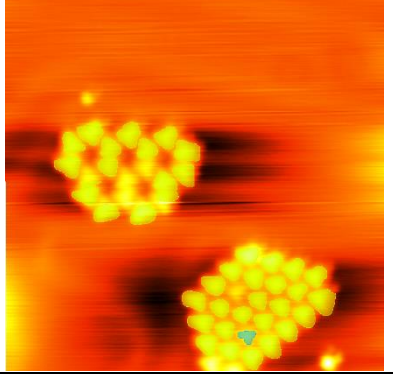
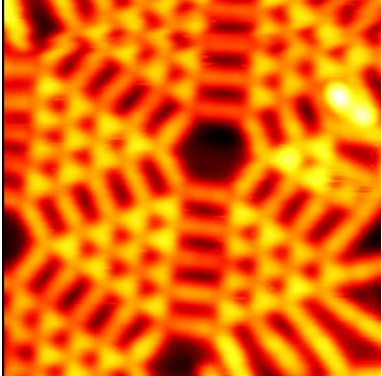
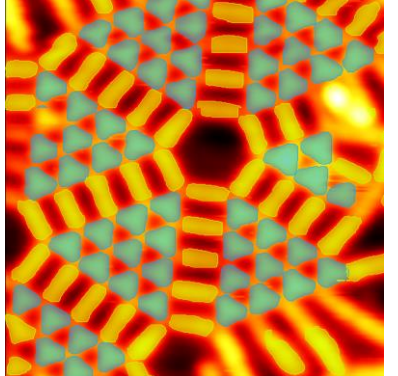
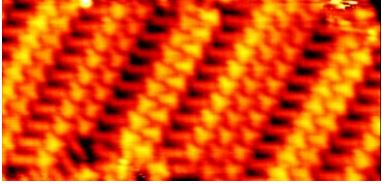
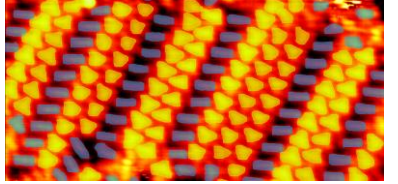
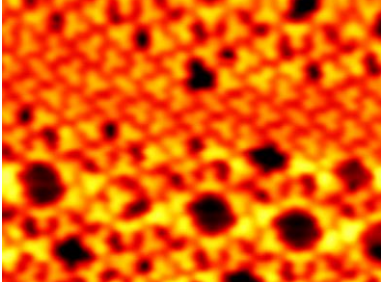
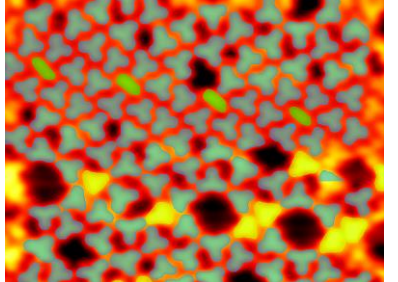


Figure S1. Structure of Resnet-50 and FPN. Sections with the same color have corresponding contents. Every block in gray corresponds to a specific operation of single-layer convolution, pooling or activation. The conv_block and identity_block consist (orange blocks) of several convolutional layers and activation functions. The five-layer structure of Resnet-50 consists of several conv_blocks and identity_blocks (except C1). The five-layer structure of FPN (green block) is obtained by upsampling the five-layer structure of resnet-50(blue block), superimposing one layer of feature maps, and then convolving one layer. For example, P3 is formed by upsampling P4 and stacking C3 with one more convolution.

4. More recognition cases.

Table S4. Model performances on STM images of other molecular nanostructures.

Molecular system	Original STM image	Output images
Binary structure with both molecules A (yellow) and B (blue)		
Binary structure with both molecules A (yellow) and B (blue) in images with system instability		
Self-assembly of molecule B (blue)		
Self-assembly of molecule A (yellow)		

<p>Binary structure with both molecule B (blue) and 4,4'-Di(Pyridin-4-Yl)-1,1'-Biphenyl (yellow)</p>		
<p>Binary structure with both molecule A (yellow) and molecule C (blue)</p>		
<p>Trinary Structure with 4,4'-dichlorobiphenyl (green), molecules A (yellow) and B (blue)</p>		

5. Model performances on images with different resolutions

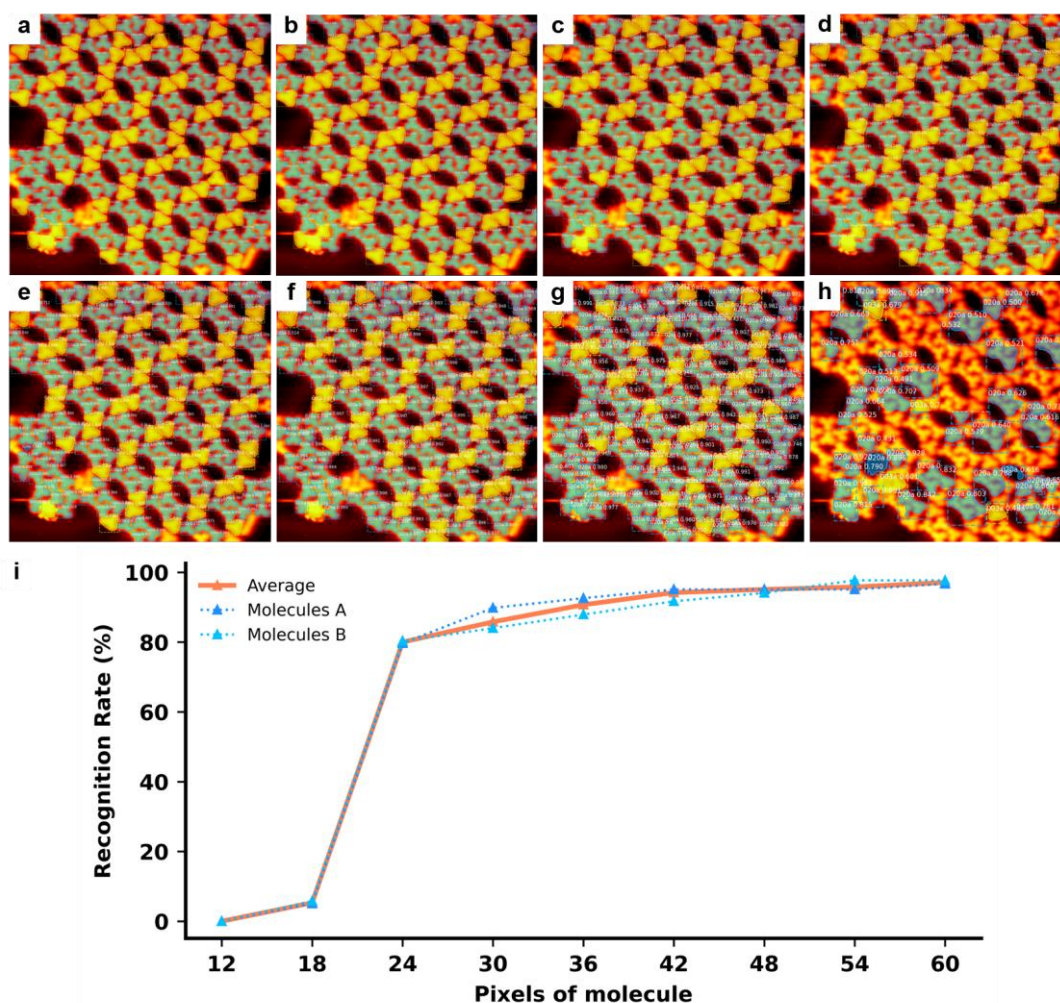


Figure S2. Model performances on STM images of the binary molecular self-assembly with different resolutions. The outputs of the testing STM images with resolutions of (a) 60*60 pixels, (b) 54*54 pixels, (c) 48*48 pixels, (d) 42*42 pixels, (e) 36*36 pixels, (f) 30*30 pixels, (g) 24*24 pixels, (h) 18*18 pixels for the individual molecules. (i) Summary of the recognition rates as a function of the resolution of molecules.

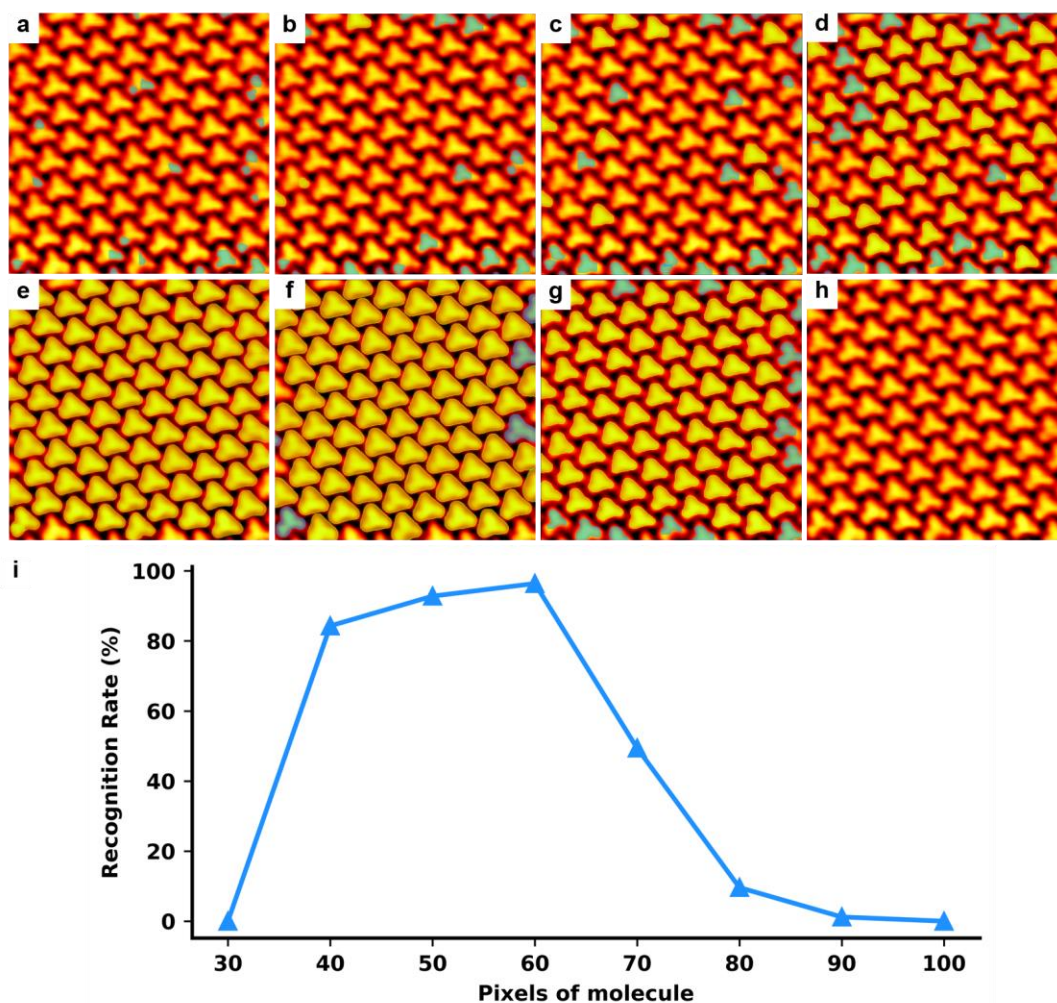


Figure S3. Model performances on STM images of the molecular self-assembly of **molecule A** with different resolutions. The outputs for testing STM images in which each molecule has a resolution of (a) 100*100 pixels, (b) 90*90 pixels, (c) 80*80 pixels, (d) 70*70 pixels, (e) 60*60 pixels, (f) 50*50 pixels, (g) 40*40 pixels, (h) 30*30 pixels. (i) The recognition rates as a function of the resolution of molecules.

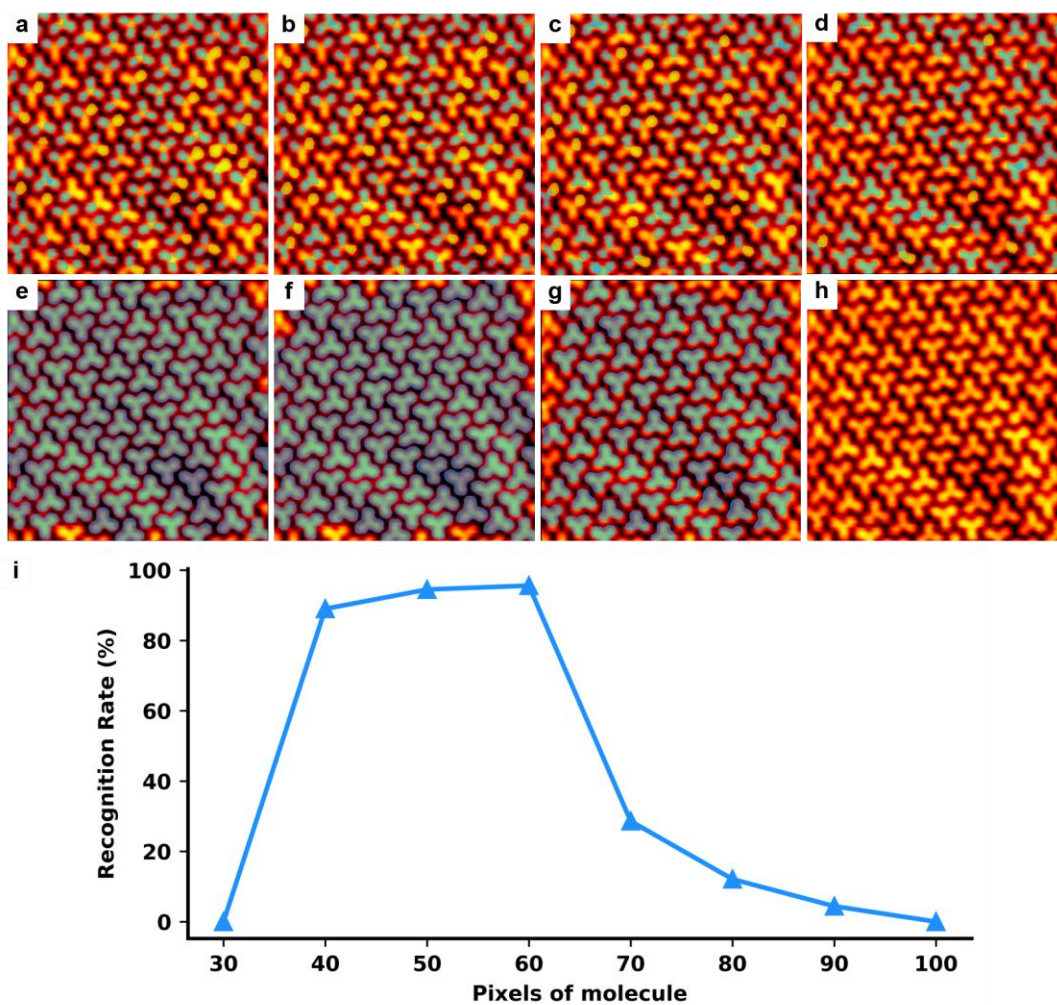


Figure S4. Model performances on STM images of the molecular self-assembly of **molecule B** with different resolutions. The outputs for testing STM images in which each molecule has a resolution of (a) 100*100 pixels, (b) 90*90 pixels, (c) 80*80 pixels, (d) 70*70 pixels, (e) 60*60 pixels, (f) 50*50 pixels, (g) 40*40 pixels, (h) 30*30 pixels. (i) The recognition rates as a function of the resolution of molecules.

6. Non-maximum suppression.

The non-maximum suppression (NMS) algorithm is used to deal with redundant Bbox. From the thousands of preselected boxes generated by the RPN network to the ultimately output prediction boxes which have the number of a few dozens, the deep learning framework applies the NMS algorithm many times. It is first applied after using the RPN network to filter the preselected boxes with a large intersection and ratio, and is applied again to filter similar results with a large intersection and ratio after the border classification has been completed.

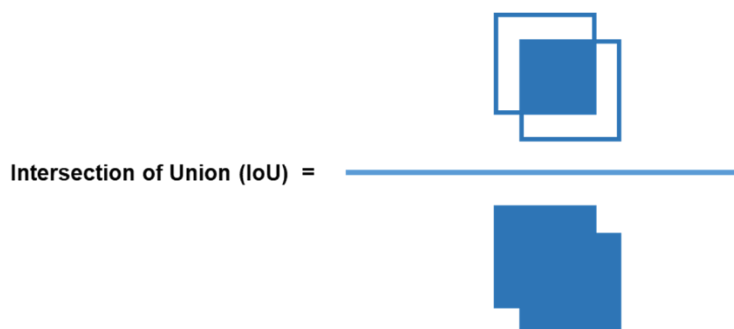


Figure S5. Definition of the intersection of Union (IoU): The area of the intersection of two boxes is divided by the area of the union.

Only two NMS filters are used in the source code of the Mask R-CNN open source project (https://github.com/matterport/Mask_RCNN). However, when identifying similar molecules, the prediction scores of both molecule A and molecule B near the same region are high, and then it is highly possible that a molecule is recognized as both molecule A and molecule B. In order to solve the problem, we add a layer of NMS filtering before the final recognition output, which could effectively reduce the false positive recognitions as indicated in Fig. S8.

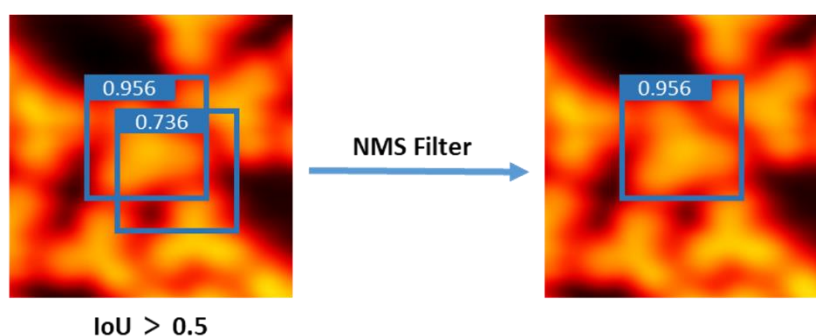


Figure S6. An example showing the output bounding box after the NMS filter.

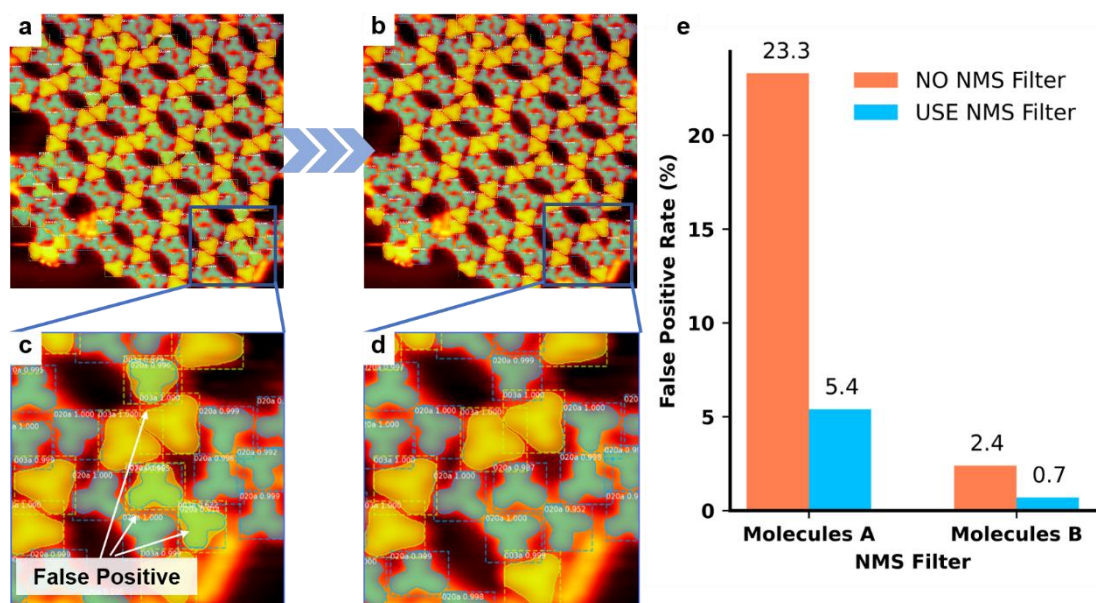


Figure S7. Applying the NMS filter. (a),(c) The outputs of the machine learning model without the NMS filter. (b),(d) The outputs of the machine learning model after adding the NMS filter. The green mask indicates regions identified as both **molecule A** and **molecule B**. (e) The false positive rate before and after applying the NMS filter.

7. Precision, Recall and mAP

In order to evaluate the performance of a target recognition model, it is obviously not comprehensive enough to evaluate through only the recognition rate. In models of computer versions, metrics like precision, recall and mean average precision (mAP) are frequently used. The criterion for judging the correct target detection is to compare the IoU of BBox and Ground Truth. We set the threshold for the IoU at a typical value of 0.5.

TP: True Positive. The classifier predicts that the result is a positive sample, and it is actually a positive sample.

FP: False Positive. The classifier predicts a positive sample, but it is actually a negative sample.

FN: False Negative. The classifier predicts a negative sample, but it is actually a positive sample.

In this study, we define:

Average Precision: Proportion of correctly predicted cases in all the predicted cases.

$$\text{Average Precision} = \frac{TP}{TP + FP}$$

Average Recall: Proportion of correctly predicted cases in all targets of the picture.

$$\text{Average Recall} = \frac{TP}{TP + FN}$$

In Fig. S9, we plot the prediction results according to the prediction confidence from high to low, and slowly lower the confidence threshold to gradually obtain more positive samples.

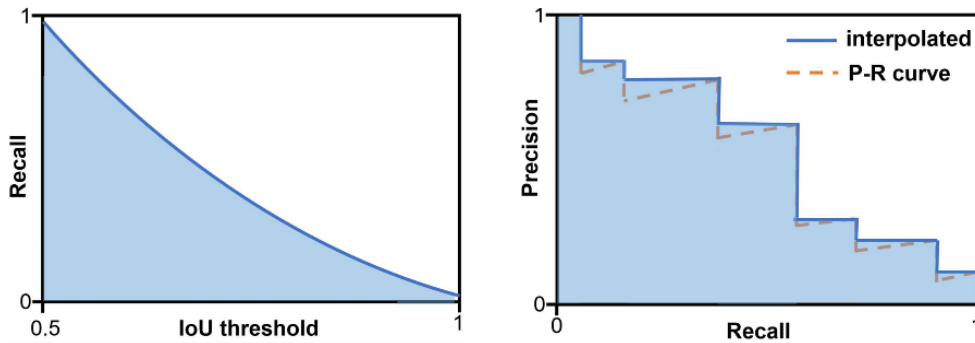


Figure S8. An example of a diagram of Recall-IoU curve, P-R curve and interpolation curve.

Another way to define AR: The recall averaged over all $\text{IoU} \in [0.5, 1.0]$ and can be computed as two times the area under the Recall-IoU curve.

During this process, each recall rate corresponds to a unique precision rate. Therefore, we can get a Precision-Recall curve (P-R curve). The area surrounded by the P-R curve and the two coordinate axes determines the value of mAP. It can also be inferred that under ideal conditions, the mAP value is 1. For the P-R curve, the interpolated precision at a certain recall level is defined as the highest precision found for any recall level.

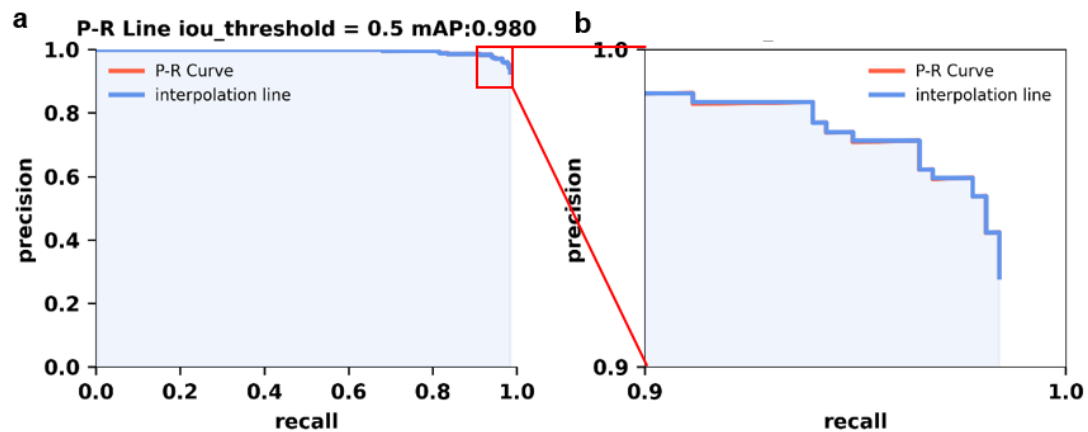


Figure S9. The P-R curve and interpolation curve with an IoU threshold of 0.5 obtained after 10 epochs of training tested on the image in Fig. 3a. The machine model was trained on the dataset of Fig. 3b with the data augmentation techniques of dropout and elastic. This model gives a mAP value of 0.98.

8. User guideline.

8.1 Programming environment requirements

Python 3.8 <https://www.python.org/> But we strongly recommend using anaconda to manage the python environment.

tensorflow 2.6.0 <https://tensorflow.google.cn/> The main body of our Mask R-CNN code comes from an open-source project. Its source code is written by the tensorflow1.0 framework, and we make it compatible with Tensorflow 2.0. We strongly recommend downloading the GPU version of tensorflow, which will substantially speed up the training process.^[1]

keras 2.6.0 In most cases, the corresponding version of Keras has been embedded in tensorflow versions of 2.0 and above, but the corresponding version of Keras is not necessarily compatible with the code.

Numpy 1.22.0 The numpy version cannot be either too high or too low, otherwise various unexpected bugs may occur.

Scikit-learn 1.0.2 <https://scikit-learn.org.cn/> Scikit-learn is a powerful machine learning library. The t-SNE algorithm API is included in scikit-learn.^[2]

cuda 8.1.10, cudatoolkit 11.2

If you want the tensorflow GPU version to call GPU resources normally, the installations of the two development packages are essential. Different versions of tensorflow have stricter requirements on the versions of cuda and cudnn. Please check the official documentation for more guidance. (https://tensorflow.google.cn/install/source_windows?hl=en#gpu)

labelme 3.16.7 <https://github.com/jameslahm/labelme> labelme is an software for image annotations of 2D or 3D images. Considering the compatibility of label software and code, please do not update labelme in the code running environment to a version higher than 3.16.7. But you can still use the latest version of labelme to label images (the new version is more user-friendly). It is worth noting that the labelme annotation file is not upward compatible with the software. But all versions of labelme annotation files can run under this code.^[3]

Imgaug 0.4.0 <https://github.com/aleju/imgaug-doc> imgaug is a library for image augmentations in machine learning projects.^[4]

Note: In the above we only list the important libraries and corresponding precautions. We do not exclude other simple error reports that can easily be solved. The detailed program running environment is accessible on our github https://github.com/gggg0034/SPM_image_Mask-R-CNN/tree/master

8.2 References of the hardware requirements

CPU AMD Ryzen 7 5800H with Radeon Graphics

GPU NVIDIA GeForce RTX 3070 Laptop GPU 8G

Memory 16G

1 epoch time cost: ≤ 25 minutes

CPU Intel Core i5 8300H

GPU NVIDIA GeForce GTX1050Ti Laptop GPU 4G

Memory 16G

1 epoch time cost: ≥ 6 hour

Note: The above computer hardware configuration is for reference only. Since the model training speed critically depends on GPU performance. It is highly recommended to choose a graphics card with 8G or more video memory.

8.3 Here, we have prepared a framework demo version to illustrate the specific operations in detail.

a) Select training images

We have prepared an original STM image named 00.png and an image of 00-cut.png which is cropped from 00.png in the `\SPM_image_Mask-R-CNN-master\images`.

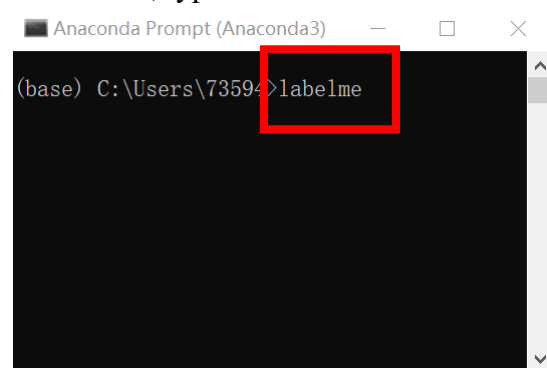
You can also crop a new 00-cut.png yourself, as long as the bit depth of your image is 24 bit (some cropping methods will change 24 bit to 32 bit because a new transparency channel is introduced to the image).

Then put the image in the folder `\SPM_image_Mask-R-CNN-master\aug_test_in`

b) Annotate training images

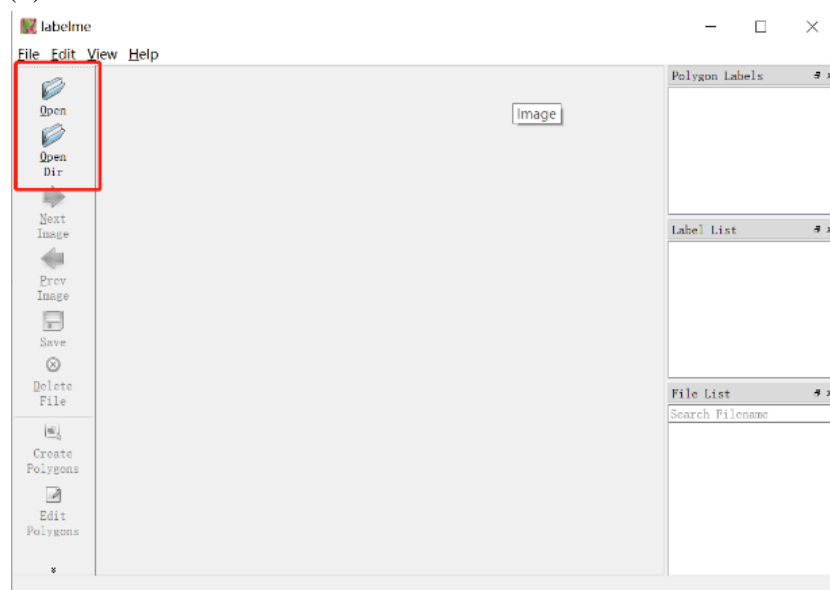
(1) open labelme

Open the Prompt console as an administrator, switch to the environment where labelme is installed, type labelme after the console and press enter.

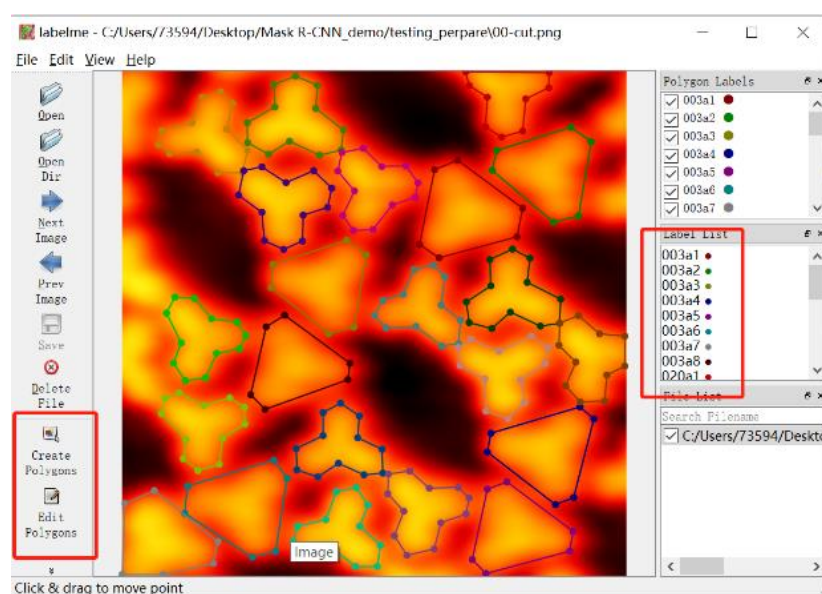


(Of course, one can also find and start it directly in the environment folder)

(2) annotate molecules



Click on one of the buttons in the red box, find the picture you want to annotate and open it.



Click the button in the red box on the left to mark the outline of the molecule. When naming the label, please add a serial number to the name to distinguish the different molecules, such as molecule A1, molecule A2, molecule B1, molecule B2, other than molecule A, molecule A, molecule B, molecule B.

Click Save after marking, and a json file with the same name as the image will be generated.

Put the image file with the json file in

\SPM_image_Mask-R-CNN-master\aug_test_in

If you do not want to modify the parameters and are willing to train with the default optimal configuration, please skip **step (c)** and open the

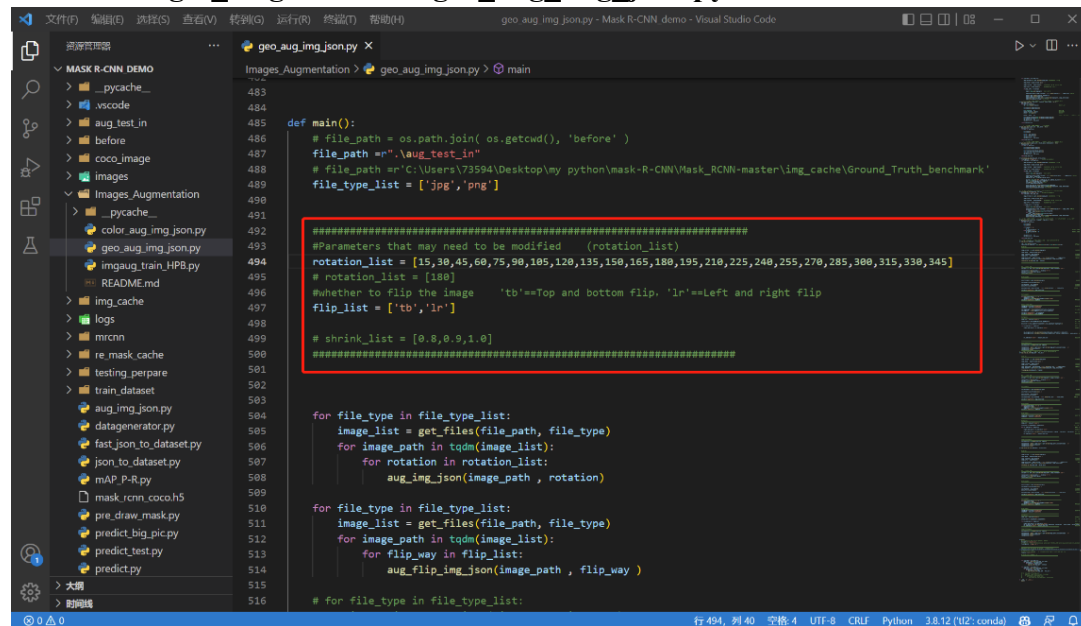
\\SPM_image_Mask-R-CNN-master\workflow.py

to run directly to get the weight parameter. This will take about 40 minutes (depending on your GPU).

c) Modify the training set data augmentation parameters

(1) L1 augmentation

Open SPM_image_Mask-R-CNN-master\Images_Augmentation\geo_aug_img_json.py



```
def main():
    # file_path = os.path.join( os.getcwd(), 'before' )
    file_path = ".\aug_test_in"
    # file_path = "C:\Users\yz18514\Desktop\my_python\mask-R-CNN\Mask-RCNN-master\img_cache\Ground_Truth_benchmark"
    file_type_list = ['jpg', 'png']

    #####
    #Parameters that may need to be modified (rotation list)
    rotation_list = [15,30,45,60,75,90,105,120,135,150,165,180,195,210,225,240,255,270,285,300,315,330,345]
    # rotation_list = [180]
    #Whether to flip the image 'tb'==Top and bottom flip, 'lr'==Left and right flip
    flip_list = ['tb','lr']

    # shrink_list = [0.8,0.9,1.0]
    #####

    for file_type in file_type_list:
        image_list = get_files(file_path, file_type)
        for image_path in tqdm(image_list):
            for rotation in rotation_list:
                aug_img_json(image_path , rotation)

    for file_type in file_type_list:
        image_list = get_files(file_path, file_type)
        for image_path in tqdm(image_list):
            for flip_way in flip_list:
                aug_flip_img_json(image_path , flip_way )

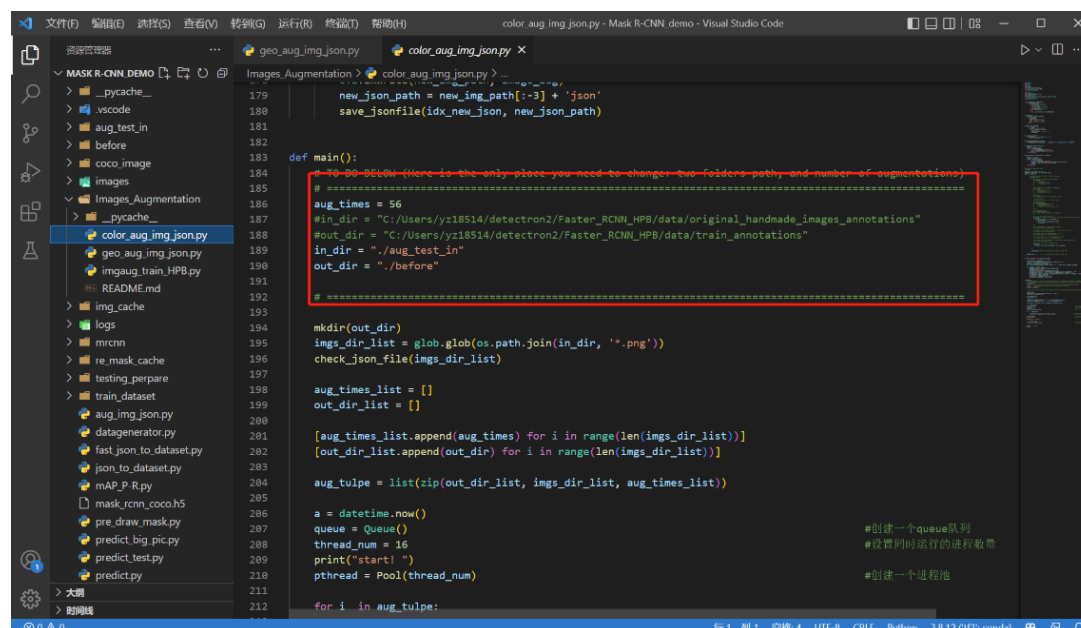
    # for file_type in file_type_list:
```

Modify the configuration you want to adjust in the code shown in the red box.

Then run the code.

(2) L2 & L3 augmentation

Open SPM_image_Mask-R-CNN-master\Images_Augmentation\color_aug_img_json.py



```
def main():
    # =====
    # TO DO: Below there is the only place you need to change the folders path, and number of augmentations
    # =====
    aug_times = 56
    #in_dir = "C:/Users/yz18514/detectron2/Faster_RCNN_HPB/data/original_handmade_images_annotations"
    #out_dir = "C:/Users/yz18514/detectron2/Faster_RCNN_HPB/data/train_annotations"
    in_dir = ".\aug_test_in"
    out_dir = ".\before"
    # =====

    mkdir(out_dir)
    imgs_dir_list = glob.glob(os.path.join(in_dir, '*.png'))
    check_json_file(imgs_dir_list)

    aug_times_list = []
    out_dir_list = []

    [aug_times_list.append(aug_times) for i in range(len(imgs_dir_list))]
    [out_dir_list.append(out_dir) for i in range(len(imgs_dir_list))]

    aug_tuple = list(zip(out_dir_list, imgs_dir_list, aug_times_list))

    a = datetime.now()
    queue = Queue()
    thread_num = 16
    print("start! ")
    pthead = Pool(thread_num)
    #创建一个queue队列
    #设置同时运行的进程数
    #创建一个进程池

    for i in aug_tuple:
```

Modify the configuration you want to adjust in the code shown in the red box.

`aug_times` is recommended to stay between **28-112**.

Then run the code.

So far, the training set is generated and saved in

`\SPM_image_Mask-R-CNN-master\before`

(3) Dataset preprocessing

Run the following three pieces of code in sequence to perform training preprocessing on the dataset.

`\SPM_image_Mask-R-CNN-master\pre_draw_mask.py`

`\SPM_image_Mask-R-CNN-master\re_write_mask.py`

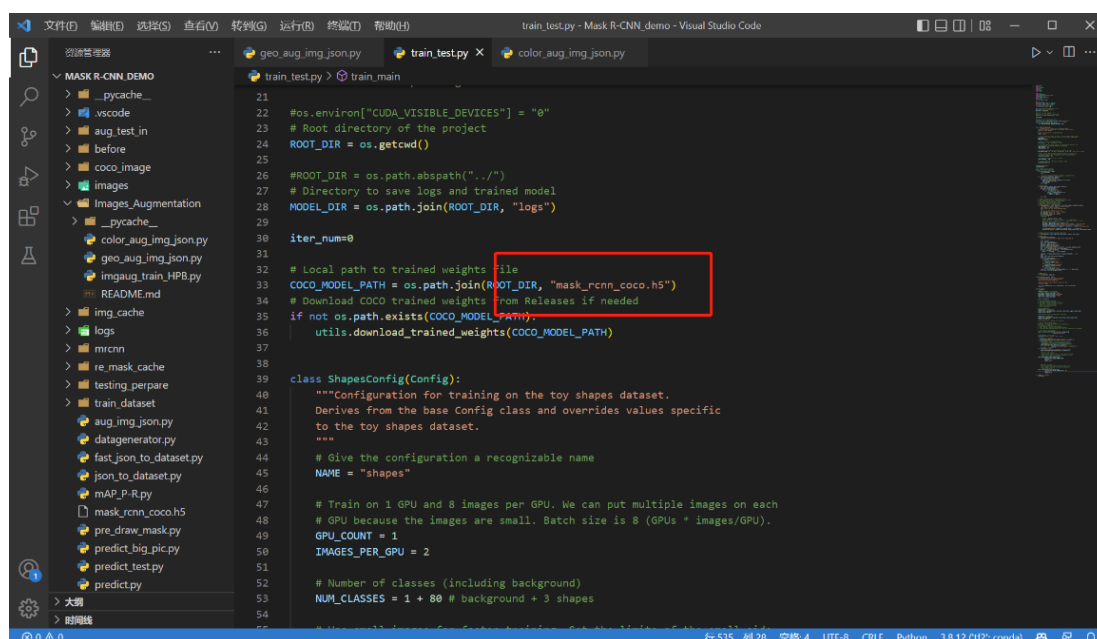
`\SPM_image_Mask-R-CNN-master\fast_json_to_dataset.py`

(4) Start training

Download pre-trained COCO weights (`mask_rcnn_coco.h5`) from the https://github.com/matterport/Mask_RCNN/releases

After the download is complete, please place the pre-trained COCO weights in the `\SPM_image_Mask-R-CNN-master` (root directory).

Open `\SPM_image_Mask-R-CNN-master\train_test.py`



```
21
22 #os.environ["CUDA_VISIBLE_DEVICES"] = "0"
23 # Root directory of the project
24 ROOT_DIR = os.getcwd()
25
26 #ROOT_DIR = os.path.abspath("../")
27 # Directory to save logs and trained model
28 MODEL_DIR = os.path.join(ROOT_DIR, "logs")
29
30 iter_num=0
31
32 # Local path to trained weights file
33 COCO_MODEL_PATH = os.path.join(ROOT_DIR, "mask_rcnn_coco.h5")
34 # Download COCO trained weights from Releases if needed
35 if not os.path.exists(COCO_MODEL_PATH):
36     utils.download_trained_weights(COCO_MODEL_PATH)
37
38
39
40 class ShapesConfig(Config):
41     """Configuration for training on the toy shapes dataset.
42     Derives from the base Config class and overrides values specific
43     to the toy shapes dataset.
44     """
45     # Give the configuration a recognizable name
46     NAME = "shapes"
47
48     # Train on 1 GPU and 8 images per GPU. We can put multiple images on each
49     # GPU because the images are small. Batch size is 8 (GPUs * images/GPU).
50     GPU_COUNT = 1
51     IMAGES_PER_GPU = 2
52
53     # Number of classes (including background)
54     NUM_CLASSES = 1 + 80 # background + 80 classes
55
```

Pretrained weight parameters loaded by the parameter bits in the red box (modification is not recommended). The pretrained weight parameters are placed in the root directory. Please modify other detailed parameters in the `ShapesConfig` class.

And add, delete or modify the categories you need to predict in the `load_mask` and `load_shapes` functions (modification is not recommended).

```

300 # ShapesConfig
301 # Load weights trained on no COCO, but skip layers that
302 # are different due to the different number of classes
303 # See README for instructions to download the COCO weights
304 model.load_weights(COCO_MODEL_PATH, by_name=True,
305                  exclude=["mrcnn_class_logits", "mrcnn_bbox_fc",
306                          "mrcnn_bbox", "mrcnn_mask"])
307 elif init_with == "last":
308     # Load the last model you trained and continue training
309     model.load_weights(model.find_last()[1], by_name=True)
310
311 # Train the head branches
312 # Passing layers="heads" freezes all layers except the head
313 # layers. You can also pass a regular expression to select
314 # which layers to train by name pattern.
315 model.train(dataset_train, dataset_val,
316            learning_rate=config.LEARNING_RATE,
317            epochs= 1,
318            layers='heads')
319
320 # Fine tune all layers
321 # Passing layers="all" trains all layers. You can also
322 # pass a regular expression to select which layers to
323 # train by name pattern.
324 model.train(dataset_train, dataset_val,
325            learning_rate=config.LEARNING_RATE / 10,
326            epochs= 2,
327            layers="all")
328
329 if __name__ == '__main__':
330     train_main()

```

The epoch of layers = 'all' is recommended to be set to twice of the epoch of layers = 'heads', and the former should preferably not be greater than 10 to avoid excessive training time.

Then run the code to start training.

d) Molecule recognition

(1) Prepare to load weight parameters

After training is complete, you can find the trained weight parameters in

\SPM_image_Mask-R-CNN-master\logs

plugins	2022/5/12 20:37	文件夹	
events.out.tfevents.1652358989.LAPTOP-5...	2022/5/12 22:24	LAPTOP-5OKTL3U7...	6,493 KB
events.out.tfevents.1652359022.LAPTOP-5...	2022/5/12 20:37	PROFILE-EMPTY 文件	1 KB
events.out.tfevents.1652365511.LAPTOP-5...	2022/5/13 0:57	LAPTOP-5OKTL3U7...	8,306 KB
mask_rcnn_shapes_0001.h5	2022/5/12 20:58	H5 文件	178,739 KB
mask_rcnn_shapes_0002.h5	2022/5/12 21:19	H5 文件	178,739 KB

Where **mask_rcnn_shapes_0001.h5** is generated after the first epoch iteration.

And **mask_rcnn_shapes_0002.h5** is generated after the second epoch iteration.

Then move the weight parameters generated by the last iteration (**For example, mask_rcnn_shapes_0002.h5 here**) to the root directory.

(2) Prepare to load prediction image

Put the picture you want to predict into **\SPM_image_Mask-R-CNN-master\images**.

(3) Start molecule recognition

Open **\SPM_image_Mask-R-CNN-master\predict_big_pic.py**

```
visualize.save_instances_sc(count, save_path, image, new_result_rec, new_r_mask_list, new_result_label,
                           class_names, new_r_score_list, colors = colors, show_bbox= False, captions= True)

if __name__ == '__main__':
    # 可能需要修改的参数
    MODEL_PATH = r'mask_rcnn_shapes_0002.h5' # 权重路径
    class_names = ['BG', '003a', '020a']
    colors = [(0.7999999999999998, 1.0, 0.0), (0.0, 0.6727272727272728, 1.0)]
    img_names = '00.png'

    DETECTION_MIN_CONFIDENCE_LIST = [0.0, 0.1, 0.15, 0.20, 0.25, 0.30, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65, 0.70, 0.75, 0.80, 0.85,
    #####
    a = datetime.now()
    for DETECTION_MIN_CONFIDENCE in DETECTION_MIN_CONFIDENCE_LIST[0:1]:
        overlap_cut(img_names)
        mini_img_names = img_names.split('.') + img_names.split('.')[1]
        mini_img_list = get_files(os.path.join(os.getcwd(), 'img_cache', 'pre_predict'), 'png')

        for count, img_name in enumerate(mini_img_list):
            predict_big_pic(count, MODEL_PATH = MODEL_PATH
                           , class_names = class_names
                           , img_path = img_name
                           , colors = colors
                           , DETECTION_MIN_CONFIDENCE = DETECTION_MIN_CONFIDENCE
            )
```

Modify the name of your weight parameter, the name of the STM image to be predicted, and modify the label list (the first label list must be "BG" to represent the background, please do not modify it).

After a while, you will finally get the output image in **\SPM_image_Mask-R-CNN-master\img_cache**

9. Materials and Methods

9.1 Molecular structures

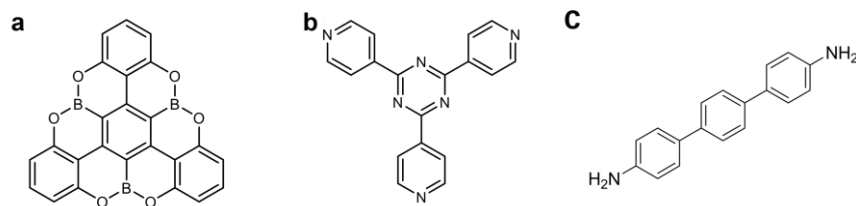


Figure S10. Molecular structures. (a) Molecular structure of molecule A (OBO-doped [4]triangulene). (b) Molecular structure of molecule B (2,4,6-tri(Pyridin-4-yl)-1,3,5-triazine). (c) Molecular structure of molecule C ([1,1':4',1''-terphenyl]-4,4''-diamine).

9.2 Methods.

STM characterization.

We used a custom-designed commercial low-temperature STM system (Bosezi (Beijing) Co. Ltd.) for *in situ* characterization under ultra-high vacuum conditions of base pressures below 1×10^{-10} mbar. The single crystals (MaTeck GmbH) were cleaned by several cycles of argon sputtering and annealing under UHV conditions until large terraces separated by monatomic steps were achieved. The measurements were performed at liquid nitrogen temperature (~ 77.6 K) if not stated otherwise. STM imaging was performed with the constant-current mode at typical bias ranges of -1.0 to -2.0 V and current ranges of 50 to 150 pA.

Sample preparation.

The molecule precursor **A** is commercially available from Tansoole and the synthesis of molecule precursor **B** is reported elsewhere.^[5] After degassing under UHV condition, the molecular precursors were thermally evaporated from a three-fold organic evaporator onto the metal surfaces in a sequential order. The sublimation rates of both molecular precursors were monitored by a quartz crystal microbalance (SQM-160, INFICON). We developed a LabVIEW based program to ensure that the molecular evaporation rate was stable for molecular evaporation.

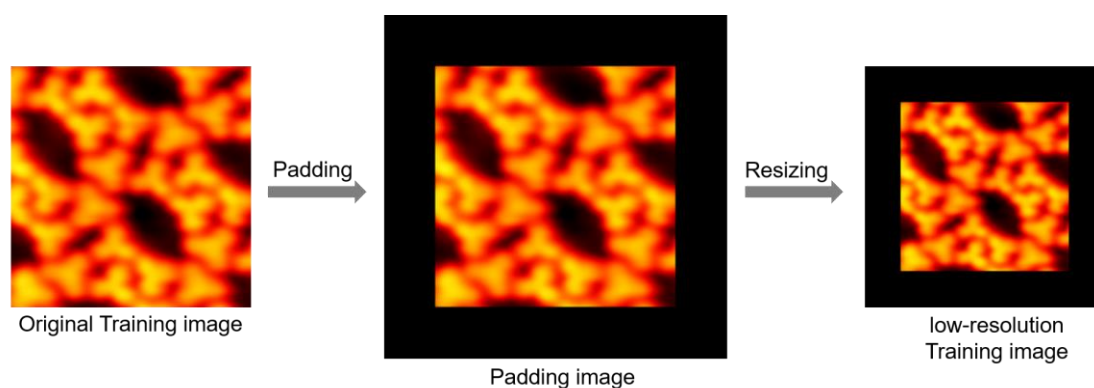
Machine learning model and the program.

a. Mask R-CNN. Computer vision has made remarkable progress with the development of deep learning and convolutional neural network (CNN). CNN enables parameter sharing so that the quantity of parameters is reduced greatly compared with fully-connected neural networks.^[6,7] Region-CNN (R-CNN) introduces object recognition based on CNN.^[8,9] The recognition efficiency and accuracy have been significantly improved in Fast R-CNN^[10] and Faster R-CNN.^[11] Eventually, pixel-to-pixel instance segmentations of objects were achieved in Mask R-CNN by applying feature pyramid and ROI align,^[12,13] which provides the most intuitive results for human in the process of analyzing digital images. Mask R-CNN consists of three layers of machine learning networks. The feature pyramid network (FPN)^[14] is responsible for extracting image

features at multiple scales. The region proposal network (RPN)^[15] determines the molecular location, and the region of interest (RoI) module can determine the type of molecules and perform semantic processing. More details can be found in Supplementary note 9.3.

b. NMS filter. NMS is used to filter the anchor boxes in the RPN and to filter the prediction boxes of the recognition results. When the Intersection over Union (IoU) of the two borders is large enough, the bounding box with the lower score will be removed. NMS has a significant effect on removing false positive rates. More details about NMS operation can be found in Supplementary note 6.

c. Padding Image. One of the future goals of this framework is to perform molecular recognitions on unknown STM images or images with minor prior knowledge, such that we will be able to obtain real-time data analysis while measuring. To this end, we have tested the model performance by varying the image resolutions between the training and testing data.



In this work, we reduced the image resolution of training datasets with the following way. As shown in the figure below, the original training image will be added with a back padding, followed by the process of resizing to produce the low-resolution image. By controlling the width of the padding, multi-scale images of training data can be obtained.

d. Transfer Learning and Training. Due to the limited data of SPM images, the convergence of the convolutional backbone network is extremely difficult, which means that transfer learning is essential. We first adopt the coco 2017 train images dataset^[16] to train our Mask R-CNN model. The coco dataset contains tens of thousands of objects' photos of daily life, including 80 categories. In this study, the weights of the bottom of the backbone network FPN would be similar even the recognition types are inconsistent with the coco dataset.^[8] During the training process, the weight parameters outside the convolutional backbone network of the Mask R-CNN, such as Classification branch, Bounding box regression branch and Mask branch, are the focus of training.

e. t-SNE. t-SNE is an unsupervised machine learning algorithm for finding patterns in the data based on the similarity of data points. The similarity of points is calculated as the conditional probability. It then attempts to minimize a cost function, which is defined as a single Kullback–Leibler divergence^[17] between joint probability distributions in the high-dimensional space and the low-dimensional space. By using a

student's t-distribution with a single degree of freedom it computes the similarity between two points in the low-dimensional space. The experimental details of t-SNE are provided in Supplementary note 1.

e. Image augmentation. Data augmentation technology is able to quickly obtain a large amount of image data.^[18-20] In the data augmentation process, we first performed geometric augmentations, followed by the other augmentation techniques. It is worth noting that the orientations and positions of molecules in the images after geometric augmentations have changed, which means that the labelling information in the corresponding image label files also needs to be changed accordingly to ensure the validity of the labelled data. Operational details of data augmentations can be found in Supplementary note 2.

f. Programming environment and hardware. The program of the deep learning framework was run on a personal computer. The requirements of the Programming environment and the references of the hardware can be found in Supplementary note 8.

9.3 More details on Mask R-CNN.

FPN extracts feature maps of different scales from input SPM images and, hence, enables Mask R-CNN to detect molecules of various sizes. In our case, they are SPM images with different resolutions. FPN includes five convolutional blocks connected in series built with a total of 50 convolutional and ReLU layers (ResNet-50)^[21]. With the increase in the number of convolutional layers, the feature map is expected to have higher semantic information, but it is also accompanied by the loss of information of the small target of the original image. The FPN layer adopts the strategy of upsampling and stacking the feature map to simultaneously retain high semantic information and small target information. The detailed architecture is provided in Figure S1.

The RPN layer extracts the regions of interest from the feature maps of each layer. For each feature map input to this layer, fifteen anchor boxes will be generated by the RPN network centered on each pixel on the feature map. The anchor boxes that exceed the image boundary will be removed. It ensures that each molecule can be covered by one or more anchor boxes. All anchor boxes will be fed into a simple CNN-based evaluation network, where the information in each anchor box is scored and the positions are adjusted based on linear regressions. The score of the anchor box is positively correlated with the probability that the target molecule is contained in the anchor box. The non-maximum suppression algorithm will filter a large part of anchor boxes with a high overlap and low scores.

The ROI align layer pools the feature map information in the anchor box into vectors of a uniform 7*7 pixels size through bilinear interpolation. Subsequently, the feature vectors are fed into three different branches as follows.

1. Classification branch. Here the feature vectors will be flattened and fed into two fully connected layers (FC)^[22], using the SoftMax function^[23] to output a three-channel matrix corresponding to the probability that the feature vector is molecule A, molecule

B or background respectively. 2. Bounding box regression branch. The feature vector will be flattened and fed into two FCs, the final linear regression outputs a four-channel matrix representing the bounding box fine-tuning parameters. 3. Mask branch. The feature vector will be processed by the deconvolution layer, and the sigmoid function will generate the mask pixel by pixel. The principle of the mask branch is the decoding part of the fully convolutional network (FCN)^[24] where the convolutional backbone network (Resnet-50) is the encoding part.

The category, bounding box position and mask information will be output after a final NMS algorithm to filter the overlapping results after integration to get the output.

10. References

- [1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, **2016**, pp. 265–283.
- [2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, *Journal of Machine Learning Research* **2011**, *12*, 2825–2830.
- [3] K. Wada, *Accessed: Oct 2016*, *2*, 2020.
- [4] A. B. Jung, K. Wada, J. Crall, S. Tanaka, J. Graving, C. Reinders, S. Yadav, J. Banerjee, G. Vecsei, A. Kraft, Z. Rui, J. Borovec, C. Vallentin, S. Zhydenko, K. Pfeiffer, B. Cook, I. Fernández, F.-M. De Rainville, C.-H. Weng, A. Ayala-Acevedo, R. Meudec, M. Laporte, others, **2020**.
- [5] X. Chen, D. Tan, J. Dong, T. Ma, D.-T. Yang, **2022**, DOI 10.26434/chemrxiv-2022-9f544
- [6] L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. A. Fadhel, M. Al-Amidie, L. Farhan, *Journal of big Data* **2021**, *8*, 1–74.
- [7] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai, *Pattern recognition* **2018**, *77*, 354–377.
- [8] R. Girshick, J. Donahue, T. Darrell, J. Malik, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, **2014**, pp. 580–587.
- [9] S. Albawi, T. A. Mohammed, S. Al-Zawi, in *2017 International Conference on Engineering and Technology (ICET)*, **2017**, pp. 1–6.
- [10] R. Girshick, in *Proceedings of the IEEE International Conference on Computer Vision*, **2015**, pp. 1440–1448.
- [11] S. Ren, K. He, R. Girshick, J. Sun, *Advances in neural information processing systems* **2015**, *28*.
- [12] K. He, G. Gkioxari, P. Dollár, R. Girshick, in *Proceedings of the IEEE International Conference on Computer Vision*, **2017**, pp. 2961–2969.
- [13] T. Bai, Y. Pang, J. Wang, K. Han, J. Luo, H. Wang, J. Lin, J. Wu, H. Zhang, *Remote Sensing* **2020**, *12*, 762.
- [14] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, S. Belongie, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, **2017**, pp. 2117–2125.
- [15] A. Mikołajczyk, M. Grochowski, in *2018 International Interdisciplinary PhD Workshop*

- (*IIPhDW*), **2018**, pp. 117–122.
- [16] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, C. L. Zitnick, in *European Conference on Computer Vision*, Springer, **2014**, pp. 740–755.
 - [17] F. Perez-Cruz, in *2008 IEEE International Symposium on Information Theory*, **2008**, pp. 1666–1670.
 - [18] C. Shorten, T. M. Khoshgoftaar, *J Big Data* **2019**, 6, 60.
 - [19] J. Wang, S. Mall, L. Perez, *Convolutional Neural Networks Vis. Recognit* **2017**, 11, 1–8.
 - [20] D. M. Pelt, J. A. Sethian, *Proceedings of the National Academy of Sciences* **2018**, 115, 254–259.
 - [21] K. He, X. Zhang, S. Ren, J. Sun, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, **2016**, pp. 770–778.
 - [22] W. Samek, G. Montavon, S. Lapuschkin, C. J. Anders, K.-R. Müller, *Proceedings of the IEEE* **2021**, 109, 247–278.
 - [23] T. Mikolov, S. Kombrink, L. Burget, J. Černocký, S. Khudanpur, in *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, **2011**, pp. 5528–5531.
 - [24] J. Long, E. Shelhamer, T. Darrell, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, **2015**, pp. 3431–3440.