# Automatic Analysis of STEM Images of Catalyst Systems Based on Deep Learning Model

Project Report: Final Report of AY2022/2023

Non-Graduating Student: YUAN Wenhao

Supervisor: Assistant Professor HE Qian

*Submitted [04, 2023]*

# Table of Contents

# ABSTRACT

As computer science research converges toward complex but highly accurate deep learning models, we are committed to advancing the application of data-driven approaches that use deep learning to motivate and augment more straightforward high-throughput in situ experimental data processing techniques, enabling results that are clearly interpreted, easily quantified, and reproduced over generalized datasets. This work is oriented towards industrial catalyst systems and develops a deep learning-based UNet++ pipeline for feature extraction of NPs from STEM images. The trained network outperforms existing work in different metrics, with the Mean Intersection over Union reaching 80.8%. The main follow-up plan of the research is to implement label-free with the help of simulation and AI reimaging, and eventually develop a fully-automatic analysis software of STEM images and attach it to electron microscopy equipment, which has potentially great engineering implications.



**Key words:** automated analysis, deep learning, STEM, nanoparticles, image segmentation

## 1. Introduction

Scanning transmission electron microscopy (STEM), which combines conventional transmission electron microscopy techniques and scanning electron microscopy techniques, has emerged as an indispensable tool for the characterization of objects in the domains of materials science, chemistry, physics, and medicine[1]. STEM offers rich and immediate information on the structure and dynamics of phenomena spanning from the atomic to the micrometer scales. Such information has enormous value both in theoretical and practical realms, and stimulated the electronics revolution that has given rise to the highly interconnected world we inhabit today[2]. Despite the remarkable progress made in the field of material characterization through STEM, many complex material problems remain unresolved. For instance, effective control over chemical reaction pathways in catalysts requires access to interchangeable and finely-tuned environments, as well as the utilization of knowledge gleaned from a vast collection of prior experimental trials[3]. Additionally, the combinatorial engineering of high-entropy alloys necessitates experimental decision-making based on automatic evaluation that must be performed in real-time[4]. Advancements in material characterization that enable the discovery of entirely new classes of materials and their potential applications will require a paradigm shift in microscopy models across.

The next generation of STEM will have three notable features: faster acquisition, high-throughput, high-integration. The faster acquisition now can achieved by in situ and operando experimental techniques, where dynamic process can be observed with high temporal and spatial resolution, have allowed scientists to observe chemical reactions, interfacial phenomena, and mass transport processes to give not only a better understanding of the physics of materials phenomena, but also a view into how materials react under the conditions in which they are designed to perform[5]. In situ electron microscopy experiments can capture high-resolution images are at very high frame rates[6]. In practice, hundreds of images can be captured per second. Methods for fast and efficient processing of high-resolution imaging data will allow for not only full utilization of existing and developing technologies, but also for producing results with more statistical insight based on the sheer volume of data being analyzed. This leads to the second necessary feature, high-throughput data processing and feature extraction. With the help of machine learning (ML)[7] and fast, high-efficiency electron detectors, automated imaging data acquisition now is feasible in STEM[8]. These recently

developed capabilities drive the development of automatic techniques to identify and connect important structural characteristics from STEM data, such as nanoparticle size, shape, and defect content, to bulk properties and the investigation of the effect of heterogeneity on bulk behavior, which is recognized as ML-based feature extraction. Further with these features from different multiple imaging techniques, such as STEM and electron energy loss spectroscopy (EELS) or energy dispersive spectroscopy (EDS), we can combine them to provide a more comprehensive picture of material's structure and composition towards highly integrated data analysis.

In this work, we focus on the promotion of next-generation STEM applied to industrial catalyst systems. In general, protocols, which outperform classical image analysis and do not require time-consuming manual analysis, are needed for high throughput data processing. Recent advances in image interpretation using deep learning (DL), segmentation, via a convolutional neural net (CNN) make promising routes toward the automatic interpretation of STEM micrographs. In detail, the particle size distribution, location, thickness, roundness and other morphology of nanoparticles (NPs) on the substrate are important for catalyst analysis and design. Automatic morphology analysis of NPs by DL-based method can eliminate the disadvantages of manual labeling such as time-costing and human-bias. Here, we employ an extremely lightweight dataset containing only 27 labeled high-resolution STEM images to train a UNet++ neural network pipeline that enables fast and highly accurate recognition of NPs in complex industrial catalyst systems. Compared with previous work, we take advantage of the less parameters-needed and high accuracy of UNet++ to significantly reduce the training dataset and improve the recognition accuracy, achieving semi-automated STEM image analysis without manual parameterization. The next stage of this research is to achieve label-free, using simulation and AI reimaging to finally achieve fully-automated STEM image analysis.

## 2. Methods

### 2.1 Sample Preparation

Pt-Sn complexes were prepared via coordination in organic phase. $K_2PtCl_4$ (Aladdin, ⩾ 99.9% metals basis) was dissolved in deionized water, and then transferred to $CH_2Cl_2$ (VWR, ⩾99%) with $NBu_4Br$ (Sigma Aldrich, ⩾98%). $SnCl_2 \cdot 2H_2O$ (Aladdin, ⩾99.99% metals

basis) was dissolved in tetrahydrofuran (VWR, ⩾99.5%). Pt and Sn solution were mixed to attain various Sn/Pt molar ratios. Color changes took place immediately. After 30 minutes' stirring to stabilize the solution, a certain amount of $Al_2O_3$ powder (Sigma Aldrich, 13 nm primary particle size) was added and the mixture was stirred at room temperature for 24 hours. During this process, the PtSn complexes could be loaded onto $Al_2O_3$ through adsorption. Centrifugation was applied to separate the catalyst and organic solvent. Finally, the obtained catalyst was dried and stored in a vacuum desiccator.

### 2.2 Experimental STEM Imaging

$PdSn/Al_2O_3$ samples were imaged in a JEOL ARM-200CF instrument operated at 200 kV on the scanning mode, at magnifications between 2500,000–5000,000X. Images were acquired at a resolution of 2,048 × 2,048 pixels with a dimension range of 38.438 $nm^2$–76.876 $nm^2$, resulting a final sampling size of 0.038–0.019 $nm \cdot pixel^{-1}$. Images of a variety of scenarios, i.e., a low and high particle load, in- and out-of-focus images, favorable and unfavorable contrast from the support material as examples, were included to mimic a more realistic automated imaging procedure and to explore the working limits of the method.

### 2.3 Data Processing

To reduce the computational demands, we first resized each micrograph into 512 × 512 images. For training dataset, these resized images were augmented using Flip, ShiftScaleRotate and GaussNoise from Albumentations[9] successively, all with a probability value of 0.5. Finally, they were normalized using Albumentations. For training dataset, it was only normalized using Albumentations without data augmentation.

### 2.4 Network Architecture

The architecture implemented was based on the UNet++ architecture developed by Zongwei Zhou et al. (2018)[10] following the same encoder-decoder structure. We modified the standard UNet++ structure for accurate segmentation and to prevent overfitting.

### 2.5 Network Training

The final resized dataset consisted of 27 512 × 512 pixel images, which was randomly split into training (80%) and validation (20%) sets. Training was performed using the laptop of ASUS ROG Strix SCAR 17 G732LWS-0003 with 16 Intel(R) Core(TM) i7-10875H CPU @

230GHz cores and a NVIDIA GeForce RTX 2070 SUPER. The laptop had 16 GB of available RAM. All programming was done in Python on PyCharm, with deep learning aspects using the PyTorch framework. The model was limited to train for a maximum of 150 epochs and set to use CUDA as first device. Batch size was set to 4, where batch size defines how many samples from the training set are propagated through the network before updating the model weights. The encoder used for UNet++ is ResNet34[11]. The model used FocalLoss[12] (the mode uses multiclass with the alpha of 0.25) and JaccardLoss[13] (the mode uses multiclass) for the loss functions and AdamW[14] as the optimizer with a learning rate of $3 \times 10^{-4}$ and a weight decay of $5 \times 10^{-4}$. StepLR was used the learning scheduler with a step size of 5, gamma of 0.9.

### 2.6 Testing

As mentioned above, 20% of the final resized dataset was used first for validation. In addition, in order to avoid inherent bias due to strong correlation between training and test sets in randomly split consecutive images, a third validation set, collected at a different time but under the same conditions, should also be included. So, we introduced a third dataset of 12 images that did not contain the images used for the initial training and testing and fed this dataset directly into the trained model to test the prediction accuracy.

### 2.7 Evaluation Metrics for Network

The UNet++ used in our work predicts the pixels in STEM images into two categories of positive examples (nanoparticles) and negative examples (background) to achieve binary semantic segmentation. Therefore, we define the true positive (TP) as the number of pixels where UNet++ correctly predicts positive examples. The true negative (TN) is defined as the number of pixels where UNet++ correctly predicts negative examples. The false positive (FP) is defined as the number of pixels where UNet++ predicts positive examples as negative examples. The false negative (FN) is defined as the number of pixels where UNet++ predicts negative examples as positive examples. To evaluate the neural network's ability to segment nanoparticles from the micrograph, we used four evaluation metrics: (1) Loss, (2) Pixel Accuracy (PA), (3) Mean Pixel Accuracy (MPA), and (4) Mean Intersection over Union (MIoU). Loss is the difference between prediction and ground truth, determined by averaged values of FocalLoss and JaccardLoss. PA is the ratio of the number of pixels with the correct

predicted class to the total number of pixels, which is calculated as the sum of diagonal elements in the confusion matrix divided by the sum of all elements, as shown in **Equation (1)**. MPA calculates the proportion of correctly classified pixels for each class, and then accumulates and averages them, which is shown as **Equation (2)**. MIoU is the ratio of the intersection and union of the model's predicted results for each class and the true value, summed and averaged, as shown as **Equation (3)**.

$$PA = (TP + TN) / (TP + TN + FP + FN) \quad (1)$$

$$MPA = \frac{1}{k} \sum_{i=0}^{k} PA \quad (2)$$

$$MIoU = [\ TP / (TP + FP + FN) + TN / (TN + FN + FP)\ ] / 2 \quad (3)$$

## 3. Results and Discussion

**Model of UNet++ Segmentation.** The UNet++ used in this work is based on the classic UNet[15] model and improves its accuracy by adding parallel multi-scale information transmission. As shown in **Figure 1**, the network structure of UNet++ consists of two main parts: the contraction path (down-sampling) and the expansion path (up-sampling). In the contraction path, the model uses a classic convolutional neural network structure to gradually reduce the size of the feature maps by stacking convolutional and pooling layers. In the expansion path, the model increases the size of the feature maps by stacking convolutional and up-sampling layers and finally generates segmentation results with the same size as the input image. In UNet++, there are multiple connections between the expansion path and the contraction path, which allow the model to perform information transmission at multiple scales and improve its accuracy. The advantage of UNet++ is that it can capture the features of images at different scales, thereby improving the accuracy of segmentation. Compared with other segmentation models, UNet++ has higher robustness and can handle targets of various sizes and shapes. It is suitable for various semantic segmentation tasks, including medical, remote sensing, and electron microcopy, among others. So UNet++ is essentially a deep network developed specifically for electron microscopy image processing, which is why we chose it as the model. For the size of our dataset and the contrast characteristics of the STEM images, we have modulated the parameters of the model (see **Methods 2.5** for details) to achieve better segmentation performance.

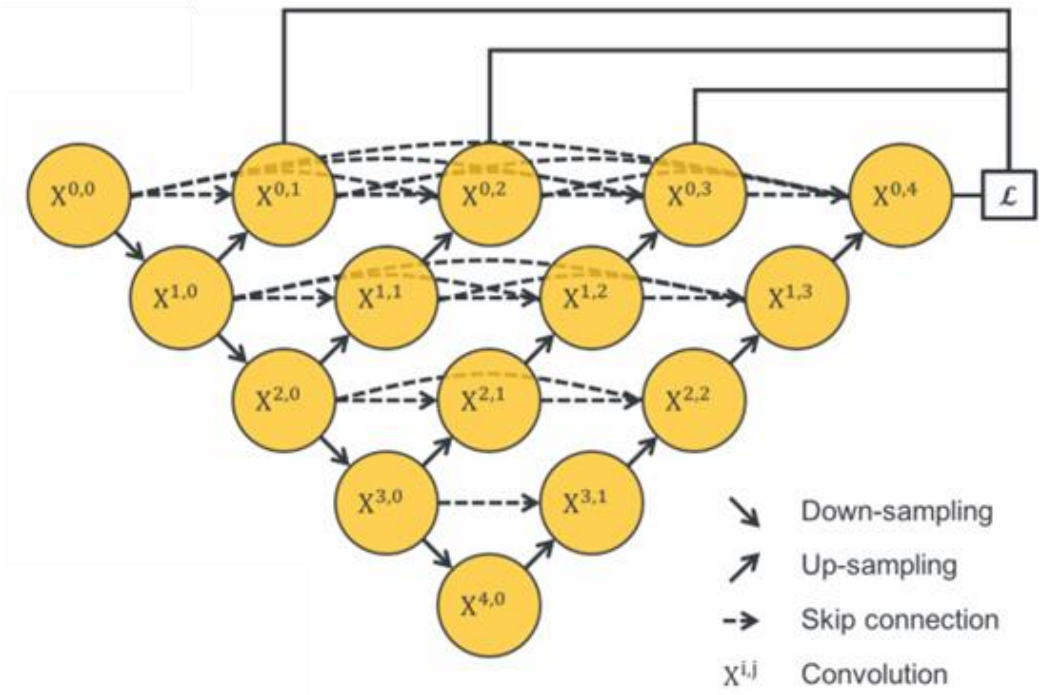


**Figure 1**. Network structure of the UNet+.

**Analysis of predictive metrics.** Based on general deep network metrics, we selected four metrics, Loss, Pixel Accuracy (PA), Mean Pixel Accuracy (MPA) and Mean Intersection over Union (MIoU), to examine the performance of the UNet++ model. The difference between ground truth and prediction, denoted as Loss, can be used to determine the gap between the expected and actual performance of the model. As shown in **Figure 2a**, for both train and test sets, the Loss first decreases rapidly with epochs and then gradually stabilizes, and there is no noise and random fluctuations after stabilization, indicating that it is a good fit rather than underfit or overfit. Based on the ground truth and prediction we derived the confusion matrix to obtain the PA, which is the accuracy of the model, with iteration in **Figure 2b**. It can be found that both train and test sets converge to 99.5% and 99.1%, respectively, after 10 epochs, indicating that the model has excellent accuracy. It also shows that the prediction of labeled NPs can reach 99.6% accuracy, which is better than 59% of the previous work using U-Net (Catherine K Groschner et al., 2023[16]) and can be applied to automatic analysis. To analyze the accuracy of segmentation under the NPs class more precisely, we introduced the pixel accuracy after class averaging, MPA. As shown in **Figure 2c**, both train and test sets also stop rising sharply after 10 epochs, but the accuracy after final

stabilization decreases compared to PA, 94.6% and 84.7% respectively, indicating that the model still has room for improvement. It is worth mentioning that the test set has significant fluctuations and a decreasing trend in the 10-45 epoch interval, which is a false high phenomenon and is common in deep training and does not affect the final results. MIoU, as a standard metric for semantic segmentation, must be evaluated for all segmentation problems to reflect the class average of the ratio of the intersection and sum of ground truth and predicted values. As shown in **Figure 2d**, the train and test sets' basically did not change much after 10 epochs, while the final values indicate that the similarity between prediction and ground truth reached 89.4% and 80.8%, which is significantly better than the work using other model (62.4% of Kevin P. Treder et al., 2023[17]).

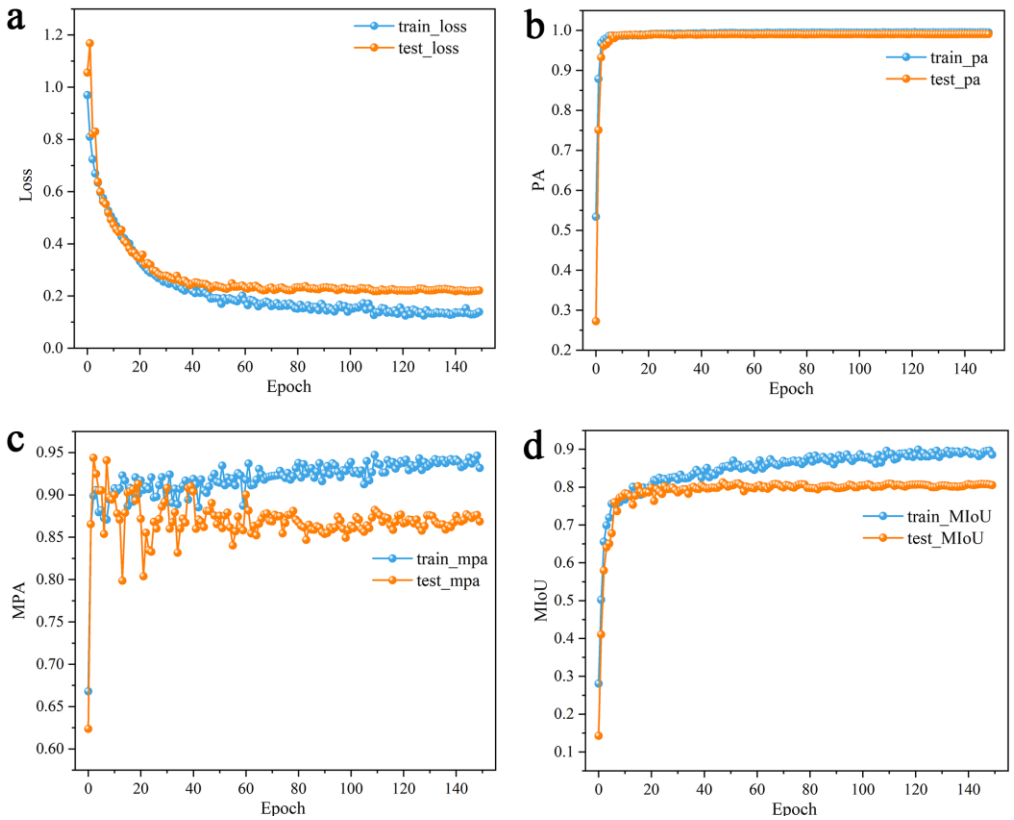


**Figure 2**. Performance metrics for train and test sets.

**Analysis of recognition examples.** Through an example in **Figure 3a, b** where UNet++ achieves excellent performance, we find that the trained model has the best recognition results for high-resolution STEM images with a general dimension of ~40 $nm^2$. By detailed comparison, visible NPs are recognized and their predicted values are with ground truth,

indicating that in in this example, 100% accuracy is achieved. **Figure 3c, d** are another high-resolution case with slightly worse recognition, and the accuracy is 97.6%, which is basically considered negligible for this degree of error rate. For the excellent performance of our UNet++ here, we attribute it to the fact that 70% of the input dataset are high-resolution, and the diversity and large number of samples make the model segment best in this resolution feature.

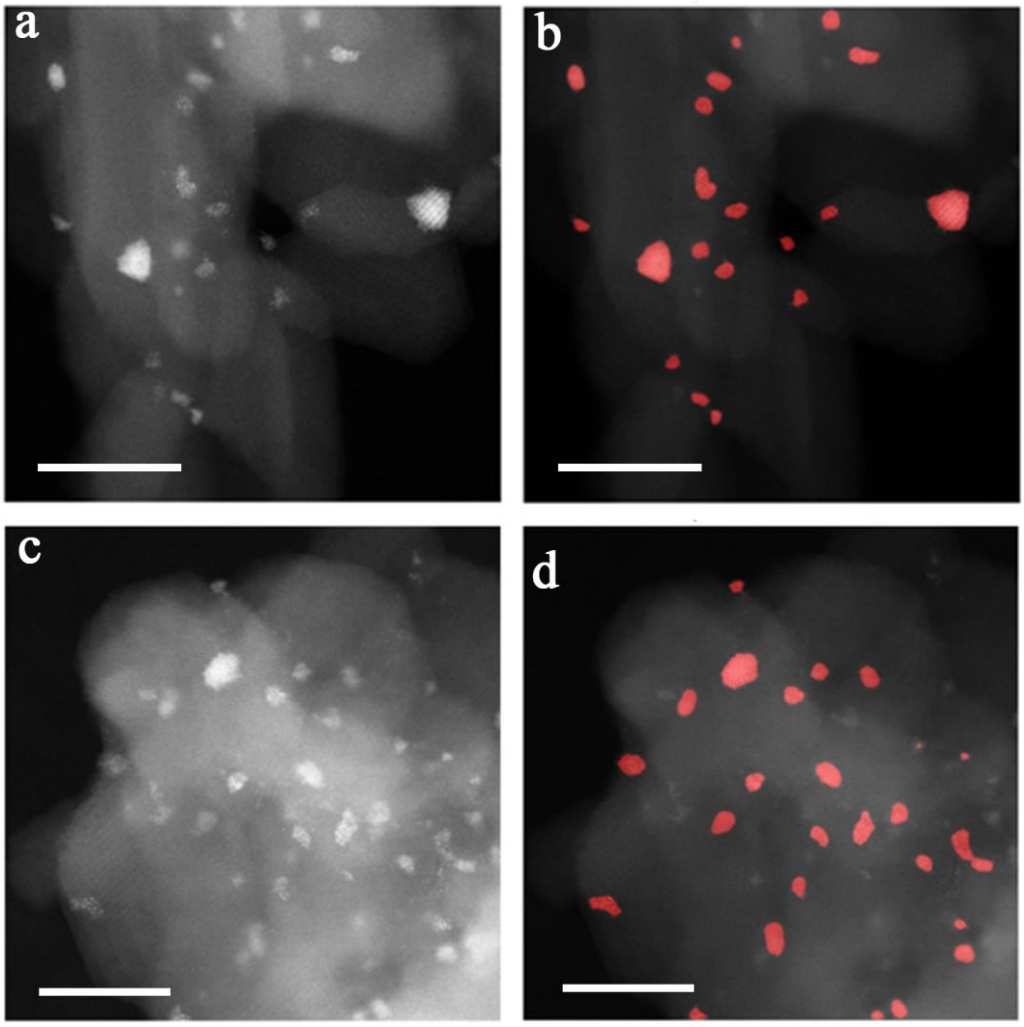


**Figure 3**. Representative test results for semantic segmentation of the UNet++ model at high-resolution. The scale bar is 10nm.

At the same time, for medium-resolution images, the average size of NPs is an order of magnitude smaller than in the high-resolution case, while the training input share for that resolution is also lower, at only 30%. This means that at this resolution feature, the

recognition difficulty rises further and requires higher pixel accuracy of the model. An example of a STEM image with a scale of 20 nm is shown in **Figure 4a, b**, which is a case where NPs with huge differences in size (area ratio of about 47 times) co-exist. Surprisingly, UNet++ still achieves an excellent accuracy (89%), and both the largest NP and the smallest NP are segmented out. This proves the extremely good performance of our model on size diversity. Further through **Figure 4c,d** we verify that with uniformly scattered smaller NPs (compared to **Figure 3**), UNet++ still achieves an accuracy of 82%, indicating that the model also performs quite well in terms of morphological diversity.

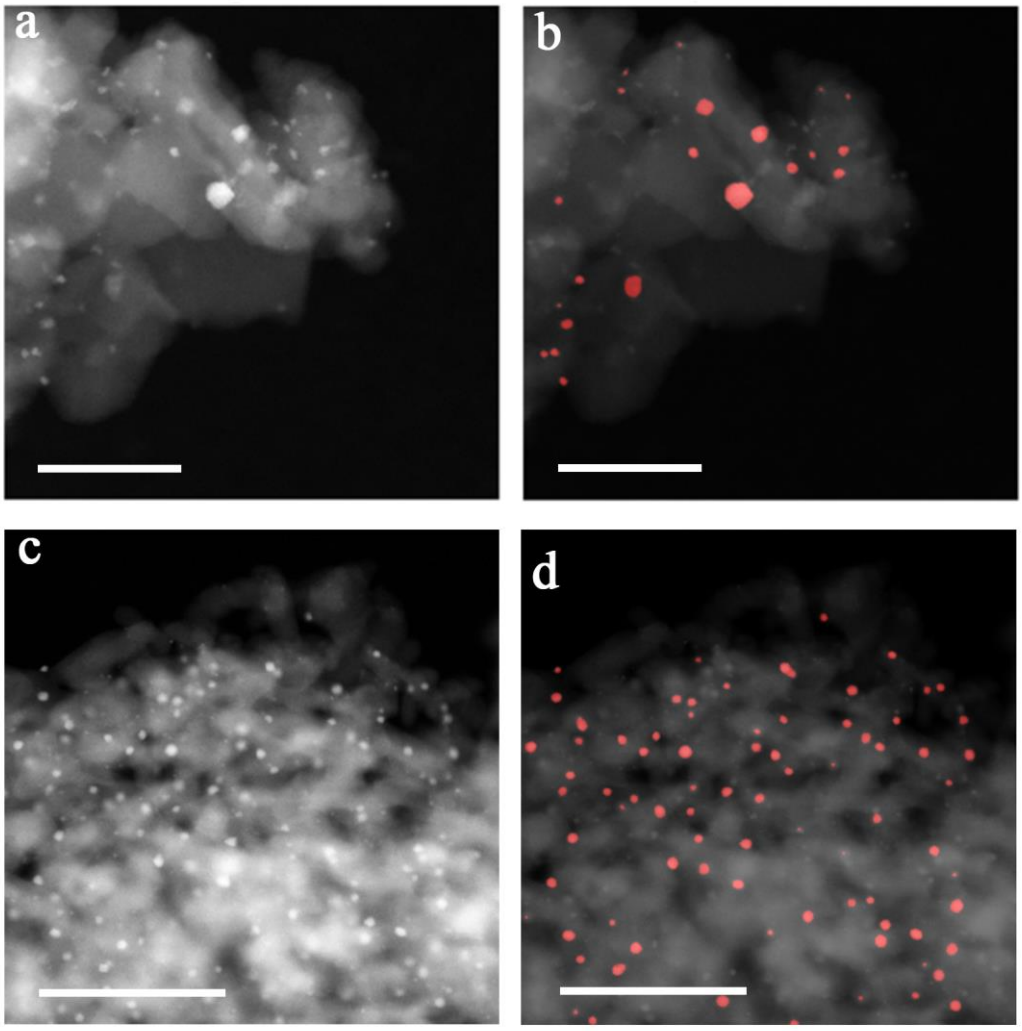


**Figure 4**. Representative test results for semantic segmentation of the UNet++ model at medium-resolution. The scale bar of a and b is 20nm.

However, when the model is applied to low-resolution features, the results are not satisfactory. As shown in **Figure 5a,b**, which is an example of an image with greater dimensionality than **Figure 4c,d**, it has an accuracy of only 65%. Another example shown in **Figure 5c,d**, an image with a dimension of 189.65 $nm^2$ (about five times that of **Figure 3**), shows a segmentation result with the worst accuracy of 5%. It is worth mentioning that we did not use this resolution feature for training, the upper limit of our training dataset is 76.876 $nm^2$, so this is the reason for the poor performance in this case. To address this, we propose to expand the training dataset to cover the entire dimensional range in the next step to achieve accurate recognition with full-resolution, morphological diversity, and size diversity.

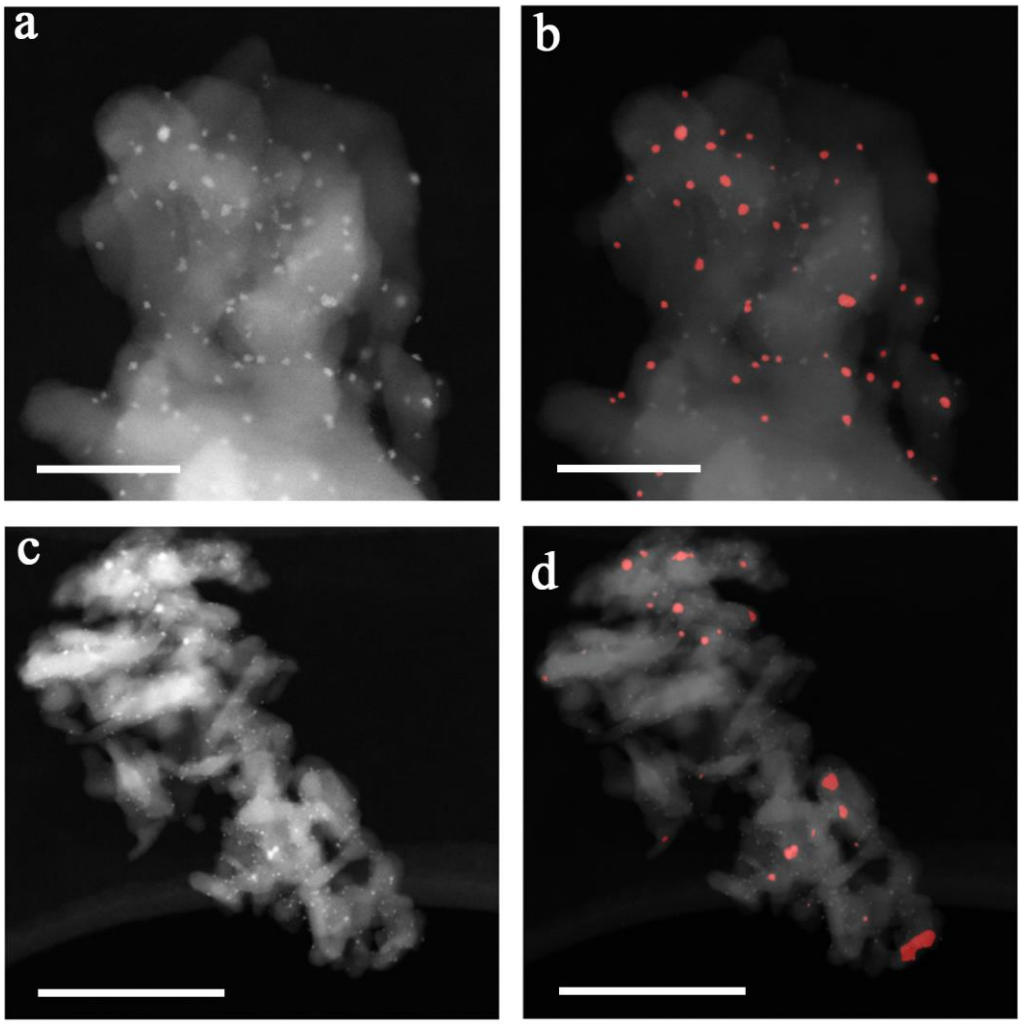


**Figure 5**. Representative test results for semantic segmentation of the UNet++ model at low-resolution. The scale bar of a and b is 20nm, the scale bar of c and d is 50nm.

## 4. Outlooks

Based on the analysis of the above results, the future planning of this study is broadly divided into three stages.

In the first stage, the manually labeled dataset will be further expanded to about 100 images, so that the data will cover the entire resolution range (dimension range of 30 $nm^2$-200 $nm^2$), while the labeling level will be more refined than before. Using the expanded data, an optimized model will be trained again based on the current model pipeline to achieve more accurate recognition of NPs at different magnifications and different focus levels, with different sizes and different shapes. This optimized UNet++ model can basically achieve the identification of NPs on complex substrates of various STEM images and can be extended to the whole field of industrial catalyst microscopy analysis.

The second stage of this research is to achieve label-free, i.e., to simulate STEM images with ground truth based on MULTEM[18], and then use these simulated images to generate automatically labeled datasets in large quantities with the help of AI reimaging, and finally to achieve fully automated STEM image analysis. In short, our goal is to make the model automatically taggable, which will be the basis for achieving self-evolution in the third stage.

The third stage will focus on the long-term purpose and is the final step to integrate this research into an add-on package to be applied to the electron microscopy equipment. This add-on package is planned to extend two functions. One is to implement real-time analysis, i.e., whenever a researcher is operating the electron microscope to capture an image, our package can analyze it in real time and automatically add it to the dataset to give statistical information about the features of all the images that have been captured. This will make electron microscopy image acquisition more efficient and scientific, because the real-time statistical information can help them grasp the sample sampling situation and know when the data is satisfied to stop the acquisition, which was previously mainly judged by the researcher's experience. The second feature is the self-evolution of the network. When a sample is switched to a material that has never been used for training before, our add-on package can automatically learn the features of the new sample and update its own network, enabling self-adaptation to material diversity.

## 5. Conclusions

In summary, the need for high-throughput data analysis, an important feature available in next-generation scanning transmission electron microscopy (STEM), is driving the development of deep learning (DL)-based feature extraction, also known as fully-automated STEM image analysis. In the field of industrial catalysts, the morphological information extraction of nanoparticles (NPs) is crucial for the construction of structure-catalytic performance relationships, but the identification and analysis of NPs on STEM images are currently mostly performed by time-consuming manual methods, which is an urgent bottleneck for the upcoming high-throughput data-driven STEM. We successfully utilize the deep network-based UNet++ pipeline to achieve image segmentation of NPs in complex industrial catalyst systems for the first time, with the advantages of extremely lightweight, high accuracy, and ultra-fast. Its Pixel Accuracy (PA), Mean Pixel Accuracy (MPA), and Mean Intersection over Union (MIoU) can reach 99.1%, 84.7% and 80.8%, respectively, better than previous works. Our work has successfully achieved semi-automatic STEM image analysis without parameterization, which we plan to further refine to fully automatic with the help of simulations and emerging AI reimaging tools and integrate into an add-on software for real-time analysis of electron microscopy equipment towards engineering applications.

## References

(1) Ruska, E. The development of the electron microscope and of electron microscopy. *Rev. Mod. Phys.* **1987**, *59* (3), 627.

(2) Varela, M.; Lupini, A. R.; Benthem, K. v.; Borisevich, A. Y.; Chisholm, M. F.; Shibata, N.; Abe, E.; Pennycook, S. J. Materials characterization in the aberration-corrected scanning transmission electron microscope. *Annu. Rev. Mater. Res.* **2005**, *35*, 539-569.

(3) Tan, W.; Xie, S.; Le, D.; Diao, W.; Wang, M.; Low, K.-B.; Austin, D.; Hong, S.; Gao, F.; Dong, L. Fine-tuned local coordination environment of Pt single atoms on ceria controls catalytic reactivity. *Nat. Commun.* **2022**, *13* (1), 7070.

(4) Chen, S.; Oh, H. S.; Gludovatz, B.; Kim, S. J.; Park, E. S.; Zhang, Z.; Ritchie, R. O.; Yu, Q. Real-time observations of TRIP-induced ultrahigh strain hardening in a dual-phase CrMnFeCoNi high-entropy alloy. *Nat. Commun.* **2020**, *11* (1), 826.

(5) Tao, F.; Salmeron, M. In situ studies of chemistry and structure of materials in reactive environments. *Science* **2011**, *331* (6014), 171-174.

(6) Zhang, S.; Chen, C.; Cargnello, M.; Fornasiero, P.; Gorte, R. J.; Graham, G. W.; Pan, X. Dynamic structural evolution of supported palladium–ceria core–shell catalysts revealed by in situ electron microscopy. *Nat. Commun.* **2015**, *6* (1), 7778.

(7) Butler, K. T.; Davies, D. W.; Cartwright, H.; Isayev, O.; Walsh, A. Machine learning for molecular and materials science. *Nature* **2018**, *559* (7715), 547-555.

(8) LeBeau, J.; Kumar, A.; Hauwiller, M. A universal scripting engine for transmission electron microscopy. *Microsc. Microanal.* **2020**, *26* (S2), 2958-2959.

(9) Buslaev, A.; Iglovikov, V. I.; Khvedchenya, E.; Parinov, A.; Druzhinin, M.; Kalinin, A. A. Albumentations: fast and flexible image augmentations. *Information* **2020**, *11* (2), 125.

(10) Zhou, Z.; Rahman Siddiquee, M. M.; Tajbakhsh, N.; Liang, J. UNet++: A Nested U-Net Architecture for Medical Image Segmentation. In *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*, **2018**, pp 3-11.

(11) He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, **2016**; pp 770-778.

(12) Lin, T.-Y.; Goyal, P.; Girshick, R.; He, K.; Dollár, P. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, **2017**; pp 2980-2988.

(13) Duque-Arias, D.; Velasco-Forero, S.; Deschaud, J.-E.; Goulette, F.; Serna, A.; Decencière, E.; Marcotegui, B. On power Jaccard losses for semantic segmentation. In *VISAPP 2021: 16th International Conference on Computer Vision Theory and Applications*, **2021**.

(14) You, Y.; Li, J.; Reddi, S.; Hseu, J.; Kumar, S.; Bhojanapalli, S.; Song, X.; Demmel, J.; Keutzer, K.; Hsieh, C.-J. Large batch optimization for deep learning: Training bert in 76 minutes. *arXiv preprint arXiv:1904.00962* **2019**.

(15) Khaireddin, Y.; Chen, Z. Facial emotion recognition: State of the art performance on FER2013. *arXiv preprint arXiv:2105.03588* **2021**.

(16) Groschner, C. K.; Choi, C.; Scott, M. C. Machine Learning Pipeline for Segmentation and Defect Identification from High-Resolution Transmission Electron Microscopy Data. *Microsc. Microanal.* **2021**, *27* (3), 549-556.

(17) Treder, K. P.; Huang, C.; Bell, C. G.; Slater, T. J. A.; Schuster, M. E.; Özkaya, D.; Kim, J. S.; Kirkland, A. I. nNPipe: a neural network pipeline for automated analysis of morphologically diverse catalyst systems. *npj Comput. Mater.* **2023**, *9* (1), 18.

(18) Lobato, I.; Van Dyck, D. MULTEM: A new multislice program to perform accurate and fast electron diffraction and imaging simulations using Graphics Processing Units with CUDA. *Ultramicroscopy* **2015**, *156*, 9-17.

# **Appendix A:** Source code of main.py

```
import warnings
warnings.filterwarnings('ignore')
import os, tqdm, cv2
import numpy as np
np.random.seed(0)
from PIL import Image
import albumentations as A
import torch.utils.data as data

import torch, time, datetime, tqdm
from sklearn.metrics import confusion_matrix
from segmentation_models_pytorch import create_model
from segmentation_models_pytorch import losses

class Mydataset(data.Dataset):
    def __init__(self, data, aug=None):
        self.data = data
        self.aug = aug

    def __getitem__(self, index):
        image, label = cv2.imread(f'{self.data[index]}/img.png'), cv2.imread(f'{self.data[index]}/label.png', 0)
        if self.aug is not None:
            aug = self.aug(image=image, mask=label)
            image, label = aug['image'], aug['mask']
        label[label > 0] = 1
        return np.transpose(image, axes=[2, 0, 1]), np.array(label, dtype=np.int)

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

def get_data(BATCH_SIZE=32):
    with open('train.txt') as f:
        train_data = list(map(lambda x:x.strip(), f.readlines()))

    with open('test.txt') as f:
        test_data = list(map(lambda x:x.strip(), f.readlines()))

    train_aug = A.Compose([
        A.Resize(512, 512),
        A.Flip(p=0.5),
        A.ShiftScaleRotate(p=0.5),
        A.GaussNoise(p=0.5),
        A.Normalize(),
    ])

    test_aug = A.Compose([
        A.Resize(512, 512),
        A.Normalize(),
    ])

    train_dataset = Mydataset(train_data, train_aug)
    train_dataset = torch.utils.data.DataLoader(train_dataset, BATCH_SIZE, shuffle=True, num_workers=4, pin_memory=True)
    test_dataset = Mydataset(test_data, test_aug)
    test_dataset = torch.utils.data.DataLoader(test_dataset, BATCH_SIZE, shuffle=False, num_workers=4, pin_memory=True)
    return train_dataset, test_dataset

def metrice(cm):
    pa = np.diag(cm).sum() / (cm.sum() + 1e-7)

    mpa_arr = np.diag(cm) / np.maximum(cm.sum(axis=1), 1)
    mpa = np.nanmean(mpa_arr)

    MIoU = np.diag(cm) / np.maximum((np.sum(cm, axis=1) + np.sum(cm, axis=0) - np.diag(cm)), 1)
    MIoU = np.nanmean(MIoU)

    return pa, mpa, MIoU

def cal_cm(y_true, y_pred):
    y_true, y_pred = y_true.to('cpu').detach().numpy(), np.argmax(y_pred.to('cpu').detach().numpy(), axis=1)
    y_true, y_pred = y_true.reshape((-1)), y_pred.reshape((-1))
    cm = confusion_matrix(y_true, y_pred, labels=list(range(2)))
    return cm

if __name__ == '__main__':
    EPOCH, BATCH_SIZE = 150, 4
    DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    train_dataset, test_dataset = get_data(BATCH_SIZE)
    model = create_model('UnetPlusPlus', encoder_name='resnet34', classes=2).to(DEVICE)

    optimizer = torch.optim.AdamW(params=model.parameters(), lr=3e-4, weight_decay=5e-4)
    lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.9)
    # lr_scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=5, eta_min=1e-5)

    loss_fl = losses.FocalLoss(mode='multiclass', alpha=0.25)
    loss_jd = losses.JaccardLoss(mode='multiclass')

    with open('train.log', 'w+') as f:
            f.write('epoch,train_loss,test_loss,train_pa,test_pa,train_mpa,test_mpa,train_miou,test_miou')
    best_miou = 0
    print('{} begin train on {}!'.format(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S'), DEVICE))
    for epoch in range(EPOCH):
        model.to(DEVICE)
        model.train()
        train_loss, train_cm = 0, np.zeros(shape=(2, 2))
        begin = time.time()
        for x, y in tqdm.tqdm(train_dataset, desc='Epoch {}/{} train stage'.format(epoch + 1, EPOCH)):
            x, y = x.to(DEVICE), y.to(DEVICE).long()
            pred = model(x.float())
            l = loss_fl(pred, y) + loss_jd(pred, y)

            l.backward()
```

```
    optimizer.step()
    optimizer.zero_grad()

    train_loss += float(l.data)
    train_cm += cal_cm(y, pred)
train_loss /= len(train_dataset)
train_pa, train_mpa, train_miou = metrice(train_cm)

val_loss, val_cm = 0, np.zeros(shape=(2, 2))
model.eval()
with torch.no_grad():
    for x, y in tqdm.tqdm(test_dataset, desc='Epoch {}/{} val stage'.format(epoch + 1, EPOCH)):
        x, y = x.to(DEVICE), y.to(DEVICE).long()
        pred = model(x.float())
        l = loss_fl(pred, y) + loss_jd(pred, y)
        val_loss += float(l.data)
        val_cm += cal_cm(y, pred)
val_loss /= len(test_dataset)
val_pa, val_mpa, val_miou = metrice(val_cm)

if val_miou > best_miou:
    best_miou = val_miou
    model.to('cpu')
    torch.save(model, 'model.pt')
print(
    '{} epoch:{}, time:{:.2f}s, lr:{:.6f}, train_loss:{:.4f}, val_loss:{:.4f}, train_pa:{:.4f}, val_pa:{:.4f}, train_mpa:{:.4f}, val_mpa:{:.4f},
        datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S'),
        epoch + 1, time.time() - begin, optimizer.state_dict()['param_groups'][0]['lr'], train_loss, val_loss,
        train_pa, val_pa, train_mpa, val_mpa,
        train_miou,
        val_miou
    ))
with open('train.log', 'a+') as f:
    f.write('\n{}, {:.4f}, {:.4f}, {:.4f}, {:.4f}, {:.4f}, {:.4f}, {:.4f}, {:.4f}'.format(
        epoch, train_loss, val_loss, train_pa, val_pa, train_mpa, val_mpa, train_miou, val_miou
    ))
lr_scheduler.step()
```

## **Appendix B:** Source code of predict.py

```
import os, glob, cv2, shutil
from PIL import Image
import numpy as np
import nibabel as nib
import matplotlib.pylab as plt
np.random.seed(0)

import albumentations as A
import torch, tqdm
import numpy as np

if __name__ == '__main__':
    colors = [(0, 0, 0), (0, 0, 255)]
    DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    test_aug = A.Compose([
        A.Resize(512, 512),
        A.Normalize(),
    ])

    model = torch.load('model.pt').to(DEVICE)
    model.eval()

    path = 'img'
    save_path = 'result'
    if os.path.exists(save_path):
        shutil.rmtree(save_path)
    os.makedirs(save_path, exist_ok=True)

    for i in os.listdir(path):
        ori_image = cv2.imread(f'{path}/{i}')
        image = test_aug(image=ori_image)['image']
        image = np.expand_dims(np.transpose(image, axes=[2, 0, 1]), axis=0)
        image = torch.from_numpy(image).to(DEVICE).float()
        output = model(image).cpu().detach().numpy()[0].argmax(0)
        ratio = np.sum(output[output == 1]) / (output.shape[0] * output.shape[1])

        output = np.reshape(np.array(colors, np.uint8)[np.reshape(output, [-1])], [512, 512, -1])
        output = cv2.resize(output, (ori_image.shape[1], ori_image.shape[0]), interpolation=cv2.INTER_NEAREST)
        output = cv2.addWeighted(ori_image, 0.5, output, 0.5, 0)

        plt.figure(figsize=(10, 5))

        plt.subplot(1, 2, 1)
        plt.imshow(cv2.cvtColor(ori_image, cv2.COLOR_BGR2RGB))
        plt.title('ori_image')
        plt.axis('off')

        plt.subplot(1, 2, 2)
        plt.imshow(cv2.cvtColor(output, cv2.COLOR_BGR2RGB))
        plt.title(f'pred ratio:{ratio:.4f}')
        plt.axis('off')

        plt.tight_layout()
        plt.savefig(f'{save_path}/{i.split(".")[0]}')
        # plt.show()
```

## **Appendix C:** Source code of metrice.py

```
import os, glob
from PIL import Image
import numpy as np
np.random.seed(0)

import albumentations as A
import torch, tqdm
import numpy as np
from main import get_data
from main import get_data, metrice, cal_cm

if __name__ == '__main__':
    BATCH_SIZE = 4
    DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    train_dataset, val_dataset, test_dataset = get_data(BATCH_SIZE)

    model = torch.load('model.pt').to(DEVICE)
    model.eval()
    test_cm = np.zeros(shape=(2, 2))
    with torch.no_grad():
        for x, y in tqdm.tqdm(test_dataset, desc='test stage'):
            x, y = x.to(DEVICE), y.to(DEVICE).long()

            pred = model(x.float())
            test_cm += cal_cm(y, pred)
    val_pa, val_mpa, val_miou = metrice(test_cm)
    print('test set pixel_accuracy:{:.3f} mean_perclass_accuracy:{:.3f} mean_iou:{:.3f}'.format(val_pa, val_mpa, val_miou))
```

## **Appendix D:** Source code of labelme2png.py

```
import os, shutil, random
random.seed(0)
from tqdm import tqdm

for i in tqdm(os.listdir('data_annotated')):
    os.system('labelme_json_to_dataset data_annotated/{}'.format(i))

data = [f'data_annotated/{i}' for i in os.listdir(r'data_annotated') if os.path.isdir(f'data_annotated/{i}')]
random.shuffle(data)
with open('train.txt', 'w+') as f:
    f.write('\n'.join(data[:int(len(data) * 0.8)]))

with open('test.txt', 'w+') as f:
    f.write('\n'.join(data[int(len(data) * 0.8):]))
```