Segmentation of herbarium sheets

Sichu Huang

C21042655

Msc Advanced Computer Science

Supervisor: Paul L Rosin

School of Computer Science and Informatic

Cardiff University

Table of Contents

Abstract

Semantic segmentation has been proposed as a tool to accelerate the digitisation workflows of natural history collections. However, the performance of most existing segmentation models strongly rely on the quality of GPU and will occupy high volume of memory. To develop a stable, portable, flexible, resilient, fast in speed and lightweight segmentation model requires not only the ability of processing different datasets with minimal time and memory cost, but also the ability to assure the high performance with minimal precision loss in mathematical calculation. In addition, the model should be flexible and portable for rebuilding and export API for frontend and backend for other use. This dissertation evaluates the differences among the semantic segmentation model and other state-of-the-art models and implements a DeepLabv3-Plus-Pytorch model based on modified MobileNetv2 network. And design a user-friendly train-predict semantic segmentation model that provide detailed guidelines, comments and automatic generation scripts. The model will run with mixed precision training FP16 and fine tune with freeze 50 epochs to minimize the memory and time cost on training. Also, it applies Adam optimizer with cosine learning rate decay to ensure the smooth loss convergence curve and achieve a low-level loss, high intersection over union, high per pixel precision and accuracy. Moreover, it proposes to replace the traditional Cross-Entropy loss with the combination of Dice loss and Focal loss for the implemented DeepLabv3+ model to solve the imbalanced dataset problem on the herbarium sheets segmentation works. Our proposed DeepLabv3-Plus-Pytorch could help the researchers and staffs in related field to efficiently either train new model on different natural history collections datasets or predict brand new unlabelled datasets. It has the ability to train a model in 4 hours on the dataset contains 1000 images without using multi-threads or high-performance GPU and predict 500 unlabelled $3744 * 5616$ images in $400ms$ on single thread CPU mode without losing the image quality. And the result of our model achieves $mIOU = 90.65\%, mPA = 95.80\%, mPrecision = 94.33\%, mRecall = 95.80\%$ and a good performance on prediction.

1. Introduction

Natural history collections are essential data for various research, analysis and reference in anatomy, physiology, biochemistry, microbiology, and ecology [1]. Digitizing these collections are in need to increase the global accessibility to scientific researchers, the efficiency of collections management and reduce the possibility of overhandling the fragile physical samples and deterioration of valuable. However, the natural history collections vary in language, imaging methodology, interpretation of certain content and storage media type (images, videos). Herbarium sheet is a unique natural history collection, which preserves plant specimens and associated data that could provide important scientific data for discovering new species, revealing ecological and evolutionary responses, and examining the impact of global climate change [3, 4]. Similar to other natural history collections, the specimens inside herbarium sheets vary in shape, colour and texture, and the consistency of other data such as labels, barcode, colour chart also vary between institutions and collections.

The non-homogeneous herbarium collections cause the traditional digitisation workflows, which rely on manual process, slow in speed with processing and publishing the online data. There is need to apply computer science methodologies to support the automation and increase the speed the information extraction from millions of diverse physical collections. Though in previous studies, several whole-drawer scanning technologies have been developed to facilitate more effective collection management, virtual curation, and digitization of collections [5, 6], it remains a laborious and largely manual process for specimens. In addition, even though the other textual data in the collections, such as labels, could be captured and processed by optical character recognition (OCR) methods, it is complicated to localize the old handwritten text [7]. To facilitate the automated identification of natural history collections images, the artificial intelligence could be used. By applying deep learning-based Computer Vision approaches, such as object detection, instance segmentation, panoptic segmentation and semantic segmentation, it is feasible to not only preserve the fragile physical collections but also make the digitisation workflow of natural history collections more automatic and less laborious [2, 8, 9, 10]. In particular, the segmentation is more challenging as it involves both object detection and localization which is achieved by assigning labels to every pixel in an image.

In 2022, the Cardiff University received 519 herbarium sheets images from Wyoming University, who would like to cooperate to perform semantic segmentation to help accelerate

their digitisation workflow. However, the Wyoming University only offers the image-level labelled dataset without pixel-level labels. And the idea of this project is inspired by the paper "Cross-validation of a semantic segmentation network for natural history collection specimens", also the most part of the train dataset (2000 labelled images) is provided by this paper [12]. Besides the model from the paper, there are also other model candidates, for example, U-Net, PSPNet, DeepLab, etc. This project is going to focus on training different semantic segmentation models with the given labelled dataset, applying them on Wyoming University unlabelled dataset, compare their performance and choose the most suitable model on Wyoming's herbarium sheets dataset. In addition, we will do several experiments and try to improve the performance of the existing model. In addition, as a bonus part of this project, we are going to perform anomaly detection to help the Wyoming University to find the potential mistakes for the herbarium sheets classification. Alternatively, we will perform object detection methodologies, such as the state-of-the-art model YOLOv5.

In this dissertation, there will be 7 sections: Introduction, Aim and objective, Background material, Problem, Approach, Product and Analysis, Conclusions. In the Aim and Objective section, we will concentrate on what our study seeks to achieve and how our study does to deal with the Wyoming University's needs. In the Background material section, we are going to show the background knowledge of the related existing work, commercial products and research papers. We will demonstrate the unique about our solutions and methods we applied to the semantic segmentation model. In the Problem section, we will give sufficient information based on the background to the problem that addressed in our study and illustrate what characteristics make the problem difficult. Also, we will state what benefits are expected to arise as a result of our study. In the Approach section, we will explain our intended approach to addressing the problem, and also how we decided upon this approach. This section will not go into detailed discussion of the mechanics of any chosen methodology, but it will compare different candidate approaches and explain why others were discarded. In the Product and Analysis section, we will clearly explain the requirements, design, implementation of our product in detail. We will also provide the results of the product stage experiments, show our hypothesis after each experiment, demonstrate how we prove the hypothesis, how we improve the product and analysis them. This will include the data argument of dataset, mixed precision training, freeze training, dice loss combined focal loss, cosine learning rate decay, comparison between Adam optimizer and SGD optimizer, and their performance on the herbarium sheets dataset. In the end of the dissertation, we will

provide a stable, portable, flexible, resilient, fast in speed and lightweight semantic segmentation model that could improve the digitisation workflows of herbarium sheets collections for Wyoming University and provide a bonus model that focused on object detection to help them detect the potential mistakes on herbarium specimen classification .

2. Aim and Objectives

Digitisation of the natural history collection could not only protect the fragile specimens but also facilitate easy access and sharing of its data to a wider scientific community. Artificial Intelligence has been proposed as a tool to accelerate the processing of natural history collection images, whose throughput of digitisation workflows could be increased by using deep learning with semantic segmentation. However, the consistency of content inside the collections varies between different institutions and collections, which will affect the performance of segmentation models. Consequently, it needs to find a stable, portable, flexible, resilient, fast in speed and lightweight model to enhance the digitisation workflows of the herbarium sheets collections.

As it mentioned in Introduction section, Cardiff University has received the metadata contains 519 images of the herbarium sheets from Wyoming University in 2022. Our aim is to apply deep learning methodologies to help them to digitize these collections. We are going to:

(1) Firstly, use the deep learning model already trained on different data in the previous project "Cross-validation of a semantic segmentation network for natural history collection specimens" [12].

(2) Secondly, apply the model into Wyoming University's dataset to see if the model could be utilized and see if the results can be improved. If the model is not good enough, we are going to find other approaches from the other candidate models.

(3) Thirdly, trying to improve the performance of the chosen approach. We will draw hypothesis that related to the technical problems, try to prove the hypothesis and fix the problems with experiments.

(4) Lastly, if the semantic segmentation process achieved some kind of success, we will try to perform anomaly detection to help modifying the classification of the Wyoming dataset.

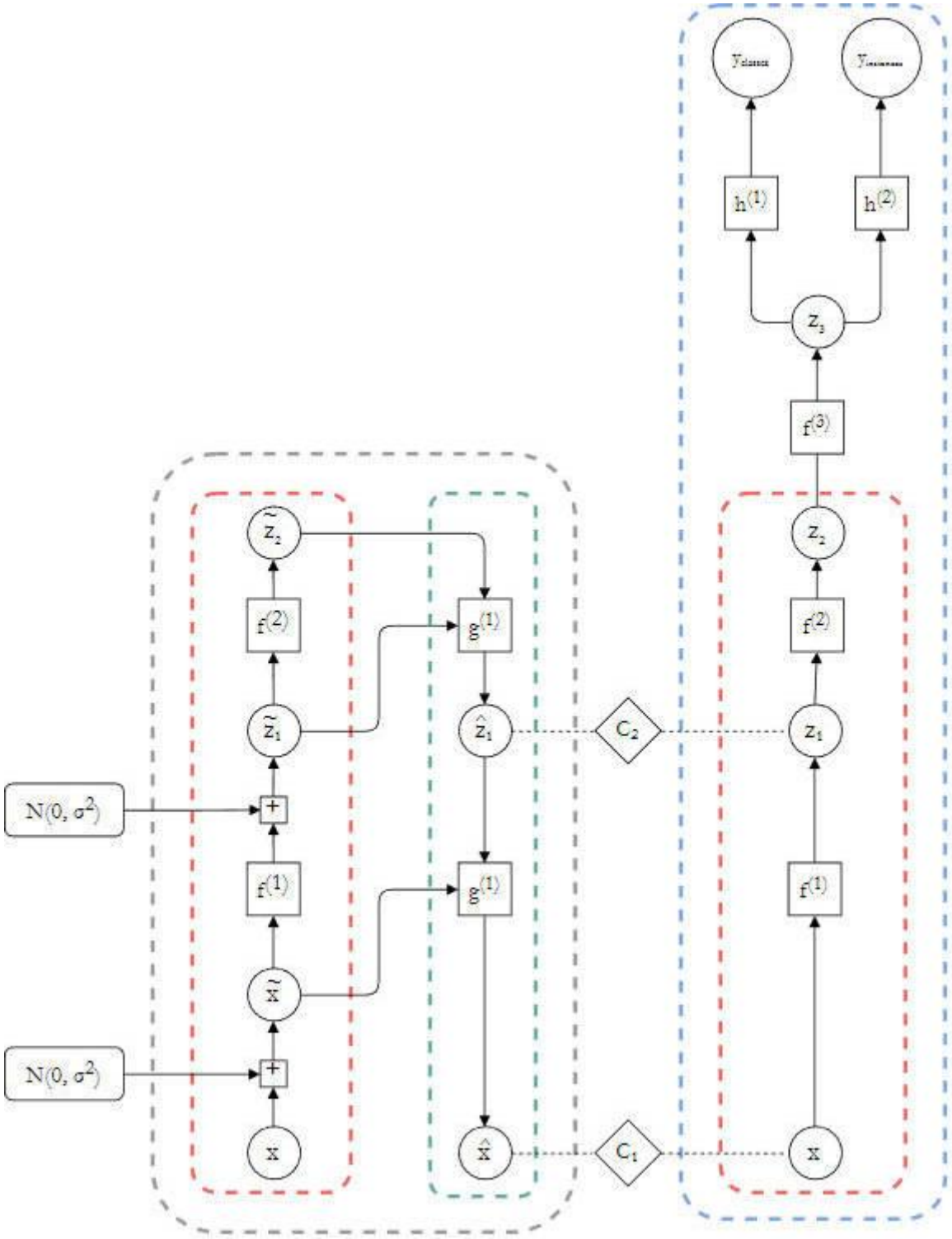


Fig 1. Architecture of the NHM semi-supervised semantic segmentation model [27].

3. Background material

Image segmentation has been a long-term computer vision problem and has been attempted with different algorithms, and convolutional neural network (CNN) in image segmentation tasks have received much attention due to their good performance in image classification tasks [17]. The segmentation methodology in computer vision field can be classified into three categories which include semantic segmentation, instance segmentation and panoptic segmentation. And in the deep learning field, there are supervised learning and semi-supervised learning. According to the previous study [12, 27], the National History Museum (NHM) has developed a semi-supervised semantic segmentation model. As it shown in Fig 1, the left part (grey block) is the reconstruction branch for regularizing by semi-supervised learning and the right part (blue block) is the primary segmentation branch. The key point of this model is to connect the two parts with cost functions $c_1$and $c_2$to minimize the differences between labelled training output and the un-labelled training output. After two High-Level layers of ResNet-18 with pretrained model based on ImageNet, which is f_1 and f_2, the model will feed into another atrous convolutional layer (f_3) for resizing. And the labelled trained result together with the un-labelled trained result will experience another two-layers CNNs in the branch $h^{(1)}$ and $h^{(2)}$ and get the final class and instance result. This semi-supervised model has been proved to have ability process 50 images in 12 minutes [11].

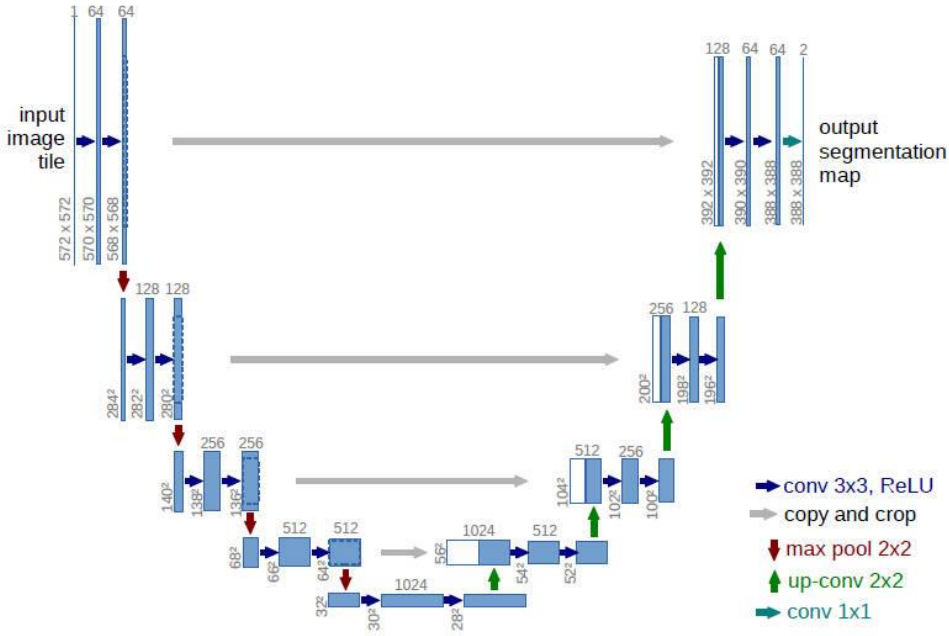


Fig 2. The architecture of U-Net [20].

Besides the semi-supervised learning for semantic segmentation, there are also some supervised learning, such as U-Net, PSPNet and DeepLab [14, 20, 21, 22]. As it shown in Fig 2, the U-Net architecture consists of a contracting path to capture context and a symmetric expanding path that enables precise localization [20]. And this model can obtain good invariance and robustness even with a relatively small number of input data. As it shown in Fig 3, the Pyramid Scene Parsing Network (PSPNet) combine the pre-trained ResNet and the expansion network to extract the features of input image, and then a pyramid parsing module is applied to harvest different sub-region representations, followed by upsampling and concatenation layers to form the final feature representation [21]. As it shown in Fig 4, DeepLabv1 used a fully connected conditional random field (CRF) to improve the ability of the pixel detail capture, and as it shown in Fig 5, DeepLabv2 proposes atrous spatial pyramid pooling (ASPP) based on the previous DCNN-CRF architecture, and the ASPP captures multi-scale image-level features by employing multiple parallel filters with different dilation rates [15]. Then DeepLabv3 introduced the Encoder-Decoder architecture as it shown in Fig 6. It not only employs atrous convolution with upsampled filters to extract dense feature maps but also add the global average pooling with batch normalization after each parallel dilated convolution, which effectively capture global range context [14].

The DeepLabv1 solves the challenge that the max-pooling and downsampling operations in DCNN could compromise the resolution of feature maps and affect the localization accuracy [15]. And the atrous convolution applied in DeepLabv1 could effectively enlarge the field of view of the filter without increasing the amount of computation or resizing the input images, and the CRF could improve the ability to capture more fine details of the feature maps [28]. In summary, the key achievement of DeepLabv1 is that it applies CRF after the deep convolutional neural network (DCNN) and ensure the pixels that adjacent to each other are more likely to be assigned with the same label class, and the probability of the assignment of these labels for the pixels can also be balanced by the iteration. The atrous convolutions ensure the efficient feature extraction and the fully connected CRFs boost the DCNN outputs.

The DeepLabv2 is based on the DeepLabv1 and introduce the ASPP module, improve the performance of segmentation and solve the problem caused by the existence of objects at multiple scales. The ASPP module in DeepLabv2 is inspired by the success of R-CNN spatial pyramid pooling method, which showed that regions of an arbitrary scale can be accurately and efficiently classified by resampling convolutional features extracted at a single scale [29]. In summary, the ASPP could be viewed as processing different feature maps into several

atrous convolutions with different dilation rate and then combine the results into one feature map as the output.

The DeepLabv3 improves the atrous convolutions in the ASPP module as it shown in Fig 7 and Fig 8. It introduced multi-grid method, designed the serial and parallel atrous convolution with $dilation\ rate > 1$ but also different dilation rates to obtain multi-scale content feature extraction. This solved the problem that the low resolution of the feature map will cause the low accuracy when resizing into the original image resolution and solved the problem of poor performance in multi-scale objects segmentation. And the author proposed a three parallel 3×3 convolution branches with $rates = (6, 12, 18)$ inside the ASPP module, and the dilation rate is actually the multiplication of dilation rate and unit rate of the convolutions: $rate = rate \times unit\ rate$ [14]. In other words, the input of the ASPP module in DeepLabv3 is the output of the DCNN, which is shown as Block4 in the Fig 8, and the output of the ASPP module will contact with another 1×1 convolution and propose to incorporate image-level features. Nevertheless, the DeepLabv3+ is an upgrade version of DeepLabv3, which replace all the max-pooling with $stride = 2$ convolutions. Additionally, it added batch normalization and $ReLU$ active function after each of the parallel 3×3 convolution branch inside the ASPP module. More details about the DeepLabv3 and DeepLabv3+ will be discussed in the Product and Analysis section of this dissertation.

As it mentioned in the Aim and Objectives section, the bonus part of our project is to find a way to help Wyoming University to identify the potential mistakes for classification and improve the digitisation workflows of the herbarium sheets. Recently others have taken a similar deep learning approach to improve the digitisation workflows of the natural history collections [18, 19]. However, they are focusing on the object detection, which is another category of computer vision. Although, in this project object detection based on YOLOv5 will be also performed to the Wyoming data because the object detection methodologies have the potential to detect and localize the different classes of the Specimen objects.

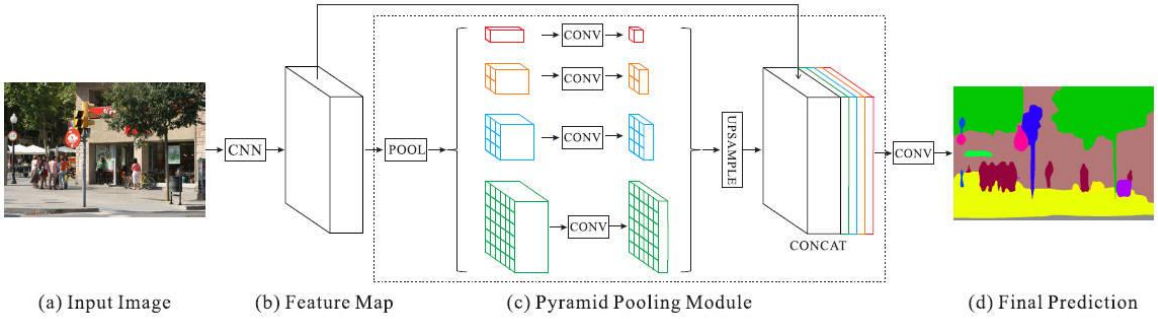


Fig 3. The architecture of PSPNet [21].

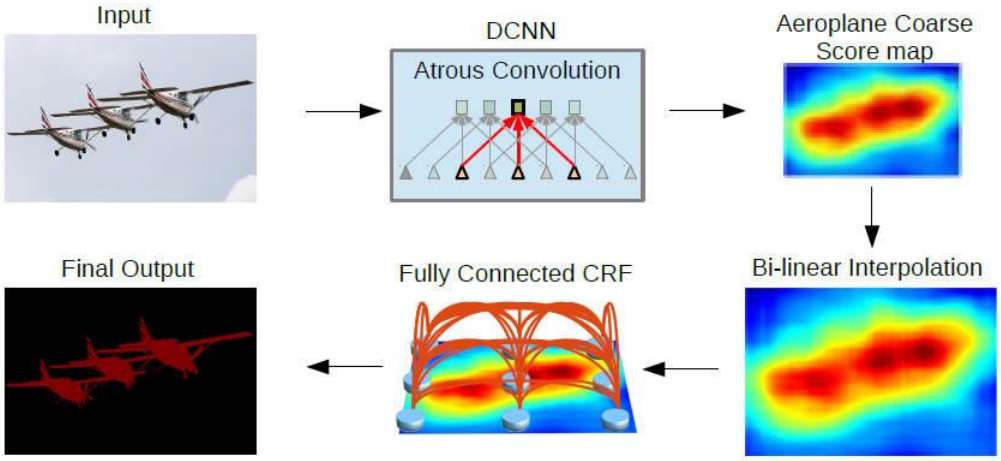


Fig 4. The DCNN and CRF architecture of DeepLabv1 and DeepLabv2 [15].

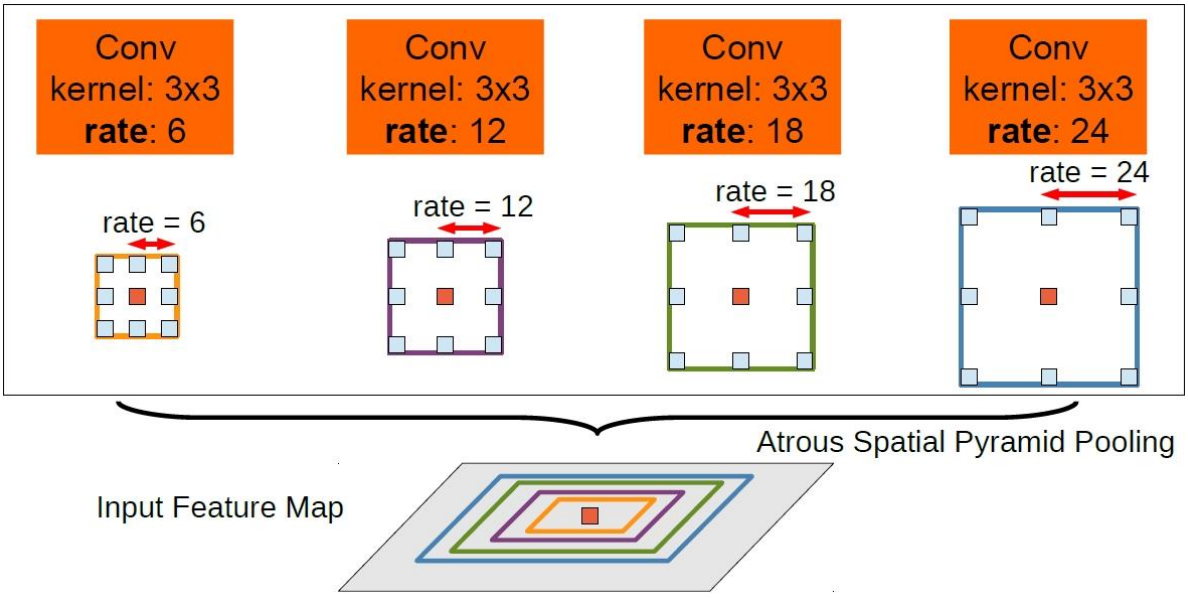


Fig 5. The ASPP module of DeepLabv2 [15].

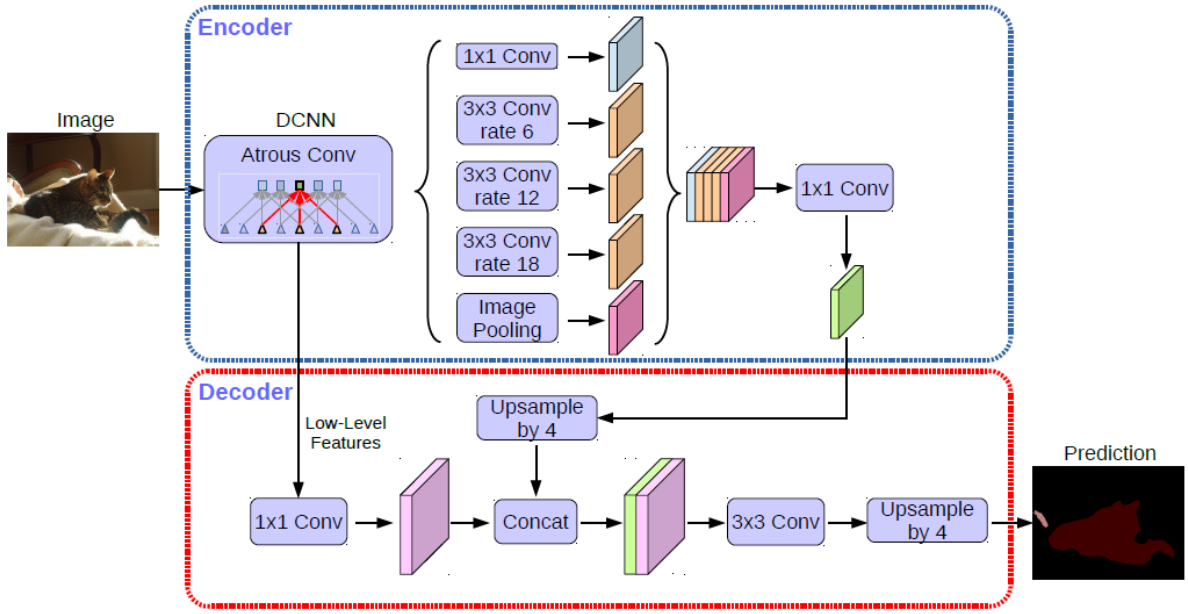
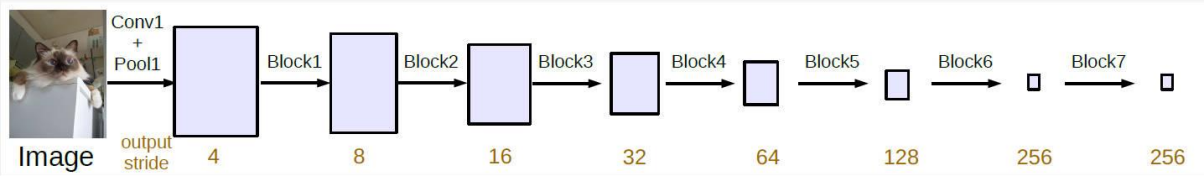
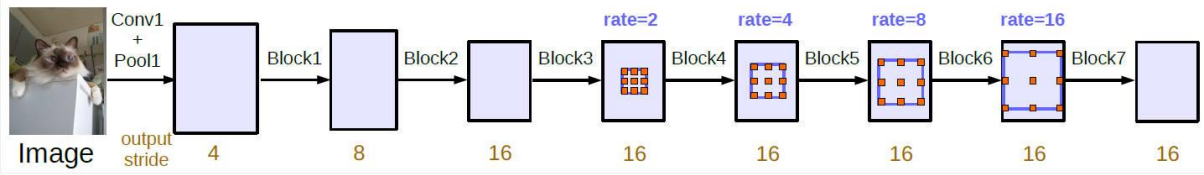


Fig 6. The Encoder-Decoder architecture of DeepLabv3 [14].



(a) Going deeper without atrous convolution.



(b) Going deeper with atrous convolution. Atrous convolution with $rate > 1$ is applied after block3 when $output\_stride = 16$.

Fig 7. The multi-grid method for ASPP in DeepLabv3 [14].

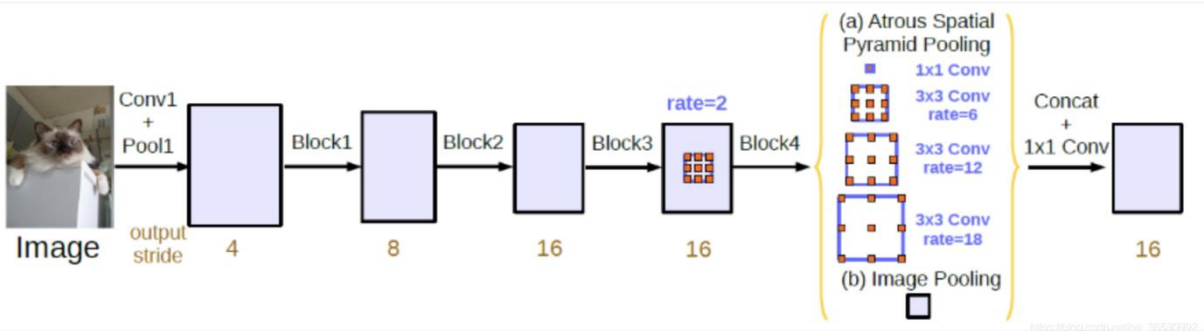


Fig 8. The ASPP architecture with parallel dilated convolutions in DeepLabv3 [14].

4. Problem

4.1 First problem

The first problem is about the choice of approach. Different models perform differently on different object classes and the U-Net, PSPNet, DeepLab are experimented based on PASCAL VOC datasets, medical datasets and Cityspaces datasets. However, our project focus on the segmentation for herbarium sheets, which include only 4 classes: (1) Barcode, (2) Colour chart, (3) Labels, (4) Scale. Therefore, the evaluation in origin papers, such as $mIOU, mPA, mPrecision$, are not convincing for our dataset. In the start of this project, we need to apply these models into our dataset and compare their output results, and choose the most stable, portable, flexible, resilient, fast in speed and lightweight model for our project. Besides, we would like to apply the methodologies in transfer learning and cross-dataset validation to the semantic segmentation model [12].

4.2 Second problem

The second problem is about the dataset for training. We have received labelled herbarium sheets data from three institution: (1) Mixseths (Mixed), (2) Mnhnhs (MNHM), (3) nmwhs (NMW). Mixed has 1000 labelled images, MNHM and NMW have 500 labelled images separately. Besides, we are given from Wyoming University with 519 unlabelled images for prediction. However, as it mentioned in previous sections, the consistency of content inside the collections varies between different institutions and collections. And there are also some mistakes in the labelled dataset, which will cause the incorrect of the training result.

4.2.1 As it shown in Fig 9, in some of the herbarium sheets images, Labels objects may be placed on both sides of the Specimen objects, some Labels objects may contain either handwritten or printed text, some even do not have text in them, and the Labels objects also vary in size and shape. This requires the classification based on pixel has ability to identify the unclear border of the adjacent objects.

4.2.2 Some Barcode objects are simple and only include the Specimen identifier, while other are printed inside Labels that also include the name of the institution, as it shown in the bottom of Fig 10.

4.2.3 Some Colour chart objects may consist inside the Scale objects, and in some herbarium sheets there are more than one Colour charts. As it shown in Fig 10, there are two Colour chart objects inside the Scale object in the bottom of the image, however, the labelled data we received from NHM does not clearly label the

difference between them. This requires to re-label the origin dataset or add more correct dataset into the current one. And the colour palettes inside Colour chart objects are different between institution and collections. For example, as it shown in Fig 11, some Colour chart objects only contains light colour palettes, others may contain dark colour palettes, which is the colours between RGB(255, 255, 255) (white) and RGB(0, 0, 0) (black). And the dark colour palettes have the similar colour range as the Scale, which may cause the difficulty for the models to identify the classes of them.

4.2.4 All the four classes objects may cover each other, which make it difficult to identify the border based on colour of each pixel because they may share the same border. And some of the Labels have pare white background, however, some may turn into yellow due to age, which is close to the colour of the background. We will need to ensure our model could identify the difference between objects and background.

4.2.5 Most state-of-the-art semantic segmentation models relies on the high-performance GPU, although the prediction is fast in speed, the training process cost a high volume of time. However, we only have a normal-performance GPU NVIDIA GTX 1060-Max-Q GPU on a laptop. In addition, the time limitation of this project is short, which only has 24 days before submission. We need to solve the problem of these, ensure our model can perform not only training but also predicting process on normal-performance GPUs, ensure our model can train or predict in as little time as possible. In other words, we need to ensure the strong portability, applicability, high speed and light weight of our model without compromising the quality of the output.

We are going to train models with each institution's dataset separately, then combined them together into dataset combined 1000 images. After this, we will analysis the results of these experiments, try to figure out the advantage and disadvantage of each institution's dataset. In addition, we will give a result which kind of dataset is more suitable for performing segmentation on Wyoming University's data. Moreover, we will add some Wyoming data into the combined dataset and see if it will help to improve the output results.

4.3 Third problem

The third problem is about the hyperparameters on the model we chosen. We are going to implement several experiments with different hyperparameters, for example, with different

optimizer and different learning rate, we will figure out which hyperparameter will fit the model focusing on segmentation for herbarium sheets dataset. For example, for the backbone, many studies use ResNet-50, ResNet-101, VGG-16, $xception$, etc. However, as we discussed before in the Background material, the MobileNet could reduce the computation of GPU, avoid the problem of OOM (out of memory) in non-HPC devices and also increase the training speed. Although the MobileNet focus on the training of Cityspaces datasets, we will try to apply the lightweight backbone MobileNet, find out if it could fit the training process of herbarium sheets datasets provided by Wyoming University.

4.4 Fourth problem

In addition to the Cross-Entropy loss that the original DeepLabv3+ model used, there are also Dice Loss and Focal Loss [24, 25, 26]. Dice Loss is equivalent to investigating from a global perspective. BCE zooms in pixel by pixel from a microscopic perspective, with complementary angles [24]. When there is a situation where the foreground and background are extremely unbalanced, for example, the Specimen object are larger than the other objects, CE loss cannot solve the extreme imbalance, however, Dice Loss will not be affected by foreground size. When the split content is unbalanced. For example, a 512*512 image has a 10*10 and a 200*200 segmentation sample. Dice Loss will tend to learn large blocks at this time and ignore small samples. But CE will still learn from small samples. Also, CE can play a guiding role in Dice. We are going to figure out if the Dice loss and Focal Loss or the combination of them could contribute to improve our model on herbarium sheets dataset.

4.5 Fifth problem

Moreover, the cost of memory and time cost of training model is the problem. We would like to make our project lightweight and fast in speed. This means our model require less memory, enabling the training and deployment of larger neural networks. However, to accelerate the math operations run on GPU will cause the reduce in precision, it is hard to find a balance between them [34]. We will need to focus on experiment and find the acceptable and stable mode of our model.

By solving all these problems mentioned above, we will find a proper method to help Wyoming University to improve their digitisation workflows. And more problems that found during the development and implementation related to previous studies and practical solutions will be discussed in the following sections.

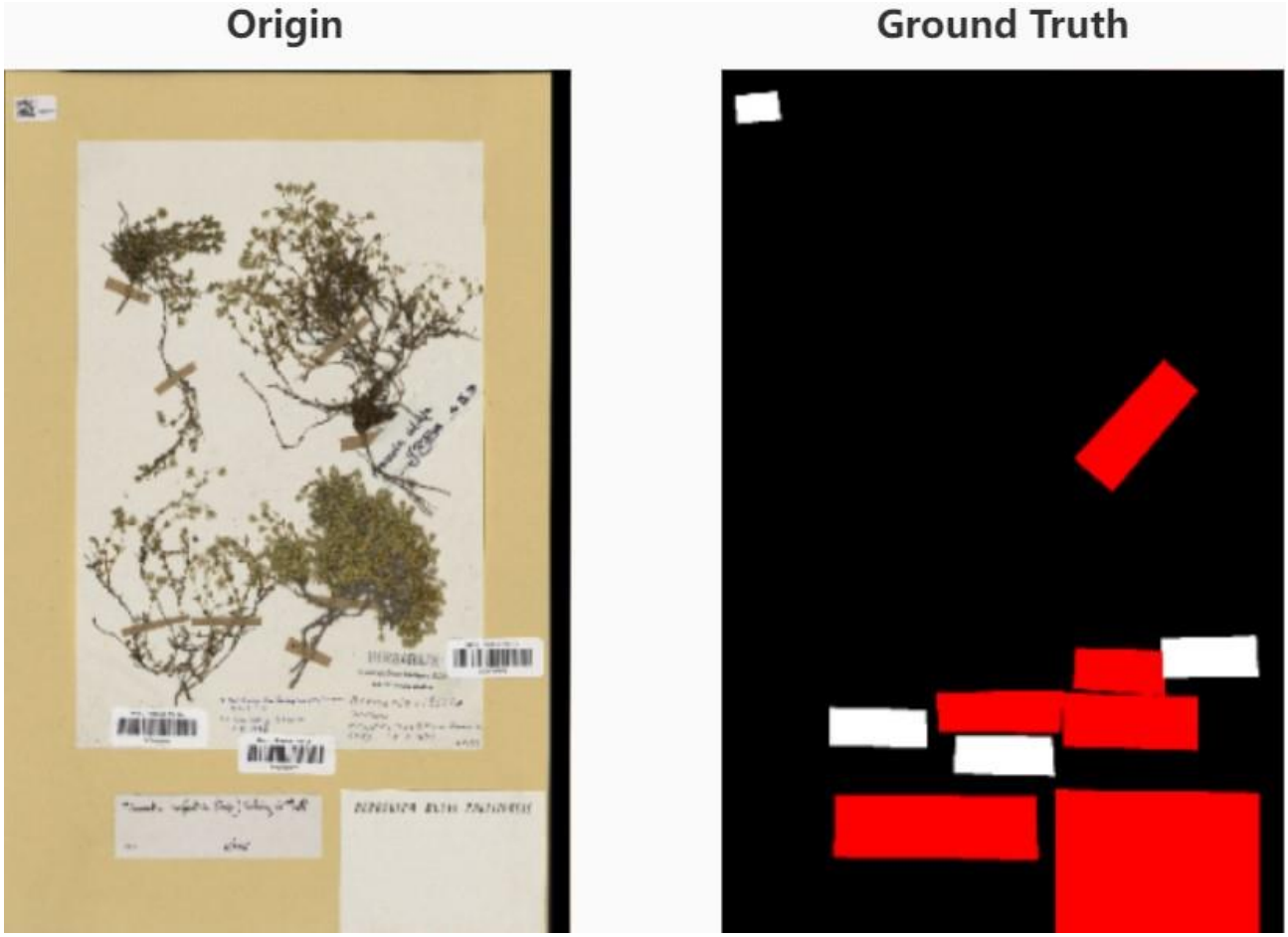


Fig 9. Label objects contain handwritten text (in the centre label object) and blank space (in bottom right label object).

Fig 10. Mistake of NHM label (mistake assign Yellow to Colour Chart object, which should be assign as Blue colour).

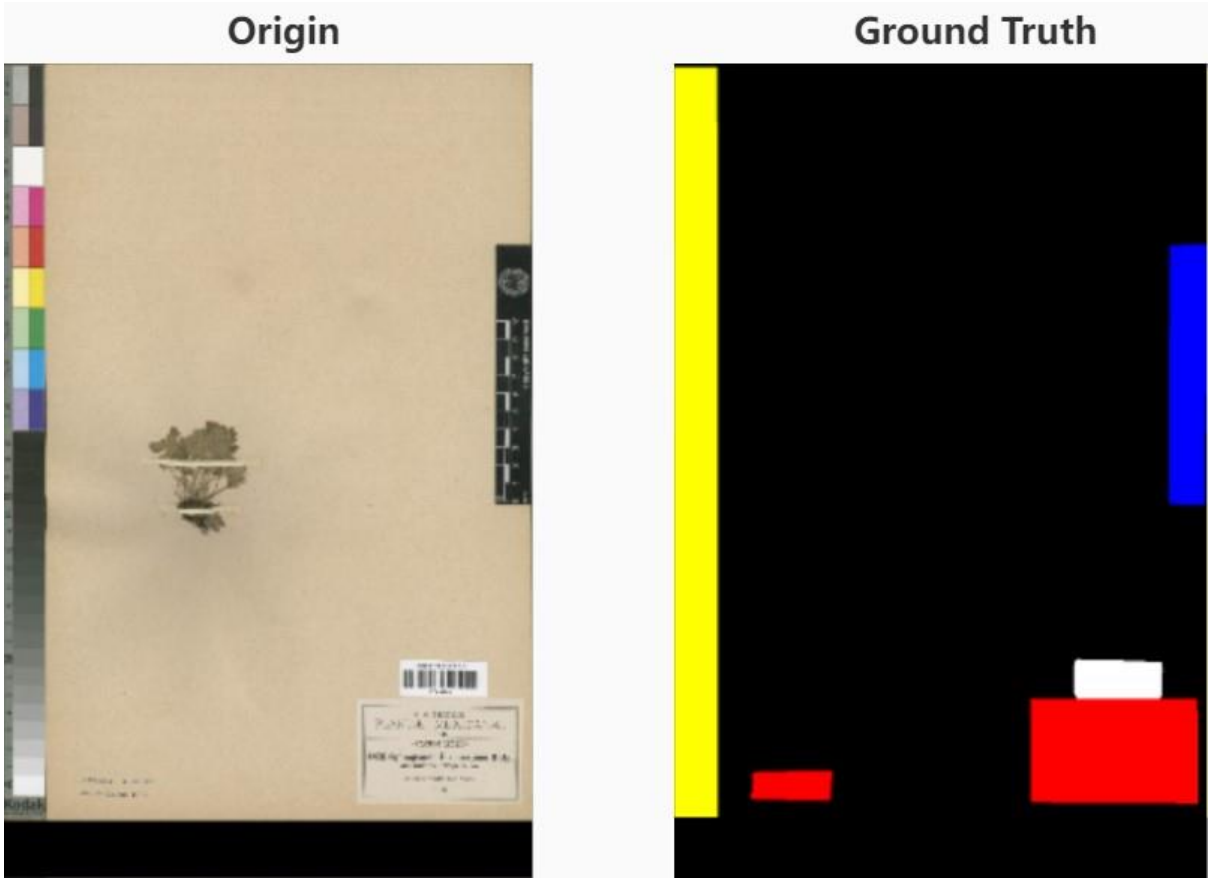


Fig 11. Multiple Colour chart with both light and dark palettes.

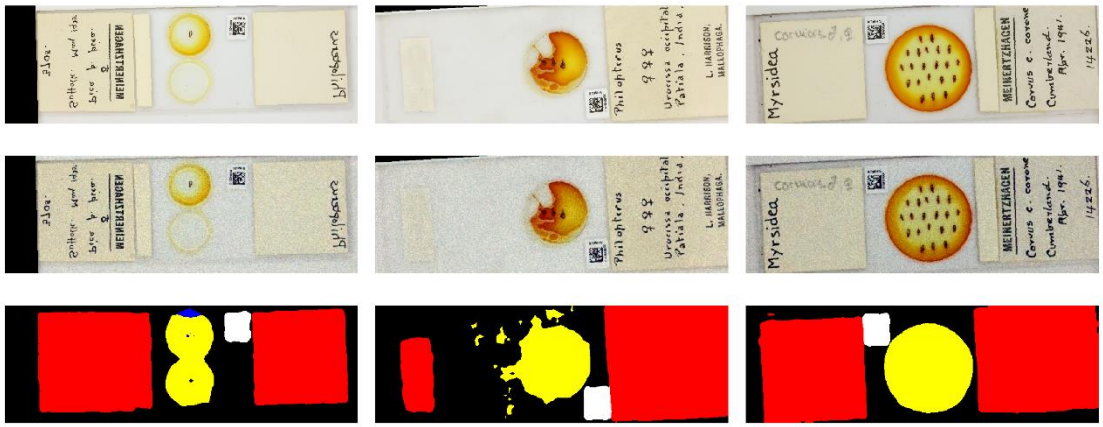


Fig 12. Example predict result of NHM model on NHM microscope slides.

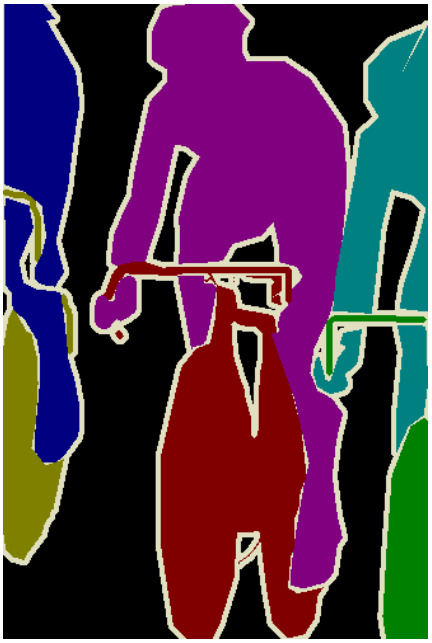


Fig 13. Example of ignore label (white outlining bound edges).

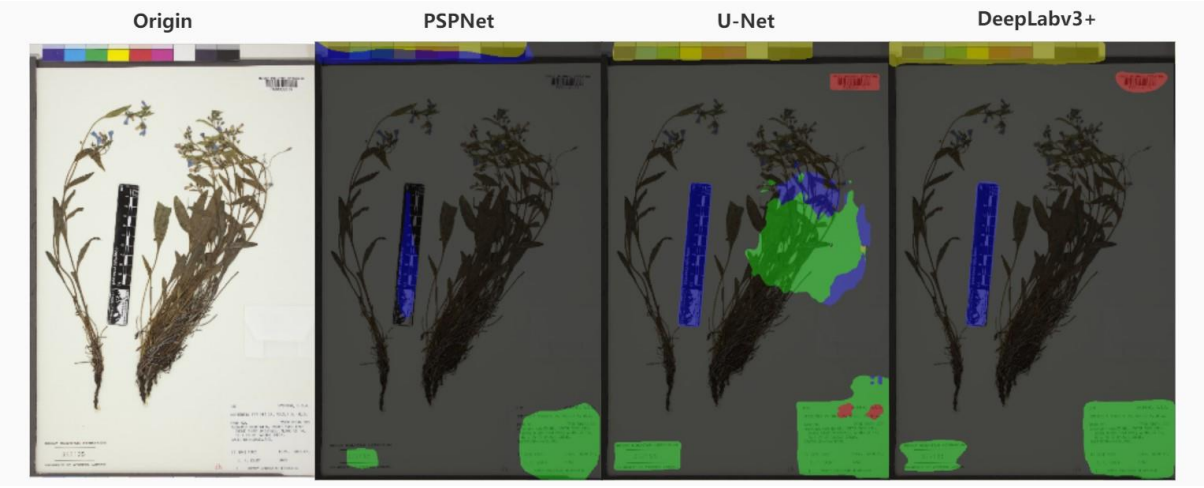


Fig 14. Example output of PSPNet, U-Net, DeepLabv3+ (Blue for Scale object, Yellow for Colour chart object, Red for Barcode object, Green for Label object).

5. Approach

5.1 Natural History Museum's semantic segmentation model

Because this project is inspired by Natural History Museum's semantic segmentation model, in the initial step, we tried to run and evaluate the feasibility of using their model into Wyoming University's data.

However, the NHM's model has problem for noise reduction processing. As it shown in Fig 12, the first row is the origin image, the second row is processed image, the third row is the output result of semantic segmentation. In this case, we could find that it cannot clearly distinguish the background and irregular Specimen objects (in second column). After reading the source code, in datasets.py, I found that it initiates the dataset segmentation with configuration $*.ini$ file. Although, it has two denoising function $g^{(1)}$ and $g^{(2)}$ in its architecture, as it shown in the Fig 1, in the configuration file it does not include the white outlining as it shown in Fig 13, which indicate the ignore label that would not be penalized during the segmentation and help the model identify the difference between the boundary of objects and the background and hence improve the overall performance.

Moreover, the NHM's model has problem for generating processed dataset information file. In this model, it uses "h5py" to process labelled data and unlabelled data into "training.hdf5" file and "test.hdf5" file, these files are extremely large and will be larger when the original image size increases . For example, it will generate 3.19GB processed data based on 105MB microscope slides data and it will generate 30GB processed data based on 421MB herbarium sheets data. This is due to the chunk layout that dump lots of extraneous information for each dataset into the HDF5 file. Intuitively, we could use Pillow (PIL) instead, which is a python imaging library that provides an efficient internal representation and powerful image process capabilities for storing data into pixel formats.

Nevertheless, the generated trained model of NHM's model is not either $*.pt$ nor $*.pth$ file, it is even not $*.onnx$ file. In other words, the output result of its model does not have portability into other models.

Lastly, as it shown in Table 1, the DeepLabv3+ baseline model has significant higher accuracy and precision on all of the three institutions' datasets than the NHM segmentation baseline model. Therefore, the NHM's segmentation model is discarded.

Trained Model	Prediction Model	Element	Accuracy	Precision
NHM Model	NMW Model	Label	0.587	0.772
		Scale	0.270	0.235
		Colour Chart	0.146	0.146
		Barcode	0.413	0.426
		Total	0.375	0.404
	MNHN Model	Label	0.566	0.903
		Scale	0.671	0.549
		Colour Chart	0.312	0.297
		Barcode	0.773	0.787
		Total	0.580	0.674
	Mixed Model	Label	0.548	0.749
		Scale	0.137	0.137
		Colour Chart	0.230	0.218
		Barcode	0.613	0.627
		Total	0.383	0.420
DeepLabv3Plus	NMW Model	Label	0.85	0.90
		Scale	0.96	0.94
		Colour Chart	0.71	0.82
		Barcode	0.98	0.96
		Total	0.86	0.87
	MNHN Model	Label	0.95	0.97
		Scale	0.98	0.95
		Colour Chart	0.69	0.85
		Barcode	0.89	0.92
		Total	0.86	0.87
	Mixed Model	Label	0.77	0.81
		Scale	0.94	0.95
		Colour Chart	0.94	0.93
		Barcode	0.81	0.81
		Total	0.82	0.80

Table 1. Comparison between DeepLabv3+ and NHM segmentation model.

5.2 U-Net

As it mentioned before in Background material section, the U-Net uses the Encoder-Decoder architecture that consists of contracting path and expansive path [20]. As it shown in Fig 2, the contracting path is a typical convolutional network architecture that contains a repeating structure of two 3×3 convolutions without padding (blue arrow), one max-pooling layer with $stride = 2$ in each repetition (red arrow), and a $ReLU$ active function. After each downsampling, it doubles the number of feature channels. And in expansive path, it applies up-convolution in every step (green arrow), half the size of feature channels and double the feature map. After the up-convolutions, it contacts with the contracting path to merge feature maps (white and blue block) and applies two 3×3 convolutions with one 1×1 convolution to transfer the 64 feature channels into the program specified number of classes and output the segmentation map.

Although the U-Net architecture applies more convolutional layers and is wide-spread semantic segmentation tool in medical field, the cascading multiple networks propagates errors from one stage to another reducing overall system accuracy and will occupy more GPU memory [30]. And it does not use the ASPP module to efficiently enlarge the field of view, in other word, the U-Net may cost more time for capturing the details and may not have good performance on multi-scale objects, for example, the multi shape Labels objects in herbarium sheets collections. As it shown in Table 2, it cost more time than DeepLabv3+ and PSPNet, and has lower $mIOU, mPA, mPrecision,\ mRecall$ values than the DeepLabv3+. In addition, as it shown in Fig 14, although it can assign the class label to objects with more rectangular like shape, it has problem in label with mistakes in large region. And in 100 epochs, as it shown in Fig 15, the loss curve cannot convergence to a small error level, which means that it may need more epochs to achieve a stable output result and will spend more time on training the model than the others. Hence, the U-Net is discarded for our project.

5.3 PSPNet

As it mentioned in Background material section before, the PSPNet introduce the Pyramid Pooling Module, which could act as an effective hierarchical global contextual prior [21]. And it proved that the global average pooling as the global contextual prior, will have better performance than max-pooling in the Pyramid Pooling Module to perform the image classification tasks [31]. The global average pooling is designed to replace the traditional fully connected layers in convolutional networks, it could help to interpret the feature maps with categories confidence maps more easily and avoid the overfitting of the result.

However, as it shown in Table 2, the PSPNet has the worst performance in $mIOU$ out. And it has the same problem as the U-Net that they need more epochs to converge the loss curve as it shown in Fig 15, and hence cost more time and memory. And as it shown in Fig 14, the PSPNet baseline cannot even identify the objects inside herbarium sheets image and assign correct class labels. Hence, the PSPNet is discarded for our project.

5.4 DeepLabv3+ and Conclusion for approach choice

DeepLabv3+ is a state-of-the-art deep learning model for semantic segmentation, it introduces the Encoder-Decoder architecture as it shown in Fig 6, the Encoder has the DCNN module and the ASPP module with parallel multi-grid method as it shown in Fig 7 and Fig 8, and the Decoder contains feature merge module and will output the final result of DeepLabv3+. In other words, the Encoder is used to feature extraction, and the Decoder is to combine the extracted features with low-level features from DCNN and the high-level features from output of the Encoder into the final result of the classes label for each pixel.

When an image input into the DeepLabv3+ model, it will firstly input into the Encoder and use a deep convolutional neural network (DCNN) to obtain two effective feature maps, one is the Low-Level effective feature map to be shifted to the Decoder part, another is the High-Level effective feature map to be further processed in the Encoder part. The Low-Level feature will have a larger width and height, the High-Level feature will do more downsample with a smaller width and height compared to the Low-Level one.

In the Encoder, the High-level feature map will be processed with ASPP module for further feature extraction. In the ASPP module, there are three $3x3$ convolution branches, with dilation rate 6, 12 and 18. These three convolutions with different dilation rate could increase the view of filed without loss of information, and hence includes a larger range of information for the output of each convolution. After the atrous convolutions, the result of three 3x3 convolution branches will be stacked with the $1x1$ convolution branch and be combined with the image pooling. And the stacked result will be input into the $1x1$ convolution branch again to adjust the number of channels and get the feature map with higher semantic information and sent as the input of the Decoder.

In the Decoder, it will upsample the high-level feature map with high semantic information and do $1x1$ convolution for the Low-Level feature from DCNN with low semantic information. Then the two feature maps will be merged by using two $3x3$ convolutions with batch normalization and activation function. After this, the Decoder will use a $1x1$

convolution to adjust the channels of the merged feature maps, and upsample the adjusted feature to resize into the same width and height as the original input image.

According to the original paper, DeepLabv3+ has higher $mIOU$ over the other models on Cityspaces dataset, in other words, it has a better performance on complex scene [14]. Even though U-Net is widely applied to segmentation of medical images [30], this dissertation report is focus on the herbarium sheets images and unlike the microscope slides, which contains small objects inside the images, herbarium sheets images are more similar to the Cityspaces dataset with herbarium specimens as the large object and other informational object, such as Barcode, Label, Colour chart and Scale objects.

Despite the existence of the standard guidelines from GPI [32, 33], the distribution of these objects varies among images, even in same institution's dataset as it discussed before in previous sections. This situation could be viewed as multi-scale objects inside the images with imbalanced consistency rate, and the DeepLabv3+ has upgrade the algorithms and methodologies after the iterations of the release version, which provide Encoder-Decoder architecture with DCNN and ASPP as it shown in Fig 6. And this architecture is considered to be the best choice of approach for our project, as it shown in Table 1 and Table 2, the DeepLabv3+ has the outstanding output values among the candidates with 90.65% $mIOU$. The loss convergence of DeepLabv3+ could be finished in 100 epochs as it shown in Fig 15, which means the model could be trained into stable mode in 100 epochs with less time spent than others, and the $mIOU$ curve of the DeepLabv3+ is smoother than the others.

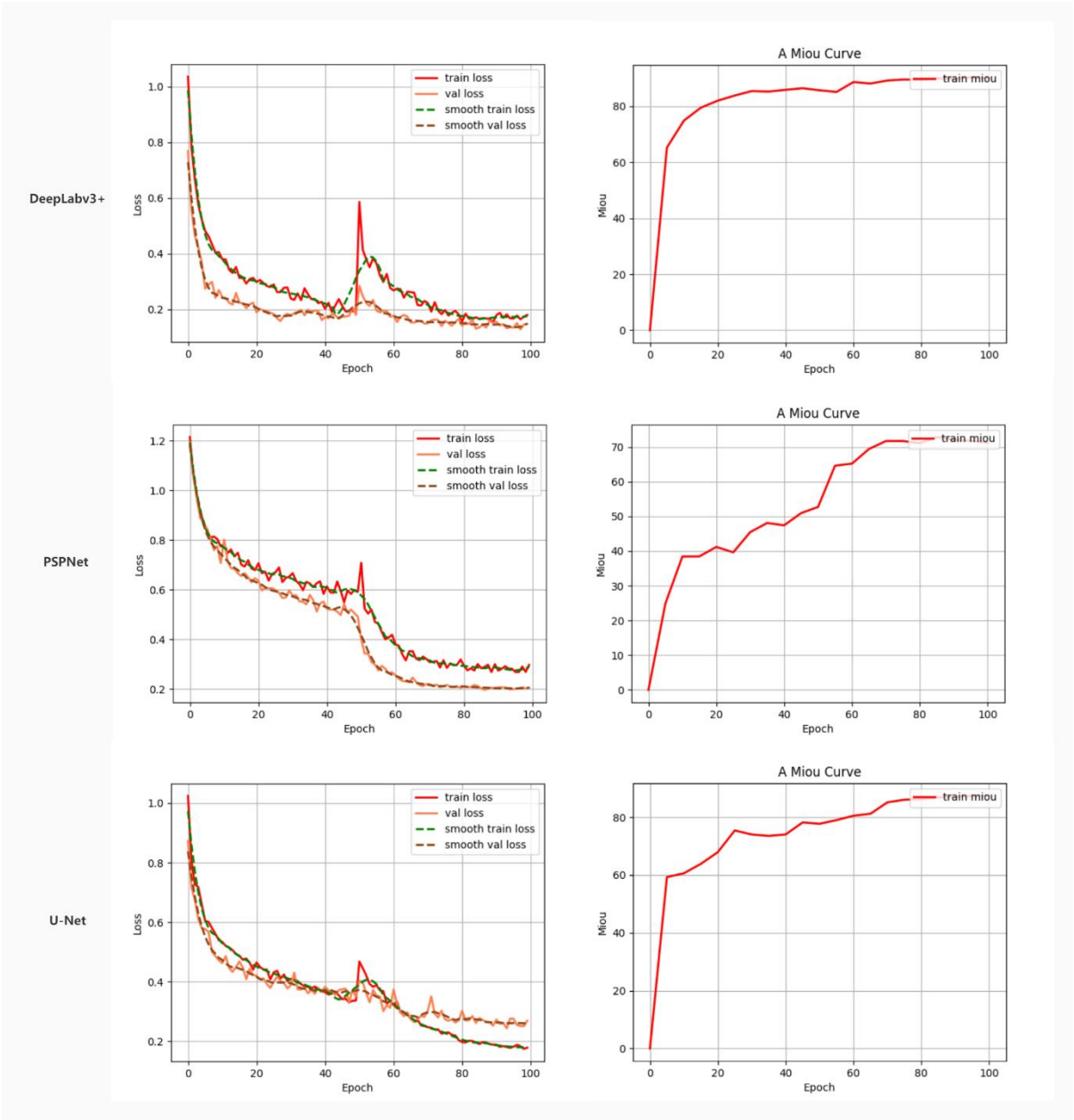


Fig 15. Loss curve and $mIOU$ curve of U-Net, PSPNet, DeepLabv3+.

#	Element	$mIOU$	mPA	$mPrecision$	$mRecall$	Time
U-Net	Scale	0.85	0.92	0.92	0.92	
	Colour chart	0.80	0.87	0.91	0.87	
	Label	0.87	0.95	0.91	0.95	
	Barcode	0.86	0.92	0.93	0.92	
	Total	0.8747	0.9321	0.9325	0.9321	383min
PSPNet	Scale	0.79	0.87	0.90	0.87	
	Colour chart	0.73	0.80	0.88	0.80	
	Label	0.75	0.82	0.90	0.82	
	Barcode	0.38	0.49	0.64	0.49	
	Total	0.7178	0.7939	0.8544	0.7939	259min
DeepLabv3+	Scale	0.92	0.95	0.96	0.95	
	Colour chart	0.88	0.96	0.91	0.96	
	Label	0.90	0.96	0.94	0.96	
	Barcode	0.85	0.94	0.90	0.94	
	Total	0.9065	0.9580	0.9433	0.9580	252min

Table 2. Results of U-Net, PSPNet, DeepLabv3+ (fp16 = True, backbone = mobilenetv2, optimizer = adam, epoch = 100, freeze epoch = 50, freeze batch size = 8, unfreeze batch size = 4, learning rate decay type = cos, dice loss = True, focal loss = True, num workers = 1).

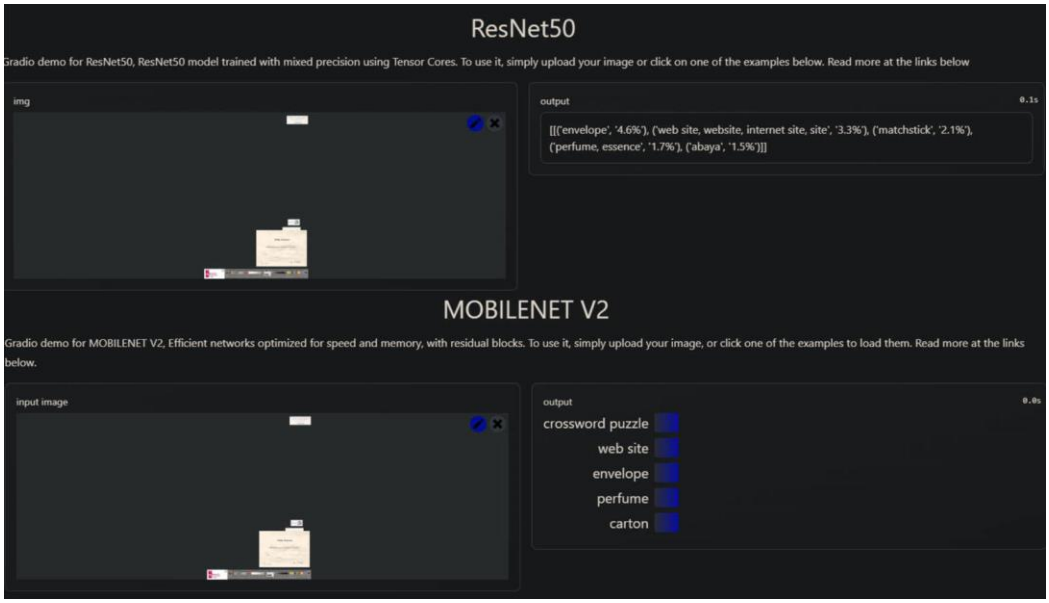


Fig 16. Comparison of RestNet-50 and MobileNetv2.

6. Product and Analysis

This section would give the detail of what we did. It will include a discussion of our research activities related to what we found out according to the experiments.

6.1 Requirements

In this project, we aim to:

(1) Reproduce the code of the DeepLabv3+, transfer it from Tensorflow2 into PyTorch framework.

(2) Though the original model only uses Cross-Entropy loss (CE loss), we will add Dice loss (DL) and Focal loss (FL) function options in our project. The user could train with CE only, FL only, CE + DL, FL + DL mode.

(3) The original model uses the Adam optimizer, we will add SGD optimizer option.

(4) The original model offers step learning decay, we will add cosine learning decay (cos) option.

(5) Ensure the easy use of the project, for example, simplify the generation of train, test, validation configuration. In other words, provide automatic module to get annotations.

(6) Provide the automatic module to justify the labelled dataset, in other words, auto convert *.txt, *.xml, *.json files into desired format of labelled dataset. For example, convert into TXT format for YOLOv5 and DeepLabv3+. In addition, provide automatic transfer from 24-bit depth RGB image into 8-bit depth RGB image in PNG format.

(7) Ensure the lightweight of the project, in other words, even with low-level GPU the training process could be run. In addition, the lightweight model will provide greater portability, for example, can be rebuild with Flask and outputting the training result as an API into either frontend or backend. And the time cost of the training process should be as less as possible.

(8) Provide README file in both English and Simplified Chinese for users to have quick acknowledge and quick start to use this model. In addition, some potential bugs and possible solutions will be provided in README.

(9) Provide necessary comments in the code, help users or other potential contributors understand the code. Also, provide “requirements.txt” file for users to have quick environments installation.

```
|   .gitignore
|   convert_labelled.py
|   deeplab.py
|   get_annotation.py
|   get_miou.py
|   labelme_to_deeplab.py
|   predict.py
|   README.CN.md
|   README.md
|   requirements.txt
|   train.py
|   tree.txt
|
├───datasets
|   ├───before
|   |       RM0008192.jpg
|   |       RM0008192.json
|   |
|   ├───JPEGImages
|   |       RM0008192.jpg
|   |
|   └───SegmentationClass
|           RM0008192.png
|
├───herbarium_sheets
|   └───wyoming_combine_origin
|       ├───ImageSets
|       |   └───Segmentation
|       |           test.txt
|       |           train.txt
|       |           trainval.txt
|       |           val.txt
|       |
|       ├───JPEGImages
|       |       AMD.100007.JPG
|       |
|       └───SegmentationClass
|               AMD.100007.png
|
├───img
|       RM0008156.jpg
|
├───img_out
|       RM0008156.jpg
|
├───logs
|   |   best_epoch_weights.pth
|   |   last_epoch_weights.pth
|   |
|   └───loss_2022_09_15_09_05_28
|           epoch_loss.png
|           epoch_loss.txt
|           epoch_miou.png
|           epoch_miou.txt
|           epoch_val_loss.txt
|
├───model_data
|       deeplab_mobilenetv2.pth
|
├───nets
|   |   deeplabv3_plus.py
|   |   deeplabv3_training.py
|   |   mobilenetv2.py
|   |   xception.py
|   |   __init__.py
├───utils
|   |   callbacks.py
|   |   dataloader.py
|   |   utils.py
|   |   utils_fit.py
|   |   utils_metrics.py
|   |   __init__.py
```

Tree 1. Example DOS-tree architecture of the project.

6.2 Design

The reproduced DeepLabv3-Plus-Pytorch is designed on Python 3.9 with Anaconda3 environment and Pytorch framework. The code is designed to use CUDA v11.7 with the relevant version of CUDNN.

The architecture of our project is designed as it shown in the Tree 1.

The "datasets" directory is designed to process raw train data into DeepLabv3+ train data for users who want to train their own dataset. For example, in the "./datasets/before" is used to store the 24-bit depth RGB *.jpg raw images and the label information $*.json$ file. The "./datasets/JPEGImages" is used to store the 24-bit depth RGB $*.jpg$ raw images, and the "./datasets/SegmentationClass" is used to store the processed 8-bit depth RGB *.jpg labelled (ground truth) images.

The "$herbarium\_sheets$" directory is designed to be the root directory of our training dataset, its hierarchy is same as the "datasets" directory. The "./ImageSets/Segmentation" directory is designed to store the configuration files of the train-test-validation configuration of the training process, which is output of the "get_annotation.py".

The "img" directory is designed to store the 24-bit RGB images for prediction.

The "img_out" directory is designed to store the output of the prediction.

The "logs" directory is designed to store the logs of the training process, it is designed to store the "$*.pth$" model every 5 epoch, and the best model out of the training process with the lowest loss. The "./logs/loss_2022_09_15_09_05_28" is designed to be the automatic generated directory which includes the loss curve and $mIOU$ curve $*.png$ files and related values in *.txt files.

The "$model\_data$" directory is designed to be the root of the pre-trained model file.

The "nets" package is designed to store the model's network code. The "./nets/deeplabv3_plus.py" is designed to contain the reproduced DeepLabv3+'s Encoder-Decoder implementation, including the DCNN and ASPP module. In our project it used MobileNetv2, hence it will have the code about the output function of the MobileNetv2. The "./nets/deeplabv3_training.py" is designed to store the code of the implementation of the CE loss, FL loss, DL loss, learning rate decay type, weight decay function, optimizer setting function. The "./nets/mobilenetv2.py" is designed to store the implementation of the MobileNetv2.

The "utils" package is designed to contain the pre-processing and output utils that frequently used in our reproduced DeepLabv3-Plus-Pytorch project. The "./utils/callbacks.py" is designed to implement the processing of the logs using $tensorboard$ and matplotlib

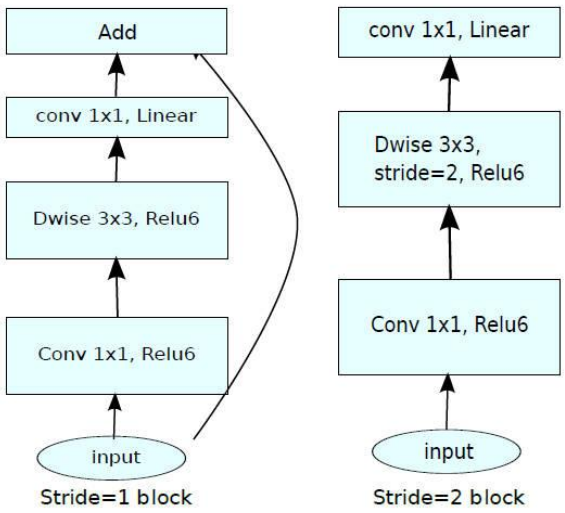
libraries. And the output of the logs will be stored in the "logs" directory mentioned before. The "./utils/dataloader.py" is a particularly important part of our reproduce process, it is designed to implement the pre-process of the input image, including get data from root, ignore label process, one-hot encoding for inputs, data arguments. The data arguments we applied to our project will include random resize, random flip, random blur, random rotate, random colour gamut transformation, add grey bars to the extra part of the images when resize the images, collate the classes label information and collate into numpy array. The "./utils/utils.py" is designed to implement the functions that frequently used in our project, such as image resize function, convert colour function, get current learning rate function, get pre-trained weight model function. The "./utils/utils_fit.py" is designed to implement the output per epoch forward to the next epoch. The "./utils/utils_metrics.py" is designed to implement the calculation of the values including the loss, $mIOU$, f score per epoch.
The "$.gitignore$" is designed to configure what file we do not want to apply version control and push commit to Git.
The "convert_labelled.py" is designed to process the NHM dataset into the dataset we need, including change the file name and convert 24-bit depth RGB image file into 8-bit depth RGB image file.
The "deeplab.py" is designed to be the override of functions and parameters that would be called in the "deeplabv3_plus.py". It is designed to receive the user's configuration, pre-process and post-process the images data before and after train and predict.
The "get_annotation.py" is designed to be the automatic tool to separate train-test-val sets according to the user's setting. It will automatically output the configuration *.txt files into "dataset_root/ImageSets/Segmentation" directory.
The "get_miou.py" is designed to get the values of the trained model and output into the "miou_out" directory.
The "labelme_to_deeplab.py" is designed to be the automatic tool to convert the raw 24-bit depth RGB *.jpg images and the $Labelme\ *.json$ file into the 8-bit depth RGB $*.png$ labelled images (ground truth).
The "predict.py" is designed to be the entrance of the prediction. It will call the "deeplab.py" and "callback.py" and provide different modes of prediction: single image prediction, image directory prediction, video prediction, fps prediction and export $*.onnx$ files. Its output will be automatically stored in the "img_out" directory as it mentioned before.

The “README.CN.md” is the Chinese Simplified README file, and the “README.md” is the English version README file.

The “requirements.txt” is designed to be the essential requirements of the project, which would help the user for quick and easy installation.

The “train.py” is designed to be the main entrance of the training process. It will provide the user an easy way to adjust the hyperparameters for training.

The “tree.txt” is designed to store the tree architecture of the project, in Windows environment, user could simply use *cmd* and run “tree /F” or “tree /f > tree.txt” and use Microsoft word with DOS-Encoding to read the file.



(d) Mobilenet V2

Fig 17. The Mobilenetv2 architecture [22].

6.3 Implementation

6.3.1 Implementation of Encoder

6.3.1.1 Implementation of DCNN inside Encoder

The first part of the implementation is to build the Deep Convolutional Neural Networks (DCNN) backbone feature extraction network, the original paper is using *xception* as backbone, however, due to the limitation of the GPU, this project proposes the MobileNetv2 as the backbone. There are also other lightweight backbone candidates, such as ResNet-50, VGG-Net. Given that there is not enough time to test every type of backbone and compare the performance on the given dataset, the online comparison tool provided by PyTorch official website could be used. After comparing the weights distribution between ResNet-50 and MobileNetv2, as it shown in Fig 16, the MobileNetv2 has a more balanced weights for

different classes and intuitively the MobileNetv2 is chosen. The MobileNet is a lightweight deep neural network model designed by Google for embedded devices such as mobile phones, and MobileNetv2 is the upgraded MobileNet. It consists of inverted residual blocks, as it shown in Fig 17, the backbone part firstly uses 1×1 convolution to increase dimension, then use 3x3 depth-wise separable convolution for feature extraction and finally use 1x1 convolution to reduce dimension, and the residual edge part directly connects the input and output [22].

As it shown in Fig 18, in "deeplabv3_plus.py", it gets the pre-setting inverted residual blocks in "mobilenetv2.py" by index, then start downsampling and output two features, one for the Encoder and another for the Decoder as it discussed in the previous section. Traditionally, the MobileNetv2 will do three times downsample and the $downsample\_factor$ is 8, however, the less the $downsample\_factor$ the more GPU memory will be required. To overcome the OOM problem, in this project, it provides additional options for 8GB GPU memory with "$downsample\_factor = 16$", which means it could do four times downsample in the DCNN.

Fig 18. Implementation of MobileNetv2 backbone.

6.3.1.2 Implementation of ASPP inside Encoder

After getting the two feature maps from DCNN, the Atrous Spatial Pyramid Pooling (ASPP) is required to be implemented to process the High-Level feature map get from DCNN in the Encoder architecture. The DeepLabv3+ revisit the [15] and applied batch normalization within ASPP [14]. As it shown in Fig 19, there are five branches, and every branch includes 2D convolution over an input signal composed of several input planes, batch normalization and the rectified linear unit function element-wise ($ReLU$) [16]. As it mentioned in [14], the sampling rate will affect the number of valid filter weights, hence, the default rate of the ASPP is designed to be 1, and the dilation rate in branch2, branch3 and branch4 is designed to be 6, 12, 18, which are the same value as it mentioned in original paper. In the branch5, it feeds the resulting image-level features to a 1×1 convolution with 256 filters, and then bilinearly upsample the feature to the desired spatial dimension. As it shown in Fig 20, the result of these five branches will then concatenated and stacked with the same compressed size (global average pooling) and pass through another 1x1 convolution and generates the final result of the Encoder part and forward the result with high semantic information into the Decoder part.

```
143         self.cat_conv = nn.Sequential(
144             nn.Conv2d(48 + 256, 256, 3, stride=1, padding=1),
145             nn.BatchNorm2d(256),
146             nn.ReLU(inplace=True),
147             nn.Dropout(0.5),
148 
149             nn.Conv2d(256, 256, 3, stride=1, padding=1),
150             nn.BatchNorm2d(256),
151             nn.ReLU(inplace=True),
152 
153             nn.Dropout(0.1),
154         )
155         self.cls_conv = nn.Conv2d(256, num_classes, 1, stride=1)
156 
157     def forward(self, x):
158         H, W = x.size(2), x.size(3)
159         # get two feature layers
160         # low_level_features: do convolution
161         # x : backbone(Encoder) enhanced feature extraction using ASPP structure
162         low_level_features, x = self.backbone(x)
163         x = self.aspp(x)
164         low_level_features = self.shortcut_conv(low_level_features)
165 
166         # enhance feature edge upsampling
167         # feature extraction using convolution after stacking with low-level features
168         x = F.interpolate(x, size=(low_level_features.size(2), low_level_features.size(3)), mode='bilinear',
169                           align_corners=True)
170         x = self.cat_conv(torch.cat((x, low_level_features), dim=1))
171         x = self.cls_conv(x)
172         x = F.interpolate(x, size=(H, W), mode='bilinear', align_corners=True)
173         return x
```

3x3 convolution

3x3 convolution

Get low-level and high-level features from Encoder

1x1 convolution low-level feature

Upsample high-level feature

Merge two feature layers

Get the final result with same size as input

Fig 21. Implementation of the Decoder of DeepLabv3+.

```python
class ASPP(nn.Module):
    def __init__(self, dim_in, dim_out, rate=1, bn_mom=0.1):
        super(ASPP, self).__init__()
        self.branch1 = nn.Sequential(
            nn.Conv2d(dim_in, dim_out, 1, 1, padding=0, dilation=rate, bias=True),
            nn.BatchNorm2d(dim_out, momentum=bn_mom),
            nn.ReLU(inplace=True),
        )
        self.branch2 = nn.Sequential(
            nn.Conv2d(dim_in, dim_out, 3, 1, padding=6 * rate, dilation=6 * rate, bias=True),
            nn.BatchNorm2d(dim_out, momentum=bn_mom),
            nn.ReLU(inplace=True),
        )
        self.branch3 = nn.Sequential(
            nn.Conv2d(dim_in, dim_out, 3, 1, padding=12 * rate, dilation=12 * rate, bias=True),
            nn.BatchNorm2d(dim_out, momentum=bn_mom),
            nn.ReLU(inplace=True),
        )
        self.branch4 = nn.Sequential(
            nn.Conv2d(dim_in, dim_out, 3, 1, padding=18 * rate, dilation=18 * rate, bias=True),
            nn.BatchNorm2d(dim_out, momentum=bn_mom),
            nn.ReLU(inplace=True),
        )
        self.branch5_conv = nn.Conv2d(dim_in, dim_out, 1, 1, 0, bias=True)
        self.branch5_bn = nn.BatchNorm2d(dim_out, momentum=bn_mom)
        self.branch5_relu = nn.ReLU(inplace=True)

        self.conv_cat = nn.Sequential(
            nn.Conv2d(dim_out * 5, dim_out, 1, 1, padding=0, bias=True),
            nn.BatchNorm2d(dim_out, momentum=bn_mom),
            nn.ReLU(inplace=True),
        )
```

Fig 19. The implementation of the ASPP module.

```python
    def forward(self, x):
        [b, c, row, col] = x.size()
        # 一共五个分支
        conv1x1 = self.branch1(x)
        conv3x3_1 = self.branch2(x)
        conv3x3_2 = self.branch3(x)
        conv3x3_3 = self.branch4(x)
        # 5th branch，global average pooling + convolution
        global_feature = torch.mean(x, 2, True)
        global_feature = torch.mean(global_feature, 3, True)
        global_feature = self.branch5_conv(global_feature)
        global_feature = self.branch5_bn(global_feature)
        global_feature = self.branch5_relu(global_feature)
        global_feature = F.interpolate(global_feature, (row, col), None, 'bilinear', True)

        # Stack the contents of the five branches
        # Then 1x1 convolution integrates the features
        feature_cat = torch.cat([conv1x1, conv3x3_1, conv3x3_2, conv3x3_3, global_feature], dim=1)
        result = self.conv_cat(feature_cat)
        return result
```

Fig 20. Implementation of the forwarding function of the Encoder.

6.3.2 Implementation of Decoder

After getting the result of the Encoder, it needs to build the Decoder part and merge the High-Level feature result of the Encoder with the Low-Level feature map from the initial DCNN. As it shown in Fig 21, in the Decoder, it will upsample the high-level feature with high semantic information by using function "$torch.nn.functional.interpolate$", and perform 1×1 convolution for the Low-Level feature with low semantic information. Then the two feature maps will be merged by using "cat_conv" function, which has two 3×3 convolutions with batch normalization(BatchNorm2d) and activation function ReLU.

The final part the Decoder is to convolve and resize the merged feature map. As it shown in Fig 21, it will use a 1×1 convolution to adjust the channels of the merged feature map to the "$num\_classes$", which indicates the class of every pixel. Then upsampling with resize to make the final output feature map has the same width and height as the original input image.

6.3.3 Implementation of pre-processing

After the implementation of the DeepLabv3+ network architecture, we are going to implement the pre-processing before the training. As it shown in Fig 22, the pre-processing of images is implemented in the "$detect\_image$" function in "deeplab.py", which uses the functions in "utils.py". As it shown in Fig 22 (1), it firstly converts the input images into RGB images using the "$cvtColor()$" function provided by "utils.py". The reason why we want RGB images is that:

(a) Only the RGB image has the classes of each pixel that could be recognized by our model.
(b) We use RGB colours to assign the labels to the objects as it defined in the "$\_\_init\_\_()$" function in "deeplab.py".

As it shown in Fig 22 (2), secondly, we use "$deepcopy()$" function to back up the original image, because we are going to perform many actions on the image in the following steps.

As it shown in Fig 22 (3), we use the "$resize\_image()$" function implemented in "utils.py" to add grey bars into the input image. This function is implemented by using "$PIL.Image.BICUBIC$", which calculates the output pixel value using cubic interpolation on all pixels that may contribute to the output value. In other words, it is similar to use the "$letterbox\_image()$" function to ensure the input images with different size will resized into the size of the DeepLabv3+ required size without losing the aspect ratio of their width and height. For example, as it shown in the bottom left of the Fig 22 is the example of grey bar resize (letterbox image). In other words, this process ensures the image quality of our model.

As it shown in Fig 22 (4), we then perform normalization, add dimension to the "$batch\_size$" and put the input image's channels at the first dimension, this is required by the Pytorch.

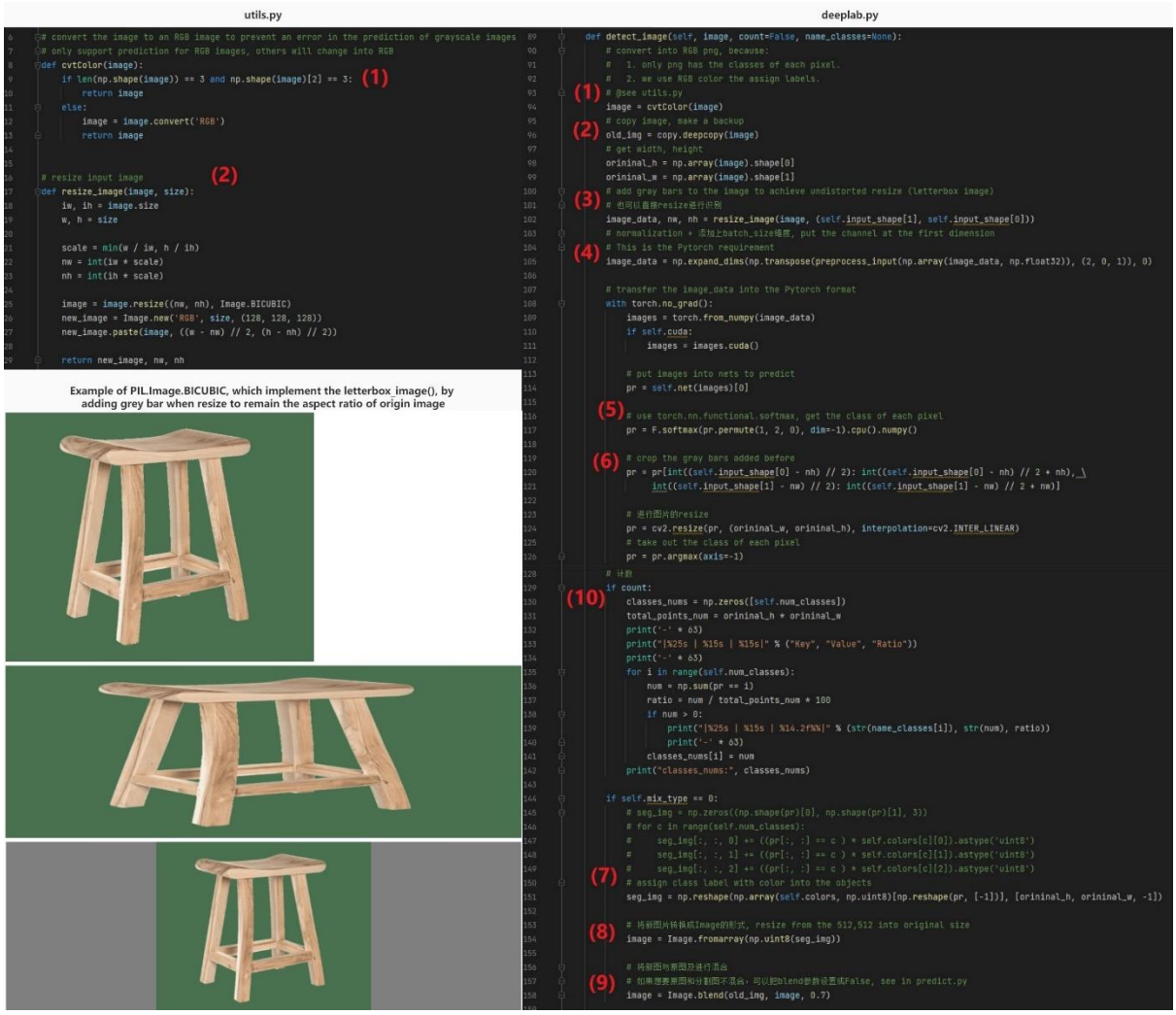


Fig 22. The pre-procsessing before training and post-processing after training.

6.3.4 Implementation of training

As it shown in Fig 23, we provide many hyperparameters for users to adjust the model according to their needs:

1. In line 38 and 43, they could choose to train with either CUDA or CPU by setting the Boolean value of the $Cuda = True$.
2. In line 43, they could set the Boolean value of $distibuted = False, sync\_bn = False$ to enable or disable the multi-GPU mode, because our training is performed on Windows, and not using the Ubuntu, we set both values to be $False$.
3. In line 46, user could turn on or off the $fp16 = True$, which is related to mixed precision training, which is supported after CUDA 8 and Pytorch 1.7.1. As it mentioned in previous sections, this could save nearly halves memory requirement on GPU and speed up arithmetic [34].
4. In line 54, user could set the $num\_classes$, its value is the count of the classes of labels they want to segment plus one, which is the background.
5. In line 57, user could set the backbone of the training, as it mentioned before, we will perform the training on herbarium sheets based on the MobileNetv2 backbone.

6. In line 62, user could set the Boolean value of $pretrained = False$, if it equals to $False$, it will load the pre-trained weight of the backbone, and if it equals to $True$, it will perform training from 0 and often it will cause bad output and requires more trains after the first time of training. Due to the time limitation of the project, we set it to be $False$.
7. In line 81, user could set the $model\_path$, which is the pre-trained model, here we use the MobileNetv2.
8. In the line 86, user could set the $downsample\_factor$, which is the downsample factor in the DCNN module in the Encoder of DeepLabv3+. In our situation, it means the number of convolutions in the MobileNetv2, if $downsample\_factor = 8$ it will perform three times convolutions, which is the same value of the origin paper as it shown in Fig 17 [22]. However, this will cause OOM problems, and in our model, we set it to be $downsample\_factor = 16$, which means we are going to do four times of convolutions in the MobileNetv2.
9. In line 89, user could set the size of the input image for the Encoder and Decoder, in our project we set $input\_shape = [512,512]$.
10. In line 100 to line 198, users could decide if they want to enable the $Freeze\_Train$, and the total training epoch and batch size of both freeze and unfreeze steps. Because we are going to use the MobileNetv2 as the backbone, we would like to train with it in the $Freeze\_Epoch = 50$ with the $Freeze\_batch\_size = 8$ to acerate the convergence of the loss and speed up the training. In addition, this will cost less memory in the freeze step. And we will train without backbone after the first 50 epochs and set the $UnFreeze\_Epoch = 100, Unfreeze\_batch\_size = 4$. The reason why we set smaller batch size in unfreeze step is also related to the limitation of hardware that we only have 8GB memory and our training is performed on the laptop which has bad cooling and the overheat of laptop will cause the shutdown. Though we could continue training with the breakpoint (we save trained model every 5 epoch), it cost a lot of time to wait the laptop cool down.
11. In line 204 and 205, the user could set the initial learning rate and minimum learning rate of the model, we set $Init\_lr = 7e-3, Min\_lr = Init\_lr * 0.01$, which is the recommended learning rate provided by the Google Research on their GitHub origin code comments.
12. In line 211 to line 220, user could set the optimizer type, we provide Adam and SGD optimizer, and provide the recommend configuration of the $momentum$ and $weight\_decay$ of each optimizer for unfreeze and freeze mode of each optimizer. Because the details about the settings have 20 lines, we are not going to discuss in this dissertation, however, we clearly comment them in our source code.
13. In line 226, user could select the learning rate decay type and we provide cosine and step type.
14. In line 228 and line 229, user could configure the $save\_period, save\_dir$.
15. In line 233 and line 234, user could configure the $eval\_flag, eval\_period$. We recommend that to save trained model after each time of evaluation, which means the value of $save\_period$ is same as the $eval\_period$, this could help protect the result of the training.
16. In line 237, user could set the root path of their training dataset, and the requirement for the dataset is mentioned above in Design section and mentioned in the README

file. Also, if the user does not like the requirements, they could reconstruct the code related to this.

17. In line 248 and line 250, we provide the settings for Dice loss and Focal loss, which would help to prevent the imbalance of positive and negative samples in the dataset. And further information about these values will be illustrated in the following section.

The hyperparameters will then override the default parameters in the python scripts in "./nets" package. As it shown in Fig 24, the $Dataloader$ of our DeepLabv3+ will: (1) perform random data arguments on the input image by using "$get\_random\_data()$" function. (2) turn the image into one-hot encoding format and ignore the white border (ignore label) in labelled data as it mentioned in previous section. (3) output the jpg data, png data and the label data back into the "training.py".

And the implementation of the data argument, which is "$get\_random\_data()$" function in the "dataloader.py" , is shown in the Fig 25:

(1) We perform random resize on the width and height of the RGB image data from the output of "utils.py". And update the image data and label data after resizing.
(2) We perform random flip on the image data and update the image data and label data.
(3) We add grey bars to the images if the size of the image changes after (1) and (2) and the reason why we need to add the grey bars has already discussed in previous sections. And we update the image and label data.
(4) We perform random blur to the image data by using $cv2.GaussianBlur()$ library. And update the image data if it performs the blur change to the image.
(5) We perform random rotate to the image data by using cv2 libraries.
(6) We perform the random colour gamut transformation on the image by using cv2 and $numpy$ libraries.
(7) After the random data argument, we return the image data and label into the "$\_\_getitem\_\_$" function as it shown in Fig 24 (1). The $image\_data$ is the jpg image, and the label is the png image that contains the classes of the labels.

```
25      def __getitem__(self, index):
26          annotation_line = self.annotation_lines[index]
27          name = annotation_line.split()[0]
28
29          # IMPORTANT! NEED TO MODIFIED HERE!
30          jpg = Image.open(
31              os.path.join(os.path.join(self.dataset_path, "wyoming_combine_origin/JPEGImages"), name + ".jpg"))
32          png = Image.open(
33              os.path.join(os.path.join(self.dataset_path, "wyoming_combine_origin/SegmentationClass"), name + ".png"))
34  (1)     # data argumentation
35          jpg, png = self.get_random_data(jpg, png, self.input_shape, random=self.train)
36
37          jpg = np.transpose(preprocess_input(np.array(jpg, np.float64)), [2, 0, 1])
38          png = np.array(png)
39          png[png >= self.num_classes] = self.num_classes
40          # 转化成one_hot的形式
41          #  self.num_classes + 1 is beause we have ignore_label, 也就是object的白边部分
42          # 我们需要将白边部分进行忽略，+1的目的是方便忽略。
43  (2)     seg_labels = np.eye(self.num_classes + 1)[png.reshape([-1])]
44          seg_labels = seg_labels.reshape((int(self.input_shape[0]), int(self.input_shape[1]), self.num_classes + 1))
45
46  (3)     return jpg, png, seg_labels
```

Fig 24. The "$\_\_getitem\_\_$" function in "dataloader.py".

```
    Cuda = True
    ...
    distributed = False
    # sync_bn           是否使用sync_bn，DDP模式多卡可用
    sync_bn = False
    ...
    fp16 = True
    # num_classes       num of claesses + 1(_background_)
    num_classes = 5

    # mobilenet, xception
    backbone = "mobilenet"
    ...
    pretrained = False

    ...
    model_path = "model_data/deeplab_mobilenetv2.pth"

    ...
    downsample_factor = 16

    # size of input image
    input_shape = [512, 512]

    ...
    Init_Epoch = 0
    Freeze_Epoch = 50
    Freeze_batch_size = 8
    ...
    UnFreeze_Epoch = 100
    Unfreeze_batch_size = 4
    ...
    Freeze_Train = True

    ...
    Init_lr = 7e-3
    Min_lr = Init_lr * 0.01
    ...
    optimizer_type = "adam"
    ...
    momentum = 0.9
    ...
    weight_decay = 0
    ...
    lr_decay_type = 'cos'
    # save log every 5 epoch
    save_period = 5
    save_dir = 'logs'
    ...
    eval_flag = True
    eval_period = 5

    # Herbarium_path  数据集路径
    Herbarium_path = 'herbarium_sheets'
    ...
    dice_loss = True
    # enable/disable focal loss to prevent the imbalance of positive and negative samples
    focal_loss = True

    # every class's loss weight，default to be balanced
    cls_weights = np.ones([num_classes], np.float32)
    ...
    num_workers = 1
```

Fig 23. Hyperparameters in “train.py”.

```
        # 对图像进行缩放并且进行长和宽的扭曲
        # random resize
(1)     new_ar = iw / ih * self.rand(1 - jitter, 1 + jitter) / self.rand(1 - jitter, 1 + jitter)
        scale = self.rand(0.25, 2)
        if new_ar < 1:
            nh = int(scale * h)
            nw = int(nh * new_ar)
        else:
            nw = int(scale * w)
            nh = int(nw / new_ar)
        image = image.resize((nw, nh), Image.BICUBIC)
        label = label.resize((nw, nh), Image.NEAREST)

        # 翻转图像
(2)     # random flip
        flip = self.rand() < .5
        if flip:
            image = image.transpose(Image.FLIP_LEFT_RIGHT)
            label = label.transpose(Image.FLIP_LEFT_RIGHT)

        # 将图像多余的部分加上灰条
(3)     # add grey bars to the extra parts of the images
        dx = int(self.rand(0, w - nw))
        dy = int(self.rand(0, h - nh))
        new_image = Image.new('RGB', (w, h), (128, 128, 128))
        new_label = Image.new('L', (w, h), (0))
        new_image.paste(image, (dx, dy))
        new_label.paste(label, (dx, dy))
        image = new_image
        label = new_label

        image_data = np.array(image, np.uint8)
        # 高斯模糊
(4)     # random blur
        blur = self.rand() < 0.25
        if blur:
            image_data = cv2.GaussianBlur(image_data, (5, 5), 0)

        # 旋转
        # random rotate
        rotate = self.rand() < 0.25
(5)     if rotate:
            center = (w // 2, h // 2)
            rotation = np.random.randint(-10, 11)
            M = cv2.getRotationMatrix2D(center, -rotation, scale=1)
            image_data = cv2.warpAffine(image_data, M, (w, h), flags=cv2.INTER_CUBIC, borderValue=(128, 128, 128))
            label = cv2.warpAffine(np.array(label, np.uint8), M, (w, h), flags=cv2.INTER_NEAREST, borderValue=(0))

        # 对图像进行色域变换
        # 计算色域变换的参数
(6)     # random color gamut transformation
        r = np.random.uniform(-1, 1, 3) * [hue, sat, val] + 1
        # 将图像转到HSV上
        hue, sat, val = cv2.split(cv2.cvtColor(image_data, cv2.COLOR_RGB2HSV))
        dtype = image_data.dtype
        # 应用变换
        x = np.arange(0, 256, dtype=r.dtype)
        lut_hue = ((x * r[0]) % 180).astype(dtype)
        lut_sat = np.clip(x * r[1], 0, 255).astype(dtype)
        lut_val = np.clip(x * r[2], 0, 255).astype(dtype)

        image_data = cv2.merge((cv2.LUT(hue, lut_hue), cv2.LUT(sat, lut_sat), cv2.LUT(val, lut_val)))
        image_data = cv2.cvtColor(image_data, cv2.COLOR_HSV2RGB)

(7)     return image_data, label
```

Fig 25. Implementation of data argument.

6.3.5 Implementation of post-processing

As it shown in Fig 22 (5), after getting the Pytorch format of image data, we apply "$torch.nn.functional.softmax()$" function to get the class label of each pixel. Then as it shown in Fig 22 (6), we crop the grey bars added before in the pre-processing step.

As it shown in Fig 22 (7), we use "$numpy.reshape()$" function to assign class labels with colors set before in the "$\_\_init\_\_()$" function of the "deeplab.py" into the detect objects. And then as it shown in Fig 22 (8), we convert the segmented image data into "$PIL.Image$" format and could alternatively perform "$PIL.Image.blend()$" function to mix the origin image with the processed image as it shown in Fig 22 (9). In this project, we do experiments with "$mixed = True$", which means we will mix the original image with the segmented image because this could help us compare the results with the ground truth and origin images.

6.3.6 Implementation of user-friendly tools

As it mentioned in Requirement and Design sections, we are going to implement some user-friendly tools to boost our project.

As it shown in Fig 26, we provide "$convert\_labelled.py$" for users to either convert the name of the files or convert 24-bit depth RGB image into 8-bit depth RGB image by using os and $PIL.Image$ libraries.

As it shown in Fig 27, we provide "$get\_annotation.py$" for users to get the train-test-val datasets for the training process. And output the configuration of the train-test-val settings into *.txt files and automatically write into disk.

As it shown in Fig 28, we provide "$labelme\_to\_deeplab.py$" for users to convert the *.json files from $Labelme$ into the 8-bit depth RGB *.png images after labelling the raw images. And automatically save these images into the disk.

```
4   # 去掉_label后缀
5   path = 'herbarium_sheets/mixseths/SegmentationClass'
6   for filename in os.listdir(path):
7       os.rename(os.path.join(path, filename), os.path.join(path, filename[:-11] + '.png'))
8
9   # 24转8bit
10  for filename in os.listdir(path):
11      img = Image.open(os.path.join(path, filename))
12      print("convert %s into %s with 8bit...." % (img, filename))
13      img = img.convert('P', palette=Image.ADAPTIVE, colors=8)
14      img.save(os.path.join(path, filename))
```

Fig 26. Implementation of "convert_labelled.py".

```
random.seed(0)
print("Generate txt in ImageSets.")
# IMPORTANT! NEED TO MODIFY HERE!
segfilepath = os.path.join(Herbarium_path, 'wyoming_combine_origin/SegmentationClass')
saveBasePath = os.path.join(Herbarium_path, 'wyoming_combine_origin/ImageSets/Segmentation')

temp_seg = os.listdir(segfilepath)
total_seg = []
for seg in temp_seg:
    if seg.endswith(".png"):
        total_seg.append(seg)

num = len(total_seg)
list = range(num)
tv = int(num * trainval_percent)
tr = int(tv * train_percent)
trainval = random.sample(list, tv)
train = random.sample(trainval, tr)

print("train and val size", tv)
print("traub suze", tr)
ftrainval = open(os.path.join(saveBasePath, 'trainval.txt'), 'w')
ftest = open(os.path.join(saveBasePath, 'test.txt'), 'w')
ftrain = open(os.path.join(saveBasePath, 'train.txt'), 'w')
fval = open(os.path.join(saveBasePath, 'val.txt'), 'w')
```

Fig 27. Implementation of “get_annotation.py”.

```
if os.path.isfile(path) and path.endswith('json'):
    data = json.load(open(path))

    if data['imageData']:
        imageData = data['imageData']
    else:
        imagePath = os.path.join(os.path.dirname(path), data['imagePath'])
        with open(imagePath, 'rb') as f:
            imageData = f.read()
            imageData = base64.b64encode(imageData).decode('utf-8')

    img = utils.img_b64_to_arr(imageData)
    label_name_to_value = {'_background_': 0}
    for shape in data['shapes']:
        label_name = shape['label']
        if label_name in label_name_to_value:
            label_value = label_name_to_value[label_name]
        else:
            label_value = len(label_name_to_value)
            label_name_to_value[label_name] = label_value

    # label_values must be dense
    label_values, label_names = [], []
    for ln, lv in sorted(label_name_to_value.items(), key=lambda x: x[1]):
        label_values.append(lv)
        label_names.append(ln)
    assert label_values == list(range(len(label_values)))

    lbl = utils.shapes_to_label(img.shape, data['shapes'], label_name_to_value)

    PIL.Image.fromarray(img).save(osp.join(jpgs_path, count[i].split(".")[0] + '.jpg'))

    new = np.zeros([np.shape(img)[0], np.shape(img)[1]])
    for name in label_names:
        index_json = label_names.index(name)
        index_all = classes.index(name)
        new = new + index_all * (np.array(lbl) == index_json)

    utils.lblsave(osp.join(pngs_path, count[i].split(".")[0] + '.png'), new)
    print('Saved ' + count[i].split(".")[0] + '.jpg and ' + count[i].split(".")[0] + '.png')
```

Fig 28. Implementation of “labelme_to_deeplab.py”.

6.4 Project management

The code management is using Git, and the GitHub repository could be accessed at: https://github.com/lyzsk/analysis-of-herbarium-sheets.

6.5 Results and Analysis

During the development of this project, we have tried considerable numbers of experiments based on different hypotheses, try to meet all the requirements and also intend to improve not only the performance but also make the training process lightweight and faster in speed.

Before starting the experiments, we need to firstly introduce the evaluation indicators of semantic segmentation. We are going to use the $mIOU, mPA, mPrecision$ and $mRecall$ as the evaluation indicators [23]:

1. The $mIOU$ (mean intersection over union) is to calculate the mean ratio of the intersection of the pixel-wise classification results with the ground truth, to their union, $mIOU = \frac{1}{k}\sum_{j=1}^{k} \frac{n_{jj}}{n_{ij}+n_{ji}+n_{jj}}, i \neq j$. The $mIOU$ calculation formula can also be written as: $mIOU = \frac{1}{k}\sum \frac{TP}{FP+FN+TP}$, where TP is the true positive, FP is the false positive, TN is the true negative and FN is the false negative.
2. The mPA (mean pixel accuracy) is to calculate the mean ratio between the amount of properly classified pixels and their total number, $mPA = \frac{1}{k}\sum_{j=1}^{k} \frac{n_{jj}}{i_j}$.
3. The $mPreision$ is the mean ratio of hits over a summation of hits and false alarms, $mPreision = \frac{1}{k}\sum_{j=1}^{k} \frac{n_{jj}}{n_{ij}+n_{jj}}$. It could be also calculated as the formula: $mPreision = \frac{1}{k}\sum \frac{TP}{TP+FP}$.
4. The $mRecall$ is the mean ratio of hits over a summation of hits and misses, $mRecall = \frac{1}{k}\sum_{j=1}^{k} \frac{n_{jj}}{n_{ji}+n_{jj}}$. It could be also calculated as the formula: $mRecall = \frac{1}{k}\sum \frac{TP}{TP+FN}$.

#	Size	Element	$mIOU$	mPA	$mPrecision$	$mRecall$
NMW	500	Scale	0.90	0.96	0.94	0.96
		Colour chart	0.61	0.71	0.82	0.71
		Label	0.78	0.85	0.90	0.85
		Barcode	0.94	0.98	0.96	0.98
		Total	0.7722	0.8623	0.8700	0.8623
MNHN	500	Scale	0.93	0.98	0.95	0.98
		Colour chart	0.61	0.69	0.85	0.69
		Label	0.92	0.95	0.97	0.95
		Barcode	0.82	0.89	0.92	0.89
		Total	0.7790	0.8685	0.8729	0.8685
Mixed	1000	Scale	0.90	0.94	0.95	0.94
		Colour chart	0.88	0.94	0.93	0.94
		Label	0.66	0.77	0.81	0.77
		Barcode	0.68	0.81	0.81	0.81
		Total	0.7050	0.8244	0.8051	0.8244
Combined	2000	Scale	0.88	0.95	0.92	0.95
		Colour chart	0.79	0.86	0.91	0.86
		Label	0.81	0.88	0.90	0.88
		Barcode	0.81	0.88	0.91	0.88
		Total	0.7615	0.8776	0.8462	0.8776
Combined with Wyoming	1000	Scale	0.89	0.95	0.94	0.95
		Colour chart	0.85	0.93	0.90	0.93
		Label	0.87	0.93	0.94	0.93
		Barcode	0.77	0.86	0.87	0.86
		Total	0.8698	0.9313	0.9269	0.9313

Table 3. The $mIOU, mPA, mPrecision, mRecall$ result of first experiment.

6.5.1 First experiment and hypothesis

The first experiment is to train the labelled dataset from NMW (500 images), MNHN (500 images), Mixed (1000 images) separately with baseline model, then combine them together into "Combined" dataset (2000 images), train with baseline model, and try to apply these trained models on Wyoming University's dataset to evaluate the prediction performance. As

it shown in Table 3, the MNHN dataset has the highest $mIOU$ and mPA among the datasets followed closely by the NMW dataset, and Mixed dataset and the combined dataset, even with the largest labelled training data, has the worst output.

Intuitively, we have the hypothesis that the MNHN and NMW datasets have the most similar images with the Wyoming University images. As it shown in Fig 33 first column, the shape and colour diversity are different among MNHN, NMW and Wyoming dataset's Colour charts, Mixed dataset has Colour chart that contains colour RGB(0, 0, 0) and RGB(255, 255, 255) on the ends of the chart, which is similar to the Scale and will make the program hard to distinguish the pixel class of the Colour chart and the Scale. In addition, after checking the datasets we found that although the Mixed dataset is more than others, the images that contains Colour chart is less. In other words, the Mixed dataset has high volumes of images that do not contains Colour chart object, and hence make the Colour chart ground truth samples less for the training model. Therefore, the training result of the Mixed dataset and Combined dataset are worse than others. And given to this, it also proves that the Combined dataset has potential to get a better performance and result.

Moreover, the prediction results show that three separated model and Combined model cannot clearly segment the different object from the background. As it mentioned above in the previous section, the program is designed to label the "Barcode", "Label", "Colour chart", "Scale" into Red, Green, Yellow, Blue. However, all these models cannot label the Barcode object in a right way and label the background into Blue by mistake. After checking the bit-depth of the dataset's labelled image, I found that the $*.png$ labelled images are 24-bit depth, however, the DeepLabv3+ requires 8-bit depth $*.png$ image as the labelled data. Hence, as it shown in Fig 26, I wrote the script "$convert\_labelled.py$", which uses $PIL.Image$ library to help process the origin 24-bit depth PNG images into 8-bit depth PNG images. After this process, as it shown in Fig 29 (the first row is before conversion of the bit depth, the second row is after), the Combined model could successfully predict the Wyoming University's data $RM0008192.JPG$, and will not label the background by mistake. However, new problem came up that the model still cannot segment the Barcode and Scale successfully as it shown in Fig 29 third column second row. The experiment based on this problem will be discussed in next section.

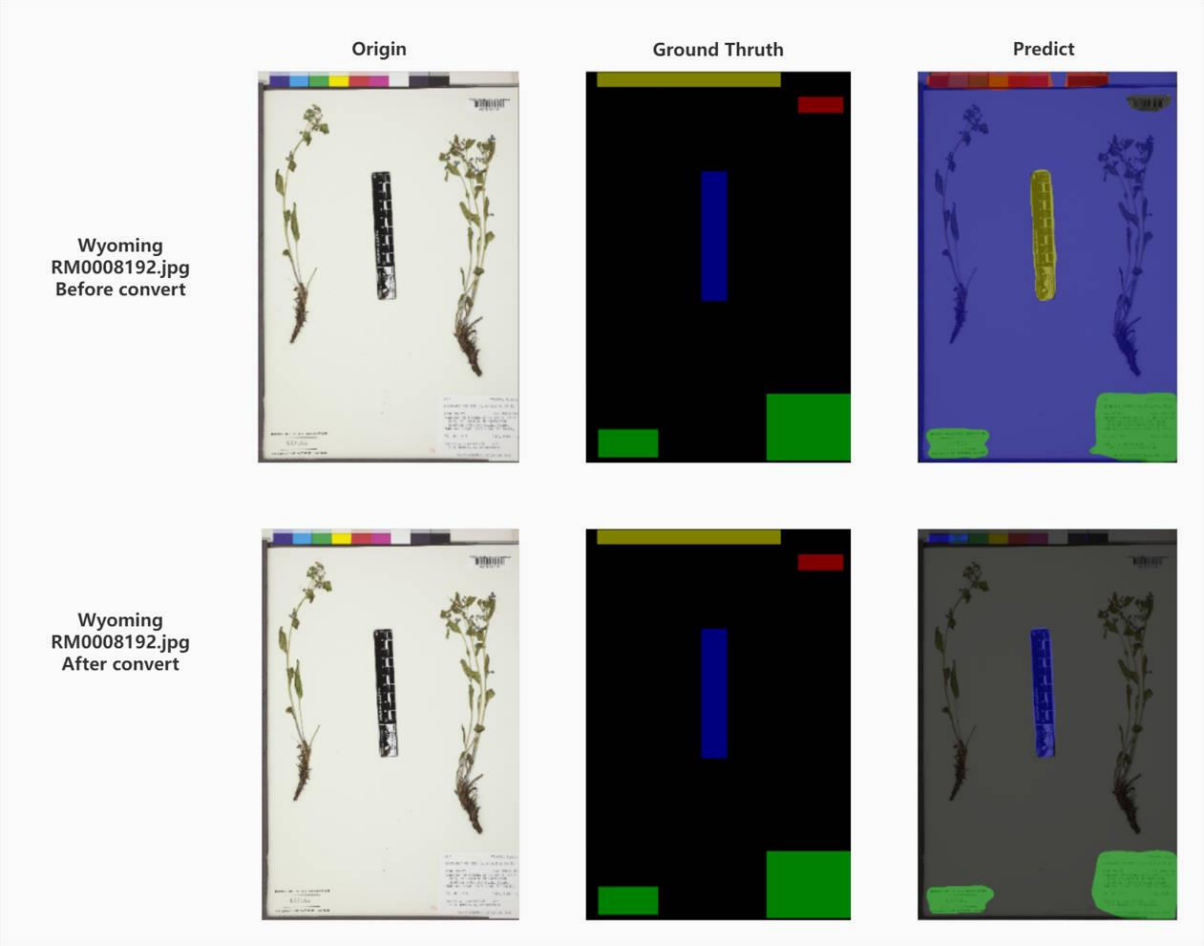


Fig 29. The example results before and after convert bit depth of labelled images.

Fig 30. The origin-ground truth-predict result of second experiment.

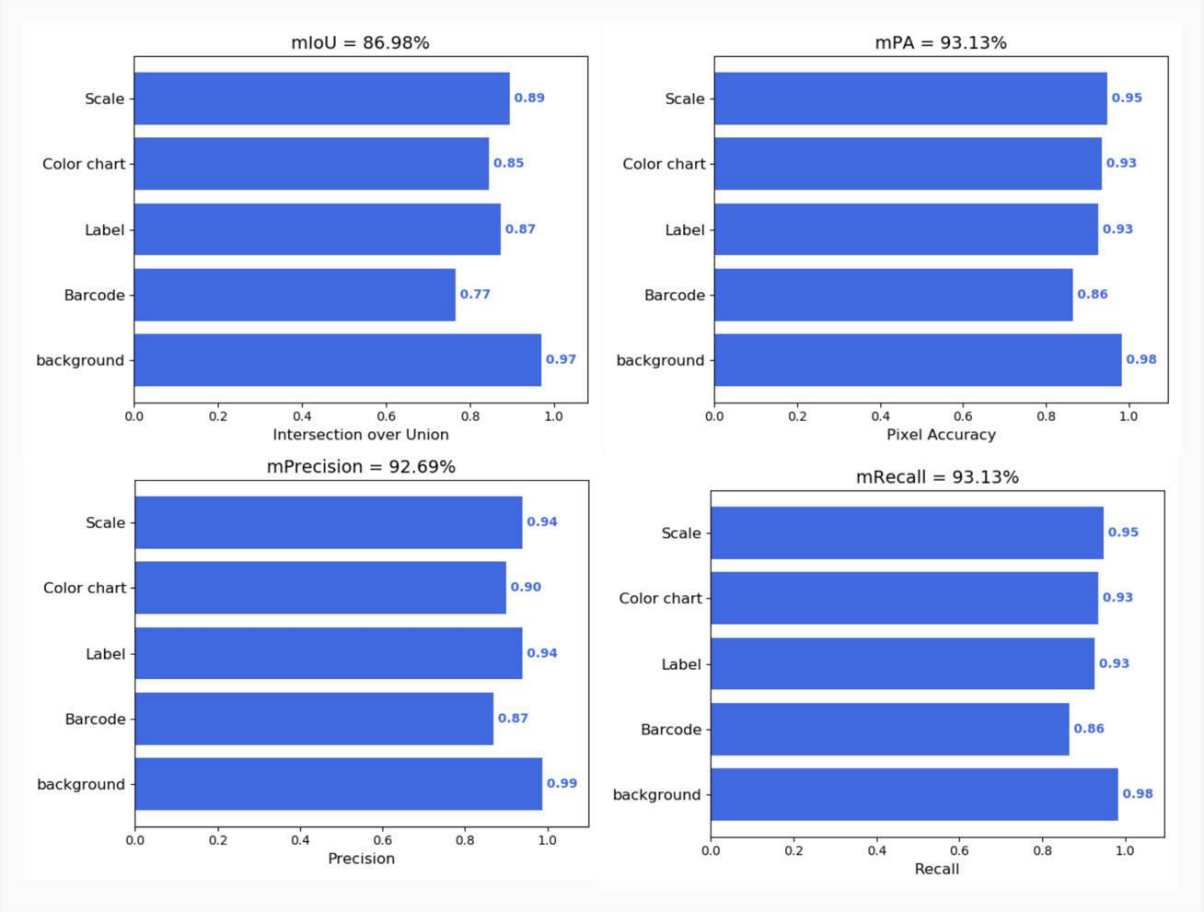


Fig 31. The output of the second experiment.

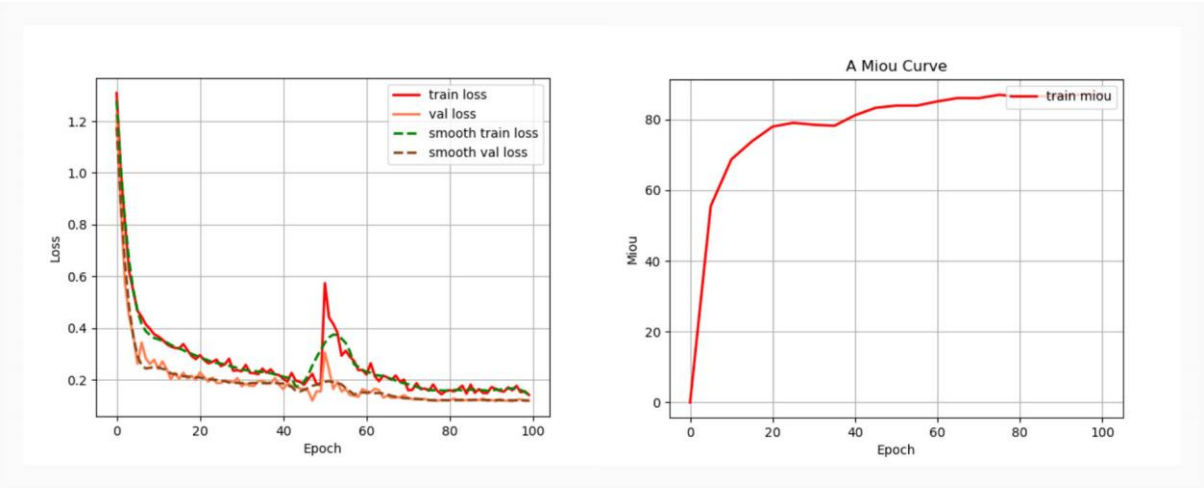


Fig 32. The loss curve and $mIOU$ curve of the second experiment.

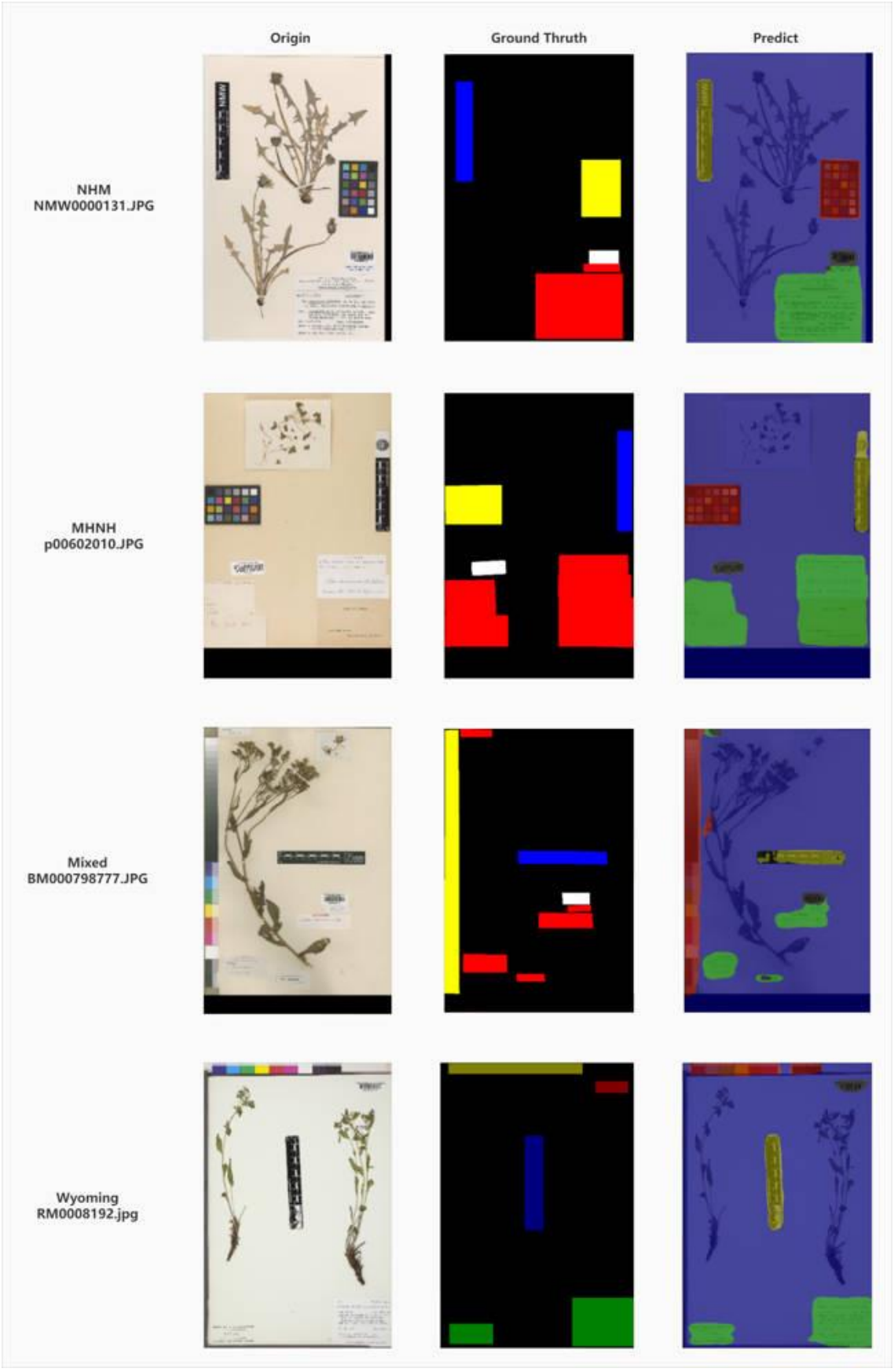


Fig 33. The origin-ground truth-predict result of second experiment.

6.5.2 Second experiment and analysis

Under the supervisor's mentor, in the second experiment, we decided to add some Wyoming images into the Combined dataset. Considering labelling new dataset takes a lot of time, and due to the project time constraints, the Combined Wyoming dataset will have only 1000 images with labels: 200 images from Wyoming University, 200 images from NMW, 200 images from MNHN, 400 images from Mixed. The process workflow of labelling Wyoming University dataset will use $Labelme$ under Anaconda3 environment. As it shown in Fig 30, the fine-tuned model of second experiment could successfully predict the Scale and Barcode, which used the same image as first experiment. And the output values are shown in Fig 31 and Fig 32, the overall values of $mIOU, mPA, mPrecision, mRecall, loss\ curve\ and\ mIOU\ curve$ are better than the previous ones. However, as it discussed in the before sections, the Wyoming University dataset varies in types from different collections. The numerical values should not the only evaluation indicators, we should also apply the fine-tuned model to predict the special image samples in the dataset. Also, the special images may not be labelled before and used as the train set to feed the training model.

As it shown in Fig 34 and Fig 35, we picked six special images:

(1) RM0256583.jpg (Fig 34 first row) is the unlabelled image that the Labels class cover some part of the Specimens. As it shown in third column, the prediction result of the Labels in bottom left corner and bottom right corner could only segment the part that contains text but cannot identify the boundary between Label and Specimen. And this problem appears commonly in all predict results.

(2) RM0273438.jpg (Fig 34 second row) is the unlabelled image that the specimen covers part of the label in the bottom right corner of the image. Same as the (1), the Label object cannot be fully identified and segment with green colour.

(3) RM0307498.jpg (Fig 34 third row) is the unlabelled image that the Label objects contains an object that do not has any text in it. And as it shown in the third column of Fig[TODO:], the predict result cannot identify the blank Label object.

(4) RM0325201.jpg (Fig 35 first row) is the unlabelled image that the Label on the bottom right corner contains handwritten text. The predict result show that, the model has the ability to segment not only printed text but also handwritten text.

(5) RM0325430.jpg (Fig 35 second row) is the labelled image that contains Label that covered by the root of the specimen (in the bottom left corner). It also contains multiple Barcode. As it shown in the predict result, unlike situation (1), the model could successfully identify the full Label class even it is covered by specimen. However, one of the Barcode has its border to be labelled with yellow, which should be labelled for Colour chart. In other words, the model might be overfitted.

(6) USFS0057359.jpg (Fig 35 third row) is the unlabelled image that contains no Colour chart. As it shown in the predict result, the model could successfully segment all the objects inside the image.

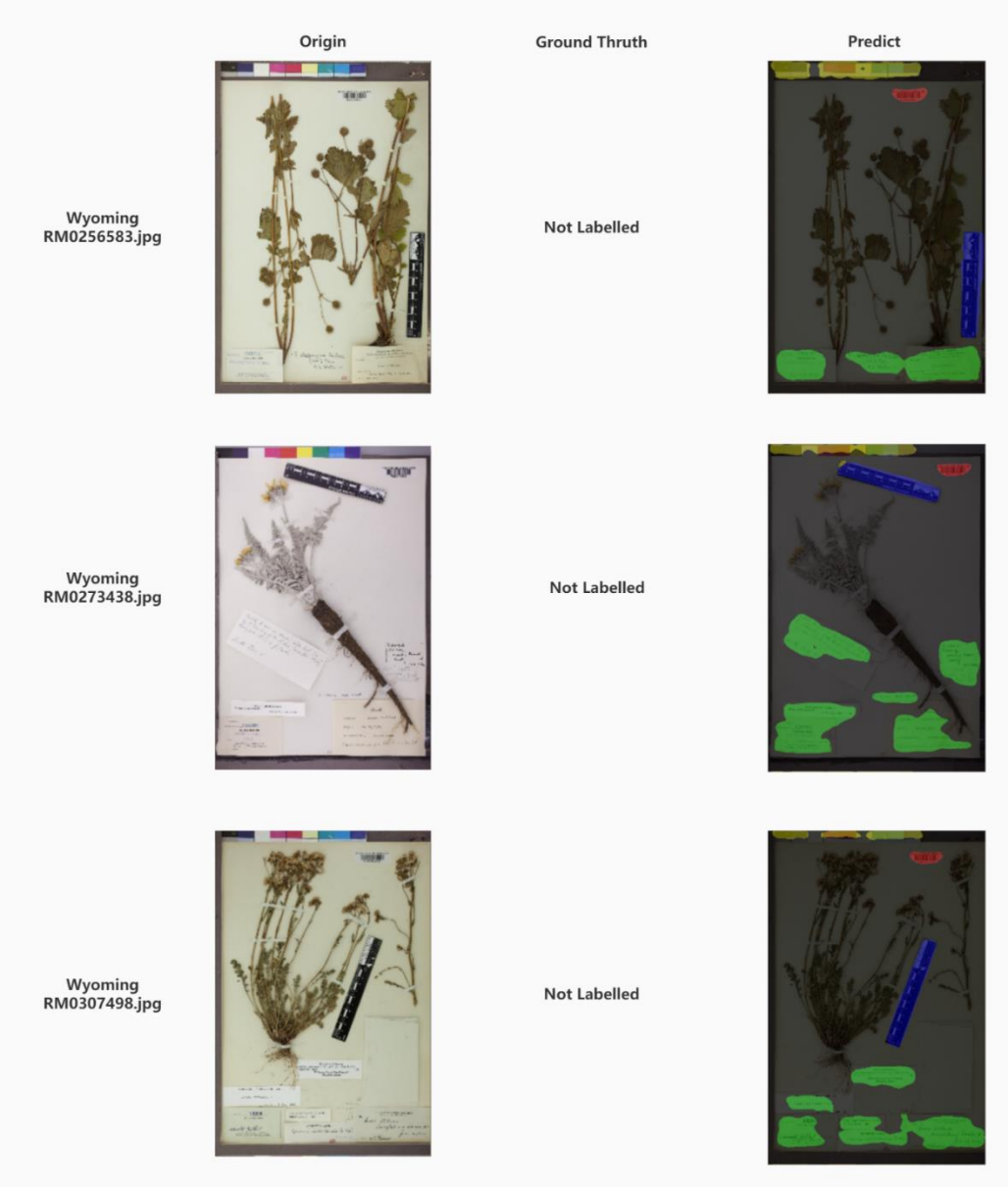


Fig 34. The origin-ground truth-predict for special images in the second experiment.

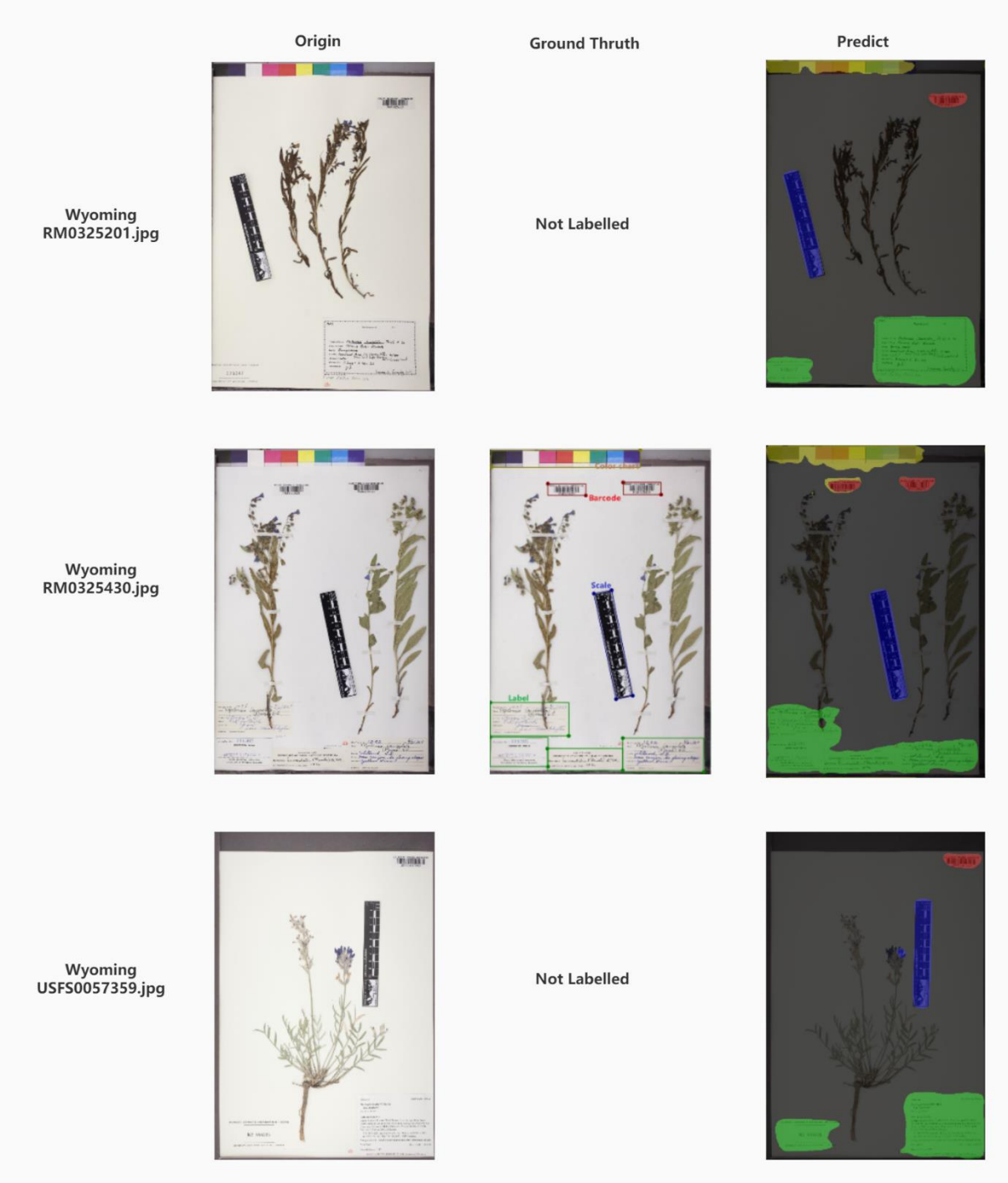


Fig 35. The origin-ground truth-predict for special images in the second experiment.

Given to the six special images predict result, we could conclude that our model succeeded to a certain extent, but it still has potential to be improved. Our aim is to label the objects more precisely. For example, by comparing the result of (1) and (5), we would like to improve our model to predict the Label objects fully in unlabelled image. In other words, we would like to not only increase the $mIOU$ value, but also improve the applicability of our rebuilt DeepLabv3+ model. In this case, I draw a hypothesis that the overfit problem for multiple

Barcode and the incomplete label for covered Label object could be caused by the loss function, optimizer, learning decay type and learning rate. And the following experiments will focus on these.

#	Element	$mIOU$	mPA	$mPrecision$	$mRecall$
CE only	Scale	0.89	0.95	0.94	0.95
	Colour chart	0.85	0.93	0.90	0.93
	Label	0.87	0.93	0.94	0.93
	Barcode	0.77	0.86	0.87	0.86
	Total	0.8698	0.9313	0.9269	0.9313
FL only	Scale	0.85	0.92	0.92	0.92
	Colour chart	0.71	0.81	0.85	0.81
	Label	0.81	0.89	0.91	0.89
	Barcode	0.58	0.77	0.70	0.77
	Total	0.7815	0.8727	0.8710	0.8727
CE + DL	Scale	0.90	0.95	0.95	0.95
	Colour chart	0.85	0.93	0.91	0.93
	Label	0.88	0.94	0.94	0.94
	Barcode	0.80	0.88	0.90	0.88
	Total	0.8794	0.9342	0.9357	0.9342
FL + DL	Scale	0.89	0.93	0.95	0.93
	Colour chart	0.84	0.92	0.90	0.92
	Label	0.88	0.95	0.92	0.95
	Barcode	0.81	0.90	0.89	0.90
	Total	0.8794	0.9378	0.9323	0.9378

Table 4. The $mIOU, mPA, mPrecision, mRecall$ result of the third experiment.

6.5.3 Third experiment and analysis

As it mentioned in 6.5.2, the third experiment is focused on the loss function. The baseline of the model relies on the cross-entropy loss (CE loss), however, as it mentioned in the Problem section before, Dice loss (DL) and Focal loss (FL) have the potential to adjust the existing model architecture. Dice loss could help to find a balance between learning the intrinsic anatomical variability of all the classes and tolerable level of class imbalance, Focal loss could act as a more efficient approach than CE loss to achieve higher accuracy on dense

imbalanced dataset, and the combination of the two could improve the performance for segmentation models even in medical dataset [24, 25, 26]. This experiment will enable Dice loss and Focal Loss separately, and then combine them together, and evaluate the outcomes.

As it shown in Table 4, Focal loss (FL) only loss function has the worst result in $mIOU$, however, its values are more average, and the Focal loss combined Dice loss (FL + DL) has the highest $mIOU$, mPA, $mPrecision$ and $mRecall$ values. However, the lowest value for FL only model does not mean that FL is worse than CE. As it shown in Fig 37, the FL + DL $mIOU$ curve is smoother than the CE + DL one and also the train loss and validation loss curve, which means the convergence of the FL + DL model is more stable.

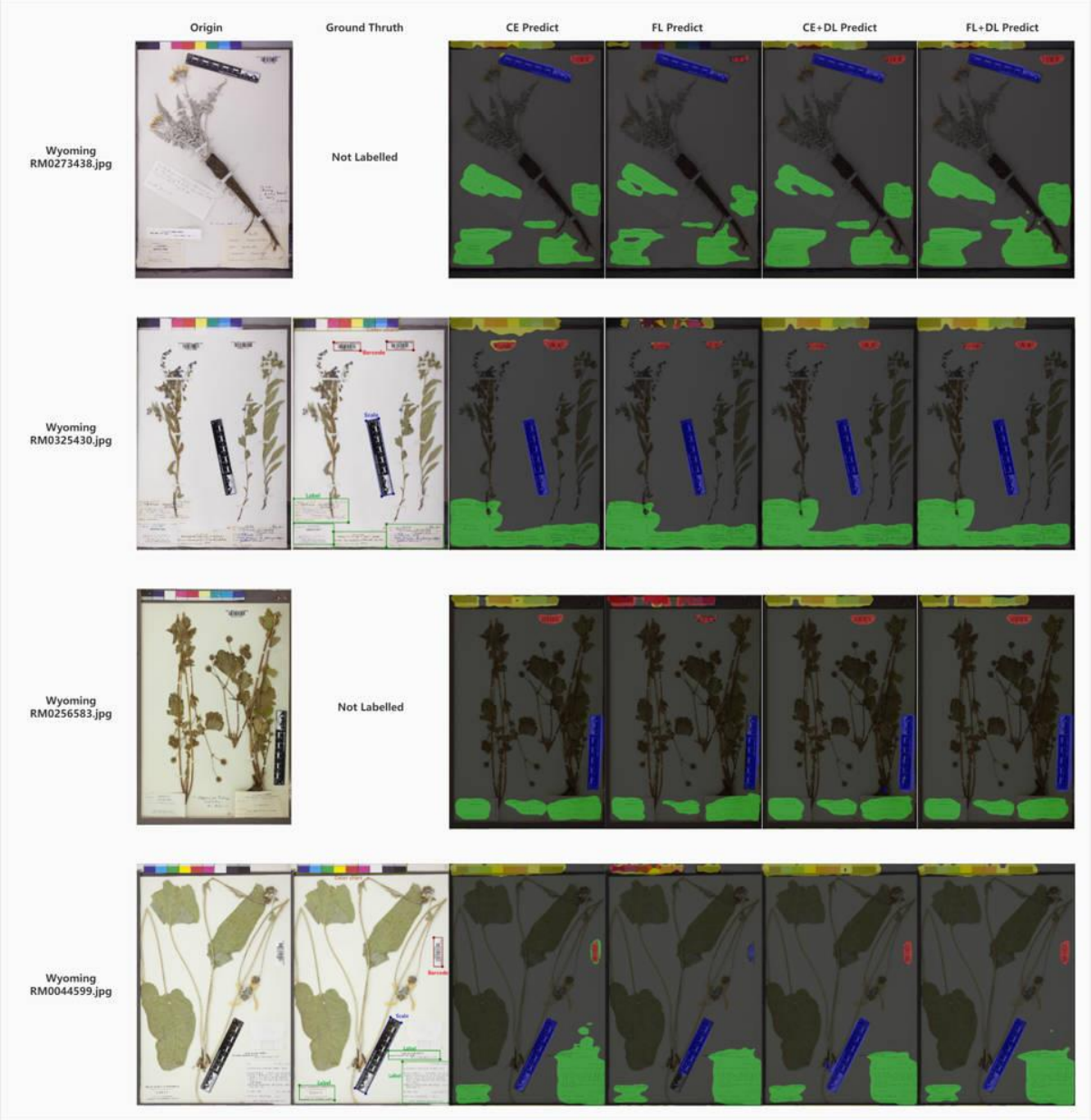


Fig 36. The origin-ground truth-predict for the third experiment.

Additionally, although the FL + DL model's evaluation values are close to the CE + DL function, they both have 87% $mIOU$, 93% mPA, 93% $mPrecision$ and 93% $mRecall$, as it shown in Fig 36, FL + DL model has better performance than CE + DL model in extreme situations:

1. RM0273438.jpg is the unlabelled image (without ground truth) that the root of the Specimen covers some parts of the Label in the bottom right. The FL + DL model could assign the covered part of the Label object into Green colour, even though not fully in Green just a little part. In addition, it could identify the blank part of the Label object without text.
2. RM0325430.jpg is the labelled image (with ground truth) that the root of the Specimen covers parts of the Label object in the bottom left. In this situation, the FL + DL and CE + DL model has similar performance that outstand the four models by comparing to the ground truth. Conversely, the CE model and FL model cannot assign classes label to the Colour chart and the Barcode rightly and identify less area of the covered part.
3. RM0256583.jpg is the unlabelled image that the Label in to bottom right corner covers the root of the Specimen. The FL model has the worst performance that it mistakes the Colour chart object into Barcode object. In other words, it assigns the classes label of Colour chart into Red. Although the CE + DL model could label more RGB(255, 255, 255) area of the Colour chart in the top and more blank area of the Label object in the bottom right, it has problem that mistake some part of the root into Scale object, and it assigns the root of Specimen object into Blue. However, the FL + DL could label the Scale more precisely into rectangular shape, and it can still label the blank part of the Label and identify the RGB(255, 255, 255) region of the Colour chart, though the recognized area will be a bit less than CE + DL model.
4. RM0044599.jpg is the labelled image that the Scale covers part of the Specimen. The CE model has the worst performance that mistake Barcode into Label object and mistake the background into Label object. The FL model cannot label the Scale object fully. The CE + DL model and FL + DL model has similar performance, however, the CE + DL model mistake some part of the root into Scale. And the FL + DL label the objects more like rectangles by comparing to the ground truth.

Based on the comparison on the previous results and analysis, it proves the hypothesis that the problem of not fully labelled objects in prediction is caused by the imbalance of the dataset that some classes labels are less in ground truth and some are smaller than others. And the Focal loss combined with Dice loss function could help to improve the balance of our dataset and improve the performance on the Wyoming University dataset, even in extreme situation as it discussed above.

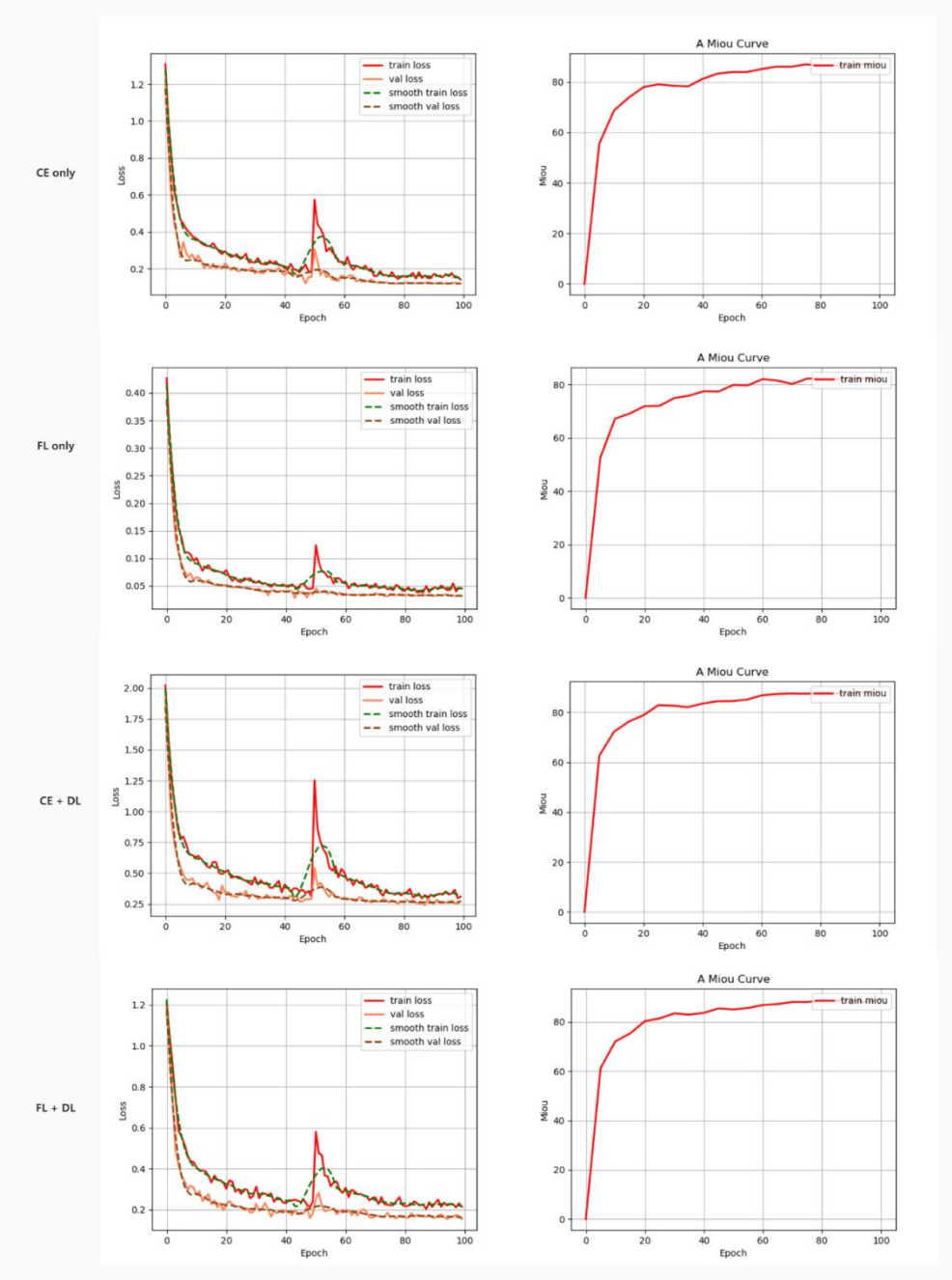


Fig 37. The loss curve and $mIOU$ curve of the third experiment.

#	Element	$mIOU$	mPA	$mPrecision$	$mRecall$
SGD	Scale	0.89	0.93	0.95	0.93
	Colour chart	0.84	0.92	0.90	0.92
	Label	0.88	0.95	0.92	0.95
	Barcode	0.81	0.90	0.89	0.90
	Total	0.8794	0.9378	0.9323	0.9378
Adam	Scale	0.92	0.95	0.96	0.95
	Colour chart	0.88	0.96	0.91	0.96
	Label	0.90	0.96	0.94	0.96
	Barcode	0.85	0.94	0.90	0.94
	Total	0.9065	0.9580	0.9433	0.9580

Table 5. The $mIOU, mPA, mPrecision, mRecall$ result of the fourth experiment.

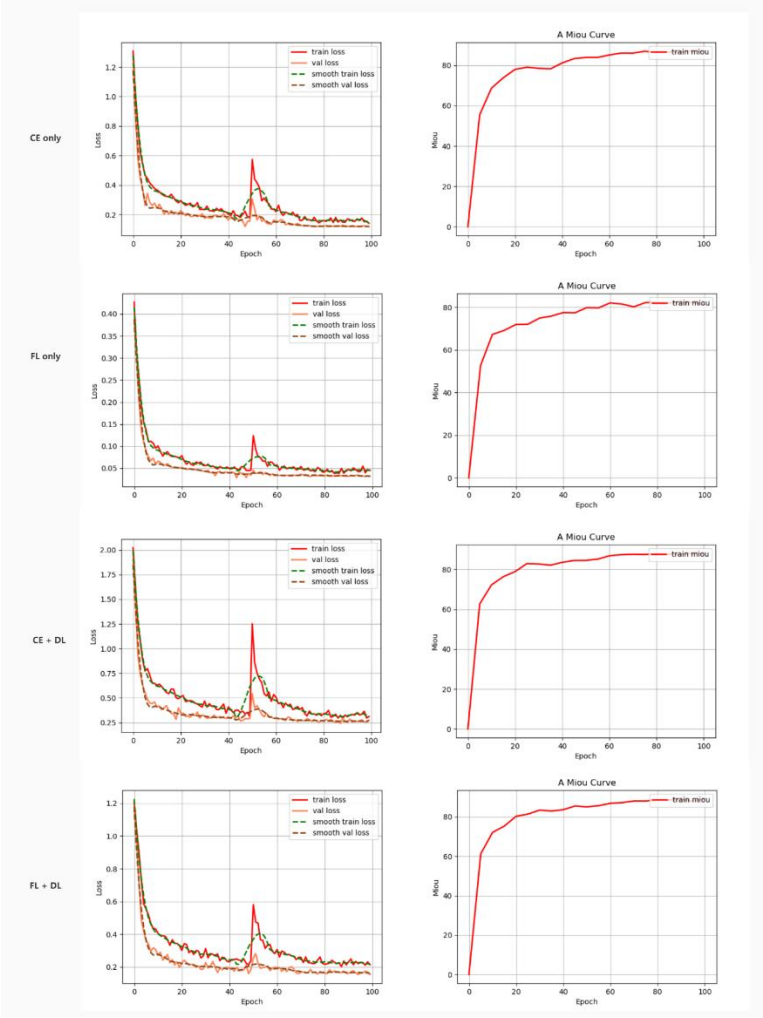


Fig 38. The loss curve and $mIOU$ curve of the fourth experiment.

6.5.4 Fourth experiment and analysis

As it mentioned in 6.5.2, we have the problem that cannot fully label the desired objects, though the 6.5.3 experiment solve the problem aroused from imbalanced dataset, the descent rate of the loss curve and ascent rate of the $mIOU$ curve increases too fast. And the $mIOU$ value of the FL + DL model still lower than our expectations, which should be as closer to 100% as possible. In this experiment, we are going to focus on the optimizer, and find out if the optimizer could help us to solve the problems.

As it shown in Table 5, the Adam optimizer has higher scores in $mIOU, mPA, mPrecision,$ $mRecall$ than SGD optimizer. And as it shown in Fig 38, the Adam optimizer loss curve and $mIOU$ curve is smoother than SGD. For the extreme situation, as it shown in Fig 39:

(1) RM0273438.jpg is the unlabelled image (without ground truth). The Adam model could label more region than SGD for the objects. For example, the Colour chart at the top of the image, the model used Adam optimizer could label it as a rectangle and successfully assign classes label to the RGB(255, 255, 255) (white) part of the Colour chart. For the Scale object, it could rightly label the white edge of the Scale that contains the name of the institution. Although the SGD could label more Label objects, in the bottom right of the image, Adam optimizer could clearly and separately label two different Label objects, however the SGD cannot.

(2) RM0325430.jpg is the labelled image (with ground truth). Similar to (1), the model used Adam optimizer has better performance for labelling the Colour chart, especially for the RGB(255, 255, 255) region.

(3) RM0256583.jpg is the unlabelled image that the Label object covers the root of the Specimen in the bottom right corner. Although SGD could label more region for Label object that has intersection with the Specimen, the model used Adam optimizer has better performance for Colour chart object as (1) and (2).

(4) RM0044599.jpg is the labelled image that the Scale object covers part of the Specimen. The model used Adam optimizer could not only successfully identify the RGB(255, 255, 255) region of the Colour chart, but also avoid mistaking the background into Label. And the region it labels as Barcode is closer to the ground truth than the SGD.

As it discussed above, we prove the hypothesis that different optimizer could help to make the loss curve and $mIOU$ curve smoother, and help the model has better performance.

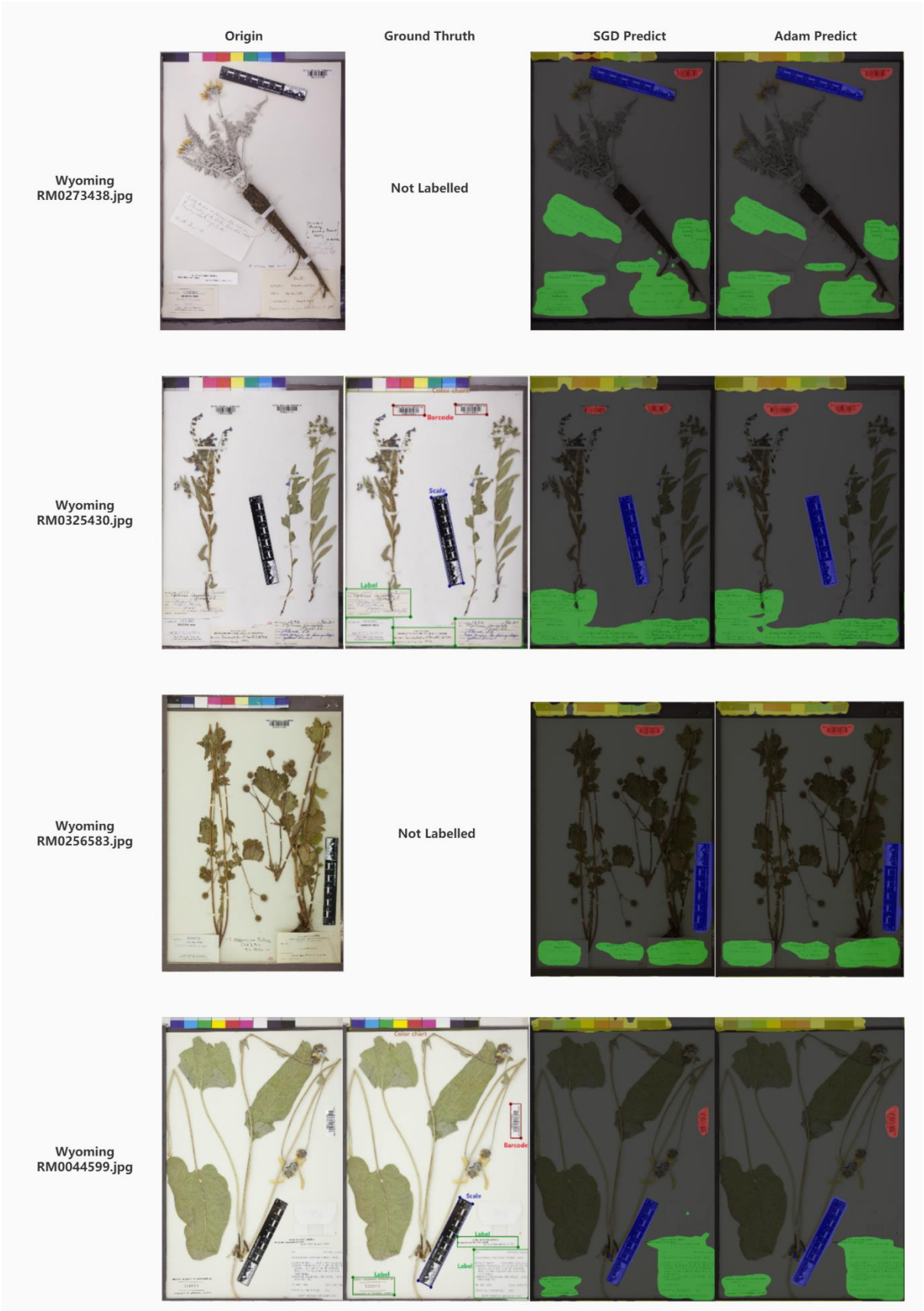


Fig 39. The origin-ground truth-predict for the fourth experiment.

6.5.5 Fifth experiment and analysis

As it mentioned in 6.5.2, this experiment is going to find out if the learning rate decay type and values could help to improve the model. In this experiment we are going to evaluate the cosine learning rate decay (cos) and the step learning rate decay (step). As it shown in Table 6, the Cos learning rate decay has better result than the Step learning rate decay in $mIOU, mPA, mPrecision, mRecall$. And as it shown in Fig 40, the Step learning rate decay's loss curve cannot even convergence in 100 epochs, which means that it may require more epoch or batch size to fit the training model, this is contrary to our requirement that develop stable, portable, flexible, resilient, fast in speed and lightweight model. Moreover, as it shown in Fig 41, the model used Step learning rate decay makes more mistakes and have worse performance. In summary, the Cos learning rate decay is more fit for our model.

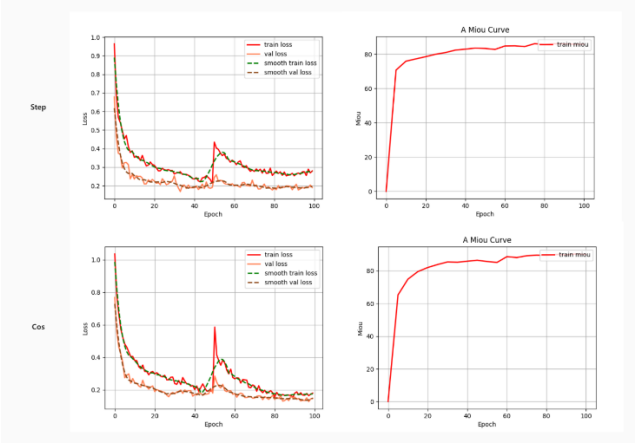


Fig 40. The loss curve and $mIOU$ curve of the fifth experiment.

#	Element	$mIOU$	mPA	$mPrecision$	$mRecall$
Step	Scale	0.86	0.93	0.93	0.93
	Colour chart	0.73	0.80	0.89	0.80
	Label	0.84	0.93	0.89	0.93
	Barcode	0.71	0.90	0.77	0.90
	Total	0.8188	0.9058	0.8930	0.9058
Cos	Scale	0.92	0.95	0.96	0.95
	Colour chart	0.88	0.96	0.91	0.96
	Label	0.90	0.96	0.94	0.96
	Barcode	0.85	0.94	0.90	0.94
	Total	0.9065	0.9580	0.9433	0.9580

Table 6. The $mIOU, mPA, mPrecision, mRecall$ result of the fifth experiment.

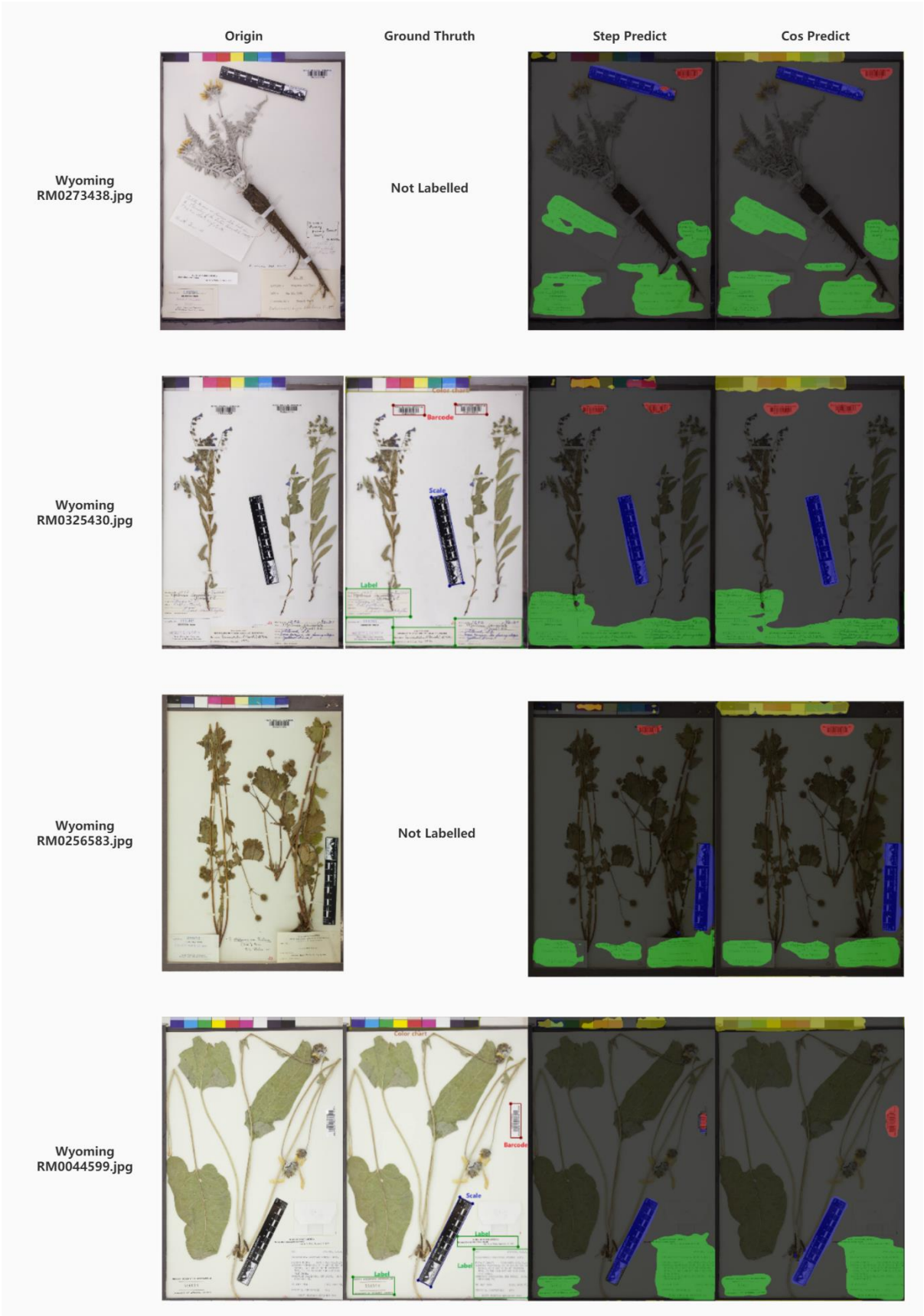


Fig 41. The origin-ground truth-predict for the fifth experiment.

6.5.6 Sixth experiment and analysis

As it discussed in above experiments, we have achieved 90% $mIOU$, which means we have achieved staged success. The sixth experiment will focus on the time spent and memory cost during the training process. We are going to use FP16 mixed precision training and find out if it could help us to reduce the time spent and memory cost during the training process under the premise of reducing numerical decrease for $mIOU$ as much as possible.

#	Element	$mIOU$	mPA	$mPrecision$	$mRecall$	Time
No FP16	Scale	0.90	0.94	0.98	0.94	
	Colour chart	0.85	0.94	0.90	0.94	
	Label	0.83	0.93	0.93	0.93	
	Barcode	0.78	0.92	0.84	0.92	
	Total	0.8742	0.9428	0.9239	0.9428	348min
FP16	Scale	0.92	0.95	0.96	0.95	
	Colour chart	0.88	0.96	0.91	0.96	
	Label	0.90	0.96	0.94	0.96	
	Barcode	0.85	0.94	0.90	0.94	
	Total	0.9065	0.9580	0.9433	0.9580	252min

Table 7. The $mIOU, mPA, mPrecision, mRecall$ result of the sixth experiment.

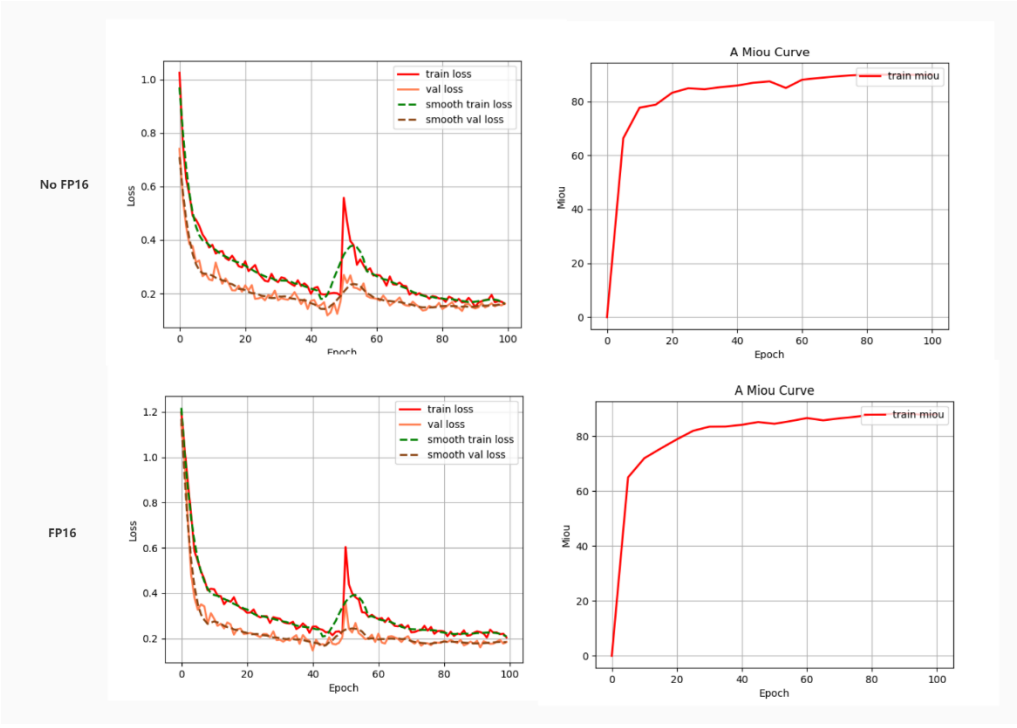


Fig 42. The loss curve and $mIOU$ curve of the sixth experiment.

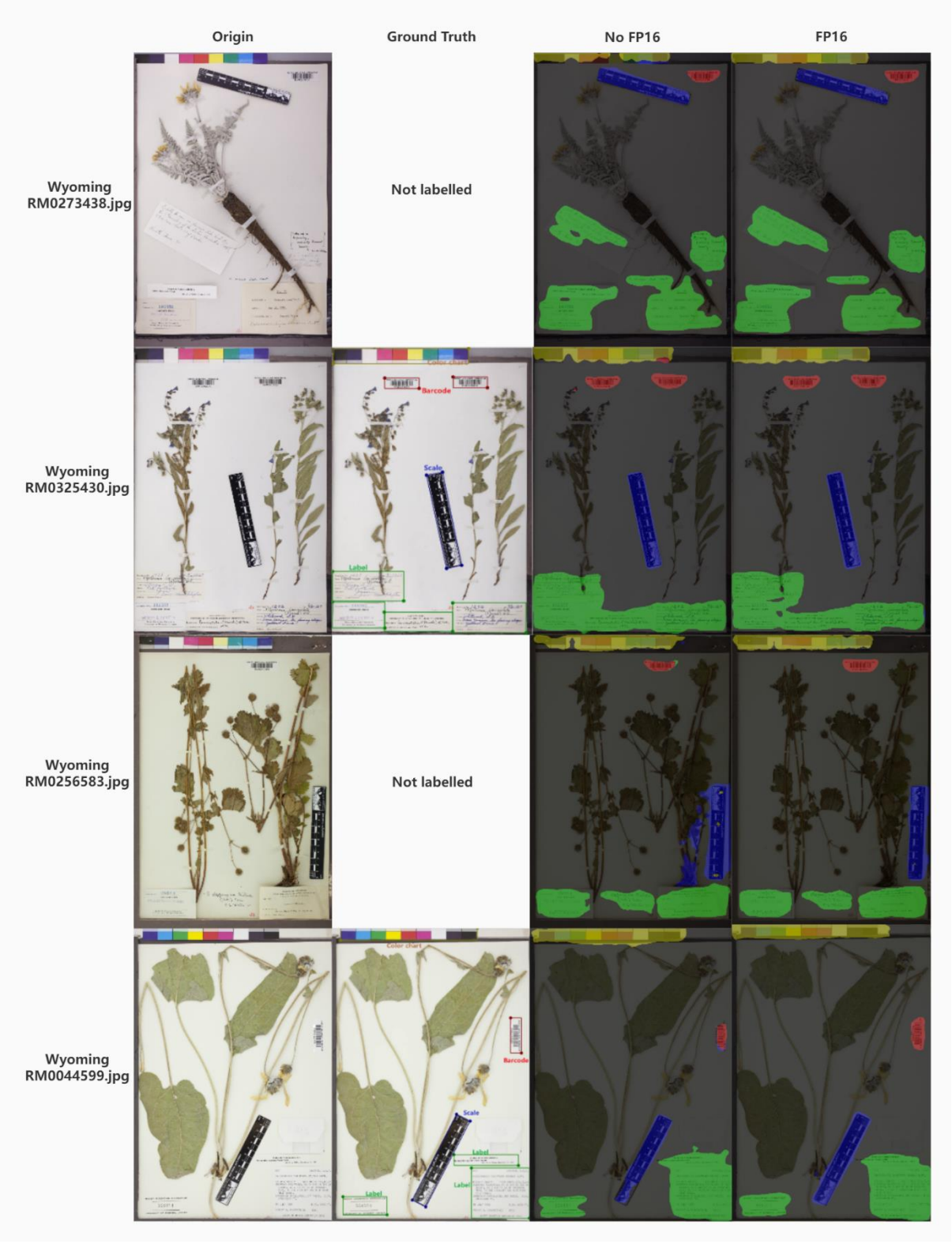


Fig 43. The origin-ground truth-predict for the sixth experiment.

As it shown in Table 7, the train model that enabled FP16 not only cost less time than the model that disable it, but also has higher $mIOU, mPA, mPrecision, mRecall$ values. For our 1000 image training dataset, it could save 96 minutes for training and save 1000MB memory. The runtime temperature for training without FP16 is around $63°C$, however, with FP16 it could cool down to $56°C$. In other words, FP16 could guarantee our aim to develop feasible model for low-level GPU.

In addition, the model with FP16 has more steady loss function when the training process unfreeze on epoch 50 as it shown in Fig 42. And the FP16 model makes less mistake and label more region than No-FP16 model as it shown in Fig 43.

6.5.7 Bonus experiment and analysis

Although we have achieved a stage success on our semantic segmentation model, as it mentioned above in previous section, we would like to perform object detection to help the Wyoming University to detect the potential mistake on classification in their herbarium sheets dataset.

Because the object detection is another category of Computer Vision, it focuses on the bounding box but not the pixel classes label, we manually labelled the Mixed, NHM, MNHM, Wyoming University's raw data and produced a labelled dataset with 500 labelled images (Mixed 200 images, NHM, MNHM, Wyoming 100 images each) by using *Labelimg* library on Anaconda3 environment. And as it shown in Fig 44, we achieved 95% mAP.

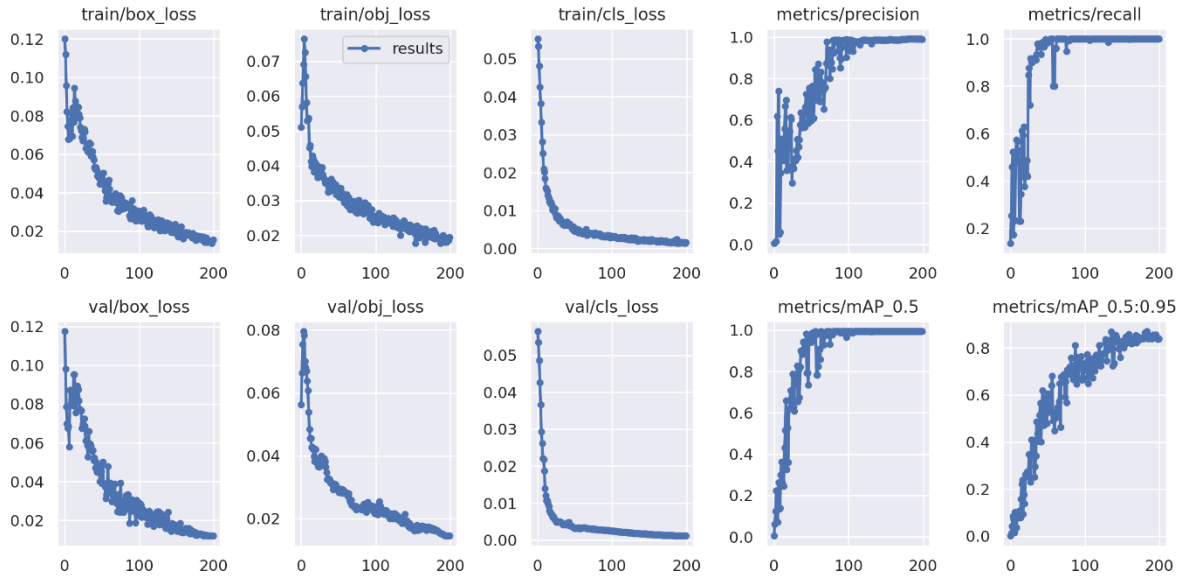


Fig 44. The result of YOLOv5 trained on YOLOv5_x.pt backbone.

Fig 45. The origin-predict result of YOLOv5.

As it shown in Fig 45, our reproduced YOLOv5 model have the ability to identify all the objects inside the herbarium sheets and label them with bounding box. The confidence of each bouding box has a considerable high value, which is 77% as the lowest and 95% as the highest. In other words, our reproduced YOLOv5 model have the ability to identify each Specimen inside the herbarium sheets, and have the potential ability to identify them with different classifications. However, due to we are not familiar with the botany, we do not know the classification of the Specimen, hence further work will required the staffs based on this area to label the Specimen objects with classification.

7. Conclusions

Consider that the student is randomly allocated to this CMT400 project and is new in Deep Learning and Computer Vision and the project was started from September 1, which means the time limitation is less than a month. In this limited time, we learn the related knowledge in this field and provide a successful program. The student has learned the different state-of-the-art models architectures, algorithms and theories, find the differences between them and reproduced them into own code in Pytorch framework. And the student learned the potential methodologies that could improve the performance of the rebuilt model, including FP16, FP32, freeze training, learning rate auto decay, cosine learning rate decay, step learning rate

decay, CE loss, Dice loss, Focal loss, etc. Besides reading papers, journals, books, and tutorials online, the student performed a large volume of experiments to evaluate these knowledges and achieved a stage success for developing the semantic segmentation model and object detection model.

In conclusion, after the experiments, we have achieved a stage success that develop a stable, portable, flexible, resilient, fast in speed and lightweight semantic segmentation model with considerable output results. All the experiments are run on Windows x64 Anaconda3 Pytorch with CUDA environment with NVIDIA GTX 1060-Max-Q GPU for Laptop. And as it shown in the sixth experiment, we achieved $mIOU = 90.65\%, mPA = 95.80\%, mPrecision = 94.33\%, mRecall = 95.80\%$, with $fp16 = True$, $backbone = mobilenetv2$, $optimizer = adam$, $epoch = 100$, $freeze\ epoch = 50$, $freeze\ batch\ size = 8$, $unfreeze\ batch\ size = 4$, $learning\ rate\ decay\ type = cos$, $dice\ loss = True$, $focal\ loss = True$, $num\ workers = 1$. The reproduced DeepLabv3-Plus-Pytorch project has complete all the requirements as it mentioned in Requirement section. Considering that our potential user will be the researchers and museum staffs, we also ensure the less run time, less memory requirement, less GPU requirement, in other words, it is considered to be lightweight and fast in speed and user-friendly tool, which could contribute to accelerate the digitisation workflows for the herbarium sheets collections. It could train 1000 images in around 4 hours with CUDA GPU and predict 500 high quality images in $400ms$ with CPU and output the result that with same size, same quality, same bit-depth of the origin input. Additionally, we help the Wyoming University perform the semantic segmentation on their dataset with the reproduced DeepLabv3-Plus-Pytorch, and provide a bonus tool, which is reproduced YOLOv5-Pytorch, to help them identify the potential mistakes on their classifications. Moreover, our project has the ability to be rebuilt in Flask and export the model as an API into either the frontend or backend to contribute to the online training and online predicting on the natural history collections or other use.

Nevertheless, due to the limitations of time and hardware, there are other methodologies that have not been evaluated in this project, for example, instance segmentation Mask-RCNN, two-stage object detection Fast-RCNN. In the future study, student will work harder, learn more relevant knowledges on Computer Science field and apply them to their work and improve the quality of human life and create a better future for all human beings.

8. Acknowledgements

Thanks to the supervision of Professor Paul L Rosin.

9. References

[1] R. C. Baird, "Leveraging the fullest potential of scientific collections through digitisation"., *Biodiv. Inf.*, vol. 7, no. 2, Oct. 2010. doi: https://doi.org/10.17161/bi.v7i2.3987

[2] Hussein, Burhan & Malik, Owais & Ong, Wee Hong & Slik, Johan. Application of Computer Vision and Machine Learning for Digitized Herbarium Specimens: A Systematic Literature Review. 2021. doi: https://doi.org/10.48550/arxiv.2104.08732

[3] Bebber, Daniel P., et al. "Herbaria Are a Major Frontier for Species Discovery." *Proceedings of the National Academy of Sciences of the United States of America*, vol. 107, no. 51, 2010, pp. 22169–71. *JSTOR*, http://www.jstor.org/stable/25757035. Accessed 11 Sep. 2022.

[4] Meineke, Emily & Davis, Charles & Davies, Jonathan. The unrealized potential of herbaria in global change biology. 2017. doi: https://doi.org/10.1101/218776

[5] Holovachov, O., Zatushevsky, A. and Shydlovsky, I. Whole-Drawer Imaging of Entomological Collections: Benefits, Limitations and Alternative Applications. Journal of Conservation and Museum Studies, 12(1), p.Art. 9. 2014. doi: http://doi.org/10.5334/jcms.1021218

[6] Mantle B, LaSalle J, Fisher N. Whole-drawer imaging for digital management and curation of a large entomological collection. ZooKeys 209: 147-163. 2012. doi: https://doi.org/10.3897/zookeys.209.3169

[7] Guillaume Besnard, Myriam Gaudeul, Sébastien Lavergne, Serge Muller, Germinal Rouhan, Alexander P. Sukhorukov, Alain Vanderpoorten & Florian Jabbour (2018) Herbarium-based science in the twenty-first century, Botany Letters, 165:3-4, 323-327, DOI: 10.1080/23818107.2018.1482783

[8] Hussein, B.R.; Malik, O.A.; Ong, W.-H.; Slik, J.W.F. Automated Extraction of Phenotypic Leaf Traits of Individual Intact Herbarium Leaves from Herbarium Specimen Images Using Deep Learning Based Semantic Segmentation. *Sensors* 2021, *21*, 4549. doi: https://doi.org/10.3390/s21134549

[9] Abdelaziz , Bassem , Walid , A deep learning-based approach for detecting plant organs from digitized herbarium specimen images, Ecological Informatics, Volume 69, 2022, 101590, ISSN 1574-9541, doi: https://doi.org/10.1016/j.ecoinf.2022.101590.
[10] Abdelaziz Triki, Bassem Bouaziz, Jitendra Gaikwad, Walid Mahdi, Deep leaf: Mask R-CNN based leaf detection and segmentation from digitized herbarium specimen images, Pattern Recognition Letters, Volume 150, 2021, Pages 76-83, ISSN 0167-8655, doi: https://doi.org/10.1016/j.patrec.2021.07.003.

[11] Nieva de la Hidalga, A., Owen, D., Spasic, I., Rosin, P., Sun, X, Use of Semantic Segmentation for Increasing the Throughput of Digitization Workflows for Natural History Collections. Biodiversity Information Science and Standards 3, e37161, 2019. doi: https://

doi.org/10.3897/biss.3.37161

[12] de la Hidalga, A.N., Rosin, P.L., Sun, X. *et al.* Cross-validation of a semantic segmentation network for natural history collection specimens. *Machine Vision and Applications* 33, 39 (2022). doi: https://doi.org/10.1007/s00138-022-01276-z

[13] Dillen M, Groom Q, Chagnoux S, Güntsch A, Hardisty A, Haston E, Livermore L, Runnel V, Schulman L, Willemse L, Wu Z, Phillips S (2019) A benchmark dataset of herbarium specimen images with label data. Biodiversity Data Journal 7: e31817. doi: https://doi.org/10.3897/BDJ.7.e31817

[14] Chen, L. C., Papandreou, G., Schroff, F., & Adam, H. Rethinking atrous convolution for semantic image segmentation. 2017. doi: https://doi.org/10.48550/arxiv.1706.05587
[15] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. doi: https://doi.org/10.48550/arxiv.1606.00915
[16] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. doi: https://arxiv.org/abs/1606.00915
[17] Marina Paolanti, Emanuele Frontoni, Multidisciplinary Pattern Recognition applications: A review, Computer Science Review, Volume 37, 2020, 100276, ISSN 1574-0137,

doi: https://doi.org/10.1016/j.cosrev.2020.100276.

[18] Sanur, Bali, Neural Information Processing: 28th International Conference, ICONIP 2021, December 8–12, 2021, Proceedings, Part I Dec 2021 pp. 402 doi: https://doi.org/10.1007/978-3-030-92185-9_33

[19] Triki, A.; Bouaziz, B.; Mahdi, W. and Gaikwad, J. Objects Detection from Digitized Herbarium Specimen based on Improved YOLO V3.In Proceedings of the 15th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 4: VISAPP, ISBN 978–989–758–402–2, pages 523–529. 2020. doi: https://doi.org/10.5220/0009170005230529

[20] Ronneberger, O., Fischer, P., Brox, T. U-Net: Convolutional Networks for Biomedical Image Segmentation. In: Navab, N., Hornegger, J., Wells, W., Frangi, A. (eds) Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015. MICCAI 2015. Lecture Notes in Computer Science(), vol 9351. Springer, Cham. 2015. doi: https://doi.org/10.1007/978-3-319-24574-4_28

[21] H. Zhao, J. Shi, X. Qi, X. Wang and J. Jia, "Pyramid Scene Parsing Network," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 6230-6239, doi: 10.1109/CVPR.2017.660.

[22] Sandler, Mark & Howard, Andrew & Zhu, Menglong & Zhmoginov, Andrey & Chen, Liang-Chieh. MobileNetV2: Inverted Residuals and Linear Bottlenecks. 2018. doi: https://doi.org/10.48550/arxiv.1801.04381

[23] Irem Ulku & Erdem Akagündüz, A Survey on Deep Learning-based Architectures for Semantic Segmentation on 2D Images, Applied Artificial Intelligence, 36:1, 2022, doi: https://doi.org/10.1080/08839514.2022.2032924

[24] Sudre, C.H., Li, W., Vercauteren, T., Ourselin, S., Jorge Cardoso, M. Generalised Dice Overlap as a Deep Learning Loss Function for Highly Unbalanced Segmentations. In: , et al. Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support . DLMIA ML-CDS 2017, 2017. Lecture Notes in Computer Science(), vol 10553. Springer, Cham. doi: https://arxiv.org/abs/1707.03237

[25] T. -Y. Lin, P. Goyal, R. Girshick, K. He and P. Dollár, "Focal Loss for Dense Object Detection," 2017 IEEE International Conference on Computer Vision (ICCV), 2017, pp. 2999-3007, doi: https://arxiv.org/abs/1708.02002

[26] Michael Yeung, Evis Sala, Carola-Bibiane Schönlieb, Leonardo Rundo, Unified Focal loss: Generalising Dice and cross entropy-based losses to handle class imbalanced medical image segmentation, Computerized Medical Imaging and Graphics, Volume 95, 2022, 102026, ISSN 0895-6111, doi: https://arxiv.org/abs/2102.04525

[27] GitHub https://github.com/NaturalHistoryMuseum/semantic-segmentation Accessed on: 31 Aug, 2022.

[28] P. Kr¨ahenb ¨ uhl and V. Koltun, “Efficient Inference in Fully Connected CRFs with Gaussian Edge Potentials,” in NIPS, 2011. doi: https://doi.org/10.48550/arxiv.1210.5644

[29] He, K., Zhang, X., Ren, S., Sun, J. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds) Computer Vision – ECCV 2014. ECCV 2014. Lecture Notes in Computer Science, vol 8691. Springer, Cham. 2014, doi: https://doi.org/10.1007/978-3-319-10578-9_23

[30] A. Harouni, A. Karargyris, M. Negahdar, D. Beymer and T. Syeda-Mahmood, "Universal multi-modal deep network for classification and segmentation of medical images," 2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018), 2018, pp. 872-876, doi: 10.1109/ISBI.2018.8363710.

[31] He, Kaiming, X. Zhang, Shaoqing Ren and Jian Sun. “Deep Residual Learning for Image Recognition.” 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016): 770-778. doi: https://doi.org/10.48550/arxiv.1512.03385

[32] JSTOR (2018). JSTOR Global Plants: Guidelines for Scanning Specimens. From: https://guides.jstor.org/ld.php?content_id=31764146

[33] JSTOR (2018). JSTOR Plants Handbook. From http://www.snsb.info/SNSBInfoOpenWiki/attach/Attachments/JSTOR-Plants-Handbook.pdf

[34] Micikevicius, Paulius & Narang, Sharan & Alben, Jonah & Diamos, Gregory & Elsen, Erich & Garcia, David & Ginsburg, Boris & Houston, Michael & Kuchaev, Oleksii & Venkatesh, Ganesh & Wu, Hao. (2017). Mixed Precision Training. DOI: https://doi.org/10.48550/arxiv.1710.03740