Report of internship TN10

Construction of 3D forest maps by on-board omnidirectional vision

ZHOU Dao

Encadrants : Pascal Vasseur

Cédric Demonceaux

Suiveur UTC : Luis Alejandro OSPINA VARGAS

Sommaire

1. Presentation of project and group

This internship is a part of the project ANR CLARA, whose context will be presented below.

Unmanned aerial vehicles (UAVs), commonly known as drones, are robots now highly used for many applications like monitoring, prevention or site surveillance. In the case of an obstacle-free environment, autonomous navigation can be considered by controlling a vehicle along GPS way-points. For more complex situations such as monitoring engineering structures, a pilot is often necessary to ensure safety during the flight. In even more complex environments such as in a forest, both autonomous and human piloting become impossible due to many constraints :

- The loss of GPS signals is classic in this kind of difficult environment making way-point very complicated if not impossible,

- Dense and unstructured environments (branches, foliage, ...) reduce visibility.

Nevertheless, the use of aerial drones in this type of environment is of great interest and can be broken down into multiple prevention and / or surveillance applications such as fire detection, search for missing people, mapping for forest maintenance, etc.

In this project, we propose to tackle the challenge of autonomous navigation of an UAV within a forest with the added objective of 3D mapping. We then distinguish three scientific hurdles to solve:

- The control of the drone in a complex and unknown environment;

- The perception and representation of a complex and unstructured environment for obstacle avoidance and the study of traversability;

- 3D reconstruction and localization of a flying robot in an environment where objects are not very discriminate / differentiable.

In the CLARA project, to respond to the identified scientific constraints and obstacles, we will assume that the drone does not have a map of the environment or does not receive. Consequently, it has to rebuild a simple map of the environment while moving autonomously towards a predefined position. For this, the drone will be equipped with an inertial unit and a stereoscopic vision system consisting of two 360-degree cameras.

This project is leaded by Guillaume Allibert, the project coordinator. LE2I(Laboratoire d'Electronique, d'Informatique et d'Image), LITIS(Laboratoire d'Informatique, de Traitement de L'Information et des Systèmes) and I3S(Laboratoire informatique, signaux systèmes de Sophia Antipolis) participate in the study of the project.

As an intern, I do my job at school ESIEE in Amiens because my teacher in charge works here this semester, but the principal group is not located in Amiens. To communicate and share the work with each other, we conduct meeting by Skype from time to time.

2. Introduction of the internship

The ANR CLARA project aims to develop an autonomous aerial drone capable of navigating in a forest environment without GPS, based on an omnidirectional stereovision system. Among the various functionalities to be developed, creating a map for location and navigation is a major challenge. Indeed, the absence of a regular and ordered geometric structure, frequent in urban or interior environment, implies using primitives different from the classic SLAM approaches.

Here we plan to use the layout of the trees (their position on the ground) and their intrinsic characteristics (diameter, inclination, ...) to estimate a 3D map which can be used subsequently to locate.

In this internship, the objective is to develop a functional module capable of detecting trees in a sequence of omnidirectional RGB images (possibly stereo), of characterizing them (dimensions, positions, shapes, ...) and of integrating them into a 3D map. To achieve this goal, the work will be divided into the following stages:

- study of the state of the art on navigation in a forest environment and its cartography
- study on the detection and identification of trees in image sequences
- development of a tree and soil recognition module based on a learning approach
- development of a 3D characterization module for soil and trees
- precise construction of a 3D map

The objective is to combine techniques related to learning and geometry to create a fast and robust tool allowing the obtaining of an accurate and useful map for location and navigation.

3. State of the art about the detection of trees and the navigation in forest

In the past, UAVs mainly relied on INS (inertial navigation systems) and GPS for navigation. However, inertial devices have accumulated errors during navigation and are too sensitive to initial values. GPS is not always available, and even if it is available, its accuracy often cannot meet the needs of navigation. The development of image processing technology and hardware such as camera allows computer vision technology to be introduced into the navigation. There are several advantages. Firstly, the real-time information provided by vision can be fused with inertial navigation and GPS information to improve accuracy of navigation. Secondly, cameras are good at capturing motion information, while traditional sensors are less sensitive to such information. Thirdly, the ability of anti-interference of visual signals is powerful, while radio and GPS signals are easily blocked. Based on these advantages, vision is more and more applied in the navigation.

The navigation of UAV in the forest is very similar to the navigation of autonomous vehicle on the road. At present, there are many studies on the perception of autonomous vehicles in the surrounding environment, such as detecting other vehicles, detecting pedestrians and lane lines, scene understanding, image segmentation, etc. But there are relatively few studies on drones like detecting trees and navigating in the forest. Most of studies deal with aerial images taken by drones, which does not meet our need (our project aims to allow drones to navigate in the forest. Therefore, the drone takes front images). Nvidia has conducted research on the navigation of drone in the forest [1], but it is based on detecting trails in the forest and making the drone navigate in the middle of the trail.

There are very few researches on the detection of intact trees based on deep learning methods. After searching, we found that there are currently researches using more traditional methods to detect tree trunks. Some work uses colour features and texture features to segment the tree trunk area [2]. Some uses multi features such as colour feature and HOG feature and SVM classifier to detect tree trunks [3]. Other work uses geometric features to detect tree trunk areas in consideration of the characteristics of the pictures taken by the camera [4]. Another work of thesis first distinguishes the area that may be the trunk according to the colour feature, then extracts the edge of the trunk, and finally merges the two edges of the same trunk [5].

Considered there been great progress in processing image by using deep learning methods. In the first place, we hoped to apply the deep learning method to our research.

4. State of the art of Image recognition

To get familiar with our work, we did some research on the state of the art of image recognition, which contains three main categories:
-- Object Detection with bounding box
-- Semantic segmentation
-- Instance segmentation

4.1 Object Detection with bounding box

Object detection means that we must not only use algorithms to determine the category of the target in the image, but also mark its position in the image, usually surrounded by a rectangular bounding box. In recent years, researchers have made great breakthroughs on the algorithms of object detection. The most popular algorithms can be divided into two categories. One is the algorithm like R-CNN based on Region Proposal (R-CNN, Fast R-CNN, Faster R-CNN, etc.). They are two-stage algorithms which need to generate region proposal first, that is the possible location of target, then classify the object in region proposal and regress the position of region proposal. The other are one-stage algorithms such as Yolo and SSD, which use only one convolutional neural network CNN to directly predict the categories and positions of different targets. The first type of algorithms are more accurate but slower, the second type of algorithms are faster, but the accuracy is lower. The picture below shows their performances.

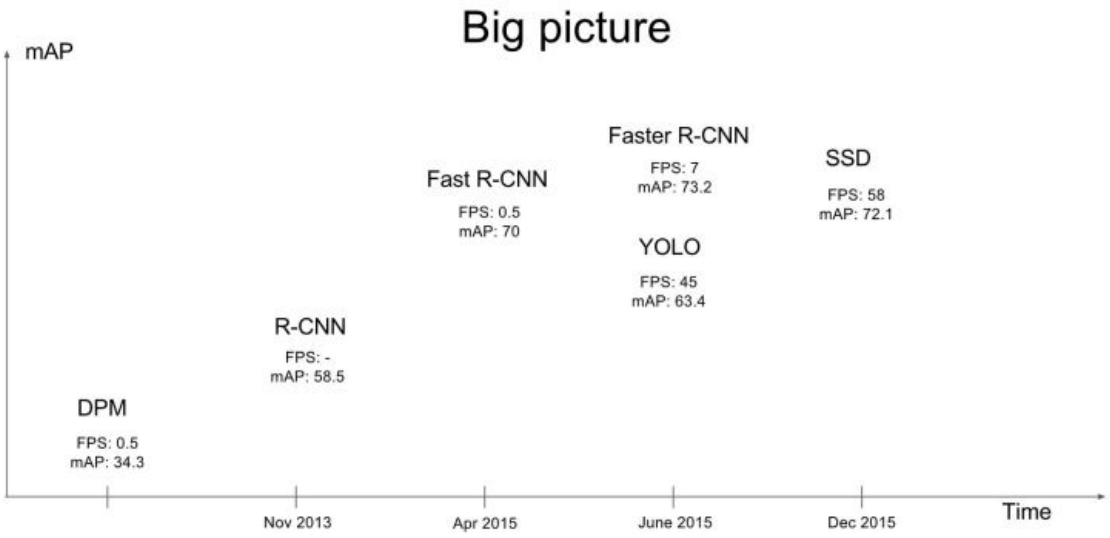


Figure 1 – Performance of the algorithms of object detection

The one-stage algorithms like R-CNN draw on the idea of sliding window and adopt the method of region recognition, specifically:

-- Extract 2000 independent candidate regions (possible regions that contain targets) from image through specified algorithm

-- For each candidate region, use a convolutional neural network to obtain a feature vector

-- For the corresponding feature vector of each region, use support vector machine SVM for classification, and adjust the size of bounding box through the method of regression

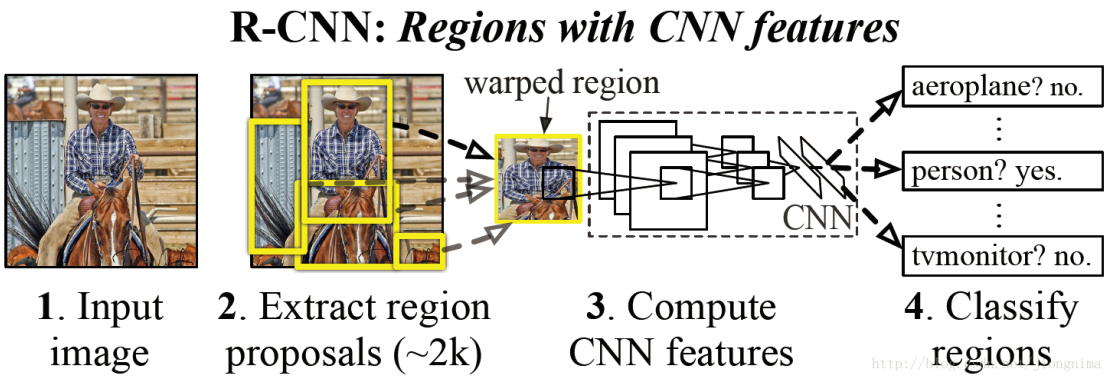


Figure 2 – Illustration of R-CNN

Considering the slow speed of the two-stage algorithms, the one-stage algorithms like Yolo creatively complete the classification and the positioning in one step, which directly regress the position of bounding box and the category of the object in bounding box in the output layer of the network. In this way, Yolo can achieve a opretion speed of 45 frames per second, which can fully meet the real-time requirement (at least 24 frames per second, human feels continuous). The whole system is shown in the figure below.

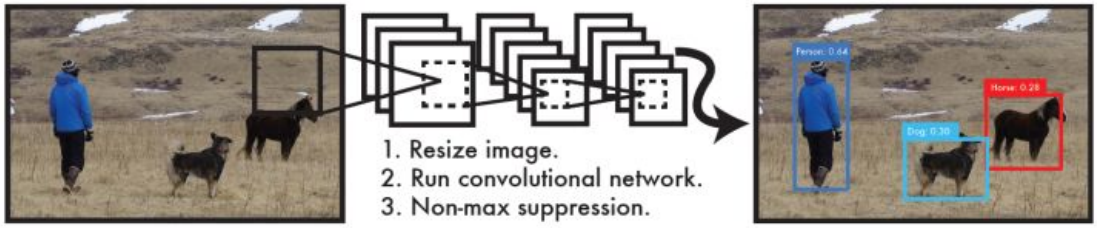


Figure 3 – Illustration of one-stage algorithm

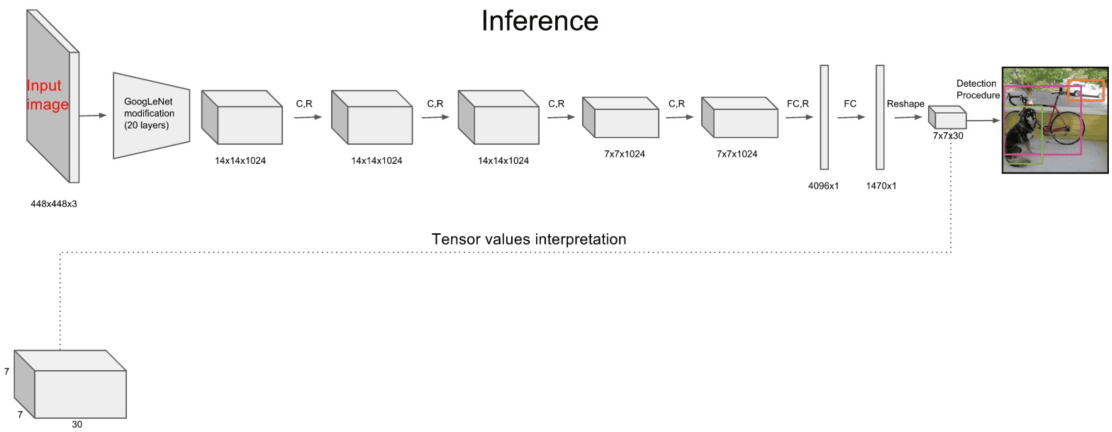


Figure 4 – Structure of the network Yolo

4.2 Semantic segmentation

Semantic segmentation is a classification at pixel level. Pixels that belong to the same category should be classified into one category. Therefore, semantic segmentation understands images from pixel level. For example, in the following pictures, pixels belonging to person should be classified into one category, pixels belonging to motorcycle should be classified into one category, and in addition, pixels of background should also be classified into one category. Note that semantic segmentation is different from instance segmentation. For example, if there are multiple people in a photo, for semantic segmentation, as long as all the pixels of people are classified into one category, but instance segmentation also needs to separate pixels of different people into different categories.

Figure 5 – Illustration of semantic segmentation

The methods of deep learning to solve the problem of semantic segmentation can be summarized into several ideas:

1.Patch classification

The original method of deep learning applied to semantic segmentation is patch classification. Patch classification, as its name suggests, the image is cut into blocks and fed to the deep neural network, and then pixels are classified. The main reason of using blocks is that fully connected layers require fixed-size images.

2. Fully convolution network

In 2014, fully convolutional network (FCN) was born, which replaces the fully connected layer by the convolutional layer and makes possible for processing images of any size, in addition its speed is much faster than Patch classification.

Despite the removal of the fully connected layer, there is still a problem of CNN model for semantic segmentation, that is the operation of downsampling (for example, pooling). The operation of pooling can expand the receptive field and thus can integrate contextual information well. This is very effective for high-level tasks (such as classification). But at the same time, due to the operation of pooling, the resolution is reduced, so the information of position is weakened. However, semantic segmentation needs rich information of position.

3.encoder-decoder architecture

The encoder-decoder architecture is based on the FCN. The encoder gradually reduces the spatial dimension by pooling, while the decoder gradually recovers the spatial dimension and detailed information.

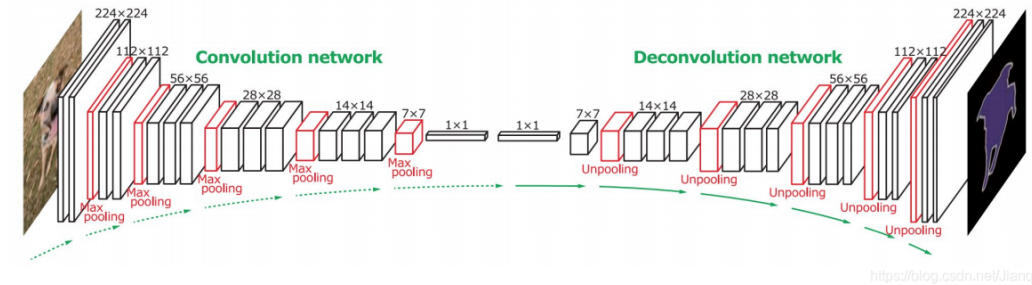


Figure 6 – Structure of FCN

Usually there is shortcut connection from encoder to decoder (that is cross-layer connection). Among them, U-net is a very popular network of this architecture, as shown below:

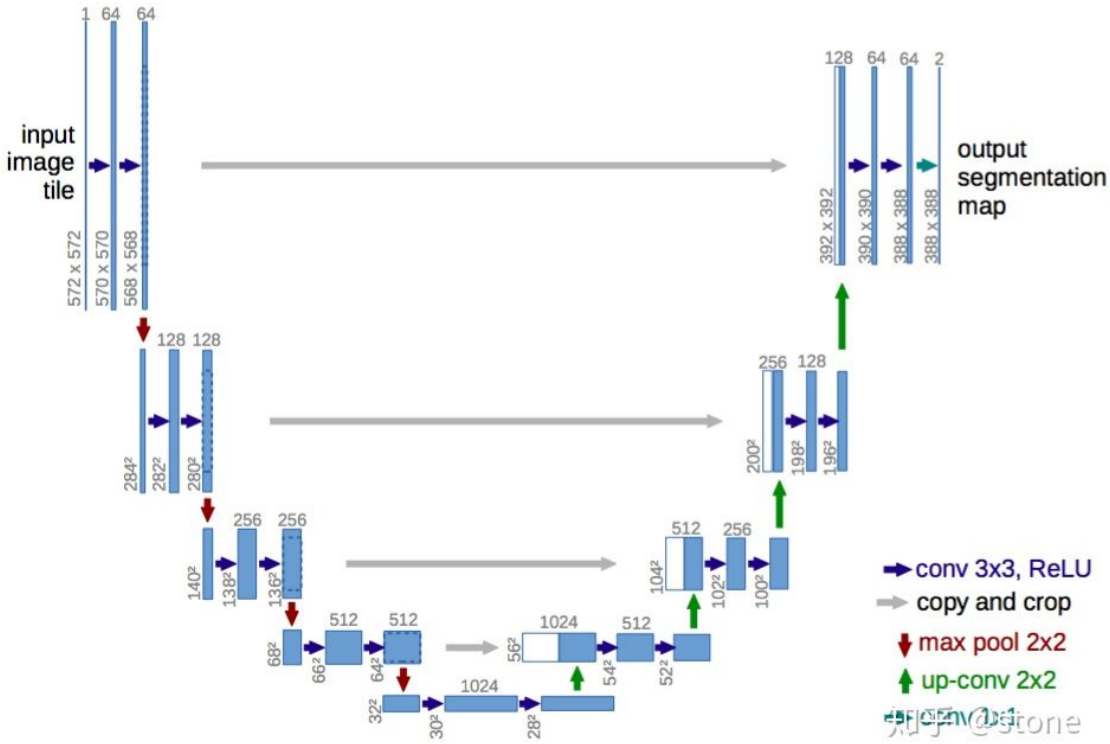


Figure 7 – Structure of U-Net

4. Hollow Convolution

The dilated/atrous (hollow convolution) architecture replaces the operation of pooling. On the one hand it can maintain the spatial resolution, on the other hand it can expand the receptive field and integrate contextual information well. As shown below:

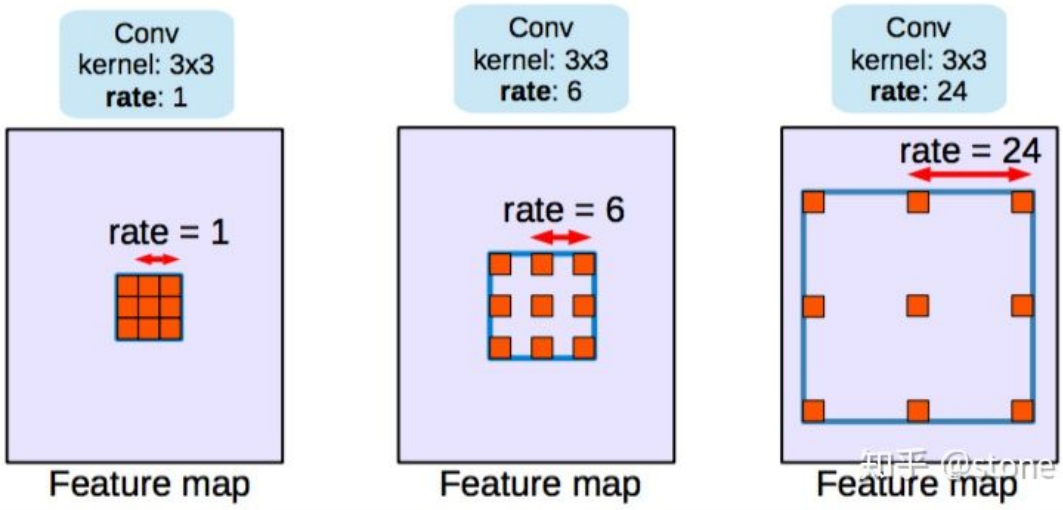


Figure 8 – Illustration of hollow convolution

5. Conditional random field

In addition to the above ideas, there is a method of post-processing the result of segmentation, that is Conditional Random Fields (CRFs). It uses the related information between pixels : adjacent pixels or pixels with similar colors are more likely to belong to the same class. The series articles of DeepLab basically adopt this method and show that it can obviously improve the result of segmentation.

4.3 Instance segmentation

On the basis of semantic segmentation, instance segmentation also needs to distinguish different individuals of the same type. The figure below shows the difference of all methods of image recognition.

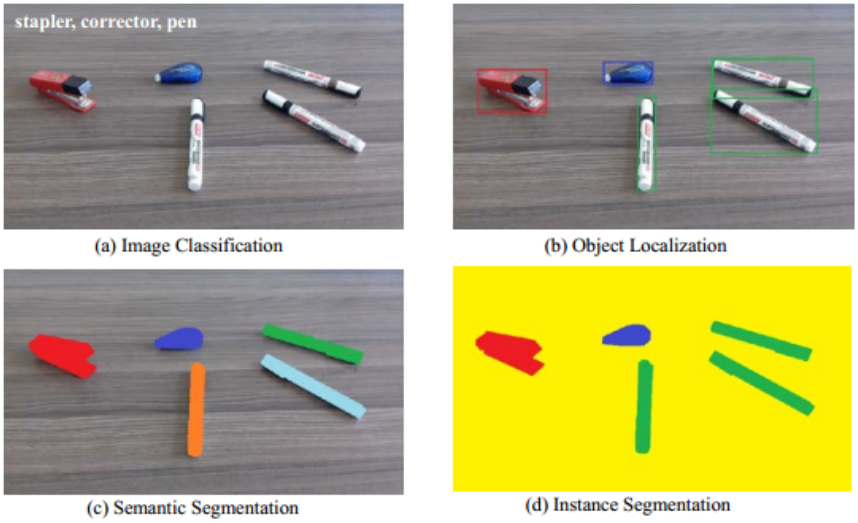


Figure 1. Object recognition evolution: from coarse- to fine-grained inference: (a) Image Classification, (b) Object detection or localization, (c) semantic segmentation, (d) Instance segmentation.

Figure 9 – Difference between methods of image recognition

There are some problems and difficulties in instance segmentation.

-- segmenting small objects. Deep neural networks generally have larger receptive fields and are more robust to pose, deformation, lighting, etc, but at the same time the resolution is lower and details are lost. Shallow neural networks have narrower receptive fields, richer details and larger resolution, but it lacks semantic information. Therefore, if an object is relatively small, its details will be less in the shallow CNN layer, and the same details will almost disappear in the deep network. The methods to solve this problem are dilated convolution and increasing the resolution of features.

-- Problem of geometric transformation. For geometric transformation, CNN is not inherently spatially invariant.

-- Problem of occlusions. Occlusion will cause the loss of the information of target. Some methods have been proposed to solve this problem, such as deformable ROI pooling, deformable convolution and adversarial network. In addition, GAN may also be used to solve this problem.

The research of instance segmentation has been divided into two categories for a long time, namely the bottom-up semantic segmentation-based method and the top-down detection-based method.

Top-down instance segmentation method

The idea is to first find out the bounding box of the instance through the method of object detection, and then perform semantic segmentation in each detection box, finally each result of segmentation is output as a different instance, a representative work is Mask RCNN.

The algorithm firstly realizing top-down dense instance segmentation is DeepMask, which uses the method of sliding window to predict a mask proposal on each spatial region. This method has the following three disadvantages:

-- The connection between the mask and the feature (local consistency) is lost

-- The representation of feature extraction is redundant

-- The information of position is lost caused by downsampling

Bottom-up instance segmentation method

The idea is to first perform pixel-level semantic segmentation, and then use clustering, metric learning and other means to distinguish different instances. Although this method maintains better low-level features (information of detail and location), it also has the following disadvantages:

-- High quality requirements for dense segmentation will lead to sub-optimal segmentation

-- The ability of generalization is poor, the method is unable to deal with complex scenarios with many categories

-- The post-processing steps are trivial

A method of another type, Single Shot Instance Segmentation, is inspired by the research of one-stage object detection. There are also two ideas, one is inspired by one-stage, anchor-based detection models such as YOLO and RetinaNet , Representative works include YOLACT and SOLO, the other is inspired by anchor-free detection models such as FCOS, representative works include PolarMask and AdaptIS.

5. Practice of object detection

5.1 Preparation of dataset

To use the method of deep learning to practice image recognition , a training dataset is essential. However, after online search, there is almost no dedicated dataset for tree detection. Some commonly used datasets such as Pascal VOC, COCO, KITTI, GOOGLE Image are all generally very large datasets, containing too many categories. They may contain the images of trees, but it is not sufficient and not in forest environment, which is not suitable for our research, so we decided to make our own datasets. I collected more than 200 images of forests and trees on GOOGLE (as shown below):

Figure 10 – Training images collected online

The images of forest are all taken from the front in real forests, not aerial images. Although they are not in the same forest, the types of trees are also very different, but these are indeed roughly standard images that we can find ourselves. Most of the images of trees contain one or several whole trees. The selection of these images is mainly to make our training network more robust, more applicable, and able to identify various trees from different perspectives.

To generate corresponding annotation of images, I used a tool of annotation LabelImg to label these images. LabelImg is a simple and easy-to-use image labeling tool. It allows users to just draw a rectangular box surrounding each target on the image and to give it a category and then to choose to generate annotation files in Pascal VOC or Yolo format. The Yolo format is simple and intuitive. It is represented in a text file, in which each line records the category and the location of a target by five numbers (the category of the target, the center x,y coordinates of the bounding box and its width and height). The Pascal VOC format, represented in xml file, is more complex and contains more information of targets and the image.

The work of annotation is boring and repetitive, but there are some standards and requirements. Some rules of Pascal VOC dataset are :

1. The rectangular bounding box should not be too small, at least more than 20 pixels.

2. For blurred, obscured and smeared targets, mark them as difficult samples. According to the needs of the algorithm, decide whether to enable these samples to participate in training.

3. Every object which can be distinguished by human need to be annotated, which cannot be distinguished by human don’t need to be annotated.

4.After drawing the rectangle, pay attention to check its boundary to ensure that the coordinates of bounding box are not on the image boundary.

During the work of annotation, I found it very difficult to label each tree successfully. First of all, the number of trees in the image is very large, and the types of trees are different in different images. Some tall trees can only be seen their trunks in the image, which can be surrounded by rectangular approximately. However, stumpy trees show the branches, which makes the surrounding rectangular bounding box very large, so that the bounding box may contain other interfering rear objects or trees, which undoubtedly makes the detection difficult. In addition, the overlap and occlusion between the trees and the blurred trees in the distance also made it difficult for me to choose the standard of annotation. In principle, all objects that can be distinguished by human eyes need to be annotated, but it is very difficult to actually carry out.

The similar work of annotating vehicles or people on the road in automatic driving is more simple because the shape of the vehicle/person can be well surrounded by a rectangular bounding box, different vehicles/people are easy to

distinguish and the number of vehicles/people in each image is limited. However, in our case, as the difficulties mentioned above,it is not too realistic to achieve a perfect annotation.

In the first round of annotation, my principle was to try to annotate every tree that can be distinguished easily. For tall trees in the forest, I annotated the visible part of trunk, and for the images containing whole tree, I annotated the whole tree. For some trees with severe overlapping branches and leaves, only the distinguishable part of trunk was annotated. For distant blurred trees, I ignored them. All the annotations are defined as the same category tree. An example is shown below.

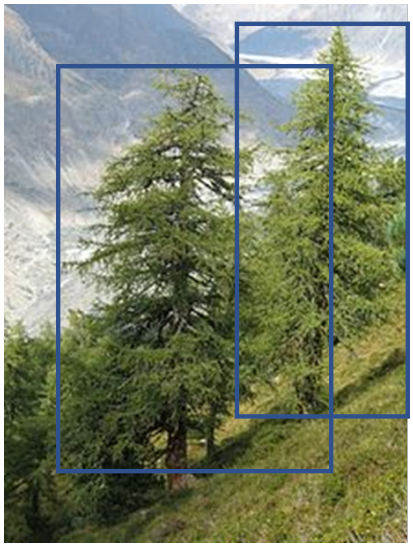


Figure 11 – Example of annotation

Similarly, I also selected 50 images of forest and trees for testing.

5.2 Select of the network and preparation of the training environment

After the annotation, it is important to choose a appropriate neural network to perform training. As I presented above, The one-stage methods of object detection appeared late and as the research progressed , achieved the comparable rate of accuracy to two-stage methods on some datasets. Its representative algorithm Yolo has undergone three generations of continuous improvement, resulting in Yolo v1, v2, v3. I chose to try Yolo v3 after the comparison.

The general idea of Yolo is to directly regress the position of the bounding box and the category of the target in the bounding box in the output layer, so as to realize one-stage. The Yolo algorithm has open source code on github. Its backbone

network, darknet, is a light-weight, open-source deep learning framework based entirely on language C and package CUDA. It supports two methods of calculation, CPU and GPU, and is very convenient for fine-tuning and training on our own dataset. An advantage of Yolo v3 compared to Yolo v1 and Yolo v2 is that it predicts bounding box at different scales, so it achieves better performance for detecting small objects. The general architecture of Yolo v3 network is shown below.

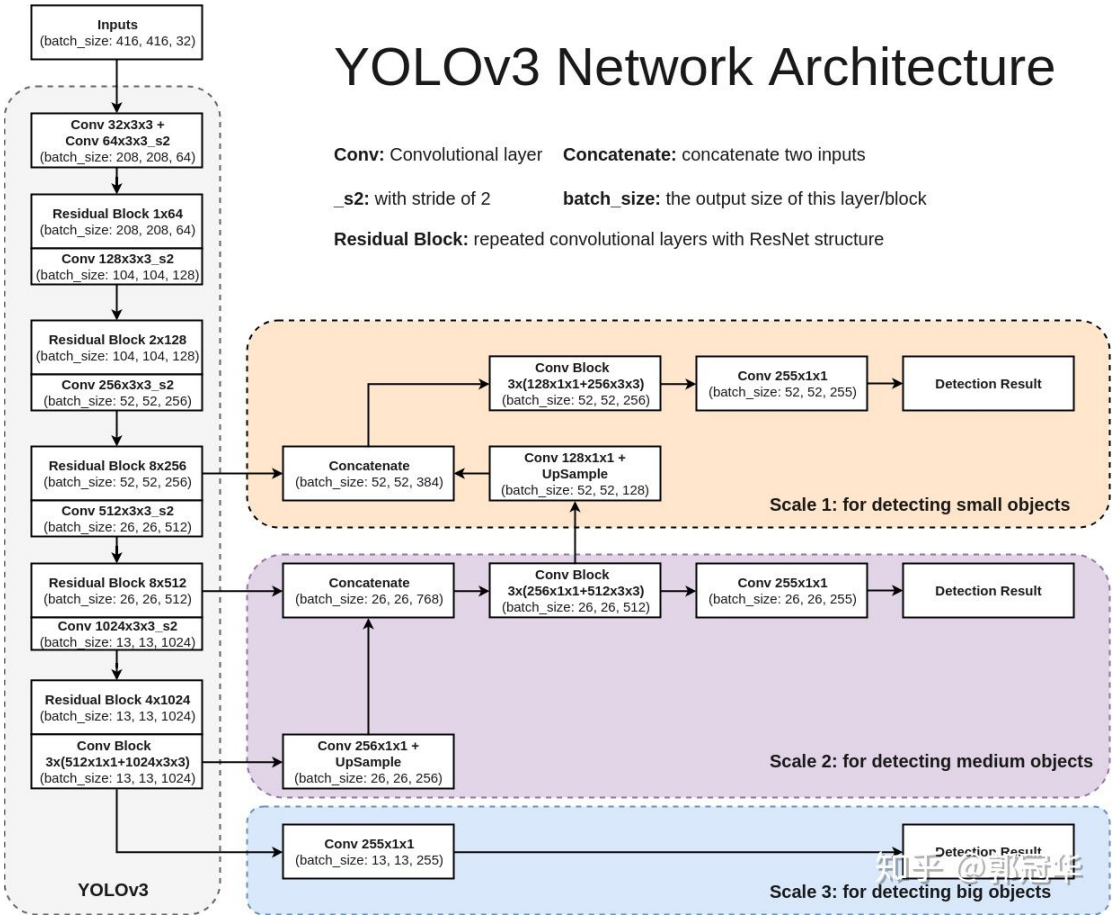


Figure 12 – Architecture of Yolo

Before training, we need to download the framework Darknet of Yolo and its dependent vs2017 development environment and package CUDA. The downloaded framework already contains some configuration files of the PASCAL VOC training dataset, which define the paths of the training dataset and testing dataset, the names of categories, and provide architectural guidelines for the neural network used. We need to modify them slightly to apply for our own training dataset.

First I made sure that the annotation file has the same name as its corresponding image, but obviously has a different extension. All the images and the annotation files should be located in the same directory.

Second I created a txt file containing the path of all the training images, each line of which points to an image. Then I modified two configuration files, their suffixes are .name and .data, the .name file defines the names of the categories, the .data file defines the number of categories, the path of the txt file mentioned above that contains the path of the training images, the path of the .name file and the directory under which the training weights of the network will be saved (as shown below).

classes = 2

train = data/train2.txt

valid = data/test2.txt

names = data/obj.names

backup = backup2/

Then I modified the configuration file yolo.cfg of the neural network, which defines the structure and training parameters of the neural network such as the size of batch, subdivisions, the width and height of the image, the learning rate, the learning decay rate and so on. The most important modification is to change the value of classes in each block yolo to the number of our own training categories.

```
779
780 [yolo]
781 mask = 0,1,2
782 anchors = 10,13,  16,30,  33,23,
783 classes=4
784 num=9
785 jitter=.3
786 ignore_thresh = .7
787 truth_thresh = 1
```

Figure 13 – Modification of configuration file

The value of filters in the block convolutional was changed to filters==(classes + 5)x3. Since there is only one class in our case, filters=18.

```
771
772 [convolutional]
773 size=1
774 stride=1
775 pad=1
776 filters=27
777 activation=linear
778
779
780 [yolo]
781 mask = 0,1,2
782 anchors = 10,13,  16,30,  33,23,  30,6
783 classes=4
784 num=9
785 jitter=.3
786 ignore_thresh = .7
```

Figure 14 – Modification of configuration file

According to the configuration of my computer, GPU training could be used. Since the configuration of my PC is not very high, if the value of training batch was set too large, the memory would not be enough, so I set the value of batch to 4, and the value of subdivisons to 2, which means in each round there would be 2 images entering the network and participating in the training.

Then I started training the neural network. After two days, the training with 11,000 rounds had been completed. The average loss of the network dropped below 1 and gradually converged, that means the ability of generalization of the neural network may have reached the upper limit, so I decided to test the results of the prediction on the testing dataset.

The testing results were not very satisfactory. For some obvious and upright trunks, the network can still predict some of them accurately, but not all of them. Although some trunks look similar, the network may only choose a few of them. For the images with complex environment containing multiple whole trees, the prediction effect of the network is not good, either there is no predicted bounding box, or there are too many repeated predicted bounding boxes. An example of a good prediction result is shown below.

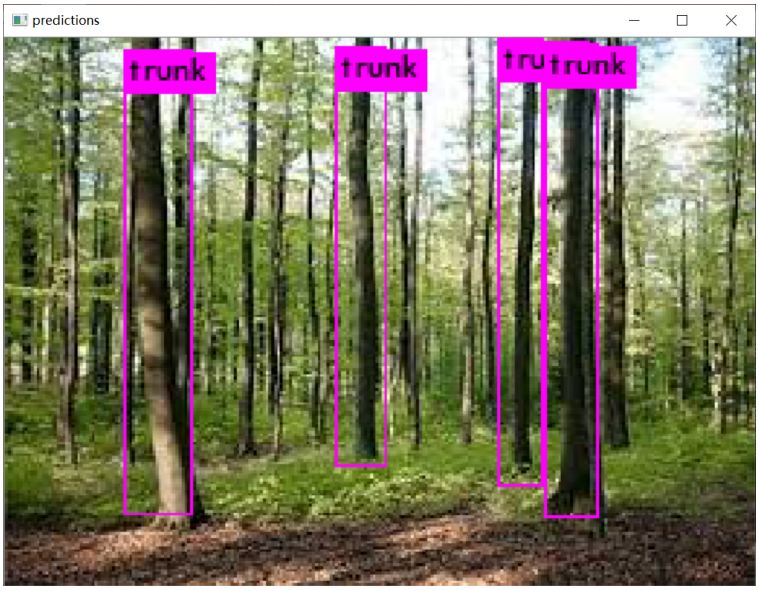


Figure 15 – Example of prediction result

After analysis and discussion, we believed that the trunk and the whole tree should not be annotated as the same category. If we do so, as the whole tree must contain the trunk, the network would be confused when facing an image containing a whole tree. Should the prediction bounding box choose the trunk or the whole tree? Therefore, I decided to change the method of annotation.

5.3 Change of the annotation and retraining of the network

Considering a whole tree mostly contains two parts, trunk and crown, I specified these two as the categories of annotation and mainly modified the annotations of the images containing whole trees. However, there were also problems by using this method of annotation. For some trees with unclear crowns or trees with too large crowns that cause rectangular bounding box too large, the annotation was still difficult.

After completing the annotation, I modified the configuration files, retrained the network and did the test. The testing results showed that the network has made progress in the prediction of images containing a single whole tree, and can more accurately predict the trunk and the crown, but there are also wrong predictions. The improvement of the prediction on forest images with multiple trunks was not obvious, only a part of them could be predicted. The prediction results on complex images are either not satisfactory.

We seriously thought about the reason of the problem, and concluded that it still lied in the annotation of the images. According to the principle of the training of network, inaccurate annotations (some targets with annotations, some without annotations) can seriously mislead the network, because during the training process the network not only learns the features of the target, but also knows those features of areas that do not contain targets should not be concerned. As with the problems mentioned above, it is unrealistic to annotate all trees. Therefore it is logical that the network could not predict all trees correctly.

6. Practice of existing models of semantic segmentation

Considering annotating the images was not a simple task, we decided to temporarily abandon object detection and to experiment with semantic segmentation, since semantic segmentation is a prediction at pixel-level and does not involve the problem of predicting a single target, so it may be more suitable for our research.

We first hoped to test the prediction results of the currently existing network models of semantic segmentation on the images of forest and tree. Although these models were trained on general datasets, it is worth trying to test.

After having searched and compared existing networks, we chose the network AdapNet, which is easily trainable on a single GPU with 12 GB of memory and has a fast inference time. It has a Tensorflow implementation, so it is convenient to practice. Similar to many other semantic segmentation networks, it is also based on

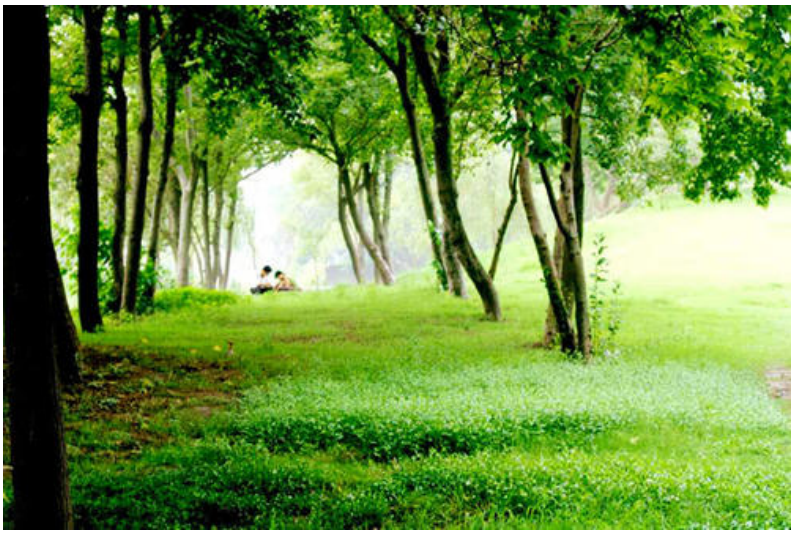
FCN, its backbone network is ResNet-50. AdapNet is benchmarked on Cityscapes, Synthia, ScanNet, SUN RGB-D and Freiburg Forest datasets and achieve almost the same accuracy as other famous networks such as SegNet and DeepLab. In particular, it was trained on Freiburg Forest dataset, whose images resemble most the forest environment we look for(as shown below). Therefore we thought that the network parameters on this training dataset may produce ideal results.

Figure 16 – Example of RGB image and predicted semantic label

There are five categories in Freiburg Forest dataset, namely trail, grass, vegetation, sky and obstacle. Although these images are not taken in a big forest and the categories are not the same as our dataset, we still decided to apply the trained models directly to the images of forest.

The original code of AdapNet does not show the result of prediction as an image. It just calculates the accuracy rate. Since we do not have the ground truth of our images, it is unable to do so. After having modified the code to assign different colour to different value of the output of the network, the prediction result can be displayed as an intuitive image, in which each semantic label is represented by a

colour. Here we presented some examples of the prediction results. (White : Void, Red : Vegetation, Green : Grass, Blue : Trail, Gray : Sky, Black : Obstacles)



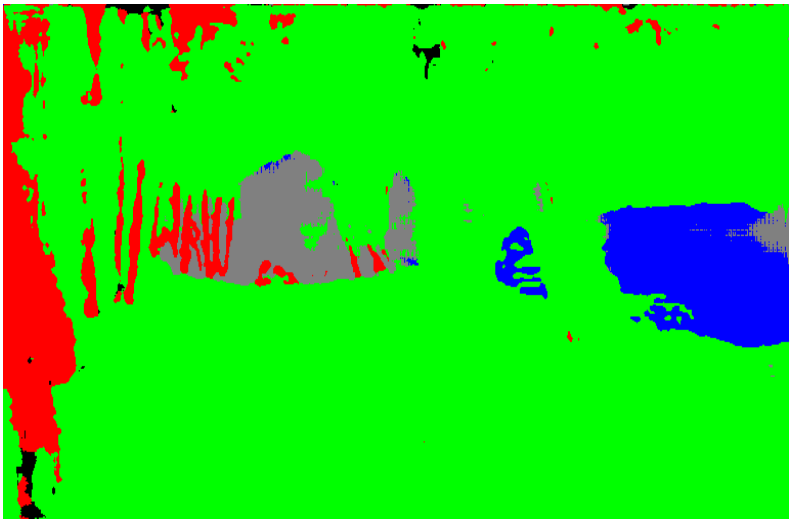


Figure 17 – Example of semantic prediction result

We found that the prediction result was not bad on the images with clear textures and clear boundaries like the images where there are trees, sky and grass. It is easier for the network to distinguish the boundaries on this type of images. However, the result on the images with big difference was not ideal, and the prediction effect for images that are not very clear or contain a single category was very general. Of course, if an image contains only trees, that is, only one category, then its prediction result should have only one colour, but such a result is of little significance. Moreover, since the categories are not the same between two datasets, the wrong label of prediction were also under consideration. For example, the network often mixed grass and trees of similar colour into one, the category obstacle is defined fuzzy and hard to predict. In all, the result was not up to standard.

We have also tested with the trained models on other datasets, but the effect is worse than that on Freiburg Forest dataset.

7. Training of the networks on synthesized dataset

7.1 Introduction of synthesized dataset

Next, we planned to be able to perform semantic segmentation on our own dataset. For semantic segmentation, manual annotation is feasible, but it is too time-consuming and labour-intensive, while the accuracy of annotation is not high. The annotations of some well-known image segmentation datasets are not perfect neither. For this we decided to use artificially synthesized dataset.

Synthesized dataset is generated by virtual environment created with 3D modelling software. Compared with real dataset, it has many advantages. The biggest advantage is that once the modeling is completed and the images RGB are

generated, there is no need for manual annotation. Software itself or third-party tools can help us complete this work, since in theory each pixel on the image is generated by the software, of course the software can correctly generate semantic labels. In addition, the created virtual environment is like a 3D game in which user can move and switch the viewpoint freely, it allows user to conveniently select images at different positions and different viewpoints. The disadvantage of synthesized dataset is that modeling is a very heavy job. Fortunately, there are generally 3D environments built by others on the forum and community, which can be used directly after downloading.

7.2 Introduction of Unreal Engine

Having compared some software, we decided to use Unreal Engine 4, the most advanced Unreal Engine. Developed by Epic, UE4 is one of the world's most widely-authorized game engines, with 80% of the global market share for commercial game engines. Unreal Engine is the most open and advanced real-time 3D creation platform in the world. After continuous improvement, it is not only a hall-level game engine, but also brings unlimited creative freedom and unprecedented control for professionals from all walks of life. Whether it's cutting-edge content, interactive experiences or an immersive virtual world, everything is in Unreal Engine.

I hadn't been in touch with Unreal Engine before, so I spent some time learning the basics of Unreal Engine. Many online tutorials are about game development and they are about the design of game logic, which has little to do with our needs. We mainly focused on the construction of three-dimensional scenes. After a week of introductory learning, I probably learned the basic process of building a virtual scene in UE4, including creating terrain, vegetation, and placing objects (called actor in UE4, which can be created ed by the geometry editing tool in the engine or imported by other modeling software), then there are rendering lighting, post-processing and other follow-up work to make the virtual scene look more realistic.

Unreal Editor (UnrealEd) in UE is an operating tool with "what you see is what you get" as the design concept. It can make up for some deficiencies that cannot be achieved in 3D Studio Max and Maya, and is well used in game development. In the visual editing window, game developers can directly control the free placement and attributes of the characters, NPCs, items and props, AI waypoints and light sources in the game, and they are all rendered in real time. And this real-time rendering also has dynamic light and shadow effects.

Real-time map editing tools allow art developers to freely adjust the height of the terrain, or directly blend and modify the map layer through a brush with an alpha channel.

7.3 Preparation of virtual environment and dataset

Designing a virtual forest environment that we need from scratch is too cumbersome. Fortunately there are many available resources on the Internet and in the Unreal Mall on the Epic platform. Finally, we chose and downloaded two of them. One was developed under the engine of the version 4.16, the other one requires UE 4.22 or higher version to run. The screenshots of these two virtual forest environments are shown below.

Figure 18 – Screenshots of virtual environments

The first environment is more likely a wood, in which there are trees of different sizes and stones. The second environment is a large forest where the trees are big and tall.

Then we needed to focus on how to get the training dataset. Image data is very easy to obtain. Even without the help of other tools, we could choose the location and perspective in the virtual world and take images by using the screenshot of software or of the system. The key issue is how to generate the corresponding annotation data. UE4 itself does not have the direct function to generate annotation files , for convenience we could rely on other extension plugins. After having searched, I found a very useful plugin UnrealCV designed for UE4 for computer vision research.

UnrealCV is an open source project to help computer vision researchers build virtual worlds using Unreal Engine 4. It extends UE4 with a plugin by providing:

-- A set of UnrealCV commands to interact with the virtual world.

-- Communication between UE4 and an external program, such as Caffe.

UnrealCV can be used in two ways. The first one is using a compiled game binary with UnrealCV embedded. This is as simple as running a game, no knowledge of Unreal Engine is required. The second is installing UnrealCV plugin to Unreal Engine 4 (UE4) and use the editor of UE4 to build a new virtual world.

Since the resources we downloaded are editable UE4 project files, not binary game files, we used the second method of installation, put the downloaded and compiled UnrealCV plugin in the specified directory, then we could use new functions of UnrealCV in UE4 engine.

UnrealCV provides a set of commands for computer vision research. These commands are used to perform various tasks, such as control camera and get ground truth. Unreal Engine provides a built-in console to help developers to debug games. This built-in console is a convenient way of trying UnrealCV commands. If we want to generate a large-scale synthetic dataset, or do active tasks, such as reinforcement learning in the virtual world, we need to allow an intelligent agent to perceive, navigate and interact in the scene. UnrealCV client allows other programs to communicate with this virtual world. The client will use a plain-text protocol to exchange information with the game. To achieve this, we also needed to install UnrealCV client, a library of python.

In order to automatically generate a dataset, a sensitive method is to manually select appropriate positions and perspectives of view, so as to ensure the quality of the obtained images, then to write a python script to record the pose of camera at each position and to save all these poses in a file, finally in another python script to move the camera to the recorded poses and to use the command of UnrealCV to take screenshot and to generate the corresponding ground truth. UnrealCV is powerful, it can generate instance segmentation masks, depth maps and normal maps. Here we need to get the ground truth of instance segmentation.

UnrealCV can select the colourable objects in the UE4 virtual world, and assign different colours to different objects to generate masks for instance segmentation. Generally, the actors with real form can be coloured, including all kinds of trees and stones, but the ground (including grass, branches on the ground) and the sky are exceptions. An example of RGB images and its ground truth of virtual forest environment based on UE 4.16 is shown as the following figure:

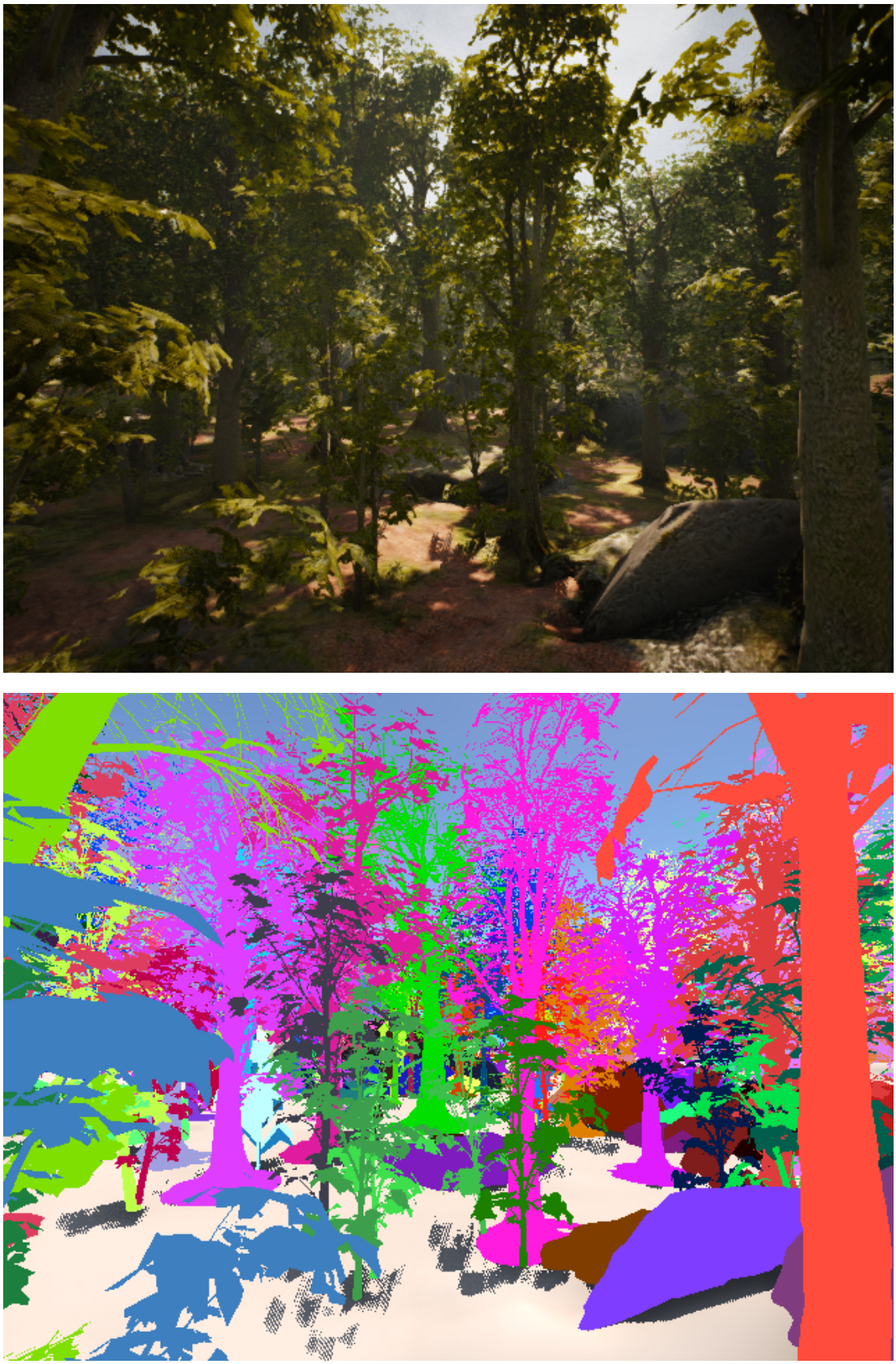


Figure 19 – Example of RGB image and instance segmentation mask

It can be found that different trees and stones are assigned with different single colours, but the colours of the sky and the ground are not uniform. If we observe carefully, we will find that their colours are different at different positions. Although their hues are generally consistent, but there are still differences. The shadows on the ground are likely grass or other small stuffs, which cannot be coloured with a uniform colour.

Since we hoped to obtain the mask of semantic segmentation of images, it requires the objects of the same category to be given the same colour. UnrealCV provides a function to get the name of each coloured object and the corresponding assigned colour. In this forest environment, the names of trees in the editor all contain the keywords tree or leaf, and the names of stones all start with rock, so we could judge the object and its corresponding category according to the colour. In this way, different trees and stones can be easily grouped together. For the sky with nonuniform colours, we can limit the colour in a range to select the pixels belonging to the sky. Finally the remaining area is the ground. Thus we could get a relatively accurate semantic segmentation mask. The effect is shown as the following figure:

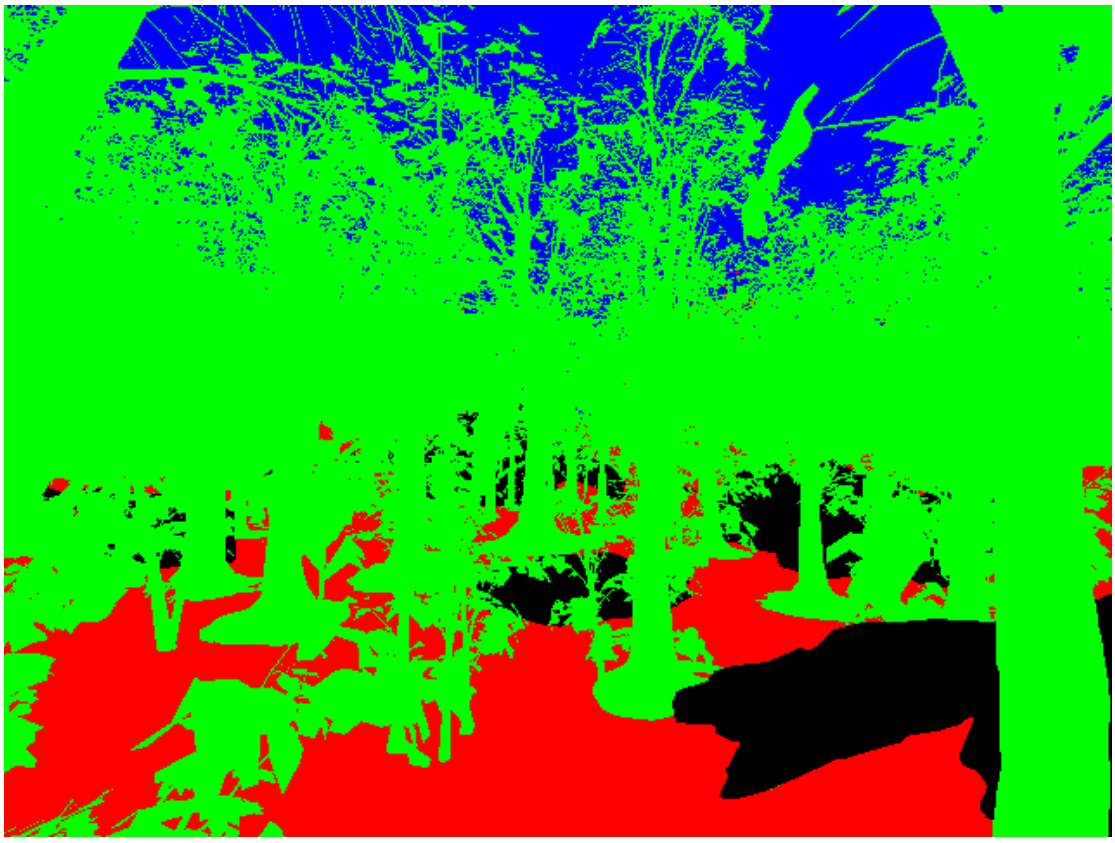


Figure 20 – Example of semantic segmentation mask

As shown in the figure, pixels of the same category are given the same color, trees are green, stones are black, the sky is blue, and the ground is red.

In addition, using the mask of instance segmentation generated by UnrealCV, we could also obtain the label of the object detection. The principle is very simple. For each tree, I picked its pixels and determined the smallest rectangular bounding box surrounding it, and saved the position and size of all bounding boxes as Yolo format. However, there are some problems here. First, there are too many trees in an image, recording all of them is too complicated. Furthermore, some trees may have too few visible pixels due to occlusion, which are unable to distinguish by human, they will still be recognized by the software, but such annotation is of little significance. Therefore, when selecting the target, we stipulated that the number of pixels of tree must be greater than a certain value before it could be recorded.

In this way, I collected more than 200 images constituting the training dataset and 25 images for the testing dataset.

7.4 Training and testing of the networks on the synthesized dataset

Next, we could start training the neural network for object detection and semantic segmentation. Due to the limited configuration of my personal computers, in order to be able to train two networks at the same time, I used Google Compute Engine to train the network of semantic segmentation. I built the environment for training on the virtual machine, downloaded the source code of the network AdapNet on Github, installed the dependent python libraries, converted the dataset into tfrecords format, uploaded it to the virtual machine and modified the training configuration file like the figure shown below.

```
gpu_id: '1'
model: 'AdapNet_pp'
num_classes: 4
initialize: 'F:\ZHD\TN10\mytrain\AdapNet-pp-master\init_checkpoint\AdapNet_pp_init.ckpt'
checkpoint: 'F:\ZHD\TN10\mytrain\AdapNet-pp-master\init_checkpoint\UEdataset'
train_data: 'F:\ZHD\TN10\mytrain\AdapNet-pp-master\config\train.record'
batch_size: 8
skip_step: 5
height: 480
width: 640
max_iteration: 500000
learning_rate: 0.001
save_step: 1000
power: 0.9
```

We must make sure that the path of training data, the height and the width of the image were set correctly, other parameters were set appropriately.

Then the training for segmentation was started. At the same time, the training for object detection was performed on my personal computer.

After a few days of training, I verified the prediction results of the latest network weights on the testing dataset. The testing results of the semantic segmentation in the same virtual environment are shown as the following figure (Green : tree, Blue : sky, White : ground, Black : stone):

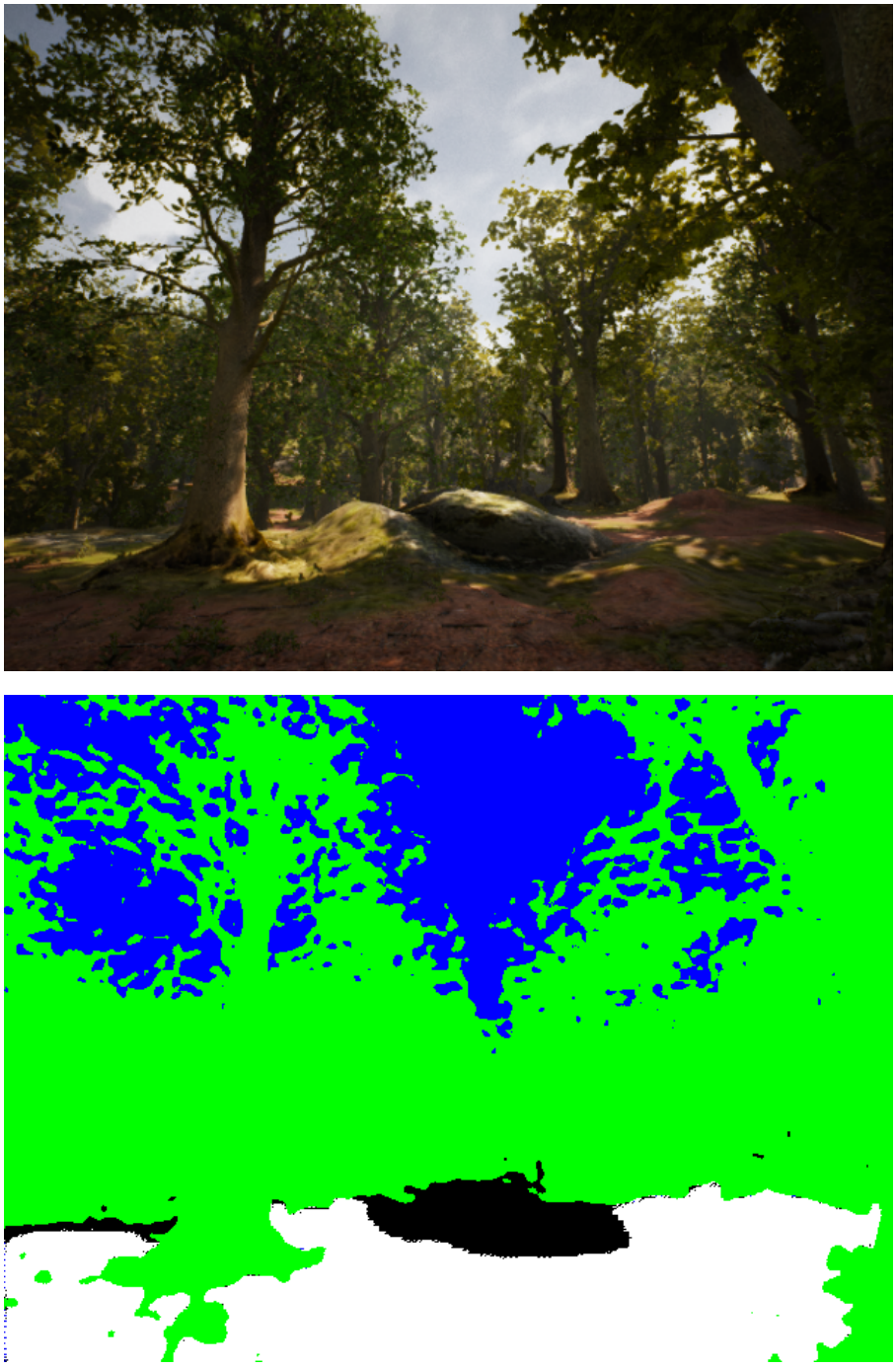


Figure 21 – Example of semantic prediction result

It could be seen that although the prediction result have some flaws on details, it is still largely receptive. The prediction of the main areas and the division of the boundaries between different categories are relatively clear.

The testing results of the semantic segmentation model in the real environment were very general. This result was expected, because the real environment is too complicated and very different from the virtual environment. The neural network cannot be out of nothing, nor can obtain unlimited generalization ability on a limited dataset.

The testing result of object detection is much worse than that of semantic segmentation. Not only the effect on the real images is not as good as before, but also the effect is not ideal on the images in the virtual environment. A common phenomenon is the appearance of many repeated and inaccurate prediction bounding boxes. Moreover, since there are many complex objects in the image, it is difficult to distinguish the range of each tree with the naked eye, and thus difficult to judge the accuracy of the prediction bounding box. An example is shown below.

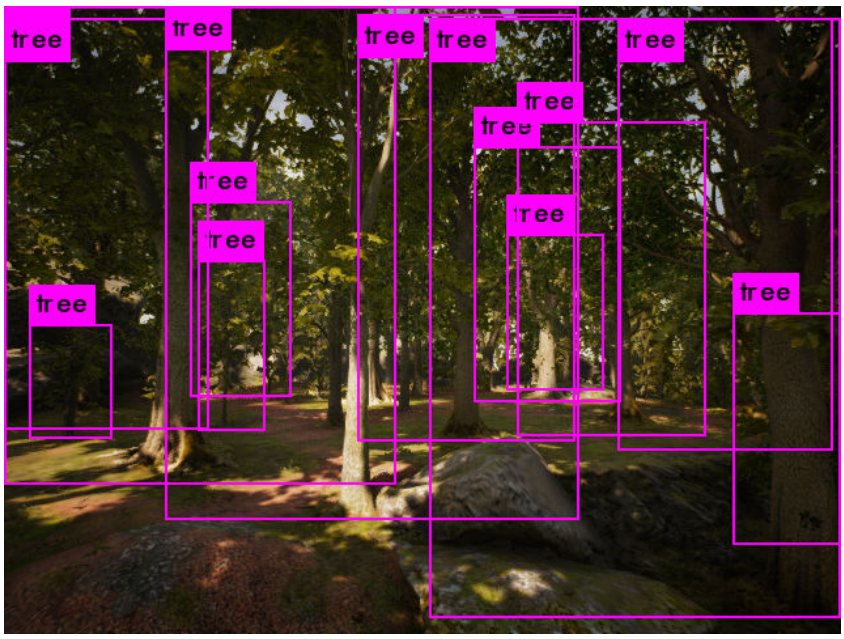


Figure 22 – Example of detection result

We analysed the reason and thought that the problem still lied on the annotation. Although we selected trees whose the number of pixels are greater than a certain value, the situation is still very complicated. Some trees can only be seen the trunk, some can only be seen the part of the branches and leaves, and others are not connected due to occlusion. That means the rectangular bounding box may contain a whole tree, may contain a trunk or may contain leaves, it may be too large to contain many irrelevant objects. Of course the trunk has different features with the leaf and the branch, but the annotation defined all of them as tree. All of these bring challenges undoubtedly to the prediction work of the network .

7.5 Optimizations and problems

Having considered that the overlapping leaves makes it impossible to distinguish which tree each leaf belongs to, we realized that predicting the whole tree is not too realistic, so we hopes that the network can only predict the trunk. To this end, we must distinguish the trunk and leaves with different colours. Such a demand cannot be realized with the functions that UnrealCV comes with, so we must do it ourselves. After having understood the principle of rendering in UE4, I found that this requirement can be achieved by replacing materials. The basic principle is to replace the material of the leaves with a material of uniform colour, and the material of the trunk with another material of uniform colour. We have tested in both virtual environments, and the effects are shown as the following figures:

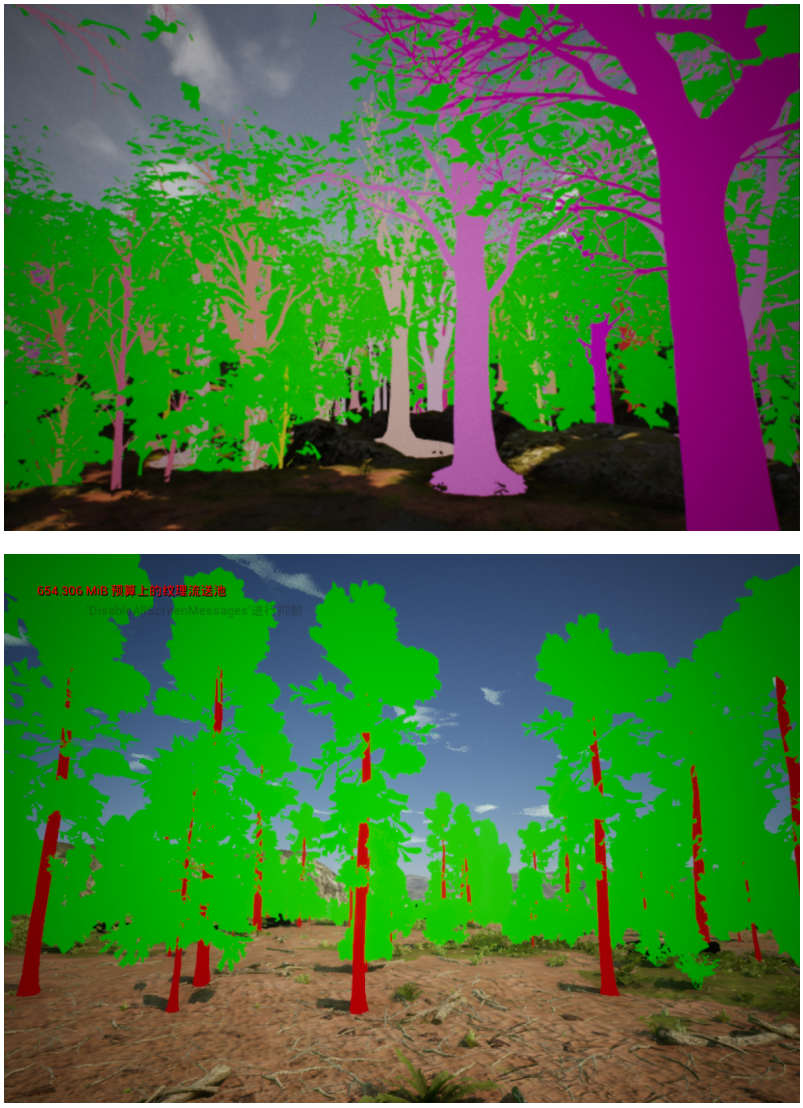


Figure 23 – Example of change of material

In the first environment, the leaves were replaced with green materials, and each trunk was given a different colour. In the second environment, the leaves are all green, but the colour of trunks are the same. This is because each tree in the first environment is an independent individual placed manually, while the trees in the second environment are automatically generated by the programmatic plant generator of UE4, and all trees are treated as a whole, each individual cannot be accessed. This means that we cannot use the second environment to generate the dataset of object detection. In addition, in the first environment there are many tree branches, but the trunk and branches are inseparable , which results in the shape of the trunk not resembling a rectangle. The rectangular bounding box will be too large and contain many interference items. It is also difficult to train the network of object detection by using this annotation.

In view of the fact that the prediction result of semantic segmentation was okay, we hoped to refine it, that is, to distinguish trunks and leaves. However, another problem was found during the process. UE4 will change the LOD material according to the distance between the object and the camera when rendering the object. Specifically, when a tree is close to the camera, the leaves and the trunk will be rendered with two different materials, but when the distance exceeds a certain range, the whole tree will be replaced with another material. The result of the above replacement is that the whole tree has only one colour, so that the trunk and leaves cannot be distinguished. The LOD material of an object is already determined during modeling, so it cannot be changed in UE4.

After we found that the critical distance of the change of material for large trees is relatively long. The big trees that are visible under the usual perspective are all smaller than this distance. The distance for small trees is close, and there will be small trees beyond this distance. So if we hide these small trees, we can achieve our goal. We believed that if we did this and trained the network on the new dataset, we would achieve expected results, but this work had not been done due to reason of time.

8. Practice of depth estimation

8.1 Introduction of depth estimation and its methods

Next, we temporarily put down target detection and semantic segmentation and tried to understand the image from another view. In addition to the semantic category of each pixel of the image, depth is also very important information. Depth estimation is a relatively unpopular branch of computer vision, but it has very important application value. It plays an important role in tasks such as robot automatic navigation, autonomous driving, VR, 3D reconstruction, etc.

There are many methods for depth estimation. The traditional SLAM method uses a stereo matching method to estimate the disparity map from the images taken by the left and right cameras and calculates the depth based on the camera internal parameters. Recently, the method of predicting the depth map directly from the image of the monocular camera through deep learning has become an hotspot application.

The monocular estimation based on deep supervised learning is based on the pixel value relationship reflecting the depth relationship. The method is to fit a function f to map the image I into the depth map D: D=f\left (I\right). Like semantic segmentation, its network architecture is also a FCN-based convolutional network, which uses the Encoder-Decoder architecture. The Encoder part is the backbone network (mostly VGG, ResNet, MobileNet, etc.) to extract features, and the Decoder part uses upsampling and Skip Connections. The method is separately fused with the features of the Encoder, and finally uses a 1*1 convolution to obtain a depth map. Some networks have more complicated optimization work. One disadvantage of this type of method is that during the training process, we need to know the reference standard of the depth value corresponding to a large number of input pictures in advance as a training constraint, so as to backpropagate the neural network and train our neural network to perform depth prediction on similar scenes. Although this type of method can bring the advantages of the neural network as a function simulator to the fullest, in reality, it is not an easy task to obtain the depth value corresponding to the scene, and the depth estimation based on unsupervised learning comes into being.

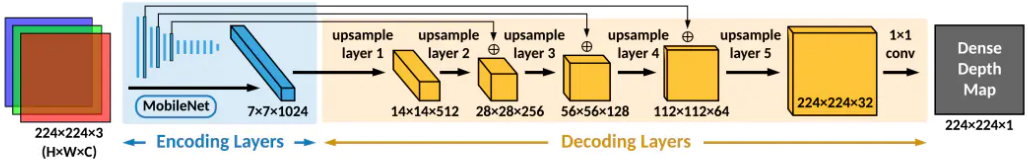


Fig. 2: Proposed network architecture. Dimensions of intermediate feature maps are given as height × width × # channels. Arrows from encoding layers to decoding layers denote additive (rather than concatenative) skip connections.

Figure 24 – Structure of a network of depth estimation

Similar to the traditional SLAM method, unsupervised depth estimation also estimates the disparity between the left and right images, but it uses a neural network to complete this task, and then the disparity is converted into depth. As long as we take the left image as the input for training and the right image as the corresponding reference standard, we establish a neural network. After training on a large number of binocular image pairs, the obtained neural network will predict the disparity with an input of an image. Thus in this way we turns an unconstrained problem into a constrained problem. At the same time, when the disparity and the parameters of camera is known, the corresponding depth can be obtained.

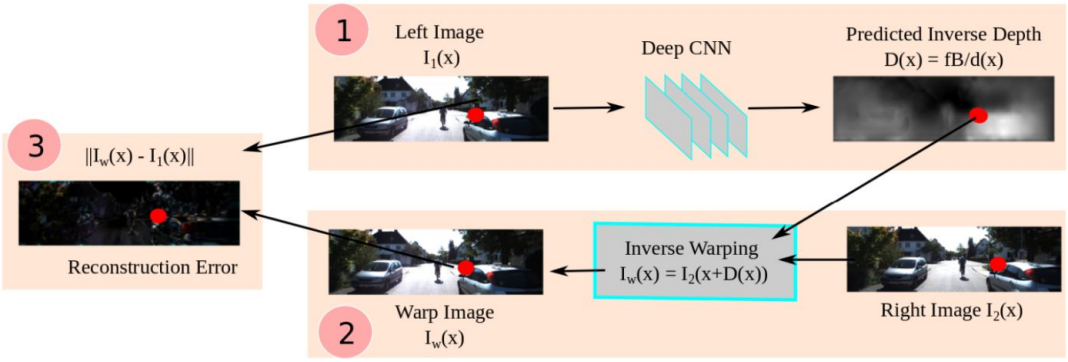


Figure 25 – Illustration of unsupervised depth estimation

8.2 Preparation of dataset

As mentioned before, UnrealCV can generate depth maps of the images in the virtual environment, so we can directly use the method of supervised learning. The same with the ground truth of instance segmentation, we obtained the depth information corresponding to each picture in the training dataset. The depth values saved as npk format by UnrealCV are all floating point numbers. With the help of OpenCV,I turned them to integers. Since the depth value of the sky is very large, I limited it to 255, the maximum value of png images, then I made the conversion and save depth information as a png grayscale image, as shown below:

Figure 26 – Example of RGB image and its depth map

The dark colour represents remote objects, and the light colour represents close objects.

8.3 Training and testing of the network of depth estimation

In terms of selecting network, we first chose FCRN-DepthPrediction, which is an Tensorflow implementation and tested on both NYU Depth V2 indoor dataset and Make3D outdoor dataset. In the first place I tested the model trained on NYU Depth V2 on our testing dataset. The result is general. Although the distance can be roughly

distinguished, it lacks accuracy , and the contour between different objects is very fuzzy, therfore it is necessary to retrain on our training dataset.

The training process was bumpy. Since the original author did not give the training code, we could only use the code published by others and modified it to meet the requirements of our data set. After a period of training and testing, we found that the results obtained were very strange. The prediction depth images are completely meaningless blocky images. It seems that the network had not learned anything useful. I browsed the issues on Github and found that someone had the same problem as mine, but these questions had not been answered by the author.

We had to abandon FCRN and chose DenseDepth ,a marked network, referring to the paper “High Quality Monocular Depth Estimation via Transfer Learning (arXiv 2018)”, which was based on Keras (Tensorflow) and trained on NYU Depth V2 and KITTI datasets. Here are some examples of the testing benchmark results.

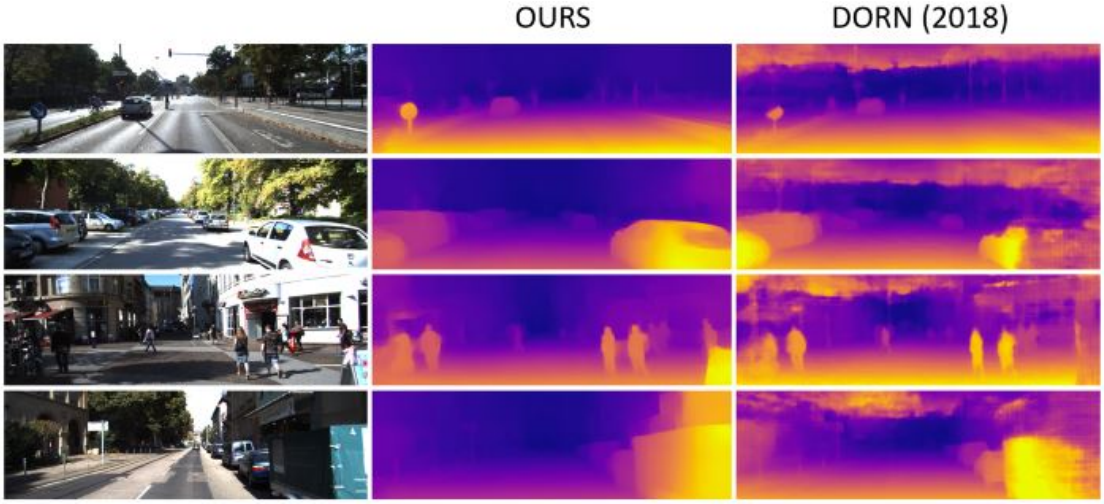


Figure 27 – examples of the testing benchmark results

In the first place I used the model trained on KITTI dataset to test, the results are shown as the following figure:

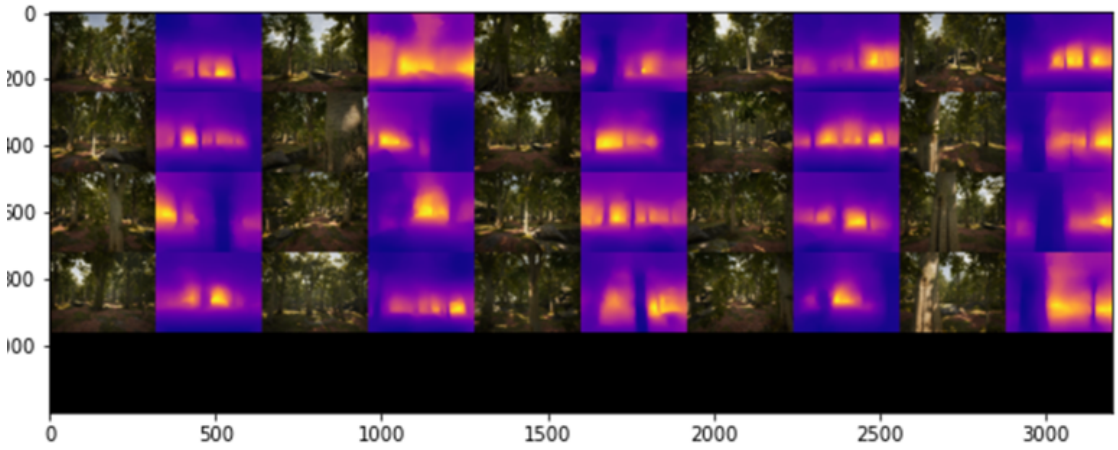


Figure 28 – testing results with existing model

We could conclude that the network just gives rough depth but cannot distinguish depth with little difference on details.

To train the network, I modified the part of the source code related to the training data, set the minimum and maximum depth values and deleted the step of normalizing depth value that is not necessary. Then the training was started. The latest model obtained by training was tested on the testing dataset, the results are amazingly perfect. An example is shown below:

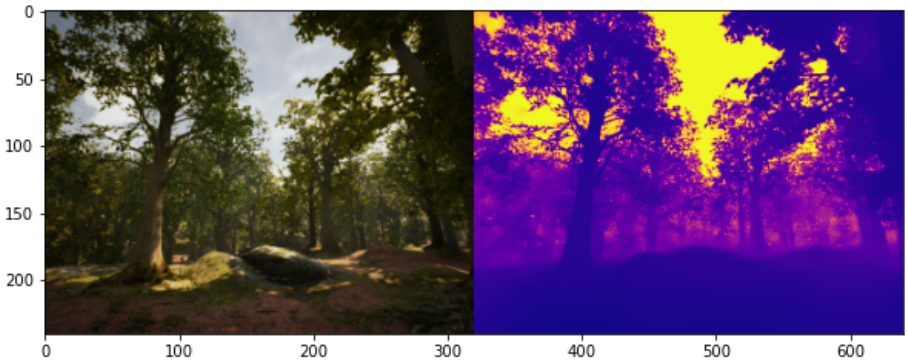


Figure 29 – examples of the prediction results

It can be seen that the prediction result has almost no defects, shows all the details, the depth values are accurate, and the contours between different objects are extremely clear, at a glance, even more obvious than that in the original RGB image.

I also tested the trained model on the images in real environment, the effect is very general, which shows the limitation and weakness of neural networks.

Here we thought about a question, since the semantic segmentation and depth estimation using networks of similar architecture, why the testing results are not the same, why the result of depth estimation is so perfect, but the result of semantic segmentation cannot reach this precision? Having compared the generated ground truth of instance segmentation and depth, we found that the mask of instance segmentation is not perfect (as shown below). That is mainly because the leaves have the problem of losing details after being replaced by a single colour, which makes that the area of leaves in the ground truth is generally smaller than that in the original RGB image . This will undoubtedly have a bad impact on the training of the network. However, The depth map does not have such problem, it is perfect, without any loss of details. In consequence the prediction result of semantic segmentation is not as good as that of depth estimation.

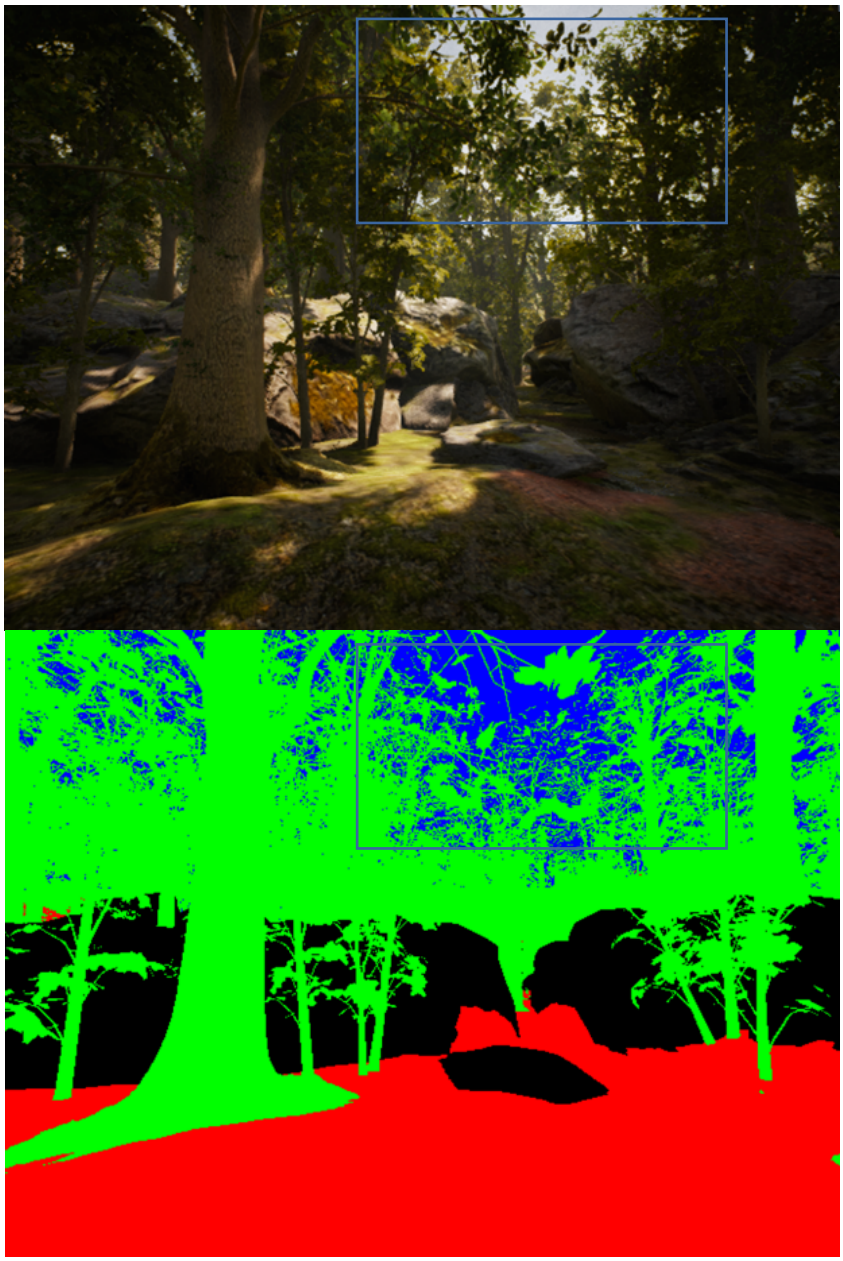


Figure 30 – Illustration of the problem of segmentation mask (the leaves are not correctly labelled in the rectangle area)

9. Fusion of detection/segmentation and depth estimation

Actually, the semantic segmentation and depth estimation are inherently related , they have some common characteristics, which can be utilized for each other. For example, semantic segmentation and depth of a scene can both reveal the layout and object shapes/boundaries. Some recent work also indicated that leveraging the depth information from RGBD data may facilitate the semantic segmentation. Therefore, a joint learning of both tasks should be considered to reciprocally promote for each other.

9.1 State of the art of the fusion of segmentation and depth estimation

After having read some paper, the fusion of segmentation and depth estimation can be concluded as three main methods presented below.

1. Simultaneous prediction of semantic information and depth information through a joint network [6]

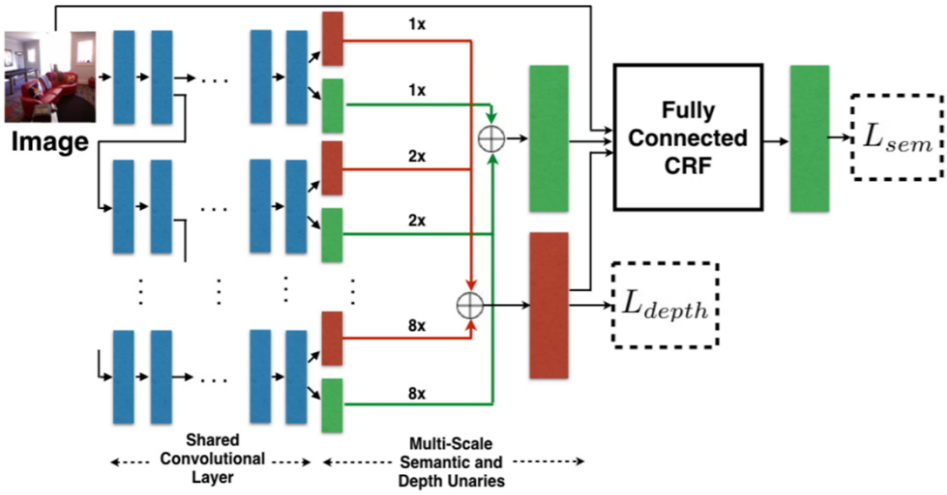


Figure 31 – Structure of the joint network

This method combines the two tasks in one network. The convolutional layers on the front end of the network are shared, and then two branches of prediction are obtained through the fusion of different layers.

2. Optimization of two independent prediction results through a joint refinement network [7]

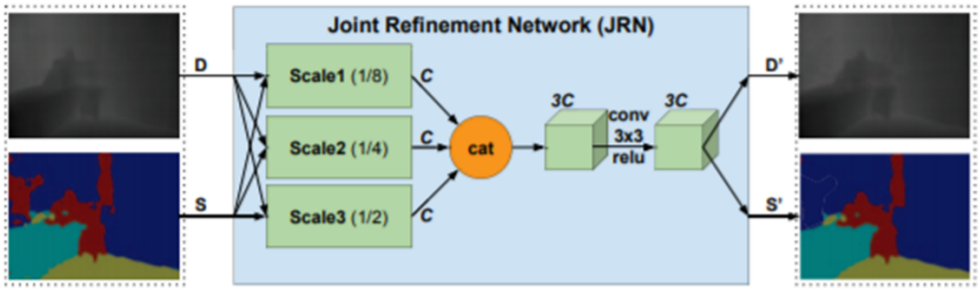


Figure 32 – Structure of the refinement network

The network optimizes two independent prediction results through the fusion of feature maps at different scales.

3. Semantic segmentation prediction based on RGBD images [8]

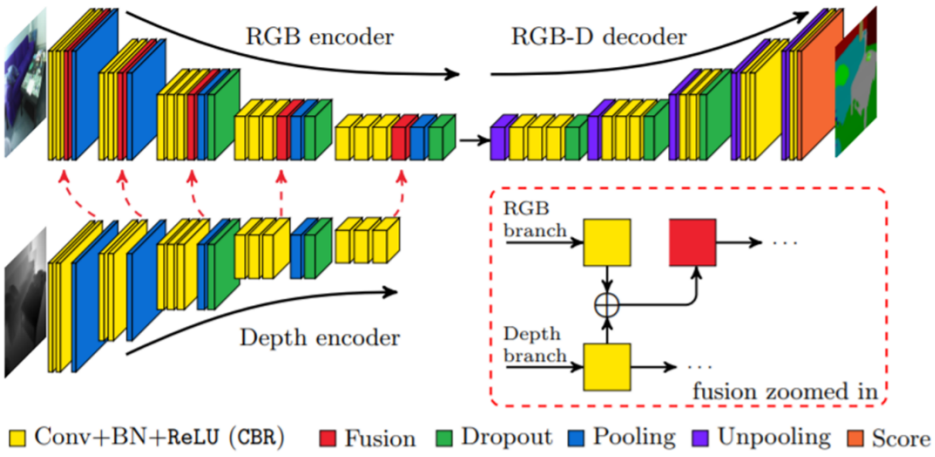


Figure 33 – Structure of the network

This type of network is used to do segmentation based on RGB information and depth information. The features extracted from two independent encoders are fused and then up-sampled by a decoder to predict semantic labels.

However, in our case, the prediction result of depth is perfect, there is no need to promote. And just as the problem mentioned above, the ground truth of the segmentation generated by UnrealCV is not accurate, If there is no absolutely perfect ground truth, then doing above optimization work will not have great effect, since no matter how we optimize it, the network cannot generate more accurate prediction results than ground truth.

9.2 Fusion of detection and depth information

To take advantage of the depth information, we thought to combine the RGB data of the image with the depth information for object detection. A recent work [9] realized this idea.

The authors used a network improved on the basis of Yolo v2. Two sub-convolution networks with the same structure are used to extract the features of both RGB image and depth image respectively and then two feature maps are fused at a certain layer, finally the prediction result outputs after a 1*1 convolutional layer.

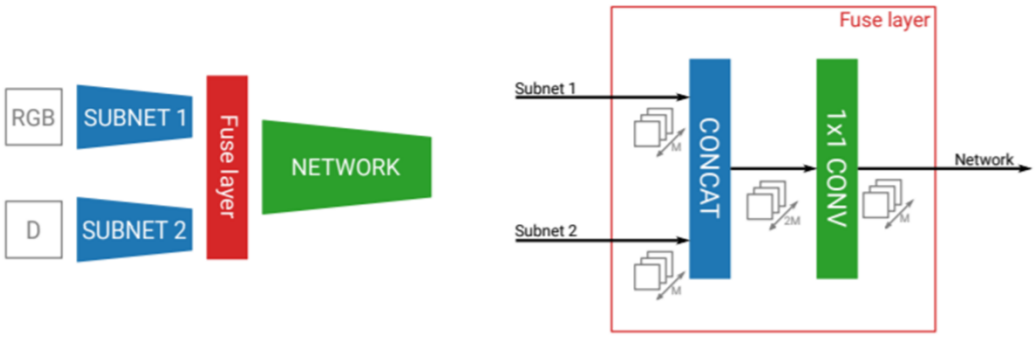


Figure 34 – Illustration of the fusion

The principle of the network is not too complicated. With the source code the authors published, we could have a try. There are not too many differences compared with the training of Yolo v3. A problem is that the format of annotation file required by the network is HDF5 or Pandas. With the help of Brambox, a python toolbox that provides the necessary tools to convert image annotations, I took the opportunity to convert the annotations to a pandas format, which is faster to parse whilst training and testing. After that I modified some configuration parameters and started training.

Unfortunately, The testing result does not show great improvement. It shows that by adding depth information, the network can predict the location of the target more accurately, but it can hardly successfully predict the targets that it cannot predict before. In the final analysis, the prediction result is up to the quality of annotation, since we do not have good annotation for trunks, it is not strange that the prediction is not ideal.

10. Attempt of instantiation of trees

10.1 bottom -up instance segmentation

Just as the objective of the project, we hoped to detect different trees and characterize them from beginning to end. Since the object detection did not work in our case, we took the instance segmentation into account. As I introduced at the beginning, there are two main types of methods for instance segmentation, namely bottom-up semantic segmentation-based methods and top-down detection-based methods, both of them belong to two-stage methods. The idea of the top-down instance segmentation method is first to find the area of the instance (bounding box) through the object detection method, and then to perform semantic segmentation in the detection box, finally each segmentation result is output as a different instance. Since it depends on the accuracy of the detection box, it is not applicable for our research. The idea of bottom-up approach is to first perform pixel-level semantic segmentation, and then distinguish different instances through clustering or metric learning method. This method may work in our case. A work in 2017 conducted this approach in three steps:

(1) Semantic segmentation : First, do semantic segmentation in the first stage to get the masks of all objects.

(2) Pixel embedding : The network is trained by using a discriminant loss function. The optimization goal of the network is to project each pixel of the image into the n-dimensional feature space, and the pixels belonging to the same instance should be as close as possible, the pixels belonging to different instances should be far. In other words, each instance corresponds to a cluster, and different clusters should stay away.

(3) Post-processing: Finally, use clustering methods (such as mean-shift) to output different instances.

However, this method has high requirements on the dataset. For example the number of instances in an image cannot be too large, and it is best to mark the centre point of each instance. Moreover, this method also has some drawbacks. Due to the holistic treatment of the image, it performs well on datasets with a lot of similarity across the images, but underperforms on datasets where objects can appear in random constellations and diverse settings. For example, if this method were trained on images with only one object, it would perform badly on an image that unexpectedly contained many of these objects. Having considered the diversity

of trees across the images in our case, we did not think this method would bring about good result.

10.2 instance segmentation via clustering

Inspired by the top-down instance segmentation method, if we can extract the pixels belonging to trunk in the images, then it is possible to divide these pixels into different instances by clustering. Because the pixels of the same trunk have similar depth values and their spatial distances are not too far, these two points can be used as the standards of clustering.

In order to prevent the case that there are too many instances in the image, we first removed the areas whose depth values exceed a certain range according to the depth information of the image, only keep the nearby objects. After this operation, the sky and the trees in the distance were removed. This will not affect the understanding of the scene in the real environment, because the drone does not need to care about the objects that are too far away. In addition, according to the results of semantic segmentation, we can get the pixels belonging to the ground and stones in the image, and after removing them, we can theoretically get the image only containing nearby trees. An rendering is shown as follows:

Figure 35 – Example of the image after filtering

It can be found that since the prediction result of semantic segmentation is not 100% accurate, some pixels belonging to the ground were not completely removed.

Next, we cold extract the trunk through colour features. In order to accurately represent the colour features of the trunk, the image was converted from the RGB colour space to the Lab colour space. The Lab colour model makes up for the deficiencies of the RGB colour model. It is a device-independent colour model and also a colour model based on physiological characteristics. The Lab colour model is composed of three elements, one element is brightness (L), and a and b are two colour channels. Considering that the brightness of different areas of the image is not necessarily the same, the relationship between the a and b channel components that are not related to the brightness can be analysed through the Lab colour model. While the three channels in the RGB color model are all related to brightness and are easily affected by brightness factor.

We randomly selected the Lab colour values of pixels at different positions of the trunk and established a Gaussian model of the colour distribution of the trunk. Then for each pixel of the filtered image, we calculated the normalized distance between its Lab color and the Gaussian model. If the distance is less than a threshold, it is judged to be a tree trunk, and if the distance is greater than this threshold, it is considered as other areas.

The image obtained by this method contains a lot of noise points and small isolated blocks. We could remove them through some image processing methods. The optimized image is as follows:

Figure 36 – Extracted trunk

It is worth noting that since the color of some pixels belonging to the ground are very close to the color of the trunk, some of the extracted areas still contain the ground. This error occurs because in the first step we cannot remove the ground correctly based on the semantic segmentation information, and cannot distinguish the trunk from the ground based on the difference of colour.

Then we used the maximum and minimum distance algorithm to achieve clustering. This algorithm does not need to know the number of clustering categories, so it is suitable for solving our problem. The maximum and minimum distance method is a heuristic-based clustering algorithm in pattern recognition. It is based on Euclidean distance and takes the object as far as possible as the cluster center. Therefore, it can avoid the clustering seeds that may be too close when selecting the initial value of the K-means method. It can not only intelligently determine the number of initial cluster seeds, but also improve the efficiency of dividing the initial dataset. The algorithm is based on Euclidean distance. First, a sample object is initially used as the first cluster center, and then a sample farthest from the first cluster center is selected as the second cluster center, and then each sample is calculated its distance with each cluster center. If the distance exceeds a threshold, it will be classified into a new category. According to this method, other cluster centers are determined until no new cluster centers are generated. Finally, the samples are classified into the nearest category according to the principle of minimum distance.

To use this clustering algorithm, it is necessary to define the Euclidean distance and the threshold to determine the new category. We defined the distance between two pixels as the weighted average value of the difference of the coordinates in the horizontal direction and the difference of depths, and set the threshold to a certain value tentatively. The result after clustering is shown in the figure below:

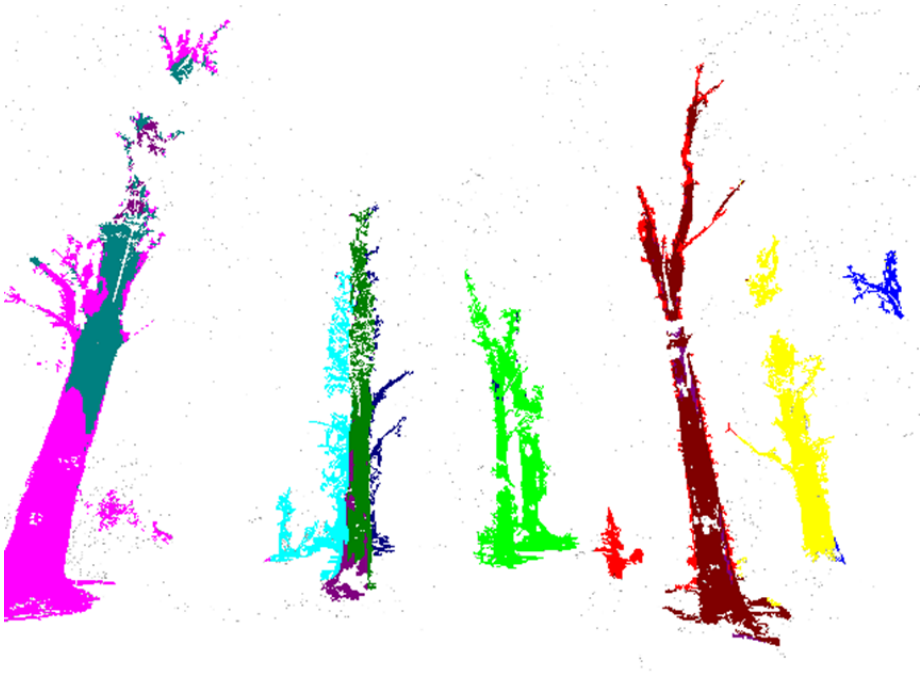


Figure 37 – Result of clustering

It can be found that the clustering results are not perfect. Although some gathered pixels can be classified into the same category, there are also incorrect classifications. Moreover, the computing time of this clustering algorithm is still relatively long, which does not meet the real-time requirements.

10.3 Possible improvements in the future

We could improve the result of this method. For example, the pixels belonging to the trunk can be extracted more accurately through the features of texture, so the interference pixels belonging to the ground can be removed. When doing clustering, the spatially gathered pixels can be divided into one or several categories according to the depth value and then different gathered blocks will be clustered again, in this way we can reduce the computational complexity by avoiding calculating the distance between each point and the cluster center irregularly.

11. Conclusions and outlooks

Due to the constraint of time, most of the work in this internship is about the perception in the forest environment, and does not involve navigation much.

In the process of practicing image recognition, the biggest difficulty is undoubtedly the production of datasets. The quality of the dataset directly affects the quality of the network's training results. Since there is no ready-made dataset, manual production is a very tedious task. Even the annotations generated by software are not perfect. If we can produce a dataset in forest environment of high quality, it will be immensely beneficial to similar researches. It is worth noting that the virtual forest environment that we selected is relatively complex, in which the trees are dense and their morphological differences are large, the colour of ground is also relatively close to the colour of trunk, which undoubtedly brings great challenges to the work of image recognition. But if we can solve the problem in a complex environment, it will be no problem to apply it in other more simple environments.

In addition, the visual navigation of UAV is also a topic worthy of research. How to apply the results of perception to path planning and how to control its movement through the planned path, all these require further discussion.

12. References

[1] Nikolai Smolyanskiy, Alexey Kamenev, Jeffrey Smith and Stan Birchfield. Toward Low-Flying Autonomous MAV Trail Navigation using Deep Neural Networks for Environmental Awareness. arXiv:1705.02550v3 [cs.RO] , 2017

[2] Fredrik Georgsson and Thomas Hellström. Visual tree detection for autonomous navigation in forest environment. Conference Paper · July 2008 · Source: IEEE Xplore

[3] Xianyi Chen, Sun'an Wang, Binquan Zhang and Liang Luo. Multi-feature fusion tree trunk detection and orchard mobile robot localization using camera/ultrasonic sensors. Computers and Electronics in Agriculture 147 (2018) 91–108

[4] Mohammed Ayoub Juman, Yee Wan Wong, Rajprasad Kumar Rajkumar and Lay Jian Goh. A novel tree trunk detection method for oil-palm plantation navigation. Computers and Electronics in Agriculture 128 (2016) 172–180

[5] Tu˘gba Yıldız. Detection of tree trunks as visual landmarks in outdoor environments. August, 2010

[6] Mousavian A, Pirsiavash H and Košecká J. 2016. Joint semantic segmentation and depth estimation with deep convolutional networks//Proceedings of the 4th International Conference on 3D Vision. Stanford: IEEE, 611-619[DOI: 10.1109/3DV.2016.69]

[7] Omid Hosseini Jafari, Oliver Groth, Alexander Kirillov, Michael Ying Yang and Carsten Rother. Analyzing Modular CNN Architectures for Joint Depth Prediction and Semantic Segmentation. 2017 IEEE International Conference on Robotics and Automation (ICRA)

[8] Caner Hazirbasy, Lingni Ma, Csaba Domokos and Daniel Cremers. FuseNet: Incorporating Depth into Semantic Segmentation via Fusion-based CNN Architecture. in ACCV 2016

[9] Tanguy Ophoff, Kristof Van Beeck and Toon Goedemé. Exploring RGB+Depth Fusion for Real-Time Object Detection. Sensors 2019