CS412 - Final Project: Semantic Segmentation Task

1451030 - Tu-Khiem Le
1451067 - Van-Tu Ninh

January 9, 2018

1 Faster RCNN Overview

1.1 Introduction

In order recognize the objects with a given set of classes, there are multiple methods which well performed this job. For instances, we can refer to some state-of-the-art algorithm in object detection like Faster R-CNN, YOLO, SSD, etc. In the scope of this project, we decided to investigate and utilize the Faster R-CNN methods.
The Faster R-CNN proves its advantages by achieving the accuracy of 73.2% mAP on PASCAL VOC 2007 and 70.4% mAP on PASCAL VOC 2012. Also, it is a great improvement from the previous models of R-CNN and Fast R-CNN by proposing the Region Proposal Network (RPN) which result in an approximately linear test execution time.
Illustrations on the result of the system detection are shown in figure 1.

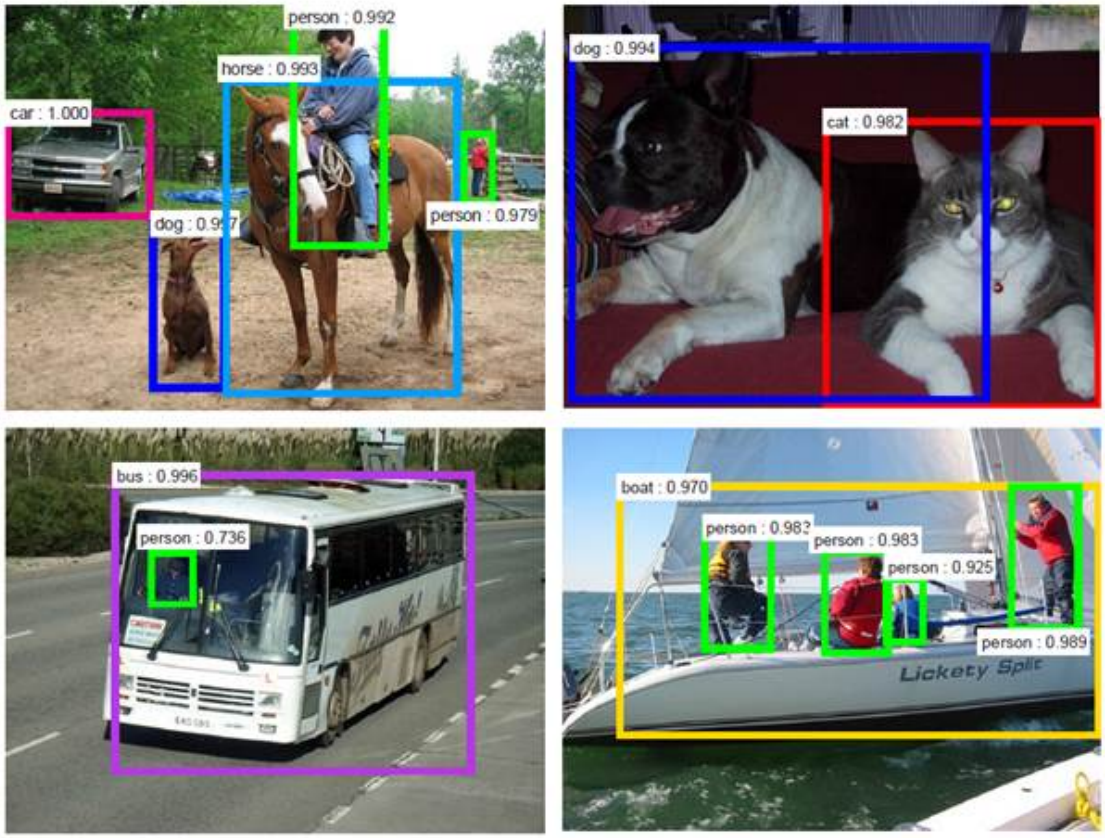

Figure 1: Illustrations of the system detection.

1.2 Architecture

Faster R-CNN is driven by the success of the region proposal method and region-based convolutional neural network (R-CNN). With the later improvement of the R-CNN method which is Fast R-CNN where applying the sharing convolution across proposals and utilizing very deep networks, it can achieve nearly linear time when ignoring the execution time of region proposals step. In Faster R-CNN, the large amount of time spent of region proposals in reduced which solving the weakness of Fast R-CNN and achieving a faster total execution time.
n general, Faster R-CNN is the combination of the Region Proposal Network (RPN) and Fast Region-Based Convolutional Neural Network (Fast R-CNN) by sharing convolutional layers. By doing this at test-time, the marginal cost for computing proposals is small, since enabling the nearly cost-free region proposals. The architecture of Faster RCNN is descibed in figure 2. The

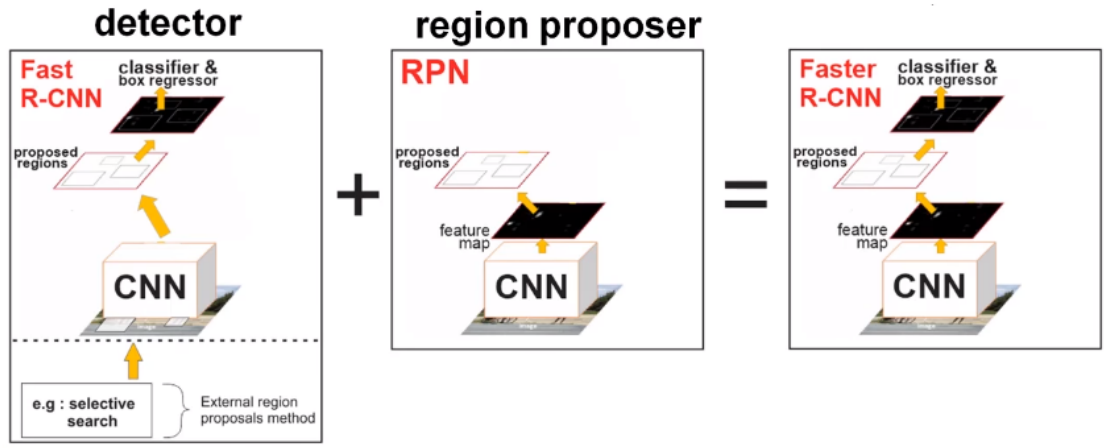

Figure 2: Architecture of Faster RCNN.

following sections describe a deeper structure of each component in Faster R-CNN model.

1.2.1 Region Proposal Network (RPN)

In this section, we are going to discover more details on the structure of RPN. The RPN takes input as an image of any size and outputs a collection of rectangular object proposal region. The term object proposal region here stands for the region with a high probability of containing an object.
The whole process of RPN is model by a fully convolutional network. And with the goal of create the sharing layers with Fast R-CNN, there are two networks are considered in the paper work by the authors. They are the Zeiler and Fergus Network (ZFNet – ILSVRC 2013 winner) and the Simonyan and Zisserman Network (VGGNet – runner-up in ILSVRC 2014). The structure of ZFNet is shown in figure 3, with 5 shareable convolutional layers. ZFNet typically is an improvement of

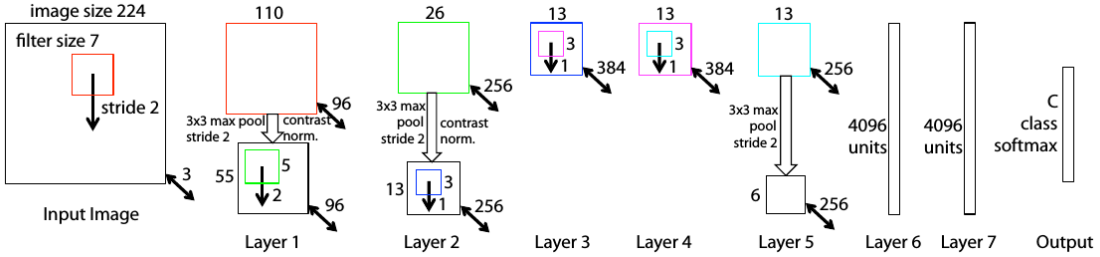

Figure 3: Architecture of ZF Net.

AlexNet with some modification, including:

- Convolutional Layer 1 changes from (11×11 stride 4) to (7×7 stride 2).

- Convolutional Layer 3, 4, 5 are deeper.

The structure of VGGNet is shown in figure 4, with 13 shareable convolutional layers. In VGGNet, it only use 3×3 filters at stride 1 and the 2×2 max pooling at stride 2. With the fully convolutional

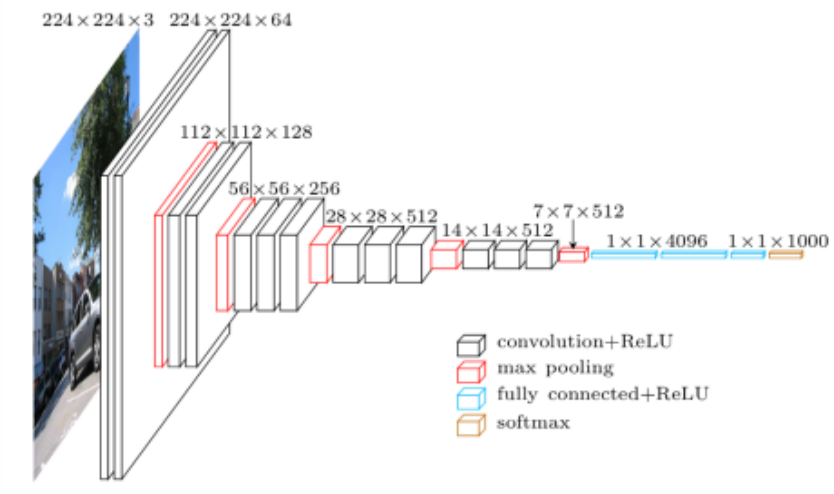

Figure 4: VGG 16-layer network architecture

network above, we can obtain a feature map in the final convolutional layer (layer 5 in ZFNet and layer 13 in VGGNet). So to generate the region proposals, the authors use small network and slide it through the feature map with an $n \times n$ sliding window ($n = 3$ is used by the authors). After that, the window is mapped to a vector with 256-d (ZFNet) or 512-d (VGGNet). This vector will be used as an input for the box-classification layer (cls layer) and box-regression layer (reg layer). This process is illustrated as in figure 5.

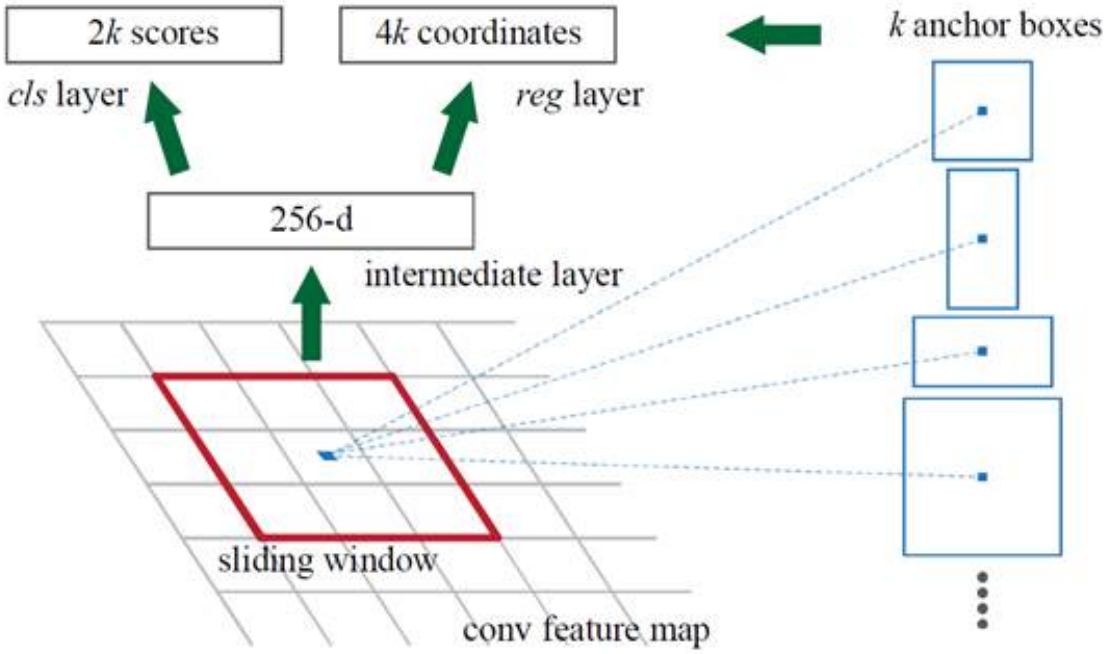

Figure 5: Object detection progress

Box-Classification and Box-Regression

For the sliding window, the authors generated some boxes called anchors with 3 ratio (2:1, 1:1, 1:2) and 3 scale (128, 256, 512). Hence, we have 9 type of anchors in total. The reason behind

of creating multiple ratio and scale of the anchors is for improving the accuracy in detecting the object shown in the image. To detect object, the authors use the generated anchors to do the estimation. The summarization of this process is as in figure 6. The figure 6 illustrate the RPN

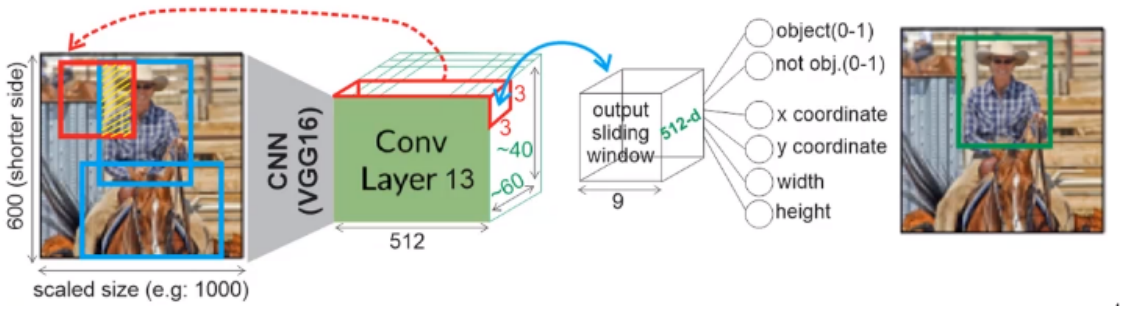

Figure 6: Region Proposal Network Progress

with the VGG16 Network, the red box is the anchor generate by the $3 \times 3 \times 512$ sliding window at the Conv Layer 13, the blue boxes are the ground truth. To analyze if an anchor contains object, the author use the IoU function, where IoU is defined as:

$$IoU = \frac{Anchor \cap GroundTruth}{Anchor \cup GroundTruth}$$

where $IoU > 0.7$: Anchor contains object and $IoU < 0.3$: Anchor doesn't contain object.
Next, after checking for the anchor, we will obtain an output sliding window, feeding it to the box-classification and box-regression layers, we obtain the result:
Classification layer

- probability the box is object
- probability the box is classification

Overall, this layer produce $2k$ elements.
Regression layer

- x, y coordinates of the region proposal
- w, h corresponding to width and height of the region

Overall, this layer produce $4k$ elements.
Overall, there are $2k + 4k$ produced elements which generate the green box (in the figure) which carry the information of containing object or not.

Loss Function

So, with the model as above, to adjust the network to learn more accurate, the user defined the loss function as below:

$$L(\{p_i\}, \{t_i\}) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{req}} \sum_i p_i^* L_{reg}(t_i, t_i^*)$$

We can notice that the loss function is the sum of two smaller loss function. The first part is the loss function for classification and the second part is for the regression.

- **Classification Loss Function**

$$L(p_i) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*)$$

where: p_i : predicted probabilitym p_i^*: 1 if anchor contain object, 0 if anchor doesn't contain object, N_{cls} : number of anchors in minibatch.
The L_{cls} function is basically the Softmax Function to minimize the error and make the classification more accurate, the Softmax Function is defined as:

$$L_i = -\log(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}})$$

- **Regression Loss Function**

$$L(t_i) = \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*)$$

where: λ : constant value, N_{reg}: number of total anchor, p_i^*: 1 if anchor contain object, 0 if anchor doesn't contain object, t_i : predicted box, t_i^*: ground truth box.
The L_{reg} is the Smooth Function, this function would help to adjust the bounding box to come closer to the groundtruth box, making the proposal region more accurate. The L_{reg}function is defined as:

$$L_{reg(t_i,t_i^*)} = smooth_{L1}(t_i - t_i^*) = smooth_{L1}(x) = \begin{cases} 0.5 * x^2 & if|x| < 1 \\ |x| - 0.5 & otherwise \end{cases}$$

1.3 Fast Region-Based Convolutional Neural Network (Fast R-CNN)

This method takes the image and the proposed region as input and classify the object proposal to the appropriate label. This is an improved version of the R-CNN with a faster training and testing time. The table is shown in figure 7.

The structure of Fast R-CNN can be summarized as in figure 8. However, the problem of Fast

	R-CNN	Fast R-CNN
Training Time:	84 hours	**9.5 hours**
(Speedup)	1x	**8.8x**
Test time per image	47 seconds	**0.32 seconds**
(Speedup)	1x	**146x**

Figure 7: Comparison table

R-CNN is that it still depends on an external region proposal which cause a high amount of time spent on getting the region proposal. In Faster R-CNN, it simply inherited this model and does the improvement by sharing the convolutional layer from RPN to Fast R-CNN to reduce time.

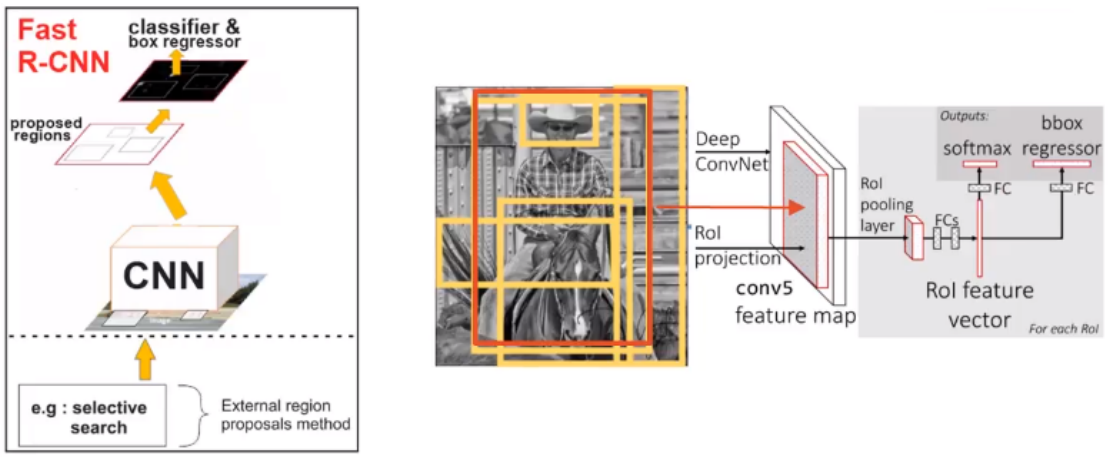

Figure 8: Fast RCNN Structure

1.4 Faster R-CNN

Now back to the general model, to utilize two network RPN and Fast R-CNN into one model, the authors conducted four steps:

- Train the RPN as describe above and initialize it with the Image-Net pre-trained model and fine-tuned end-to-end for region proposal task.
- Train detector network with Fast R-CNN using region proposal from step 1 and initialized with the Image-Net pre-trained model.
- Use the detector network (step 3) to initialize RPN training, but fixing the shared convolutional layers and only fine-tune the layers unique to RPN. Now the two networks share the convolutional layers.
- Fixing the shared convolutional layers fixed, fine-tuning the fully connected layers of the Fast R-CNN. Now both networks not only share the convolutional layers but also form a unified network.

2 Fully Convolutional Networks Overview

2.1 Introduction

Fully Convolutional Networks are robust networks which are efficient at learning and inference. They receive input of arbitrary size and give the output of correspondingly-sized output (corresponding spatial dimensions). Fully convolutional networks inherit the models from contemporary classification networks such as GoogleLeNet, VGGNet and AlexNet. Fully convolutional networks use their learned representations to adapt to the segmentation problem. The approach works quite well on semantic segmentation problems because the networks perform segmentation along with classification of objects in the image. However, merging the two methods may raise errors especially on classification problem. It mis-classifies classes which severely affect the result of the semantic segmentation. The problem of fully convolutional networks is mentioned later in our proposed method in semantic segmentation problem as comparison between the output of fully convolution networks used in semantic segmentation and our proposed solution.

2.2 Background

1. **Fully Convolutional Networks general input layer structures**
   Fully Convolutional Networks are actually Convolutional Networks (convnets in brief). Therefore, they receive and transform input into the form of three-dimensional array of size $h \times w \times d$, where h and w are spatial dimensions, and d is the feature or channel dimension in all the layers in the network.
2. **Dense Prediction**
   Dense prediction is the task of predicting the label for each pixel in the image.
3. **Receptive fields**
   Receptive fields are locations in higher layers which are corresponding to the locations in the image that they are path-connected to.

2.3 Fully Convolutional Networks

2.3.1 Convert classification nets for dense prediction

Use classification nets as a step in fully convolutional networks can produce the coarse output maps. We can use them for pixel prediction by stitching the output maps together to generate the pixel labels prediction.
Recognition nets often obtain fixed-sized input and produce non-spatial outputs. It is because of the fixed-size fully connected layers of its networks throw away the spatial coordinates. We can address this problem to increase the capability of the networks to obtain the arbitrary size input by changing our view. Indeed, convolutions with kernels that cover their entire input regions can be considered as homologous as the fully connected layers in those recognition nets. By doing so, we can convert the classification nets into fully convolutional networks which take arbitrary size input and output classification maps. The resulting maps are equivalent to the ones in the original

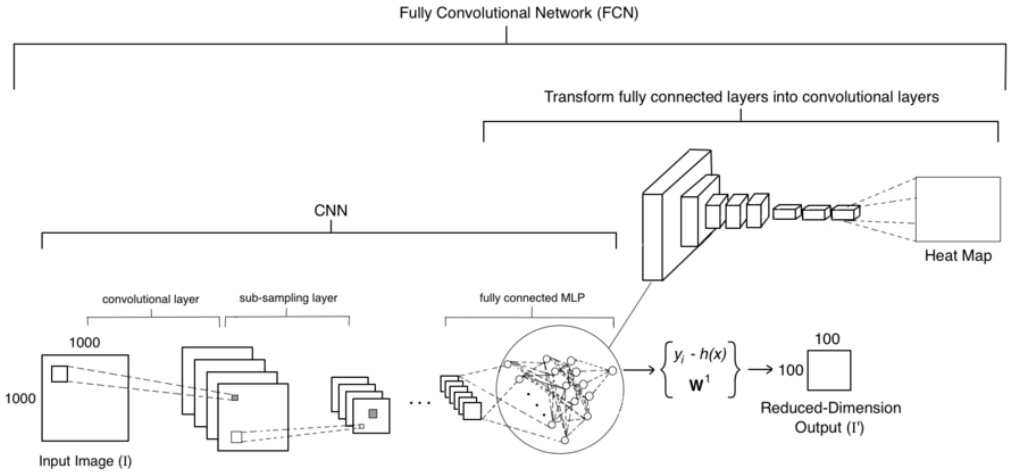

Figure 9: Replace fully connected layers with convolutional layers in fully convolutional networks

nets. Figure 9 describes the general process of replacing fully connected layers with convolutional layers in fully convolutional networks.

These output maps of the convolutional layers is a good choice for dense prediction problems like semantic segmentation. Although the replacement of fully convolutional layers provide the ability to accept arbitrary size input, its output dimensions are reduced by subsampling. Its reason is to keep the filters small and maintain the computational requirements reasonable.

2.3.2 Fast and effective learning dense prediction with upsampling

There are many ways to connect coarse outputs to dense pixels such as shift-and-stitch trick or interpolation. One way that the authors choose to use in their fully convolutional networks in upsampling step is using interpolation to re-produce the dense pixels from coarse outputs of previous step.

Indeed, upsampling with a factor f is doing convolution with a fractional input stride of $\frac{1}{f}$. We can use backwards convolution (or deconvolution in alternative name) with an output stride of f to complete the task of upsampling in this process. The deconvolution filter in this layer does not need to be fixed but can be learned.

Due to upsampling by backwards convolution's simplicity in reversing the forward and backward passes, it is often performed in-network for end-to-end learning by backpropagation from the pixelwise loss. And in fact, in the experiments of fully convolutional networks for semantic layers in semantic segmentation, the authors claim that this method is fast and effective when using it to learn dense prediction.

2.3.3 Input size training

Instead of using patchwise training by dividing the input size in order to train batch-by-batch, the authors argue that using patchwise training does not yield faster or better convergence for dense prediction. Therefore, the whole image is used for effectively and efficiently training.

2.4 Segmentation Architecture

The authors train and validate on the PASCAL VOC 2011 segmentation challenge. They use per-pixel multinomial logistic loss for training and standard metric of mean pixel intersection over union (the mean taken over all classes, including background) for validation. When training, they ignore ambiguous or difficult pixels in the ground truth. Fully convolutional networks for semantic

segmentation is described in figure 10.

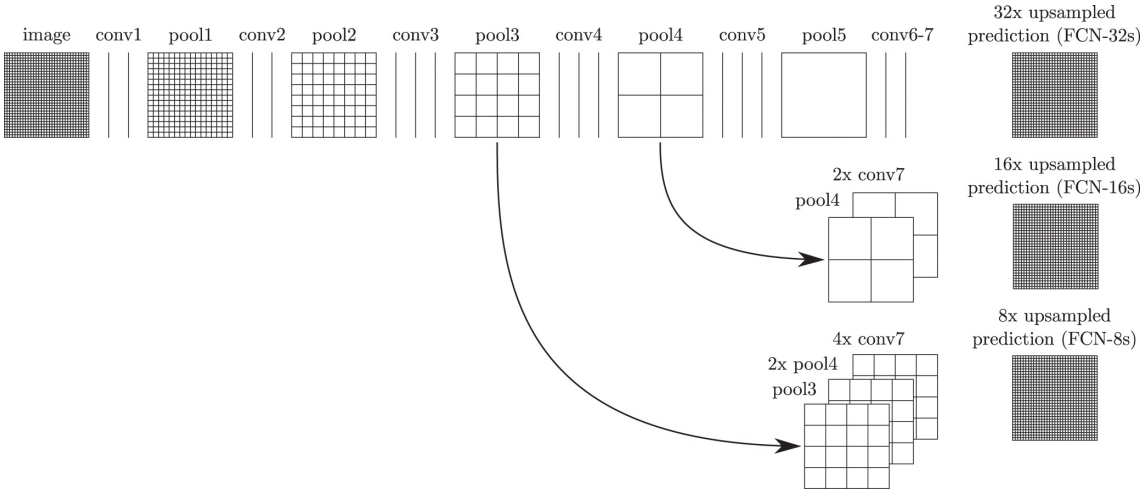

Figure 10: Fully convolutional networks for semantic segmentation architecture. In this architecture, the third row (FCN-8s) provides higher precision.

The authors choose to pick VGG 16-layer net which is shown in figure 4 as the first step to process input, only use the final loss layer and discard the final average pooling layer in GoogleLeNet to improve the performance. The authors chop off the final classifier layer of each net and convert all fully connected layers into convolutions. The authors predict the score by adding 1×1 convolution with channel dimension of 21 for each of the coarse output locations, followed by a deconvolution layer of upsampling to generate pixel dense outputs.

In the figure 10, the prediction made from FCN-32s performs badly due to the limitation of the scale of detail in the upsampled output. It is caused by the information loss when recover the image from upsampling process. Therefore, we need to append more details to the result. It can be done by combining the output from fine layers and coarse layers together which is often called *skips*. This combination can make the local prediction while keeping the global structure. The main idea is dividing the output stride in half by predicting from a 16 pixel stride layer. The modification of architecture for generating FCN-16s is shown in the second row of figure 10. By doing this, the learning rate decreases by a factor of 100. The authors continue to do this fusion lower level process to produce the FCN-8s architecture which gives minor improvement. The modification process is shown in the third row of figure 10. FCN-8s is the final fusion layer of the architecture because the authors show that further fusion does not improve but harm the precision of the networks. Therefore, FCN-8s is the saturated architecture of fully convolutional networks in semantic segmentation task.

3 Our solution to semantic segmentation task

We inherit the networks structure of fully convolutional networks for semantic segmentation task while improving its performance on outliner images. In fact, we use the output from dense prediction of fully convolutional networks as a mask to further coloring the regions in image for each class. We cover the case that fully convolutional networks architecture fails to do segmentation on the correct class but generates wrong classification segmentation.

3.1 Dataset and models

We choose to work on PASCAL-VOC 2007 dataset and inherit the pre-trained models of PASCAL-VOC dataset in our both proposed method. As the limitation in our hardware of computer, we cannot run all the dataset to perform specific evaluation but show the visualization of the obtained result.

3.2 Our contribution

- Propose solution to effectively segmentation overlapping objects of many classes in an image where fully convolutional networks used in semantic segmentation fails to properly generates correct class segmentation, especially on overlapping objects in an image.
- Propose semantic segmentation preprocessing of object detection using Faster RCNN to corroborate classes segmentation result.
- Propose to use fully convolutional networks result as mask to color the objects in image.
- Propose solution to color overlapping regions of objects in image.

3.3 Problem of Fully Convolutional Networks for semantic segmentation

Fully Convolutional Networks perform well on object segmentation. It correctly determines the regions where objects appear but give bad result on class segmentation. Figure 11 shows the outliners of fully convolutional networks method that we discover in the process of studying that architecture.

As in the figure 11, first row of image is the original image. The second row is the result

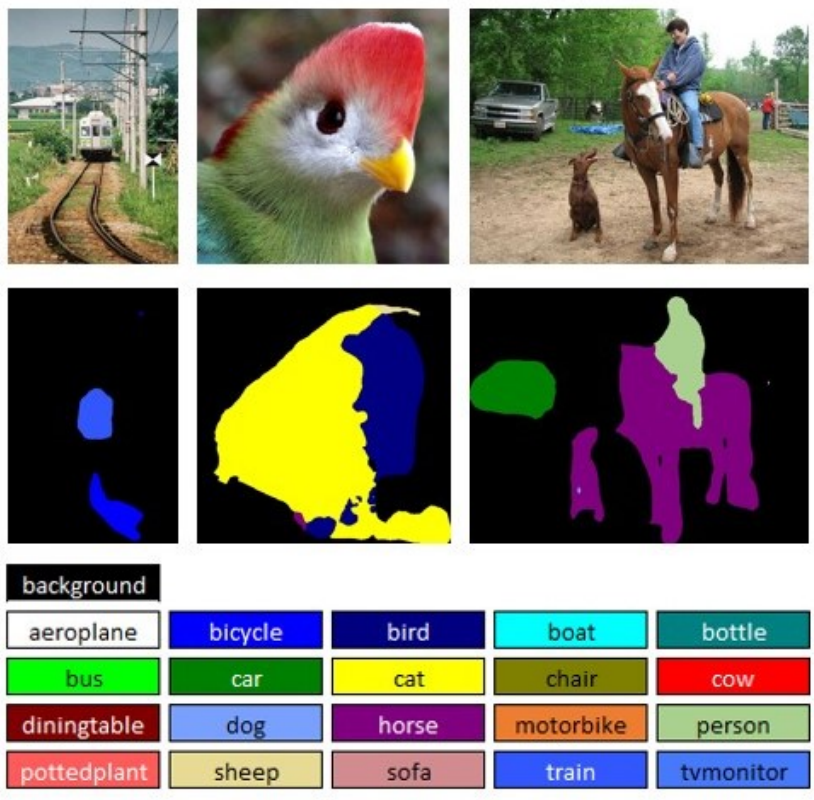

Figure 11: Fully Convolutional Networks Problems

of Fully Convolutional Networks for Semantic Segmentation. As intuitive results in figure 11, fully convolutional networks execute classes segmentation badly. There are many noises in the image which lead to this bad performance. The method not only mis-classifies the object but also generates the wrong segmentation regions of the object. In the next section, we present our improvement for semantic segmentation of classes by combining object detection networks with fully convolutional networks along with our proposed region coloring method which can improve the class segmentation significantly.

3.4 Our proposed solution for semantic segmentation

Due to the shortage of resources and convenience of each step result visualization, we decide to keep the two processes split instead of merging them into one code only. Note that the two process of Object detection using Faster RCNN and Fully Convolutional Networks mask exploitation are executed independently because the whole original image is used in the two processes without waiting for any result from one of the mentioned processes.

3.4.1 Object detection using Faster RCNN

In order to properly execute class segmentation, we use Faster RCNN to perform object detection so as to locate the bounding boxes of the objects in the image and their belonging classes.
The obtained result from this steps is the top-left coordinate, width, height of the bounding box and class of every detected objects in the image. This result of this process is saved to folder ${PROJECT_PATH}/data/result/bbox with its file name equals to {IMAGE_NAME}.bbox under the following structure:

{class_1}:x0 y0 width0 height0 x1 y1 width1 height1 ...
{class_2}:x0 y0 width0 height0 x1 y1 width1 height1 ...
...

The visualization result of this step is crop images which are shown in figure 12. The purpose of this process is to localize the position where the object appears in the image to focus on do segmentation on that region only after we have obtained the mask from Fully Convolutional Networks.

Figure 12: Cropped-out detected object after Faster RCNN process

3.4.2 Fully Convolutional Networks mask exploitation

In this process, we use the result map from Fully Convolutional Networks as mask for coloring. The map resulted from this process is a 2-dimension matrix which is $w \times h$ size where w and h are the width and height of the original image respectively. The mask contains the raw indices of the detected regions of classified objects in input image. But we do not use them for coloring because they may output the wrong segmentation result.

3.4.3 Our proposed region coloring method

This is our important step in our proposed solution. We make use of the results of the two preprocessing steps of object detection and mask exploitation. Combining two results, we proposed an effective method for region coloring to do semantic segmentation by classes efficiently. We use the bounding box result of the object detection process to localize the region that we need to color the object. However, the bounding box is a rectangle, therefore, we would like to use mask obtained from Fully Convolutional Networks to focus on coloring the precise boundary of the object. However, we get two problems with our proposed method:

1. The order to color overlapping regions of the bounding boxes.
2. Assign class color to result image corresponding to mask's indices due to the failure of classification of Fully Convolutional Networks

We present our solution to address those two mentioned problems and would like to divide it into two subprocesses: find other classes' overlapping regions of the current processed bounding box and determine the segmentation color to the mask's indices to the remaining regions after eliminating all other classes' overlapping regions.

Find other classes' overlapping regions of the current processed bounding box

For each processing bounding box, we know its class but do not know which indices of mask to assign class color. We think of counting the indices and get maximum value to assign class color but in mathematics point of view, this approach may get into problem when there exists a region where other classes take over a large part of the image. Therefore, we come up with an idea of eliminating the overlapping regions of current bounding box of its class with other classes' bounding box. We find the overlapping regions and return both the non-overlapping part and the list of overlapping regions to the main process.

Determine the segmentation color to the mask's indices to the remaining regions after eliminating all other classes' overlapping regions

After eliminating all overlapping regions and obtaining the non-overlapping section of the image, we start to count the index and assign class color to the index which appears the most frequently in the remaining mask region. This approach is mathematically reasonable owing to the localization of object detection. The remaining part of the bounding box which contains the detected object must have the class's index appeared the most frequently because it is the exact location of the object.

Note that we can do this because the mask obtained from Fully Convolutional Networks is good enough for further improvement by voting index method. The noises in the mask are not considerable and the object takes over a large regions in its detected location. That is why how our method perform significantly well this segmentation task. However, there are some outliners in our approach when the non-overlapping region contains a small number of the exact detected object. But in most of the cases, our method perform well.

3.4.4 Result visualization for comparison

In this section, we present visualization of our result for intuitively comparison between our work and Fully Convolutional Networks for semantic segmentation approach. Figure 13 shows the result of our segmentation compared to the one from Fully Convolutional Networks approach.

As in the figure 13, first row of image is the original image. The second row is the result of Fully Convolutional Networks for Semantic Segmentation. The third row is our result from our proposed system.

4 Project Deployment

4.1 Our project structure

Our project structure includes important folders in source folders:

- **data**: contains the data of Faster RCNN models saved in subfolder faster_rcnn_models, Fully Convolutional Networks models saved in subfolder voc-fcn8s, testing images saved in subfolder images, execution result saved in subfolder result.

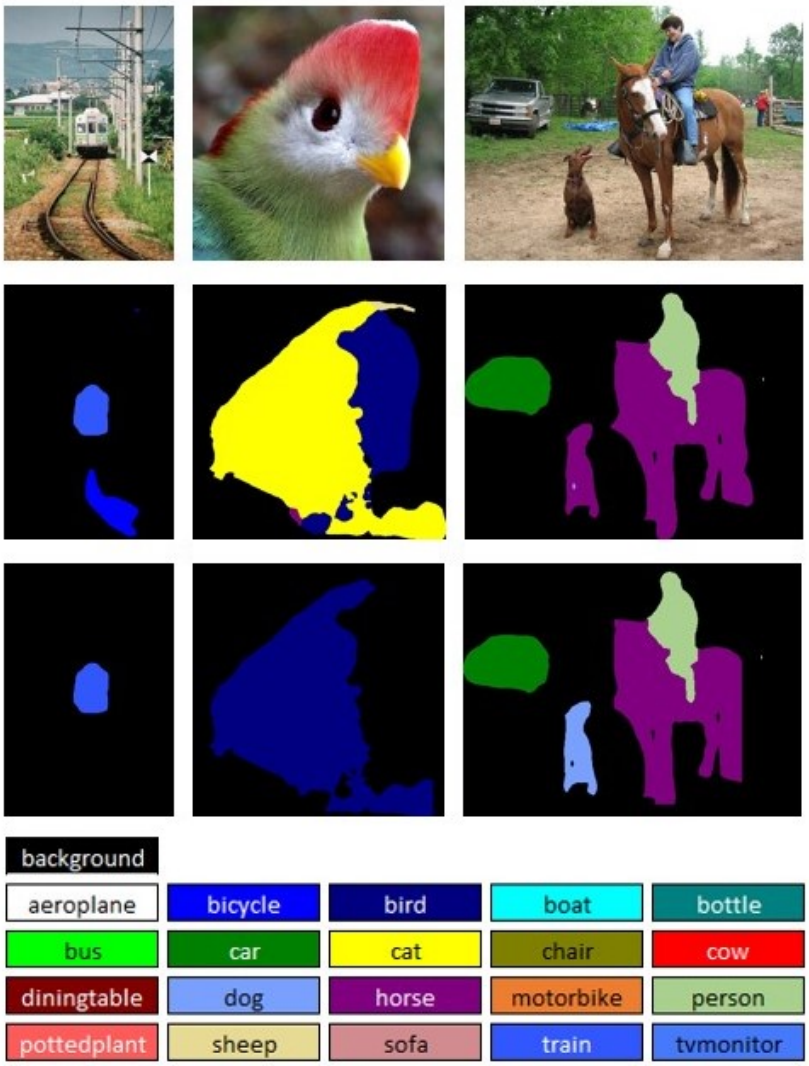

Figure 13: Our proposed method result

- **src**: contains the source code of our project
- **models**: contains the architecture of Faster RCNN to load in running process.
- **caffe**: caffe framework for Fully Convolutional Networks code.
- **caffe-fast-rcnn**: caffe framework for Fast RCNN code.
- **lib**: extra library for Faster RCNN code.

4.2 Caffe deployment

Download caffe library from two repositories of Faster RCNN and Fully Convolutional Networks respectively as below:

- https://github.com/rbgirshick/py-faster-rcnn.
- https://github.com/shelhamer/fcn.berkeleyvision.org.

Following the instruction from the main site of caffe: http://caffe.berkeleyvision.org/installation.html.

4.3 Run our system

1. Download pre-trained model into folder /data/faster_rcnn_models by executing file faster_rcnn_models/fetch_
2. Download pre-trained model into folder /data/voc-fcn8s by following link in file caffemodel-url.
3. Put images in data/images folder and clean the result in data/result/bbox, data/result/detected_objects, data/result/segment_result
4. Execute frcnn_detect.py by running $ python src/frcnn_detect.py
5. Execute segment.py by running $ python src/segment.py

The result is saved in /data/result/segment_result. You can see the result of each process in each folder in /data/result.

References

[1] Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun - "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks".

[2] Jonathan Long, Evan Shelhamer, Trevor Darrell - "Fully Convolutional Networks for Semantic Segmentation"