LAB REPORT FOR CHRISTOPHE BOBDA'S GROUP

WEEK 4&5

Xianhan Li

INTRODUCTION

This report includes 3 tasks: In Task1, I have learned how to use opencv_dnn module for image classification by using GoogLeNet trained network from Caffe model zoo; In Task2, I have learned to perform Object Detection and Instance Segmentation using Mask-RCNN with OpenCV; In Task3, I have run the single person pose estimation using OpenCV.

TASK1 (From *Going Deeper with Convolutions 2015 https://www.cv-foundation.org/openaccess/content_cvpr_2015/papers/Szegedy_Going_Deeper_With_2015_CVPR_paper.pdf)*

GoogLeNet is a CNN deep learning method proposed by Google in 2014. It won the 2014 ILSVRC championship, and its error rate was lower than that of VGGNet at that time. It was published on CVPR in 2015. GoogLeNet is also known as Inception v1. Because later it also proposed a lot of improved versions, including v2, v3, v4 and so on.

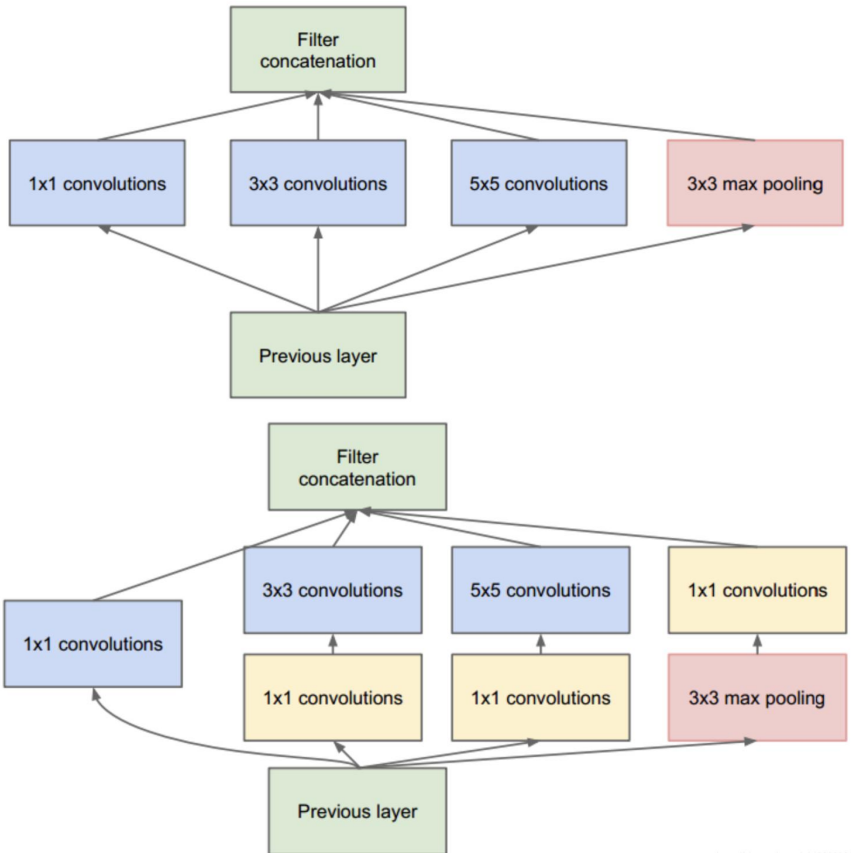
The main goal of GoogLeNet is to increase the width of the network and reduce the number of parameters. From the results, GoogLeNet uses the inception structure to form a 22-layer huge network, but its parameters are much lower than the previous AlexNet network. It is a very good CNN structure.

Using a deeper and wider network is a very safe solution, but there are two important problems: one is that a larger network means more parameters, making the network easier to overfit; the other problem is a larger network need more computing resources. The ultimate way to solve these problems may be to move from a fully connected network to a more sparsely connected structure, even inside the convolution. The work of Arora proves that if the distribution of the data set is represented by a large sparse deep neural network, then the optimization work can be done by analyzing the statistical correlation between the activation function of the nearest layer and the highly correlated neurons between the outputs' properties to build layer by layer. Although this proof requires strong assumptions, it resonates with the famous Hebbian theory: neurons that fire together, wire together. Therefore, even if the conditions are not strictly met, similar principles can be applied in practice.

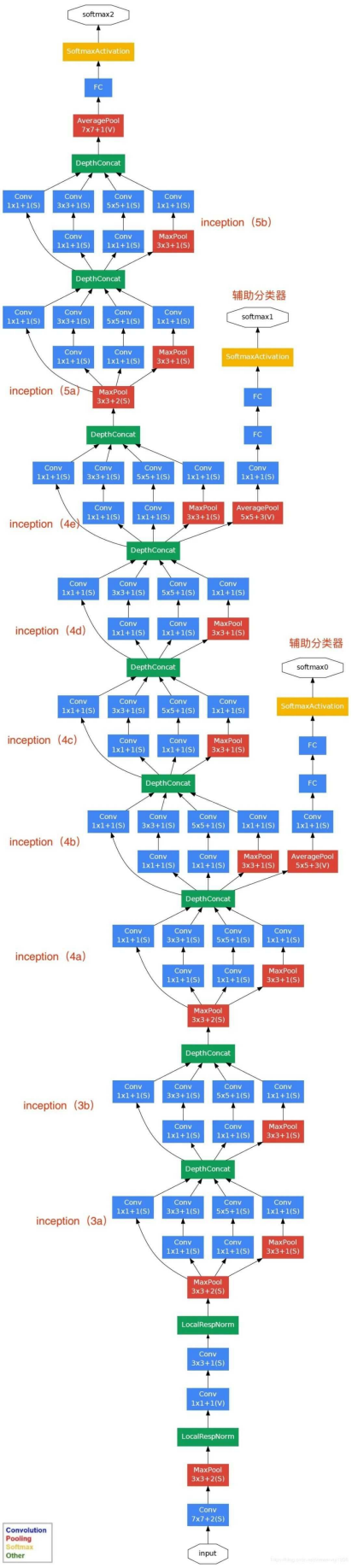
The main idea of the Inception architecture is how the optimal local sparse structure in a convolutional network can be approximated and covered by easily obtained dense components. Inception networks are built from convolutional layers. All the work is to find the optimal local structure, and repeat. Arora believes that a layer-by-layer construction network should first analyze the correlation statistics of the nearest layer, and then group highly correlated units into the same clusters, and these clusters form the units of the next layer, and connected to the units of the previous layer. This paper assumes that each unit of the previous layer is related to some region of the input image, and these units should be divided into a set of filters (filter banks). In lower layers (closer to the input layer), these units are concentrated in local regions. This means

that in the end we will end up with many clusters, all of them focusing on the same region, and they can all be covered by a 1x1 convolution in the next layer.

The structure of an inception:



The combination of many Inceptions finally forms GoogLeNet. The network structure of GoogleNet based on Inception is as follows, with a total of 22 layers.


softmax2
SoftmaxActivation
FC
AveragePool 7x7+1(V)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
MaxPool 3x3+1(S)
inception（5b）
辅助分类器
softmax1
inception（5a）
MaxPool 3x3+2(S)
inception（4e）
AveragePool 5x5+3(V)
inception（4d）
softmax0
inception（4c）
inception（4b）
inception（4a）
inception（3b）
inception（3a）
LocalRespNorm
Conv 3x3+1(S)
Conv 1x1+1(V)
Conv 7x7+2(S)
input
Convolution
Pooling
Softmax
Other

Here is my implement of GoogLeNet:

Step1: Load model, labels and image.

Step2: Remove the alpha channel of the image to match the RGB format expected by the model input, if it has one.

Step3: Use the “cv.dnn.blobFromImage()” function to convert the image into the Blob format required by the model. A Blob is a special data structure used to represent image data in deep learning. Here, we resize the image to 224x224 pixels and perform mean subtraction on pixel values.

Step4: Input the preprocessed Blob into the model for inference.

Step5: Use “np.argmax()” to find the index with the highest probability in the output vector, which corresponds to the predicted class.

Step6: Extract the probability value corresponding to the predicted class from the model's output, and print them out.

The codes of this task are in the file **“T4_task1.py”**.

Result of Implement:

```
Best prediction: church, church building
probability: 0.960027
```

```
Best prediction: analog clock
probability: 0.503636
```

```
Best prediction: bell cote, bell cot
probability: 0.453668
```

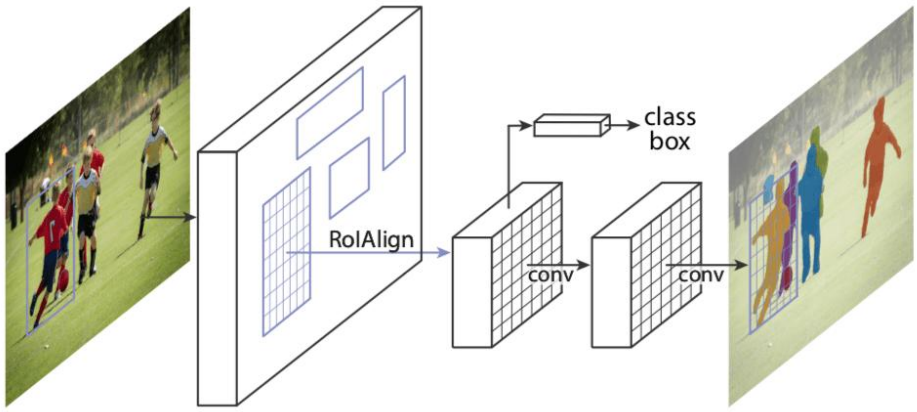
TASK2 (From https://www.learnopencv.com/deep-learning-based-object-detection-and-instance-segmentation-using-mask-r-cnn-in-opencv-python-c/)

Mask-RCNN is a result of a series of improvements over the original R-CNN paper (by R. Girshick et. al., CVPR 2014) for object detection. R-CNN generated region proposals based on selective search and then processed each proposed region, one at time, using Convolutional Networks to output an object label and its bounding box.

Fast R-CNN (R. Girshik, ICCV 2015) made the R-CNN algorithm much faster by processing all the proposed regions together in their CNN using a ROIPool layer.

Faster R-CNN (S. Run et al., PAMI, 2017) pushed it even further by performing the region proposal step too using a ConvNet called Region Proposal Network(RPN). Both the RPN, and the classification and bounding box prediction network worked on common feature maps, thus making inference faster. On a GPU, Faster R-CNN could run at 5 fps.

Mask R-CNN (He et al., ICCV 2017) is an improvement over Faster RCNN by including a mask predicting branch parallel to the class label and bounding box prediction branch as shown in the image below. It adds only a small overhead to the Faster R-CNN network and hence can still run at 5 fps on a GPU.



The Mask-RCNN network has two major parts: The first one is the Region Proposal Network which generates around 300 region proposals per image. During training, each of these proposals (ROIs) go through the second part which is the object detection and mask prediction network, as shown above. Note that since the mask prediction branch runs in parallel to the label and box prediction branch, for each given ROI, the network predicts masks belonging to all the classes.

During inference, the region proposals go through Non-Maximum Suppression and only the top scoring 100 detection boxes are processed by the mask prediction branch. So, with 100 ROIs and 90 object classes, the mask prediction part of the network outputs a 4D tensor of size 100x90x15x15, where each mask is of size 15×15.

Here is the implement of Mask-RCNN:

STEP1: Load the model and classes.The file mscoco_labels.names contains all the objects for which the model was trained. We read class names. Then we read and load the colors.txt file containing all the colors used to mask objects of various classes. Next, we load the network using two files: frozen_inference_graph.pb, mask_rcnn_inception_v2_coco_2018_01_28.pbtxt.

STEP2: Read the input. In this step we read the image, video stream or the webcam. In addition, we also open the video writer to save the frames with detected output bounding boxes.

STEP3: Process each frame. Transform the input image into a blob format using the blobFromImage function and fed to a neural network. The network processes the blob, predicts bounding boxes and masks, and filters out low-confidence boxes. Save the final image with bounding boxes and masks, and show the inference time.

STEP4: Post-processing the network’s output. The network’s output masks object is a 4-dimensional object, where the first dimension represents the number of detected boxes in the frame, the second dimension represents the number of classes in the model and the third and

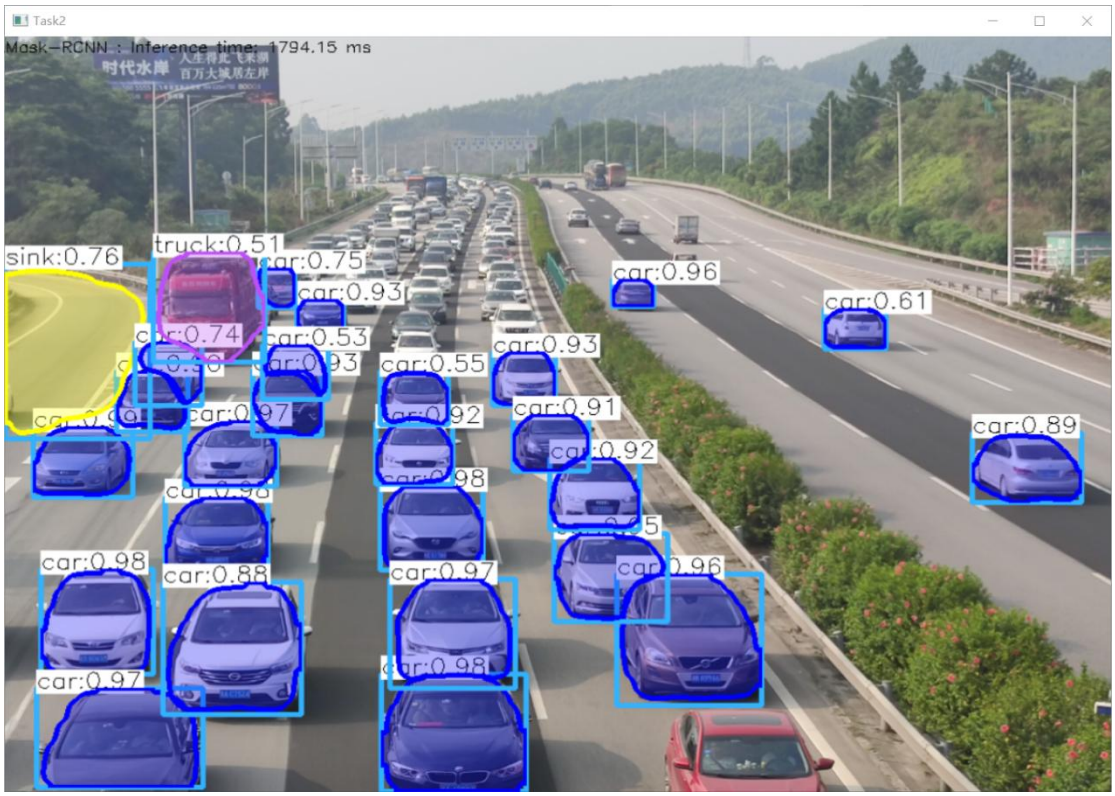
fourth dimensions represent the mask shape(15×15) in our example.If the confidence of a box is less than the given threshold, the bounding box is dropped and not considered for further processing.

STEP5: Draw the predicted boxes. Draw the boxes that were filtered through the post-processing step, on the input frame with their assigned class label and confidence scores. We also overlay the colored masks along with their contours inside the boxes.

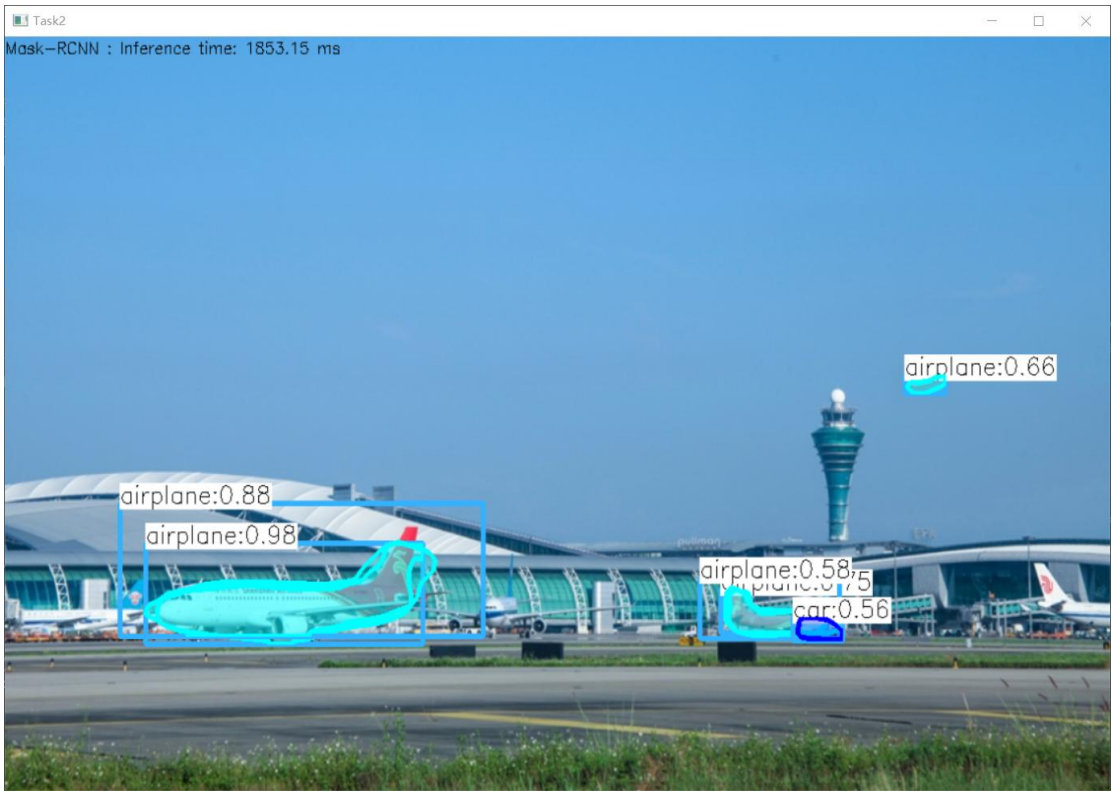
The codes of this task are in the file **“T4_task2_image.py”** and **“T4_task2_video.py”**.

Results of Implement:

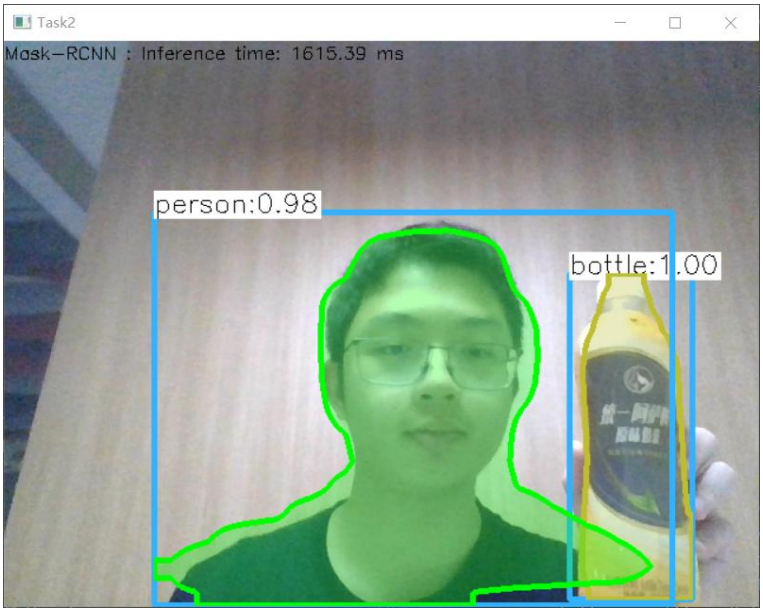
Vehicle detection:



Airplane detection:



Video: (Mask_RCNN_output.avi)



Task3 (From https://www.learnopencv.com/deep-learning-based-human-pose-estimation-using-opencv-cpp-python/)

Deep Learning based Human Pose Estimation using OpenCV. We have used Caffe model that won the COCO keypoints challenge in 2016.

Pose Estimation is a general problem in Computer Vision where we detect the position and orientation of an object. This usually means detecting keypoint locations that describe the object.

We have used MPII Human Pose Dataset to solve this problem. MPII Human Pose dataset is

a state of the art benchmark for evaluation of articulated human pose estimation. The dataset includes around 25K images containing over 40K people with annotated body joints.

MPII model outputs 15 points: Head – 0, Neck – 1, Right Shoulder – 2, Right Elbow – 3, Right Wrist – 4, Left Shoulder – 5, Left Elbow – 6, Left Wrist – 7, Right Hip – 8, Right Knee – 9, Right Ankle – 10, Left Hip – 11, Left Knee – 12, Left Ankle – 3, Chest – 14, Background – 15.

Here is the implement of Pose Estimation:

STEP1: Load Network. We are using models trained on Caffe Deep Learning Framework. Caffe models have 2 files: pose_deploy_linevec_faster_4_stages.prototxt and pose_iter_160000.caffemodel.

STEP2: Read Image and Prepare Input to the Network. The input frame that we read using OpenCV should be converted to a input blob (like Caffe) so that it can be fed to the network. This is done using the blobFromImage function which converts the image from OpenCV format to Caffe blob format. The parameters are to be provided in the blobFromImage function. First we normalize the pixel values to be in (0,1). Then we specify the dimensions of the image. Next, the Mean value to be subtracted, which is (0,0,0). There is no need to swap the R and B channels since both OpenCV and Caffe use BGR format.

STEP3: Make Predictions and Parse Keypoints. Once the image is passed to the model, the predictions can be made using a single line of code. We check whether each keypoint is present in the image or not. We get the location of the keypoint by finding the maxima of the confidence map of that keypoint. We also use a threshold to reduce false detections.Once the keypoints are detected, we just plot them on the image.

STEP4: Draw Skeleton. Since we know the indices of the points before-hand, we can draw the skeleton when we have the keypoints by just joining the pairs.

Results of Implement:

Video: (output.avi)

Conclusion

By completing several assignments of this week, I have got deeper comprehension of GoogleLeNet, Mask RCNN and Pose Estimation. Since I am not very familiar with principles of these algorithms and methods in OpenCV package, I have looked up many information to finish my assignments, which helps me learn a lot about computer vision. In this process, I find it hard but interesting to use these amazing datasets to detect different features and items from images and videos. I am very willing to learn more about this area.