3rd Solution to the CVPR 2018 WAD Video Segmentation Challenge

Wudi Wang (wwoody827@gmail.com)

Overview

This is my solution to the Kaggle CVPR 2018 WAD Video Segmentation Challenge(https://www.kaggle.com/c/cvpr-2018-autonomous-driving). The goal of this challenge is to segmentation on instance level to objects like cars and persons in a typical city road. My solution for this challenge is based on Mask-RCNN(https://arxiv.org/abs/1703.06870), one of the state-of-art instance segmentation architecture. My final submition reaches 3rd place in the private leaderboard.

Dataset

The image resolution is 3384x2710 (from 4k video frames). The targets for this challenge are 33: 'car', 34: 'motorbicycle', 35: 'bicycle', 36: 'person', 38: 'truck', 39: 'bus' and 40: 'tricycle'. Due to the postion of camera, objects of interest will only localize in the lower half of the images (you will not expect a car in the sky, unless it's owned by Arthur Weasley). So during training and prediction, I first performed a ROI cropping to the original images to cut the top half off.

Model

My solution is based on Mask-RCNN. A key observation is that many of the objects are very small. So in contrast to original implementation of Mask-RCNN, where all images has to resized and padded into a fixed size, model used in my solution and take images with different input size. This can be done since the RCNN is a fully connceted network and can take images with various size.

Due to the limit of graphics card memory, the original image cannot be fitted into one card even after ROI cropping. So during traning I randomly cropped image into smaller patches, with size (1024, 1024). The random cropping also helps the model from overfitting. During prediction, the model takes ROI cropped image as input and the prediction is padded to its original size

Mask RCNN

I started with a keras implement of Mask RCNN (https://github.com/matterport/Mask_RCNN). Their great work is very helpful for me to start this challenge. The initial weights are from matterport Mask RCNN, which has ResNet 101 backbone and pretrained on MS COCO. Training toke around 4 or 5 days on my GTX 1070. I also got access to P100 on Google cloud platform with some free credits.

Later on I switched to pytorch implement of Mask RCNN(https://github.com/multimodallearning/pytorch-mask-rcnn) because I am more familiar with pytorch and this will allow me to do further modification to the model. I tried several ways to

improve model performance.

Backbone

I tried to improved model performance by chaning backbone from resnet101 to a more powerful one, like se-resnext101 or resnext101. The init backbone weights are from imagenet and I trained other layer from scratch. Both of them yeild similar results with resnet101 backbone. All backbone weights used in this experiment are from (https://github.com/Cadene/pretrained-models.pytorch)

RPN scale

The original implement uses RPN_ANCHOR_SCALES = (32, 64, 128, 256, 512). To capture small objects, I changed it to (16, 32, 64, 128, 256). This made results worse. The possible reason is that the "small" objects is small relative to very large original image. Each instance still has pixel counts at least 100 or more. So using smaller rpn scale is not necessary.

Input image resize

Since the largest Anchor is 512, a large object, such as a bus, cannot be captured as one object and often been split into multiple instances. Thus I tried to resize image to 50% of its original size and rescale predictions back. This does not improve results (and actually make ensemble worse). The reason probably is objects so large is just a corner case, and resized image will cause a lot of some objects been merged into one and cause additional difficulty for ensembling.

Mask shape

One observation of prediction results is that large objects, like cars nearby, always have more complex edges, while mask branch only has size (28, 28). By adding additional upsampling layer to mask branch and changing mask branch to (56,56), model can be more capable of capturing details for large objects.

Post processing and ensembling

Post processing

The post processing contains several steps. First some predicted instances with very few pixels were filtered out by using a pixel threshold. Then, predicited objects with low confidence were filtered out to keep total number of predictions low (otherwise evaluation on kaggle server will time out). In addition, predictions often overlap with each other. So when converting predictions to rle, objects with low confidence with be converted first and objects with high confidence with be converted later to make sure high confidence objects with not be covered by low confidence objects, and the overlapping area will be assigned to high confidence objects.

In addition, I also tried to use traditional image process tools for post processing. One observation is that the predicted objects sometimes have holes or broken parts. To fill the holes and connect broken parts, I used skimage morphology library (binary_closing and/or binary_opening.) This sometime improve results by a little.

Ensembling

Ensembling is also an important way to reach high score. My final submission is an ensemble of 4 different predictions from 3 different models. During ensembling, I only keep objects that have been predicted by at least two models to lower false postive. For overlapping objects, I keep the one with highest confidence. I determine two objects in two predictions to be a same object if their IOU is larger than a preset threshold. My final submission is an enseble of two resnet with mask shape of (28, 28) and a resnet_v2 with mask shape of (56, 56). Test time augmentation (TAA) of horizonal fliping were used for resnet.

Further work

I did not use time information in my final model. I tried with optical flow but did not have enough time to finish. So instead of "video instance segmentation", my solution is still based on single frame. Hope to see a solution that make use of true "video"!