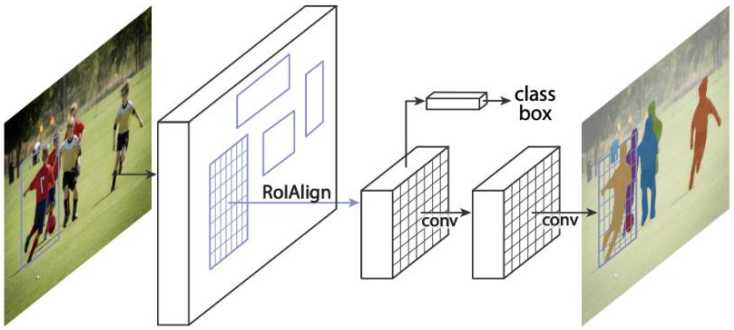
Lane Detection with Mask R-CNN

Aim: To detect Road Lane lines and Car using Mask R-CNN model (Regional-Convolutional Neural Network).

Introduction: This is a straightforward example on how to implement a simple object detection and segmentation system with Mask R-CNN. This documentation will cover the whole pipeline which includes model introduction, data collection, annotation, coding, training and testing. To make it understanding, this implementation illustrates 2 class object detection which is the lane (marks) and car.

Mask R-CNN: Mask R-CNN itself is a modification to R-CNN and Faster R-CNN to detect objects. The ability to perform image segmentation is done by adding a fully convolutional network branch onto the existing network architectures. So while the main branch generates bounding boxes and identifies the object's class the fully convolutional branch, which is fed features from the main branch, generates image masks.



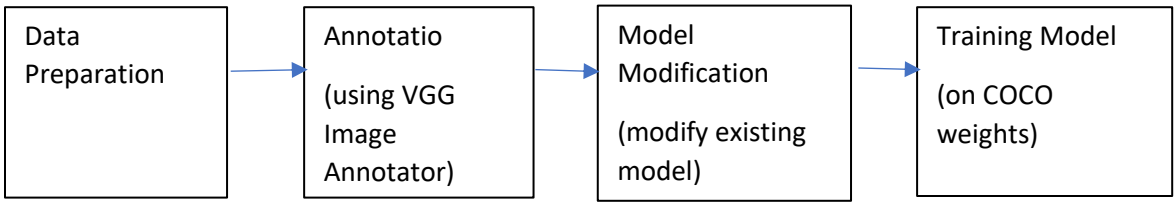
Another interesting thing on Mask R-CNN is RoIAlign. RoIAlign is not using quantization for data pooling. It provide higher precision than RoIWarp and RoIPool.

	align?	bilinear?	agg.	AP	AP_{50}	AP_{75}
RoIPool			max	26.9	48.8	26.4
RoIWarp		✓	max	27.2	49.2	27.1
		✓	ave	27.1	48.9	27.1
RoIAlign	✓	✓	max	30.2	51.0	31.8
	✓	✓	ave	30.3	51.2	31.5

So basically Mask R-CNN is a two stage convolutional network which serves object detection and object masking.

Overview: The below pipeline is to show how to build the model up from scratch. Firstly, data is important. Without proper data, no model can be trained for good results. The data source used in this implementation is collected from google.

Annotation is a time costing part. In layman's terms, you need to 'teach' the model how the object you want looks like, before it can recognize. Thus, the annotation is done frame by frame manually. In this demonstration, the lane marks are four edged polygons which is relatively easy to annotate. When it comes to a more edged or irregularly shaped object, the annotation will require more time and patience.



Model modification and training is based on pretrained coco weights. Also, you may train from scratch, which may require more data and training time. The model will download coco weights itself during the first launch. It gives the model a general impression on the various objects. You may find more information on coco dataset.

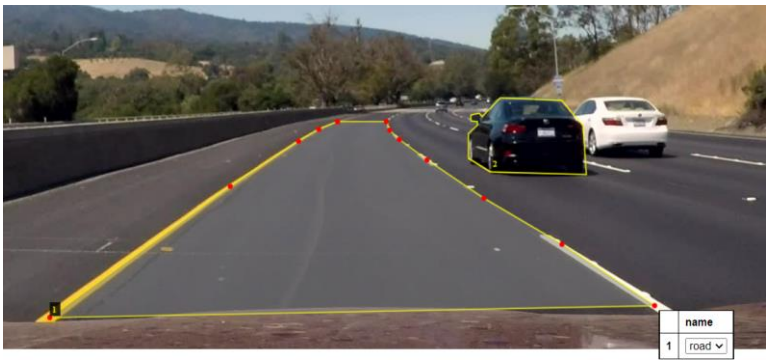
Data Collection and Annotation:

Data collection actually can be done in different ways. You may record your own dashcam video, use existing video or even collect frames/photos. The key things are variety and quantity. So I downloaded the images of road lanes from google.

For annotation I used **VGG Image Annotator** which is an easy-to-use interface for object detection and image segmentation annotation.

The major change here for me was just having to make sure each polygon region is labeled with the class name by using the Region Attributes in the annotator. A JSON file will be generated to record down all the annotated point coordinates belonging to each image.

As you can see from below screenshot, the annotation is quite straightforward. Highlight polygon then annotate out the objects in the image. There is a sequence if you annotate more than one objects within same class.



After annotation is done, export annotations as JSON. The JSON file will be read by the model to recognize the objects. Also, you may re-organize the JSON file by removing the header and put the content in structure to ease the reading.

Training and Coding:

There are some requirements needed.

1. Numpy
2. Scipy
3. Pillow
4. Cython
5. Matplotlib
6. scikit-image

7. tensorflow==1.13.1
8. keras==2.0.8
9. opencv-python
10. h5py==2.10.0
11. Imgaug

- **Download weights:**
  To speed up the training, it is recommended to train from ***coco weights***. The program will automatically download the coco weights if you start a fresh training.

You can see my code in Lane Detection.