PEOPLE DETECTION RESEARCH

1- Introduction

The global project is to study the feasibility of a concrete 3D printer attach to a crane to create buildings from it. The printhead will be equipped with a vision captor to control its position and the human activity under it.

Our part in this project is the detection of human activity under the printhead from the vision captor to guarantee the safety of workers on the building site.

There are many methods to detect people in a video. Two cases are to be expected: when the camera is static and when it is mobile. Since the vision captor will be at the end of a wire attached to a crane, we will treat the case where the camera is mobile. In this case, the most used methods are the one which works with detectors learned from a base of annotated images.

To achieve the goal of this project, we will use a method like Rcnn, based on artificial intelligence, implemented from the TensorFlow library.

2- TensorFlow library

TensorFlow is an open-source software library used for machine learning application such as neural networks. It was developed by the Google Brain team for internal Google use and was released as an open-source library on November 9, 2015. Since then it has become a reference in Deep Learning for several reasons:

- Multi-platform of development (Linux, Mac OS, Android, iOS)
- Python, C++ and Java APIs
- Short compilation time
- Support CPU and GPU calculations

It is currently used by Google in all its principal products: Gmail, Google Photos, Voice recognition, …

TensorFlow represents the calculations in the form of an execution graph, each node represents an operation to be performed, and each link represents a tensor.

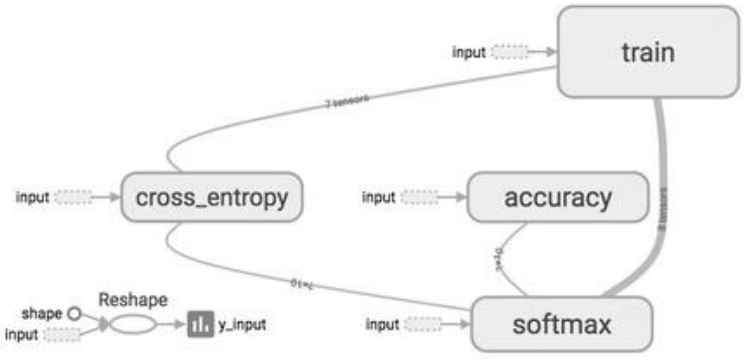

Figure 1 TensorFlow's graph

An operation can range from a simple addition to a complex matrix differentiation function. Each operation takes zero input, one or more tensor, performs a calculation, and returns zero, one or more tensor.The creation of the graph is automatically managed by TensorFlow once the tensors and operations are implemented and instantiated. This allows optimization and parallelization of code and execution at launch.

One of the fundamental properties of TensorFlow is to start the graph in a session. A session places the operations of the graph in devices like CPU or GPU and provides methods to execute them. This avoids going back in the development world (Python) at every step and allows executing all graph operations at once in the same optimized backend.

TenserFlow has extensive support for creating Deep Learning specific operations, making it easy to build a neural network and use the associated mathematical operations to train it with the right optimizers.

Next, we will see how two Tensoflow's object detection models which are Faster R-CNN and SSD work and how we started to use the second one for our project.

3- Object detection models

Faster R-CNN is a model used in object detection. It consists of applying RPN algorithm (Region Proposal Network) in order to detect regions in which there is objects. The objects found will be classified and box bounded using some algorithms like SVM and linear regressor. Faster R-CNN is an extension of its two previous models: R-CNN and Fast R-CNN. We will explain those three models and specify the differences between them as well as the SSD model.

R-CNN:

The R-CNN (Region-based Convolution Neural Network) follows three steps:

- The first step consists of extracting proposals regions from the input image. A proposal region is a region on the image that have a high probability to contain an object.

  To extract the proposals regions, the input image is scanned using an algorithm called selective research.

- The second step is to run the CNN on every proposal region in order to extract objects.

- The third step is to classify the CNN outputs using SVM and tightening the bounding box of the object.

The idea is that we first extract proposal regions, and then extract features. The inconvenient of this method is basically time of testing each image which is very slow. Applying the CNN on 2000 proposal region takes about 50 seconds per image.

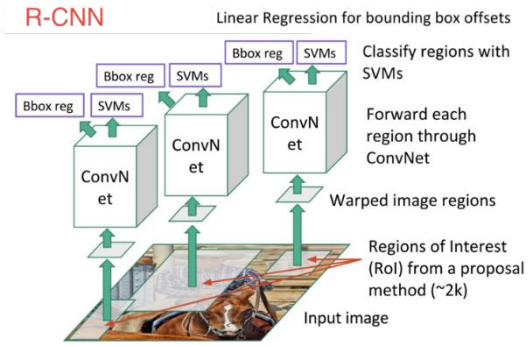

Figure 2 3steps of R-CNN

Fast R-CNN:

Fast R-CNN improves the test time of its predecessor by going from 50 seconds per image to 2 seconds per image. This method consists of two steps:

- Extracting features from images before dividing input images into proposal regions and running only one CNN over the entire image.
- Replacing the SVM by a SoftMax layer that gives the class of each object directly.

Therefore, this model allows extracting regions from the CNN output, not from the input image.

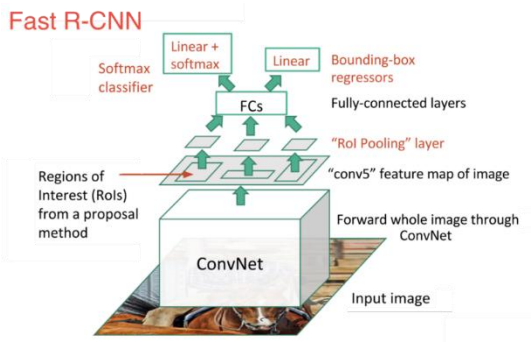

Figure 3 Fast R-CNN steps

Faster R-CNN:

The purpose of the Faster R-CNN is to apply a fast-neural network instead of using the selective search algorithm. It uses a Region Proposal Network algorithm. The RPN algorithm consists of the following steps:

- The input image goes through a convolution network which will give convolution feature maps.

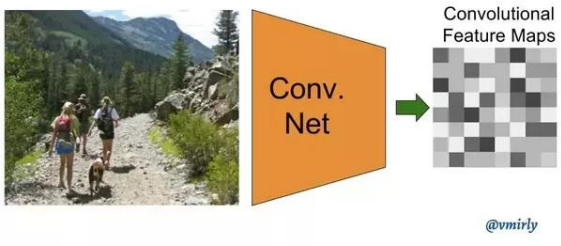

Figure 4 Convolution Feature Maps extracted from the input image

- Then it run a sliding window with a size of **(3 x 3)** on those future maps. For each sliding window a set of 9 anchors are generated which all have the same center(x_a, y_a). The 9 anchors have 3 different ratios and three different scales:

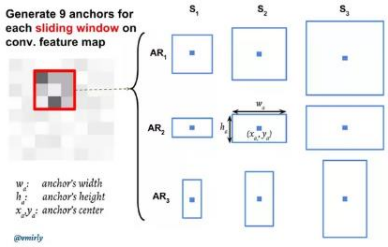

Figure 5 Anchors generated by the sliding window

- For each anchor, it calculates the IoU parameter which is:

$$IoU = \frac{Anchor \cap GTBox}{Anchor\ U\ GTBox}$$

Where GTBox is the box containing an object

- After that, it attributes to each anchor a value calculated as below:

$$\begin{cases} p = 1\ if\ IoU > 0.7 \\ p = -1\ if\ IoU < 0.3 \\ p = 0\ otherwise \end{cases}$$

- Finally, the spatial features extracted from convolution maps are transmitted to a network that have two tasks: classification and regression. The regressor determines a predicted bounding box by its coordinates: its center (x,y) and its width and height (w,h). The classification indicates if the predicted bounding box contains an object or not.

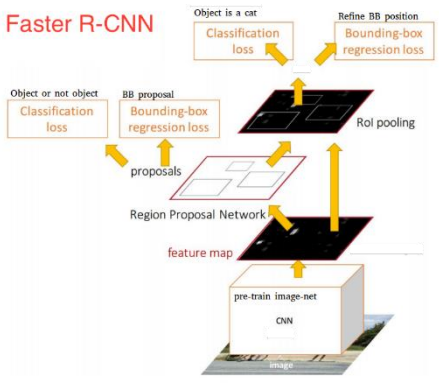

Figure 6 Faster R-CNN steps

SSD:

SSD (Single Shot Multibox Detector) is a deep neural network that skips resampling pixels or features for bounding box purpose, and yet is still accurate as other networks that do it. SSD offers both speed and accuracy to be used in real time application.

Method	mAP	FPS
Faster RCNN [2] (VGG16)	73.2	7
YOLO [5]	63.4	45
SSD300	72.1	58
SSD500	75.1	23

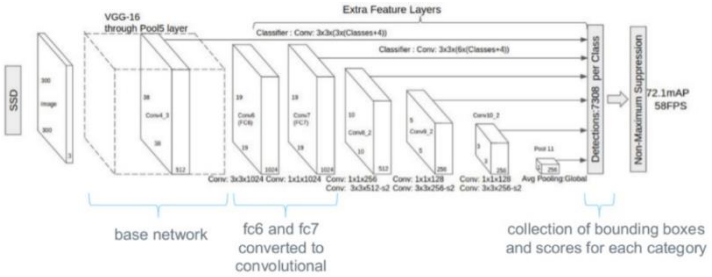

Figure 7 SSD steps

The input image goes through multiple convolutional layers of different size, the next layer being smaller than the previous, to allow prediction in different scale. For each feature map, SSD put default bounding boxes. Each box contains the bounding box offset and class probabilities. Among these predicted boxes, SSD labels "positive" boxes with the highest match of the ground truth.

As mentioned, each feature map yields a set of default bounding boxes, resulting in many bounding boxes flagged as negative examples. To counter this issue, SSD uses non-maximum suppression to group very similar bounding boxes into one box.

4- Practical example

In order to test what we explained earlier, we used the program from https://github.com/balancap/SSD-Tensorflow and try to understand how it works. We mainly used python to run SSD network and detect objects out an image. Below is the prototype of the function contained in the main program:

```
def process_image(img, select_threshold=0.5, nms_threshold=.45, net_shape=(300, 300)):
```

Inputs needed are the image to be analyzed, thresholds that will be explained later, and the shape of the network, in this case we use SSD300, so a 300 x 300 shape.

```
# Run SSD network.
    rimg, rpredictions, rlocalisations, rbbox_img = isess.run([image_4d, predictions, localisations, bbox_img], feed_dict={img_input: img})
```

First, we run image through the SSD network to get features map.

```
# Get classes and bboxes from the net outputs.
    rclasses, rscores, rbboxes = np_methods.ssd_bboxes_select(
            rpredictions, rlocalisations, ssd_anchors,
            select_threshold=select_threshold, img_shape=net_shape, num_classes=21, decode=True)
```

Then we get all bounding boxes from those feature map, along with class probabilities.

```
rbboxes = np_methods.bboxes_clip(rbbox_img, rbboxes)
    rclasses, rscores, rbboxes = np_methods.bboxes_sort(rclasses,
rscores, rbboxes, top_k=400)
    rclasses, rscores, rbboxes = np_methods.bboxes_nms(rclasses,
rscores, rbboxes, nms_threshold=nms_threshold)
    # Resize bboxes to original image shape. Note: useless for
Resize.WARP!
    rbboxes = np_methods.bboxes_resize(rbbox_img, rbboxes)
```

Then it is a matter of sorting through all bounding boxes using non-maximum suppression method, with a threshold of 0.5 in the case of SSD to get accurate prediction.

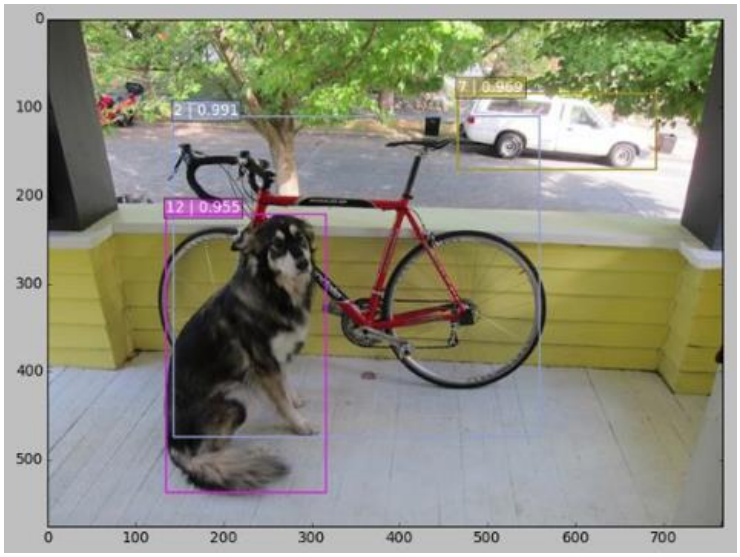

Figure 8 Detecting objects using SSD

In output, we have an image with boxes surrounding objects present in it. These boxes represent the type of the objects with a number (for example: a dog is represented by the number 12) and its accuracy (0.955 for the dog).

5- Conclusion

After understanding the operating principles of people detection applications, our goal is now to apply the program using Single Shot MultiBox Detector in TensorFlow to a video or a series of image in order to detect human rather than other objects.