A Deep Learning Approach of Video Compression

CentraleSupélec

François Castagnos, Mehdi Arsaoui

Submitted in partial satisfaction of the requirements for the degree of

MSc Intelligent Systems and Networks

Centralesupélec

Supervision : Marine Picot

2020

Abstract

This project tackles the issue of video compression with a Deep Learning approach. More precisely, it focuses on the case of slideshows broadwast (which have become popular with the developping of Internet, and even more popular this last year with the sanitary crisis). With Deep Learning tools, we detect the different elements of a slide to process them differently in function of their type (text, image, background, charts ...) minimizing the quantity of information to send.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivations

The share of video in the total data traffic on Internet is getting bigger every year. Indeed, it will reach 82% by 2022, 15 times more than is was in 2017 [Cisco, 2020]. With the developpment of telecommuting, videotelephony platforms must guaranty a reliable and high-standard service which can be energy consuming. On the other hand, the climate change issue encourages us to favor digital sobriety. Developping innovative video compression methods using Artificial Intelligence tools can be a way to tackle the two issues at stake here.

1.2 State of the Art

Recently, **NVIDIA** has annouced the development of a video compression tool using Generative Adversarial Networks (GAN_s) to reconstruct the movements of a face by identifying keypoints (nose, lips, eyes...) in the case of video chats, reducing by 90% the bandwidth requisite to follow the quality of H.264 standart [Sharma, 2020].

1.3 Objectives

The goal of this project is no find innovative solutions using Deep Learning methods to compress slideshows images, to implement them and to compare the results with common compression methods.

Chapter 2

Architecture

Before starting the implementation of any Deep Learning solution, it is important to determine the structure of the compression pipeline. That is to say, which algorithms to use and how to use them to perform optimal image compression.

2.1 General Architecture

A first idea of a compression pipeline is presented on the figure below :

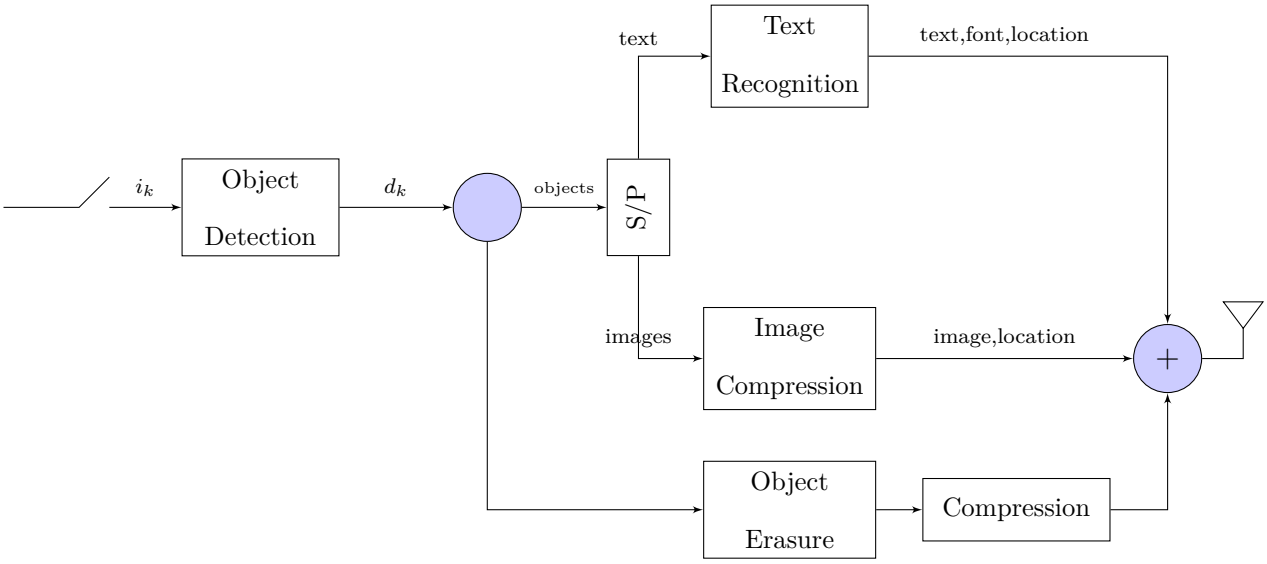


Figure 2.1: Block diagram of a more optimized architecture allowing more compression

For each image of the incoming video stream, the following steps would be applied :

1. Object detection is applied to identify the different elements of the image (texts, images,

logos, charts..). In section The objects to recognize we give more precisions about the different objects that we want our algorithm to recognize.

2. Once the objects are identified, we remove them from the image, and deduce the background. In ?? we explain how we retrieve the background by knowing the slide's objects.

3. Each object identified is treated differently.

   (a) If it is text, we perform text recognition to identify the text (tokens, font, size).

   (b) Otherwise we consider the object as an image and we compress it.

4. The differents outputs (background, texts, images) are sent in a unique packet.

The main advantage of this approach is that perfoming text recognition allows us to send only the text's information (tokens, font, localization...) and not the pixels. But this can be improved. Indeed, in slideshows, the image often stay the same for a certain time, and when it changes, the background can still stay unchanged.

2.2 A More Optimized Architecture

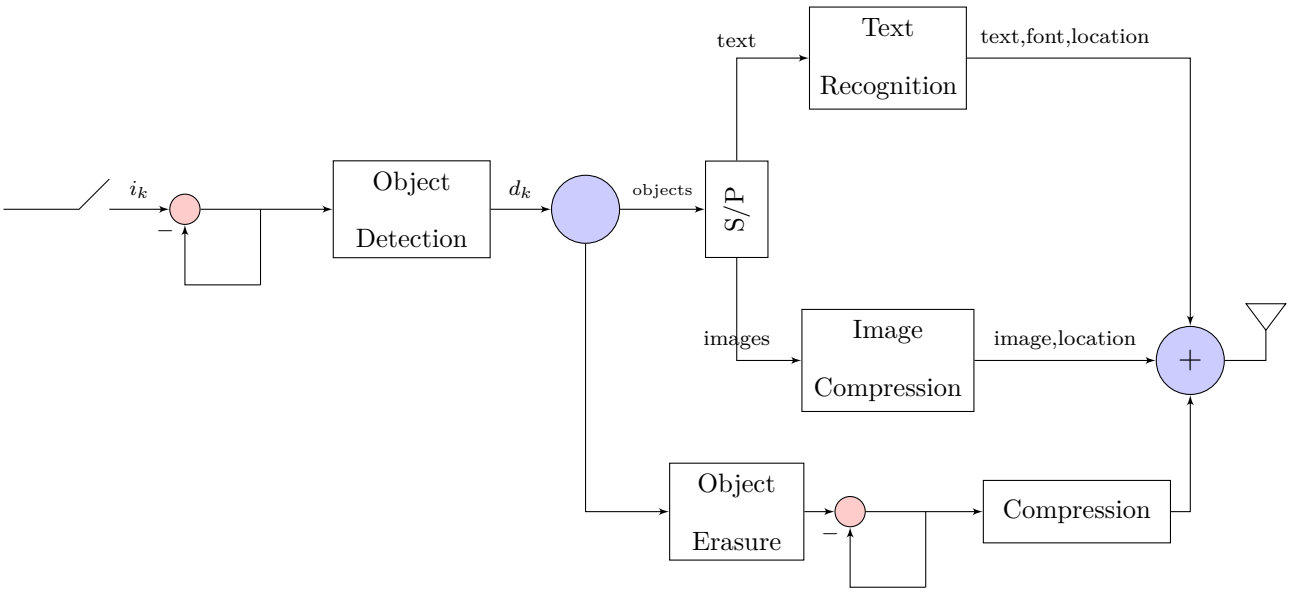


Figure 2.2: Block diagram of a more optimized architecture allowing more compression

For each image of the incoming video stream, the following steps would be applied :

1. If the input image i_{k+1} is the same as i_k, no data is sent at time k.
2. If the input image has changed, object detection is applied to identify the different elements of the image (texts, images, logos, charts..).
3. Once the objects are identified, we remove them from the image, and deduce the background.
4. If the background b_{k+1} equals b_k, we do not send its data.
5. Each object identified is treated differently.
   (a) If it is text, we perform text recognition to identify the text (tokens, font, size).
   (b) Otherwise we consider the object as an image and we compress it.
6. The differents outputs (background, texts, images) are sent in a unique packet.

To perform Object Recognition and Text Recognition, we have used Deep Learning methods as they give state of the art results.

Chapter 3

Introduction to Deep Learning

3.1 Deep Neural Network

3.2 Convolutional Neural Networds - CNN

3.3 Transfert Learning

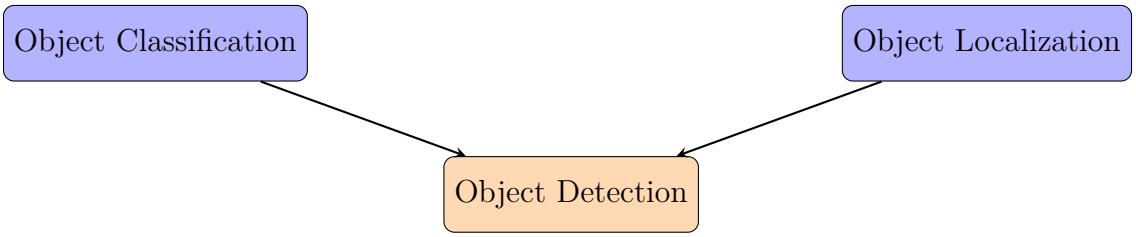
Chapter 4

Object Detection

4.1 Principle

Object Detection is the general task that involves identifying objects from a defined list in an image. It relies on the combination of two tasks : *image classification* and *object localization*. The three computer vision tasks can be defined as follows :

- **Image Classification**: Predict the class to which belongs an image amongst given classes.
- **Object Localization**: Localize objects in an image and return their location with a bounding box.
- **Object Detection**: Localize objects in an image and return their class and their location with a bounding box.



4.2 The Deep Learning models

4.2.1 R-CNN Model

In a paper published in 2014, Ross Girshick, *et al.* have demonsrated a first successful application of convolutional neural networks (CNN) in the domain of object detection [Girshick, 2014]. It has achieved state-of-the-art results on the puble VOC-2012 dataset. The model comprises three modules :

1. **Region Proposal** : Generate candidate bounding boxes.
2. **Feature Extractor** : Extract features from the regions proposed using a deep CNN.
3. **Classifier** : Classify the regions with the computed features into one of the given classes.

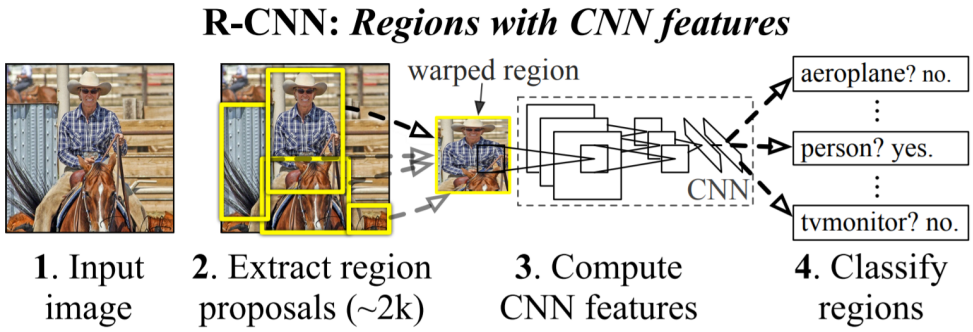


Figure 4.1: Object detection system overview

4.2.2 Additional Models

Since the publication of the R-CNN paper, a lot of new models have been proposed. Amongst many, we can cite :

- **Fast R-CNN**: R-CNN is a slow multistage pipeline model. To make it faster, Ross Girshick proposed a single model method(*).
- **SSD**: Single Shot Multibox Detector is a model introduced by Wei Liu, *et al.*, which discretizes the output space of bounding boxes into a set of default boxes. Then, at

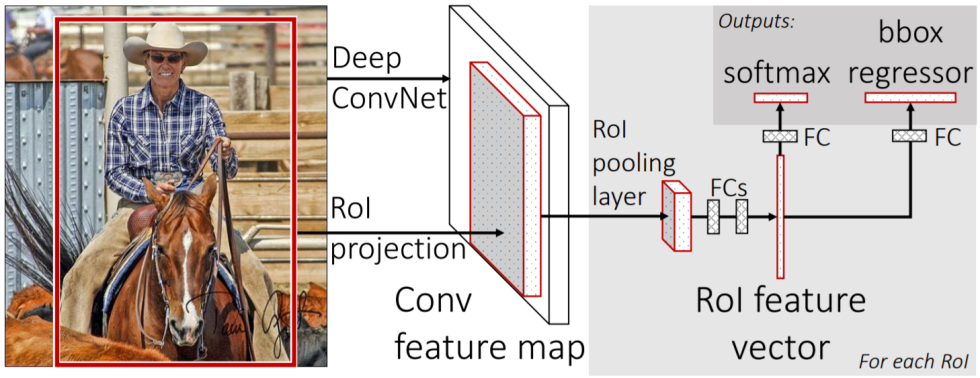
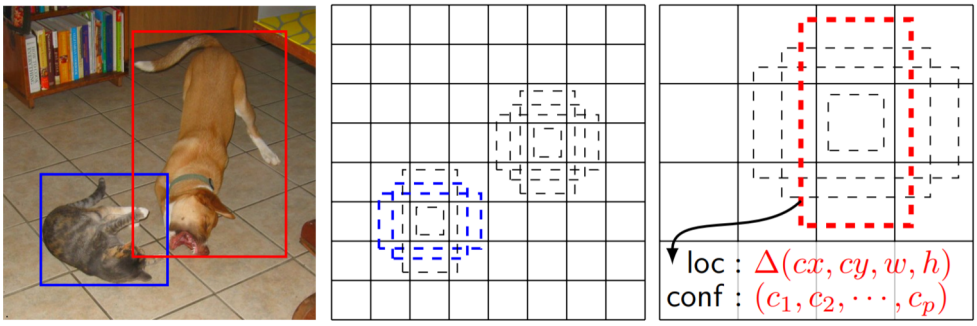


Figure 4.2: Fast R-CNN architecture

prediction time, the network generates scores for the presence of each object category in each default box and produces adjustments to the box to better match the object shape.



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

Figure 4.3: SSD framework

4.2.3 The TensorFlow 2 Detection Model Zoo

The Tensorflow team has made available on their GitHub repository a list of models pre-trained on the COCO 2017 Dataset. The COCO (Comon Objects in Context) dataset is composed of more than 200K labelled images.

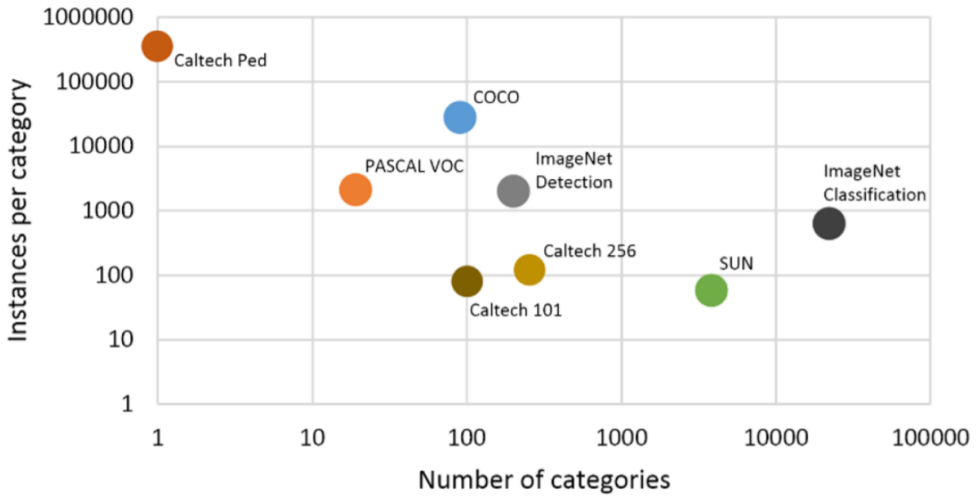


Figure 4.4: Comparison of different Object Detection datasets. COCO dataset presents a good tradeoff between *number of categories* and *instances per category*.

The different models are listed with their respective performance and precision. We have chosen to use **CenterNet Resnet101 V1 FPN 512x512** as a pre-trained model. It means that to train a Neural Network on our data set, we have performed transfert learning on this model. In **??** we give further explanations on how we did it.

Model name	Speed (ms)	COCO mAP	Outputs
CenterNet HourGlass104 512x512	70	41.9	Boxes
CenterNet HourGlass104 Keypoints 512x512	76	40.0/61.4	Boxes/Keypoints
CenterNet HourGlass104 1024x1024	197	44.5	Boxes
CenterNet HourGlass104 Keypoints 1024x1024	211	42.8/64.5	Boxes/Keypoints
CenterNet Resnet50 V1 FPN 512x512	27	31.2	Boxes
CenterNet Resnet50 V1 FPN Keypoints 512x512	30	29.3/50.7	Boxes/Keypoints
CenterNet Resnet101 V1 FPN 512x512	34	34.2	Boxes
CenterNet Resnet50 V2 512x512	27	29.5	Boxes
CenterNet Resnet50 V2 Keypoints 512x512	30	27.6/48.2	Boxes/Keypoints
EfficientDet D0 512x512	39	33.6	Boxes
EfficientDet D1 640x640	54	38.4	Boxes
EfficientDet D2 768x768	67	41.8	Boxes

Figure 4.5: Models from the TensorFlow 2 Detection Model Zoo. COCO mAP is the score to evaluate model.

4.2.4 The mean Average Precision metric (mAP)

To train a model and evaluate it, it is necessary to use the right metric that is going to tell us if the model performs well or not. The object detection task differs from most of the other tasks as it consists of two distinct task :

- Determining if an object is in the image (Classification)
- Determining the location of the object (Localization $\rightarrow$ Regression)

The combination of these two tasks make the evaluation not trivial. Before explaining what the Average Precision measures, it is important to explain the following terms :

Precision : Measures the percentage of predictions that are correct.

Recall : Measures how good a model finds all the positives. Mathematically, these measures are defined by :

$$Precision = \frac{TP}{TP + FP} \quad \text{and} \quad Recall = \frac{TP}{TP + FN}$$

with $TP = True\ Positive$, $FP = False\ Positive$ and $FN = False\ Negative$.

IoU (Intersection of Union) : Measures the overlap between two regions. In the object detection case, it is used to measure how much a predicted boundary overlaps with the ground truth. Depending on the IoU, a predicted boundary is consider a True Positive or a False Positive (Generally the threshold $\simeq 0.5$)

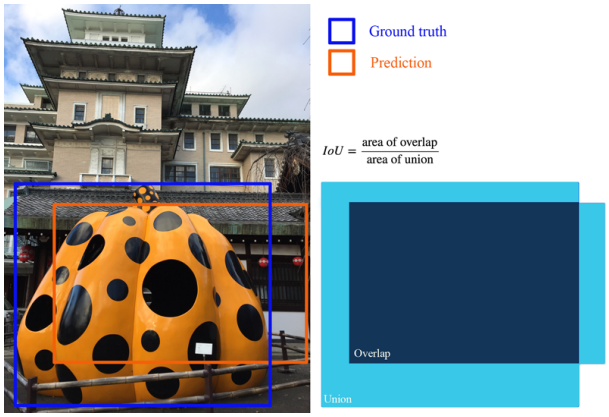


Figure 4.6: IoU is calculated by comparing the predicted region and the ground truth

Average Precision

Now, to understand the Average Precision let's consider a simple example. We have got a dataset of slides (converted to images) and we want to detect the logos that are in the different slides. In this example, the whole dataset contains 6 logos. To compute the Average Precision, we first rank the predictions by the decreasing predicted confidence level (i.e. for each prediction, how much the model is certain of this prediction). It means that the rank 1 prediction is the one the model is the most sure of. Then, we draw this table :

Rank	**Positive**	**Precision**	**Recall**
1	True	1	0.17
2	True	1	0.33
3	True	1	0.5
4	False	0.75	0.5
5	True	0.8	0.67
6	False	0.67	0.67
7	False	0.57	0.67
8	True	0.63	0.83
9	False	0.56	0.83
10	True	0.6	1

Table 4.1: Prediction of logos in the dataset

To understand how this table is computed, let's take the prediction with rank 5. This prediction is a True Positive. Of the 5^{th} most confident predictions, 4 are True Positives so $Precision = \frac{4}{5} = 0.8$. And with the 5^{th} most confident predicitons, the model found 4 out of the 6 logos to find so $Recall = \frac{4}{6} \simeq 0.67$.

When the table is computed, we draw the Precision - Recall curve and look at the area under the curve. More precisely, the curve is smoothed so that it is only decreasing. The **Average Precision** metric is defined as :

$$AP = \int_0^1 p(x)\,dx$$

where p is the function assiociated to the smoothed Precision - Recall curve. As *Precision* and *Recall* take value in $[0, 1]$, $AP \in [0, 1]$.

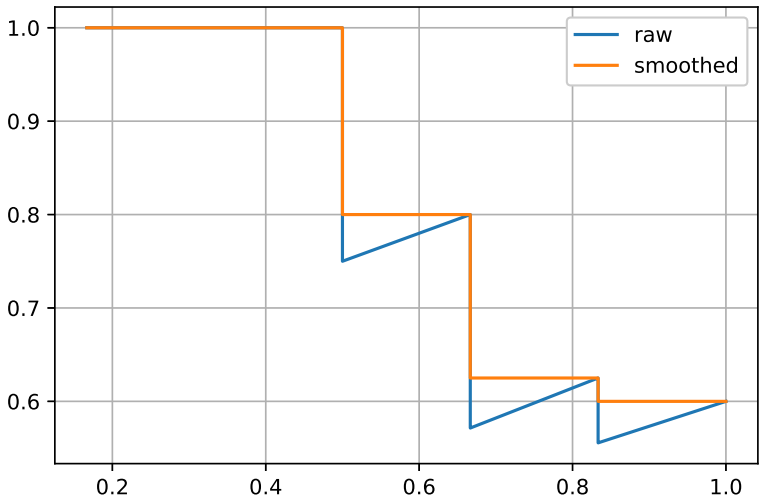


Figure 4.7: Precision - Recall curve

COCO mAP

To evaluate models on the COCO dataset, the *COCO mAP* metric has been introduced. The *COCO AP* is the average AP over multiple *IoU*. More precisely, for the COCO competition (finding the best model for the COCO dataset), **COCO AP = AP@[.5:.05:.95]**. It means that the AP is averaged for *IoU* ranging from 0.5 to 0.95 with a step equal to 0.05. Then, the *COCO mAP* is computed by averaging the *COCO AP* over the different categories (text, images, shape... in our case).

4.3 Creation of a Training Dataset

To perform object detection, we must use an adequate dataset to teach our model to recognize the different objects in a slides. As there is no such dataset available on the web, we had to create this dataset ourselves.

4.3.1 The objects to recognize

Before creating the desired dataset, we must decide of the different elements to recognize and agree on the rules. The different categories are :

- **Text** : Blocks of text
- **Image** : Pictures, photographie, elements with complex texture
- **Shape** : Simple shapes such as arrows, circles etc...
- **Logo** : Logos and more complex shapes
- **Table** : Tables containing text
- **Figure** : Charts, diagrams...

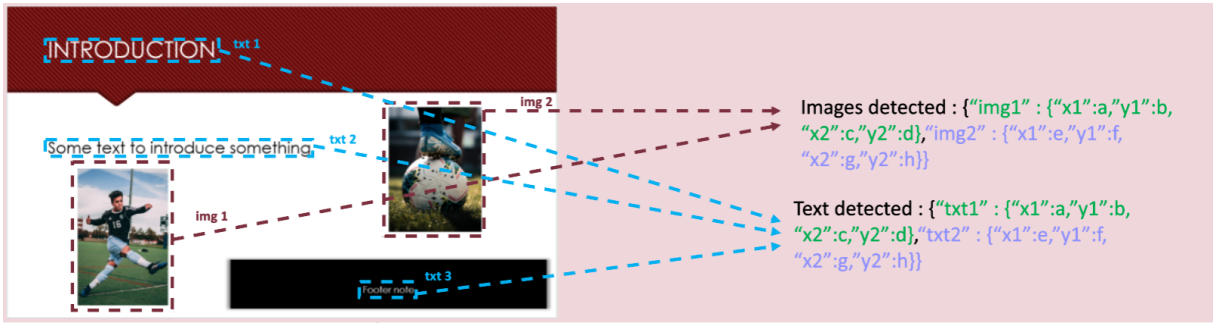


Figure 4.8: Example of objects to detect

4.3.2 The LabelImg tool

To prepare the dataset, we use the **LabelImg** tool that can be forked on Github. It is a graphical image annotation tool written in *Python* that, for each image, outputs an XML file containing the information of the different objects labelled.

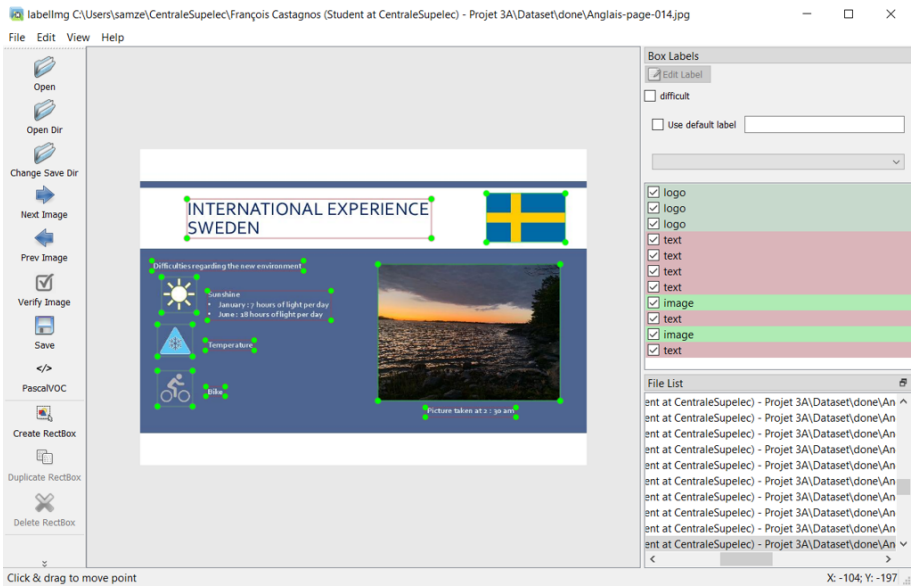


Figure 4.9: LabelImg software is used to annotate the images gathered

4.4 First results

After having fine-tuned a SSN RESNET model for more than 13k steps, here is the result of the model on a sample image. We see that the model finds all the objects but there are also a lot of False Positives.

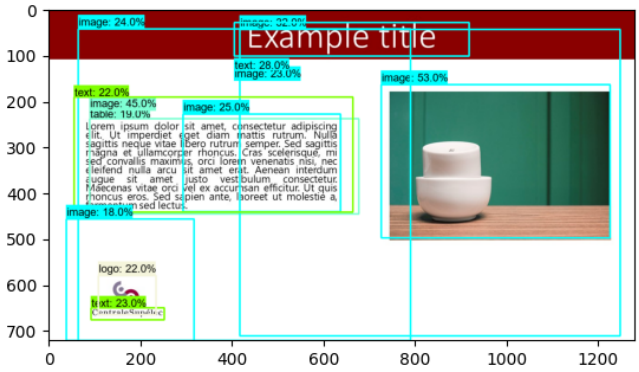


Figure 4.10: Object detection on an example slide using a SSD RESNET model

4.5 Generating Images and labels

The first method used to gather a dataset for the training of the object detection algorithm was really tedious and uneffective for the following reasons

- **Tedious**: exporting many slideshows to images, with a wide variety of templates is long, and labelling the different elements in the images is even more.
- **Uneffective**: the slideshows labelled (around 1000) all belonged to a few different presentations so the variety of training inputs was not really large. Moreover, as the labelling was handmade, it was not precise and consistent.

To tackle this issue, we then thought of an alternative : generating random slides and corresponding labels with a Python script. Thanks to the **PIL** library, it is possible to create and edit images (paste objects, write text, change color...) with a few lines of code. The images inside the generated slide come from the COCO Dataset (we retrieved around 5000 images), as it offers a wide variety of images. The logos have been retrieved from the Flaticon website. Concerning the text, we wanted it to be as random and realistic as possible as possible. To do so, we used a deep learning text generation model for the Hugging Face Community. Last but not least, with the different elements in our images automatically generated, we were able to generate the *.xml* file corresponding to the labels.

Thanks to this strategy, we were able to generate 10000 slides with a frequency of 3 slides per second. The entire Python script is available in our Github Repository.

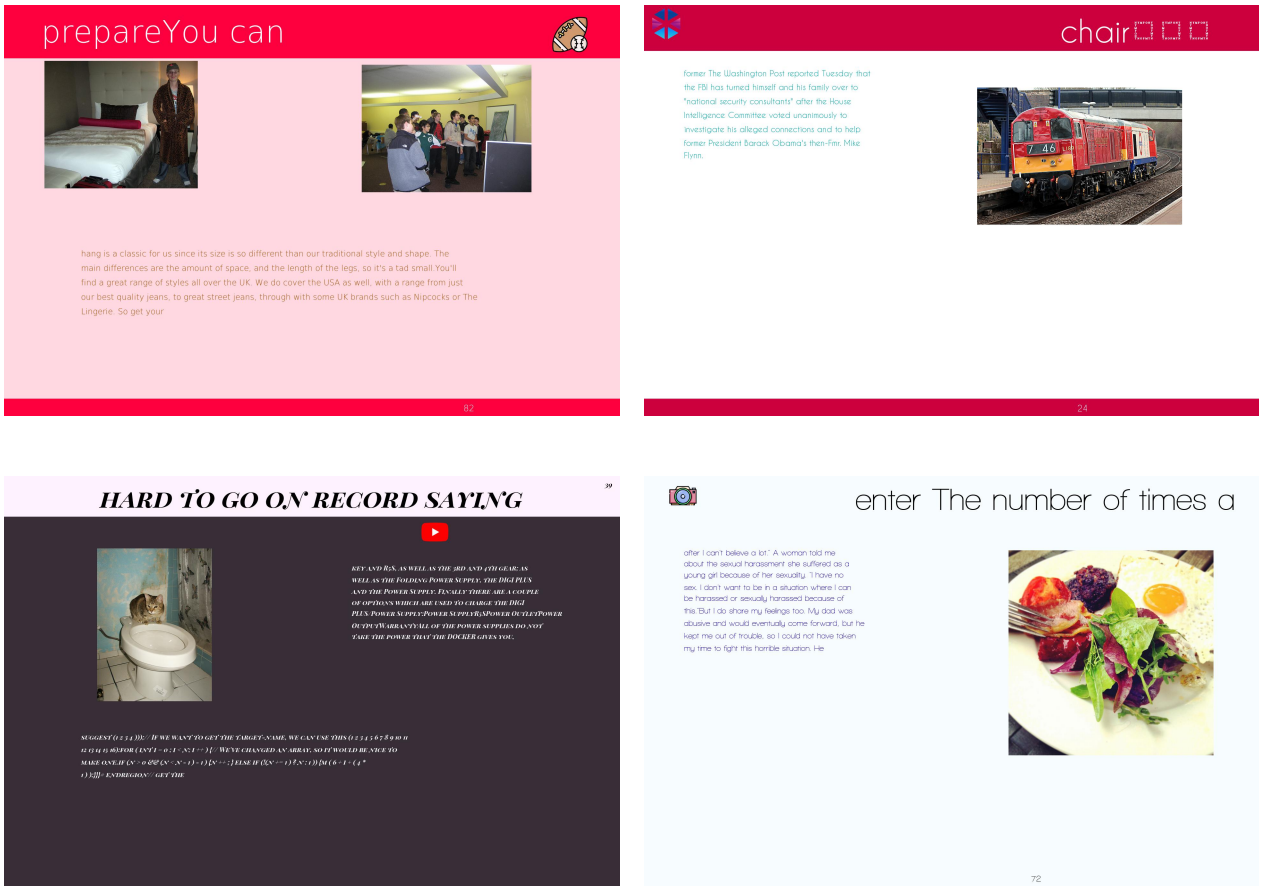


Figure 4.11: Sample of images generated

Chapter 5

Text Recognition

5.1 Principle

Chapter 6

Dash Application

6.1 The *Dash* Library

Dash is a Python framework for building web applications. It is useful for building data visualization apps with custom user interfaces in pure Python.

We decided to use it to display our algorithm output, in order to detail the different steps, but also to evaluate the code's performance.

First, we are currently focusing on the app layout - the "Front-end

Moreover, we plan to add callbacks which are functions that are making the app interactive, triggered when an input or property changes. The logic behind those callbacks can be considered as the "Back-end".

Chapter 7

Conclusions

References

Cisco (2020). 'Cisco Annual Internet Report'. In: *white paper* (cit. on p. 1).

Girshick, Ross (2014). 'Rich feature hierarchies for accurate object detection and semantic segmentation'. In: (cit. on p. 7).

Sharma (2020). 'AI Can See Clearly Now: GANs Take the Jitters Out of Video Calls'. In: *NVIDIA blog* (cit. on p. 1).