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

Acknowledgements

We would like to thank all the people who contributed to the success of our project.

First, we would like to thank our supervisor, Mrs Picot Marine, who advised us during the whole project, from the subject elaboration to the final presentation.

Moreover, we would like to thank M. Yang Sheng for accepting the submitted topic as a final year project, as well as all the other jury members for assisting to our final presentation.

Abstract

This project tackles the issue of video compression with a Deep Learning approach. More precisely, it focuses on the case of slideshows broadcasts (which have become popular with the development 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.

All our code is available on the following Git repository.

Table of Contents

1 Introduction **1**
1.1 Motivations . . . 1
1.2 State of the Art . . . 1
1.3 Objectives . . . 1

2 Architecture **2**
2.1 General Architecture . . . 2
2.2 A More Optimized Architecture . . . 3

3 Introduction to Deep Learning **5**
3.1 Deep Neural Networks . . . 5
3.1.1 Introduction . . . 5
3.1.2 General Architecture . . . 6
The perceptron . . . 7
Loss function . . . 8
Gradient descent and backpropagation . . . 8
3.2 Convolutional Neural Networds - CNN . . . 9
3.2.1 Concept and specificity . . . 9
3.2.2 Feature extraction and convolution layer: the kernel . . . 10
Convolution operations . . . 10
Non-linearity . . . 11
Dimensionality reduction: Pooling . . . 12
3.2.3 Architecture description for a classification example . . . 12
Feature learning . . . 13
Classification . . . 13
3.3 Transfer Learning . . . 14
3.3.1 General idea . . . 14
Inductive Transfer Learning . . . 15
Unsupervised Transfer Learning . . . 15

3.3.2 Current usage and potential . . . 15
Pre-trained models as feature extractors: . . . 16
Pre-trained models as basis for adjustment: . . . 16
Performance . . . 17

4 Object Detection **19**
4.1 Principle . . . 19
4.2 The Deep Learning models . . . 20
4.2.1 R-CNN Model . . . 20
4.2.2 Additional Models . . . 20
4.2.3 The TensorFlow 2 Detection Model Zoo . . . 21
4.2.4 The mean Average Precision metric (mAP) . . . 23
Average Precision . . . 24
COCO mAP . . . 25
4.3 Creation of a Training Dataset . . . 25
4.3.1 The objects to recognize . . . 26
4.3.2 The LabelImg tool . . . 26
4.4 First results . . . 27
4.5 Generating Images and labels . . . 28
4.6 New results . . . 29
4.6.1 Metrics Results . . . 30

5 Text Recognition **32**
5.1 Tesseract . . . 32
5.1.1 Principle . . . 32
5.1.2 Python Wrapper . . . 33
5.2 Results . . . 33

6 Background Reconstruction **34**
6.1 The PIL Library . . . 34
6.2 Hypotheses . . . 34
6.3 Algorithm description . . . 35

7 Font Detection **40**
7.1 The Dataset . . . 40
7.2 The Model . . . 41
7.3 The Results . . . 42

7.4 Transfer Learning 42
7.5 New Results 42

8 Dash Application **44**
8.1 The *Dash* Library 44
8.2 Our App 44
8.2.1 Object detection 45
8.2.2 Background retrieval 45
8.2.3 Text recognition 46
8.2.4 Performance 47

9 Conclusions **48**

List of Figures

2.1 Block diagram of a more optimized architecture allowing more compression . 2
2.2 Block diagram of a more optimized architecture allowing more compression . 3

3.1 Example of a Fully-Connected Network architecture for animal recognition . 6
3.2 General architecture of a network with 2 hidden layers 7
3.3 General parameters and activation of a perceptron 8
3.4 Gradient descent on a simple case with two weights 9
3.5 General architecture of a C-NN for digit recognition 10
3.6 Example of a convolution on a 5x5 picture with a 3x3 kernel 11
3.7 The ReLU function 11
3.8 Output of a 2x2 Max Pooling on a 4x4 kernel 12
3.9 Feature learning on a classification C-NN 13
3.10 Classification Layers on a C-NN 14
3.11 Illustration of Inductive Transfer Learning principle 15
3.12 Illustration of a network used for facial recognition 17
3.13 Performance of transferred vs state of the art models 18

4.1 Object detection system overview 20
4.2 Fast R-CNN architecture 21
4.3 SSD framework 21
4.4 Comparison of different Object Detection datasets. COCO dataset presents a good tradeoff between *number of categories* and *instances per category.* . . . 22
4.5 Models from the TensorFlow 2 Detection Model Zoo. COCO mAP is the score to evaluate model. 22
4.6 IoU is calculated by comparing the predicted region and the ground truth . . 23
4.7 Precision - Recall curve 25
4.8 Example of objects to detect 26
4.9 LabelImg software is used to annotate the images gathered 27
4.10 Object detection on an example slide using a SSD RESNET model 27

4.11 Sample of images generated 29
4.12 A sample inference with the new model. Right: labeled input input. Left: Output of object detection 30
4.13 Loss evolution with the number of steps 30
4.14 mAP evolution with the number of steps 31

5.1 A sample image of a text section detected in a slideshow 33

6.1 Example of slides following (or not) the upper hypotheses 35

7.1 Sample of images generated 40
7.2 Summary of our Font Detection Model 41

8.1 An overview of the Object Detection section 45
8.2 Overview of the Background section 46
8.3 Overview of the Text Detection section 47

List of Tables

4.1 Prediction of logos in the dataset 24

7.1 Dataset used for training 41
7.2 Performances of our font detection model 43

Chapter 1

Introduction

1.1 Motivations

The share of video in the total data traffic on Internet is getting larger 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 to find innovative solutions using Deep Learning methods to compress slideshows images, to implement them and to compare the results with classical 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 :

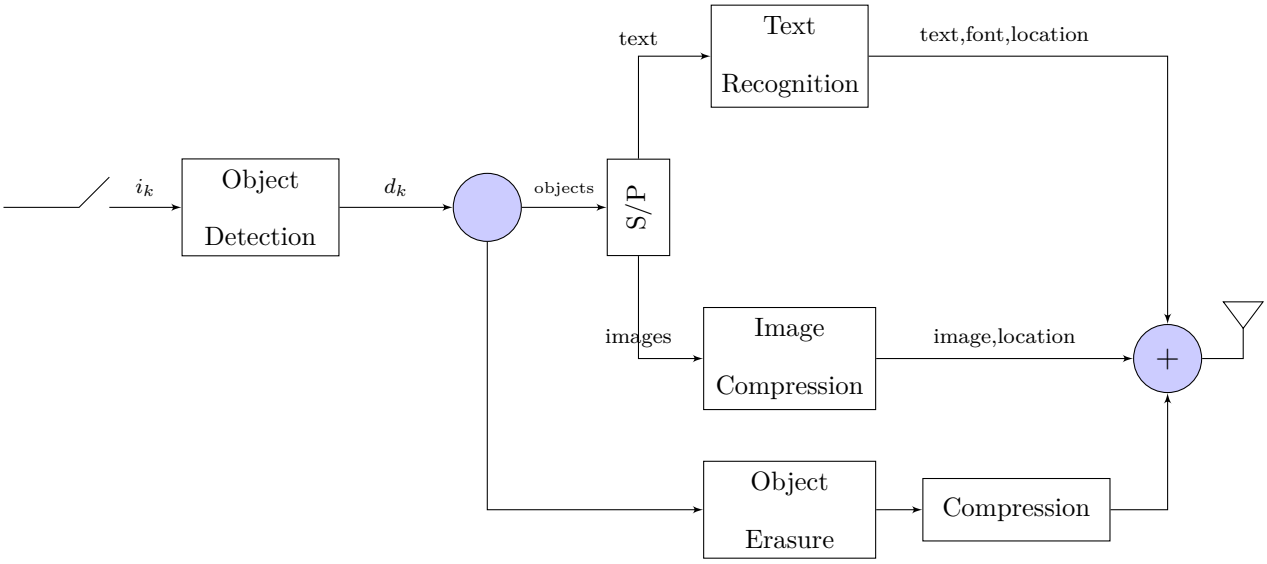


Figure 2.1: Block diagram of a more optimized architecture allowing more compression

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 Background Reconstruction 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 different 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

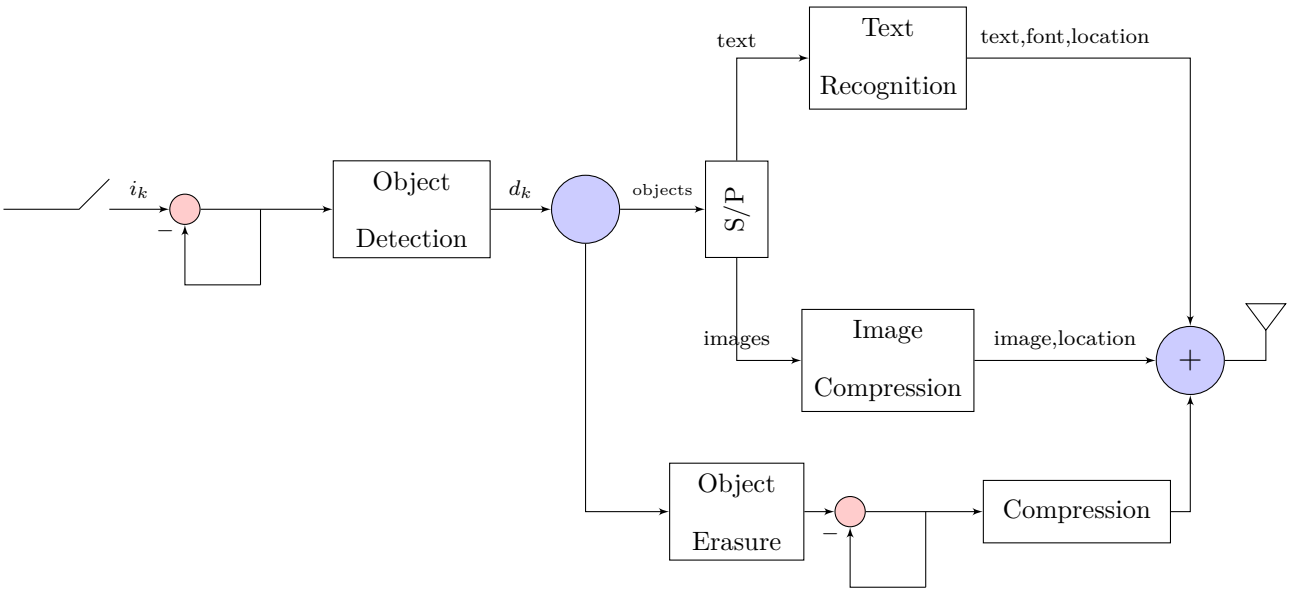


Figure 2.2: Block diagram of a more optimized architecture allowing more compression

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 different 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 Networks

3.1.1 Introduction

Deep Neural Networks are at the core of Deep Learning. They are fully inspired by the human brain network of neurons and hence the thinking process. Since its premises in 1943, and much more significantly over the past decade, its implications have become extremely large: Video synthesis, self driving cars, human level game AI, and more.

Generally speaking, deep learning is a machine learning method that takes in an input X, and uses it to predict an output of Y.

Considering an important dataset of input/output pairs - a labellized dataset, a Deep Learning algorithm will try to undermine the difference between its prediction and the expected output. By doing this, the general idea is to identify the association between given inputs and outputs through training, that will then be useful to generalize to unknown inputs.

As an example, let's consider inputs as images of dogs and cats, and outputs as labels for those images (i.e saying if the input picture is a dog or a cat).

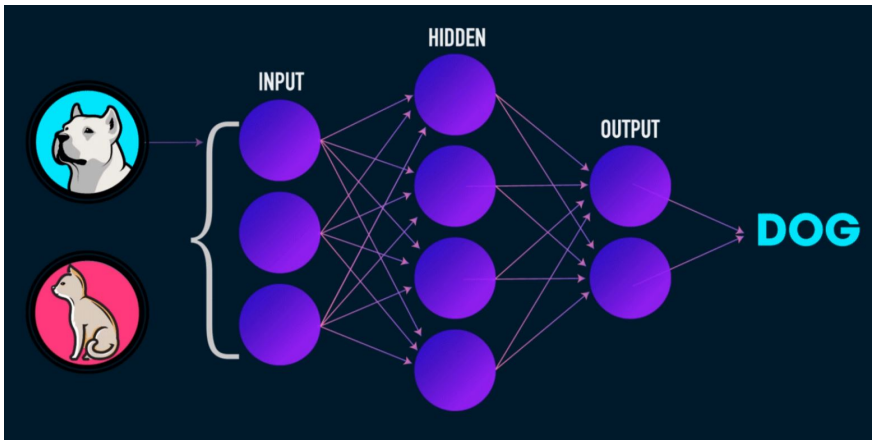


Figure 3.1: Example of a Fully-Connected Network architecture for animal recognition

If an input has a label of a cat, but the deep learning algorithm predicts a dog, then the deep learning algorithm will learn that the features of the current image (moustache, cat eyes) are going to be associated with a cat.

3.1.2 General Architecture

A neural network is composed of input, hidden, and output layers — all of which are composed of nodes, called "perceptrons". Input layers take in a numerical representation of data (e.g. RGB pixel representation of an image), output layers deliver predictions, while hidden layers are correlated with most of the computation.

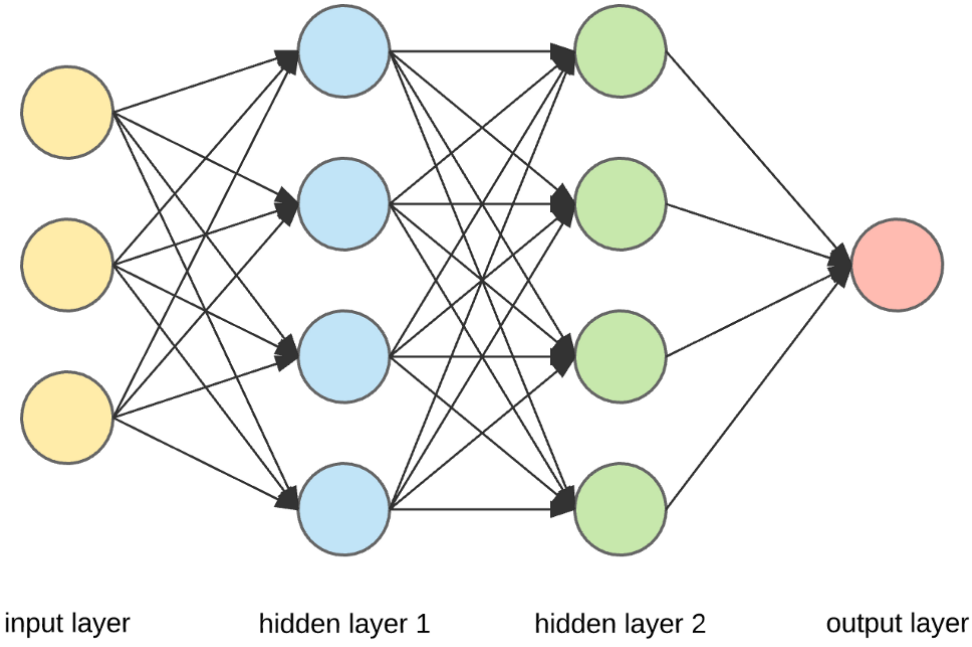


Figure 3.2: General architecture of a network with 2 hidden layers

Before going into detail about the whole model architecture, let's talk about the fundamental building block of a network, the perceptron.

The perceptron

A perceptron is in fact a node, or a single neuron, located in a layer, that takes binary values (usually 0 or 1). It is connected to all the nodes of the previous layer, and all of the nodes of the next layer, through connexions having "weights". The idea behind is to replicate the "activation" of a single neuron through connections with other neurons, which is the basis of human thoughts.

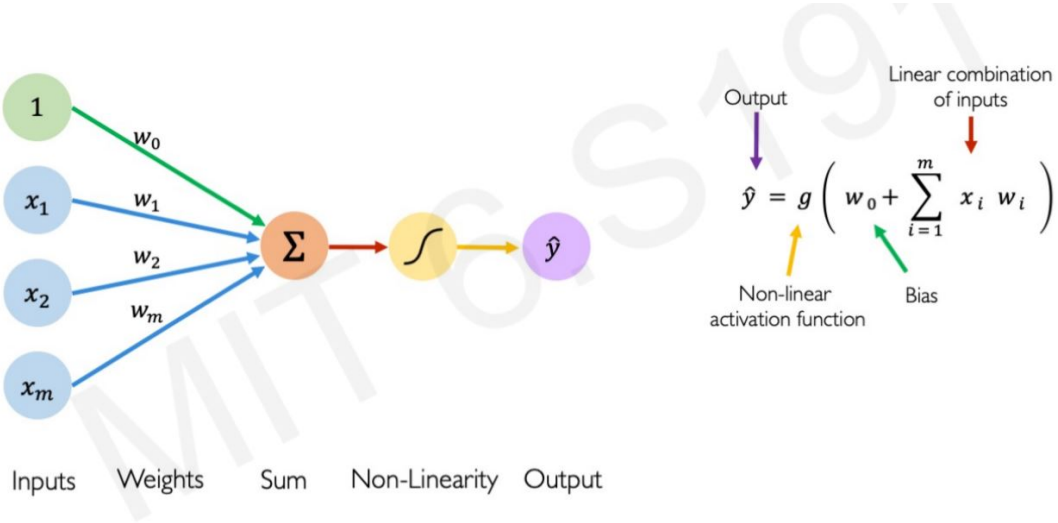


Figure 3.3: General parameters and activation of a perceptron

Generally, the activation functions' role is to bound node values between 0 and 1, and the biases to replicate cognitive biases.

Hence, a Deep Neural Network is a set of weights and biases that are being trained to activate the right perceptrons in consecutive layers, for a specific purpose (feature recognition in cat images for example). But how does this training work?

Loss function

After the neural network passes its inputs all the way to its outputs, the network evaluates how good its prediction was (relative to the expected output) through what is called a loss function. A common loss function used is the "Mean Squared Error". Let Y_i be the real labels of the images, and $\hat{Y}_i$ the predicted ones, than the Mean Square Error is given by :

$$MSE = \frac{1}{n}\sum_{i=1}^{n} Y_i - \hat{Y}_i$$

Gradient descent and backpropagation

After every execution of the algorithm for a pre-labellized input, it calculates the loss function based on the expected output. Then, it uses the gradient descent on its parameters (weights, biases of every node), to identify how they should be updated for a smaller loss.

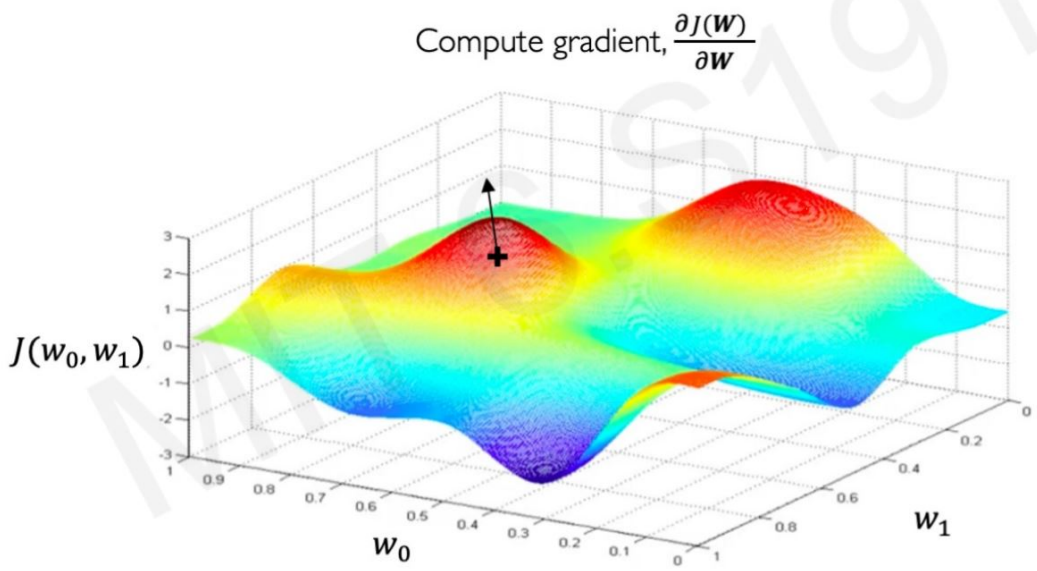


Figure 3.4: Gradient descent on a simple case with two weights

This update process is called backpropagation, the weights and biases are progressively updated layer per layer, in the opposite way of the gradient of the loss function.

This process is repeated until convergence, after setting a threshold for the loss function, ie a desired precision, or when the loss function does not decreases anymore.

3.2 Convolutional Neural Networds - CNN

3.2.1 Concept and specificity

In 1979, K.Fukushima designed the first convolutional neural network, for handwriting character recognition. 40 years later, they are widely used for object detection purposes, classification or segmentation.

What differeciates them from Fully-Connected Networks (seen previously) is the inclusion of consecutive convolution tasks from the input before entering to a multi-layer perceptron.

The architecture of a ConvNet is analog to that of the connectivity pattern of Neurons in the

Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.

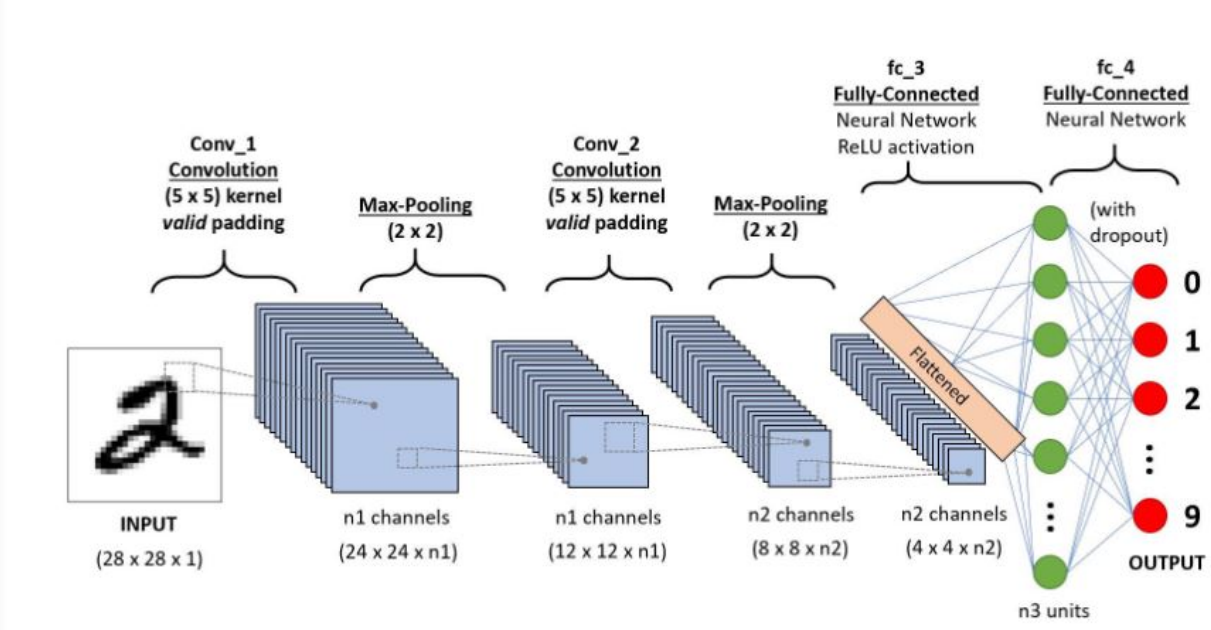


Figure 3.5: General architecture of a C-NN for digit recognition

3.2.2 Feature extraction and convolution layer: the kernel

Convolution operations

Every convolution layer is in fact a kernel, a filter that will, given an input image, perform a convolution, and based on the value of the operation, assess if a feature has been detected or not on specific zones.

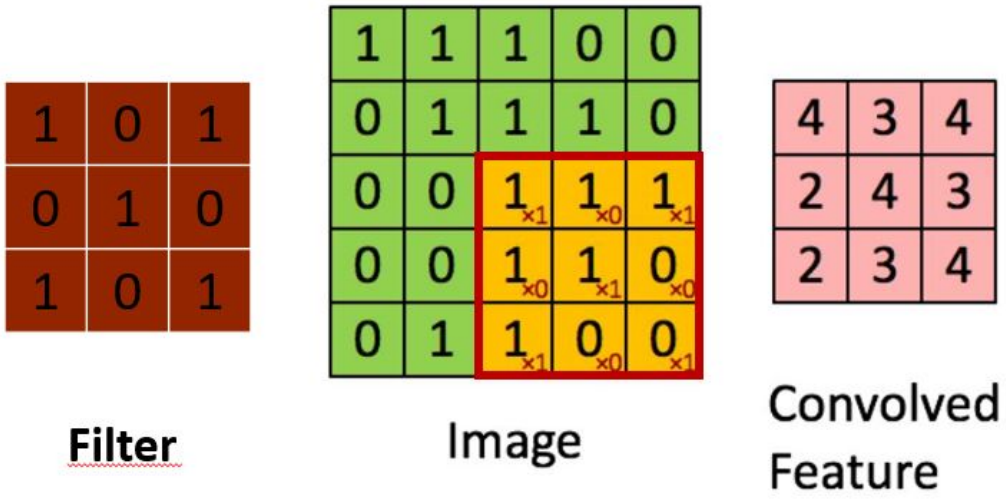


Figure 3.6: Example of a convolution on a 5x5 picture with a 3x3 kernel

The objective of the Convolution Operation is to extract the high-level features such as edges, from the input image.

Non-linearity

Usually, after every filter application, a non-linearity operation is performed in order to detect features in a non-linear way.

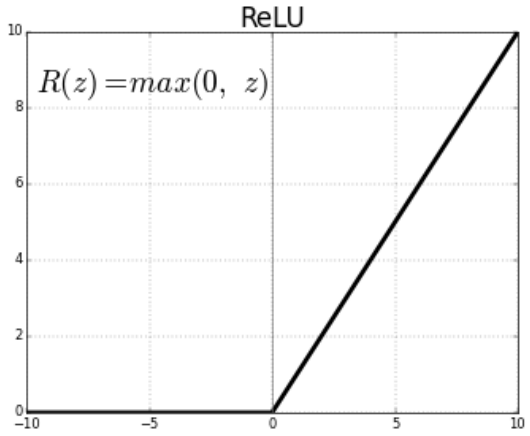


Figure 3.7: The ReLU function

A common function used is the ReLU function, that changes negative filter outputs to zero and leaves other values unchanged.

Dimensionality reduction: Pooling

Then, a pooling operation is performed, as there is often a need to reduce the spacial dimensionality of the features before entering to the Fully-Connected Layer. Different pooling types exist: Max Pooling and Average Pooling.

Max Pooling returns the maximum value from the portion of the image covered by the Kernel. On the other hand, Average Pooling returns the average of all the values from the portion of the image covered by the Kernel.

For its Noise Suppression properties, Max Pooling are more widely used in ConvNets.

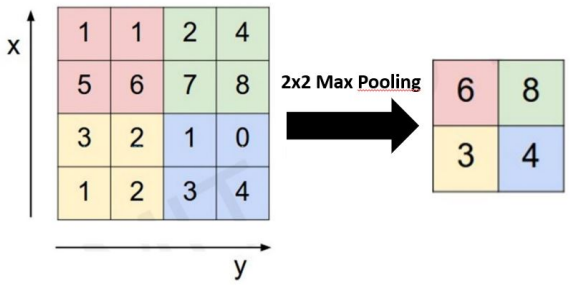


Figure 3.8: Output of a 2x2 Max Pooling on a 4x4 kernel

The Convolutional Layer and the Pooling Layer, together form the i-th layer of a Convolutional Neural Network. Depending on the complexities in the images, the number of such layers may be increased for capturing low-levels details even further, but at the cost of more computational power.

These consecutive operations and architecture enable the model to understand the features.

3.2.3 Architecture description for a classification example

Given the previous operations described separately, a C-NN architecture consists in two main steps.

Feature learning

Feature learning represents the first part of the Network, consisting in applying consecutive Convolutional Layers, followed by Pooling operations.

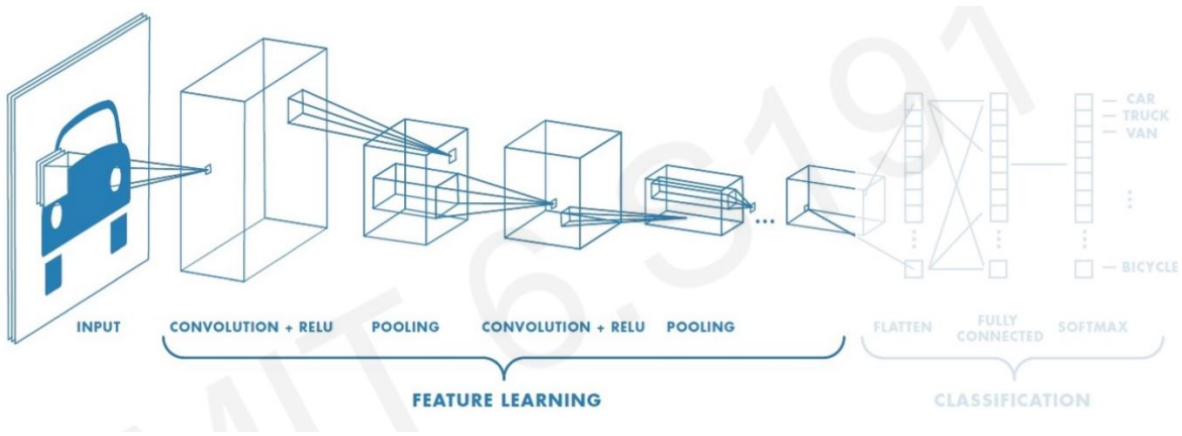


Figure 3.9: Feature learning on a classification C-NN

Convolutional Network need not to be limited to only one Convolutional Layer. Conventionally, the first Layer is responsible for capturing the Low-Level features such as edges, color, gradient orientation, etc. With added layers, the architecture adapts to the High-Level features as well, giving us a network which has the wholesome understanding of images in the dataset, similar to how we would.

Classification

After going through consecutive kernels, the filtered image is flattened into a 1-D array and introduced into a classic Fully-Connected Network.

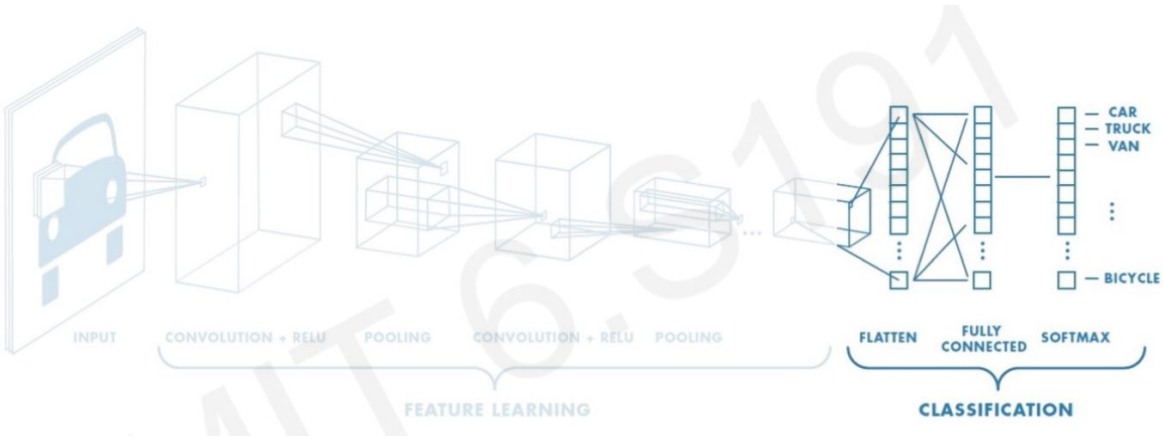


Figure 3.10: Classification Layers on a C-NN

The output for a classification application is often a 1-D array of probabilities of the image belonging to specific labels (cars, trucks, etc in Figure 3.11). The learning process is also done through backpropagation, as described in section 3.1.2.

3.3 Transfer Learning

3.3.1 General idea

In Deep Learning, Transfer Learning refers to the set of methods that allow the transfer of knowledge acquired from solving a given problem to another problem.

It is strongly inspired by the process of human learning. If we take the example of a guitarist who wants to learn to play the piano, he can capitalize on his music knowledge to learn to play a new instrument. The same way, a car recognition model can be quickly adapted to truck recognition.

Transfer Learning is based on a simple idea, that of re-using knowledge acquired in other settings (sources) to solve a particular problem (target).

It has been quite popular recently, as the models often require high computation times and significant resources. Moreover, by using pre-trained models as a starting point, Transfer Learning makes it possible to quickly develop high-performance models and efficiently solve complex problems in Computer Vision or Natural Language Processing, NLP.

In this context, we can distinguish two types of Transfer Learning:

Inductive Transfer Learning

In this configuration, the source and target domains are the same (same data), and their tasks are different but close. The idea is then to use the existing models to advantageously reduce the scope of the possible models (model bias) as pictured below.

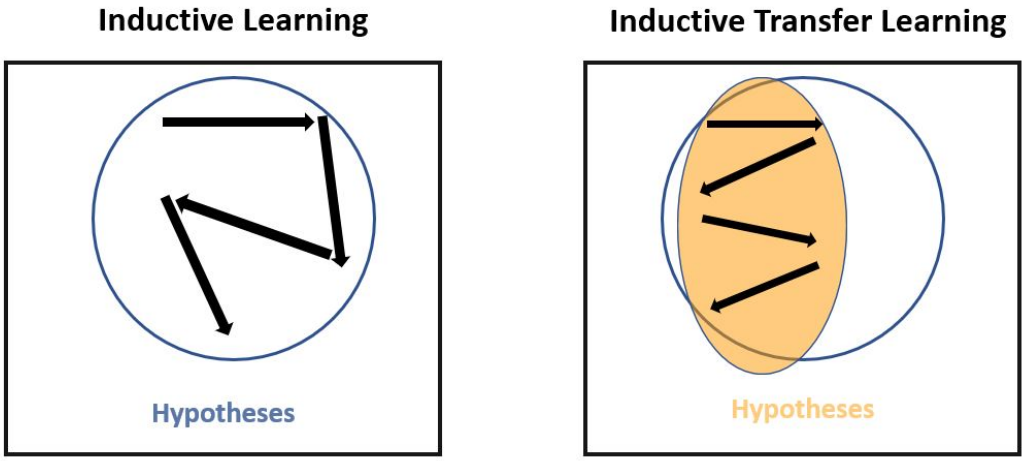


Figure 3.11: Illustration of Inductive Transfer Learning principle

Unsupervised Transfer Learning

As in the Inductive Transfer Learning context, the source and target domains are similar, with different tasks to perform. However, the data in both domains is not labeled.

It is often easier to obtain large amounts of unlabeled data from databases and web sources than labeled data. This is where results the interest of using unsupervised learning combined with transfer learning, as large training datasets maximize chances to have a performant model.

3.3.2 Current usage and potential

The use of Transfer Learning in Deep Learning mainly consists in exploiting efficient pre-trained models, that have been developed on large databases and are freely shared.

In its current usage there are 2 strategies:

Pre-trained models as feature extractors:

The architecture of Deep Learning models is often a stack of layers of neurons, that "learn" different features depending on the level at which they are located. The last layer (usually a fully connected layer) is used to obtain the final output. (see Figure 3.5)

The idea is to reuse a pre-trained network without its final layer. This new network then works as a fixed feature extractor for other tasks.

For example, if we want to create a model able to identify the species of a flower from its image, we could use the first layers of the convolutional neural network model AlexNet, initially trained on the ImageNet image database for image classification.

Pre-trained models as basis for adjustment:

This is a more complex technique, in which not only the last layer is replaced to perform classification or regression, but other layers are also selectively re-trained. Indeed, deep neural networks are highly configurable architectures with various hyperparameters.

While the first layers capture generic features, the last layers focus more on the specific task at hand, as below

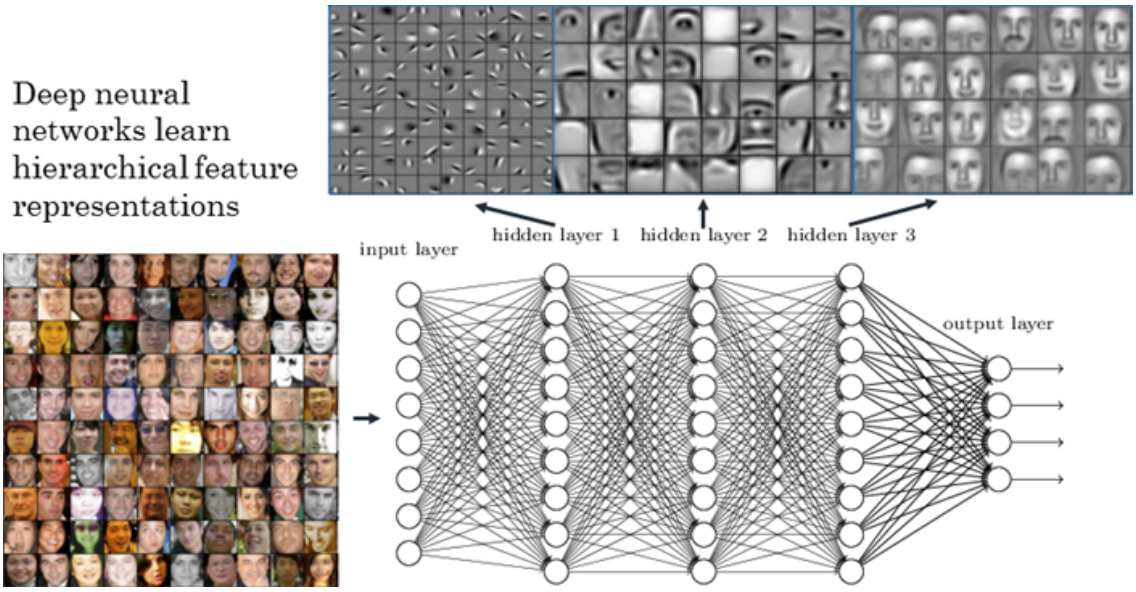


Figure 3.12: Illustration of a network used for facial recognition

The idea is therefore to freeze (i.e. fix the weights) of some layers during training and refine the rest to fit the problem.

This strategy allows to reuse the knowledge of the global architecture of the network and to exploit its states as a starting point for the training. It thus allows to obtain good performances with a shorter training time.

Performance

As many models are freely available online, we have considered this approach as a good match between performance and training time. Indeed, in most fields of Deep Learning, the difference between the best state of the art models and transferred models is not significant in terms of accuracy.

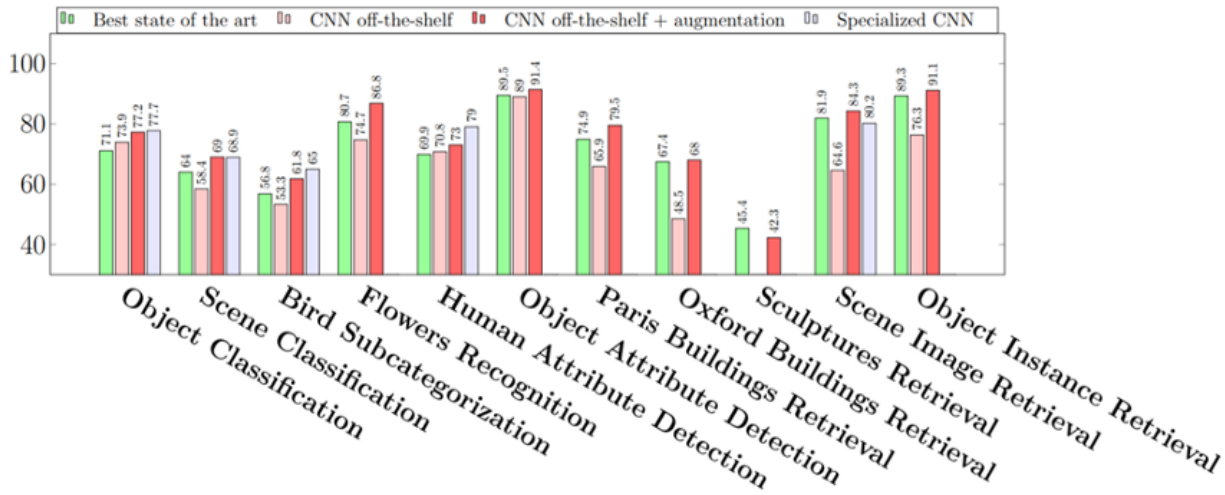


Figure 3.13: Performance of transferred vs state of the art models

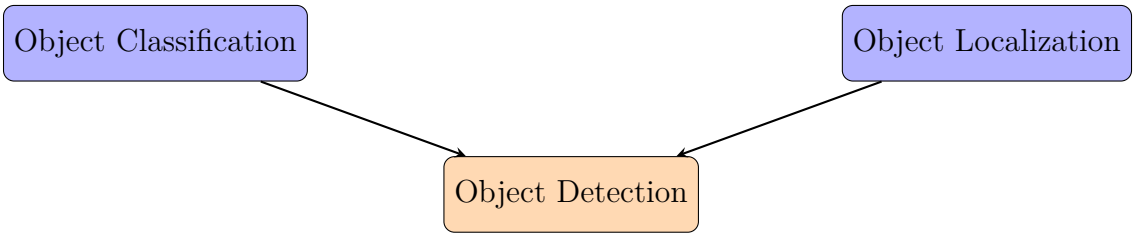
Chapter 4

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.

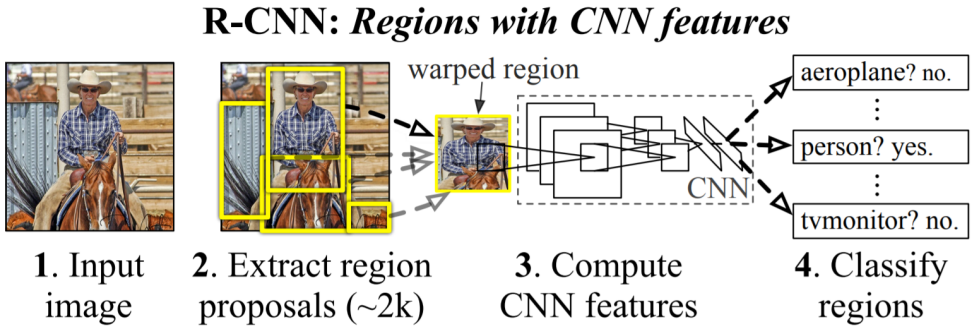


Figure 4.1: Object detection system overview

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.

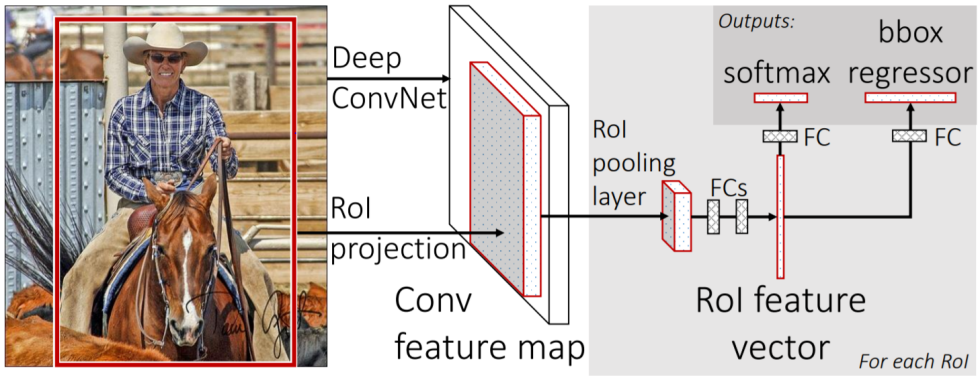
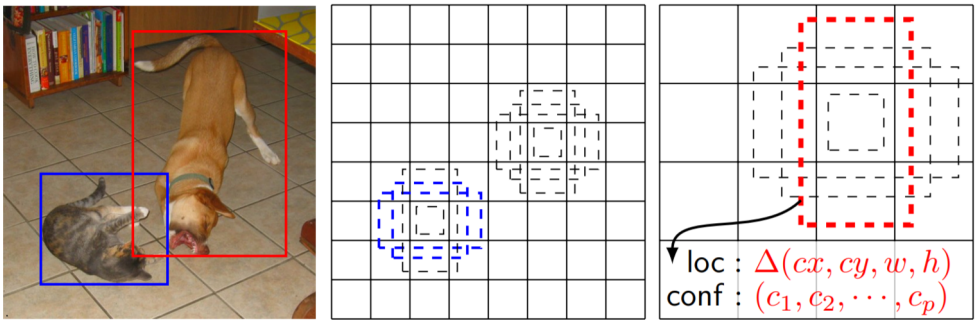


Figure 4.2: Fast R-CNN architecture

- **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 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.

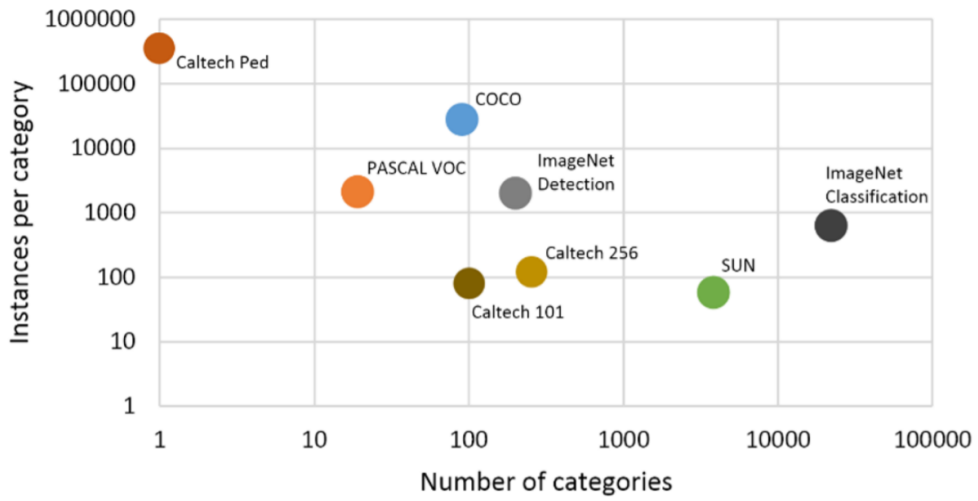


Figure 4.4: Comparison of different Object Detection datasets. COCO dataset presents a good tradeoff between *number of categories* and *instances per category*.

The different models are listed with their respective performance and precision. We have chosen to use **SSD ResNet50 V1 FPN 640x640 (RetinaNet50)** 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.

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 $\simeq 0.5$)

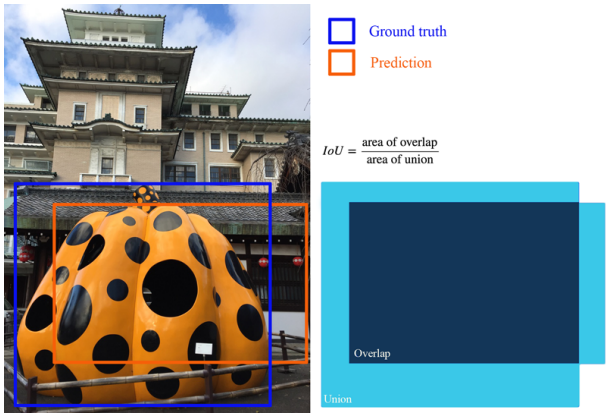


Figure 4.6: IoU is calculated by comparing the predicted region and the ground truth

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 $[0, 1]$, $AP \in [0, 1]$.

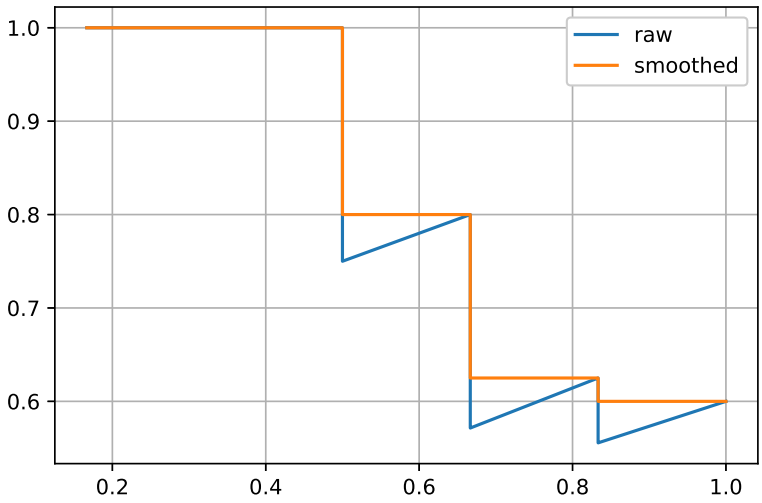


Figure 4.7: Precision - Recall curve

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...

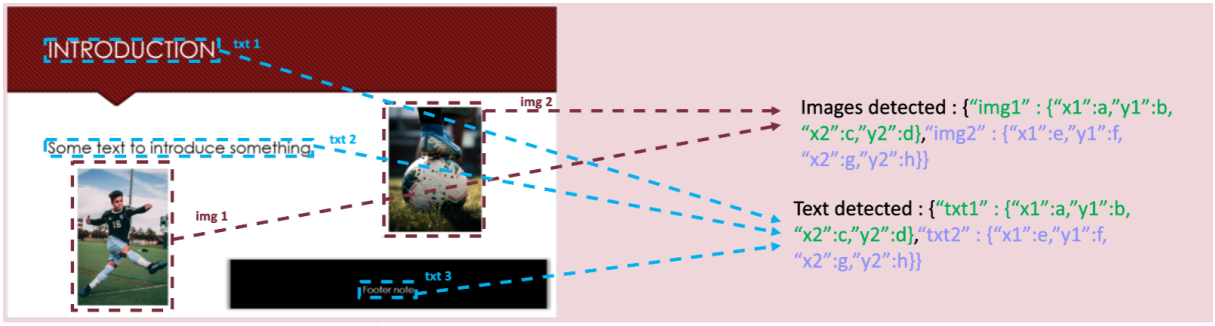


Figure 4.8: Example of objects to detect

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.

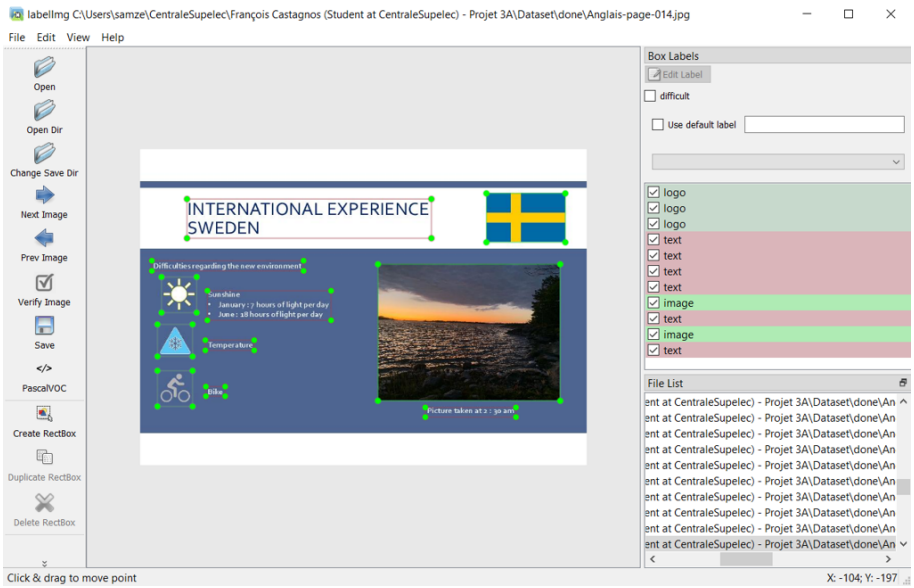


Figure 4.9: LabelImg software is used to annotate the images gathered

4.4 First results

After having fine-tuned a SSD RESNET model[1] 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.

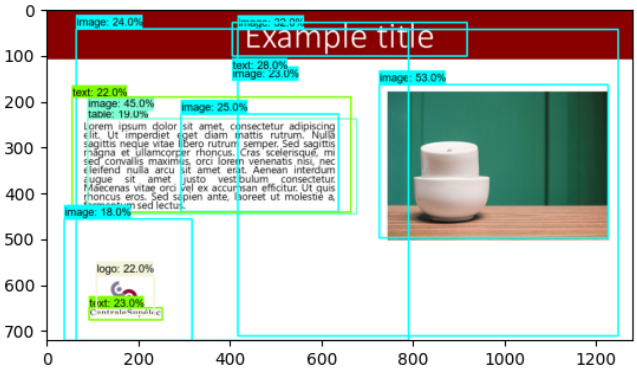


Figure 4.10: Object detection on an example slide using a SSD RESNET model

[1]Useful Link: Training Custom Object Detector

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.

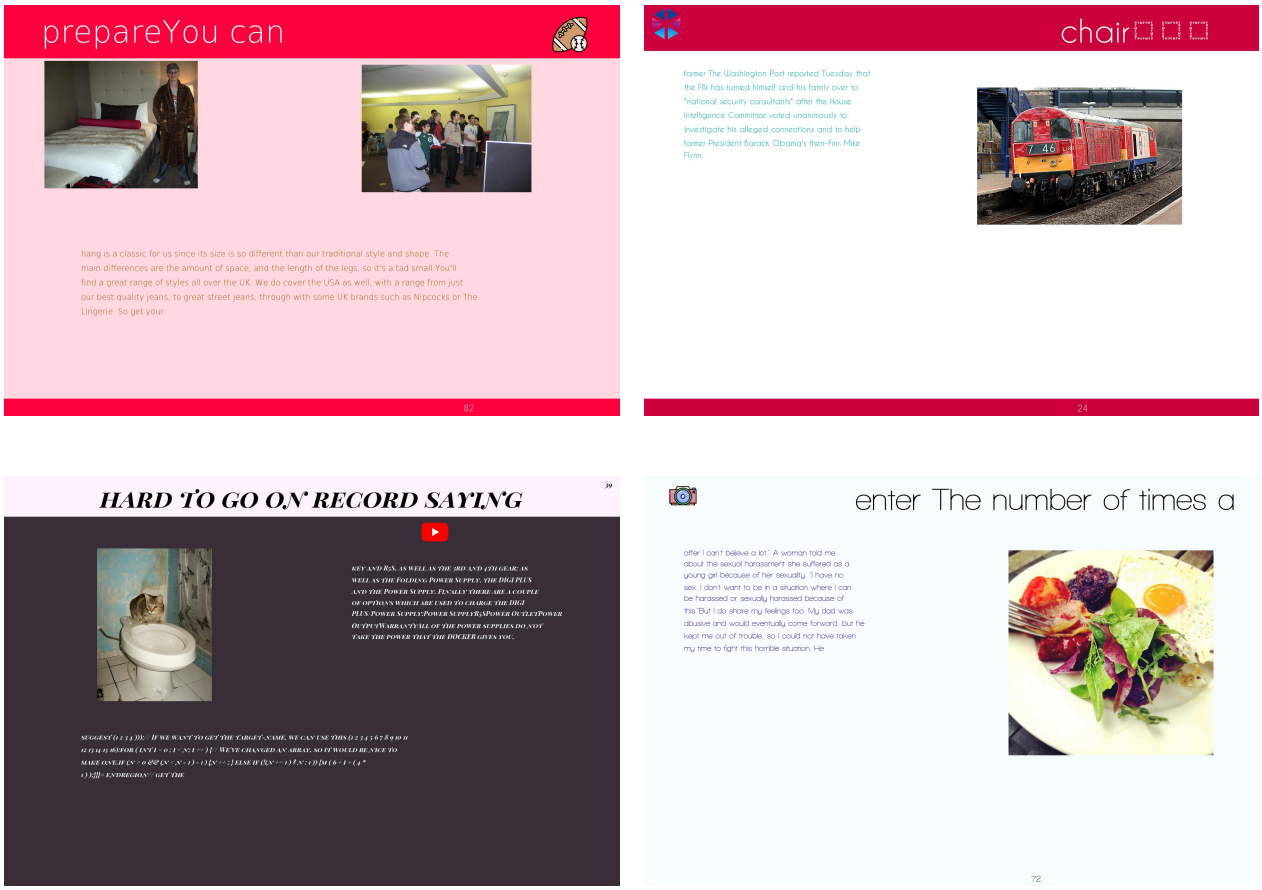


Figure 4.11: Sample of images generated

4.6 New results

Once the slides have been generated, we have been able to retrain our RESNET model on the whole new dataset. As the dataset is bigger, more varied and a bit simpler (less classes to detects, and images with less complex objects), we have been able to experience a big improvement.

Figure 4.12: A sample inference with the new model. Right: labeled input input. Left: Output of object detection

As we can see on Figure 4.12, the results were still not perfect:

- Titles and page numbers are still not detected, probably because they are of a different size than most text sections. One simple way to tackle this issue is to lower the detection threshold (confidence in a detection to be taken into account, default threshold is 50%)
- The boxes are not a 100% precise, especially for text sections

4.6.1 Metrics Results

Thanks to the Tensorboard functionnality, we are able to visualize the evolution of the training in real time.

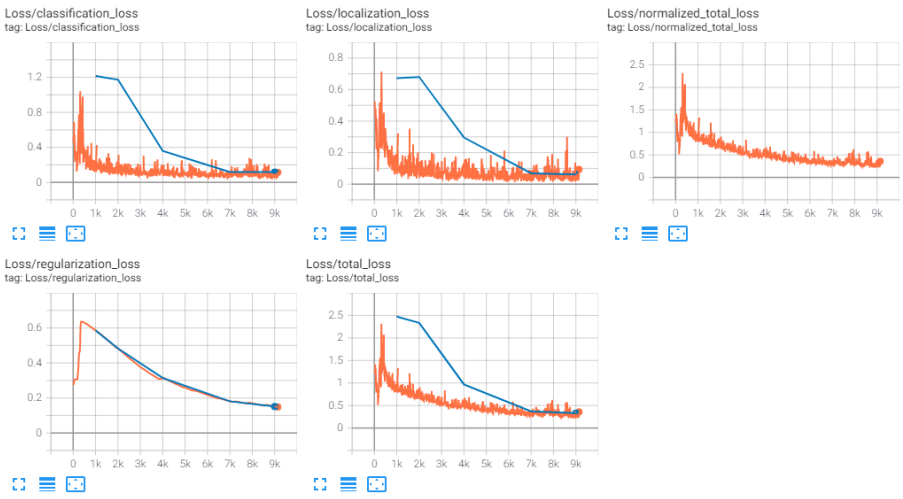


Figure 4.13: Loss evolution with the number of steps

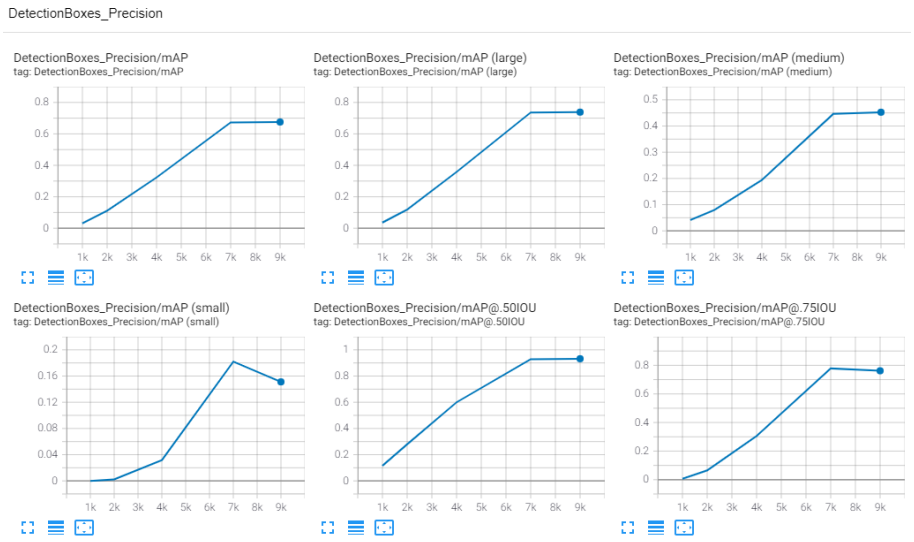


Figure 4.14: mAP evolution with the number of steps

Chapter 5

Text Recognition

Once the different objects are detected. We want to process them in function of their type. If an image is detected, it is only cropped and sent as it is. If a text is detected, we perform Optical Character Recognition (OCR) on it, which means that we convert the part of the image where the text is, into text, which takes a lot less storage.

5.1 Tesseract

For this part, we could have adopted 2 different startegies : implementing our own OCR or using an open source engine. For performances purpose and the sake of not reinventing the wheel, we chose the second solution.

In the field of OCR, a performing engine is Tesseract. Tesseract is a free software released under the **Apache Licence**. It was originally developped by **Hewlett-Packard** (HP), made open source in 2005, and have been developped by **Google** since 2006. It is used for text detection on mobile devices, in video, and in Gmail image spam detection [1].

5.1.1 Principle

In [Smith, 2007], the architecture of the engine is briefly explained. Basically, the algorithm first detect the different lines, then its tries to split the different characters. Finally, it tries to recognises the differents characters cropped. Recently, the engine have been improved

[1] For more information : visit tesseract-ocr.github.io

by adding a LSTM network, often used in Neural Language Processing, to ensure that the detected text is meaningful.

5.1.2 Python Wrapper

To allow Python developpers to use the Tesseract engine easily, a Python has been made available and can be installed as a library with pip (pytesseract

5.2 Results

The *pytesseract* library gives pretty good results as we can see with the sample image shown in Figure 5.1 which was cropped from one of the generated slides.

Figure 5.1: A sample image of a text section detected in a slideshow

Output of Pytesseract OCR

```
teacher had been a volunteer for the state's Board of Regents for six years before he took over for
Dean. "! don't know if it was the job, but he's going to be a very different person, he said of
Dean's leadership. "Even if he were to resign, | wouldn't be happy with any changes of my own to my
relationship with the governor or the police departmentall I'd agree with being the one to
decide."And because you didn't want to be part of it, | really decided against it. | wanted to be on
time and out of the office, and | decided | wouldn't be the only one."At a news conference
Wednesday, Anderson said he will
```

As we can see, the only error that occurs is that the word "I" is replace by a mid bar |. As this can easily be fixed, the *Pytesseract* is thus validated as the chosen OCR engine for our architecture.

Chapter 6

Background Reconstruction

After detecting correctly all the objects defining the slide, we aim here to recontruct its background, in order to be able to compress and send the object boxes separately, and then display them. We excpect to save the amount of information sent by storing the reconstructed background as it often remains unique in a PowerPoint presentation.

6.1 The PIL Library

To perform this task, we decided to use the Python Imaging Library. PIL is an image processing library that allows us to open, manipulate, and save different graphic file formats. Most importantly, it enables us to draw located rectangles on top of an image, function that we predominantly used.

6.2 Hypotheses

For our reconstruction algorithm, we made the following hypotheses:

- **The background is composed of unicolor bands only**
- **These bands are either horizontal or vertical (not both)**

These assumptions might seem under-simplifying the problem, but in fact they include a large amount of backgrounds, as highlighted by the examples below.

Figure 6.1: Example of slides following (or not) the upper hypotheses

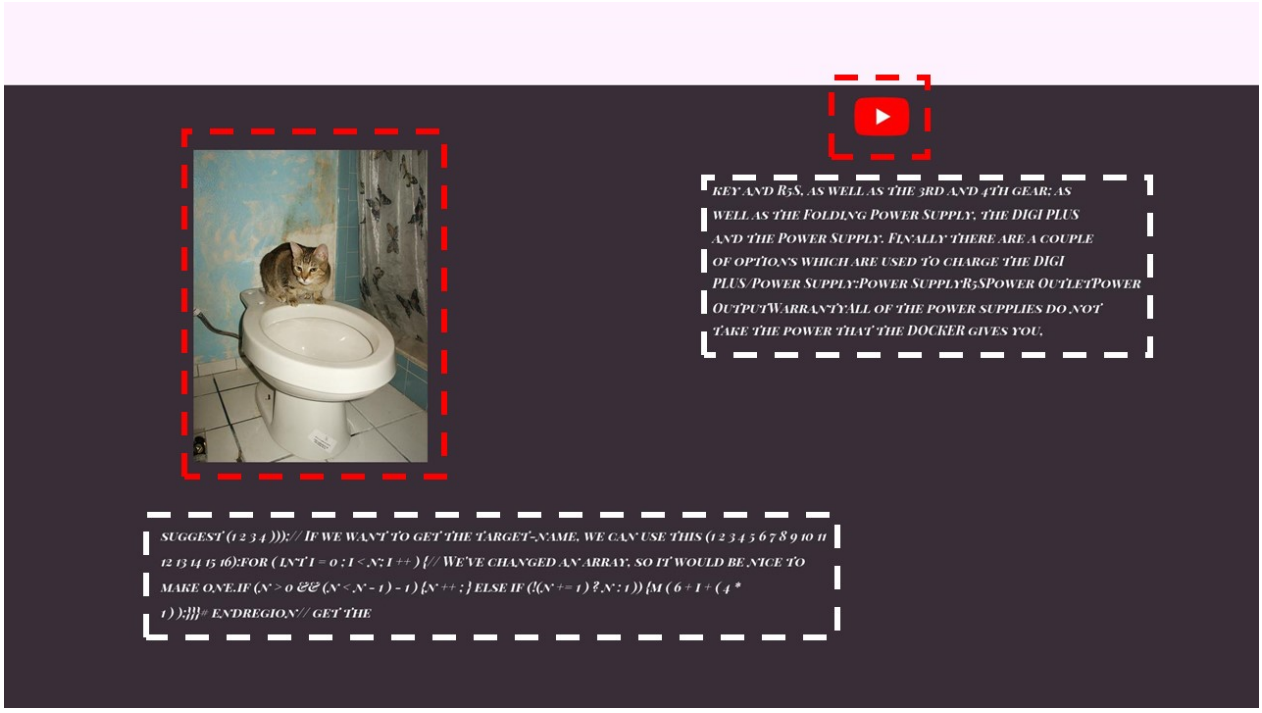
6.3 Algorithm description

The Background reconstitution function takes as input the current slide, and the boxes coordinates ([[ymin1, xmin1, ymax1, xmax1], [ymin2, xmin2, ymax2, xmax2]...]).

It is decomposed in three steps, for every box:

- horizontal band detection
- vertical band detection
- filling of the box

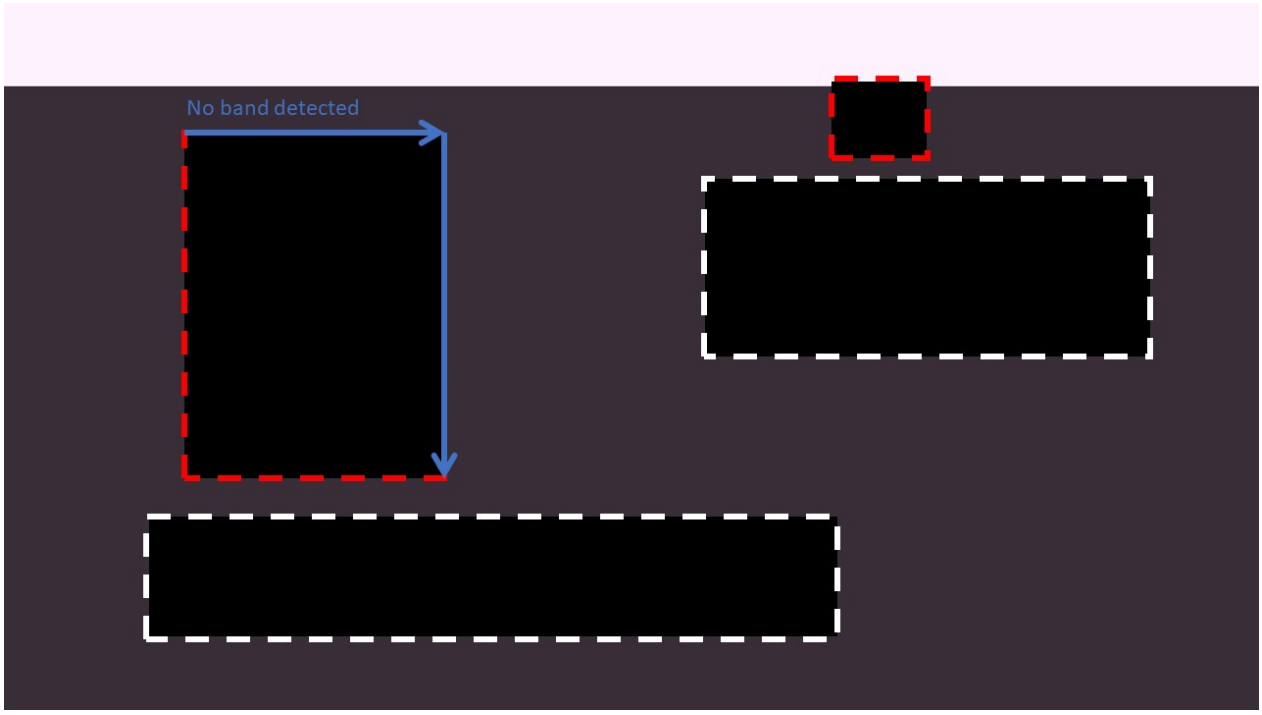
The best way to explain how it works it to walk through a simple example. Let's observe the slide below.



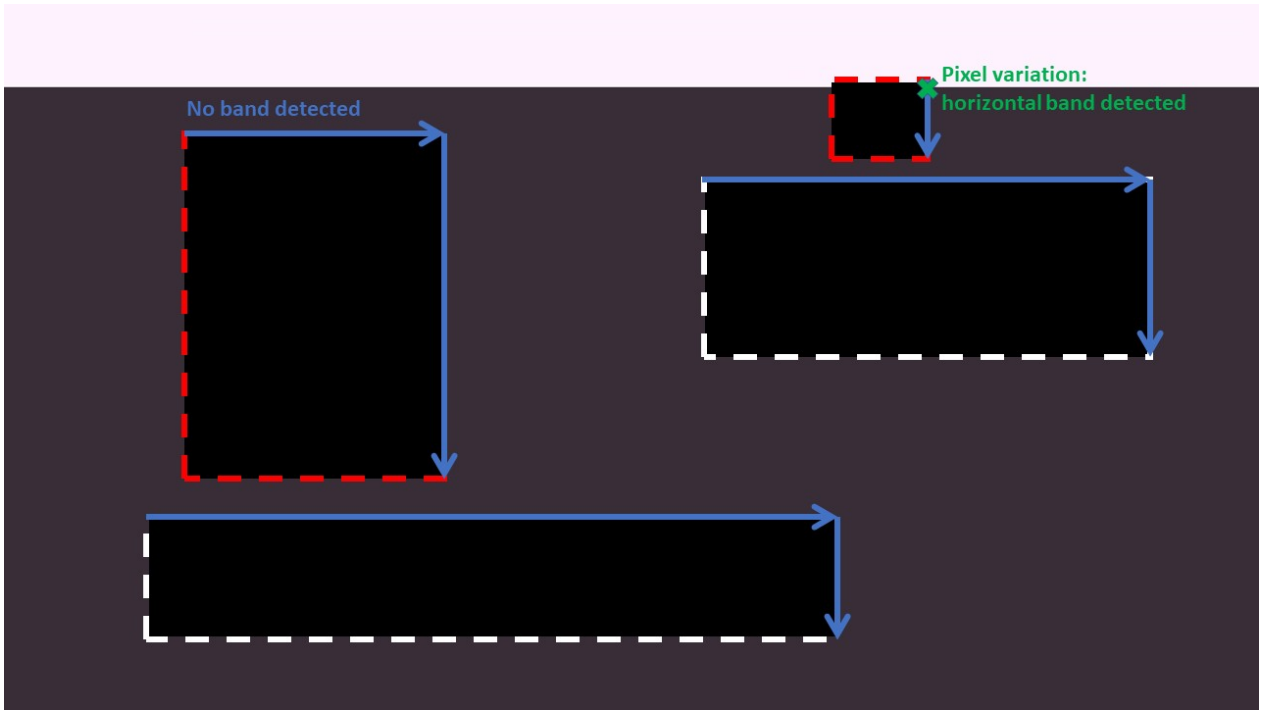
It is composed of two image boxes (in red), and two text boxes (in white). Behind, its background is purple with a white band.

First, we remove all the boxes contents, that will be compressed and sent separately.

We then focus on the first box, previously a cat image, and perform horizontal and vertical band detection. This task is done by checking along borders that there is no pixel variation (see arrows). As this box is entirely surrounded by the purple part of the slide, no band is detected.

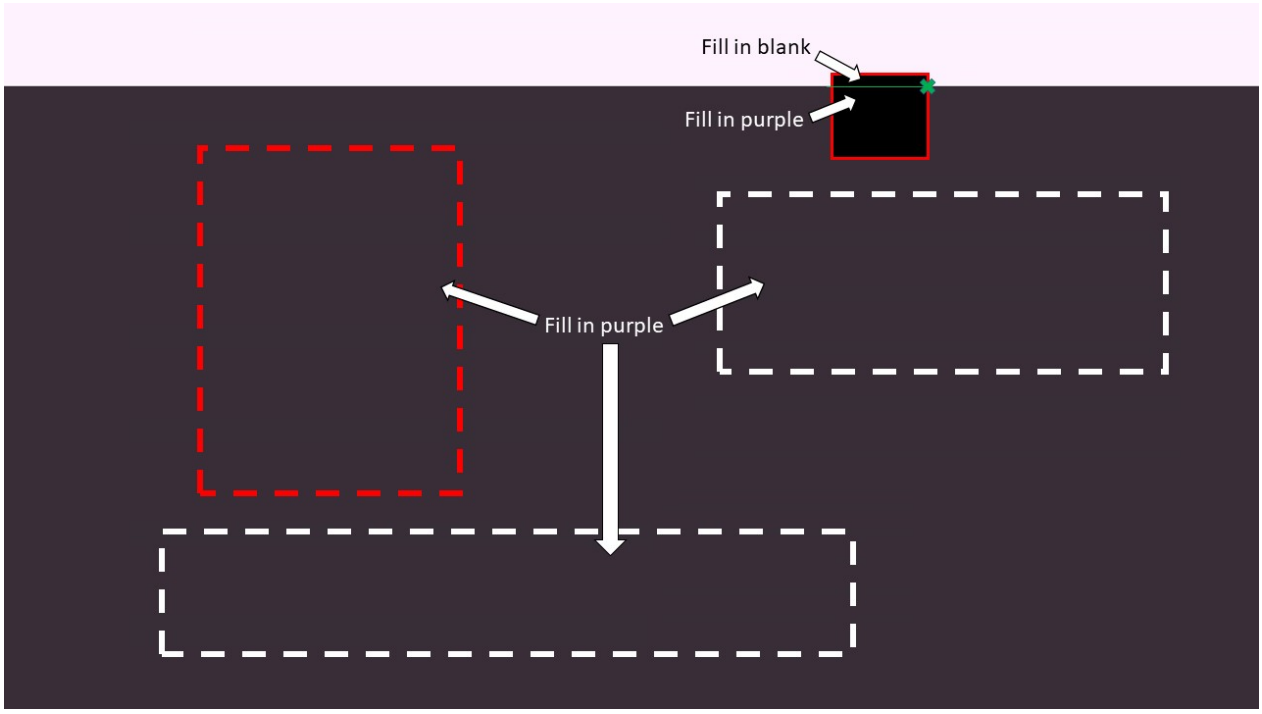


We then perform this task for all the boxes. No band is detected except for the Youtube logo.

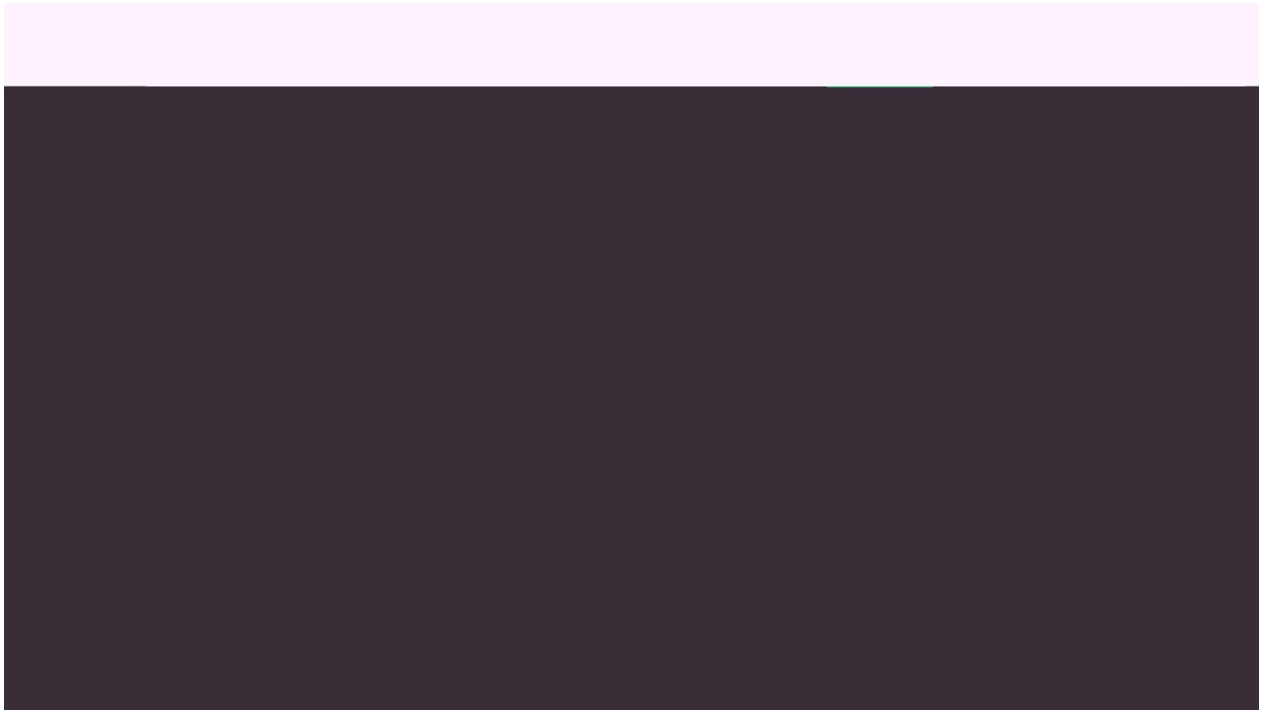


Let [ymin, xmin, ymax, xmax] the coordinates of the boxes. For all the boxes surrounded by one color (here purple), we fill them with the color of the pixel located in (xmin, ymin-1).

For the remaining box, we have detected two consecutive pixels that are different in the "horizontal check", one is white and one is purple. In the PIL Library, this means two different RGB values using the PIL.getpixel() function. Let (x0,y0) be the coordinates of the first purple pixel detected in the line.



We then fill the box in white for (x,y) in [xmin, xmax]x[ymin,y0-1] and in purple for (x,y) in [xmin, xmax]x[y0, ymax]. Finally, the initial background is reconstructed.



This version has however several drawbacks:

- It is vulnerable to multiple bands going through a box
- It is vulnerable to diagonal bands. However, a more elaborated version checking the four borders can be considered, but would increase the complexity of the algorithm.
- It requires a perfect object detection accuracy, otherwise we might detect false "pixel changes" across borders

Moreover, to devise an algorithm that recognizes more complex backgrounds, the possibility to exploit the data contained in text boxe scould be explored.

Chapter 7

Font Detection

In order to be able to reproduce completely any input slide, the last processing needed is font identification. Indeed, for each text section detected, we need to detect the text size, color and its font. As there is no clear solution available for this kind of task, our stategy was to train our own Neural Network.

7.1 The Dataset

Based on the same idea we had in the Object Detection part (introduce in 4.5), we have generated a dataset of images to train a neural network that is able to predict the font of a text inside an image.

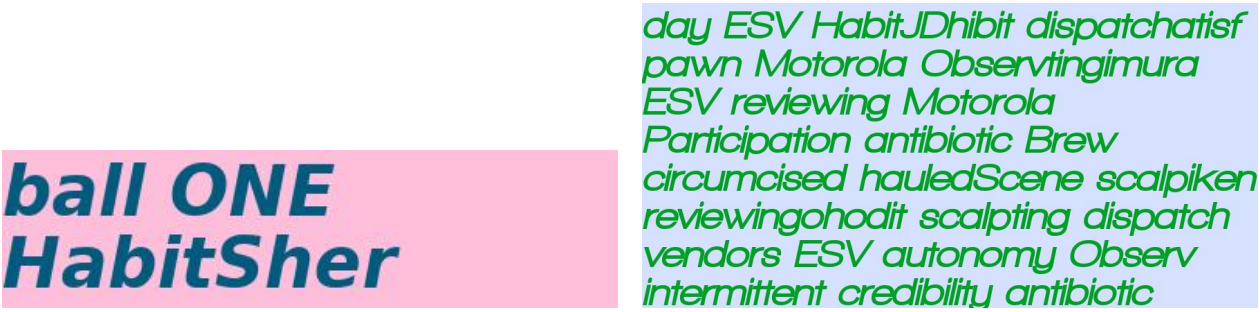


Figure 7.1: Sample of images generated

As we can see in the Figure ??, images can have different sizes, background color, text color and text font.

Training Set	Test Set	Labels
3432 images	870 images	(font,size,color)
		84 different fonts

Table 7.1: Dataset used for training

7.2 The Model

To initialize our model we have used the library Keras and Tensorflow. It is composed of 3 *2D Convolutional* Layers,3 *Maxpooling* Layers (One after each convolutional layer) and 2 *Dropout* Layers (To prevent overfitting[1]). The last layer is a *Dense* layer of length 84 because it corresponds to the number of different fonts to detect.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
rescaling_1 (Rescaling)      (None, 180, 180, 1)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 180, 180, 16)      160
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 90, 90, 16)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 90, 90, 32)        544
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 45, 45, 32)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 45, 45, 32)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 45, 45, 64)        2112
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 22, 22, 64)        0
_________________________________________________________________
dropout_3 (Dropout)          (None, 22, 22, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 30976)             0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               3965056
_________________________________________________________________
dense_3 (Dense)              (None, 84)                10836
=================================================================
Total params: 3,978,708
Trainable params: 3,978,708
Non-trainable params: 0
_________________________________________________________________
```

Figure 7.2: Summary of our Font Detection Model

[1]Dropout is form of regularization useful in training neural networks. Dropout regularization works by removing a random selection of a fixed number of the units in a network layer for a single gradient step. The more units dropped out, the stronger the regularization. Intuitively, the network does not rely on certain nodes in the graph. For full details : [Srivastava, 2014]

7.3 The Results

Our model was too simple for such a complex task. The results were not satisfactory as the accuracy on the detected fonts reached a plateau of 10%.

7.4 Transfer Learning

Just like we did in the Object Detection part, we have used a pre-trained model, here the **MobileNetV2** model (a good image classification model) have used it as a base model for our font detection task.

7.5 New Results

To evaluate our model, we have used 2 metrics :

- **Accuracy**: The accuracy is defined by the ratio of the correctly predicted images over the total number of images to predict. It represents the percentage of good predictions.
- **F1-Score**: During the training, the metrics used to mesure the progress was the accuracy (percentage of samples correctly labelled). Its main advantage is that it is easily understandable. However in the case of maximizing customer satisfaction, the model's accurary is not a proof of performance, especially when the classes to predict are very unbalanced. If 90% of the dataset belong to 1 class, and the rest to other classes, then having an accuracy of 90% is really easy and does not mean that the model will predict well the other classes. The F1-Score is often preferred to monitorize classification models.

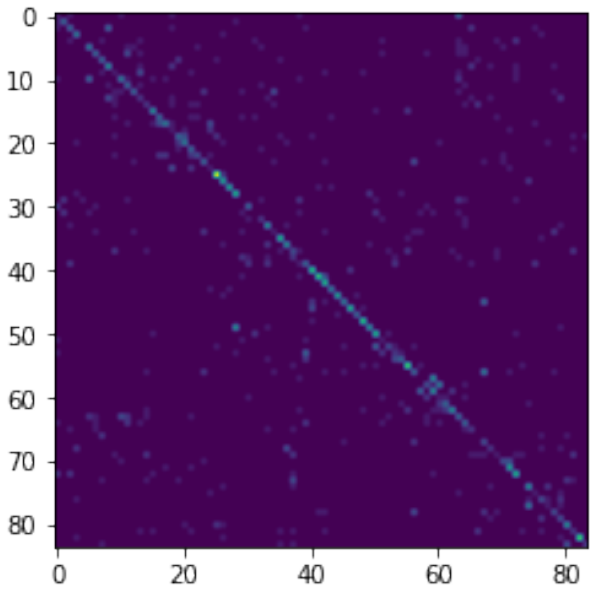
Moreover, to measure our model, for each images we have assigned 2 labels :

- **Labels**: The font and typos. Each image belongs to one of the 84 fonts (2 can differ if on is bold (*e.g.* "Times New Roman italic" and "Times New Roman Bold").
- **Sublabels**: The main fonts. Fonts such as "Times New Roman italic" and "Times New Roman bold" are grouped in the same class. There are 16 different sublabels.

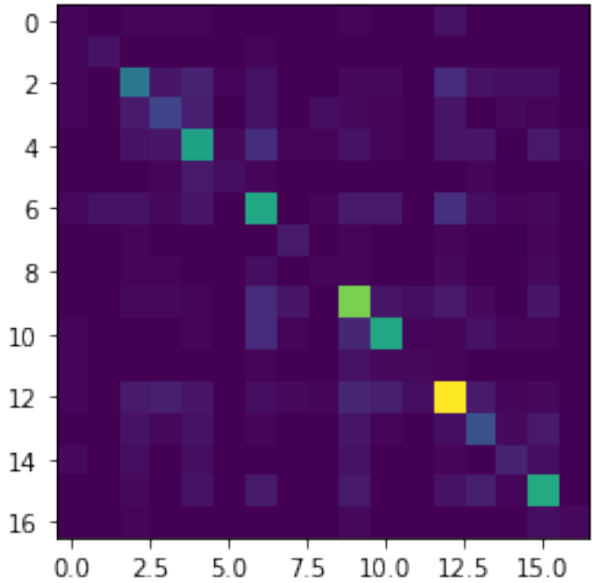
The results are the following :

Accuracy Labels	**Accuracy Sublabels**	**F1-Score Labels**	**F1-Score Sublabels**
0.402	0.523	0.384	0.426

Table 7.2: Performances of our font detection model



(a) Confusion Matrix on the labels



(b) Confusion Matrix on the sublabels

We can observe that the results have sharply increased by using a pre-trained model. However the performances are not satisfactory yet. Creating a bigger dataset and changing the hyperparameters for better fine tuning are possible solutions.

Chapter 8

Dash Application

8.1 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 of slide compression, but also to evaluate the code's performance.

First, we are focused on the app layout - the "Front-end.

Then, whe added 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".

8.2 Our App

Our App is declined into 5 sections, each of them picturing a step of the algorithm.We also added an "About" section with further information on the project. To view a demo of how it works, we have made this GIF

8.2.1 Object detection

In this section, which is also the welcome page, the user is invited to add a slide to compress (in JPEG or PNG) from its computer. Then, the slide appears on the page, with a "Run object detection" button. By pressing on it, the object detection model is called on the pipeline and runs, before returning the image with the detected objects.

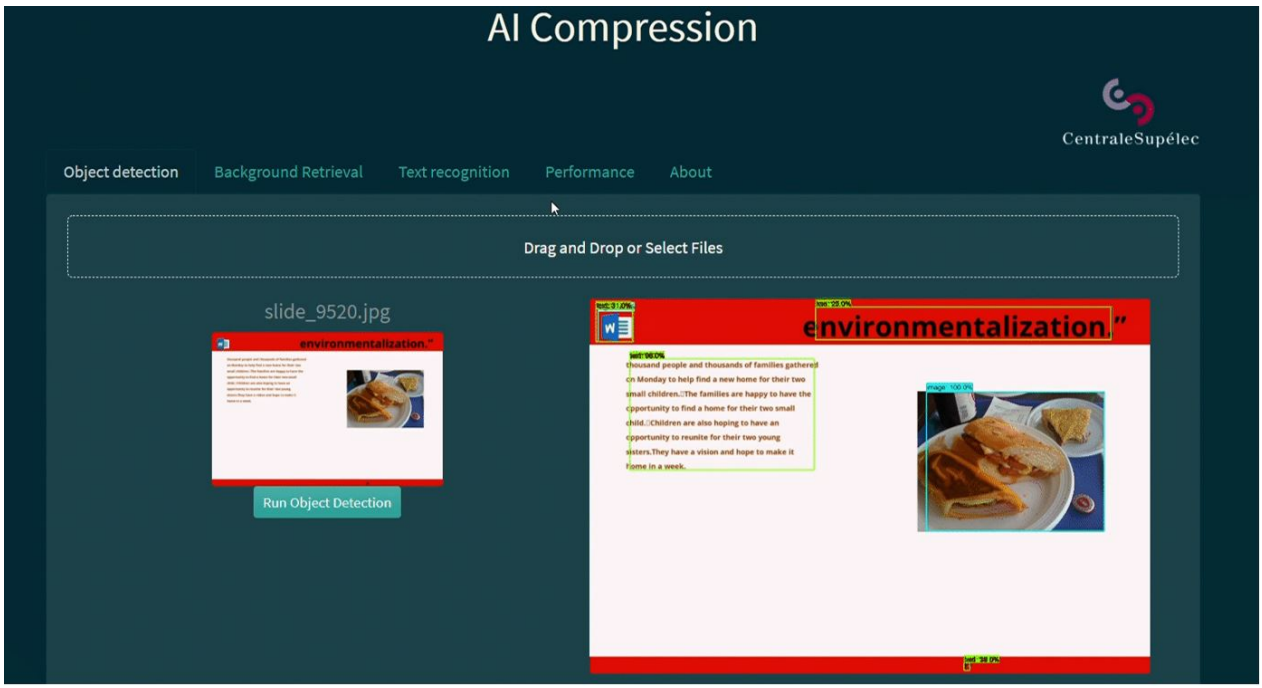


Figure 8.1: An overview of the Object Detection section

After getting this output, the user can simulate the output of the backround retrieval function.

8.2.2 Background retrieval

Here, the user must have added an input slide on the previous section and run object detection before making the simulation. The "run" background retrieval" button is displayed only when these tasks have been performed.

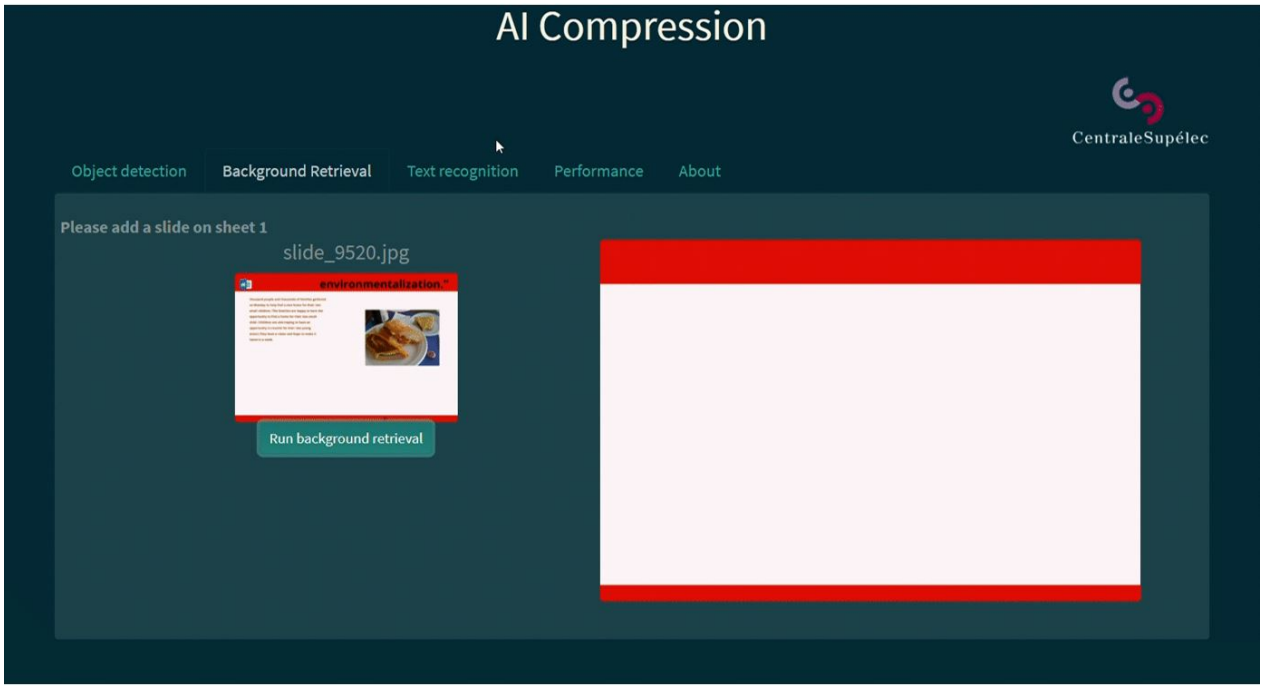


Figure 8.2: Overview of the Background section

After pressing this button, the interpreted beckround is displayed. The program keeps the object detection output in memory so the execution time remains small.

8.2.3 Text recognition

In this 3rd section, the uses can check how all the detected text boxes have been detected and interpretated. It is composed of screenshots of texts in the slide, next to interpretations from our OCR program. The interpretation is only at a "character" level so far, even though we planned to display the accurate font too. However, our font detection model is not performant enough to be integrated.

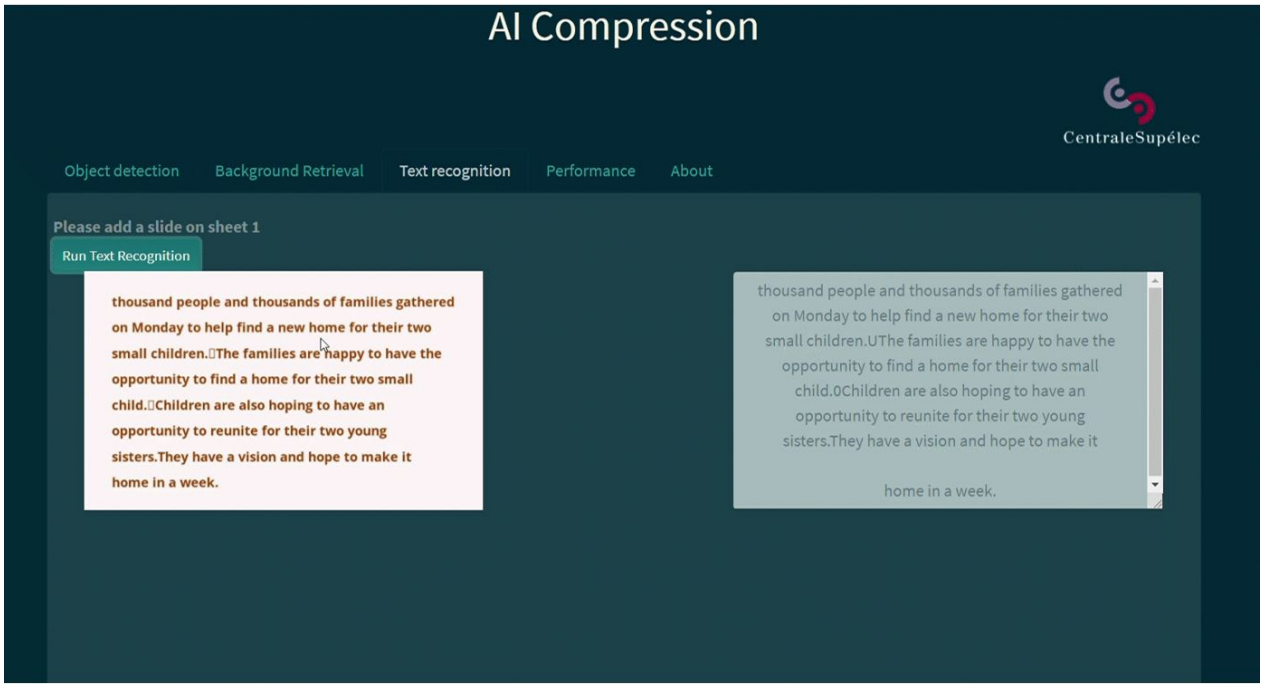


Figure 8.3: Overview of the Text Detection section

8.2.4 Performance

This section is expected to be evaluating the compression performance of our model. It is supposed to compare, after the whole slide reconstitution, the economy in data sent between the initial slide and the background + different objects sent separately.

This section is in standby so far, as the "slide reconstitution" program couldn't be coded yet.

A more optimized version would be to perform these calculations on a live presentation flow, which would give us a real case performance evaluation.

Chapter 9

Conclusions

This project has helped us to work on a wide range of topics, from subject research to app development, from Natural Language Processing to Object Detection. As a result, we were able to develop many skills, that will for sure be valuable for us in our professional career.

The results are encouraging so far, we were able to accurately identify text, images and logo objects on a slide, remove them and reconstitute the initial background of the presentation, and perform text recognition on text objects. All of those programs are executable on an intuitive app, implemented with Dash Plotly. We truly believe that this project could lead to a good compression solution, that might have potential for a real usage. This is why we made our code available online so that other groups could take over the work done. The following aspects could be deepened:

- Monitor the model performance in compression
- Apply and/or adapt the model to live video streams

Regarding the process, we faced early obstacles during the dataset building that were time consuming and hence led us not being able to perform end to end video compression, as the "full slide reconstitution" part could not be implemented in time. These obstacles however enabled us to adapt our strategy in order to move forward, which also represents a valuable skill developed during the project.

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. 20).

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

Smith, Ray (2007). 'An Overview of the Tesseract OCR Engine'. In: (cit. on p. 32).

Srivastava, Nitish (2014). 'Dropout: A Simple Way to Prevent Neural Networks from Overfitting'. In: *Journal of Machine Learning Research* (cit. on p. 41).