ECOLE
POLYTECHNIQUE
DE BRUXELLES

ULB

UNIVERSITÉ LIBRE DE BRUXELLES

Exploring People Counting using Vision Transformers

An effectiveness aware study

Mémoire présenté en vue de l'obtention du diplôme
d'Ingénieur Civil en Informatique à finalité spécialisée

Maxime Langlet

Directeur
Professeur Olivier Debeir

Service
LISA: Laboratoire de l'Image: Synthèse et Analyse

Année académique
2021 - 2022

ULB UNIVERSITÉ LIBRE DE BRUXELLES

ULB UNIVERSITÉ LIBRE DE BRUXELLES

Exemplaire à apposer sur le mémoire ou travail de fin d'études,
au verso de la première page de couverture.

Fait en deux exemplaires, **Bruxelles, le ……..**

Signature

Réservé au secrétariat :	**Mémoire réussi***	**OUI**
		NON

CONSULTATION DU MEMOIRE/TRAVAIL DE FIN D'ETUDES

Je soussigné

NOM : Langlet

PRENOM : Maxime

TITRE du travail : Exploring People Counting using Vision Transformers An effectiveness aware study

AUTORISE*

~~**REFUSE***~~

la consultation du présent mémoire/travail de fin d'études par les utilisateurs des bibliothèques de l'Université libre de Bruxelles.

Si la consultation est autorisée, le soussigné concède par la présente à l'Université libre de Bruxelles, pour toute la durée légale de protection de l'œuvre, une licence gratuite et non exclusive de reproduction et de communication au public de son œuvre précisée ci-dessus, sur supports graphiques ou électroniques, afin d'en permettre la consultation par les utilisateurs des bibliothèques de l'ULB et d'autres institutions dans les limites du prêt inter-bibliothèques.

Formulaire version 04/2015

** Biffer la mention inutile*

** Biffer la mention inutile*

Acknowledgments

I would like to thank Pr. Olivier Debeir for his continued help knowledge and guidance during this year. I would like also to thank Mr. Feras Almasri for the tips, clever ideas, and guidance.

I would also like to thank my friends and loved ones for helping me, challenging me, guiding me for all these years, and making me a better version of myself, for the good moments as well as the bad ones. Specifically, Nathan, Henri, Thomas, Adilson, Ahmed, Lucas, Pierre, Louan, Pacs, Eliott, and Mara, thank you for everything.

Lastly, I would like to thank my family, especially my mother, for always being there for me, offering help, and pushing me to do my best. I also have a short thought for my father, whom I am sure is proud of me.

Abstract

This master's thesis explores the problem of people counting. The main objective is to find an efficient way to count people in a classroom where the pictures are taken by a microcontroller, the predictions are also done on the same electronic module. The initial idea was to count by detection using face detection. The avenue of crowd counting was also explored. Well-established techniques are already available, to make this master's thesis original, we explore the possibility to apply Vision Transformers to both problems (face detection and Crowd counting).

The DETR model for object detection was used for the face detection. It was fine-tuned in order to detect faces on the WIDER face data-set, using the ability of transfer learning of the backbone convolutional network. Although the results are disappointing compared to state of the art methods, some hope is present for future work. It will be shown that, despite it's poor performance, bigger faces are being recognized reasonably well.

For crowd counting, no previous work seems to have been done including Vision Transformers (ViT). Hence, a custom approach was explored. A naive regression and a density map generator inspired by CSRNet, both using Vision Transformers, were investigated. Some interesting results arose, despite the lackluster performance.

In conclusion, this master's thesis did not find any revolutionary methods for crowd counting or face detection, but some insight is given into ViTs, how they work and what seem to be their limitations. Some state-of-the-art methods were still found for the detection problem on the microcontroller, improving on the previous MTCNN method used.

Contents

List of Figures

Introduction

This master's thesis was undertaken in the context of the project *People detection and counting GDPR by design* organized by the Laboratory of Image Synthesis and Analysis and supervised by Olivier Debeir. Since this project already has an initial solution that works well enough, the need for an innovative subject was required. The goal is to detect and count people in a scene, which might be for example a classroom, efficiently since the image acquisition and counting process will be executed on a Raspberry Pi. To do so, two avenues can be taken, counting by detection or crowd counting. Face detection is one of the fundamental problems at the center of pattern recognition and computer vision. It plays an important role in multiple face-related applications. Furthermore, another aspect of the project is crowd counting, which will also be investigated. Crowd counting, like face detection, also has some solutions available for estimating the number of people in large crowds.

Traditionally, computer vision applications are dominated by Convolution Neural Networks (CNNs). Recently, Vision Transformers (ViT) has taken the computer vision world by storm, showing close to state-of-the-art performances in various applications. While the Transformer architecture is usually associated with natural language processing tasks, its application to computer vision is still new. Hence, the project revolves around face detection and "crowd" counting using ViTs, some relatively new applications to such an architecture. Vision transformers have demonstrated many benefits over previous CNNs architectures. Some of them are : **General modeling capabilities**, **Convolution complementarity**, **Stronger modeling capabilities**, **Scalability to model and data size** and **Better connection of visuals and language**[2].

0.1 Objectives

The objective of the thesis is to build a solution to counting people (in a classroom) efficiently since the data acquisition and prediction are done using a Raspberry Pi4 8GB with a raspberry HQ camera with a wide-angle camera lens. The OS used on this module is a 32 bits Raspbian (based on Debian 10(Buster)). To do so, since people are seated, the most straightforward way should be to detect faces present in the classroom. Note that, the LISA laboratory already implemented a solution using the MTCNN method. We will discuss to see if this technique is the best one available in future sections. Crowd counting can also be applied, the advantage of this method will be the output of a density map, showing the distribution of faces.

All the computations will be done on the before said Raspberry Pi, efficiency will be also an important aspect of this study. An example of the type of images captured that will be studied can be found in Figure 1. We can see that the face size in the figure can vary a lot in size.

Figure 1: Example of image captured by the Raspberry Pi

Of course, generalization is important, hence the software should be able to detect faces or count crowds well enough no matter the environment, size, orientation, etc. In that case, we will use data sets with a large representation of faces and crowds rather than restricting ourselves to images captured by our electronic module.

Furthermore, it's important to find an original way to tackle the problem. The existence of Vision Transformers was found early in the research phase of this master's thesis. It is an existing technique but newly applied to computer vision applications, hence the original part of the thesis was found. Let's first explore the problem of face detection.

0.2 Face detection

Face detection has become an important task in recent years, it is the backbone of multiple face-related applications like face recognition, face-tracking, expression analysis, etc. It has always been a challenging problem since its emergence in the 1990s. With the progress of CNNs comes solutions for our particular problems. However, some challenges still present themselves for face detection problems, for instance, CPU utilization is a clear bottleneck in state-of-the-art solutions since we can have large visual variations of faces in images and a large search space of possible face positions and sizes. High-accuracy detectors tend to be computationally expensive. It is paramount that our solution is efficient since the acquisition and detection will be done on a Raspberry Pi, hence restricting

computational resources.

Commonly, face detection methods, and more broadly object detection methods address the prediction task by looking into smaller portions of images, reducing the detection in smaller classification problems on a large set of proposals, anchors, or window centers.

0.3 Crowd Counting

Crowd counting has become an interesting task recently due to the pandemic. It can be useful to have tools estimating the number of people in a given space. To do so, multiple methods exist:

- Detection-based methods
- Regression-based methods
- Density estimation-based methods
- CNN-based methods

The detection-based method will be more detailed in section 0.2, we will thus concentrate also on regression-based methods and CNN-based methods.

0.4 Vision Transformers

Transformers have become the go-to for natural language processing tasks, but their application to image processing was still limited until recently. Vision transformers are models for image classification that employs Transformer-like architecture over patches of an image. These patches are fixed-sized, each of them linearly embedded, position embeddings are added, and the resulting vectors are fed to a standard Transformer encoder. The classification is performed by adding an extra learnable "classification token". Vision transformers attain excellent results compared to state-of-the-art convolutional networks, while requiring substantially fewer computational resources to train, but need to train on a lot of images (14 million training images to obtain state-of-the-art classification results).

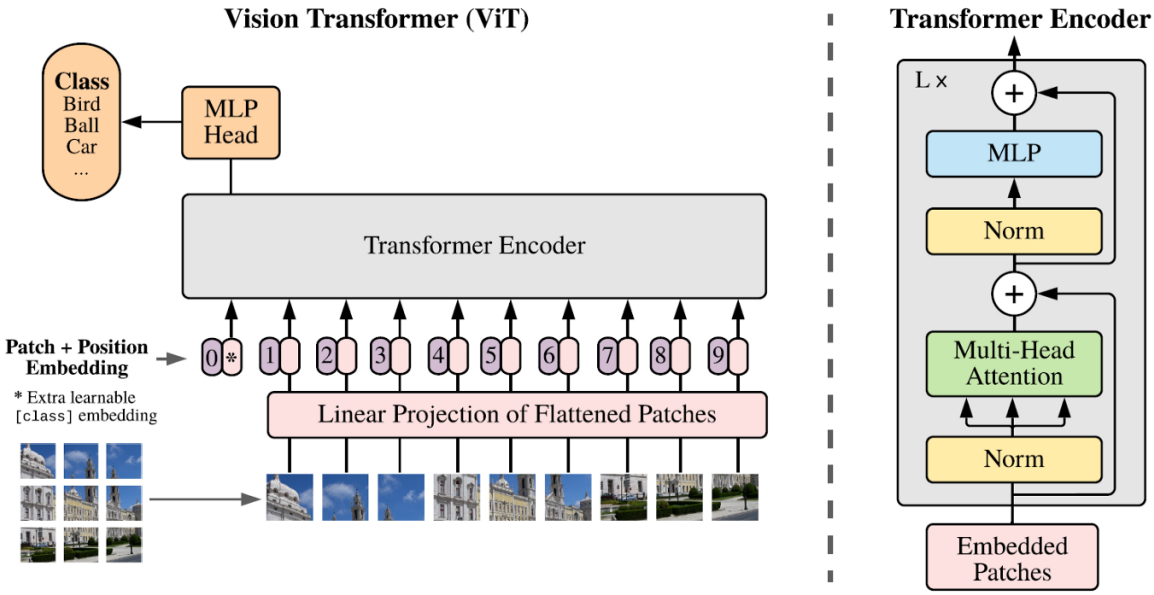


Figure 2: ViT Model Overview

More precisely, an image $x \in \mathbb{R}^{HxWxC}$ is resized into a sequence of *flattened* 2D patches $x_p \in \mathbb{R}^{Nx(P^2.C)}$ where (H, W) is the original size of the image, C is the number of channels, (P, P) is the size of each patches and $N = \frac{HW}{P^2}$ is the resulting number of patches. Transformers uses a constant size vectors for all of its vectors, so the patches extracted are flattened and mapped to D dimensions with a trainable linear projection. This projection will be referred as the patch embeddings. A learnable embedding is added to the sequence of embedded patches. In classification applications, a Multi Layer Perceptron (MLP) with one hidden layer is present at the output of our model. The position embeddings are added to the patch embeddings to retain positional information of those patches, which use a standard learnable 1D position embedding. The resulting sequence of embedding vectors serves as input to the encoder. The transformer encoder is composed of multiple layers of multi-headed self-attention and MLP blocks (refer to section 3.3 for more details about Multi-Head attention), where a layernorm (LN) is applied before every block, and residual connections after every block [9].

Let's note that ViT has less image-specific inductive bias than CNNs. Indeed, ViTs, only the MLP layers are local and translationally equivariant, while the self-attention heads are global, whereas CNNs use local 2D neighborhood structures and translation equivariance is used at each layer of the entire model. While ViT also uses a 2D neighborhood structure, it does so very sparingly, only at the beginning of the model by cutting the image and at fine-tuning time for adjusting the position embeddings. Indeed, the spatial relations between the patches have to be learned and the positional embeddings contain no information about the 2D positions of the patches at initialization time.

The next sections will cover different aspects of the study of this master's thesis. The first one will be covering various *state-of-the-art* (chapter 1) methods that will be used in this master's thesis. The next one will detail the *Material* (chapter 2), going over the different data sets used and analyzed, detailing also the specs of the computer utilized. Next, the *Methodology* (chapter 3) for the different algorithms will be detailed. That is explaining the overall pipeline, then going more into the specifics in explaining the possible pre-processing done on data. Then explaining the architecture of the various models studied as well as define the training details of each of them. Then, the *Evaluation* chapter (chapter 4) will explain the different methods of comparing and evaluating the multiple algorithms brought forward. After that, a *Results* chapter (chapter 5) will present the different results and analyze them. Lastly, a *Conclusion* chapter (chapter 6) will conclude the findings and the possible future work to be explored with the Vision Transformers.

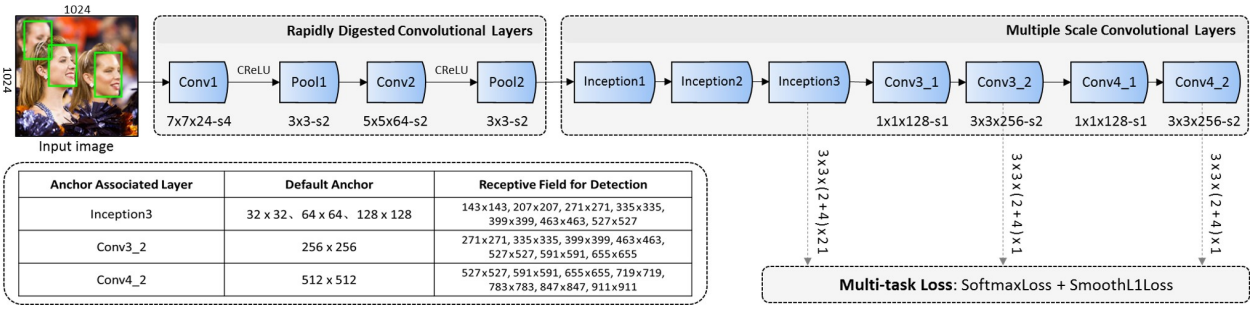
Chapter 1

State-of-the-Art

This chapter will bring forward different studies concerning face detection using deep learning, object detection using Vision Transformers, and Crowd Counting methods. Let's recall the need for an efficient algorithm since our detector will run on a small electronic module with limited computing power. Hence, the research guided us to the following sub-section: Faceboxes [36], which is a face-detector achieving real-time detection on a CPU while maintaining high performance. We will then follow this study by looking at object detectors using Vision Transformers. In particular, this research detailed is : Detection Transformers [6]. Then we will introduce the state-of-the-art method for crowd counting, CSRNet [20].

1.1 Faceboxes

The Faceboxes method can be divided into three main contributions : **the Rapidly Digested Convolutional Layers (RDCL)**, **the Multiple Scale Convolutional Layers (MSCL)** and **the anchor densification strategy**. We will see the details of each methods next. It turns out that the **anchor densification strategy** is crucial for accurate predictions, while **MSCL** increases the performance slightly. Finally, the **RDLC** is efficient (allows real-time CPU predictions) and accuracy preserving. We will use the Faceboxes algorithm as a baseline for face detection accuracy and efficiency.

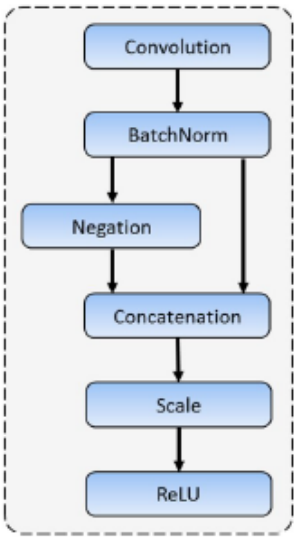


Anchor Associated Layer	Default Anchor	Receptive Field for Detection
Inception3	32 x 32、64 x 64、128 x 128	143x143, 207x207, 271x271, 335x335, 399x399, 463x463, 527x527
Conv3_2	256 x 256	271x271, 335x335, 399x399, 463x463, 527x527, 591x591, 655x655
Conv4_2	512 x 512	527x527, 591x591, 655x655, 719x719, 783x783, 847x847, 911x911

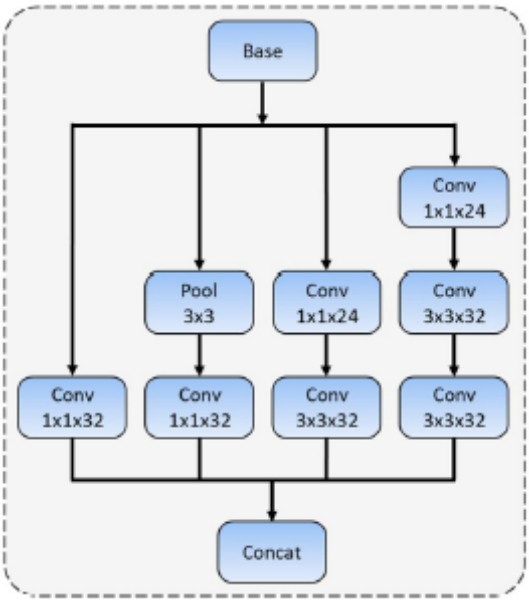
Figure 1.1: Faceboxes architecture with detailed information table about the anchor designs

1.1.1 Rapidily Digested Convolutional Layers (RDLC)

This first step is designed to shrink the input spatial size rapidly. To do so, the RDLC sets a series of large stride sizes for its convolution and pooling layers. More specifically, the first layers Conv1, Pool1, Conv2, and Pool2 use stride size 4,2,2, and 2 respectively, adding up to 32 times reduced input in 4 layers. This is the main contribution that makes this method achieve real-time speeds on CPU devices. Appropriate kernel size is important for the first few layers. Indeed, the kernel size of one network should be small to speed up, while large enough to ease the information loss of the spatial reduction. The authors thus chose kernel sizes 7x7, 5x5, and 3x3 for Conv1, Conv2, and the pooling layers respectively (Figure 1.1).
Lastly, they reduce the number of output channels by using the C.ReLU activation function (Figure 1.2a). This activation function is motivated by the observation that, in CNNs, the filters in the lower layers form pairs. Thus C.ReLU can double the number of output channels by concatenating outputs before applying the ReLU. An observation is that C.ReLU increases the speed of the software while only having a negligible decline in accuracy.



(a) C.ReLU modules where Negation simply multiplies -1 to the output of Convolution



(b) Inception modules

Figure 1.2: C.ReLU and Inception modules

1.1.2 Multiple Scale Convolutional Layers (MSCL)

The MSCL method is based on Region Proposal Networks (RPN), which is perfect for single detection tasks, like face detection. Usually, RPNs are not able to reach competitive performance, which may come from two aspects. Anchors are associated with the last

convolutional layer of the model whose feature and resolution are too weak to detect variable-sized faces. Then, the anchor-associated layer has only a single receptive field within its range of scales and has difficulty matching different scales of faces. The MSCL will solve those problems the following way :

- The MSCL contains multiple layers which decrease in size progressively to form some multi-scale feature maps. The anchors will be associated with those multi-scale feature maps. These multi-scale designed layers help with faces of various sizes.

- The output features of the anchor-associated layers should cover different sizes of the receptive field, which is covered by the Inception module. It has multiple convolution branches with different kernels, which enriches the receptive field. As advantage, the Inception module is a cost effective module that helps capturing different scales of faces. It's implementation is shown in Figure 1.2b

1.1.3 Anchor Densification Strategy

Firstly, the authors impose a 1:1 aspect ratio for the default anchors since bounding boxes of faces can be considered square. The last Inception module has anchors of sizes 32, 64, and 128 pixels. For Conv3_2 and Conv4_2 layers (see Figure 1.1), the anchor boxes are 256 and 512 pixels respectively. The tiling density follows this equation :

$$A_{density} = \frac{A_{scale}}{A_{interval}} \tag{1.1}$$

where A_{scale} is the scale of the anchor and $A_{interval}$ is the tiling interval of the anchors. For example, the stride size is 64 pixels for Conv3_2 and the anchor is 256x256 pixels, meaning there is an anchor of 256x256 every 64 pixels on the input image. A tiling density imbalance occurs for our current strides and anchor size, causing a low recall rate of small faces.
To solve this issue, an anchor densification strategy is proposed. To densify one type of anchor n times, they uniformly tile $A_{number} = n^2$ anchors around the center of one receptive field instead of only tiling one at the center of this receptive field to predict. In particular, 32x32 anchors and 64x64 anchors are densified 4 and 2 times respectively. This guarantees the maintenance of the tiling density for different-sized anchors.

1.1.4 Training

The training was done on 12 880 images of the WIDER face training subset. Data augmentation was applied to the images. Some of these augmentation techniques were: Color distortion, Random cropping, Scale transformation, Horizontal flipping, etc. The matching strategy (i.e. determination of which anchor corresponds to a face) is the best corresponding Jaccard overlap and then matches anchors to any face with Jaccard overlap higher than a threshold of 0.35. The loss function is the same as for RPN, hence a 2-class softmax loss for classification and the smooth L1 loss for regression.

1.1.5 Results

It was found that the Faceboxes method can achieve up to 20fps on CPU with state-of-the-art accuracy. It has a lightweight yet powerful structure with the components of RDLC and MSCL. The Faceboxes detector is fast enough to achieve 20 FPS for VGA-resolution images on CPU and can be accelerated to 125 FPS on GPU.

1.2 Detection Transformers (DETR)

The DEtection TRansformers (DETR) method, which we will use it for face detection, tackles the task of object detection. Its goal is to predict a set of bounding boxes with a label for each object of interest. To simplify the post-processing steps of other detectors, the DETR proposes a direct set prediction approach to bypass the surrogate tasks. The authors decide to view object detection as a set prediction problem. ViTs, which incorporate self-attention mechanisms, make their architecture adequate for specific constraints of set predictions like removing duplicate predictions. In particular, the DETR detector predicts all objects at once and is trained with a loss function that performs bipartite matching between the prediction and the ground truth. It also deletes the use of hand-designed components like non-max suppression or spatial anchors.

The main features of DETR are the conjunction of the bipartite matching loss with non-autoregressive parallel decoding. It turns out that this new model performs similarly to Faster R-CNN, with better performance on big objects (maybe related to the non-locality computations of the transformers) and lesser performance for smaller objects. The DETR model requires long training schedules and benefits from auxiliary decoding losses in the transformer [6].

Two ingredients are essential for the direct set predictions in detection :

- a set prediction loss that forces unique matching between predicted and ground truth boxes
- an architecture that predicts (in a single pass) a set of objects and models their relation.

1.2.1 Object detection set prediction loss

The DETR model deduces a set of N predictions, in a single pass through the decoder. Note that N needs to be larger than the number of objects in a typical image. The difficulty comes from scoring objects with respect to their ground truth. The loss function used yields an optimal bipartite matching between predicted and ground truth objects.

Let's define some notations:

- y is the ground truth of the object (can be seen as size N padded with $\emptyset$)
- Each element y_i can be seen as (c_i, b_i) where c_i is the class of the object and $b_i \in [0,1]^4$ is the object bounding box relative to the image size
- $\hat{y} = \{\hat{y}_i\}_{i=1}^{N}$ the set of N predictions

- $\mathcal{L}_{match}(y_i, \hat{y}_{\sigma(i)})$ is a pair-wise matching cost between ground truth y_i and a prediction with index $\sigma(i)$
- The predictions with index $\sigma(i)$, we define the probability of class c_i as $\hat{p}_{\sigma(i)}(c_i)$ and the predicted box as $\hat{b}_{\sigma(i)}$

To find a bipartite matching between these two sets we search for a permutation of N elements $\sigma \in \vartheta_N$ with the lowest cost:

$$\hat{\theta} = \arg\min_{\sigma \in \vartheta_N} \sum_{i}^{N} \mathcal{L}_{match}(y_i, \hat{y}_{\sigma(i)}) \tag{1.2}$$

With these notations, we define $\mathcal{L}_{match}(y_i, \hat{y}_{\sigma(i)})$ as $-\mathbb{1}_{\{c_i \neq \emptyset\}} \hat{p}_{\sigma(i)}(c_i) + \mathbb{1}_{\{c_i \neq \emptyset\}} \mathcal{L}_{box}(b_i, \hat{b}_{\sigma(i)})$. This finding procedure plays a similar role as heuristic assignment of modern detectors with difference that it needs to find one-to-one matching for direct set prediction without duplicates. Now, the next component is to calculate the loss function, the *Hungarian Loss* for all pairs matched in the previous step. It's defined as follows :

$$\mathcal{L}_{Hungarian}(y, \hat{y}) = \sum_{i=1}^{N} \Big[-\log \hat{p}_{\sigma\hat{(i)}}(c_i) + \mathbb{1}_{\{c_i \neq \emptyset\}} \mathcal{L}_{box}\big(b_i, \hat{b}_{\hat{\sigma}}(i)\big) \Big] \tag{1.3}$$

The bounding box loss is calculated the following way :

$$\mathcal{L}_{box}\big(b_i, \hat{b}_{\hat{\sigma}}(i) = \lambda_{iou} \mathcal{L}_{iou}(b_i, \hat{b}_{\sigma(i)}) + \lambda_{L1} ||b_i - \hat{b}_{\sigma(i)}|| \tag{1.4}$$

where $\mathcal{L}_{iou}$ and $\mathcal{L}_{L1}$ are generalized IoU loss and $l1$ loss respectively, λ_{L1} and λ_{iou} are hyperparameters.

1.2.2 DETR architecture

The architecture of DETR is depicted in figure 1.3. It is composed of three elements: a CNN backbone to extract compact feature representation, an encoder-decoder transformer, and a simple Feed-Forward network (FFN) for the final predictions. In particular, the CNN backbone serves to learn the 2D representation of an input image. Then, the model flattens the resulting feature maps with their positional embeddings before passing it into a transformer encoder. The transformer decoder uses as input a fixed number of learned positional embeddings (called object queries). It finally feeds the output embedding of the decoder to a shared FFN that predicts either a class or no objects.

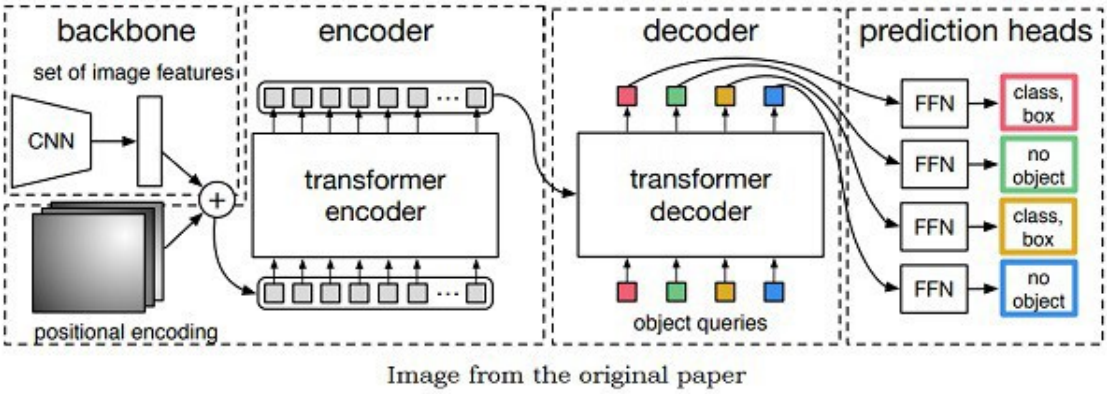


Figure 1.3: DETR global architecture

1.2.3 Training

The training was done on the COCO 2017 detection and panoptic segmentation dataset, with 118k training images and 5k validation images. On average, there is 7 objects per image and up to 63 instances in a single image, those objects annotated by bounding boxes and panoptic segmentation. The different DETR models were trained on the stuff and things COCO dataset.

1.2.4 Results

The results shows that the DETR is dominant on the stuff classes, and the hypothesis is that the global reasoning allowed by the encoder attention is the key element to this result. As for the things class, despite some deficit up to 8 mAP (mean Average Precision), the DETR obtains competitive PQth (Panoptic Quality (PQ) and the break-down on things (PQth)).

1.3 CSRNet: Dilated Convolutional Neural Networks for Understanding the Highly Congested Scenes

Crowd counting is becoming more and more relevant in recent years. Furthermore, demand for the actual density maps of people in images follows real-life applications. Those distribution maps can help get more accurate and comprehensive information for correct decision-making in a high-risk environment for instance. The authors of this paper designed an algorithm for crowd counting and generates a high-quality density map. In particular, it utilizes a CNN-based density map generator. It is first composed of the first 10 layers of the VGG-16 application [25] as the front-end and dilated convolution layers as the back-end to enlarge receptive fields and extract deeper features without losing resolutions [20]. This innovative structure is able to outperform the state-of-the-art crowd solutions created so far.

1.3.1 CSRNet architecture

As the front end of the CSRNet, the authors used the VGG-16 network because of its transfer learning properties and flexible architecture for concatenating the back-end density map generation. The output of those first layers outputs feature-maps 1/8th the size of the original input. To keep this size as the size of the outputted density maps, they deployed dilated convolutional layers as the back-end to get deeper information of saliency.

1.3.2 Dilated Convolution

The dilated convolution can be defined as follows :

$$y(m,n) = \sum_{i=1}^{M}\sum_{j=1}^{N} x(m + r \times i, n + r \times j)w(i,j) \tag{1.5}$$

where $y(m,n)$ is the output of the dilated convolution from input $x(m,n)$ and a filter $w(i,j)$ with the length M and N respectively, r is the dilated parameter, if $r = 1$ then we have normal convolution as seen in figure 1.4. Dilated convolutional layers have already demonstrated improvement of accuracy in the past, hence their utilization in this paper. It is also a good substitute for pooling layers, which tends to lose spatial information.

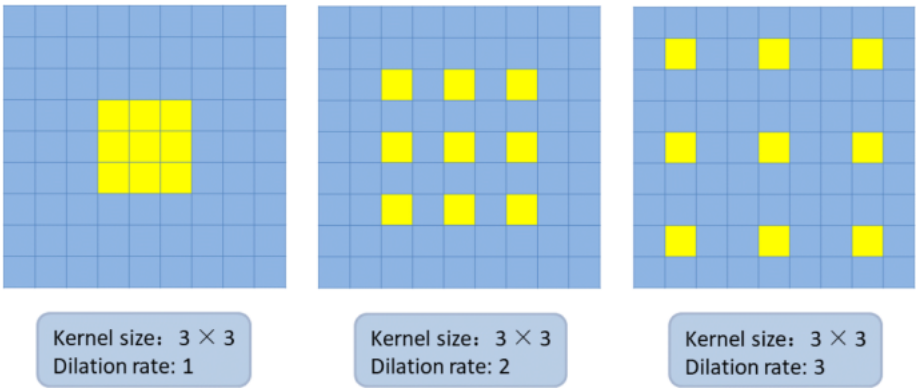


Figure 1.4: 3×3 convolution kernels with different dilation rate as 1,2,and 3

Dilated convolutional layers also present some advantages when maintaining the resolution of feature maps. For instance, rather than using the traditional convolution + pooling + deconvolution layers, a dilated convolution with a 3 x 3 Sobel kernel to a dilated kernel with a factor 2 holds more detailed information in the output rather than the former while the output is sharing the same dimension as the input.

1.3.3 Network Configuration

The network configuration is shown in figure 1.5.

Configurations of CSRNet			
A	B	C	D
input(unfixed-resolution color image)			
front-end (fine-tuned from VGG-16)			
conv3-64-1 conv3-64-1			
max-pooling			
conv3-128-1 conv3-128-1			
max-pooling			
conv3-256-1 conv3-256-1 conv3-256-1			
max-pooling			
conv3-512-1 conv3-512-1 conv3-512-1			
back-end (four different configurations)			
conv3-512-1	conv3-512-2	conv3-512-2	conv3-512-4
conv3-512-1	conv3-512-2	conv3-512-2	conv3-512-4
conv3-512-1	conv3-512-2	conv3-512-2	conv3-512-4
conv3-256-1	conv3-256-2	conv3-256-4	conv3-256-4
conv3-128-1	conv3-128-2	conv3-128-4	conv3-128-4
conv3-64-1	conv3-64-2	conv3-64-4	conv3-64-4
conv1-1-1			

Figure 1.5: CSRNet Architecture

As you can see from figure 1.5, four network configurations are proposed. They all have the same front-end structure (10 layers of VGG network with 3x3 kernels) with different dilation rates as the back-end structure. The front-end structure inherited from the VGG network is without its fully convolutional layers, hence keeping only the first 10 layers to get the best trade-off between accuracy and resource efficiency. The density maps output from the network is smaller (1/8th of the size), and a bilinear interpolation was chosen with the factor 8 for scaling.

1.3.4 Training

One important aspect of the training is density map generation. They used a geometry-adaptive kernels to tackle highly crowded scenes. Each head annotation is blurred using a Gaussian kernel (normalized to 1), thus generating our ground truth images. The geometry adaptive kernel is defined as :

$$F(x) = \sum_{i=1}^{N} \delta(x - x_i) \times G_{\sigma_i}(x) \tag{1.6}$$

with $\sigma_i = \beta \overline{d_i}$

For each object x_i in the ground truth δ, we use $\overline{d_i}$ to indicate the average distance of the first k neighbors. Then the ground truth $\delta(x - x_i)$ is convolved with a gaussian kernel with the parameter σ_i, where x is the position of the pixels in an image.
In order to help the training, some data augmentation can be done. For instance, they cropped each images into 9 patches with 1/4th of the size of the initial image. The patches can also be mirrored to double the size of the datasets.

Lastly, the loss function used for training is an Euclidean distance to measure the difference between the ground truth and the estimated density map of the output (which is equal to the mean absolute error). The loss function is defined as followed :

$$L(\Theta) = \frac{1}{2N} \sum_{i=1}^{N} ||Z(X_i, \Theta) - Z_i^{GT}||_2^2 \tag{1.7}$$

where N is the batch size, $Z(X_i, \Theta)$ is the output generated with the parameters Θ and X_i is the input image, Z_i^{GT} is the ground truth result of the image.

1.3.5 Results

The CSRNet model achieved state-of-the-art performance on multiple crowd-counting datasets. The dilated convolutional layers expand the receptive field of the neurons without losing resolution. They also expanded their model to vehicle counting where they reached the best accuracy overall compared to state-of-the-art techniques.

Chapter 2

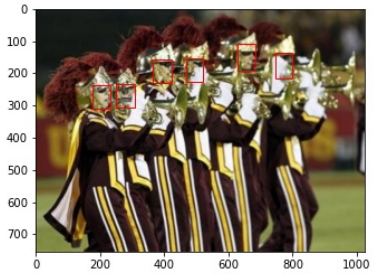
Material

2.1 Data-sets used

For face detection, some data sets are pretty common when building face detection algorithms. The ones used in this study are the following : **WIDER FACE: A Face Detection Benchmark** [35]for training/testing and **FDDB: Face Detection Data Set and Benchmark** [17]for testing.
As for Crowd counting, we will refer to the WIDER-face dataset and with the ShanghaiTech Crowd Counting data set for training and testing in the task of crowd counting.

2.1.1 WIDER face

The WIDER face data set is a face detection benchmark data set, where the images present in it are from the publicly available WIDER data set. It has 32 203 images with 393 703 faces with a high degree of variability in scale, pose and occlusion as depicted in the example images in Figure 2.1. It is organized on event classes, 61 to be exact (e.g. Parade, Handshaking, etc). For each event class, the data was randomly selected to build the training, validation, and testing subsets with a 40/10/50 split respectively. The bounding boxes of the testing images are not released to the public. In particular, we will use the WIDER data set for training, validation, and testing.



(a) Example of Parade event image from the WIDER data-set

(b) Example of Handshaking event image from the WIDER data-set

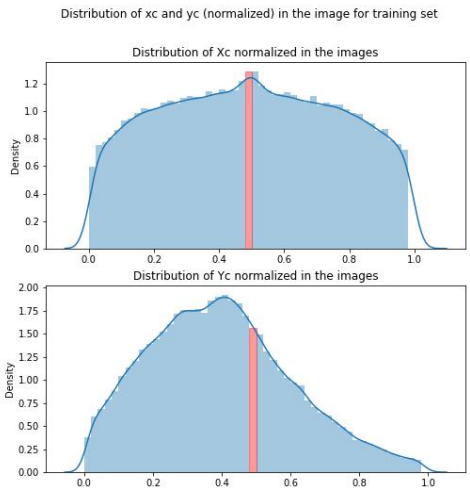
Figure 2.1: Example of images in different event class in the WIDER data-set, the red rectangles representing the annotations of faces

Let's extract some of the statistics about our WIDER data set, those are presented in table 2.1 below.

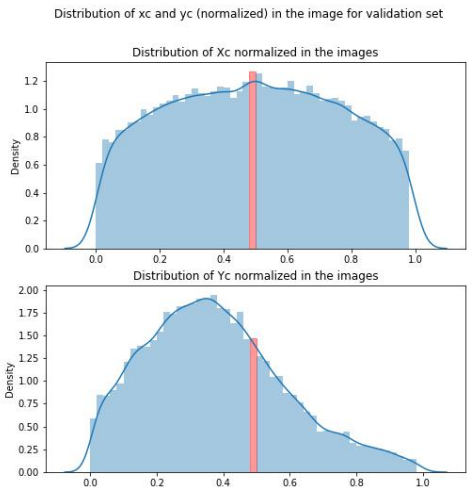
	Training set	Validation set
Number of images	12880	3226
Number of faces	159420	39708
Min nbr of faces	0	1
Max nbr of faces	1968	709
Mean nbr of faces	12.38	12.31
σ of nbr of faces	45.17	39.82
Mean size of faces (pixels)	3851.33	3834.88
σ of size of faces (pixels)	23078.15	23451.13

Table 2.1: Statistics of WIDER data-set

We can see from the σ values in table 2.1 that we have some considerable variations in the number of faces in the images and the size of those faces. It can be interesting to examine the distribution of the coordinates of bounding boxes in the images. To do so, we will take the center of the bounding boxes normalized by the size of the images respectively.



(a) Distribution of center coordinates of bounding boxes in images for training set



(b) Distribution of center coordinates of bounding boxes in images for validation set

Figure 2.2: Distribution of center coordinates of bounding boxes in images

As we can see from figure 2.2, the faces are distributed pretty evenly on the x axis while there are fewer on the y axis. Most faces are placed above the middle part of the image, it could be a potential bias of our models in the future that may need testing.

The DETR model uses object queries to detect objects in images. This number of queries delimits the number of objects that can be detected in an image, and this

parameter is set as 100 by default [12]. The number of queries can be changed, but it was kept at 100, thus limiting the maximum number of faces we can predict in an image. Hence, the images with more than 100 faces in the image will be excluded. There are 242 of those images in the training set, while there are 59 in the validation set. That leaves us with 98.2% of images left in both data sets.

2.1.2 FDDB

Similarly, the Face Detection Data-set and Benchmark (FDDB) is a data-set of face regions designed to study the problem of unconstrained face detection. It contains 5171 faces in 2845 images taken from the Faces in the Wild data set. It contains colored and grey scaled images.

(a) First example

(b) Second example

Figure 2.3: Examples of images found in the FDDB data-set, the red ellipses representing the annotations of the heads

We can see that the faces here are annotated by elliptic regions rather than rectangle ones. Like with the WIDER face data set, we can extract some statistics from the FDDB data set. Those are summarizes below in table 2.2.

	FDDB data-set
Number of images	2845
Number of faces	5171
Min nbr of faces	1
Max nbr of faces	27
Mean nbr of faces	1.82
σ of nbr of faces	1.58
Mean size of faces (pixels)	13417.77
σ of size of faces (pixels)	15011.42

Table 2.2: Statistics of the FDDB data-set

Contrary to the WIDER data set, this data set presents fewer faces in the images. The images are mainly composed of only a couple of faces per image. With that, the

faces present are bigger than their average size. Lastly, let's look at the distribution of the center of the ellipse in the images. Figure 2.4 shows just that.

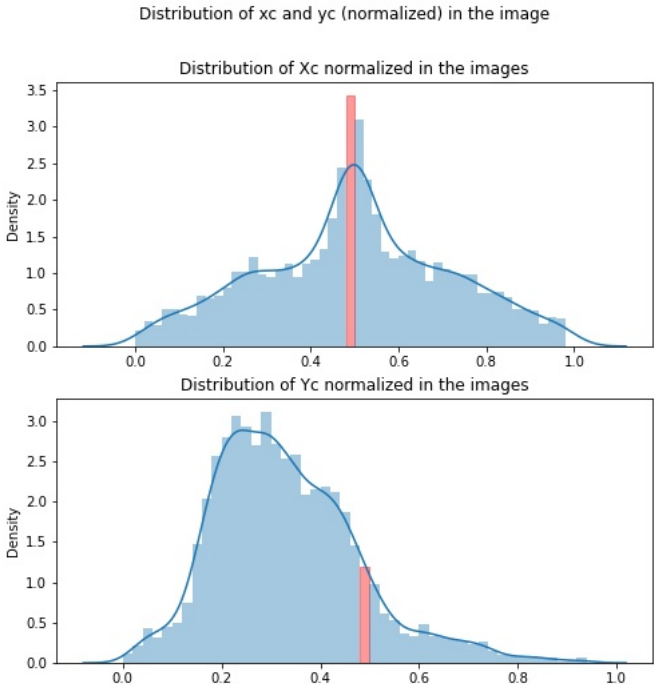


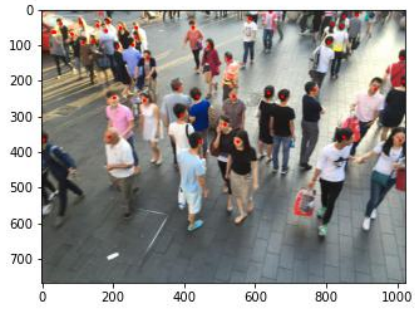
Figure 2.4: Distribution of center coordinates of bounding boxes in images

We can see that we have more of a concentration of faces at the center of the images on the x axis while, similarly as for the WIDER sets, the y axis present faces more on the top of the pictures as nicely illustrated by figure 2.3. It makes sense that the faces are more concentrated in the center for this data set since we have way more images with only a single face in them, hence a centered portrait shot.

2.1.3 ShanghaiTech Crowd Counting

The ShanghaiTech dataset is composed of 1198 annotated images of crowds. It is divided into part A and part B which there are respectively composed of 482 and 716 images. Part A is divided into a 300/182 images training/testing split and where collected on the internet while part B is divided into a 400/326 images training/testing split and where collected in the streets of Shanghai. Each person in the crowd images is annotated with a single point close to the center of the head, where we have in total 330,165 annotated faces.

(a) Example of images in the part A of the ShanghaiTech dataset, heads annotated by a red dot



(b) Example of images in the part B of the ShanghaiTech dataset, heads annotated with a red dot

Figure 2.5: Exemple of images in the ShanghaiTech dataset

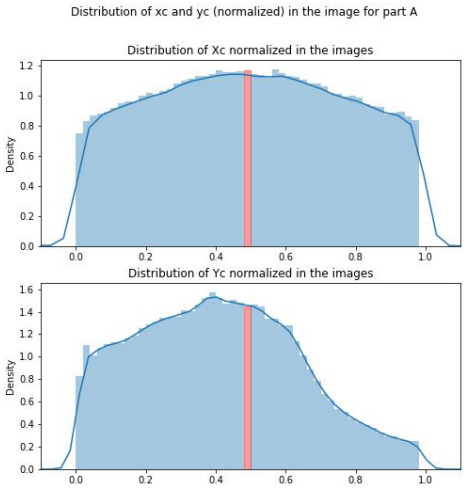
Here are some additional statistics of the ShanghaiTech dataset:

	part A	part B
Number of images	482	726
Number of faces	241677	88488
Min nbr of faces	33	9
Max nbr of faces	3139	578
Mean nbr of faces	501.40	123.59
σ of nbr of faces	456.49	94.46

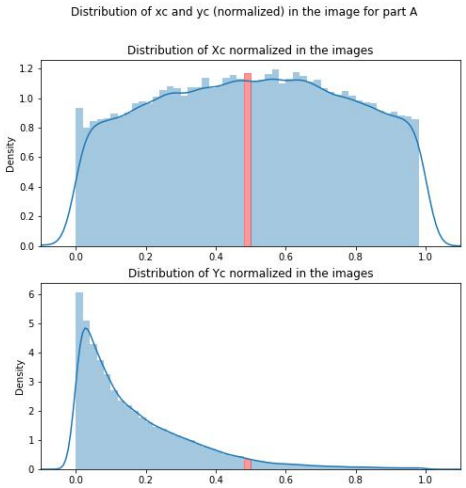
Table 2.3: Statistics of ShanghaiTech data-set

The head positions in the images have the following distributions (see figure 2.6). We can see that the head placements are pretty balanced on the x axis, while they are not on the y axis, especially for the part B dataset.

We've already covered how the density maps are generated in sub-section 1.3.4, now let's look at some examples of those density maps. Figure 2.7 shows one instance of an image with its density map in both parts of the data set. The distribution of faces is more difficult to distinguish from sub-figure 2.7a than from 2.7b since the crowd is considerably larger. We can still see some details of a concentration of faces in the back of the image. Looking at the other sub-figure 2.7b, we can easily make the distinction between the faces in the image and the gaussian-activated pixels in the density maps.

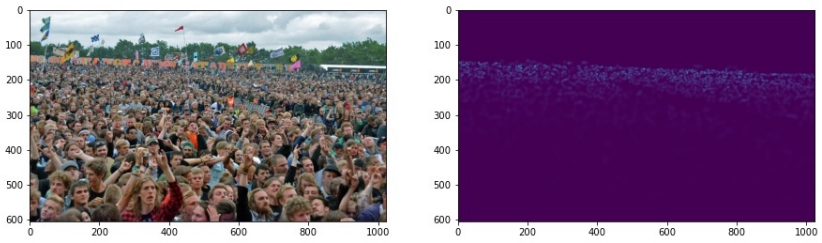


(a) Distribution of center coordinates of head positions in images for part A

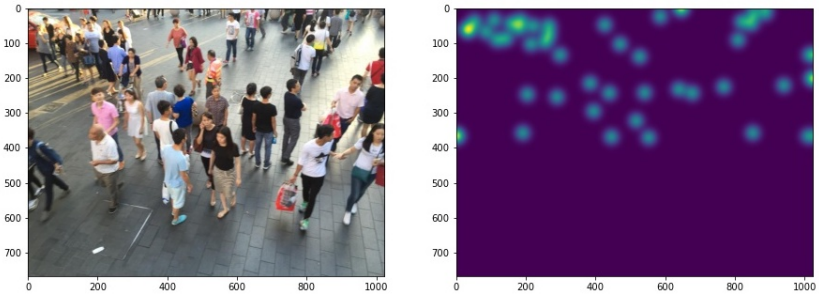


(b) Distribution of the coordinates of head positions in images in part B set

Figure 2.6: Distribution of head positions in images in the ShanghaiTech dataset



(a) Example of density maps with it's corresponding image in the ShanghaiTech dataset part A



(b) Example of density maps with it's corresponding image in the ShanghaiTech dataset part B

Figure 2.7: Example of density maps with their corresponding image in the ShanghaiTech dataset

2.2 Hardware

Most of the work was done on a Desktop workstation having the following specificities :

- **OS**: Windows 10 Family
- **Type**: x64
- **CPU**: AMD Ryzen 5 1500X Quad-Core Processor, 3500Hz
- **RAM**: 16 GB
- **GPU**: NVIDIA GeForce GTX 1060 6GB

Chapter 3

Methodology

This chapter will present the overall data pipeline in the different tests orchestrated, with the different methods studied. The programming language used is *Python*, using diverse GUIs (Sublime Text) but mostly Jupyter Notebook, a web-based interactive computing platform, helping with visualization. To build and train the different models discussed in the following, two different APIs were used depending on the ease of use for a different context. In particular, those are Tensorflow and Pytorch.

3.1 Overall Pipeline

DETR applied to face detection

Following the tutorials listed in the DETR-TensorFlow GitHub repository, it is possible to load a custom dataset to fine-tune the DETR model to detect some custom objects [3], faces in our case. To do so we will define a custom configuration to the DETR model containing the data directories, the size of the resized images, etc. We will then create and load the weights of the pre-trained DETR model but with only 2 classes as output (rather than the 92 initially present), those being the background and the faces classes. The DETR model being heavy in its number of parameters, it is required to freeze the weights of the backbone. Some small adjustments need to be brought to the annotations to use the DETR model. For instance, it is required to pass the initial size of the image to the model, information that is not initially available in the annotations files of the WIDER-face data set. The training set and validation set were loaded separately to and we can launch the training.

After getting the predictions, a Non-Max Suppression is done to suppress the boxes that are overlapping until a certain threshold (0.5 in our case). This Non-Max Suppression was implemented since, in certain cases, the predictions seem to overlap a lot.

Crowd Counting using Vision Transformers

Multiple attempts were done for crowd counting using Vision transformers, not all of them will be detailed, only the ones with the most promise. The first one is a "naive" regression task on the WIDER data set. The second one is using the attention maps of

the transformers as a substitution for the feature maps that we can find in convolutional deep learning models. Hence, the ViT is the backbone of the model, feeding the attention maps to a deconvolutional block. In both cases, the images are loaded and fed to the different models. Their labels and ground truth maps are also fed to the models, the creation of those is detailed in the next section. Note that, for the regression model, the labels are normalized to 1. To ease the training, the same images given to the DETR model will be used for the regression model, hence having a maximum number of people of 99.

For the regression task, no post-processing is required since a number is given, which we will multiply by 99 to get the actual number of people predicted in the image. As for the density maps generator, the number of people in the crowd is extracted by summing the pixel values of the density map generated. Since 2 parts data-sets are available concerning the density map generator, the second model (density map generator) will be tested on both parts separately.

3.2 Pre-processing

DETR applied to face detection

The pre-processing used by the DETR method is already implemented in the reference repository. It normalizes the input as well as resizes the input images. The model resizes the input such that the smallest side is at most a certain length, while the bigger side doesn't pass a certain size. This resizing causes images to have different sizes in a batch. To solve this issue, the DETR model will pad the images up to the largest size in this batch and creates a mask indicating where are the original pixels and where are the padded ones [12].

As previously explained in section 2.1.1, the number of object queries is initially kept at 100, thus limiting the number of predictions in an image. Hence, some images in the training and validation set were excluded, those with more than 100 faces in them. This leaves us with a total of $98,2\%$ for both data sets. Furthermore, some data augmentation is available for us to use but was not explored. Due to the size of the DETR model, training time is already important for only a portion of the total data set available. We will detail more about the number of images and faces used for training in the section 3.4.

Crowd counting using Vision Transformers

To do the regression task, the images are loaded as for the DETR architecture, but rather than looking at the position of the faces, only the number of faces was kept. Since the Vision Transformer architecture is already fit for a regression task, we will only normalize the output between 0 and 1. Since the same WIDER data-set portion is used for the DETR model, we only have a maximum of 99 faces in an image. Before feeding the images to the regression model, the images were resized to (512,512) to limit the size of the model.

As for the density map generation model using ViT as a "backbone", the original density maps with the images needed some modification (see section 1.3.4, for density maps generation). It was chosen to take normalized fixed-sized images, like the regression model, the size being (512,512). Hence, the ground truth density maps needed to be altered also, since they represent the 2D distribution of faces in the image. To create those new density maps, we will normalize the positions of the faces by this operation :

$$\text{new\_pos}_i = \frac{\text{pos}_i}{\text{size}_i} * 512 \tag{3.1}$$

where new_pos_i is the new coordinate, pos_i is the original position, size_i is the size of the image and $i = x, y$. Once the new positions are calculated, the same operations to create the density maps are applied, resulting in a density map the same size as the input image.
Furthermore, right before training, the ground truth density maps are resized to 1/8th of the size of the image to fit the output size of the first 10 layers of the VGG-16 model as the backbone of the CSRNet model. Hence, we also apply this transformation to our ground truth maps, resulting in density maps of size (64,64).

3.3 Network architecture

DETR for face detection

The network architecture for the DETR follows the same as in section 1.2.2 until the transformer output. To fit with the new custom data that we will feed the model, a couple of "layers" are added where both take as input the output of the transformer. The first one is a *Sequential* layer consisting of 3 Dense layers having 256 as the number of units except for the last one which has only 4. This Sequential layer will help with the predictions of the bounding boxes for each query (4 outputs for $[x, y, w, h]$). The other layer added to the DETR network is also a Dense layer, this one having only 2 units, helping with the classification of each object query (in our case, either background or face). In total, the network is composed of 41,449,152 parameters with a total of 17 496 582 learnable parameters.

Crowd counting using Vision Transformers

Let's examine the network architecture of the model used for regression using ViTs. As said before, it was a naive try to apply vision transformers to the task of regression. The ViT from the vit-pytorch library is fit for a classification task out of the box. Hence, we can choose the number of classes as output to be 1, thus transforming the classification into regression. The last Dense layer of the network architecture will have one output. It was chosen to have a depth of 6 while each encoding layer has 8 heads. The dimension of the first multi-layer perceptron was chosen as 128 to limit the number of parameters since we do not have that much training images. Lastly, the last Dense layer has a size of 1 to have a regression task, as said before. In the end, the model has 1811457 parameters. The global architecture is illustrated in figure 3.1.

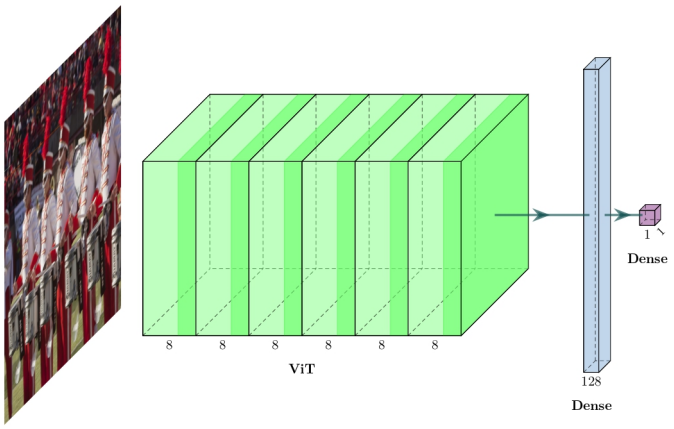


Figure 3.1: Illustration of the model used for regression using ViT

The second model using ViT for crowd counting will output density maps. To do so, we will use the Vision Transformer as the backbone and then a deconvolutional block. Indeed, Vision transformers, being composed of a few encoding blocks, where every block has a few attention heads (i.e. multi-head attention is multiple attention layers in parallel), that are responsible, for every patch representation, for fusing information from other patches in the image [14]. Every attention head are composed of [21] :

- Q: Vector(Linear layer output) related with what we encode(output, it can be output of encoder layer or decoder layer)
- K: Vector(Linear layer output) related with what we use as input to output.
- V: Learned vector(Linear layer output) as a result of calculations, related with input

The attention function is essentially mapping that Q, K, and V vectors to an output. The output is computed as a weighted sum of the values. The multi-attention head is represented in figure 3.2.

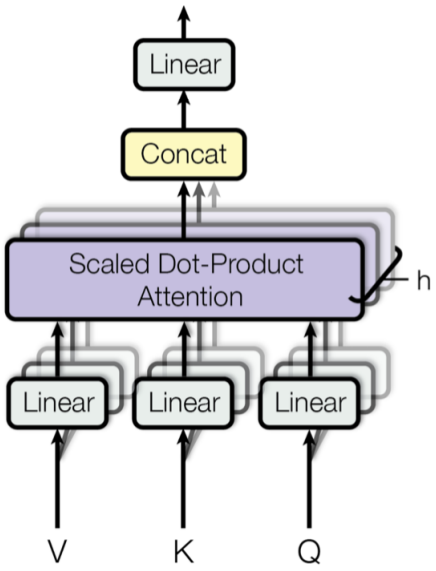


Figure 3.2: Multi-Head attention

The attention is called "Scaled Dot-Product Attention" and is calculated the following way :

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{3.2}$$

where d_k is the dimension of the queries Q and the keys K. It was found beneficial to linearly project Q, K and V h times with different learned linear projections to d_k, d_k and d_v dimensions respectively. The advantage of Multi-head attention is that it allows to attend information from different representation subspaces at different positions. The Mutli-head attention is calculated the following way :

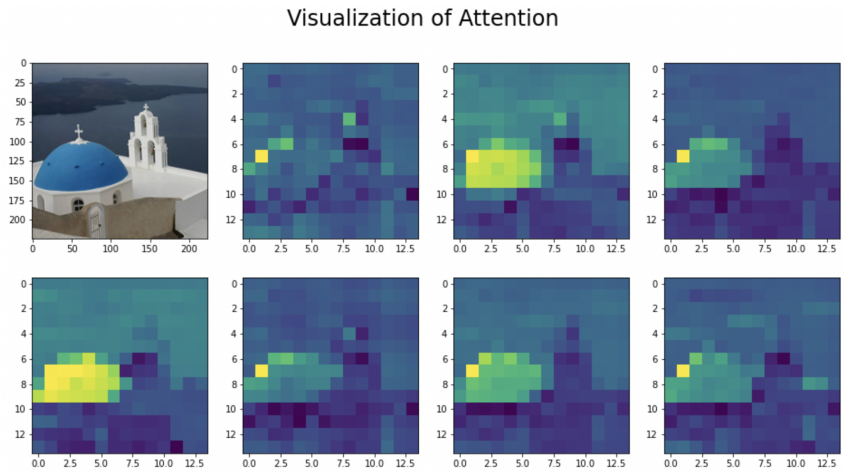
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O \tag{3.3}$$

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_I^K, VW_i^V)$ and the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$ [29]. It is those attention maps that we will feed the deconvolution block in order to generate our density maps. The idea came about when exploring the visualization possible for Vision Transformers while searching examples implementation of DETR. We can see from figure 3.3 that we can expect a visual representation of objects in the attention maps. This visual representation was the drive behind feeding these attention maps to some convolutional block (deconvolution in our case).
In fact, the attention maps shown are the visualization of the 100th rows of attention matrices in the 0-6th heads for subfigure 3.3a. The original figure can be found in this Google Colab link. Similarly, the original figure for sub-figure 3.3b can be found with this Google Colab link.

It is required to reshape the output of the multi-head attention since we have an output of dimension (batch size, depth, num_heads, PxP+1, PxP+1) where P is the

patch size and the +1 is the position embeddings. Hence we will reshape this output by taking the last layer, 3 of the 6 heads, and ignoring the position embedding of the last dimension. Including more heads would have increased the number of parameters and the memory usage too much. Hence we get the following dimension (batch_size, PxP+1, P, P). More precisely, the parameters of the transformer were :

- image_size = 512
- patch_size = 32
- num_classes = 1
- dim = 128
- depth = 3
- heads = 8
- mlp_dim = 128



(a) Example of attention maps visualization for classification method

(b) Example of attention maps visualization for detection method

Figure 3.3: Example of attention visualization

These hyper-parameters leave us with attention maps of shape (batch_size, 257*3, 16, 16). As precise before, these attention outputs will be fed into a deconvolution block to return to the expected size of the ground truth images, those being (64,64).

This deconvolutional block is composed of 6 layers, the configuration was chosen to retrieve the right shape is the following :

- Deconv1 : out_channels=32, kernel_size=9,stride=1
- Deconv2 : out_channels=32, kernel_size=6,stride=2
- Deconv3 : out_channels=32, kernel_size=5,stride=1
- Deconv4 : out_channels=32, kernel_size=5,stride=1
- Deconv5 : out_channels=32, kernel_size=5,stride=1
- Deconv6 : out_channels=1, kernel_size=1,stride=1

The output of the entire model is an image of dimensions (64,64). The model contains 3231400 parameters. The entire model is represented in figure 3.4.

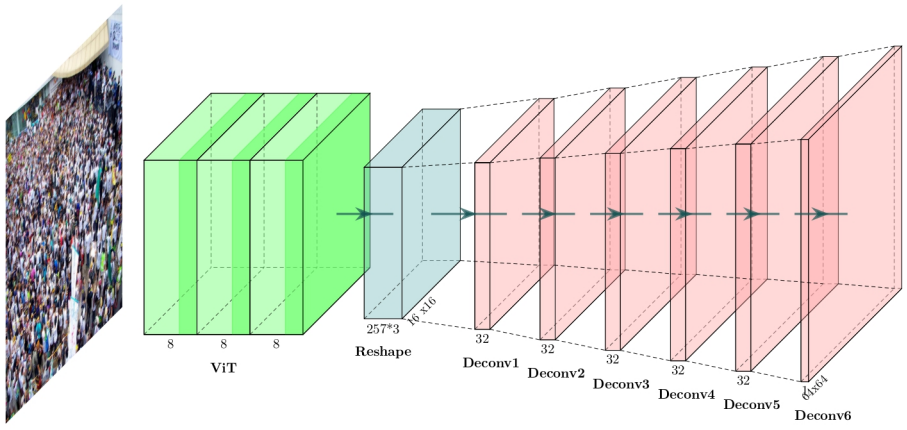


Figure 3.4: Illustration of the model used for density maps generation using ViT

3.4 Training details

DETR for face detection

For training, not much was changed from the training configuration that the DETR team chose. The loss function used is the set prediction loss detailed in section 1.2.1. The learning rate for the backbone, the transformers, and the output layers are $10^{-5}, 10^{-4}$ and 10^{-4} respectively. To run the training on the personal computer available, it was required to reduce the batch size to 1 for memory availability reasons.

Since the time of training is lengthy on the available computer, it was chosen to only take part of the available training and validation set when learning. Using only one-third

of the total training set available, the training time was already over 48 hours for 100 epochs. That leaves us with 4135 images for the training set and 1038 for the validation set, with a total of 33723 and 8358 faces present in the training set and validation set respectively. On the complete data set, training time would have taken roughly 200 hours, with the material available at the time.

Crowd counting using Vision Transformers

For the regression task, the *Mean Squared Error* was chosen for the loss function with the *Mean Absolute Error* as the metric. The Mean Absolute Error is calculated the following way :

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|C_i - C_i^{GT}| \tag{3.4}$$

where N is the size of the batch, C_i^{GT} is the ground truth count of the number of people in the image. The Mean Squared error is defined as :

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(C_i - C_i^{GT})^2 \tag{3.5}$$

The Adam optimizer algorithm was used for the training as it benefits from the advantages of the Adaptive Gradient and the Root Mean Square Propagation algorithm and is well suited for computer vision applications. In particular, some advantages are the following [18]:

- Straightforward to implement.
- Computationally efficient.
- Little memory requirements.
- Invariant to diagonal rescale of the gradients.
- Well suited for problems that are large in terms of data and/or parameters.
- Appropriate for non-stationary objectives.
- Appropriate for problems with very noisy/or sparse gradients.
- Hyper-parameters have intuitive interpretation and typically require little tuning.

The training was ran on 50 epochs with a learning rate of $3 * 10^{-5}$.

For the density map generator, we will follow the training details of the CSRNet model [20]. Hence, the loss function is the same as 1.3.4. Here also we used the Adam optimizer for the same reasons as for the regression problem. Lastly, we ran the training for 100 epochs with also a learning rate of $3 * 10^{-5}$.

3.5 Unfruitful methods

As said earlier, all the methods tried were not presented in this study. Initially, for face detection, the development of a Region Proposal Network was attempted but with no luck. Figure 3.5 shows an example of the output generated from this attempt. The red boxes are the 20 boxes where the model has the most confidence in the presence of a face. As we can see, the model fails to recognize the faces accurately.

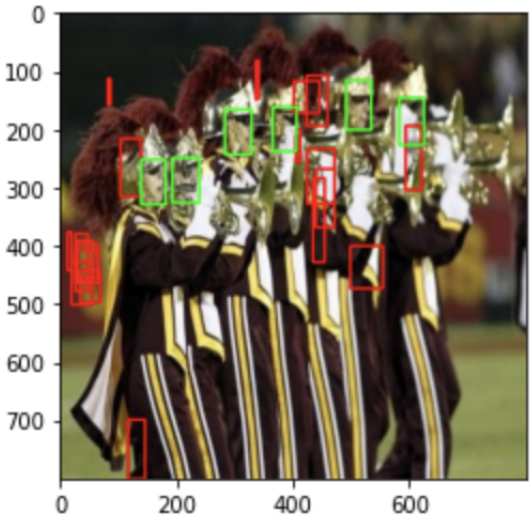


Figure 3.5: Example of output from RPN

An attempt of combining ViT with the FaceBoxes method was also performed but with no luck. While it was a naive attempt to replace the feature maps given by a convolutional network with attention maps yielded by a ViT, it did not converge while training. It also did not yield any predictions when testing this model on newly presented images.

Then, the idea to use an object detector using ViTs and apply it to face detection came. Learning to detect faces from scratch was a lost cause since the data set was far from large enough for the number of parameters present in the model ($\approx$ 41 million parameters), the model would not learn anything. It would be pointless to show you an example of the output of this training instance since there were simply no predictions no matter the input.

Lastly, the crowd counting part of this study came about. From the preceding models presented, multiple iterations of those were tested, and either they would not present convergence in the training loss functions, they would diverge at some point, or present very similar results as for the ones presented. One instance tried to use the transfer learning ability of the VGG16 model by using it as the backbone before passing the features maps into the ViT [13]. This instance didn't present any meaningful results since the training presented no convergence in the loss function of his metrics. Another attempt tried to add convolutional layers after the deconvolutional block of our density map generator. The same convolutional block was used for the CSRNet model [20]. But, once again, no meaningful convergence was present during training.

Chapter 4

Evaluation

This section will detail the methods of evaluation of each algorithm discussed so far. These algorithms of machine learning will have different metrics, and a good metric is essential to extract meaningful information from the predictions.

4.1 Face Detection

4.1.1 Average Precision

The WIDER-face evaluation metrics are the same as in the PASCAL VOC challenge [11]. In particular, the *Precision Recall* curves and the *Average Precision* will be used to determine the performance of our model.

Let's first define the *Intersection over Union* (IoU), since it will help us understand other variables in the future. The IoU is given by the overlapping area of the ground truth bounding box and the predicted box divided by the union of both areas. An illustration of such a metric is given in figure 4.1. The mathematical expression is as follows :

$$\text{IoU} = \frac{\text{area}(B_p \cap B_{gt})}{B_p \cup B_{gt}} \tag{4.1}$$

In order to calculate the Average Precision, we will need to define the Precision and Recall :

$$\text{Precision} = \frac{TP}{TP + FP} \tag{4.2}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{4.3}$$

where TP corresponds to a correct detection (i.e. a detection with a IOU $\geq$ threshold, True Positive), FP corresponds to a wrong detection (i.e. a detection with a IOU $<$ threshold, False Positive) and FN a ground truth not detected (False Negative) [10]. The Precision-Recall curve usually is composed of multiple curves for object detection. In our case, only one curve will be present since the only object detected is the "faces". A detector is considered good as the Precision stays high as the Recall increases. It shows

the tradeoff between precision and recall at different thresholds. A high area under the curve corresponds to a high value of recall for a high value of precision, where a high precision translates to a low FP rate (see equation 4.2). Hence, a high area under the curve means accurate results (high precision) where the majority of positive results are returned (high recall).

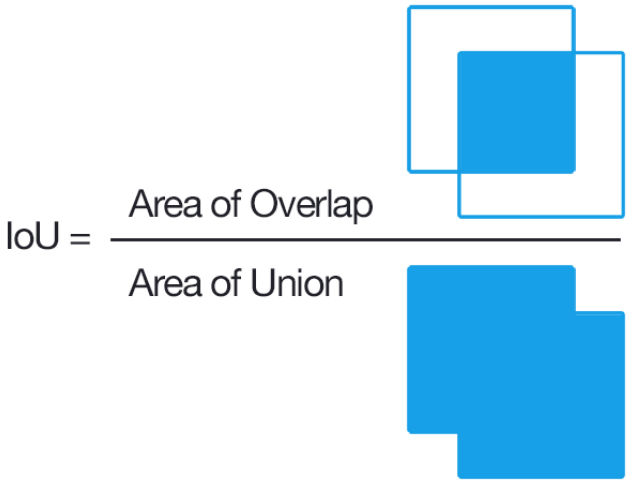


Figure 4.1: IoU calculation example

The Average Precision summarizes the relationship between precision and recall. It is calculated by measuring the area under the curve of the previously discussed Precision-Recall curve. It is the weighted mean of precisions achieved at each threshold, where the increase of recall from the previous threshold is used as weight. We can calculate it the following way [24]:

$$\text{AP} = \sum_n (R_n - R_{n-1}) P_n \tag{4.4}$$

where P_n and R_n are the precision and the recall at the nth threshold.

4.1.2 Receiver Operating Characteristics

The evaluation of the FDDB data set is done by plotting the *Receiver Operating Characteristic* curve (ROC). It is a graph that shows the performance of our classification model at different thresholds. ROC is a probability curve and the area under the curve (AUC) represents the degree or measure of separability. In fact, it plots 2 parameters: True Positive Rate (TPR) and False Positive Rate (FPR) defined respectively as

$$TPR = \frac{TP}{TP + FN} \tag{4.5}$$

$$FPR = \frac{FP}{FP + TN} \tag{4.6}$$

where TP corresponds to a correct detection (True positive), FP corresponds to a wrong detection (False Positive), TN detection correctly determined as background (True Negative), and FN a ground truth not detected (False Negative) [8]. The better the model, the better the AUC. Hence, an excellent model has an AUC close to 1. The more the curve tends to identity line, the less the model differentiates the classes [22]. To plot the curves, we compare the TPR and the FPR at different classification thresholds. By lowering the threshold, more objects are classified as positives, thus increasing FP and TP. In our case, the TN variable isn't an output of the model since the face detection models will only identify what the model deems as positive areas, hence the ROC curve will be constructed with the FP as the abscissa.

4.2 Crowd Counting

4.2.1 Regression model evaluation

To evaluate the regression method studied we will use the same metrics as in the CSRNet paper [20], those being the MAE and MSE defined at 3.4 and 3.5 respectively. We will add also the *Root Mean Square error* define as follows :

$$\text{RMSE} = \sqrt{MSE} \tag{4.7}$$

To that, we will add the *Coefficient of determination* (R^2) to the metrics of evaluation. It measures the variability in the dependent variable that the model can explain. It does a good job determining the fitting of dependent variables but does not take into account the overfitting problem. The R^2 is defined as follows [31]:

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2} \tag{4.8}$$

4.2.2 Density map generator evaluation

The evaluation of the density map generator will base itself on the evaluation of the CSRNet model [20]. The evaluation metrics are the MAE and the RMSE defined at equation 3.4 and 4.2.1 respectively. Furthermore, the *Structural similarity index* (SSIM) and the *Peak Signal to Noise Ratio* (PSNR) will be also examined. Those will serve to judge the quality of the generated density maps.

Peak signal-to-noise ratio (PSNR) is the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation [32]. The PSNR is defined as follows :

$$\text{PSNR} = 20 \log_{10} \left(\frac{L-1}{\text{RMSE}}\right) \tag{4.9}$$

where L is the maximum value of the pixels (256 in our case), and RMSE is the root Mean Square error defined earlier at 4.2.1. The CSRNet paper [20] utilizes a specific pre-processing in order to resize with interpolation and normalization the ground truth

images and the generated density maps based on this article [26]. This specific transformation was not found in the reference paper, so another transformation was used to calculate the PSNR values. It was chosen that the ground truth maps are resized to the size of the generated ones. There are then both normalized to 256 before performing the calculations.

The Structural Similarity Index (SSIM) is a metric that quantifies the image degradation quality caused by some processing like data compression or loss by data transmission. It compares an original image with its degraded one. Hence, we can use this index to compare the quality of the generated density maps. It is defined as follows [33]:

$$\text{SSIM}(x, y) = [l(x, y)]^{\alpha} \times [c(x, y)]^{\beta} \times [s(x, y)]^{\gamma} \tag{4.10}$$

where

$$l(x, y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \tag{4.11}$$

$$c(x, y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \tag{4.12}$$

$$s(x, y) = \frac{\text{cov}_{xy} + C_3}{\sigma_x\sigma_y + C_3} \tag{4.13}$$

$$\tag{4.14}$$

The variables are :

- μ_i is the mean of $i = x, y$
- σ_i is the variance of $i = x, y$
- cov_{xy} is the covariance of x and y
- $C_1 = (k_1 L)^2$, $C_2 = (k_2 L)^2$ and $C_3 = \frac{C_2}{2}$ and L is the maximum value of pixels.
- $k_1 = 0.01$ and $k_2 = 0.03$

As for the PSNR evaluation method, we will resize the ground truth to the size of the generated density maps.

Chapter 5

Results

This section will present the results generated following the evaluation metrics given is chapter 4. Once again, this chapter is divided into sections that will present the results by the methods used, starting with Face Detection.

5.1 Face Detection

5.2 Training history

Figure 5.1 shows the training history of the different metrics of the DETR model, those being the *Classification Error* (CE), the *Generalized IoU* (GIoU) and the *L1 score*. The GIoU is defined the following way :

$$\text{GIoU} = \text{IoU} - \frac{|C \setminus (B_p \cap B_{gt})|}{|C|} \tag{5.1}$$

where IoU is defined at equation 4.1 and C is the smallest convex hull that encloses B_p and B_{gt} [28]. The L1 loss is calculated by taking the absolute difference between each coordinates of the predicted boxes and the ground truth ones, summing this difference for all coordinates, then averaging it for the 4 coordinates. It can be represented by this equation :

$$L1_{\text{loss}} = \frac{1}{4} \sum_{i=1}^{N} \sum_{j=1}^{4} |y_{ij} - \hat{y}_{ij}| \tag{5.2}$$

where y_{ij} is the predicted bounding box, $\hat{y}_{ij}$ is the ground truth bounding box and N is the batch number. Finally, the CE represent the sparse softmax cross entropy between the target class and the predicted class.

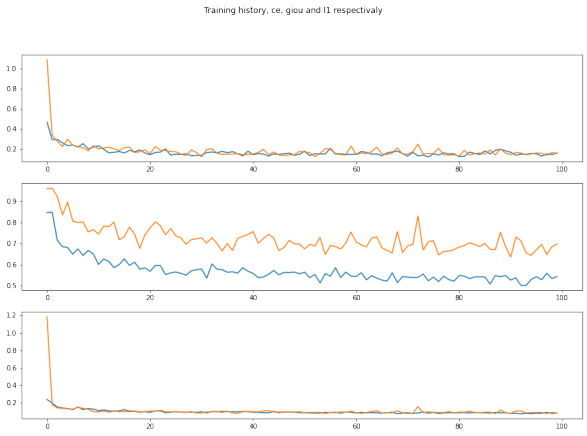


Figure 5.1: Training history of CE, GIOU and L1 respectively (Blue = training, Orange = validation)

We see a global convergence for the training set as well as the validation set, implying some learning over the detection of faces. The convergence of the CE means that the boxes proposed are correctly predicted with the right class. The GIoU metric convergence implies that the predicted bounding boxes approaches more and more the target boxes, although it seems that it is the more complicated of the tasks. Lastly, the L_1 loss is also converging, implying that the difference between the coordinates of the predicted boxes approaches the ones of the target boxes.

5.2.1 FDDB data-set results

Before presenting the results of the evaluation, let's look at some intuitive results by showing the predictions on some of the testing images of the FDDB data set (see figure 5.2). We can observe that the predictions seem to be more or less accurate. There is a false positive on the top left example image, and some of the background faces are less accurate than the foreground ones. Nonetheless, the model seems to reasonably recognize faces on the FDDB data set. The next step is to look at the ROC curves. Figure 5.3 compares the ROC curves for the DETR, the FaceBoxes, and the MTCNN methods.

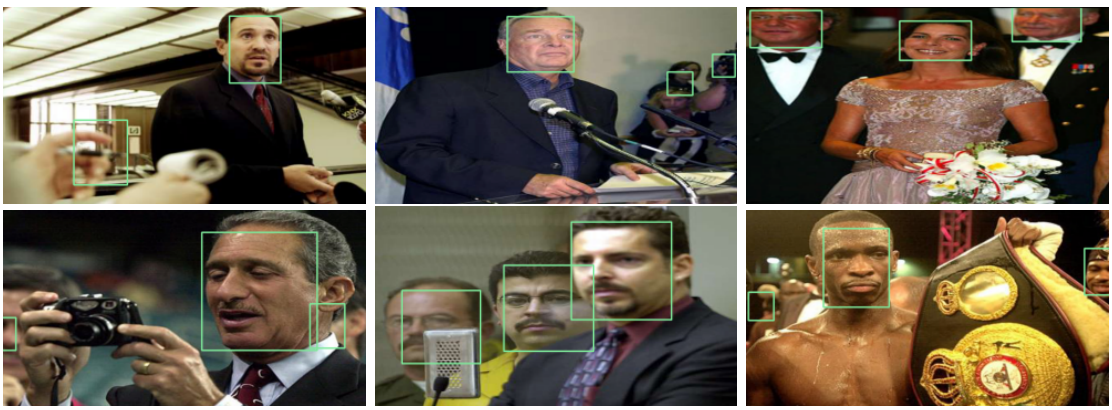
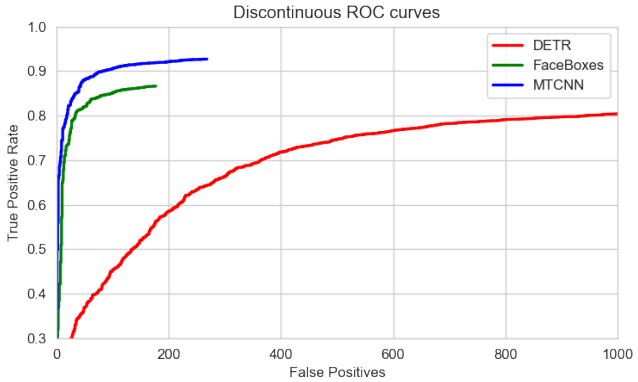
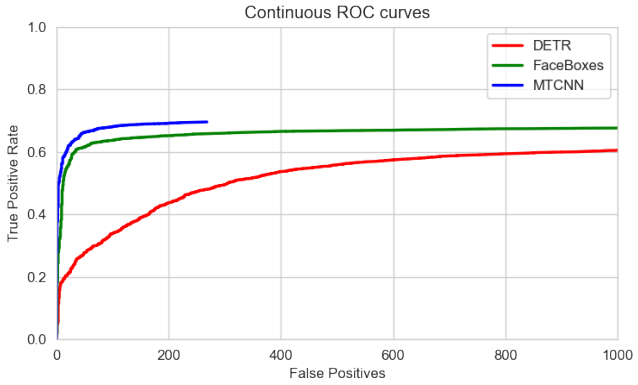


Figure 5.2: DETR predictions on some testing images of the FDDB data-set



(a) Discontinuous ROC curve



(b) Continuous ROC curve

Figure 5.3: ROC curves

Our model outputs rectangular boxes, like FaceBoxes and MTCNN, contrary to the elliptical annotations of the FDDB data set. This inconsistency has an impact on the continuous ROC curve. Nevertheless, since all three methods have the same rectangular output, it is still possible to compare them. These ROC curves show us that the DETR model applied to face detection is worse than the other methods presented here. The number of false positives exceeds what the other methods output while presenting a lesser TPR, i.e. being his its predictions being less accurate compared to the ground truth than the other methods. This can be explained by the fact that the DETR model seems to be missing more faces than the other methods. Furthermore, since the FP is greater than the other methods, we can conclude that it also tends to mistake part of the background for faces, as we saw in the example prediction in figure 5.2.

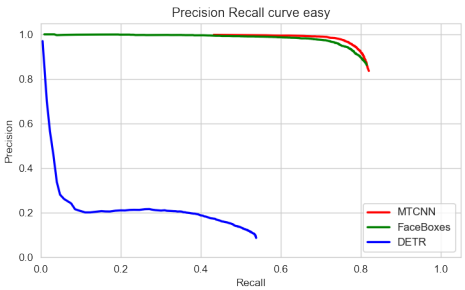
5.2.2 WIDER-face data-set results

Like sub-section 5.2.1, let's first look at some predictions. Figure 5.4 presents some prediction examples on the WIDER-face data-set. We can already observe that the predictions are quite different from the FDDB data set. For images with sizable but few faces, the predictions seem to be more or less accurate. One other observation we can make is that the performance of the DETR model for face detection seems to be related

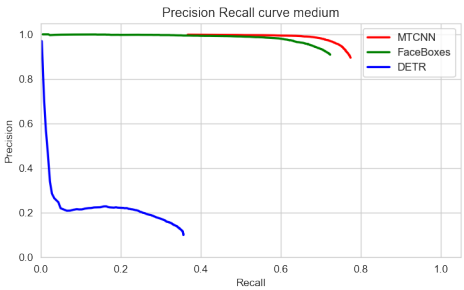
to the size of faces in the images. Indeed, looking at the top left and the top center images, although the number of faces is similar, the prediction quality is quite different. The center image presents predictions with more accuracy than the left image, where we have duplicate boxes and the position of predictions seem to be less centered on faces. The bottom left image presents a large crowd, and the model fails to recognize faces accurately. It seems to output random boxes, but to be fair, this image is labeled as a hard one.

Figure 5.4: DETR predictions on some testing images of the WIDER-face data-set

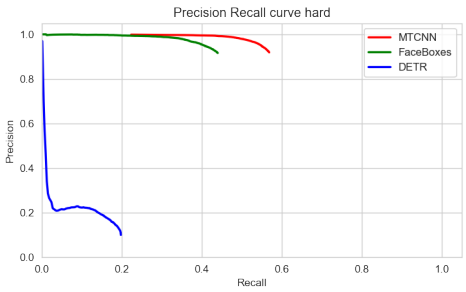
Let's look at the PR curves of the DETR, FaceBoxes, and MTCNN models shown in figure 5.5. The PR curves are divided into 3 classes, the images classified as easy, medium, and hard. We immediately see that our DETR model is way worse than the other methods. The WIDER-face data set seems to be more challenging for this model since it presents more faces per image with a smaller area per face. The model misses a lot of them (lots of FN, harming recall) as well as predicting lots of false positives (FP, hurting precision), resulting in this dip at the beginning of our PR curves. It goes to show that the FaceBoxes and MTCNN methods are better on every difficulty of images by a large margin. They are similar in performance with an edge to MTCNN for more difficult images. Table 5.1 presents the AP results of the methods. The table summarizes the conclusions we found from the PR-curves, hence that the MTCNN has a slight edge over the FaceBoxes algorithm, while both are considerably better than the DETR method applied to face detection.



(a) Easy



(b) Medium



(c) Hard

Figure 5.5: Precision-recall curves on the WIDER FACE validation subset.

	Easy	Medium	Hard
FaceBoxes	0.81811	0.73233	0.44905
MTCNN	**0.81244**	**0.76891**	**0.56390**
DETR (ours)	0.16818	0.13138	0.07405

Table 5.1: AP results for Easy, Medium and Hard sets of faces

5.2.3 Speed of methods

Now that we have seen the performance for each method, let's compare their prediction speeds. All prediction times were measured on GPU. Table 5.2 summarizes the measurements :

Method	Average time (s)	σ
FaceBoxes	**0.0185**	**0.00043**
MTCNN	0.246	0.014
DETR	0.268	0.0055

Table 5.2: Speed and standard deviation comparison

It shows that the FaceBoxes, although slightly worse than the MTCNN method, is more than 13 times faster with a small standard deviation. The DETR and the MTCNN models are comparable in their prediction times, with a slightly bigger standard deviation

for MTCNN. The transformer architecture should present some speed advantages compared to convolutional networks, but with the DETR model, it seems that the number of parameters limits the effectiveness of the predictions.

5.2.4 Discussion

One possible explanation for the disappointing results is surely related to the number of images used for training. As stated before, only a third of the available images were used for training ($\approx$ 4000 images instead of $\approx$ 12000) which is low compared to the 118 000 training images the DETR used for object detection. Data augmentation would also surely help the training process, resulting in better performance. Furthermore, the model being heavy in its number of parameters, the architecture is maybe not appropriate for single-class detection problems. So many parameters may not be necessary for face detection. An alternative model architecture with fewer parameters tailor-made for face detection might bring more satisfying results.

It was shown that the DETR model performance worsen the smaller the faces in the image get. The original model build for object detection, is less appropriate for small objects, like it was quickly discussed in section 1.2.

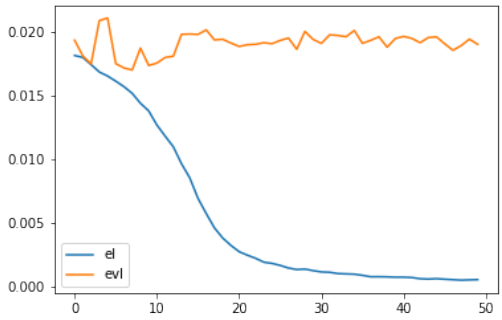
As it stands, the best model taking into account performance quality and time of predictions is the FaceBoxes method, combining fast and accurate results.

5.3 Crowd Counting

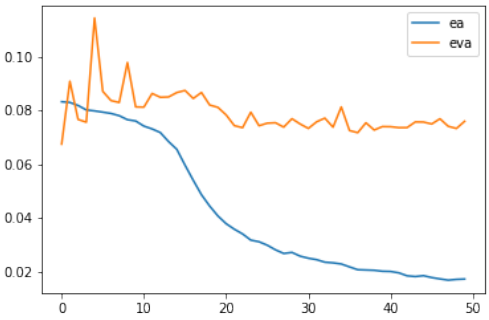
5.3.1 Regression model results

This model was trained on the WIDER data-set since, despite my attempts, no configuration was found presenting a glimpse of learning with the ShanghaiTech data-sets. It will be difficult to compare those results with our reference model CSRNet since the training sets differ. But, as you will see, our attempt at regression will not be successful. Figure 5.6 shows the training history of the *MSE* loss function with the training history of the *MAE* metric. Although, the loss and the metric present a convergence on the training set, the validation doesn't present any learning, the MSE loss and MAE metric stays basically constant, which is a clear sign of over-fitting. This also shows in the evaluation metrics, we have the following results on the testing set :

- RMSE = 13.816
- $R^2 = -0.015$



(a) Training history of the MSE loss function for regression task



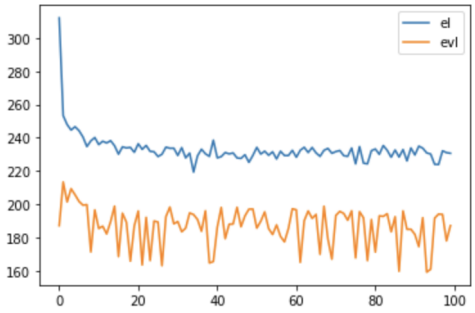
(b) Training history of the MAE metric for regression task

Figure 5.6: Training history of the ViT regression model

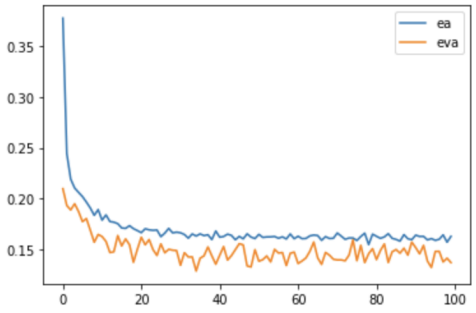
Despite being fit to execute regression tasks, it seem that our configuration for the ViT is unable to learn the task that is being asked. The model only learns the features from the training images, with no translation to the testing images. It shows in the R^2 and the RMSE metrics that our model basically outputs random values.

5.3.2 Density map generator results

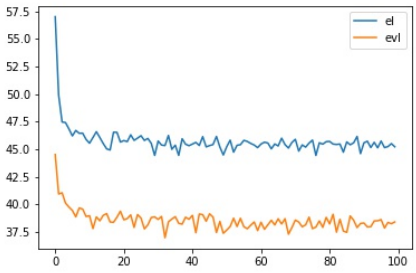
We trained the model separately on both part that compose the ShanghaiTech data-set. The results will both be presented. Let's start with the training history of the model on both parts (see Figure 5.7). Note that the validation curves are in orange while the training curves are in blue.



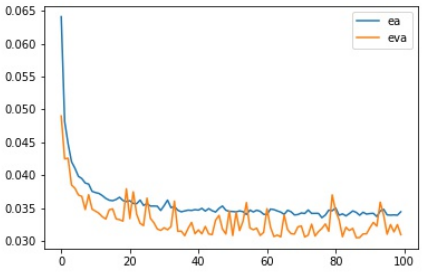
(a) Evolution of euclidean distance during training on part A of the data-set



(b) Evolution of the Mean Absolute Error between pixels values on part A of the data-set



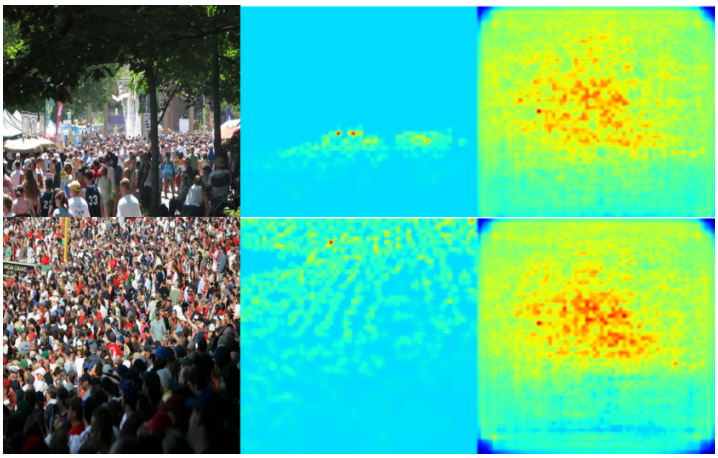
(c) Evolution of euclidean distance during training on part B of the data-set



(d) Evolution of the Mean Absolute Error between pixels values on part B of the data-set

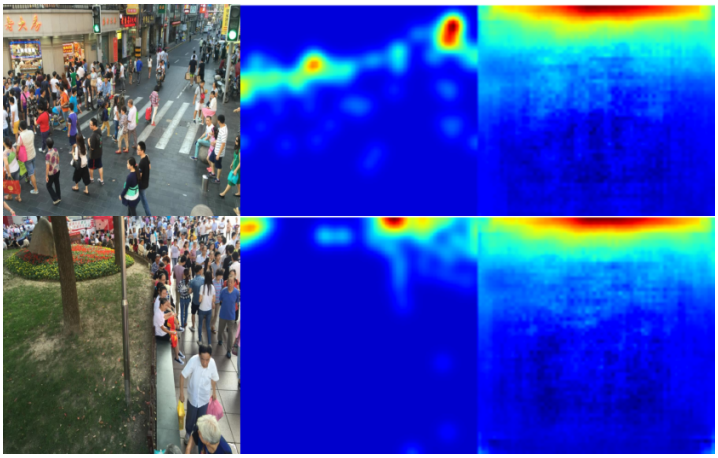
Figure 5.7: Training history of the Density maps Generator using ViTs

While some learning is present when training this model, we need to look closer at the axis to find out how well it does. As you may remark, for sub-figure 5.7a, the loss function seem to arrive at a plateau around 240. Similarly, sub-figure 5.7c, arrives at a plateau around 47.5. In both cases, these values are high, begging the question if any true learning is done. Furthermore, looking at sub-figure 5.7b and 5.7d,the Mean Absolute error present also a semblance of learning since the averaged difference between pixels generated and the ground truth pixels is diminishing. Once again, these curves are a little bit deceiving since the average difference between pixel values is stabilizing around 0.175 and 0.035 while the average pixel value is 0.002 and 0.0004 for part A and B respectively. To investigate what may cause this problem, let's look at the output of the model. The output is represented in figures 5.8 and 5.9. One may see that the generated density maps by the model are both similar, for images from part A and part B respectively.



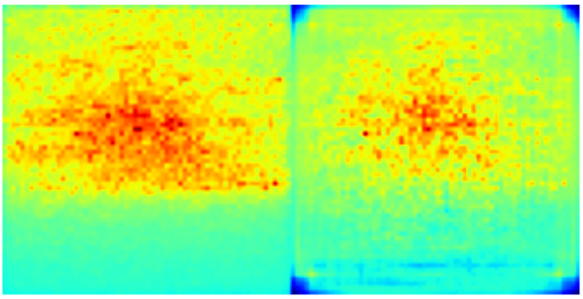
Left: original image, Center: ground truth density map,
Right: generated density map

Figure 5.8: Density map visual comparisons for part A of the data-set

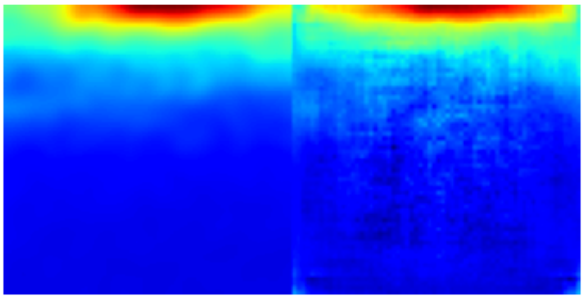


Left: original image, Center: ground truth density map,
Right: generated density map

Figure 5.9: Density map visual comparisons for part B of the data-set



(a) Left: mean ground truth density maps, Right: generated density map, both for part A of the data-set



(b) Left: mean ground truth density maps, Right: generated density map, both for part B of the data-set

Figure 5.10: Comparison between the mean ground truth density maps and the generated ones

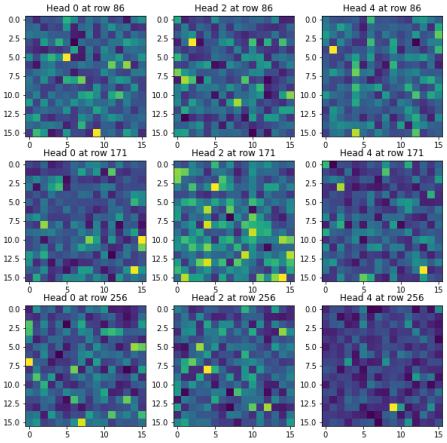
Another observation we can make is that the generated density maps do not really predict the position of faces, but rather the average distribution of faces in the images. Indeed, looking at figure 5.9, knowing that most of the faces are present at the top of the image (from section 2.1.3) and from the example images that we have, it seems that the model learns the average 2D distribution of faces rather than the distribution of a given image. The same observation can be done for the model trained on part A of the data-set. To verify this idea, let's compare the generated density map from an arbitrary example image compared to the mean ground truth density map. The average ground truth density map (AGT) is calculated the following way :

$$\text{AGT(i,j)} = \frac{1}{N}\sum_{n}^{N} p_n(i,j) \quad \forall i,j \in (W,H) \tag{5.3}$$

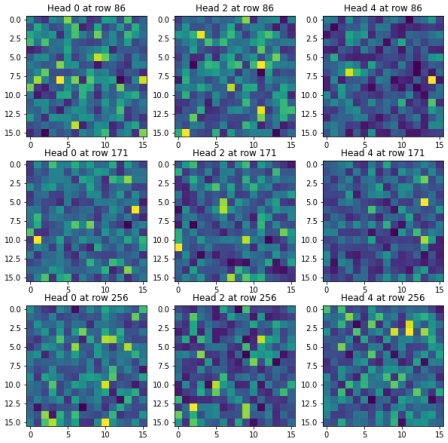
where AGT(i,j) is the pixel at position (i,j) of the AGT, $p(i,j)$ is the pixel value of the ground truth density map n at position (i,j), W and H are the width and height of the density map (in our case (64,64)). The comparison between the AGT and the arbitrary chosen generated density map is illustrated in figure 5.10 (Note: the generated density maps illustrated are normalized for demonstration sake since the pixel values tend to be bigger than from the ground truth ones). Indeed, from this figure, we can see a clear resemblance between the AGT and the generated density maps, especially from the

illustration of part B of the data set. Sub-figure 5.10a also shows some similarity between the AGT and the output, but isn't as similar as the other sub-figure 5.10b shows. It seems that our model does a better job at estimating the average distribution, thus yielding a count somewhat close to the average number of people in an image, rather than counting the number of people from the input image. The count of people is always around 447 and 108 while the average number of people are around 501 and 124 for part A and B of the data set respectively.

We can look at some of the attention maps given to the deconvolutional layers to maybe understand what fails in the training process. Figure 5.11 shows the attention maps of heads 0,2 and at rows 86, 171, and 256 for part A and B respectively. The input image resulting by these attention maps are the second original image (the ones below) from figure 5.8 and 5.9. Unlike the examples in section 3.3, we can't see any significant activation of the attention, while we could have expected an activation more or less at the center of the attentions maps for sub-figure 5.11a and more on the top of the attention maps for sub-figure 5.11b. One possible conclusion we can make is that the transformer is unable to recognize where the attention needs to be focused. It may be interesting to modify the model to increase the dimensions of the attention maps we give the deconvolution layers since it seems (from a human point of view) that the attention maps do not have the resolution required to make conclusions, at the expense of the increase of the number of parameters constituting the model.



(a) Attention maps of heads 0,2 and 4 for rows 86,171 and 256 for part A of the dataset



(b) Attention maps of heads 0,2 and 4 for rows 86,171 and 256 for part B of the dataset

Figure 5.11: Attention maps of multiple heads for multiple rows

Nevertheless, we can still evaluate the generated density maps. Unsurprisingly, table 5.4 and 5.3 shows that the reference algorithm CSRNet is better in every categories of evaluation. The quality of the density maps generated is far from state-of-the-art quality, which was expected given the discussion given above. Thus, the count of people suffers

also from the lackluster performance of the model.

	part A		part B	
Method	PSNR	SSIM	PSNR	SSIM
CSRNet	**23.02**	**0.45**	**27.43**	**0.87**
DMG(ours)	10.78	0.09	14.62	0.63

Table 5.3: Quality of density map

	part A		part B	
Method	RMSE	MAE	RMSE	MAE
CSRNet	**106.02**	**66.30**	**15.97**	**9.83**
DMG(ours)	352.79	259.95	96.37	68.99

Table 5.4: Estimation errors on ShanghaiTech dataset

5.3.3 Discussion

Once again, the two different parts of the data set both seem to be a little too small for the models proposed here. Some data augmentation would surely benefit any model trying to learn crowd counting following the given pipeline.

The density map generator learns the average distribution of the face placements rather than the actual face positions like we quickly discussed. It seems that the model is having a hard time to really localize the features from the input. One idea that came was to maybe develop a residual layer in the ViT, to reinforce the positional features of the input. Some implementation exists (ReViT) but for medical imaging applications, it could be interesting to apply those to similar problems that we discussed.

5.4 Future Improvements

The first improvement proposed is related to data. In both problems (face detection and crowd counting), data seem to be a limiting factor. The DETR method was trained on only one-third of the total data-set, and would surely benefit from a more complete training set. As for the crowd counting, the number of images in the training set is too low, hence some data augmentation would surely improve the training of our models.

Developing a custom architecture that would be able to feed attention maps (or other 2D features from ViTs) to convolutional layers would be beneficial in my opinion. The ViTs are able to identify features from the entire input, rather than local ones from convolution, this could serve as a useful backbone to convolutional layers to quickly enlarge their receptive field. The implementation of the density map generator, taking only a portion of the attention maps, wastes numerous parameters constituting the ViT.

Due to the novelty of the Vision Transformers, new research surrounding them comes out almost daily. Some of these research may be interesting to apply to our challenges. For instance, some good results seem to be present for *Attention Augmented Convolutional Networks* [4], where they combine self-attention mechanisms to convolution layers. This is to improve on the weakness of convolutional layers that operate only on local neighborhoods, while self-attention can capture long-range interactions. This study achieves a 1.3% top-1 accuracy improvement on ImageNet classification over a ResNet50 baseline and outperforms other attention mechanisms for images.
Another promising research is *Convolutions to Vision Transformers* (CvT) [34]. It introduces to ViTs convolutions, yielding the best of both designs. It allows to have a convolution-like output, hence feature maps. It shows that the CvT achieves state-of-the-art performance over other Vision Transformer implementations and ResNets on ImageNet-1k, with fewer parameters. This work would have potentially been helpful if discovered earlier, especially for the density map generator using ViTs.

5.5 Application on auditorium images

Our initial objective was to count the number of people in an auditorium image captured by a RaspberryPi. One may ask about what is the best method based on the ones we saw in this master's thesis. We can already rule out our regression model and density map generator proposed since those haven't learned to recognize the problem. This leaves us with the DETR applied to face detection, the FaceBoxes, the MTCNN, and the CSRNet (crowd counting) method. Let's first compare the face-detection algorithms. Figure 5.12, note that bigger copies of those images are available in the Annexes section 6 with some additional images, in order to better distinguish the predictions.

As our results show, the DETR model is not able to recognize small faces, hence a lot of duplicate boxes are present with poor precision. The MTCNN and FaceBoxes methods show similar results. Both have few false positives, detecting a similar number of people in the images. Of course, the performances might fluctuate depending on the position of the camera. In my opinion, the FaceBoxes method seems slightly preferable. It seems that the method presents lesser false positives overall while having similar right detections as the MTCNN. In addition, the FaceBoxes method uses significantly less computing power than the MTCNN.

Figure 5.12: Predictions for the different face detection methods, first line is the predictions of the DETR, second line is the predictions of the FaceBoxes, third line is the predictions of the MTCNN

Let's see what the CSRNet methods tell us when feeding auditorium images. Some examples are shown by figure 5.13, note that bigger copies of these figures are available in annex section 6, with additional examples. The text added to the density map is the count predicted by the model, while the left image is the input to the model. We can see that the CSRNet model seems to recognize some spatial information about the position of faces, the density maps seem, from afar, more or less accurate, but the actual count is not. The model over-counts for most of the example images. Lastly, the predictions take a considerable amount of time compared to the face detection methods.

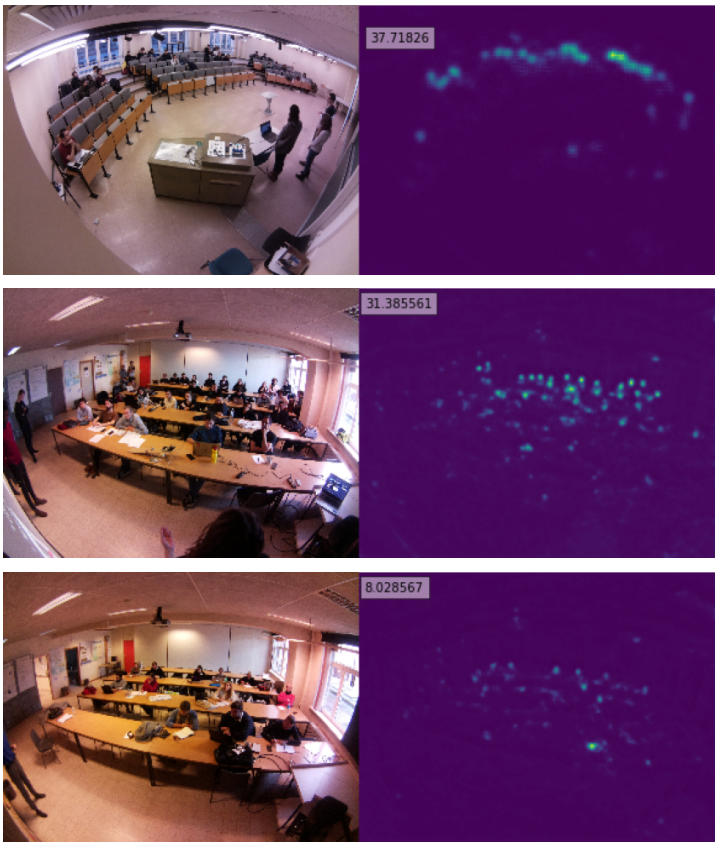


Figure 5.13: Density maps of classrooms pictures

We can conclude that the best technique for people counting applied to our auditorium images is either the FaceBoxes or the MTCNN method, with a personal slight preference for the FaceBoxes. The DETR method is not able to be accurate enough, plus doesn't show time advantages compared to the MTCNN method, while FaceBoxes does. Furthermore, the CSRNet method is not an appropriate solution for people counting in small auditoriums as we looked at, where a precise count for a low number of people is required. It may be more appropriate to use them for bigger auditoriums and thus bigger crowds.

Chapter 6

Conclusion

Despite the disappointing results of our proposed methods, we still found some improvements compared to the original algorithm used by the microcontroller thanks to the research of the state-of-the-art surrounding detection techniques. For face detection, the FaceBoxes method seems to be the best when balancing the efficiency and accuracy of predictions.

The Vision Transformers and their intricacies were explored. It discover more about this new machine learning algorithm, and its advantages and disadvantages. In this research, some particular advantages of the Vision Transformers over the more traditional Convolutional Neural Networks were not found, but still, have lots of room for future research and testing. Despite the frustration caused by the lack of successful results, it allowed me to learn a lot about the work that goes into the research and development of new computer vision methods. One positive aspect of this work is that, research is still to be done, and some avenues were slightly explored in this study. Further improvements are meant to be for crowd counting and face detection problems using Vision Transformers.

While redacting this master's thesis, some questions arose. Does the promise of Vision Transformers and their performances justify the cost of having to create such large data sets necessary for ViTs to become competitive ? Are ViTs a trend or are they the solution to computer vision problems ?

References

[1] Sayak Paul Aritra Roy Gosthipaty. *Investigating Vision Transformer representations*. URL: https://keras.io/examples/vision/probing_vits/. (accessed: 25.07.2022).

[2] Microsoft Research Lab - Asia. *Five reasons to embrace Transformer in computer vision*. URL: https://www.microsoft.com/en-us/research/lab/microsoft-research-asia/articles/five-reasons-to-embrace-transformer-in-computer-vision/. (accessed: 07.06.2022).

[3] Visual Behavior. *DETR-tensorflow*. URL: https://github.com/Visual-Behavior/detr-tensorflow. (accessed: 23.05.2022).

[4] Irwan Bello et al. *Attention Augmented Convolutional Networks*. 2019. DOI: 10.48550/ARXIV.1904.09925. URL: https://arxiv.org/abs/1904.09925.

[5] Lokesh Boominathan, Srinivas S S Kruthiventi, and R. Venkatesh Babu. *CrowdNet: A Deep Convolutional Network for Dense Crowd Counting*. 2016. DOI: 10.48550/ARXIV.1608.06197. URL: https://arxiv.org/abs/1608.06197.

[6] Nicolas Carion et al. *End-to-End Object Detection with Transformers*. 2020. DOI: 10.48550/ARXIV.2005.12872. URL: https://arxiv.org/abs/2005.12872.

[7] Cheng Chi et al. *Selective Refinement Network for High Performance Face Detection*. 2018. arXiv: 1809.02693 [cs.CV].

[8] Google Developers. *Classification: ROC Curve and AUC*. URL: https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc?hl=fr. (accessed: 18.08.2022).

[9] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2021. arXiv: 2010.11929 [cs.CV].

[10] Vijay Dubey. *Evaluation Metrics for Object detection algorithms*. URL: https://medium.com/@vijayshankerdubey550/evaluation-metrics-for-object-detection-algorithms-b0d6489879f3. (accessed: 18.08.2022).

[11] M. Everingham et al. "The Pascal Visual Object Classes Challenge: A Retrospective". In: *International Journal of Computer Vision* 111.1 (Jan. 2015), pp. 98–136.

[12] Hugging Face. *Object detection with Vision Transformers*. URL: https://huggingface.co/transformers/v4.7.0/model_doc/detr.html. (accessed: 10.08.2022).

[13] Rohini G. *Everything you need to know about VGG16*. URL: https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918. (accessed: 12.08.2022).

[14] Jacob Gildenblat. *Exploring Explainability for Vision Transformers*. URL: https://jacobgil.github.io/deeplearning/vision-transformer-explainability#q--k--v-and-attention. (accessed: 12.08.2022).

[15] Jacob Gildenblat. *Unofficial Walkthrough of Vision Transformer*. URL: https://jacobgil.github.io/deeplearning/vision-transformer-explainability. (accessed: 25.07.2022).

[16] Hiroto Honda. *Unofficial Walkthrough of Vision Transformer*. URL: https://colab.research.google.com/github/hirotomusiker/schwert_colab_data_storage/blob/master/notebook/Vision_Transformer_Tutorial.ipynb#scrollTo=J9lOBfezfPCX. (accessed: 23.05.2022).

[17] Vidit Jain and Erik Learned-Miller. *FDDB: A Benchmark for Face Detection in Unconstrained Settings*. Tech. rep. UM-CS-2010-009. University of Massachusetts, Amherst, 2010.

[18] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. DOI: 10.48550/ARXIV.1412.6980. URL: https://arxiv.org/abs/1412.6980.

[19] Yuhong Li. *CSRNet-pytorch*. URL: https://github.com/leeyeehoo/CSRNet-pytorch. (accessed: 03.08.2022).

[20] Yuhong Li, Xiaofan Zhang, and Deming Chen. *CSRNet: Dilated Convolutional Neural Networks for Understanding the Highly Congested Scenes*. 2018. DOI: 10.48550/ARXIV.1802.10062. URL: https://arxiv.org/abs/1802.10062.

[21] mustafac. *Understanding Q,K,V In Transformer(Self Attention)*. URL: https://medium.com/analytics-vidhya/understanding-q-k-v-in-transformer-self-attention-9a5eddaa5960. (accessed: 12.08.2022).

[22] Sarang Narkhede. *Understanding AUC - ROC Curve*. URL: https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5. (accessed: 19.08.2022).

[23] Facebook Research. *Hands-on tutorial for DETR*. URL: https://colab.research.google.com/github/facebookresearch/detr/blob/colab/notebooks/detr_attention.ipynb#scrollTo=eg4RK8JiYTEl. (accessed: 13.06.2022).

[24] Scikit-learn. *Precision-Recall*. URL: https://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html. (accessed: 18.08.2022).

[25] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014. DOI: 10.48550/ARXIV.1409.1556. URL: https://arxiv.org/abs/1409.1556.

[26] Vishwanath A. Sindagi and Vishal M. Patel. *Generating High-Quality Crowd Density Maps using Contextual Pyramid CNNs*. 2017. DOI: 10.48550/ARXIV.1708.00953. URL: https://arxiv.org/abs/1708.00953.

[27] Rana Talha. *CROWD-COUNTING-USING-CSRNET*. URL: https://github.com/RTalha/CROWD-COUNTING-USING-CSRNET. (accessed: 03.08.2022).

[28] Stanford University. *Generalized Intersection over Union*. URL: https://giou.stanford.edu. (accessed: 19.08.2022).

[29] Ashish Vaswani et al. *Attention Is All You Need*. 2017. DOI: 10.48550/ARXIV.1706.03762. URL: https://arxiv.org/abs/1706.03762.

[30] Phil Wang. *Object detection with Vision Transformers*. URL: https://github.com/lucidrains/vit-pytorch. (accessed: 12.04.2022).

[31] Wikipedia. *Coefficient of determination*. URL: https://en.wikipedia.org/wiki/Coefficient_of_determination. (accessed: 14.08.2022).

[32] Wikipedia. *Peak signal-to-noise ratio*. URL: https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio. (accessed: 14.08.2022).

[33] Wikipedia. *Structural Similarity*. URL: https://fr.wikipedia.org/wiki/Structural_Similarity. (accessed: 14.08.2022).

[34] Haiping Wu et al. *CvT: Introducing Convolutions to Vision Transformers*. 2021. DOI: 10.48550/ARXIV.2103.15808. URL: https://arxiv.org/abs/2103.15808.

[35] Shuo Yang et al. "WIDER FACE: A Face Detection Benchmark". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[36] Shifeng Zhang et al. *FaceBoxes: A CPU Real-time Face Detector with High Accuracy*. 2017. DOI: 10.48550/ARXIV.1708.05234. URL: https://arxiv.org/abs/1708.05234.

Annexes

Auditoriums facedetection (bigger size for observations)

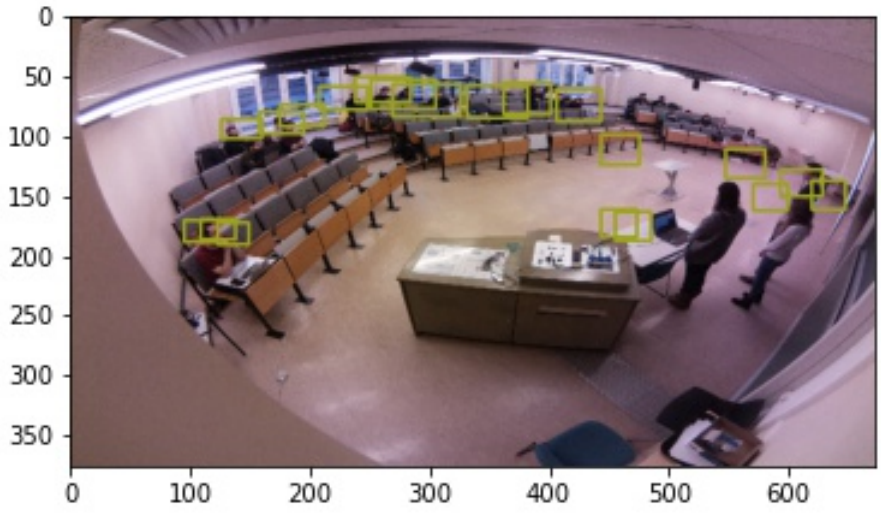


Figure 6.1: DETR predictions for auditoriums image 1

Figure 6.2: FaceBoxes predictions for auditoriums image 1

Figure 6.3: MTCNN predictions for auditoriums image 1

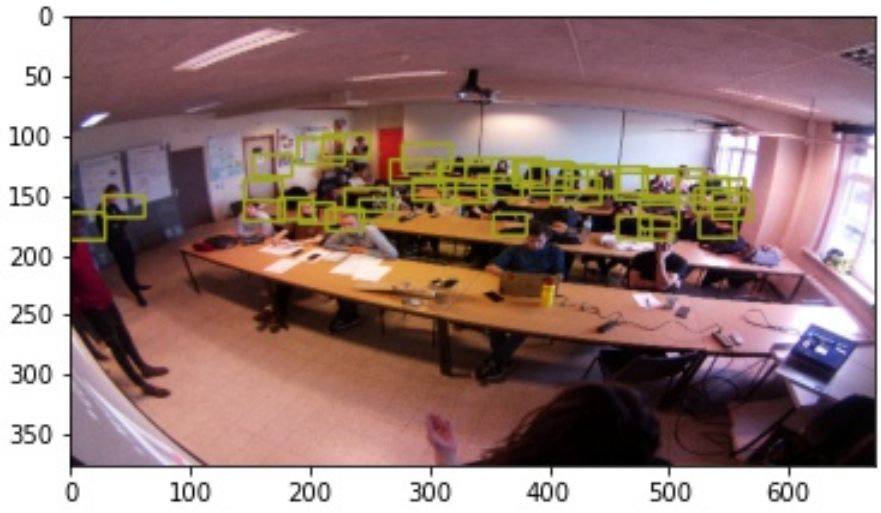


Figure 6.4: DETR predictions for auditoriums image 2

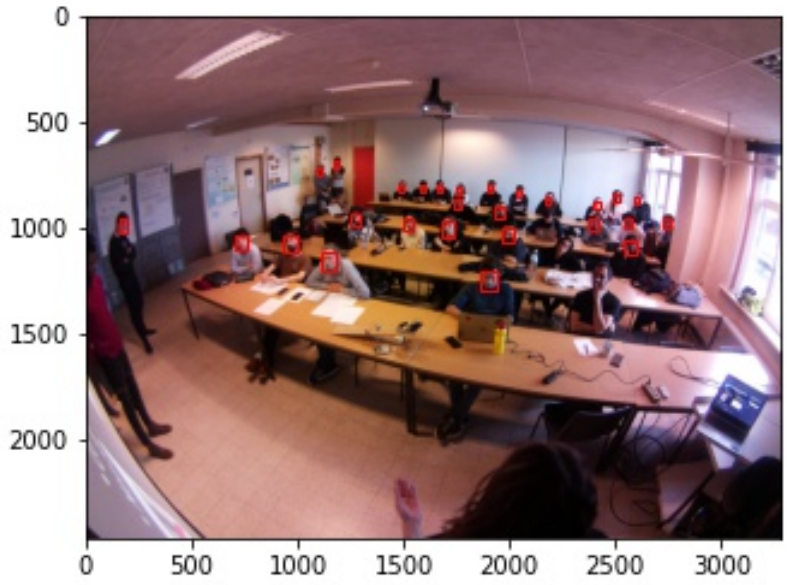


Figure 6.5: FaceBoxes predictions for auditoriums image 2

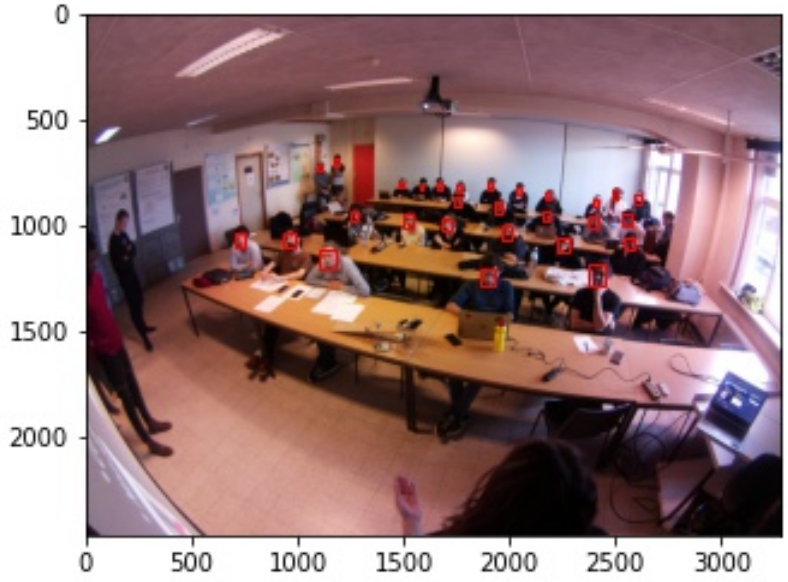


Figure 6.6: MTCNN predictions for auditoriums image 2

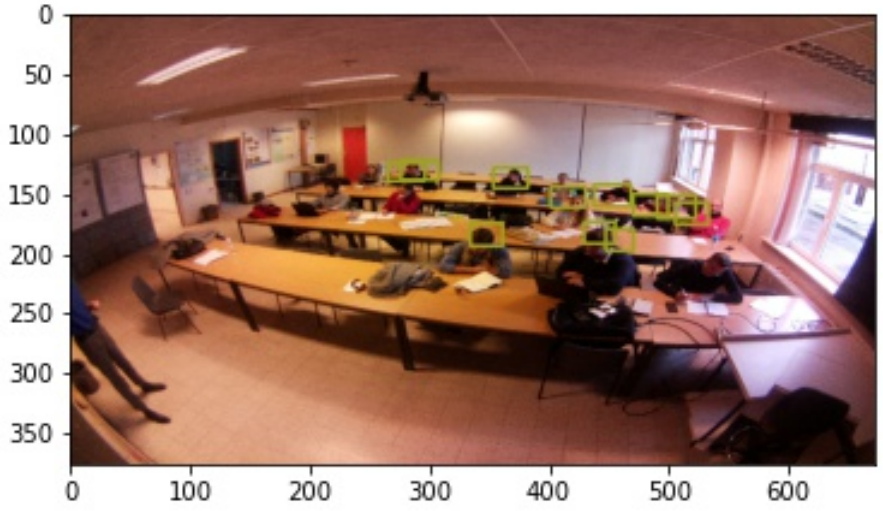


Figure 6.7: DETR predictions for auditoriums image 3

Figure 6.8: FaceBoxes predictions for auditoriums image 3

Figure 6.9: MTCNN predictions for auditoriums image 3

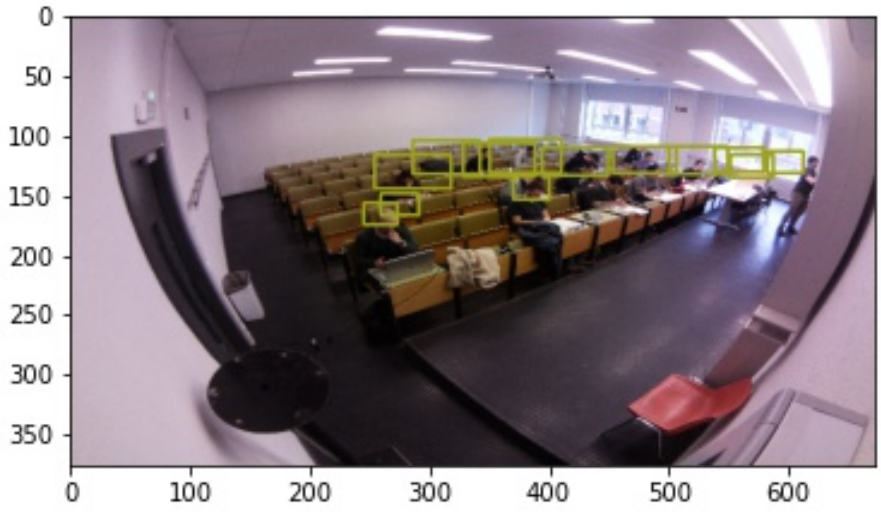


Figure 6.10: DETR predictions for auditoriums image 4

Figure 6.11: FaceBoxes predictions for auditoriums image 4

Figure 6.12: MTCNN predictions for auditoriums image 4

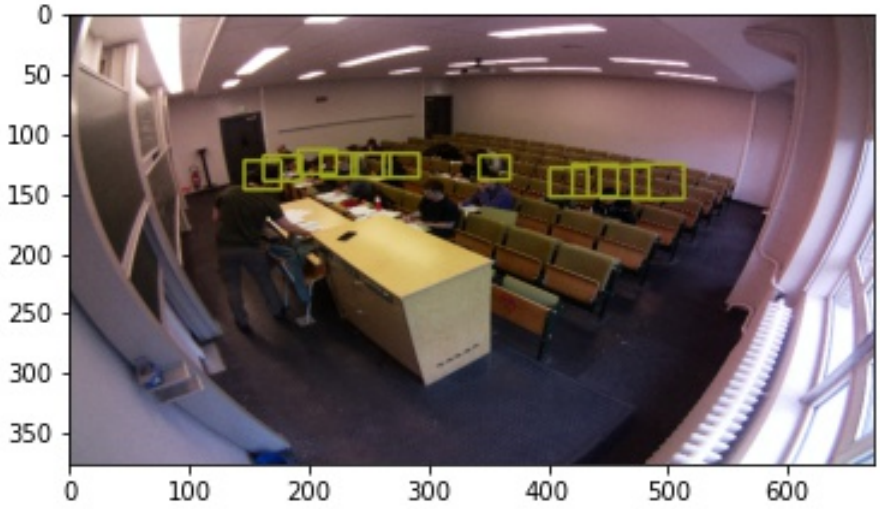


Figure 6.13: DETR predictions for auditoriums image 5

Figure 6.14: FaceBoxes predictions for auditoriums image 5

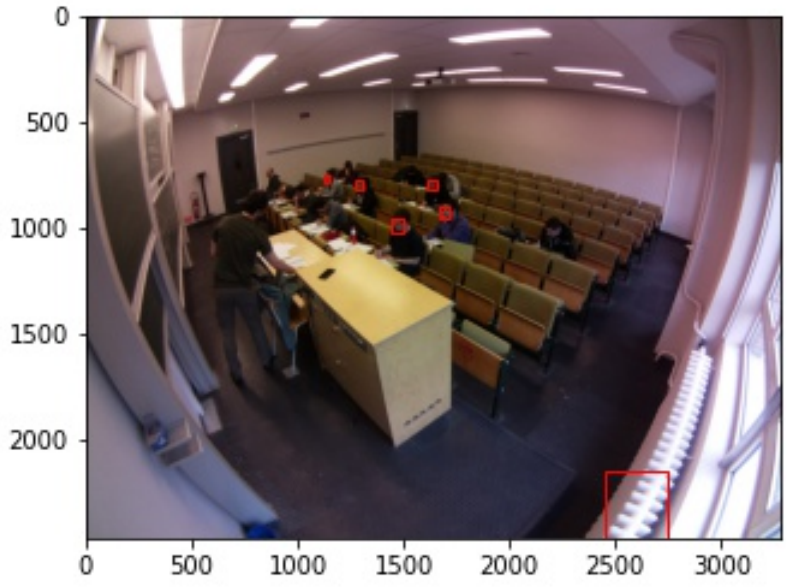


Figure 6.15: MTCNN predictions for auditoriums image 5

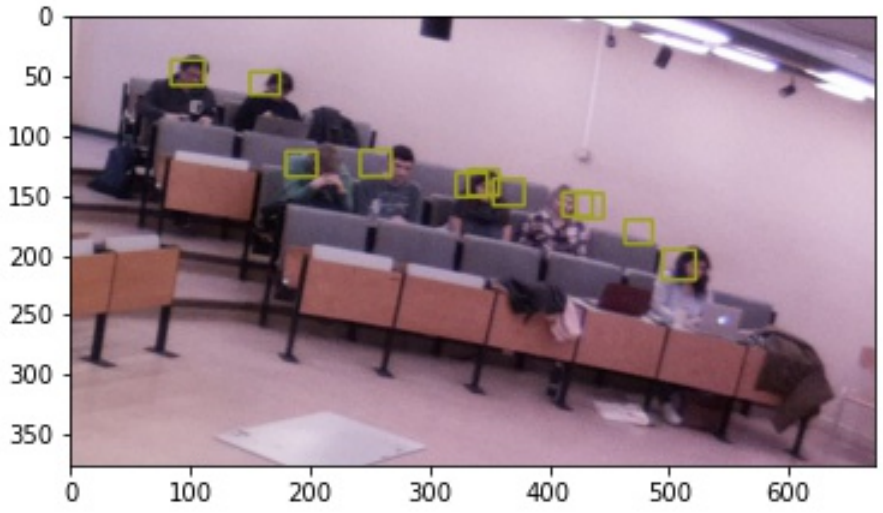


Figure 6.16: DETR predictions for auditoriums image 6

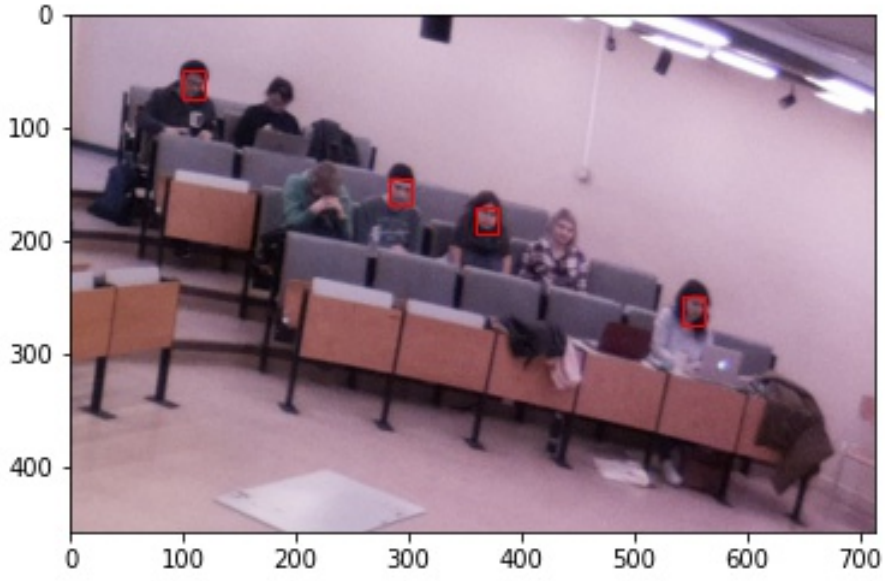


Figure 6.17: FaceBoxes predictions for auditoriums image 6

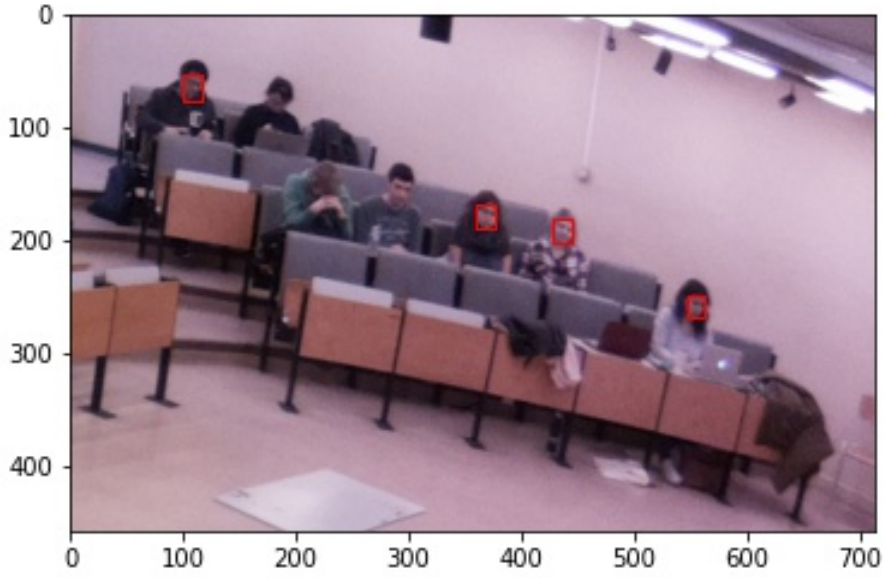


Figure 6.18: MTCNN predictions for auditoriums image 6

Auditoriums density map generation (bigger size for observations)

Figure 6.19: CSRNet predictions for auditoriums image 1

Figure 6.20: CSRNet predictions for auditoriums image 2

Figure 6.21: CSRNet predictions for auditoriums image 3

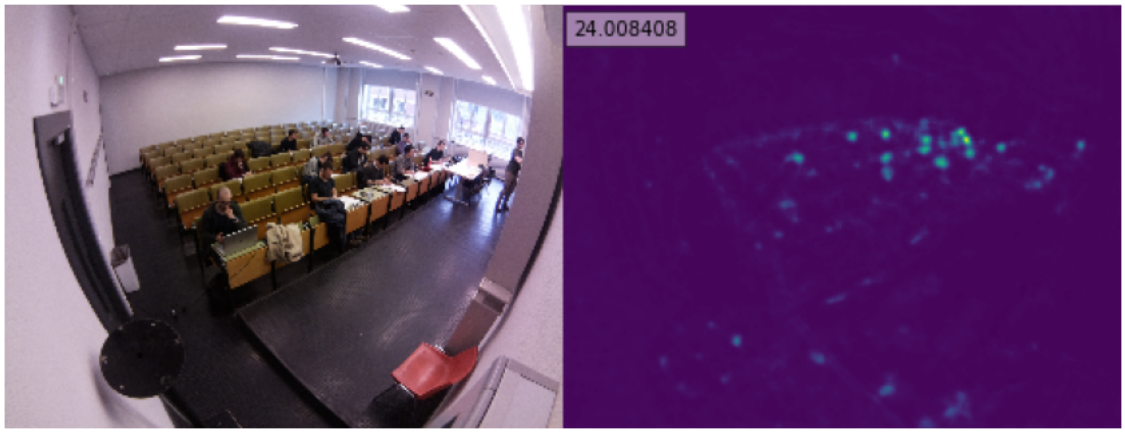


Figure 6.22: CSRNet predictions for auditoriums image 4

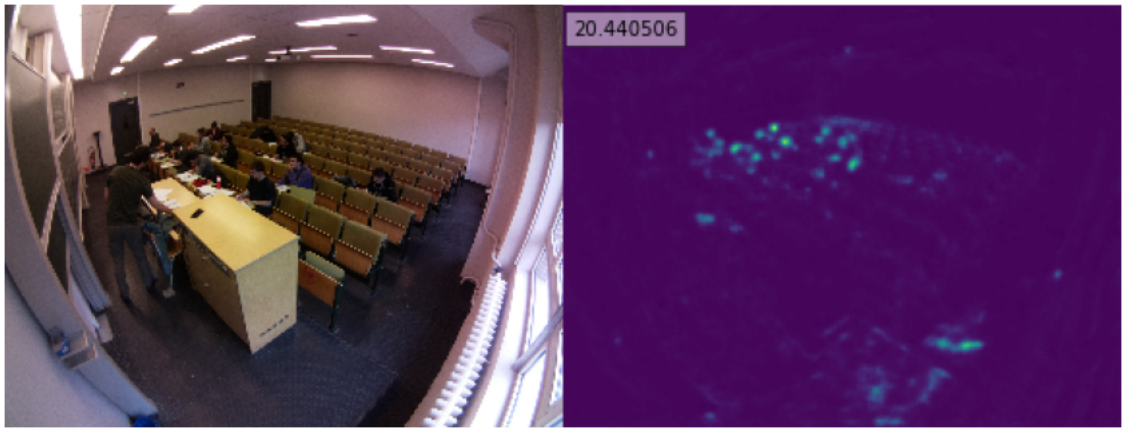


Figure 6.23: CSRNet predictions for auditoriums image 5

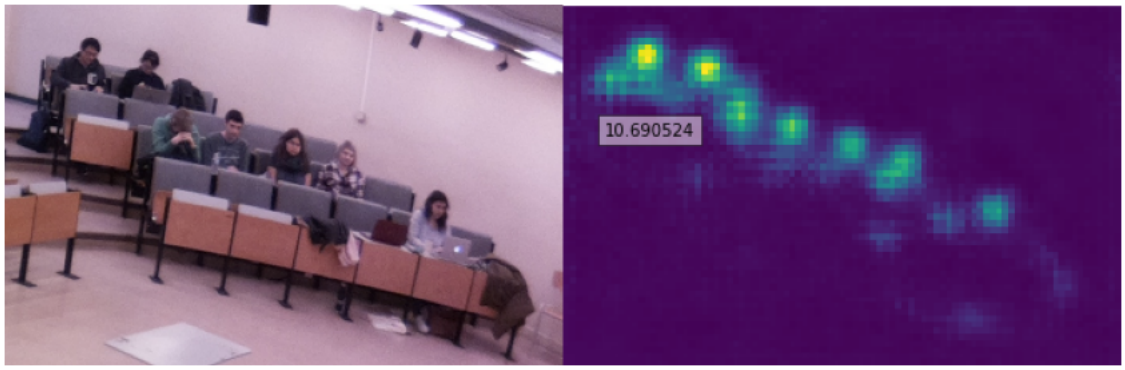


Figure 6.24: CSRNet predictions for auditoriums image 6