IPOL IMAGE PROCESSING ON LINE

Published in Image Processing On Line on YYYY–MM–DD.
Submitted on YYYY–MM–DD, accepted on YYYY–MM–DD.
ISSN 2105–1232 © YYYY IPOL & the authors CC–BY–NC–SA
This article is available online with supplementary materials,
software, datasets and online demo at
https://doi.org/10.5201/ipol

Two Deep Learning Solutions for Automatic Blurring of Faces in Videos

Roman Plaud[1], Jose-Luis Lisani[2]

[1] École des Ponts, ParisTech, France
(plaud.roman@gmail.com)
[2] Universitat de les Illes Balears, Spain
(joseluis.lisani@uib.es)

Communicated by *Demo edited by* Jose-Luis Lisani

PREPRINT June 9, 2023

Abstract

The widespread use of cameras in everyday life situations generates a vast amount of data that may contain sensitive information about the people and vehicles moving in front of them (location, license plates, physical characteristics, etc). In particular, people's faces are recorded by surveillance cameras in public spaces. In order to ensure the privacy of individuals face blurring techniques can be applied to the collected videos. In this paper we present two deep-learning based options to tackle the problem. First, a direct approach, consisting of a classical object detector (based on the YOLO architecture) trained to detect faces, which are subsequently blurred. Second, an indirect approach, in which a Unet-like segmentation network is trained to output a version of the input image in which all the faces have been blurred.

Source Code

The source code and documentation for this algorithm, together with an online demo, are available from the web page of this article[1]. The original source codes are borrowed from http://www.ipol.im/pub/art/2022/403/, https://github.com/jantic/DeOldify and https://github.com/elyha7/yoloface. Usage instructions are included in the README.md file of the archive.
This is an MLBriefs article, the source code has not been reviewed!

Keywords: privacity; surveillance; face blurring, YOLO, DeOldify, Unet

[1] https://doi.org/10.5201/ipol

1 Introduction

The emergence of laws that enforce of the privacy of individuals appearing in images and videos taken at public spaces has led to the development of methods to anonymize visual data. In this paper we address the problem of face anonymization, in particular we seek to investigate methods to automatically blur people's faces.

We focus on the use of deep-learning techniques, and we investigate how two popular architectures can help to tackle the problem.

First we take a direct approach: detect the faces and then blur them in a post-processing step. This can be achieved using an object detector and we choose in our tests the well-known YOLO architecture, since it is fast and accurate and a version specifically trained to detect faces is publicly available[2]. Section 2 gives details of the architecture, the training set and the post-processing of the detections to obtain the final results.

We also investigate the use of another approach in which faces are not directly detected, but they are directly blurred using a segmentation network. A recently proposed Unet-based network called DeOldify[3] has shown outstanding results for the colorization of gray images. An open source implementation of the method, together with a rigorous analysis of its steps, was presented in [9]. Even if the network is not designed to detect the objects or textures present in an image, it has the ability to ascertain which is the appropriate color for each pixel of the images. We wanted to check if this architecture, when trained with pairs of original and processed images (in which the faces had been blurred), was able to direcly produce the blurred results, without the need of a detector network. Details about DeOldify and its training are provided in Section 3.

Experiments comparing the results obtained with both approaches are displayed in Section 4 and some conclusions are exposed in Section 5.

2 Face Blurring using YOLO

In this section, we explore the world of face detection and the utilization of YOLOv5Face [5], a powerful face detection model, to accomplish this task. Our objective is to detect faces within images and subsequently apply a blur to these detected faces. To do so we rely on [5]. We will explain the architecture of the YOLOv5Face model as well as its and training methodology. Then we will focus on the inference methodology employed to detect and blur faces effectively.
Yolo5face is a model that draws its inspiration from the popular architecture known as YOLO (You Only Look Once). YOLO, developed by Joseph Redmon and Ali Farhadi in 2016 [6], revolutionized object detection algorithms with its real-time performance. YOLO introduced the concept of a single-stage object detector, which significantly reduced the computational complexity compared to traditional two-stage models like Faster R-CNN. YOLO5face builds upon this foundation, utilizing YOLO's efficient architecture while specializing in facial detection tasks. By leveraging YOLO's speed and accuracy, Yolo5face offers a highly effective and practical solution for detecting faces in various applications.

2.1 Architecture

YOLOv5Face is a convolution neural network (CNN) which takes as input an image and returns coordinates of faces in that image. It returns also pixels values of 5 landmarks of detected faces, which we do not use in this study. These face coordinates are encoded as boxes, namely it returns

[2] https://github.com/deepcam-cn/yolov5-face
[3] https://github.com/jantic/DeOldify

pixel coordinates of the top left and bottom right points of the boxes. We show in Figure 1 some results YOLOv5Face.

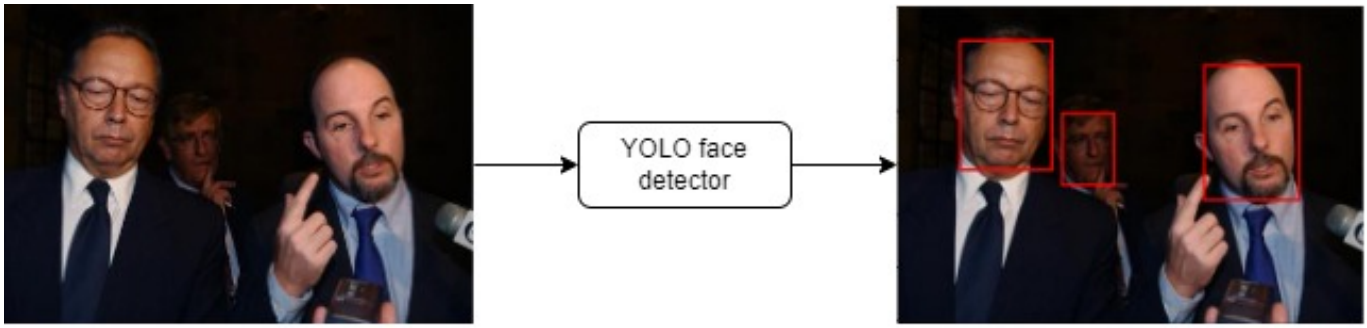

Figure 1: YOLOv5Face

As it is extensively detailed in [5] we do not delve into details regarding the YOLOv5Face architecture. In a nutshell, it consists of the backbone, neck, and head whose architecture rely on YOLOv5 [11] as depicted in Figure 2. There exist different sizes of architecture from Nano to X-Large whose number of parameters range from 1.73M to 141.1M. To speed up the experiments, our implementation is based on the Nano model which is the tiniest one and has 1.73M parameters.

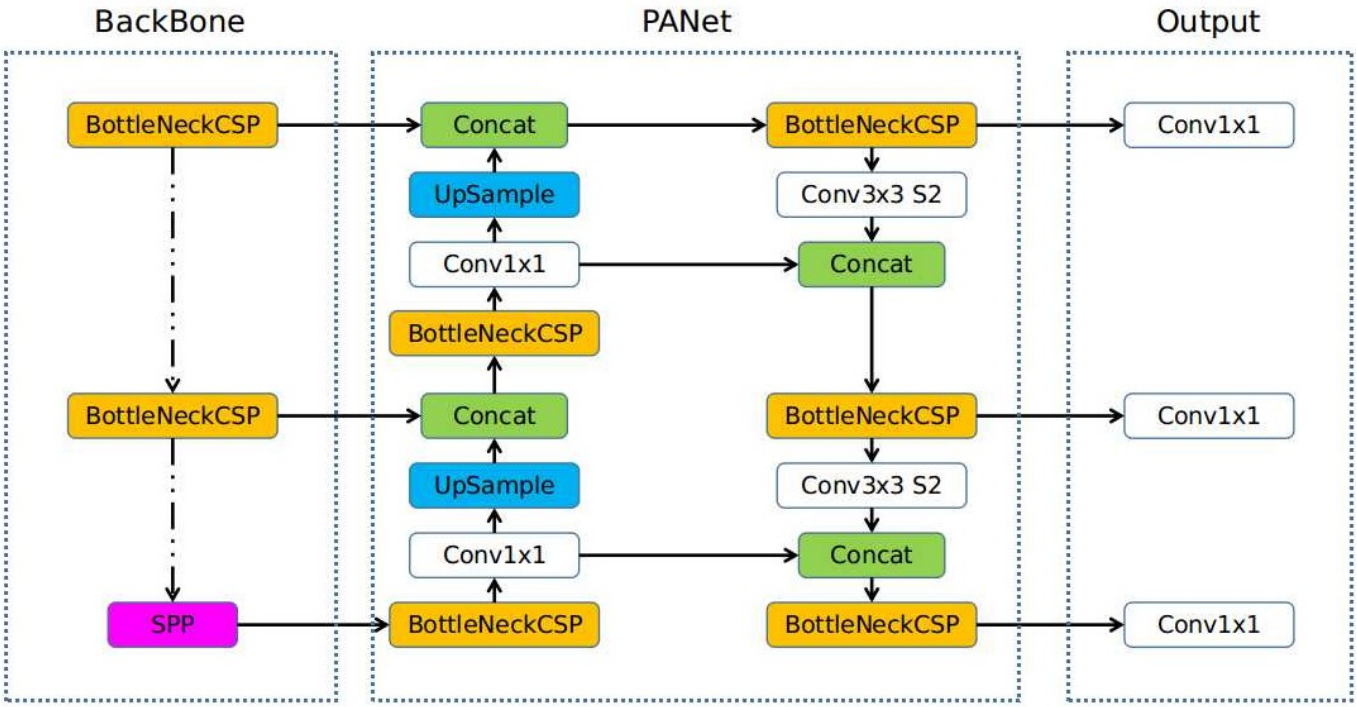

Figure 2: YOLOv5 architecture

2.2 Dataset and Training

All models were trained on the WIDER dataset [10]. Regarding the loss function, as for many YOLO model, it is a combination of losses: objectness score, class probability score, and bounding box regression score. In addition to these losses, a landmark regression loss was added to perform the 5 landmark regression. A SGD optimizer was used used and the initial learning rate was set to

1e-2 (and a final learning rate of 1e-5). The weight decay is set to 5e-3. A momentum of 0.8 is used in the first three warming-up epochs. After that, the momentum is changed to 0.937. The training runs 250 epochs with a batch size of 64.

2.3 Inference methodology

The YOLOv5Face is only a face detector so that we need postprocessing steps to blur faces. It is important to remark that, when the method is applied to a video, we operate at a frame level, whithout taking into account any temporal information. Let's consider an image, we detail below how inference is performed and sum it up in Figure 3.

Figure 3: Yolo inference methodology

1. **Face detection**. As explained, YOLOv5Face detects faces and returns a list of boxes coordinates of faces (see Figure 1). Input image can be resized to speed up detection.

2. **Transformation of boxes into ellipses**. To have a better rendering we transform the detected boxes into ellipses. To do so, we consider the ellipses of angle 0 and whose major and minor radius correspond to half the width and height of the box. We generate a mask where white pixels correspond to the areas in the image in which the network has detected a face.

3. **Face blurring**. Our face blurring technique aims at being independent of the size of the faces in the images. In fact, we do not want to apply Gaussian blur with the same standard deviation to two faces in two different images. In fact, applying a Gaussian blur with the same standard deviation would result in a blurring which would be dependent on the size of the faces. We then choose, at a frame level, the standard deviation of the blurring kernel as a function of the minimum dimension of the face detected in that frame. We then blur the whole frame with the selected standard deviation. Then we replace all the pixels of the previously created mask by its blurred version.

Remark. We could apply a Gaussian blur whose standard deviation is a function of each detected face dimension. But in the case of overlapping face, the rendering is not satisfying, creating edges that are undesirable.

Some results are displayed in Section 4.

3 Face Blurring using DeOldify

To perform the face blurring task, we heavily rely on the work in [9]. We detail in this section our motivations. First, we did not want to train a face detector, as there are already a great amount of them trained on a greater amount of data [5]. With a face detector we would have a second step that would consist in blurring all the faces detected, as explained in Section 2. What we wanted is to directly operate on the image and automatically perform blurring. In a nutshell, it corresponds to train a network that directly perform both tasks, detection and blurring.

To do so, the UNet-like architecture described in [9] seem well fitted. Notably, this model has demonstrated success in colorizing images, particularly skin tones and faces. This indicates the model's capability to perform two tasks: skin/face detection and its colorization. Consequently, we were motivated to select the same architecture for our approach.

3.1 Architecture

We keep the architecture used in [9], that is a UNet architecture [7], composed of an encoder whose weights are initialized with ResNet50 [2] checkpoint trained on ImageNet [8]. These encoder weights are **freezed** during training. The decoder is a standard Unet decoder, except from the fact that a self-attention layer is added as displayed in Figure 4 [4]. We also conducted experiments getting rid of the self-attention layer. (see Section 4).

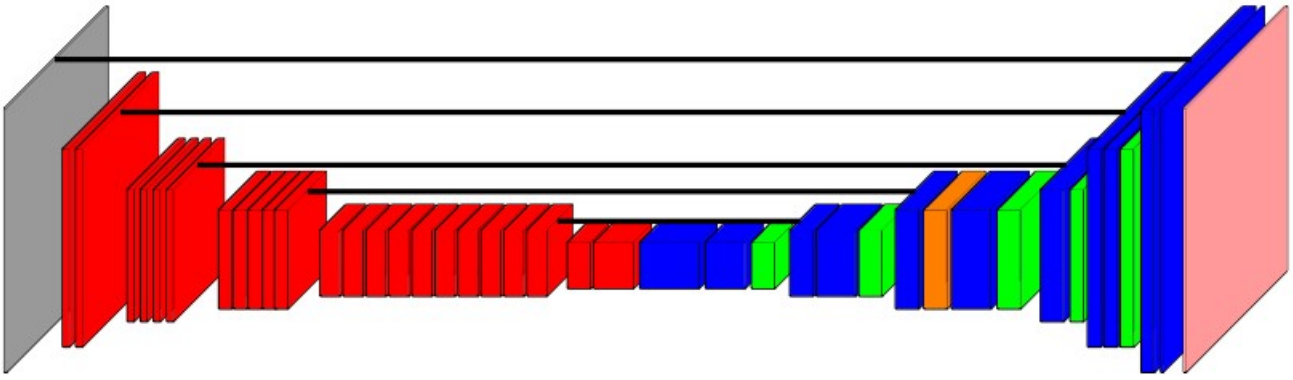

Figure 4: DeOldify architecture. In red: pretrained ResNet, in blue: convolutional blocks, in green: upsample layers, in orange: self-attention layer, and in rose: sigmoid layer. The black lines stand for the skip connections

3.2 Dataset for training

3.2.1 Datasets overview

In this section, we detail how we perform the dataset construction. We use two different datasets:

1. Face Detection Data Set and Benchmark (FDDB) [3]
2. WIDER FACE: A Face Detection Benchmark (WIDER) [10].

Both datasets are face detection datasets, which means that each image comes with annotations that provide information about the localization of every human face in the image.
The FDDB dataset contains annotations for 5171 faces in a set of 2845 images. For each image the face annotations are encrypted in an ellispse fashion (r_a, r_b, θ, x, y) where :

- (r_a, r_b) are the major and minor axis radius of the ellipse (in pixels)
- θ is the orientation angle of the ellipse
- (x, y) are the pixel coordinates of the center of the ellipse.

The WIDER dataset contains 32,203 images and label 393,703 faces. For each image the face annotations are encrypted in as rectangle fashion (x, y, w, h) where :

[4] This figure is directly extracted from [9]

- (x, y) are the pixel coordinates of the top left corner.
- (w, h) are the width and the heigh of the rectangle (in pixels)

One observation that immediately stands out is that the FDDB face annotations appear to be more precise, as they are encrypted as ellipses. However, it is worth noting that the WIDER dataset is significantly larger in size.

3.2.2 Construction methodology

As previously mentioned, the input object for this task should be the raw image, while the desired output is the same image with blurred faces. To achieve this, we must combine the image with its corresponding annotations in order to generate the blurred version. In order to achieve this we proceed exactly identically as in Section 2.3.

For the FDDB dataset annotations come in an ellipse fashion so that we perform directly **step 3** (Face Blurring). For the WIDER dataset annotations come as boxes, so that we perform **step 2** (Transformation of boxes into ellipses) and **step 3** (Face Blurring).
It is worth noting that there is no clear ground truth in our problem as several blurs can perform as efficiently the task we want to implement. We display in Figure 5 an example of both inputs and targets for both datasets.

Figure 5: Correspondance inputs-targets for an image of FDDB dataset (up) and WIDER dataset (down).

We then build a train/validation split consisting of 15148 training images pairs and 3803 validation images pairs. As the WIDER dataset came with a train/validation split we keep that split and create one only for the FBBD dataset.

3.3 Training

Training is performed in a supervised fashion with the couple of input-targets described in the previous section. In this section we detail the choice of loss function and of training procedure.

3.3.1 Loss function

In the original paper [9], training was performed using a feature loss inspired by [4]. It corresponds to computing a weighted average L1 loss of intermediate outputs of a VGG network. As we consider that the VGG network extracted features of a face and its blurred counterpart are intuitively similar, we did not think relevant to use this loss. Instead we focus on more classical losses : L1 loss and Mean Square Error (MSE) for which we recall the definition below.

- L1 loss : $l_1(\mathbf{u}, \mathbf{v}) = \frac{1}{|\mathbf{u}|} \sum_{\mathbf{i} \in \mathbf{u}} |\mathbf{u}(\mathbf{i}) - \mathbf{v}(\mathbf{i})|$
- MSE : $MSE(\mathbf{u}, \mathbf{v}) = \frac{1}{|\mathbf{u}|} \sum_{\mathbf{i} \in \mathbf{u}} (\mathbf{u}(\mathbf{i}) - \mathbf{v}(\mathbf{i}))^2$

Where u and v are, respectively, the output of the network and the ground truth. i represent a pixel and $|\mathbf{u}|$ the number of pixels of u.

3.3.2 Training Procedure

To train the network, we follow procedure in [9] which proposes a progressive training that begins by learning on smaller image size and to progressively increase the sizes of images until we reach computations limitations. At each step,

- The model is initialized with the previous step final model (Randomly if Step 1)
- Batch size is selected to match computation resources
- Learning rate is selected with regards to batch size
- Model is trained for 20 epochs on the dataset constructed

We sum up in the Table 1 the training hyperparameters of each steps of training.

	Image size	Batch Size	learning rate init.
Step 1	64×64	80	10^{-3}
Step 2	128×128	20	10^{-4}
Step 3	192×192	8	5×10^{-5}

Table 1: Training details

Training is performed using an AdamW optimizer coupled with a exponential decay learning rate scheduler of multiplicative factor of 0.8. We use a GPU NVIDIA GeForce RTX 2080 Ti of 11GB.

3.4 Inference methodology

Upon completing the training process, our objective is to develop a methodology that performs face blurring in an input image of arbitrary dimensions. As in the case of YOLO, when we want to blur a video we operate at a frame level, getting rid of any temporal information.
A naive strategy would be to directly give the raw image as input to the network but the higher the image resolution the higher the computation time will be. If we do not control the input image resolution we can not control the computation time. Then, we develop a inference methodology which we detail below and which is summed up in Figure 6. Let's consider an image, we explain below each specific block of Figure 6.

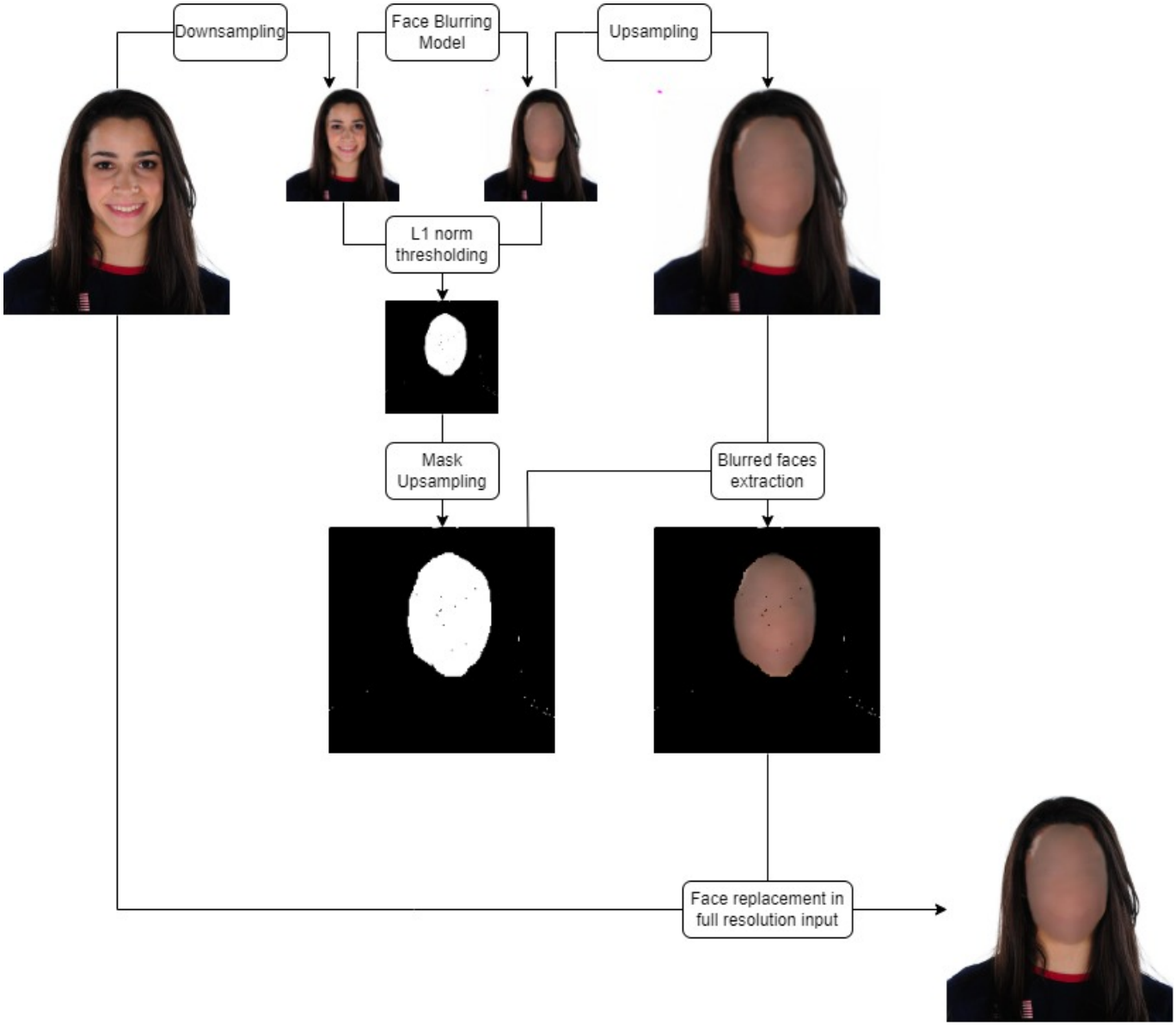

Figure 6: Inference methodology

- **Downsampling**. The input image is downsampled to a given size. In practice we use either 192×192, 256×256 or 512×512.

- **Model forward**. The resized image is passed through the network.

- **Upsampling**. The blurred image is resized to its original input size.

- **L1 norm threshold**. We compare each pixel of the downsampled input image to the downsampled blurred image. We create a binary mask with an empiric thresholding.

- **Mask Upsampling**. The mask is resized to obtain a mask with the dimension of the original input

- **Blurred faces extraction**. We combine the resized mask and the resized upsampled output to extract only faces from the output.

- **Face replacement in the full resolution input**. We combine the full resolution input with the blurred faces extracted at the previous step to construct the full resolution image with blurred faces.

What is left to determine is the choice of hyperparameters in that methodology, namely the dimension on downsizing, as well as the level of thresholding for mask extraction.

- **Choice of thresholding**.The choice of thresholding is totally empirical, after a careful study, we see that a wide range of thresholds give almost the same results. In fact, following the code implementation in [9], outputs values range from -3 to 3 so that if the model blurs a face the difference in the pixel in significantly different from 0 (on average a pixel blurred has a difference of 0.4 with respet to the original value). We then set the l_1 threshold at 0.1

- **Donwsampling size**. Regarding the size of downsampling, this hyperparameter is crucial. In fact, during training, the bigger the face is in the image, the bigger is the difference between input and target, the easier it is for the network to learn to detect and blur the face. Automatically bigger faces tend to be much more easily detected and blurred. At inference time, what we observe is that if we downsample at the size of the last step of network training (192×192) it fails to detect small faces that do not cover much pixels (see Section 4 for examples of the phenomenon). For this reason, it is interesting to set a higher downsampling size (512×512 for example), so that small faces cover more pixels and consequently are more easily detected by the network. Of course, the counterpart is that inference time in then higher. This tradeoff will be discussed in Section 4.

4 Experiments

4.1 Evaluation Methodology

In the face blurring context, evaluation can not be done completely quantitatively. In fact, for a given image, there is a large amount of different blurs that can complete the task. Still, we can compare methods using two criteria.

1. **Visual evaluation and face counting**. We evaluate visually if face blurring is correctly done through a straight-forward criteria. Correctly blurred means we cannot recognize the original person in the output image. For each test set image, we count all correctly blurred faces.

2. **Computation time**. This is a quantitative metric, which allows us to compare the inference time computation between models.

To perform evaluation we construct a test set of 6 images which features several specific particularities, namely:

- Variations in face distance to the camera
- Variations in orientation of faces (faces not facing the camera)
- Variations in number of faces
- Variations in the face itself (partially masked, wearing glasses or hats)

In this section we will test the influence on the results of several hyperparameters of Unet network to both several methods that features different hyperparameters. We will extensively study the downsampling dimension for inference, the choice of the loss, the relevancy of self-attention layer. We will also conduct some experiments using the YOLOv5Face face detector.

4.2 Experiments using DeOldify Unet

4.2.1 Influence of the downsampling dimension for inference

In this section, we study the influence of the downsampling dimension using the UNet method. We recall that the dowsampling dimension corresponds to the size to which the image is resized before passing it through the face blurring model. The last stage of training of the model uses input images of size 192×192, we then could think the network is more fitted to be inferred at this size. But practically and as we said in Section 3.4, bigger faces tend to be more easily detected and blurred, as they cover more pixels during training, which facilitates detection. That observation motivates the study of the influence of the downsampling dimension.
In this section, all results correspond to models that were trained with l_1 loss.

Visual evaluation
We show in Figure 7 results obtained for three downsampling dimensions from left to right : 192×192, 256×256 and 512×512

We observe several phenomena.

- When a face is big, meaning it covers a great percentage of the whole image (as in the two first images of Figure 7), small downsampling dimensions seem more fitted. In fact, we see in the first image, that for dimensions 192×192 and 256×256 faces are completely blurred whereas we can distinguish some face features for dimension 512×512 (The face remains unrecognizable still). Then, on such images, small downsampling dimension seem better fitted.

- When faces are smaller, the option that uses dimensions 192×192 starts to struggle a lot to blur faces. First, it does not succeed in blurring the face of a person with glasses (third image) and also misses a lot of faces in the two last images. On the other hand, the option with 512×512 dimensions almost never misses a face, and completely blurs it. The last image is blattant, the 192×192 option misses more than half of the faces, 256×256 misses a quarter of them, whereas 512×512 only misses the three smallest faces.

Table 2 displays the count of correctly blurred faces in each of the images used in the tests.

	Number of faces	Number of faces blurred			Best visual result
		192×192	256×256	512×512	
image 1	1	**1**	**1**	**1**	192×192
image 2	1	**1**	**1**	**1**	192×192
image 3	5	4	**5**	**5**	512×512
image 4	6	**6**	**6**	**6**	512×512
image 5	18	13	**18**	**18**	512×512
image 6	51	20	38	**48**	512×512

Table 2: Face counting results (the best result for each image is in bold)

We see there that, depending on the input image, the optimal downsampling size can be different. In the attached implementation, one can vary the input image size.
Still, overall the 512×512 downsampling size seem better in most cases, surpassing all other dimensions when it comes to blur small faces and still performs face blurring of big faces, even if it is visually a little worse than with smaller dimensions. However, a higher downsampling size comes also with a higher computation time. That is what we are going to briefly discuss in the next section.

Computation Time.
To perform inference and to be able to compare effectively. We always use our GPU (NVIDIA GeForce RTX 2080 Ti of 11GB) to infer images We conduct three experiments :

- One experiment assuming that input images have already been downsampled to the desired input size of the network (either 192 x 192, 256 x 256 or 512 x 512). In this case the downsampling, binary mask extraction and upsampling steps described in Figure 6 do not need to be performed.
- One experiment assuming that input images are of size 1024×1024 (we follow algorithm described in Figure 6)
- One experiment assuming that input images are of size 2048×2048 (we follow algorithm described in Figure 6)

.

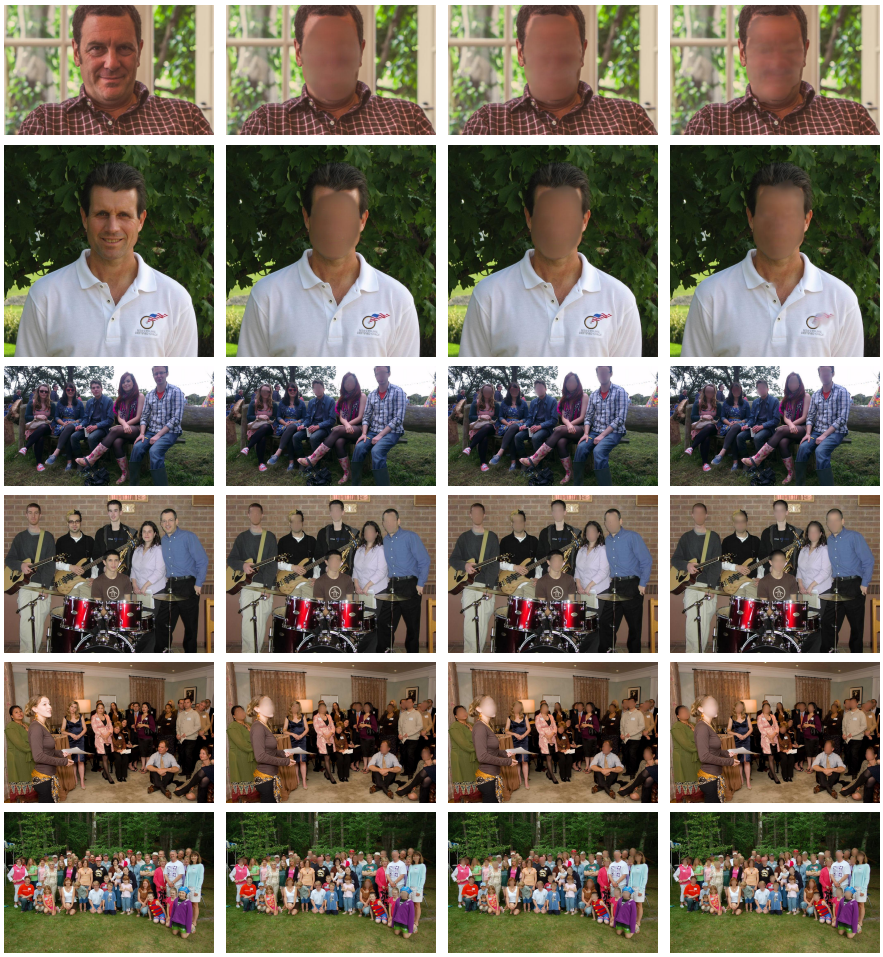

Figure 7: Test set results. Original image (left), result for 192×192 inference dimension (center left), result for 256×256 inference dimension (center right), result for 512×512 inference dimension (right)

We calculate the computation time for 100 images and average them for each experiments. Results are displayed in Table 3 and expressed in frames per second.

We observe that all the resizing steps (see Figure 6) greatly deteriorate the algorithm performance. To avoid this phenomenon, we could take advantage of multiprocessing using several cores. We did not implement this idea, but as images are processed independently, we think it could greatly improve performances. We also observed that the bigger the input image is, the more similar are all computation times.

	192×192	256×256	512×512
No resizing needed	28.4	22.0	9.6
Input size of 1024×1024	9.2	8.3	5.4
Input size of 2048×2048	3.0	2.8	2.4

Table 3: Number of frames per second processed by the algorithm in function of the downsampling size

4.2.2 Influence of the loss function

As in the previous section, we use a visual evaluation of the results to assess the influence of the loss function. We compare two models, one which was trained using MSE loss and another using l_1 loss. For both models, we use 512×512 as inference downsampling size.

Figure 8: Test set results. Original image (left), result for MSE loss (center), result for l_1 loss (right). Downsampling size $= 512 \times 512$

What stands out in Figure 8 is that results are very similar. At first sight, we struggle to distinguish any difference. But in fact, there are some. First if we look closer a at "big faces" images (the first two ones), we can see less face features in these images : face blurring is better performed with MSE. Then, for images 3, 4 and 5, results are almost the same. Finally, we see in the last image, that the MSE trained network succeeds in blurring the tiniest faces that the l1 trained did not blur. That is a already a strong improvement : MSE tends to provide better results for small face than l_1.
To be able to assert and verify this previous statement, we investigate the performance when considering an even smaller smaller downsampling size. In that framework, faces appear to cover less

pixels, turning them into smaller faces for the network. We consider a downsampling size of 192×192 and display the results for both losses in Figure 9. We observe, as expected, that the MSE trained network detects and blurs the great majority of faces in all the images while the l_1 trained network misses some of them in images 3, 5 and 6. Most impressive results being for image 5 in which the MSE trained network detects and blurs 5 more faces than the l_1 trained network. Same for image 6 in which it detects and blurs 25 more faces. This shows that MSE is more sensitive to small faces while performing as well as l_1 loss on bigger faces. We sum up in Table 4 the blurred face counting.

Figure 9: Test set results. Original image (left), result for MSE loss (center), result for l_1 loss (right). Downsampling size $= 192 \times 192$

	Number of faces	Number of faces blurred			
		MSE		l_1	
		192×192	512×512	192×192	512×512
image 1	1	**1**	**1**	**1**	**1**
image 2	1	**1**	**1**	**1**	**1**
image 3	5	**5**	**5**	4	**5**
image 4	6	**6**	**6**	**6**	**6**
image 5	18	**18**	**18**	13	**18**
image 6	51	45	**50**	20	48

Table 4: Face counting results (the best results for each image is in bold)

We do not compare the computation times in this section as there are exactly the same as in

the previous section. In fact, models have the exact same number of parameters and the pre and postprocessing steps are identical.

4.2.3 Influence of the self-attention layer

In this section, we want to study the importance of the self-attention layer and if it is relevant in our case. In [9], the introduction of this layer is motivated by the long-time studied question of non-locality in images. Paper [1] was one of the first paper to propose a non-local algorithm in the context of denoising. This seminal work proved that to denoise a given patch of an image, all other patches of the images, provided that they are similar to the original patch, contain information relevant to denoise the patch. The self-attention mechanism is a similar process, thus more general. In the context of colorizing, this layer appears to be relevant in order to combine information, and in particular to propagate colorization to patches that are similar.
In our context, we hypothesize that this could combine detected faces to improve the face blurring and robustify the blurring when there are several faces in the images.

Visual evaluation
In Figure 10, we compare the results of the MSE trained network, with a downsampling size of 512×512 with and without the self-attention layer.

Figure 10: Test set results. Original image (left), result without self-attention layer (center), with self-attention layer (right). Model trained with MSE loss. Downsampling size $= 512 \times 512$

Visually, on Figure 10 we struggle finding differences for a downsampling size of 512×512. We then display in Figure 11 the results on test set for a downsampling size of 192×192.

Figure 11: Test set results. Original image (left), result without self-attention layer (center), with self-attention layer (right). Model trained with MSE loss. Downsampling size $= 192 \times 192$

Once again, it is very hard to spot any visual difference between two models on Figure 11. Our conclusion is that it is not relevant to add this self-attention layer to our model.

Computation time

It in interesting to compare computation times with and without the self-attention layer. In fact the self-attention layer has a quadratic complexity with respect to the input dimension. To this extent, we expect that inference time with self-attention layer has a higher computation time. Here we conduct only one experiment assuming that input images are already of the right downsampling size. In fact, the only difference in computation time is when we pass through the model, pre and post processing steps are exactly the same.
We calculate computation time for 100 images and average them for each experiments. Results are displayed in Table 5 and expressed in frames per second.

	192×192	256×256	512×512
With self-attention	28.4	22.0	9.6
Without self-attention	28.8	22.6	9.8

Table 5: Number of frames per second processed by the algorithm with and without self-attention

The computation time is slightly reduced by getting rid of the self-attention layer. Still the difference is negligible considering that, in practice, we have pre and post processing steps that will

make the difference even tinier. In our study, we cannot conclude that adding a self-attention layer is relevant. First, visual differences in results are imperceptible and the difference in computation time using one method or the other is negligible.

4.2.4 Experiments using YOLOV5Face

In this section, we want to evaluate Yolo-based face blur method described in Section 2. We recall that the YOLOV5Face methodology consists in detecting faces and then blur them (see Section 2.3).

Visual evaluation

The YOLO method is very straightforward. We display in Figure 12 visual results for several input resizing (192×192, 256×256, 512×512) as well as for no resizing (keeping original input size).

Figure 12: Test set results. Original image (left), result for 192×192 inference dimension (center left), result for 256×256 inference dimension (center), result for 512×512 inference dimension (center right), result for original input size (right)

We can say that the YOLO-based method is very efficient, faces are very often detected, especially when they are "big", meaning the face covers a great number of pixels. However, and as the Unet before, for small downsampling sizes, YOLO struggles to detect small faces in images 5 and 6.
But, the higher we set the input dimension size, the higher number of faces YOLOv5Face detects. When we keep the original dimension, almost no faces are missed except one in the last image. We sum up the results in Table 6.

	Number of faces	Number of faces blurred			
		192 × 192	256 × 256	512 × 512	Original input size
image 1	1	**1**	**1**	**1**	**1**
image 2	1	**1**	**1**	**1**	**1**
image 3	5	**5**	**5**	**5**	**5**
image 4	6	**6**	**6**	**6**	**6**
image 5	18	16	16	**18**	**18**
image 6	51	40	43	49	**50**

Table 6: Face counting results (the best results for each image is in bold)

4.2.5 Computation time

We conduct three experiments :

- One experiment assuming that input images have already been downsampled to the desired input size of the network (either 192 x 192, 256 x 256 or 512 x 512). In this case the downsampling, binary mask extraction and upsampling steps described in Figure 6 do not need to be performed.
- One experiment assuming that input images are of size 1024 × 1024
- One experiment assuming that input images are of size 2048 × 2048

.

	192 × 192	256 × 256	512 × 512	1024 × 1024	2048 × 2048
No resizing needed	52.5	48.9	26.7	5.7	1.9
Input size of 1024 × 1024	7.2	6.9	6.3	5.7	
Input size of 2048 × 2048	4.4	4.2	2.4	2.0	1.9

Table 7: Number of frames per second processed by the algorithm in function of downsampling size

What stands out in Table 7 is that the number of frames processed by second by the YOLOv5Face methodology is always higher than with the Unet based method. In fact, we use here the Nano model, which has the smaller number of parameters which speeds up the inference. We observed as for the Unet based method, that all the resizing steps greatly deteriorate the algorithm performance.

5 Conclusion

In this paper, we have investigated the automatic blurring of faces in images and videos. We have presented two methods, one one-step direct method based on the Unet architecture (DeOldify [9]), and another two-step method which relies on the well-known YOLO object detector [6, 5].

The YOLO-based method first detects faces which are then blurred using a Gaussian kernel. The Unet-based method directly outputs images in which faces are blurred. We have constructed a dataset of pairs of original and face-blurred images to train this network. The original images come from two popular face-detection datasets (FDDB [3] and WIDER [10]).

The experiments show that both methods are able to correctly blur faces in images and that they are robust to variations in size and pose. In terms of computation time, the YOLO-based method is faster, since it benefits from all the optimitations introduced in the YOLO architecture to increase

its speed. However, the ability of the Unet-based network to blur the faces without detecting them explicitely is an interesting property that is worth exploring and, in the future, we will investigate how to optimize its speed.

An online demo is available for the interested readers that want to test the performance of both methods in their own videos.

Acknowledgment

For the second author the publication is part of the project PID2021-125711OB-I00, financed by MCIN/AEI/10.13039/501100011033/FEDER, EU.

Image Credits

All the original images displayed in the paper come from the FDDB [3] and WIDER [10] datasets.

References

[1] A. Buades, B. Coll, and J.-M. Morel, *A Non-Local Algorithm for Image Denoising*, in Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2 - Volume 02, CVPR '05, Washington, DC, USA, 2005, IEEE Computer Society, pp. 60–65, https://doi.org/10.1109/CVPR.2005.38.

[2] K. He, X. Zhang, S. Ren, and J. Sun, *Deep Residual Learning for Image Recognition*, CoRR, abs/1512.03385 (2015).

[3] V. Jain and E. Learned-Miller, *FDDB: A Benchmark for Face Detection in Unconstrained Settings*, (2010).

[4] J. Johnson, A. Alahi, and L. Fei-Fei, *Perceptual Losses for Real-Time Style Transfer and Super-Resolution*, CoRR, abs/1603.08155 (2016).

[5] D. Qi, W. Tan, Q. Yao, and J. Liu, *YOLO5Face: Why Reinventing a Face Detector*, 2022.

[6] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, *You Only Look Once: Unified, Real-Time Object Detection*, 2016.

[7] O. Ronneberger, P. Fischer, and T. Brox, *U-Net: Convolutional Networks for Biomedical Image Segmentation*, CoRR, abs/1505.04597 (2015).

[8] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and L. Fei-Fei, *ImageNet Large Scale Visual Recognition Challenge*, CoRR, abs/1409.0575 (2014).

[9] A. Salmona, L. Bouza, and J. Delon, *DeOldify: A Review and Implementation of an Automatic Colorization Method*, ImageProcessingOnLine, 12 (2022), pp. 347–368. https://doi.org/10.5201/ipol.2022.403.

[10] S. Yang, P. Luo, C. C. Loy, and X. Tang, *WIDER FACE: A Face Detection Benchmark*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[11] X. Zhu, S. Lyu, X. Wang, and Q. Zhao, *TPH-YOLOv5: Improved YOLOv5 Based on Transformer Prediction Head for Object Detection on Drone-Captured Scenarios*, 2021.