People Detection in Foggy Environments

Lusvardi Gianmarco Mezzina Marco Millunzi Monica

ABSTRACT

Human visual and cognitive capabilities are unique, but often they are not enough to live with nature in safe conditions. Although we are in the 21st century and sciences like computer vision are helping humans to approach challenges and obstacles in any social environment; in fact, now more than ever, Artificial Intelligence is starting to live with us and we are responsible to guide its use to achieve goals truly significant for the social community, in respect of ethical principles. One of the most interesting problem in computer vision applications for us is seeing through the fog*: we want to study the effect that fog has on the application of a classical pipeline of computer vision. The following report illustrates the computer vision software idea we realised, explaining the problems that made us think about such kind of system, the pipeline to complete the stated goals and the possible extrapolation sources of interest data.*

1 POSSIBLE APPLICATIONS

There are many possible applications of the mentioned problem of interest[1]:

- Autonomous driving in adverse weather conditions
- Drones flying in fog conditions
- Flight intelligent system to taking off and landing in adversarial fog situations

and so on, but we want to interact with a more specific aspect of this general topic, the **people detection in foggy environments**.

2 RELATED WORKS

To develop our project we inspected what had already been done, in order to have a general idea about the different existing approaches. From the beginning, we became aware that this is a sectorial problem and that we would not have much material to take inspiration from. However, we noticed that the main direction, in order to work with foggy environments, is to generate artificial fog through classical preprocessing operations (as it is discussed in the chapter 3). In fact, there are not a lot of datasets to work with our task. So, in this direction we follow the approach suggested by [5]. Initially, we read the work of [6], and we decided to work in a more general area (not limited to the viewpoint of a car driver) and without using special cameras. In fact, we worked with a completely virtual and non-physical environment for fog generation (MOTSynth dataset [2]), and used a classical deep learning pipeline for people detection (Faster-RCNN).

3 CORE STRUCTURAL ELEMENTS

The system we would like to discuss about is able to work with multimedia data in order to *detect people in foggy scenarios: to understand how much is possible to recognize human shapes in foggy weather conditions and to localize their corresponding bounding boxes.* The related experiments are usually based on very expensive data acquisition sensors (thermal cameras, LiDAR visual receptive systems) and are generally focused on street scenarios, from a car-eye perspective. What we've done involves a relaxation of these constraints, to work in very heterogeneous scenes from different field of view perspective.

Our system follows a simple pipeline:

(1) **Multimedia data acquisition**: we used a synthetic dataset called MOTSynth [2] in order to extract images with different viewpoints and conditions.
(2) **Preprocessing operations**: thanks to the depthmaps included in the MOTSynth dataset, we managed to add a simulated fog over the images, using the approaches described in [3] for creating the FRIDA dataset (Koschmieder's law).
(3) **Deep People Detector**: we trained a Faster-RCNN based on ResNet (50 layers) to do people detection.
(4) **Head Detector**: we used a neural network trained to extract the borders of images with fog and used the resulting borders to identify the head of a person using the hough transform for detecting circles.

Figure 1: Seeing through fog [4]

In order to train a fog-resistant people detector, we took two alternatives into consideration:

(1) Training a people detector with unprocessed images, then training a network specifically designed to remove the fog from the images (a denoising autoencoder).
(2) Training a people detector with images that present different thickness of fog, thus creating a network which should be able to generalize, thus detecting people disregarding the fog, which is considered noise.

We decided to go for the second approach, because the first one would require a much larger effort to train the denoiser. That's because even if the scene is foggy, some details, such as the weather, could change when the fog is removed, without compromising the general acceptability of the image. So, for each input image,

we could have many acceptable output images. Therefore if we trained such a denoiser, we would have needed a loss function which disregarded those details (for instance, the weather behind the fog or the sky color).
As the amount of datasets with images showing real fog is very limited and they have not enough images, we decided to use the MOTSynth dataset because it contains all the necesasary annotations (bounding boxes, keypoints and segmentation for people) and the depthmap for each image, a component that is very important to add artificial fog in the training images.

4 DATA PREPROCESSING

In order to add artificial fog to the input images, we followed the approach of [3] who used the Koschmieder's law to model uniform artificial fog in images, depending on the depthmap. Let L_0 be the luminance component of the original image, $d(x, y)$ the distance of the object at pixel (x, y), then the Koschmieder's law says that

$$L(x, y) = L_0(x, y)e^{-kd(x,y)} + L_s(1 - e^{-kd(x,y)}) \tag{1}$$

where k is a parameter that regulates the fog thickness (the higher k is the thicker the fog) and L_s is the luminance of the sky (it regulates the fog color).
In practice, we've seen that the same formula applies even if we replace the luminance component with an image channel, hence we avoided conversions to other color spaces and sticked with RGB.
Fog could also be non-uniform: sometimes there are patches that can have different shades of gray, patches that can have different thickness or both. Hence, always following the approach of [3], we generated two different noise maps using Perlin noise and use them to perturb k and L_s over the image. Perlin noise is a kind of noise specifically designed to be continuous and to look "natural" [7], therefore is perfect for this task. The only parameter for the perlin noise generator is its scale (which basically control the amount of "change" in a given area).
Once the perlin noise maps were generated, they were multiplied by k and L_s. If we call P_k and P_{L_s} respectively the perlin noise map generated for k and L_s, we can weight these according to a weighting coefficient w_k and w_{L_s} obtaining $\mathcal{P}_k = (1 - w_k)k + w_k P_k$ and $\mathcal{P}_{L_s} = (1 - w_{L_s})L_s + w_{L_s} P_{L_s}$

$$L(x, y) = L_0(x, y)e^{\mathcal{P}_k d(x,y)} + \mathcal{P}_{L_s}(1 - e^{-\mathcal{P}_k d(x,y)}) \tag{2}$$

The weighting coefficients mediates between a uniform fog (when $w = 0$) and fully heterogeneous fog (when $w = 1$) in both thickness and color. We noticed, however, that due to the perlin noise image being in the interval $[0, 1]$, a fully heterogeneous fog has a lot of "holes" inside, hence it is not very realistic.

4.1 Calculating visibility inside fog

In order to be able to correctly train the network, we couldn't neglect the problem of the visibility of a person inside the fog. Although it is true that the ground truth contains all the visible people in the image, once fog is added to an image, that data is no longer correct: if the fog is thick enough, a person could become completely invisible.
Therefore, when adding fog, we also created a map according to formulas 1 and 2 where we supposed that L_0 is constant everywhere and equal to 0.
So for each image, there is an associated visibility map that, if sampled inside the body of a person, can easily tell how much fog has been added on top of that person, thus revealing how much the person is visible through the fog. In order to achieve this, for each image, we iterated over the visible people before adding the fog, then, by using each person's keypoints (i.e. points that lie on specific body parts of a person) and their visibility information, we sampled from the visibility map and, if the average visibility was below a fixed threshold, the person was discarded and the network behaved as the person did not exist. This is done both during the training and the testing of the networks.

4.2 Adding artificial fog in training and testing images

In order to add artificial fog in training and testing images, a few hyperparameters have been used:

(1) A parameter that controls the probability to have uniform fog in an image (`fogProbability`)
(2) A parameter that controls the probability to have heterogeneous fog in an image (`perlinProbability`)
(3) A parameter that controls the visibility threshold of a person.
(4) Some intervals that control the range over which k (for both uniform and heterogeneous fog), the scale of the perlin noise maps s^x_k, s^y_k, $s^x_{L_s}$ and $s^y_{L_s}$ and the weights w_k and w_{L_s} (only for heterogeneous fog) can change.

5 DETECTION WITH NEURAL NETWORKS

Our preprocessing pipeline, built using classical computer vision algorithms, was integrated with two deep learning architectures, one used for the people detection task, the other for edge detection of human heads. In order to calculate the effectiveness and metrics of our pipeline, we performed several training and testing sessions to select the best hyperparameters, not only for the supervised models, but also for the algorithms used to perform the Hough transform for circles and the Canny edge detector.
The results are discussed in the related chapters.

5.1 Architectures

The training and testing supervised phases were organised using two different neural networks. For the people detection task we used a pre-trained Faster-RCNN based on the *RESNet 50 fpn* backbone. This neural network was trained and tested on the MOTSynth dataset, after an extensive preprocessing phase for the volumetric fog addition according to the FRIDA model [3]. For the edge detection task, necessary for the correct behaviour of the edge based Hough transform to detect people's heads, we created our own architecture, trained on the foggy version of MOTSynth.

5.2 Dataset

In order to train and test the neural networks created for this task, we used the *MOTSynth* dataset [2]: a huge synthetic dataset for pedestrian detection and tracking in urban scenarios created by exploiting the highly photorealistic video game Grand Theft Auto V developed by Rockstar North. A set of 768 full-HD videos, 1800

frames long, recorded at 25 fps. MOTSynth is born from the collaboration between UNIMORE and the Technical University of Munich. Although we started from the idea to make it work only for synthetic images, we wondered if it could also work on real images. Therefore, we also checked out the *Cerema pedestrian database* [8]: produced by Cerema in Fog and Rain R&D Platform. It contains 51 302 images acquired by DFW-SX700 Sony camera at 7.5 fps and with a resolution of 1024x631 (aperture = 8.6 ; focus = 30m ; zoom = 8mm). In this database, 10 pedestrians perform an identical route, which contains 6 directions, in 10 different weather conditions.
MOTSynth is a dataset composed by videos, so we built up a training dataset by performing a quantization of such videos. Due to the big dimension of MOTSynth, we tried to obtain a *time independent* dataset, by selecting non consecutive frames.

5.3 Data augmentation

During the construction and partial testings of the various people detector configurations, we also tried to feed real images, from the Cerema dataset, to the people detector network and the results were really bad. That's why we decided to add data augmentation to be used during training. We hoped that the addition of noise and other random perturbations on the training images could help the network perform better with real, non-synthetic images.

```python
import imgaug.augmenters as iaa

seq = iaa.Sequential([
  iaa.OneOf([
  iaa.GaussianBlur((0, 3.0)),
  iaa.AverageBlur(k=(1, 7)),
  iaa.MedianBlur(k=(1, 11)),
  ]),
  iaa.SomeOf((0, 5),
  [
    iaa.Sharpen(alpha=(0, 1.0), lightness=(0.75, 1.5)),
    iaa.Sometimes(0.5, iaa.imgcorruptlike.MotionBlur(
   severity=(1, 3))),
    iaa.LinearContrast((0.5, 2.0), per_channel=0.5),
    iaa.imgcorruptlike.GaussianNoise(severity=(1, 2)),
    iaa.SaltAndPepper((0.01, 0.2), per_channel=True),
    iaa.Add((-10, 10), per_channel=0.5),
    iaa.Multiply((0.5, 1.5), per_channel=0.5),
  ], random_order=True),
  ], random_order=True)
```

Listing 1: Data aug with ImgAug Sequential module

After some experiments we decided to add affine transformations, to change image format (scale out/in, 2D translations etc.), as follows:

```python
import imgaug.augmenters as iaa

# Affine transformation
img_h, img_w, _ = img_aug.shape

aug_scale = np.random.uniform(0.5, 2)
aug_h = np.random.uniform(0, 1)
aug_w = np.random.uniform(0, 1)

# convert the offset calculated for image points to
   offset useful for the imgaug library Affine
   transformation
aug_offset_h = -(aug_h - .5) * (img_h * aug_scale -
   img_h)
aug_offset_w = -(aug_w - .5) * (img_w * aug_scale -
   img_w)
affine = iaa.Affine(scale=aug_scale,
translate_px={'x': int(round(aug_offset_w)), 'y': int(
   round(aug_offset_h))})
```

Listing 2: Affine transformation with ImgAug

Unfortuately, we were not able to do any tests of the network trained with the data augmentation.

5.4 Loss functions

Our computer vision system is constituted by two main architecture, the Faster-RCNN network and our custom deep fog-resistant edge detector, so we used two different loss functions. The Faster-RCNN network [9] presents the following RPN (*Region Proposal Network*) loss:

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*) + \lambda\frac{1}{N_{reg}}\sum_i L_{reg}(t_i, t_i^*) \quad (3)$$

where

$$p_i^* = \begin{cases} 1 & \text{if IoU} > 0.7 \\ 0 & \text{if IoU} < 0.3 \\ & \text{otherwise it does not contribute to the loss} \end{cases} \quad (4)$$

Symbol	Explanation
p_i	Predicted probability of anchor i being an object.
p_i^*	Ground truth label (binary) of whether anchor i is an object.
t_i	Predicted four parameterized coordinates.
t_i^*	Ground truth coordinates.
N_{cls}	Normalization term, set to be mini-batch size.
N_{reg}	Normalization term, set to the number of anchor locations.
λ	A balancing term, L_{cls} and L_{box} are equally weighted.

The remaining part, the final *multi-task loss*, is similar to the Fast-RCNN one [10], as summation of the *Log loss* and the *Smooth L1 loss.*

The loss function selected for our custom deep edge detector is a *binary cross entropy* (*Logg loss*) as follows:

$$Logg_{loss} = -\frac{1}{N}\sum_i y_i * log(p_i) + (1-y_i) * log(1-p_i) \quad (5)$$

in a first moment we chose the MSE, but due to the very unrepresentative values we decided to change it into a BCE, in order to classify a pixel as a border or not.

5.5 People detection

We decided to use a very famous and efficient network to perform this task, and during its training we had some difficulties in the network parameters choice. Especially for what concerned the visibility threshold (to take or not a person partially or completely occluded by fog) and the learning rate. In conclusion, with the usage of a lower learning rate the results may be better than the ones presented here.

5.6 Deep Edge detector

The deep edge detector architecture, a feature of our project, is organised as follows:

```
self.conv1 = nn.Conv2d(in_channels=3,out_channels=64,
   kernel_size=(3,3), stride=1, padding=(1,1),
   padding_mode='replicate')

self.conv2 = nn.Conv2d(in_channels=64,out_channels=128,
    kernel_size=(3,3), stride=1, padding=(1,1),
   padding_mode='replicate')

self.conv3 = nn.Conv2d(in_channels=128,out_channels
   =256, kernel_size=(3,3), stride=1, padding=(1,1),
   padding_mode='replicate')

self.conv4 = nn.Conv2d(in_channels=256,out_channels
   =256, kernel_size=(3,3), stride=1, padding=(1,1),
   padding_mode='replicate')

self.conv5 = nn.Conv2d(in_channels=256,out_channels
   =256, kernel_size=(3,3), stride=1, padding=(1,1),
   padding_mode='replicate')

self.conv6 = nn.Conv2d(in_channels=256,out_channels=1,
   kernel_size=(1,1), stride=1, padding=0)

out = self.conv1(x)
out = F.relu(out)
out = self.conv2(out)
out = F.relu(out)
out = self.conv3(out)
out = F.relu(out)
out = self.conv4(out)
out = F.relu(out)
out = self.conv5(out)
out = F.relu(out)
out = self.conv6(out)
out = F.sigmoid(out)
```

Listing 3: Custom deep edge detector

This network has been trained only on the first quarter of the bouding box of a person (i.e. the box is split horizontally into 4 parts and only the topmost one is analysed).

The convolutive layers were designed such that:

(1) The input size is preserved throughout the entire forward pass
(2) The padding is less invasive as possible regarding the gradient: as the simple method for edge detection is computing the image gradient, by replicating nearby pixels we think we can help the network avoiding to misclassify pixels near the borders due to discontinuities produced by padding.

At the end, a sigmoid brought the output of the network in the interval $[0, 1]$, so that we could use the Binary Cross Entropy loss in order to train it.

5.6.1 Training of the deep edge detector. As for the training part of the deep edge detector, we selected a sample of 175 videos and we used 4 images per video. We converted each image into the YCbCr colorspace in order to have the luminance component of the image ready to be used. Then, for each image, we generated 7 copies of it, each convoluted with a gaussian kernel with $\sigma^2 \in \{0, 0.5, 1, 1.5, 2, 2.5, 3\}$ and appropriate kernel size for each σ^2 ($k_s = 5\sigma^2$). After that, for each visible person of each copy of the image, we cut the first quarter of the bounding box (we split the bounding box horizontally by dividing its height for 4.01) to make sure that the only circular part that could possibly be detected is the head. Then, we used the Canny edge detector on the luminance channel of all the people in the image, with different lower and upper thresholds ($T_L = \{20, 40, 60, 80, 100, 120, 140, 160\}$,
$T_H = \{40, 80, 120, 160, 200, 240, 280, 320\}$) by doing each possible combinations of values in T_L and T_H ($T = \{(l, h) | l \in T_L, h \in T_H, l < h\}$). Finally, we computed the Hough transform for circles to find out the location of the head.

The best detection of the head was kept according to the following metric. We computed a predicted head by using both the keypoints and segmentation annotations provided with the dataset. This metric was necessary due to the lack of head segmentation inside the dataset. In particular, the following steps were performed:

```
Find the line passing between the two keypoints of the
    head
By following the line along the axis where it grows
    slower, arrive to the boundaries of the segmentation
     map: we found the topmost point of the head (T)
Calculate the head diameter as 2 * d(kt, T) + d(kt, kh)
    where T is the topmost point of the head, kt and kh
    are the two keypoints of the head (kt is the upper
    one)
Calculate the center of the head as the mean point
    between kt and kh.
Then calculate the Manhattan distance between the center
    of the head and the center calculated by the Hough
    transform (md)
Calculate the absolute difference between the radiuses (
    rd)
The result is -(md+rd)
```

Listing 4: Finding the head from the annotated ground truth and measure its deviation between the detected head

Only the edges that maximise the metric above were saved and used as training data. Together with the images, we also saved the video and frame number they were extracted from, the bounding box of the person and the parameters used for the Canny edge detector.

With this method, we were able to generate approximately 4000 different training images.

The deep edge detector has been trained with (a maximum) of 1000 images. For each image from the previously generated training set, we loaded the original image and added random fog (see 7.2 for the exact numerical values). Then, all people whose visibility was below a certain threshold were discarded. The network was then fed with the input image cropped at one quarter of the height, as previously said, and the ground truth was the edges map computed with the aforementioned method.

The loss used is a simple binary cross entropy (see equation 5). The optimizer used is a simple stochastic gradient descent, with an initial learning rate at 0.005 which is divided by 10 every 4 epochs. The total training consisted in 10 epochs.

At the end, the output of the network was a real image which values were in the interval $[0, 1]$; in order to find the real edges, we simply applied a threshold equal to 0.5.

6 HEAD DETECTION

Once an image has been fed into the people detector network, each first quarter of the bounding boxes was fed into our deep edge detector. Then a fixed threshold equal to 0.5 was applied to results of the deep edge detector. After that, the Hough transform algorithm for circles was run and we looked for a circle which radius was in the interval $[h/5, h]$ where h is the height of the topmost quarter of the bounding box. Finally, the maximum point in the Hough space was selected, which corresponds to the most probable head: the output is a triple (r, c_x, c_y) where r is the radius of the detected circle and (cx, cy) its center (relative to the bounding box of the current person).

7 EXPERIMENTS RESULTS

7.1 Hyperparameter optimization

We conducted the first experiments using only a subset of the MOT-Synth dataset already mentioned, in order to speed up the various training processes and be able to do lots of tests. Each one was slightly modified with respect to the previous one in order to see changes in the results based on the variations applied and consequently be able to choose the best parameters for the network. In particular, we fixed the random seed in order to be able to replicate or have comparable results among the different attempts and we focused on these hyperparameters:

(1) *lr*: the **learning rate**. Tests with lr = 0.0005 and lr = 0.00005 have been conducted and the second one was chosen as the final Average Precision in output was 67.03% in the first case and 68.85% in the second one.
(2) *visibility_cutoff_threshold*: it's used to select, above the people in an image, only the bounding boxes surrounding people whose **visibility** is over this **threshold** (the green ones shown in the various images). Two values have been tested: 0.027 and 0.06 and the greater one showed an increase in the Average Precision of about 20%.
(3) *data augmentation*: a flag **dataAug** has been used to perform data augmentation on images in order to prevent the network form learning irrelevant patterns It was at first introduced to work on the Cerema dataset but for what concerns the synthetic dataset, as we expected, performances didn't show an improvement since the images were already good enough to make the network generalize. So this parameter has been set to 0 in this case.
(4) Another aspect that has been considered but discarded later, as it did not bring any significant contributions to the results, is the balance of the total number of people in the foreground and in the background. The reasoning behind this was that we noticed that the network failed to detect some easy foreground instances; however, this showed no significant improvements over the results without it, hence it had been dropped.

In the following graphs it's possible to notice the trainings different behaviours in the case of *visibility cut-off threshold* equals to 0.027 (black line) and the one fixed to 0.06 (light blue line).

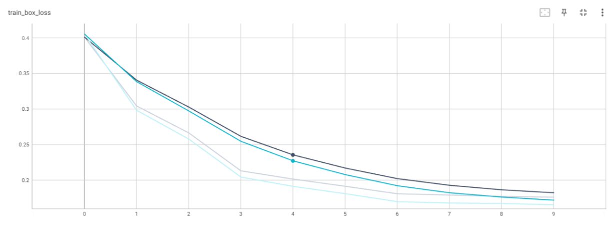


Figure 2: TensorBoard view that shows multi-task loss values

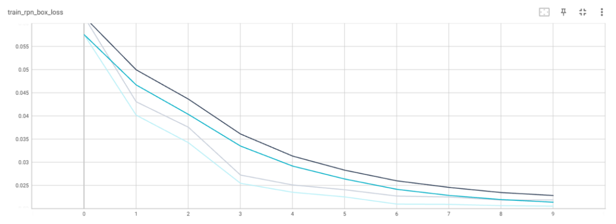


Figure 3: TensorBoard view that shows region proposal network loss values

7.2 Training

Once all the parameters of our network were fine-tuned, we did two main training on the dataset. We conducted one training adding syntetic fog as described in [4] and another one with the original images.

Both times, 5000 images were taken from 500 different videos to be used as the *training set*, and another 1500 from the same videos were used as the *validation set*. The training consists of 10 epochs and the learning rate decreases by 10x every 3 epochs. For each training phase, results of the evaluations have been saved in order to be able to test the two models later on never seen data.

The parameters for the fog generation for the training are (see section 4.2 for descriptions of these parameters):

(1) A probability to have fog in an image (`fogProbability`) equal to 0.75 for the model with added fog and 0 for the model trained without added fog.
(2) A probability to have heterogeneous fog equal to 0.35.
(3) A k uniformly sampled in the interval $[0.05, 0.3]$.
(4) Two scale factors for the perlin noise associated to k (s_k^x and s_k^y) uniformly sampled in the interval $[1, 3]$.
(5) A weight for the perlin noise associated to k (w_k) in the interval $[0.2, 0.7]$.
(6) Two scale factors for the perlin noise associated to L_s ($s_{L_s}^x$ and $s_{L_s}^y$) uniformly sampled in the interval $[1, 3]$.
(7) A weight for the perlin noise associated to L_s (w_{L_s}) in the interval $[0.02, 0.1]$.

7.3 Testing

Both models have been tested on unseen data taken from $\tilde{9}00$ images from 100 different videos, each one with fog added and same seed fixed, in order to compare results on the same exact images. In these images we added artificial fog with a probability of 1 using the parameters described in section 7.2.

Model type	IoU	Area	AP	AR
Fog model	0.5	all	75.26%	37.53%
No-Fog model	0.5	all	72.60%	37.64%

As it is possible to see from the table 7.3 and also from some significant images we reported, the model trained with foggy images and the one with the original images show very similar results both in terms of AP percentages that in the general behaviour. In particular, seems that out model (the fog-resistant one) is slightly more sensitive than the other one to false positive, but, even though only in a few cases, it is also more accurate in detections concerning people that are not very visible, as we can see from the red boxes in the background (where the visibility is under the threshold chosen).

7.3.1 Testing for the deep edge detector. The deep edge detector is tested by using different metrics because we needed to infer the real location of the head, given that the ground truth annotations we have do not contain such precise information. In particular, we computed:

(1) A metric very similar to the one used for creating the training set for the deep edge detector (see 5.6.1), but the head diameter is calculated neglecting the distance between the two head keypoints.
(2) A classical intersection over union between the detected head and the predicted head
(3) $\frac{|(S \cup P_H \cup D_H) \setminus (S \cup P_H)|}{|D_H|}$ where S is a person's segmentation map, P_H is the person's predicted head and D_H the person's detected head. The idea behind this is that, if the head is all inside the segmentation map or the predicted then it's well predicted. Of course, this is not automatically true, but, if together with this condition we also have that the two head keypoints fall inside D_H, then we can have a good hint that the head has been correctly detected.
(4) We also detect the head of a person on its segmentation map preprocessed with a Laplacian filter to find the edges, then we take the Manhattan distance between the center found in this way and the absolute difference of the radiuses.

We combined the results taken from the metrics above and used them to evalutate the performance of the deep edge detector.
A standalone test for this component is run on 51 images from 51 different videos, obtaining 352 different head detections on the same number of correctly detected people (i.e. people that are really in the image, not false positives). For each people detected with the people detector network, both our deep edge detector and a Canny edge detector (with a low threshold equal to 20 and a high threshold equal to 40, which are the most successful Canny parameters found during the costruction of the training set for the deep edge detector) are applied on the first quarter of the bounding box.
In order to evaluate the performance of the edge detectors, we consider that the head cannot be detected on a certain person if the circle detected with the Hough transform on the segmentation map does not contain any head keypoints. Given that, we can say that the head cannot be detected in 118 cases and can be detected in 232 cases. As it is necessary to associate a ground truth with each person detected to be able to compute the metrics on the detected head, we used the person whose ground truth bounding box has the maximum IoU with the detected bounding box for each person. This approach, however, is not guaranteed to find the right keypoints and segmentation maps; in fact, in 2 cases, the method fails to do so and therefore, these are ignored.
In all cases in which the head can be detected, the Manhattan distance between the center of the detected head on the segmentation map and the detected head on the edges computed with our deep edge detector or the aforementioned Canny edge detector is computed, together with the absolute difference of radiuses.
The results are summarized in the following table (μ_c is the average Manhattan distance between centers, μ_r is the average absolute difference between radiuses and μ_{IoU} the average intersection over union between the predicted head from given annotations by using the method described in 5.6.1 but neglecting the distance between the two head keypoints and the detected head with the Hough transform). The average precision is calculated by considering a true positive every instance where the head can be detected (from the segmentation map, as previously described) and at least one head keypoint is inside the detected head; a false positive, instead, is an instance where the head can be detected (from the segmentation map) but the Hough transform fails on the edges calculated by one of the two methods. Any other case where the head cannot be detected on the segmentation map is ignored in this test.

	μ_c	μ_r	μ_{IoU}	AP
Our d.E.d.	8.29	1.15	0.34	78.9%
Canny	8.06	1.61	0.33	79.0%

7.3.2 Test for the hough transform for detecting heads. In order to measure the accuracy of the Hough transform for detecting people's heads, we simply tried to use it on the Laplacian of the first quarter of the segmentation map. As previously stated, the zeros of the Laplacian of the segmentation map allow us to easily identify edges in this map because of the absence of noise in the segmentation map.
Due to the absence of proper annotations, we consider the head to be correctly detected (true positive) if there is at least one keypoint of the head inside the detected head, otherwise it's a false positive. Note that there are no negatives in this case as we make the assumption that each person in the ground truth has a head. With this approach, over the 352 correctly detected people:

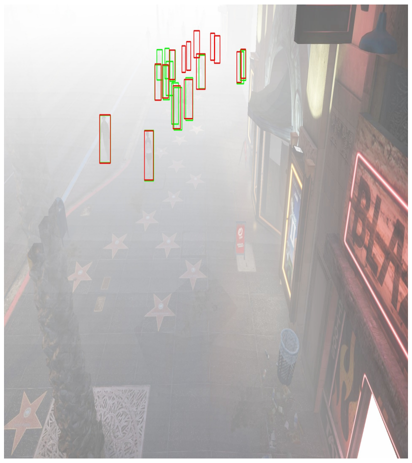
- for 232 cases the head was detectable by the Hough transform
- for 118 cases the head was not detectable by the Hough transform
- for 2 cases we were not able to correctly associate the ground truth to the detected person

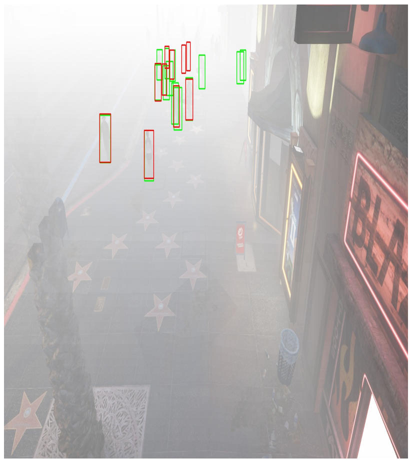
The average precision for this head detection is 65%.

8 FINAL CONSIDERATIONS

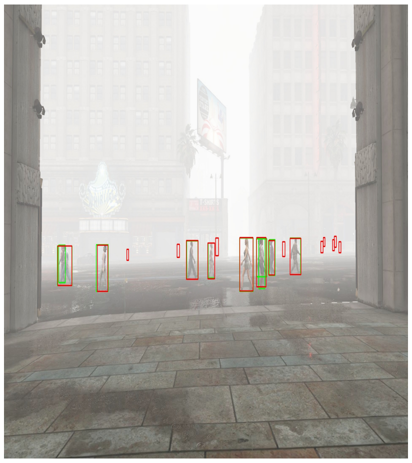
With this project we had the possibility to study and implement a pipeline for a classical computer vision problem, with the combination of image processing and deep learning. We achieved acceptable results from our fog-resistant people detector; we think that, because of the availability of a lot of training data, it should be possible to continue the training and do more fine-tuning of the networks

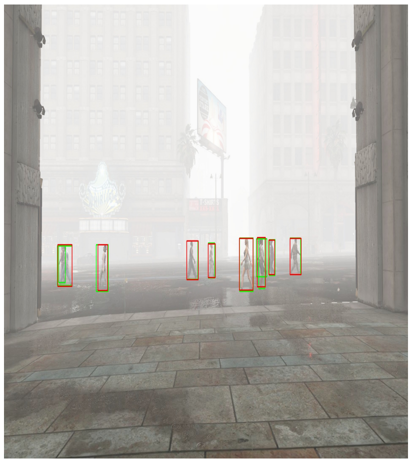
(a) Ground truth

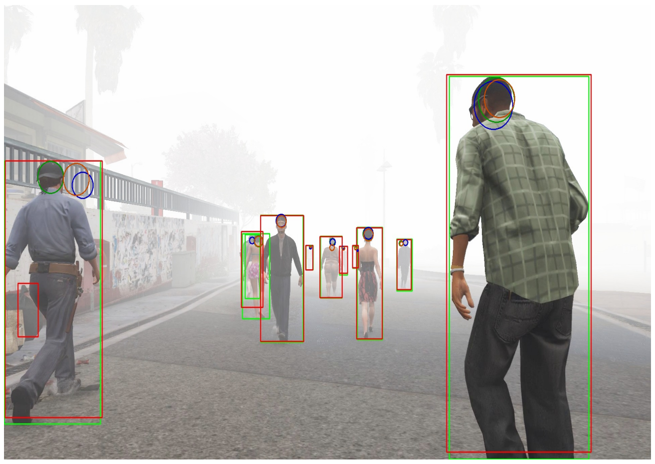

(b) Fog model

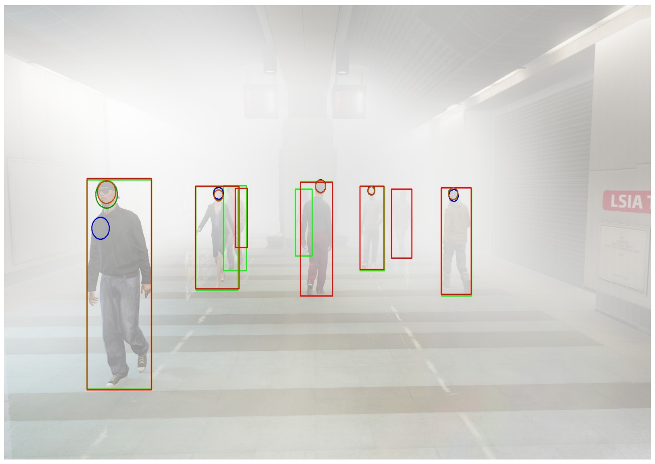

(c) No fog model

(a) Ground truth


(b) Fog model


(c) No fog model





hyperparameters to achieve better results. Nonetheless, it is clear that fog does not represent an unsurmountable obstacle even for a fog-unaware model. These results also extend to the deep edge detector model.

We tried to modify the algorithm parameters to obtain better performances for both hand made processing (fog, Hough transform etc.) and neural network models and we noticed that one of the most difficult problem is represented by the huge number of false positive predictions when volumetric and real fog are more visible independently by the presence of more or less environment light. So, in order to solve this problem we decided to increase the visibility threshold in order to effectively take into account, during the training process, only real visible people, despite the detection in a very difficult visibility situation.

We found the interaction with large datasets very interesting, as well as the use of different data formats. Given the heterogeneity of information at our disposal, especially for the MOTSynth dataset, we used the segmentation information and keypoints for evaluating head detection results.
Given that our objective was the training of a fog resistant people and head detector, we are globally satisfied with the results, although we hoped to reach a more significant difference in the performances. The Faster-RCNN trained on fog data does not seem to perform considerably better than the same architecture trained on fog-free data, but a quick visual comparison shows that our model is, in some cases, better at detecting people over long distances and more occluded by haze. The discrepancy between these considerations and the results achieved is dictated by a high number of false positives, which degrades the precision.

REFERENCES

[1] MIT Media Lab. Seeing through realistic fog — mit media lab. URL https://www.media.mit.edu/projects/seeing-through-fog/overview/#faq-what-are-the-main-advantages-of-this-method. (Accessed on 04/06/2021).
[2] Matteo Fabbri, Guillem Brasó, Gianluca Maugeri, Aljoša Ošep, Riccardo Gasparini, Orcun Cetintas, Simone Calderara, Laura Leal-Taixé, and Rita Cucchiara. Motsynth: How can synthetic data help pedestrian detection and tracking?, 2021.
[3] J.-P. Tarel, N. Hautière, A. Cord, D. Gruyer, and H. Halmaoui. Improved visibility of road scene images under heterogeneous fog. In *Proceedings of IEEE Intelligent Vehicle Symposium (IV'2010)*, pages 478–485, San Diego, California, USA, 2010. http://perso.lcpc.fr/tarel.jean-philippe/publis/iv10.html.
[4] Terry Rickard. Seeing through the fog. https://blog.renewintl.org/seeing-through-the-fog. (Accessed on 04/06/2021).
[5] Jean-Philippe Tarel, Aurélien Cord, Houssam Halmaoui, Dominique Gruyer, and Nicolas Hautière. Frida (foggy road image database) image database. http://perso.lcpc.fr/tarel.jean-philippe/bdd/frida.html. (Accessed on 04/06/2021).
[6] Guy Satat. Overview ‹ seeing through realistic fog — mit media lab. https://www.media.mit.edu/projects/seeing-through-fog/overview/#faq-why-is-visible-light-essential-for-imaging-through-fog-why-not-just-radar, 04 2019. (Accessed on 09/26/2021).
[7] Ken Perlin. An image synthesiser. *SIGGRAPH Computer Graphic*, 19(3), 1985.
[8] Khouloud Dahmane, Najoua ESSOUKRI BEN AMARA, Pierre Duthon, Frédéric Bernardin, Michèle Colomb, and Frederic Chausse. The cerema pedestrian database: A specific database in adverse weather conditions to evaluate computer vision pedestrian detectors. pages 472–477, 07 2017. doi: 10.1109/DT.2017.8012150.
[9] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2016.
[10] Ross Girshick. Fast r-cnn, 2015.