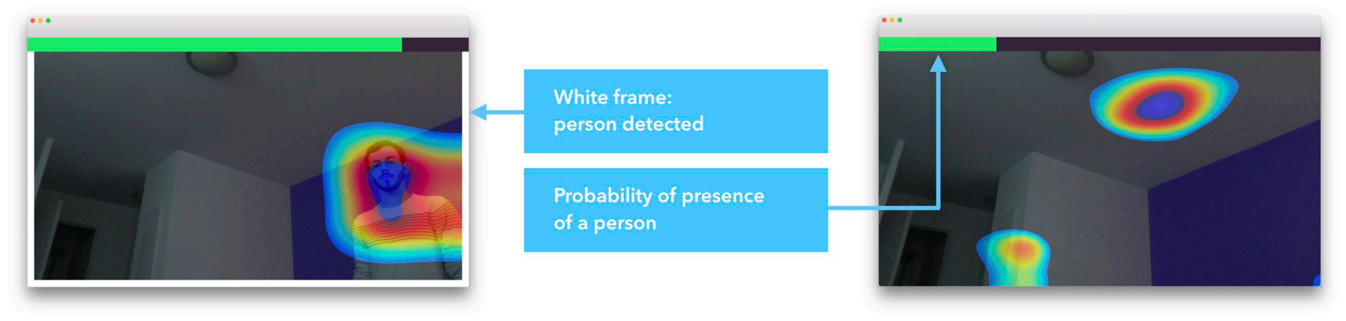
Live Person localisation



```
python3 webcam_cam.py --model ./saved_model/mobilenet.h5
```

> **Requires:**
> Tensorflow version >= 1.7
> Keras version >= 2.1
> OpenCV version >= 3.3

I. Definition

Project Overview

The capability to automatically detect the presence of humans in a video has multiple applications: surveillance cameras, human-machine interaction, pedestrian detection, automatic focus in digital cameras, video compression in videoconference software where the quality of the image can be reduced outside the bounding boxes etc.

Problem Statement

The aim of this project is to create an algorithm capable of detecting the presence of individuals in a video and to localise them. In all the applications listed above, the method used should be fast enough to process a live video.

Metrics

The algorithm will be evaluated according to metrics:

- The accuracy of the detections
- The computation time of a detection

II. Analysis

Data Exploration

The data used to train the classifier comes from the INRIA person dataset.

This dataset contains images with persons (positive) and without persons (negative). Originally, this dataset is unbalanced: there are 1669 negatives and 900 positives which means that a classifier can reach an accuracy of 70% by always predicting negatives. The excess of negatives is therefore not loaded.

Visualization

Most of the photos in this dataset are outdoors urban photos and the photos containing people are never close-ups. We can already expect a model trained on this dataset to be limited because of the lack of variance.

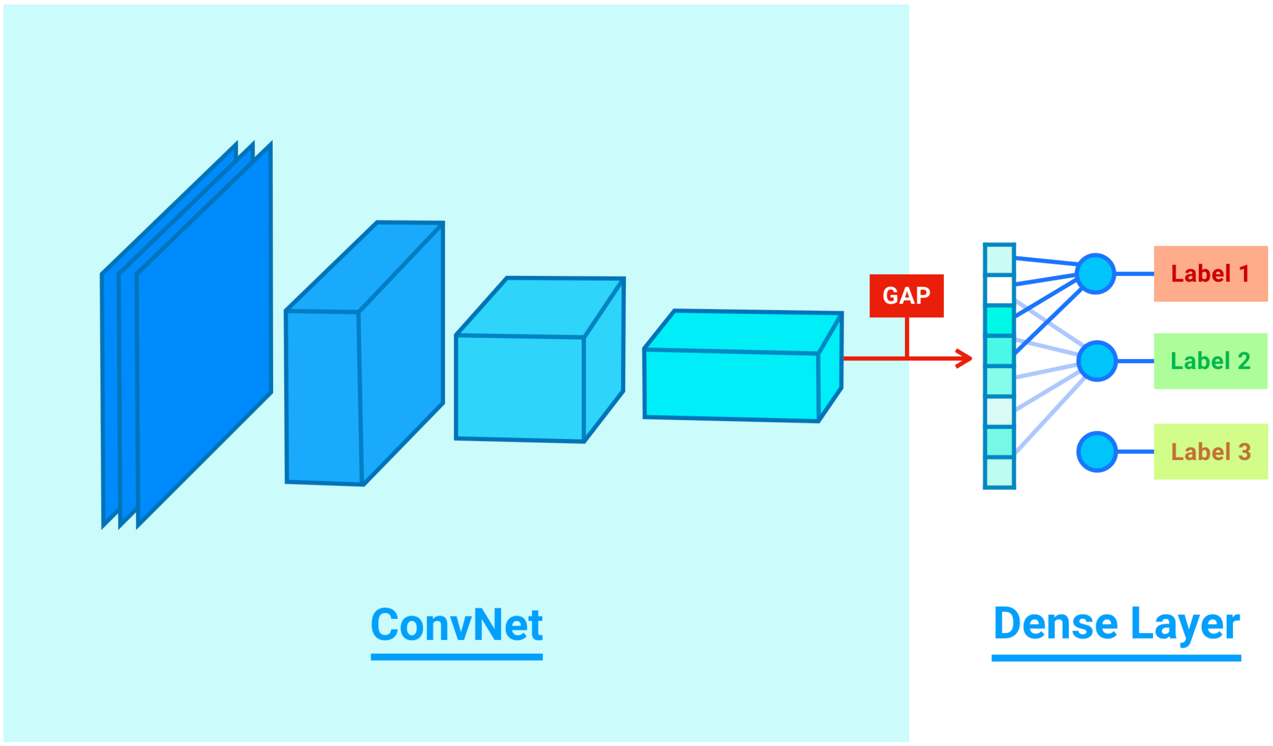
Algorithms and Techniques

The method used to solve the detection and localisation problem is described in the following paper: Learning Deep Features for Discriminative Localization.

Here is a short explanation of this method:

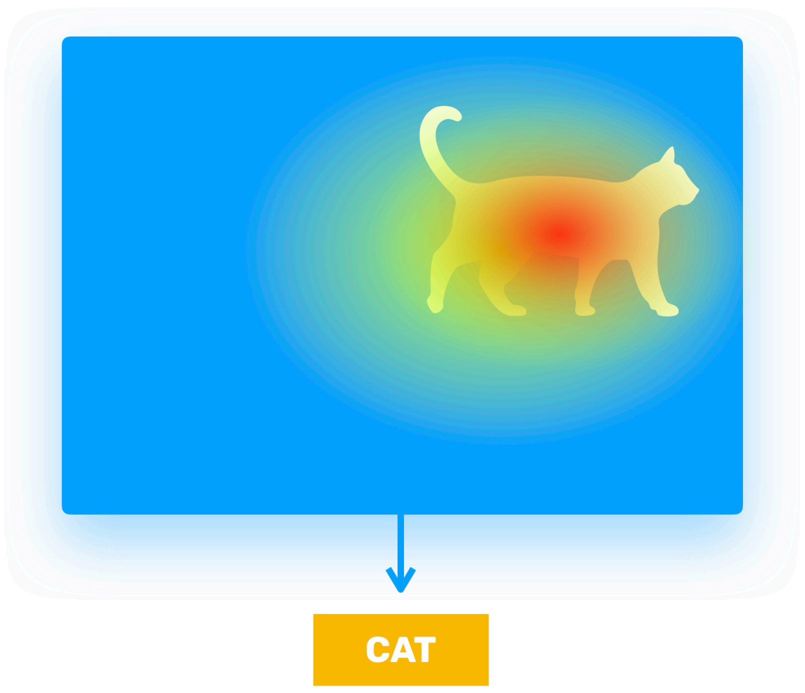
According to the authors of the papers the following convolutional neural, when trained only as a classifier, can be used to localise the classes it classifies. It, therefore, doesn't need bounding box during its training but only labels of the different classes.

The architecture of this neural network is the following:
A convolutional neural network where the last layer is flattened by a Global Average Pooling layer (GAP). Followed by a 1 layer dense network containing as many neurones as there are classes.

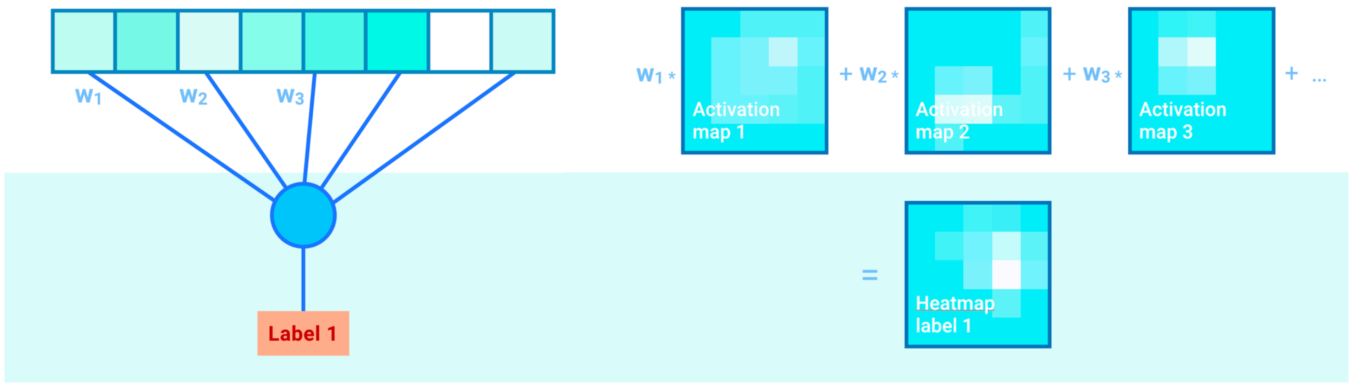


In our case there are 2 classes, therefore 2 neurones in the last layer.

To extract the localisation of the detected classes, the authors propose to extract the Class Activation Maps (CAM). These maps are heat-maps where the hot regions are the regions used by the ConvNet to evaluate a class probability given the input image.



The CAMs can be computed using the activation maps of the last convolutional layer (just before the GAP layer). Because each neurone corresponds to a class, it is possible to weigh each there activation maps with the weights of the neurones.



Intuition: If the weight w3 of the neurone associated to the label 1 is large, this means that the 3rd pixel (obtained by computing the GAP of the 3rd activation map) is activated in the presence of an object of class 1. Since the activation maps contain spatial data, each pixel corresponds to a region of the input image.

Benchmark

For the localisation to work correctly, the classifier should be very accurate. We want the accuracy to be above 90%.

The detection should be fast enough to process a live video. For the output detection video to be smooth we need it to be able to process at least 5 images per second.

III. Methodology

Data Preprocessing

Loading the data

- ... transforms images into a 4D tensor of shape (samples_size, 224, 224, 3) suitable for supplying to a Keras CNN
- ... transforms class labels into one hot encoded labels.
- ... creates a validation folder using 20% of the original training data (The INRIA person dataset originally doesn't contain validation data)
- ... balances the data.

Implementation

Development environment

Training neural nets on a laptop is a lost cause. One training epoch on a 50 layers neural network (ResNet 50) with data augmentation takes more than 45 minutes.

The following Google Cloud compute engine configuration was used for this project:

- 6 CPUs, 32 GB memory
- GPU NVIDIA Tesla K80
- CUDA toolkit and cuDNN installed to ensure that the GPU is used for the computation

With this architecture, a training epoch can be completed in 3 min.

The algorithm was implemented using Python 3 and the Keras and Tensorflow libraries.

Transfert learning

In 2017, the UC Berkeley team composed of Yang You, Zhao Zhang, James Demmel, Kurt Keutzer boasted:

> “We finish the 100-epoch ImageNet training with AlexNet in 24 minutes, which is the world record. Same as Facebook’s result, we finish the 90-epoch ImageNet training with ResNet-50 in one hour. However, our hardware budget is only 1.2 million USD, which is 3.4 times lower than Facebook’s 4.1 million USD.”

Training a neural network from scratch is therefore not accessible to individuals. A solution is to reuse a pre-trained network.

In this project, we used nets pre-trained on ImageNet with the following configuration:

- Convolutional layers of the pre-trained network followed by a GAP layer and a dense layer
- 2 last convolutional layers unfrozen because the dataset is small
- ReLU activation functions in the hidden layers to avoid the 'vanishing gradient'
- Optimisation algorithm: adam (step: 0.001)
- L2 regularisation (0.1) to avoid overfitting
- Batch-Normalisation to avoid overfitting

Training configuration

- Batch-size: 32
- Data augmentation to avoid overfitting

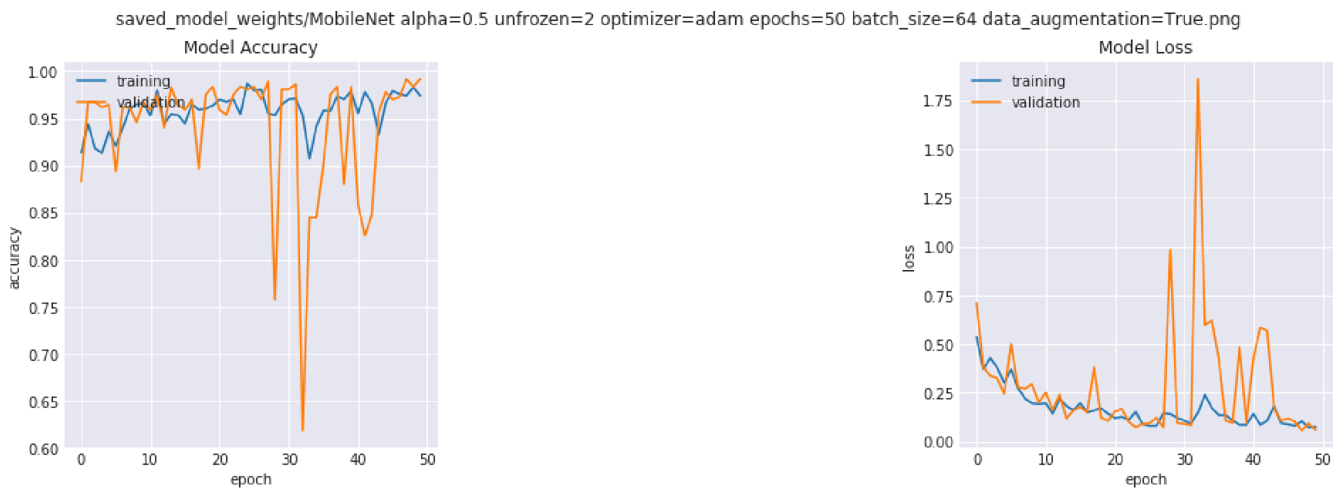
Refinement

The pre-trained model used in this project is MobilNet. This architecture was introduced in 2017, therefore after the Learning Deep Features for Discriminative Localization paper (2015) and is more performant (similar accuracy but smaller computation cost) than the architectures proposed in the paper.

IV. Results

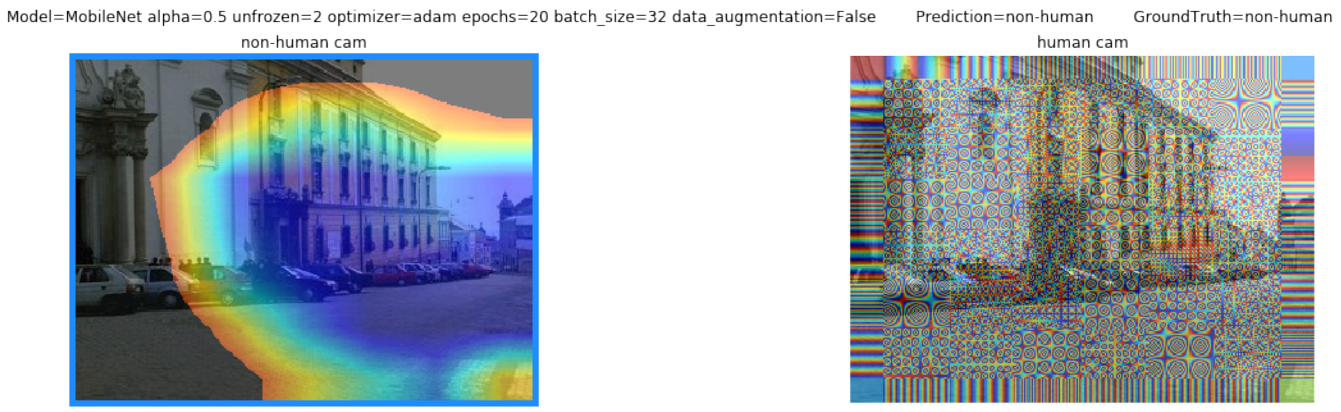
Model Evaluation and Validation

In less than 50 epochs, the MobilNet gives an **accuracy of 97.3282% and a loss of 5.20%** (on the test set). Here are the learning curves:



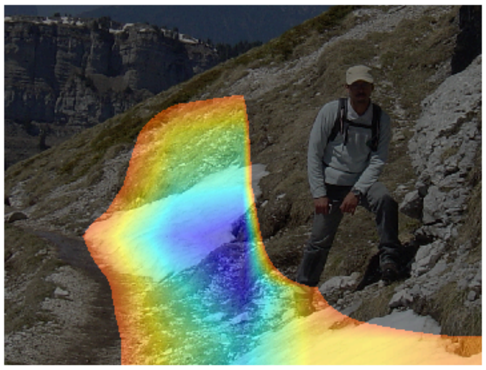
More interesting, the creation of a CAM from an input image (prediction of the class + generation of the heat-map) takes **0.31 second on a laptop**.

Here are some example outputs. The detected class highlighted with a blue frame:

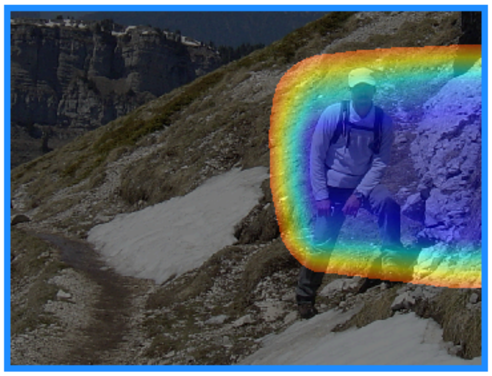

Model=MobileNet alpha=0.5 unfrozen=2 optimizer=adam epochs=20 batch_size=32 data_augmentation=False
Prediction=non-human
GroundTruth=non-human
non-human cam
human cam

Model=MobileNet alpha=0.5 unfrozen=2 optimizer=adam epochs=20 batch_size=32 data_augmentation=False Prediction=human GroundTruth=human

non-human cam

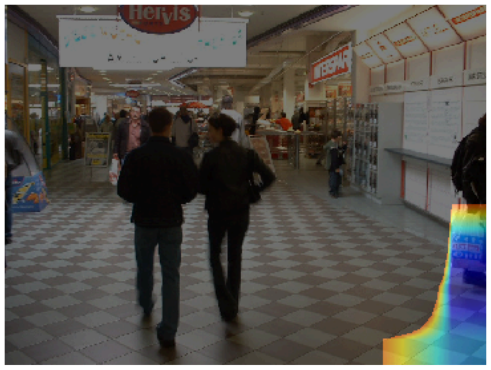


human cam

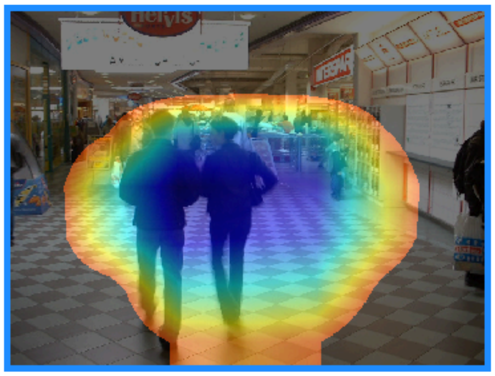


Model=MobileNet alpha=0.5 unfrozen=2 optimizer=adam epochs=20 batch_size=32 data_augmentation=False Prediction=human GroundTruth=human

non-human cam

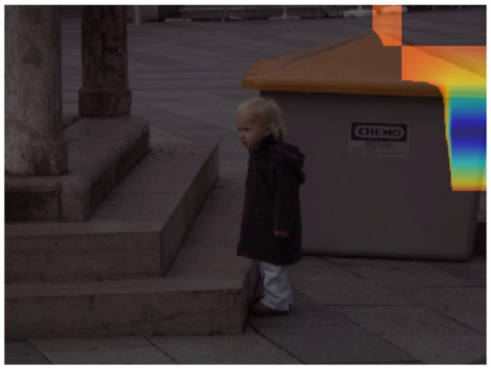


human cam

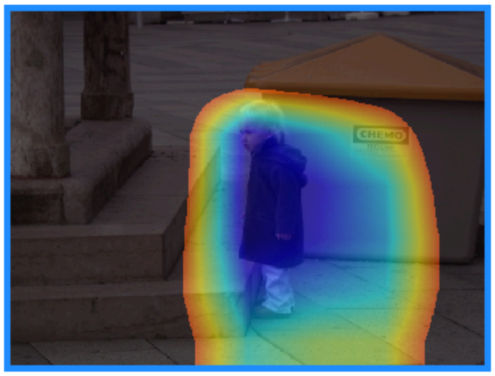


Model=MobileNet alpha=0.5 unfrozen=2 optimizer=adam epochs=20 batch_size=32 data_augmentation=False Prediction=human GroundTruth=human

non-human cam

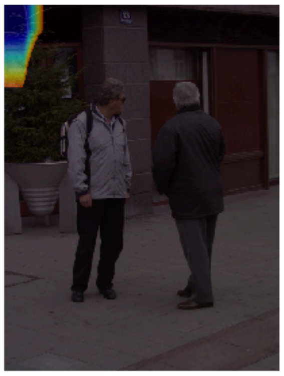


human cam

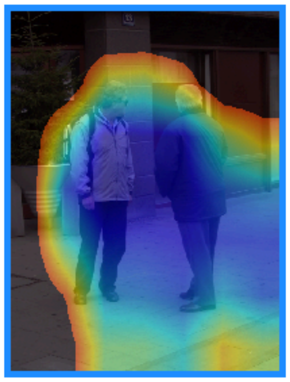


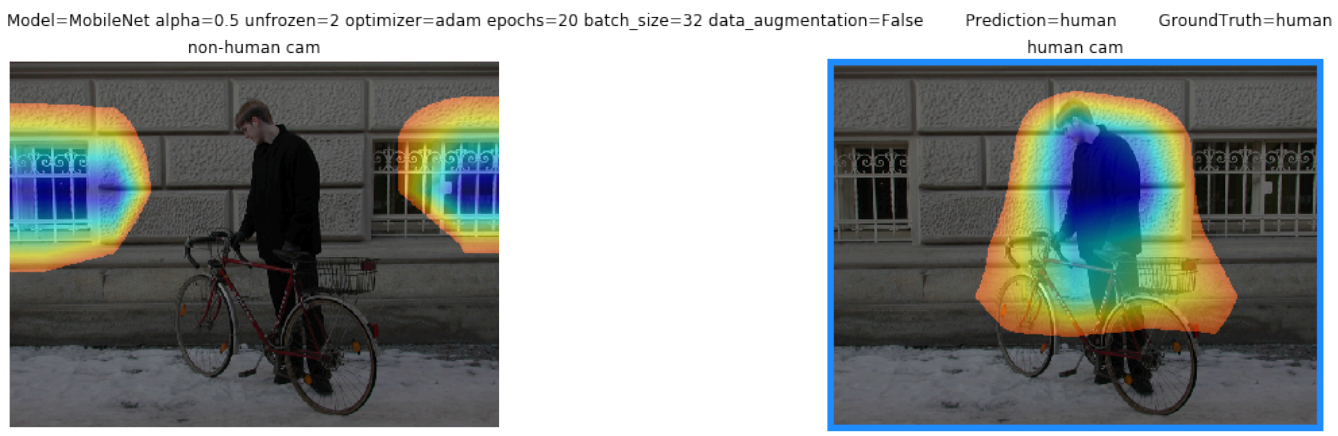
Model=MobileNet alpha=0.5 unfrozen=2 optimizer=adam epochs=20 batch_size=32 data_augmentation=False Prediction=human GroundTruth=human

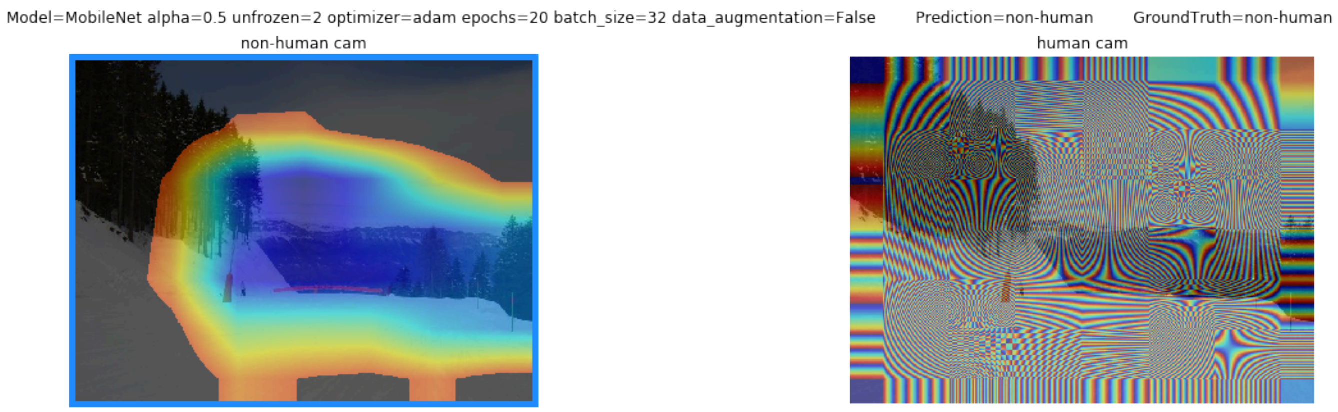
non-human cam



human cam




Model=MobileNet alpha=0.5 unfrozen=2 optimizer=adam epochs=20 batch_size=32 data_augmentation=False Prediction=human GroundTruth=human


Model=MobileNet alpha=0.5 unfrozen=2 optimizer=adam epochs=20 batch_size=32 data_augmentation=False Prediction=non-human GroundTruth=non-human

Observation:

- When no human is detected, the 'negative' heat-map highlights lines, buildings, grids etc.
- When no human is detected, the 'positive' heat-map doesn't highlight anything in particular
- Groups of humans aren't separated by the algorithm

Justification

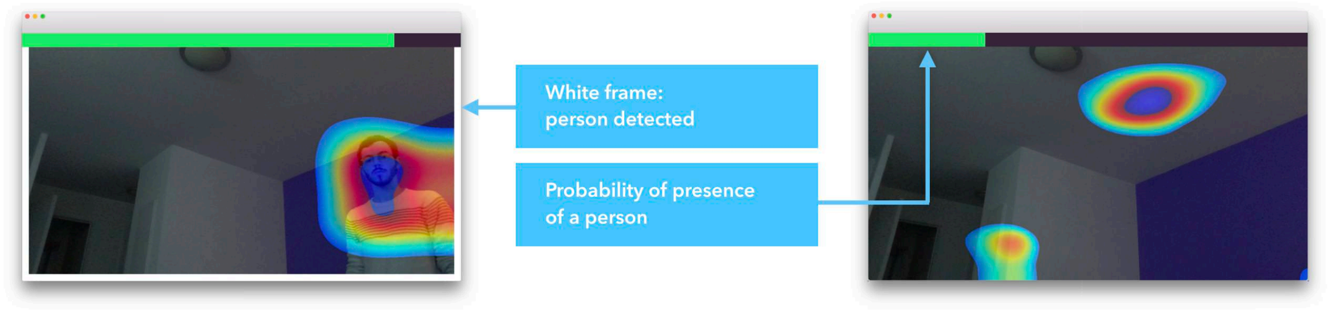
The final accuracy and computation time results found stronger than the benchmark result.

The localisation results are satisfying for the method proposed in the paper and for the size of the dataset but there is one limitation to this method: the network doesn't separate individuals when they are in a group, close to each other.

V. Conclusion

Visualization

Here are some screenshots of an application of the method implemented in this project (this application can be launched from with the `webcam.py` script -- see `python3 webcam.py --help`).



Reflection

The classification and computing time are very good. However, the localisation results are limited. This results should be put into perspective: The authors of the paper used at least tens of thousands of images per class, when the model presented above only uses 855 images in total in the training set. The results are quite satisfying for such a small data set (thanks to data augmentation in particular).

Improvement

The 2 principal limitations of the solution presented above are the following:

- The resolution of the localisation is limited by the size of the last activation map.
- The training set used in the project is very small, and it is difficult to find a large, balanced dataset corresponding exactly to the problem.

Based on these limitations, the possible improvements are the following:

- Using sense encoder-decoder networks as described in the following paper Fully Convolutional Networks for Semantic Segmentation
- Using a larger database like Google Open Dataset