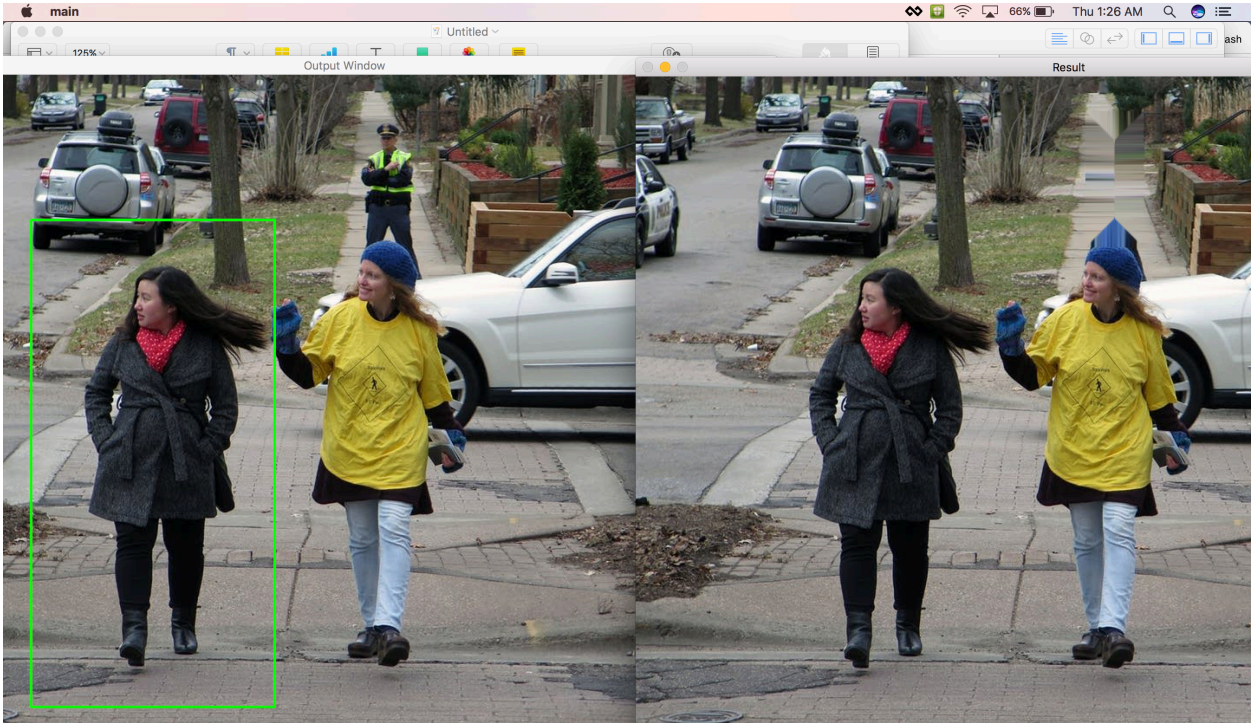
Computer Vision Challenge

Interactively remove people from images



Shreyash Pandey

10/31/2017

Haar Cascades

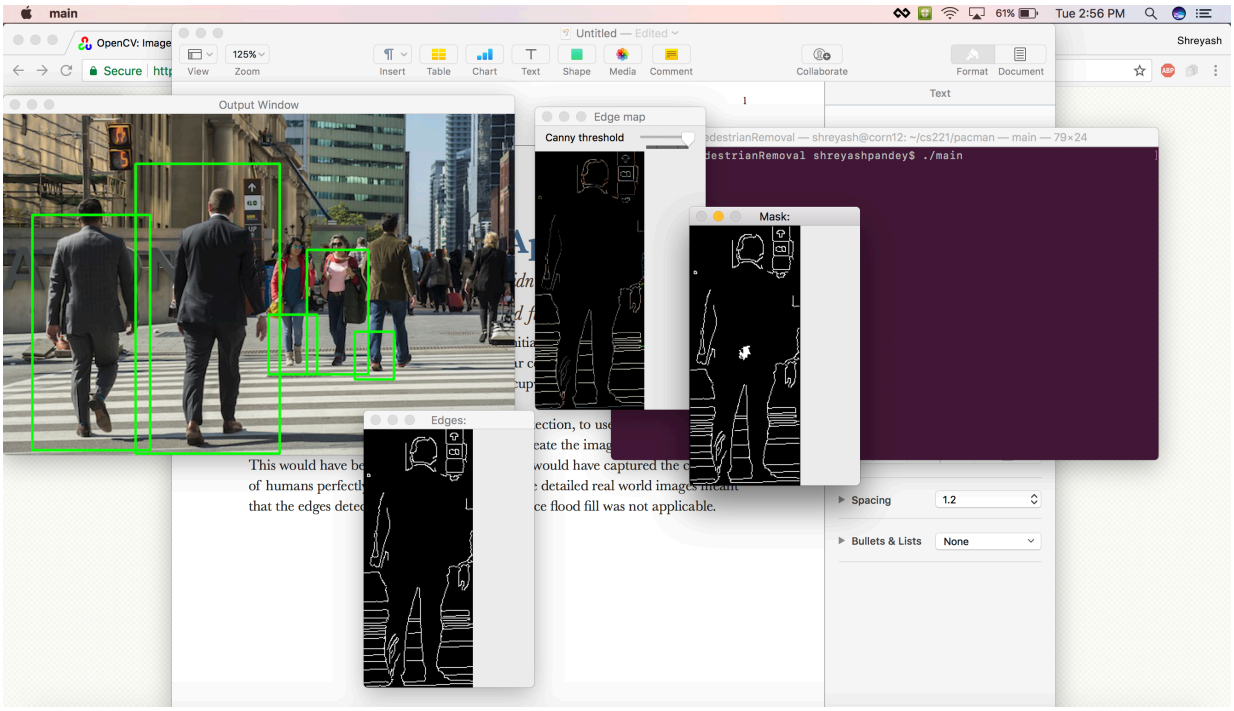
A Haar-like feature considers adjacent rectangular regions at a specific location in a detection window, sums up the pixel intensities in each region and calculates the difference between these sums. The difference is then used to categorize subsections of an image. In the detection phase of the Viola–Jones object detection framework, a window of the target size is moved over the input image, and for each subsection of the image the Haar-like feature is calculated. This difference is then compared to a learned threshold that separates non-objects from objects. Because such a Haar-like feature is only a weak learner or classifier (its detection quality is slightly better than random guessing) a large number of Haar-like features are necessary to describe an object with sufficient accuracy. In the Viola–Jones object detection framework, the Haar-like features are therefore organized in something called a *classifier cascade* to form a strong learner or classifier.

A lot work has been done in the pedestrian detection area using Haar cascades. OpenCV has a library (objdetect.hpp) that performs inference provided we are given the Haar cascade architecture. There are some open-source pre-trained classifiers that detect upper-body, lower-body and full-body of the pedestrians. I implemented a Non-maximum Suppression on top of these detections to obtain the best predictions out of the lot. These pedestrian detections form the base of this Computer Vision challenge.

Image Inpainting

The basic idea of image inpainting is simple: Replace the missing/bad marks in the image with its neighbouring pixels so that it looks like the neigbourhood. There are multiple algorithms that do this - "An Image Inpainting Technique Based on the Fast Marching Method" by Alexandru Telea is perhaps the most common traditional technique. I also experimented with "Navier-Stokes, Fluid Dynamics, and Image and Video Inpainting" technique that is based on fluid dynamics and utilizes partial differential equations.

Edge Detection, Flood fill, Watershed

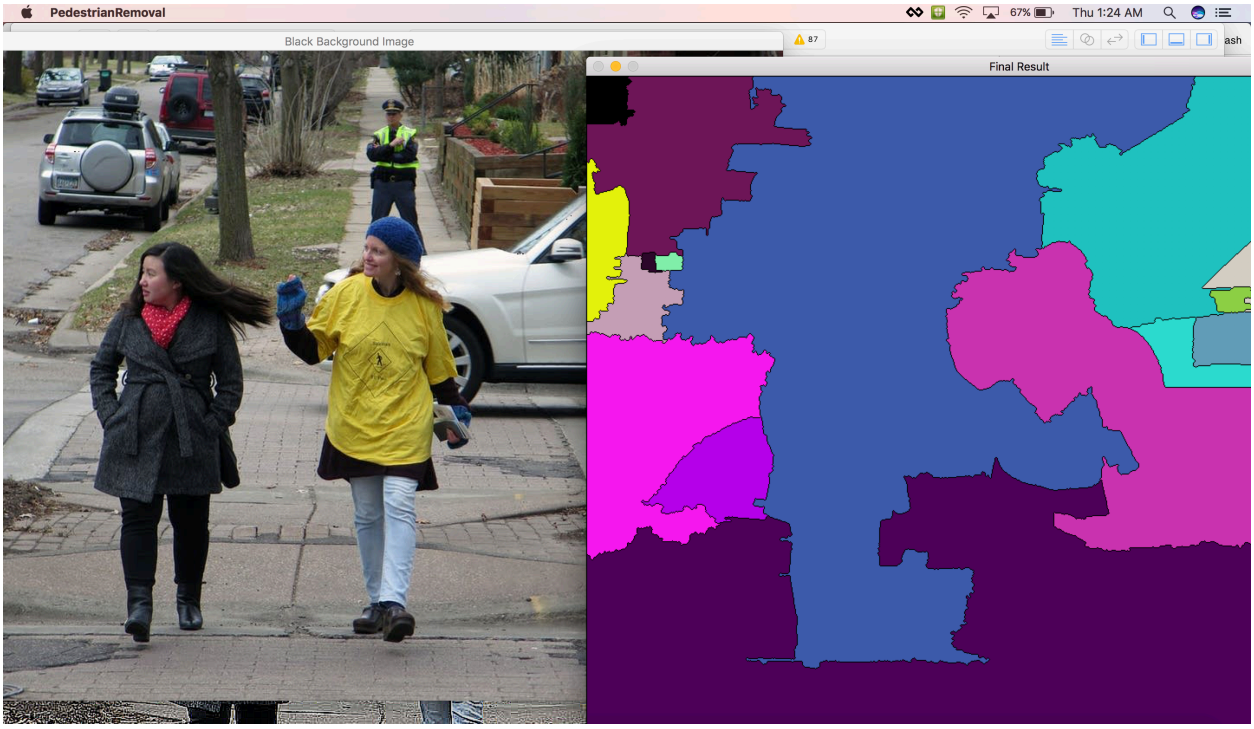


Since the user input type was left to us, my initial thought was to minimize user input, by making the user click on a particular coordinate in the image, and my algorithm magically removing the person occupying that pixel.

1.) One basic approach was to perform Canny edge detection, to use edges as the boundaries for flood fill, which would help us create the image inpainting mask. This would have been perfect if it worked, as it would have captured the contour of humans perfectly. But the noise present in the detailed real world images meant that the edges detected were imperfect, and hence flood fill was not applicable. Shown above is a demo of the mask created by flood fill. Clearly nowhere close to what I had hoped it would be. (The green rectangles are the output of the pedestrian detection model.)

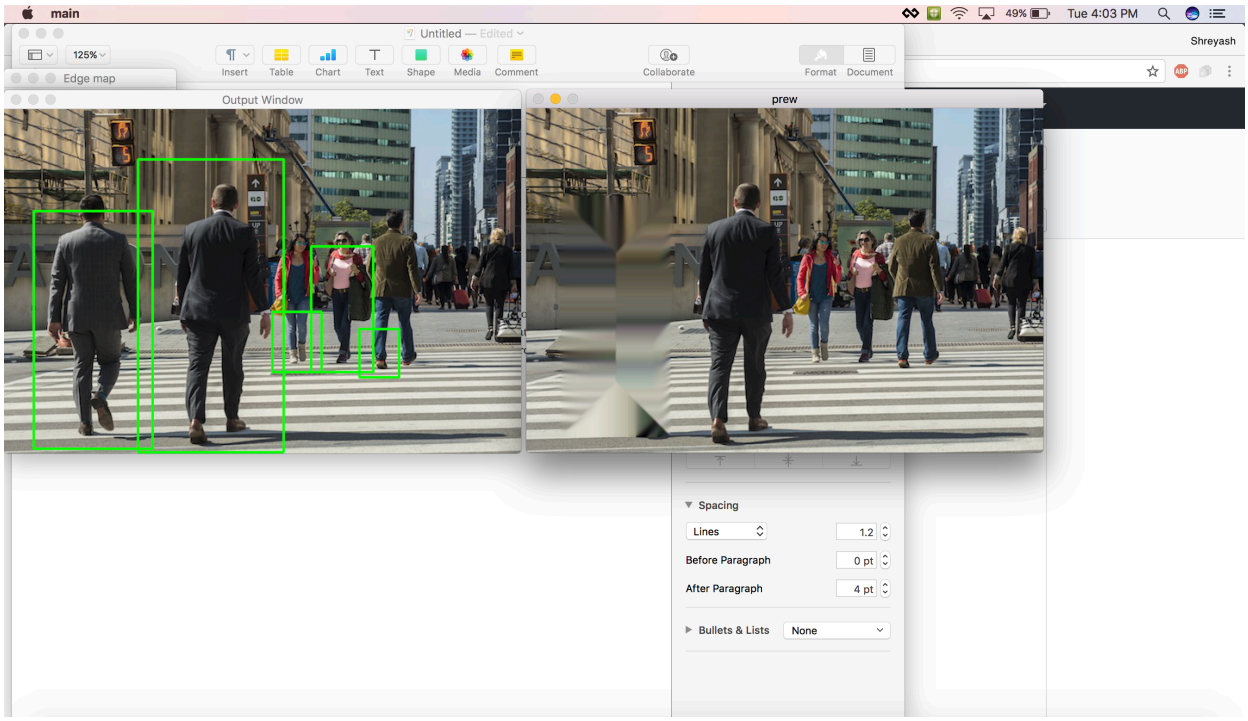
2.) Watershed Algorithm performs image segmentation (provided with defined contours). Image is transformed to have dark values for the background and lighter values for foreground. Then a contour mask is calculated based on the distance transform of different regions. This contour is then fed into the watershed

algorithm that performs image segmentation. My hope was that the contour that we obtain would reflect real world edges and boundaries, thereby making the watershed segments are perfect inpainting masks for the final task. But again, real world data is too complex and the contours obtained through watershed are not reflective of the minute details such as person contour. Example is shown below.



Minimum User Input

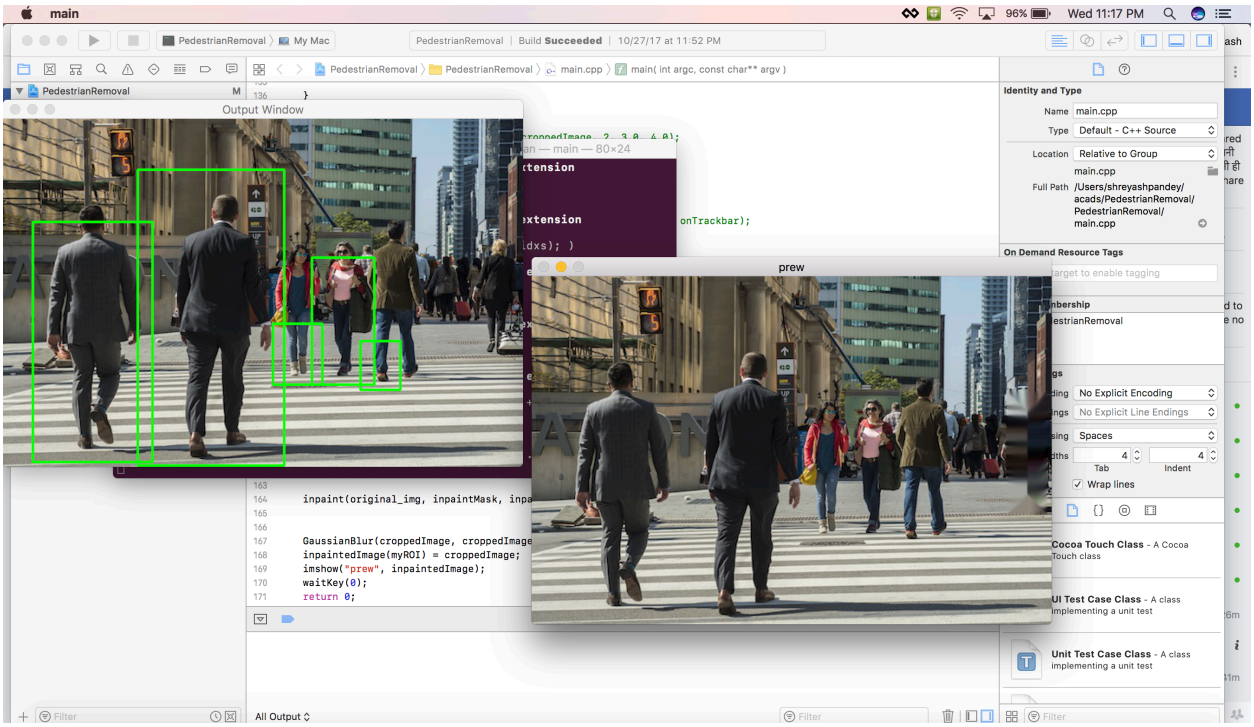
The backup idea was to use the bounding boxes generated by the pedestrian detection model as the contour mask for image inpainting. Even though it's suboptimal, in the sense that it captures a lot of the background along with the pedestrian (making inpainting all the more harder), it was a good baseline that we could improve upon.



Detailed User Input

With the choices of multiple pedestrians detections available to the user, it might be better if the user is allowed to refine the predictions in some way. Using the setMouseCallback functionality available in OpenCV and highGUI, I record the clicks made by the user. Only two clicks are needed - top left corner of the rectangle, and bottom-right corner of the rectangle. It is generally expected that the user would give a tight bounding box around the pedestrian that would make the image inpainting task slightly easier.

This is how the user inputs can be used to refine the detections proposed by the pedestrian detector. The inpainted results are still not up to the mark, which is a limitation of the deterministic approach of image inpainting. Other limitations are listed below, followed by suggested improvements and future work.



Limitations

1.) From the pre-trained classifiers GitHub Repo, “You will notice that successful detections containing the target do not sit tightly on the body but also include some of the background left and right. This is not a bug but accurately reflects the employed training data which also includes portions of the background to ensure proper silhouette representation. If you want to get a feeling for the training data check out the CBCL data set: http://www.ai.mit.edu/projects/cbcl/software-datasets/PedestrianData.html"

2.) The detectors only support frontal and back views but not side views. Side views are trickier and it makes a lot of sense to include additional modalities for their detection, e.g. motion information.

3.) The image inpainting module is deterministic and not context or shape aware. This means that if the pedestrians occupy a large position in the image (i.e. it’s a closeup of the person), then our algorithm fails miserably.

Future Work

There are two components of our framework that are independent of each other - pedestrian detector and image inpainting. This modularity of our approach can help us achieve much better result by improving both components simultaneously.

Pedestrian Detector: Using deep Convolutional Neural Networks can greatly improve the accuracy of our pedestrian detection model. Current state of the art in object detection include the YOLO object detection framework, Faster RCNN and Single Shot Multi-box detector. There are three reasons I didn't experiment with them in this challenge:

1.) Lack of a large dataset. Open-source pedestrian detection datasets include Caltech and Inria datasets that have a handful of images. Deep networks need millions of images to train their parameters.

2.) Lack of GPUs. I could not have trained deep networks on my laptop. It would have taken weeks to get the desired accuracy.

3.) **Language Limitation**. The challenge required us to code extensively in C/C++/Java. All the state-of-the-art pedestrian detection models are available in Python based frameworks such as PyTorch, TensorFlow and Theano. Writing a C++ inference engine for a Caffe model was one way to achieve this, but it was not possible in the limited time that I had.

Image Inpainting: Using Generative Adversarial Networks (GANs) would enable us to perform semantic context-aware image inpainting. Following are some results from the current SOTA image inpainting paper. As can be seen from the results, our deterministic image inpainter can be improved substantially by generative learning approaches. But this again requires training on a large dataset, and training for a long period of time.

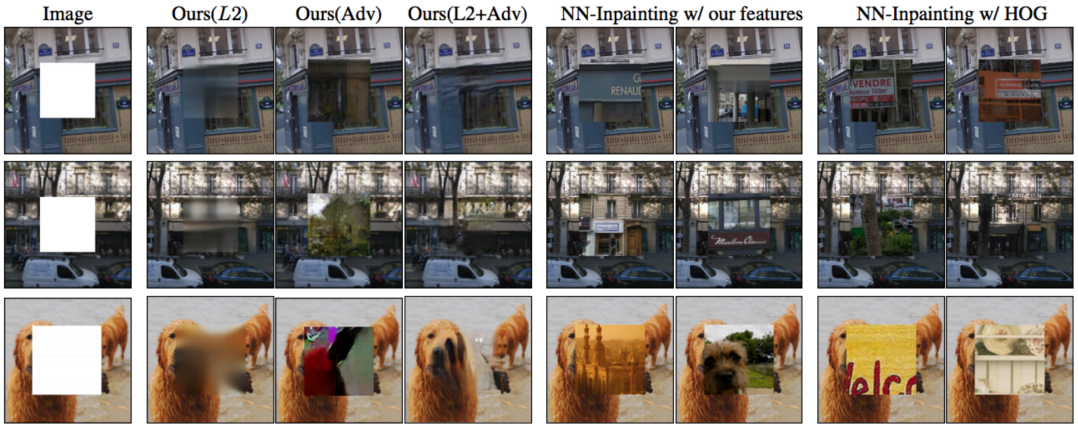


Figure 6: Semantic Inpainting using different methods on *held-out* images. Context Encoder with just L2 are well aligned, but not sharp. Using adversarial loss, results are sharp but not coherent. Joint loss alleviate the weaknesses of each of them. The last two columns are the results if we plug-in the best nearest neighbor (NN) patch in the masked region.

Code Usage

In a unix terminal,

Compilation: g++ $(pkg-config --cflags --libs opencv) main.cpp -o main

Running: ./main

Options: Enter 1 for automatic removal, 2 for single click removal, 3 for two click removal. [increasing levels of interactive human input]