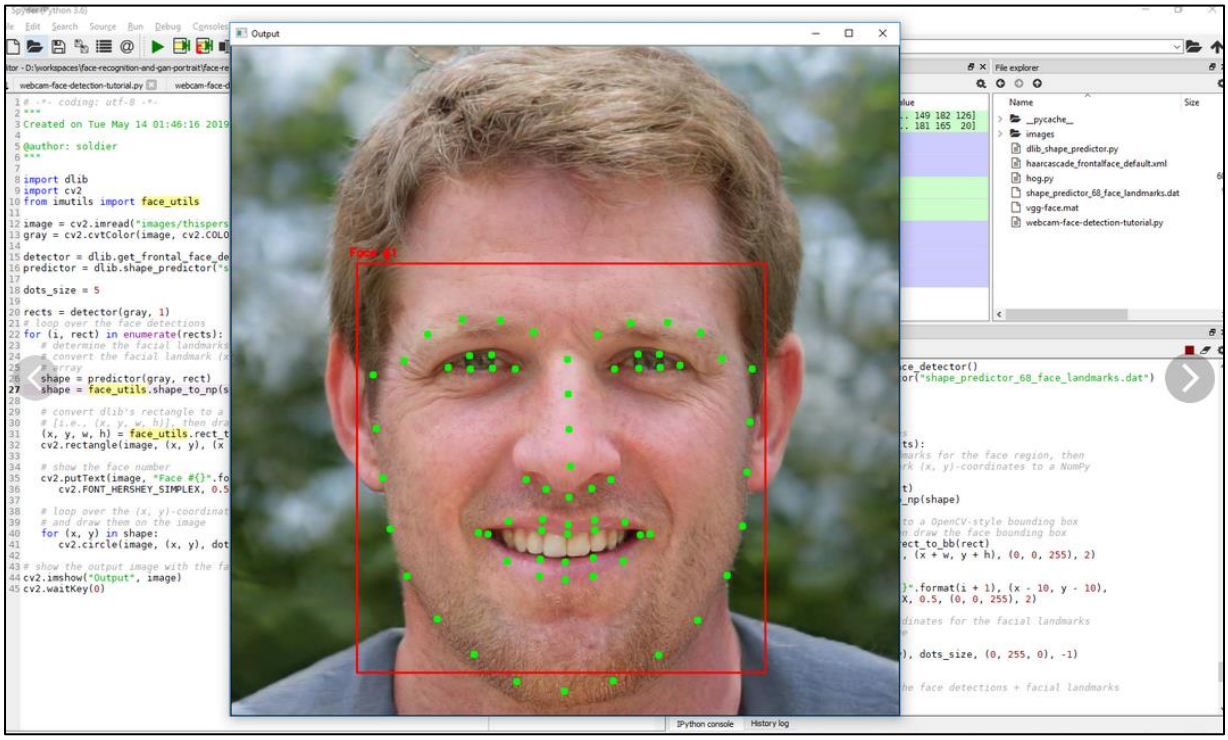
Facial Detection and Recognition With Dlib

By Martin Anderson



First published ***November 10th, 2021*** *at:*

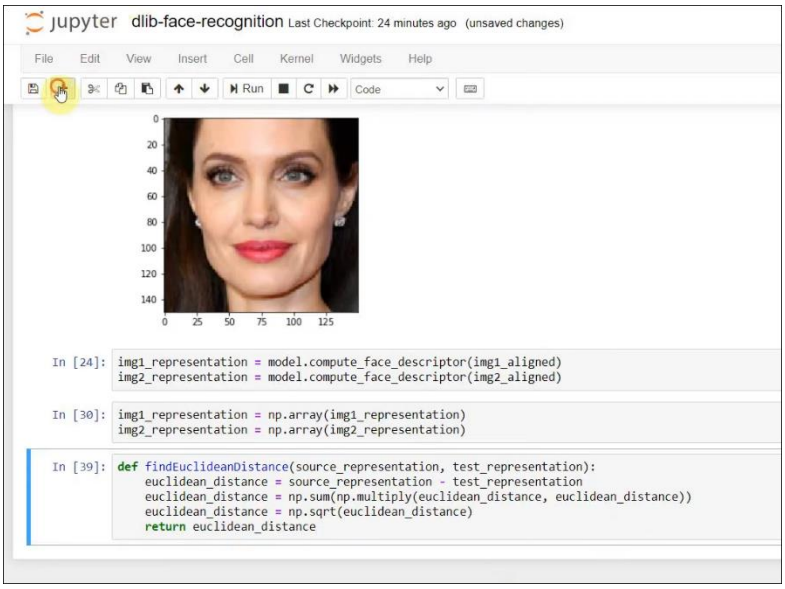
https://www.width.ai/post/facial-detection-and-recognition-with-dlib

Web-archived version

Dlib is an open source suite of applications and libraries written in C++ under a permissive Boost license. Dlib offers a wide range of functionality across a number of machine learning sectors, including classification and regression, numerical algorithms such as quadratic program solvers, an array of image processing tools, and diverse networking functionality, among many other facets.

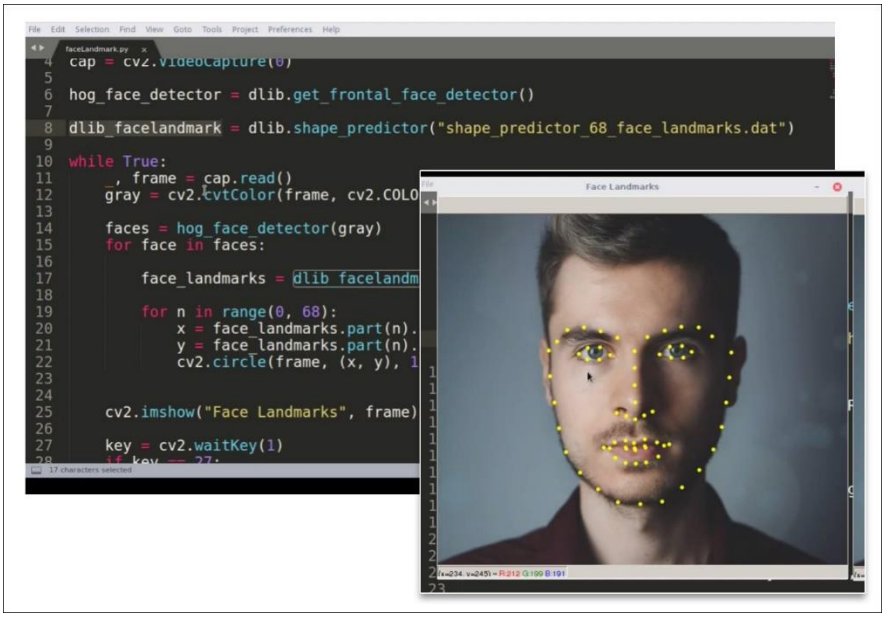
Dlib's HoG in action. Source: https://www.youtube.com/watch?v=GZ2p2hj2H5k

Dlib also features robust tools for object pose estimation, object tracking, face detection (classifying a perceived object as a face) and face recognition (identifying a perceived face).



Dlib being used in a Jupyter notebook as part of a novel facial recognition framework. Source: https://www.youtube.com/watch?v=SIZNf_Ydplg

Though Dlib is a cross-platform resource, many custom workflows involving facial capture and analysis (whether recognition or detection) use the OpenCV library of functions, operating in a Python environment, as in the image below.

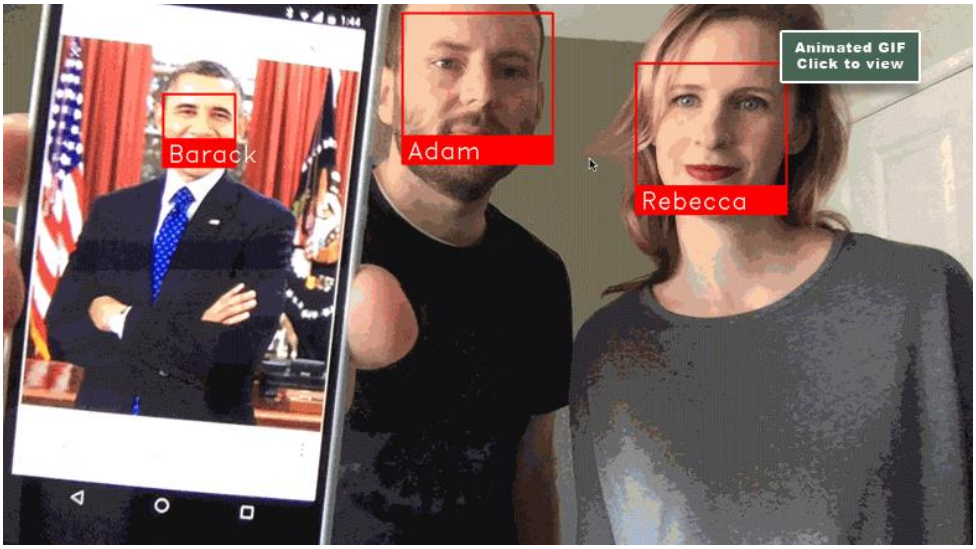


A common environment for use of Dlib – with Python OpenCV. Source: https://www.youtube.com/watch?v=SIZNf_Ydplg

Dlib as a Versatile FOSS Resource

There are a number of novel APIs and interfaces for Dlib, many of which provide additional functionality not directly anticipated by the original creators, such as real-time recognition of multiple faces.

Programmer Adam Geitgey offers a FOSS face recognition API that leverages Dlib.

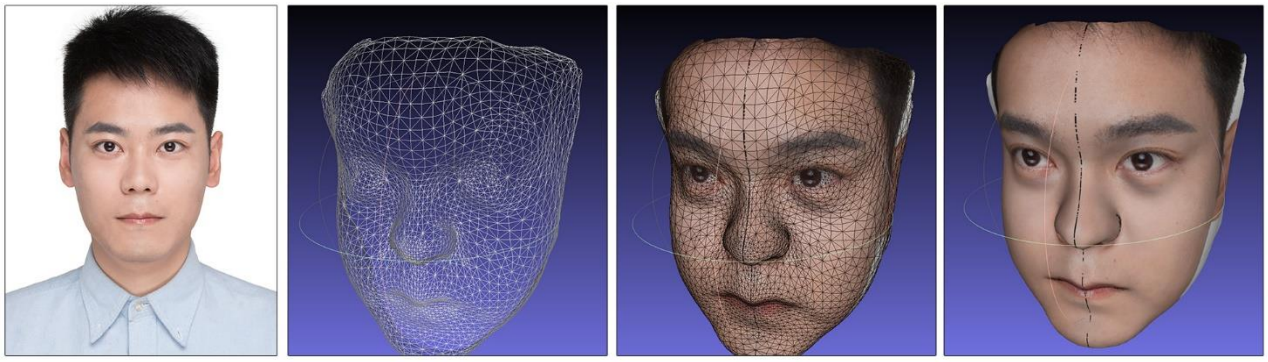


Source: https://github.com/ageitgey/face_recognition

Geitgey's CUDA-capable library has demonstrated 99.83% accuracy on the University of Massachusetts' Labeled Faces in the Wild benchmark dataset. The project encodes Dlib's face captures into 128 data points per face, resulting in unique parameters for the hash of each face across a variety of different photos. Subsequently a Support Vector Machine (SVM) is trained on the derived faces via scikit-learn, resulting in an agile FR model that can run with minimal latency in the right conditions.

Facial Reconstruction

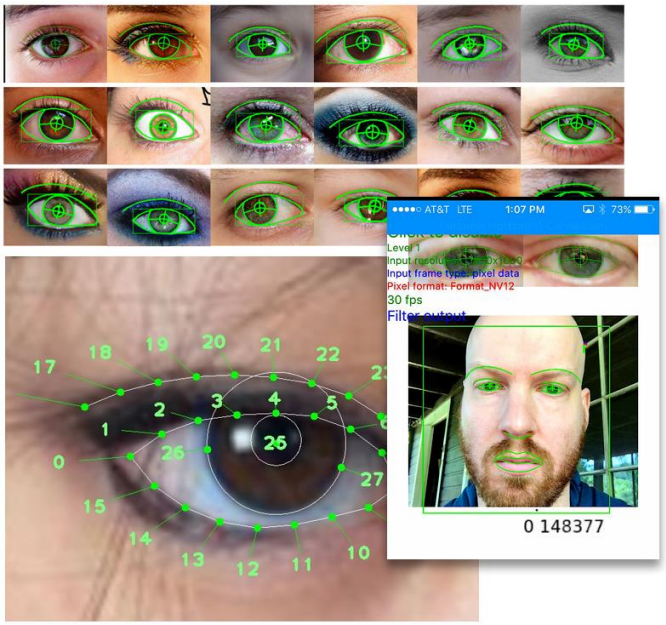
One popular Chinese GitHub repository uses Dlib to power facial reconstruction, and the eos and 4dface libraries to compute geometry and capture textures, for the creation of photorealistic mesh heads.



Facial detection landmarks combine with texture-map capture to provide 3D reconstruction capabilities. Source: https://github.com/KeeganRen/FaceReconstruction

Real Time Eye Tracking for Embedded and Mobile Devices

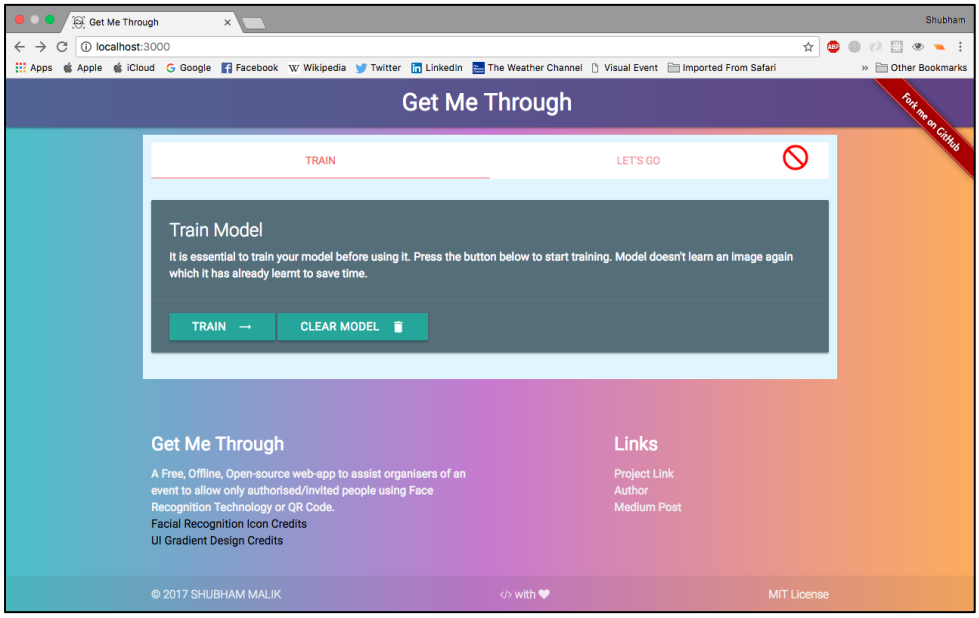
Many projects that utilize Dlib are themselves intended as tool-chain resources, such as Drishti, a real-time eye-tracking framework written in C++11, and intended for iOS and Android devices, as well as embedded ARM and other lightweight computing environments.



Drishti can make use of CMake for iOS deployments, with native Xcode generation smoothing the development process on Apple's platform. Since CMake retains a few extra issues in Android Studio, the developers offer some workarounds to implement Drishti in Android.

Event Security

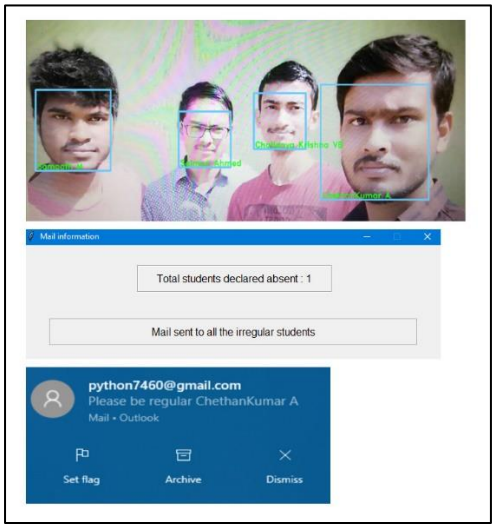
Get Me Through is a Python-based FOSS solution for recognizing and admitting invitees to an event. Besides Dlib, the project uses MongoDB and Node.js v8.1.4+, is written in C++ 11, and supports MacOS and Linux, with untested support for Windows.



Get Me Through *offers a localhost interface to supervise the training of Dlib-recognized faces.* Source: https://github.com/malikshubham827/get-me-through

Attendance Monitoring with Dlib

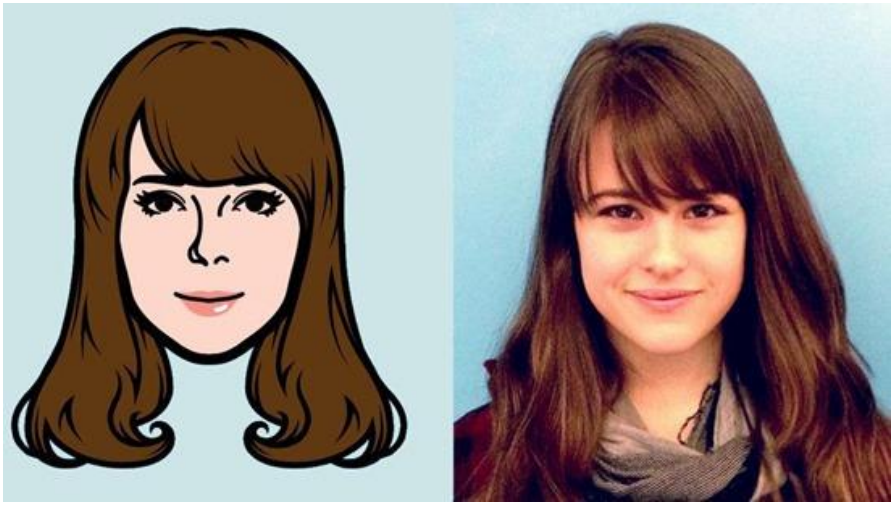
A number of repositories use Dlib as the facial recognition engine for attendance monitoring frameworks. One such project from India offers an automated pipeline, including a webcam recognition framework and the automation of warning mails to students that were not registered by the system during an attendance period.



Source: https://github.com/CKVB/Face-recognition-based-smart-attendance-managment-system-using-dlib

Avatar Generation

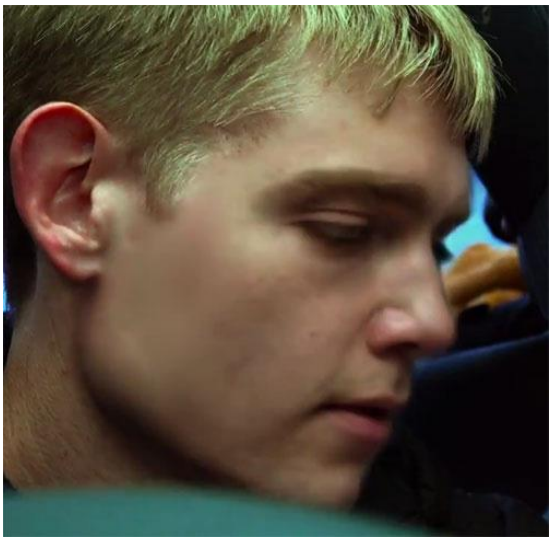
Another India-based C++ GitHub project powered by Dlib uses facial captures to generate stylized anime avatars.



Source: https://github.com/anandpathak/AnimeAvataar/

Dlib in Image Synthesis

Dlib is increasingly being used in image synthesis applications that involve the reconstruction of faces, style transfer, or deepfake images. One legitimate use for the latter is the anonymizing of faces of 'at risk' subjects, as was accomplished for the 2020 release *Welcome To Chechnya*, where a modified version of DeepFaceLab was employed to superimpose 'alternative' faces on interview subjects.

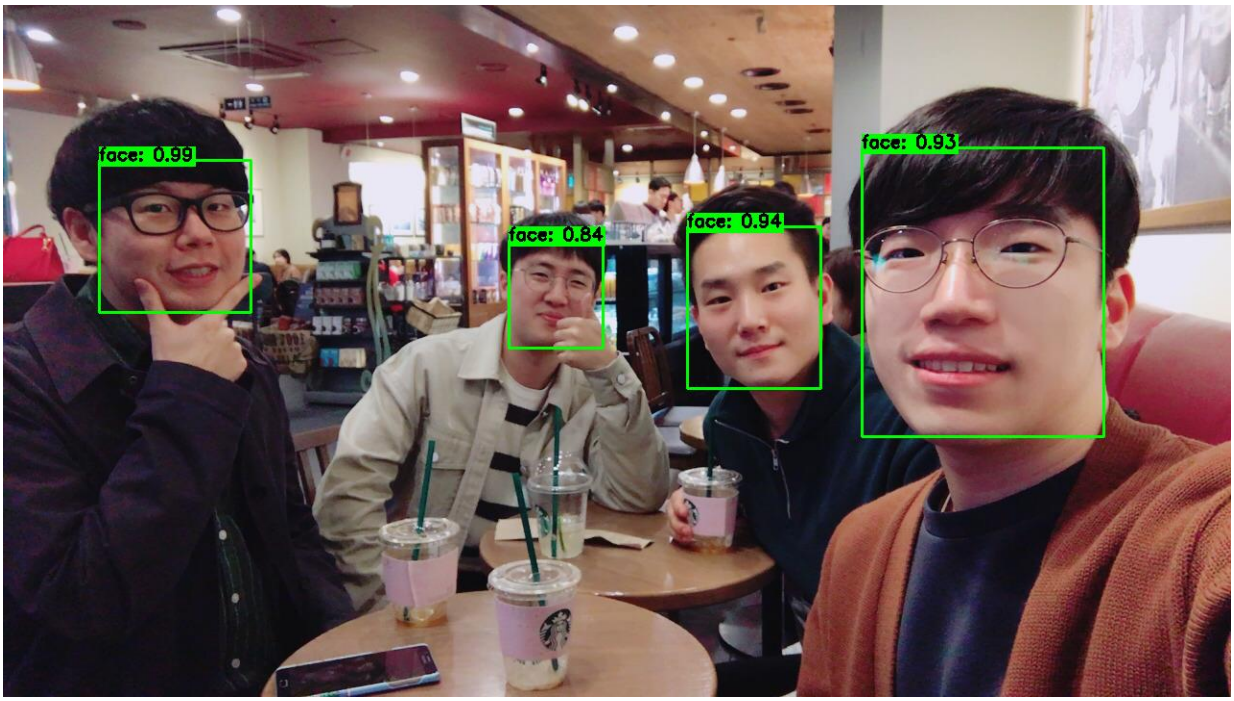


A superimposed identity in Welcome To Chechnya *(2020).*

DeepFaceLab offers Dlib as a face extraction tool, together with the Python library MTCNN, which has its own strengths, but is prone to return more false positives than Dlib. Other popular face recognition libraries include Single Shot Scale-invariant Face Detector (S3FD), which can operate well on a mobile GPU, but may

not get access to it, depending on resource allocation; and which runs poorly on CPU, compared to its stablemates.

To aid early development of a facial recognition/detection framework in your particular operating environment, you can compare Dlib's performance to its peers with Awesome face detection, which allows you to pit six competing libraries against each other: OpenCV Haar cascade, Dlib HoG (see below), Dlib CNN (see below), MTCNN, S3FD and InsightFace.



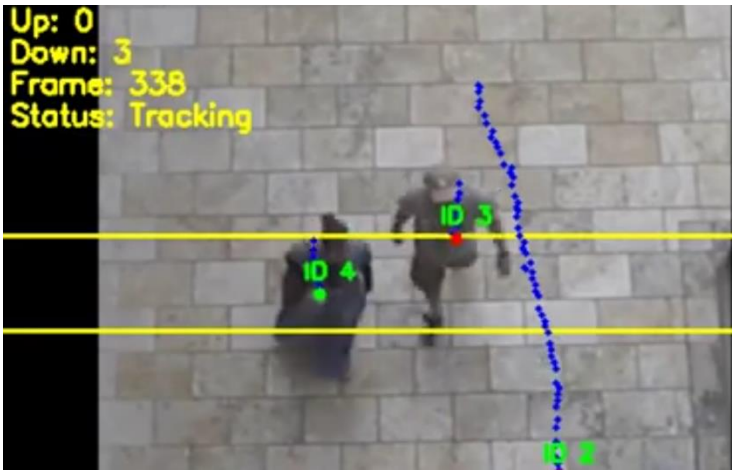
Source: https://github.com/Team-Neighborhood/awesome-face-detection

Commercial Use Cases for Dlib

There are many more directly commercial use cases for Dlib's face detection capabilities, where the objective is to individuate a face from images or a video stream. These include:

- Simple security surveillance deployments that are designed to create alerts or begin logging or recording when a face (rather than a facial identity) is picked out by the system.
- Crowd-counting, for congestion monitoring in transport infrastructure and large public events, or to monitor footfall in retail environments.
- Identifying unmasked faces in times of public restrictions regarding the wearing of face masks.

Face detection can be used to count the numbers of participants at large gatherings, though dedicated 'crowd counting' systems are emerging to handle the high volume of faces per image/video. In this case, we see that the minimum requirements for a face to be identified by a pure face detection system are not met by some of the most distant participants (left image), requiring the training of models that take this 'small presence' into account (image on right). Source: https://arxiv.org/pdf/1906.07538.pdf



A closed circuit TV surveillance system looks down on passing pedestrians and manages to identify faces at extremely acute angles, for purposes of counting footfall, among others. Source: https://www.youtube.com/watch?v=DdtSWV5AdeY

Naturally, face detection also operates as a precursor or initial phase for facial recognition, where the system will attempt to maintain consistent focus on an identified face and run its characteristics through a database that's likely to turn up an ID match.

General or 'abstract' use of this application is (for obvious reasons) limited to periodic state-led technology initiatives by police and similar authorities, though these often attract controversy.

In private corporate environments, facial recognition can be used to monitor attendance and facilitate security access in diverse ways, from gaining building access to unlocking timed-out workstations, among a myriad of other possibilities. Other uses include:

- Mobile-based, lightweight **facial authentication**.
- **Image synthesis techniques**, such as the use of encoder-decoder systems to generate 'face-swaps'.
- **Facial control frameworks** for disabled people with very limited mobility.

Dlib is incredibly fast and very lightweight. It can comfortably operate at 30fps in standard environments, and can potentially detect facial landmarks in a single millisecond, though only in the most ideal conditions. It can also operate on hardware as basic as a Raspberry PI.

Additionally, it's possible to train Dlib to identify specific shape traits in a face, for general research or for medical applications.

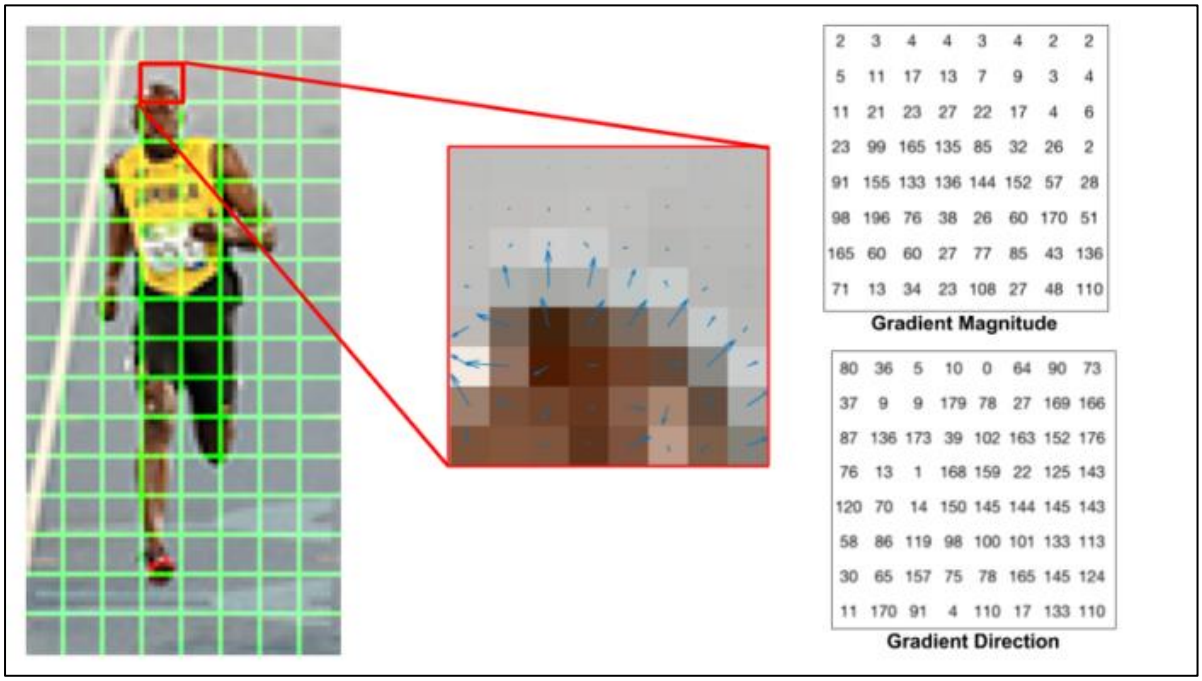
Dlib's Two Methods

Dlib offers two different functions for facial capture:

HoG + Linear SVM

The Histogram of Oriented Gradients (HoG) + Linear Support Vector Machine (SVM) algorithm in Dlib offers very fast recognition of front-on faces, but has limited capabilities in terms of recognizing face poses at acute angles (such as CCTV footage, or casual surveillance environments where the subject is not actively participating in the ID process).

It also supports passport-style profile faces, though with very little margin for error (faces pointing up or down, etc.). HoG + SVM is suitable for constrained situations where the sensor can expect a direct and unobstructed view of the participant's face, such as ATM and mobile framework ID systems, as well as mobile traffic surveillance recognition systems, where cameras are able to obtain a straight profile shot of drivers.



HoG is a reductive but speedy algorithm that breaks down an image into constituent groupings of pixels, from which it derives 'features' that can be correlated to known types of object. To accomplish this, it first creates a low-level histogram, which outlines the lineaments of objects in the image, based on pixel intensity and the extent to which it suddenly drops off (i.e. at the edge of an object). Source: https://medium.com/analytics-vidhya/a-gentle-introduction-into-the-histogram-of-oriented-gradients-fdee9ed8f2aa

Max-Margin (MMOD) CNN face detector

MMOD is a robust and reliable, GPU-accelerated face detector that leverages a convolutional neural network (CNN), and is far more capable of capturing faces at obscure angles and in challenging conditions, suiting it for casual surveillance and urban analysis.

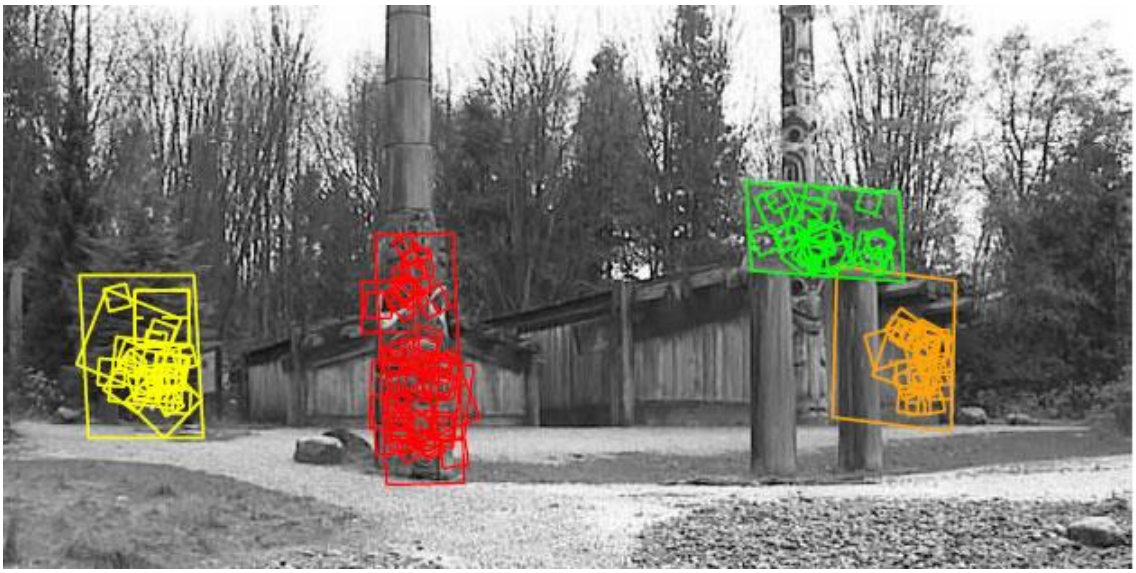
MMOD is not a distinct alternative to HoG + Linear SVM, but rather can be applied to HoG itself, or to any bag-of-visual-word model, which treats discovered pixel groupings as explorable entities for potential labeling — including the identification of faces.

Shepherding Pixels and Entities Into Groups

In such cases, these explorable entities are discovered via a three-step process:

1: Feature Extraction

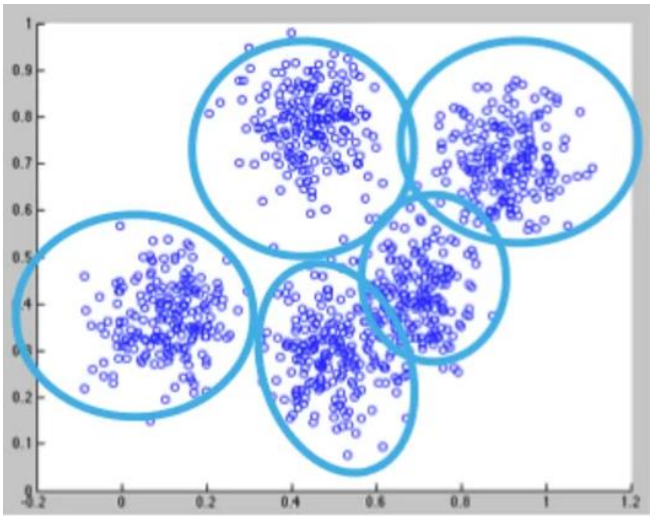
Where key points in the image are detected and assigned to Scale-Invariant Feature Transform (SIFT) features.



SIFT identifies groups and sub-groups in a photo of totem poles, with the non-skewed rectangles representing the higher grouping. Source: https://aishack.in/tutorials/sift-scale-invariant-feature-transform-introduction/

2: Codebook/'Vocabulary' Construction (normally k-means)

At this point, it's necessary to classify the discovered groups, and to segment them from background information. Unsupervised K-Means clustering can accomplish this well by iterating over all this unlabeled data until it has calculated the minimal sum of squared distances between all the captured points and the center of the cluster. When all those centers have been calculated, each will form the apex of a grouping, which can be fed into the next stage.



Here K-Means has easily distinguished the center of each grouping, but in many cases it will be more difficult to assign nodes to one group or another, due to overlap and other ambiguities. Source: https://www.javacodegeeks.com/2016/11/monitoring-real-time-uber-data-using-spark-machine-learning-streaming-kafka-api-part-1.html

3: Vector Quantization

Vector Quantization (VQ) hails from early signal processing research, and has been a central plank of compression technologies, since it deals with the definition of minimum units from a 'noisy' environment. In our work-flow, VQ calculates the number of clusters found in stage #2 (see above) against the frequency of recurring patterns in order to provide a feature representation layer, and converge the estimated groupings into usefully distinct entities.

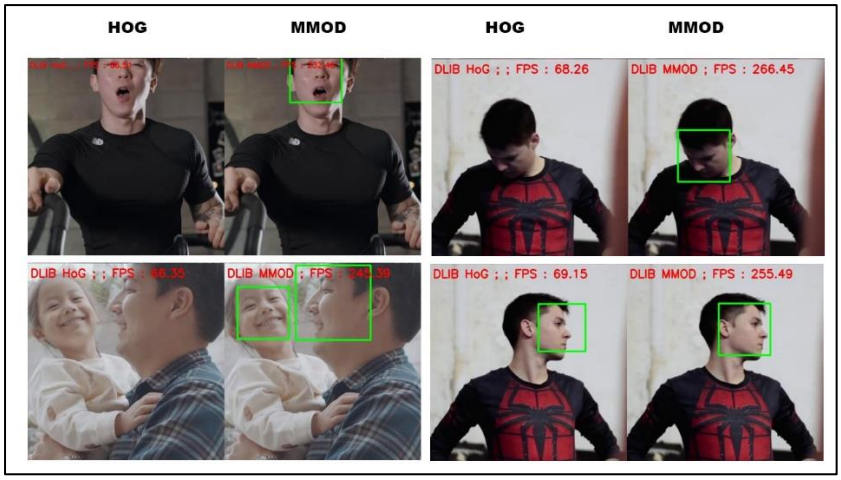
HoG Vs. MMOD

The appeal of **HoG + Linear SVM** under Dlib is its low use of resources; its efficacy when operating on CPU; the fact that it has at least some latitude for non-frontal faces; its low-impact model requirements; and a relatively capable occlusion detection routine.

Negatively, a default deployment requires a minimum face-size of 80x80 pixels. If you need to detect faces below this threshold, you'll need to train your own implementation. Additionally, this approach gives poor results on acute face angles; generates bounding boxes that may over-crop facial features; and struggles with challenging occlusion cases.

The advantage of **MMOD (CNN)** under Dlib is (perhaps above all) its ability to recognize difficult face orientations (which may be the deciding factor, depending on your target environment); its impressive speed when allowed access to even a moderately-specced GPU; its lightweight training architecture; and its superior occlusion handling.

Negatively, it can produce bounding boxes even more restricted than HoG + Linear SVM in a default deployment; performs notably more slowly on a CPU than HoG/LSVM; and shares HoG/LSVM's native inability to detect faces smaller than 80 pixels square — again, necessitating a custom build for certain scenarios, such as acute street surveillance viewpoints that extend into the distance.



HoG = left; MMOD = right. Top left: MMOD finds a face in spite of a framing occlusion, while HoG misses it; top-right: HoG cannot discern a face turned to this angle, whereas MMOD easily distinguishes it; bottom-left: Again, HoG's training is not up to these angles, even though one is a straight profile, whereas MMOD captures both; and finally, bottom-right – HoG has been trained on uncomplicated profile shots, and manages to match MMOD in identifying the angle. Note, however, the limited bounding box of each algorithm, which end at the sum of the central features (eyes/nose/lips) that are used by both to identify a face. Source: https://learnopencv.com/face-detection-opencv-dlib-and-deep-learning-c-python/

Model Repository and Resources

The creator of MMOD, Davis King, has provided a number of useful open source trained models for Dlib, many (but not all) of which center on facial recognition. These include:

A Dlib Face Recognition Network model with 29 convolutional layers, an optimized version of the well-used ResNet-34 network. This model was trained on three million faces across various datasets, including Face Scrub, Oxford's VGG set, and the author's own web-scraped data.

A human face detector dataset forked from a standard Dlib set, with images annotated using Dlib's native imglab tool.

A **shape predictor** facial landmarking model trained on the 7198 faces in Dlib's 5-point face landmark dataset, usable with either HoG or the CNN detector.

An **Imperial College London** dataset designed for HoG, which is excluded from commercial use.

Conclusion

Dlib is a versatile and well-diffused facial recognition library, with perhaps an ideal balance of resource usage, accuracy and latency, suited for real-time face recognition in mobile app development. It's becoming a common and possibly even essential library in the facial recognition landscape, and, even in the face of more recent contenders, is a strong candidate for your computer vision and facial recognition or detection framework.