CV Practice Class
Lecture 5

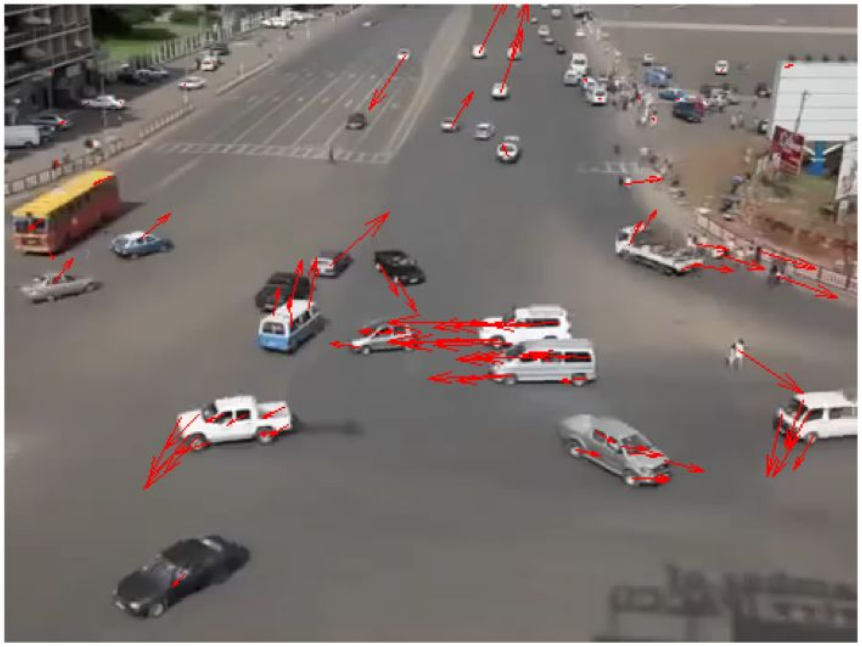
2017-07-10

FIRA 인공지능 에이전트 과정
SNUVL Lab

Contents

Optical Flow

Video Input/Output
Optical Flow Introduction
Lukas-Kanade Optical Flow



Content credit: Gunhee Kim's CV slides, OpenCV tutorials

Video Input: `cv2.VideoCapture`

- Capturing video from files or devices

```
v_in = cv2.VideoCapture("video.mp4") # Open a video
v_in = cv2.VideoCapture(0) # Open the default camera
```

- Grab a from the file/device

```
ret, frame = v_in.read()
```

- Get properties of the video

```
prop = v_in.get(cv2.CAP_PROP_FRAME_WIDTH) # frame width
```

- Close the file/device

```
v_in.release()
```

Video Output: `cv2.VideoWriter`

- Choose a video codec and initialize video writer

```
fourcc = cv2.VideoWriter_fourcc(*'X264') # Define FOURCC
v_out = cv2.VideoWriter('out.mp4', fourcc, FPS, (width, height))
# Write video file
```

  - X264 is the FourCC of H.264/MPEG-4 AVC
  - OpenCV looks for a proper library of H.264 and initialize(Installed ffmpeg will be used)

- Write a frame(image) to the file

```
v_out.write(frame)  # image size should match
```

- Close the file

```
v_out.release()
```

Display an Image on Window

- Open a named window and display an image on it

```
cv2.namedWindow("Window")
cv2.imshow("Window", image)
```

  - Just calling `cv2.imshow()` opens the window if there's no window with the name specified

- Wait for a key pressed and Draw the window

```
c = cv2.waitKey(10)  # wait 10ms for keyboard input
      # and for the message loop to draw the window
```

- Close the window/windows

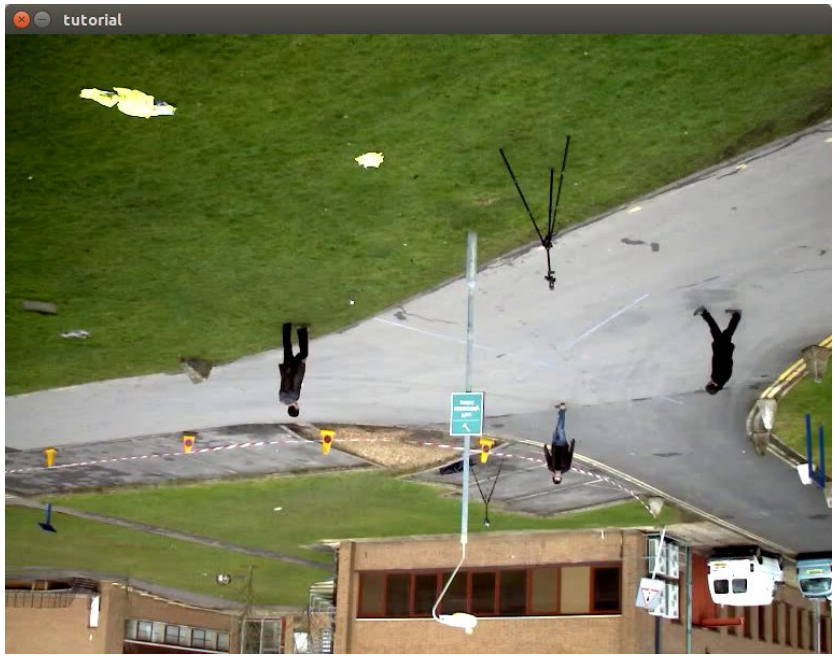
```
cv2.destroyWindow("Window")  # Close a specific window
cv2.destroyAllWindows()  # Close all wind
```

Let's Check the Code

`video_tutorial.py`

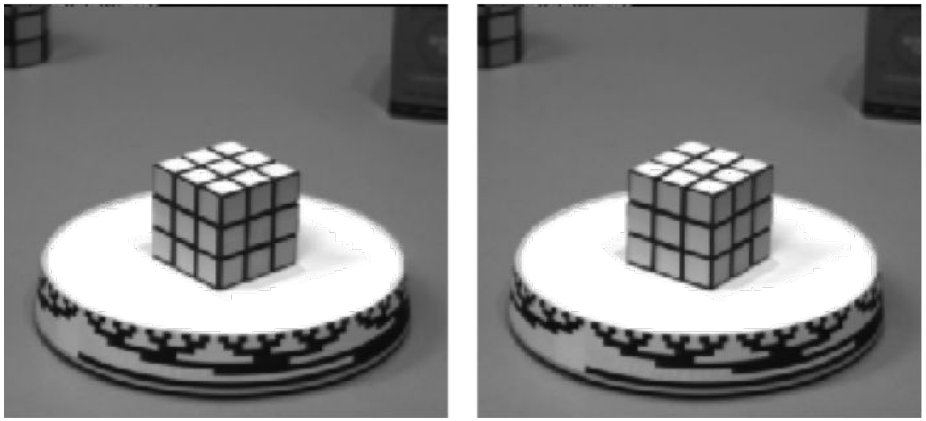
- To run the script:

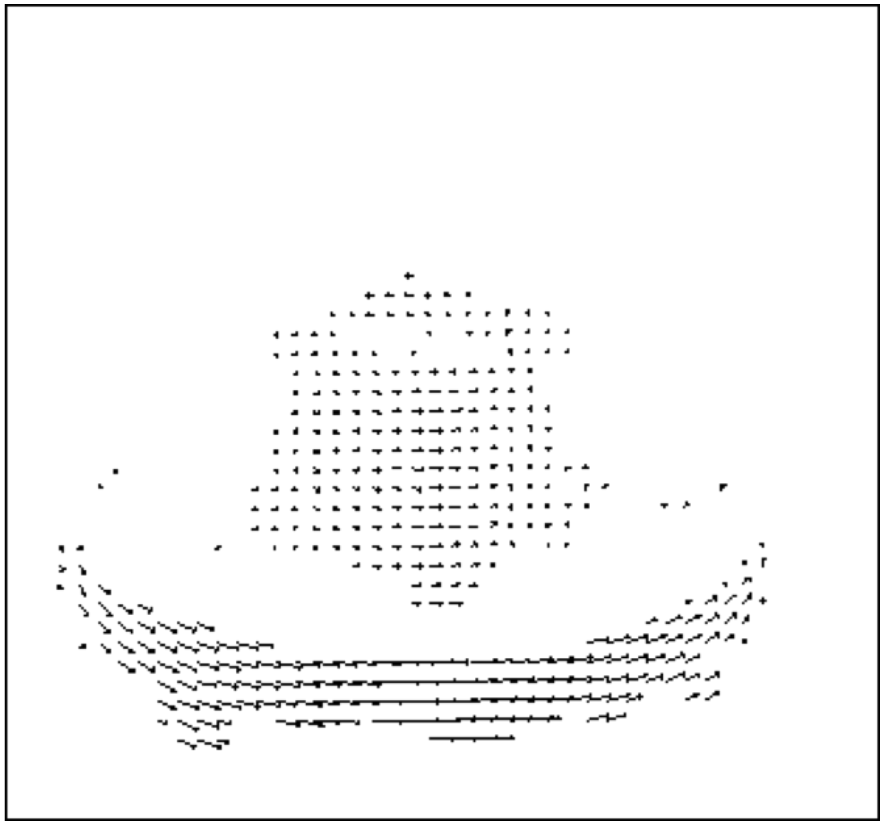
```
$ python video_tutorial.py
```



Lukas-Kanade Optical Flow

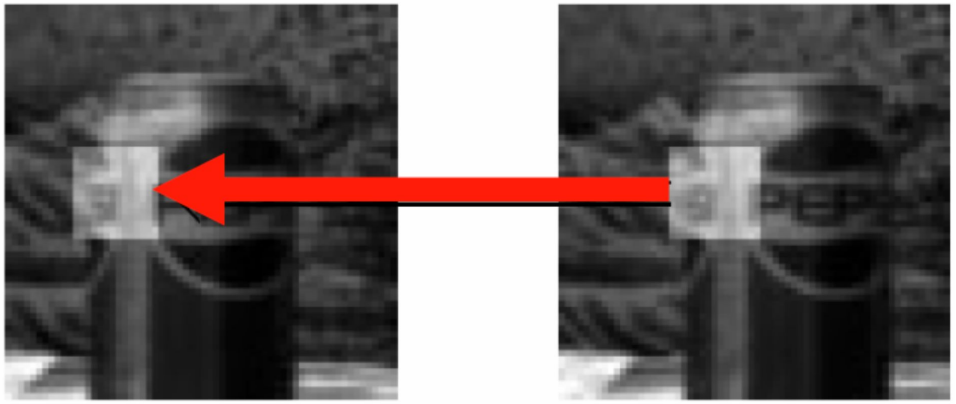
- Optical Flow
  - Motion of brightness pattern in the image
  - Ideally optical flow = motion field

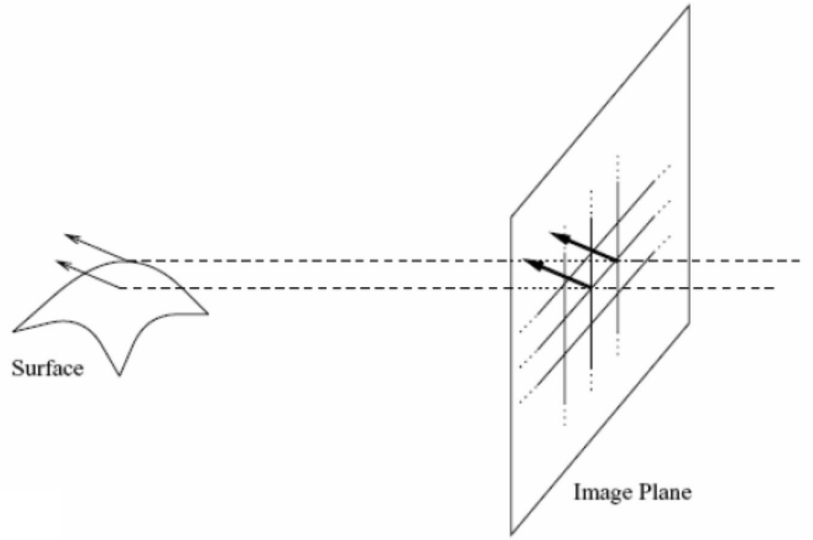


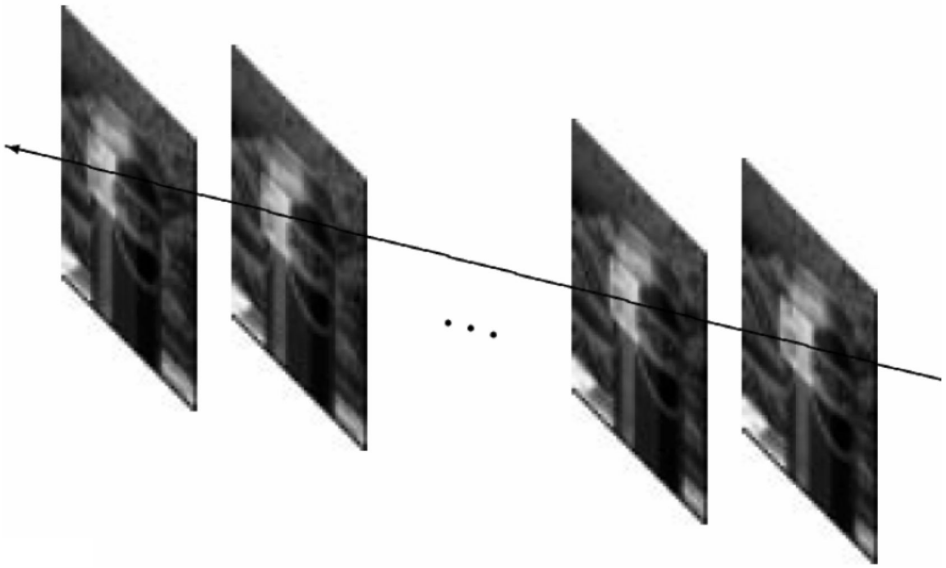


Lukas-Kanade Optical Flow

- Three assumptions
  - Brightness consistency
  - Spatial coherence
  - Temporal persistence







Lukas-Kanade Optical Flow

- Lukas-Kanade Method(for Translation)
  - A point (x, y), translation (u, v), Template image(before) $T(x, y)$, Input Image(after) $I(x, y)$, and Image gradient I_x, I_y,
  - For small window around (x, y), find (u, v) that minimize error:

$$E(u, v) = \sum_{x,y} (I(x + u, y + v) - T(x, y))^2$$

  - Tayles expansion:

$$I(x + u, y + v) \approx I(x, y) + uI_x + vI_y$$

  - Setting the gradient w.r.t u and v to zero gives,

$$\begin{bmatrix} \sum_{x,y} I_x^2 & \sum_{x,y} I_x I_y \\ \sum_{x,y} I_x I_y & \sum_{x,y} I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum_{x,y} I_x(T(x, y) - I(x, y)) \\ \sum_{x,y} I_x(T(x, y) - I(x, y)) \end{bmatrix}$$

  - Iteratively find u and v using Newton's method

Lukas-Kanade Optical Flow

- LK Optical Flow in OpenCV-Python

```
nextPts, status, err = cv2.calcOpticalFlowPyrLK(prevImg, nextImg,
                                                prevPts, nextPts, ...)
```

  - Compute optical flow of input points using the iterative LK method.
    - `prevImg`: First 8-bit(grayscale) input image.
    - `nextImg`: Second input image of the same size and type as prevImg
    - `prevPts`: Vector of 2D points for which the flow needs to be found
    - `nextPts`: Output vector of 2D points
    - `status`: Output status for each point of prevPts that are found in nextImg
    - `err`: Output error for each point

- Finding initial points - Shi-Satomi corner detector

```
pts = cv2.goodFeaturesToTrack(gray_img, mask=None, **feature_params)
```

    - Find corder points that are good to be tracked by trackers($\min(\lambda_1, \lambda_2) > k$)

Lucas-Kanade Optical Flow

- Calculating LK Optical Flow

1. Set points to be used in LK optical flow
   a. Grid points with every N(=10) pixels
2. For every frame,
   a. Calculate optical flow on grid points with consecutive frames using `cv2.calcOpticalFlowPyrLK()`
   b. Mark the moved points as arrows

Let's Check the Code

`lk_optical_flow.py`

- To run the script:

```
$ python lk_optical_flow.py
```

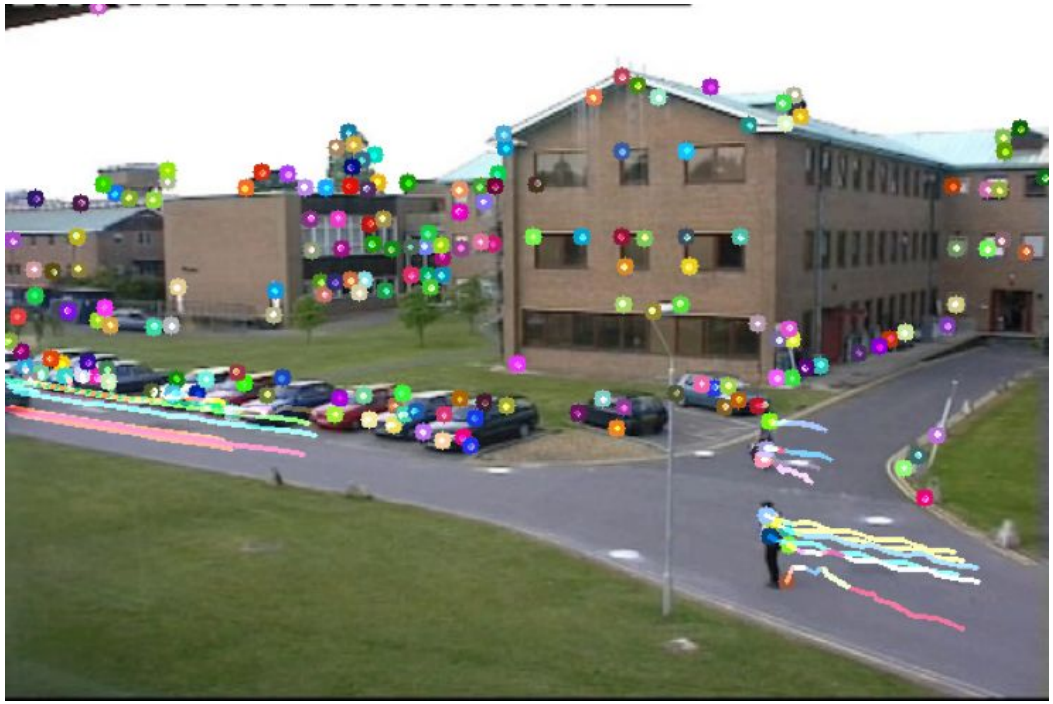
Lucas-Kanade Optical Flow

- Calculating LK Optical Flow
  - Merely grid points are not good to be tracked. The optical flow calculated on those points are prone to be miscomputed

Lucas-Kanade Optical Flow

- Point Tracking with LK Optical Flow

1. Find salient points from the first frame
   a. Call `cv2.goodFeaturesToTrack()` with the first frame image
2. For every frame,
   a. Calculate optical flow and errors(cv2.calcOpticalFlowPyrLK())
   b. Mark the tracks of moved points with low error



Let's Check the Code

`lk_tracking.py`

- To run the script:

```
$ python lk_tracking.py
```

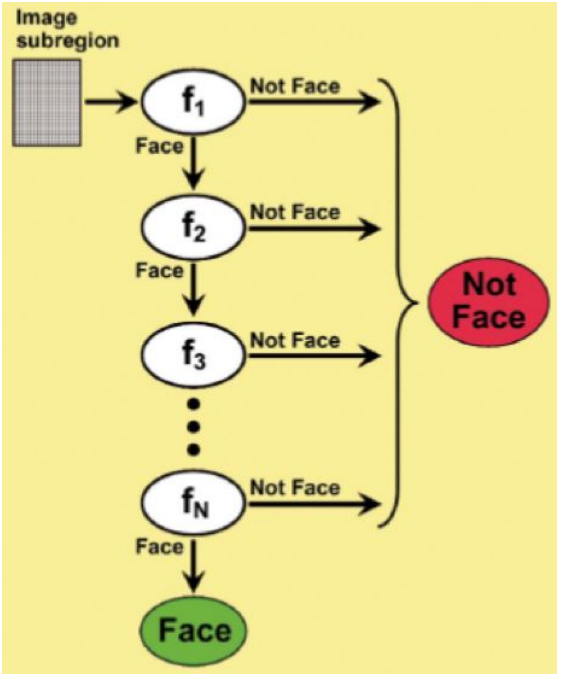
Face Detection

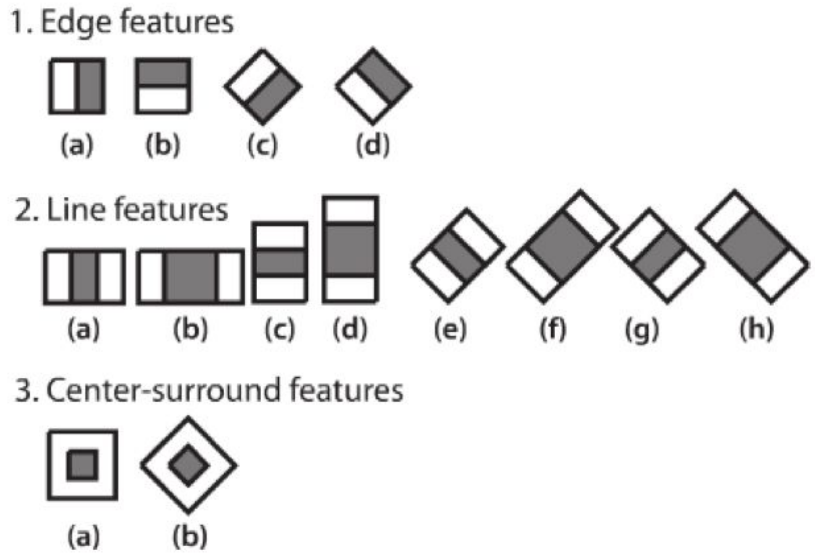
Viola-Jones Face Detection

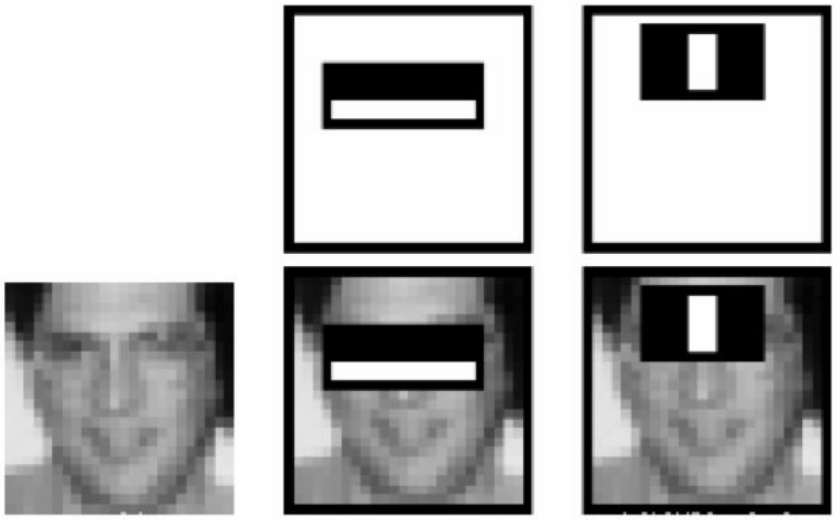
Content credit: OpenCV tutorials, A blog

Viola-Jones Face Detection

- Cascade of Classifiers
  - Cascading weak classifiers
  - Effective / Efficient
- Haar Feature Selection
  - All human faces have similar properties
  - ex) The eye region is darker than the upper-cheeks
  - Can be computed efficiently





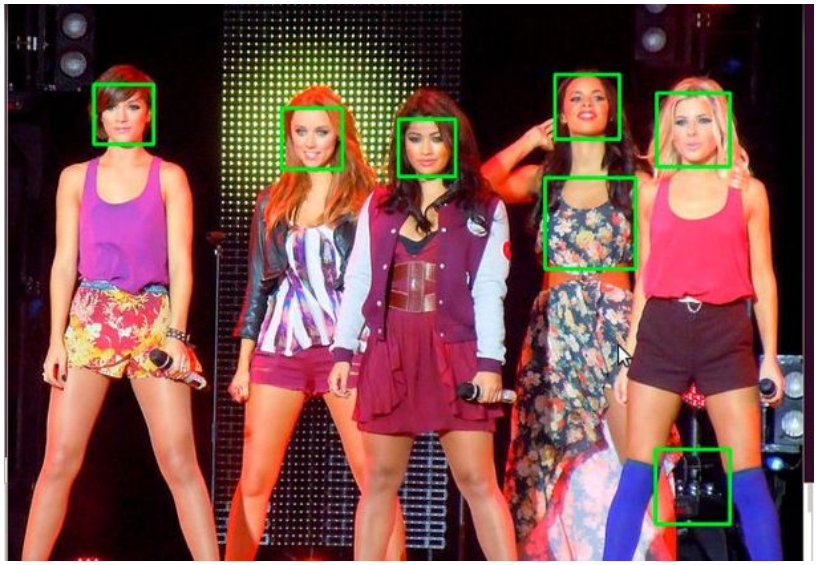


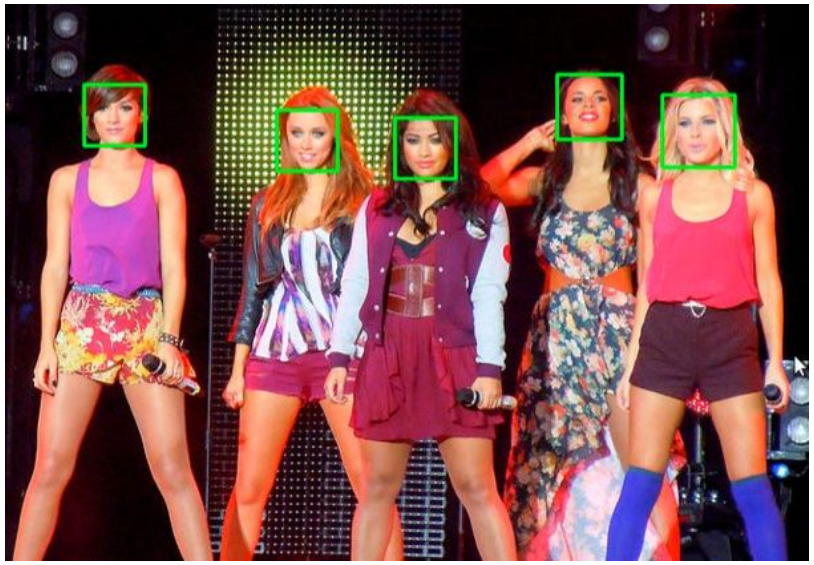
Face Detection in OpenCV

- Face Detection using Haar Cascades

```
# Create the haar cascade
cascPath = 'detect/haarcascade_frontalface_alt.xml'
faceCascade = cv2.CascadeClassifier(cascPath)
```

```
# Detect faces in the image
faces = faceCascade.detectMultiScale(gray, 1.2, 5)
```

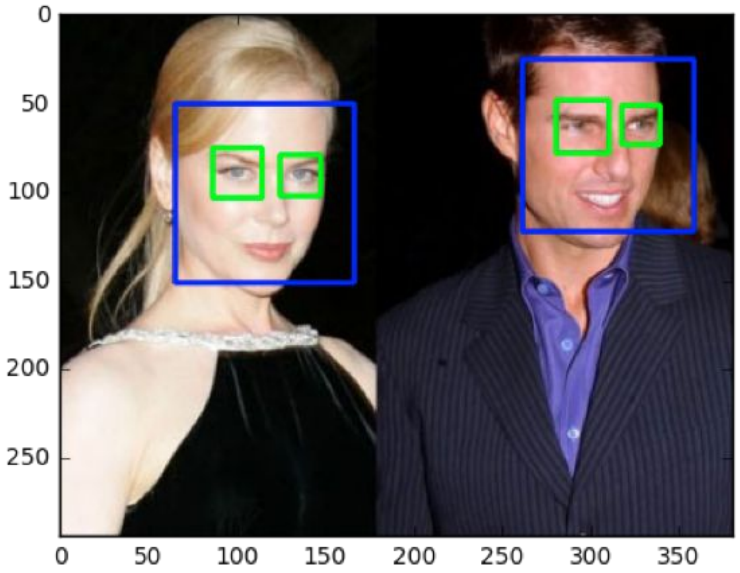




Practice - Face & Eye Detection

- Given an image of two people

1. Detect two faces.
2. In each ROI of face, detect two eyes.



Let’s Check the Code

`face_detection.ipynb`