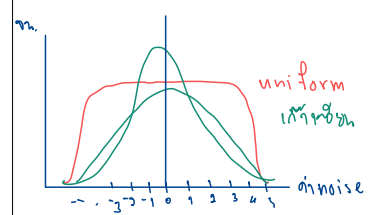


# OpenCV

## Previous Lecture



## Previous Lecture

- You did learn ...
  - Machine learning in computer vision
    - K-Nearest Neighbour (kNN)
      - Blue & red families
      - OCR of hand-written data (digits and alphabets)
    - Support Vector Machines (SVM)
      - Linearly separable data
      - OCR of hand-written digits (with HOG)
    - K-Means Clustering (Unsupervised)
      - T-shirt size problem (one feature & multiple features)
      - Color quantization

# OpenCV

## Computational Photography

## Image Denoising

กำจัด noise

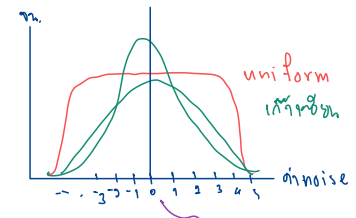
- Goals
  - You will ...
    - Learn about **Non-local Means Denoising** algorithm to remove noise in the image.
    - See different functions like ...
      - `cv2.fastNlMeansDenoising()`
      - `cv2.fastNlMeansDenoisingColored()`
      - etc.

## Theory

- In **earlier lectures**, we have seen many **image smoothing techniques** like Gaussian Blurring, Median Blurring, etc.
  - They were **good to some extent** in removing **small quantities of noise**.
  - In those techniques, we took a small neighbourhood around a pixel and did some operations like gaussian weighted average, median of the values, etc. to replace the central element.
  - In short, **noise removal** at a pixel was **local to its neighbourhood**.
- What if the quantity of noise is large?

ใช้หลายรูป

## Theory



- There is a **property of noise**.
- Noise is generally considered to be a **random variable** with **zero mean**.
- Consider a noisy pixel,  $p = p_0 + n$  where  $p_0$  is the true value of pixel and  $n$  is the noise in that pixel.
- You can take large number of **same pixels** (say  $N$ ) from **different images** and computes their **average**.
  - Ideally, you should get  $p = p_0$  since mean of noise is zero.

$$\begin{aligned} p^1 &= p_0 + n^1 \\ p^2 &= p_0 + n^2 \\ &\vdots \\ p^m &= p_0 + n^m \end{aligned} \quad \left| \quad \text{Avg} = \frac{mp_0 + (n^1 + n^2 + \dots + n^m)}{m} \right.$$

$$= p_0 + \frac{n^1 + n^2 + \dots + n^m}{m}$$

$$= p_0$$

ใช้ได้ก็ต่อเมื่อเวลา: ภาพต่อเนื่อง | IRL: ไม่สามารถนำรูปที่มีเสียงรบกวนมาหลายภาพ  
ซึ่งมีตำแหน่งที่ต่างกัน

## Theory

- You can **verify** it yourself with a **simple setup**.
  - Hold a static camera to a certain location for a couple of seconds.
  - This gives you **plenty of frames** (a lot of images) of the **same scene**.
  - Then write a piece of code to find the **average of all the frames** in the video (This should be simple for you now).
  - Compare the **final result** with the first frame.
    - You will see a **reduction in noise**.
- Unfortunately, this method is **not robust to camera and scene motions**.
- Also, often there is only **one noisy image available**.
- What should we do?

## Theory ใช้แค่ 1 ภาพ

- So, the idea is simple. ใน 1 ภาพ เราจะใช้ พท. ที่ใกล้เคียงกันมาหาค่าเฉลี่ย
  - We need a set of **similar images (patches)** to average out the noise.
- Consider a small window (say 5x5 window) in the image.
  - It is likely that the **same patch** may be **somewhere else in the image**.
  - Sometimes in a small neighbourhood around it.
  - What about using these similar patches together and **find their average**?
    - For that particular window, that is fine.

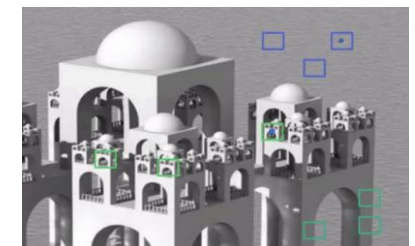
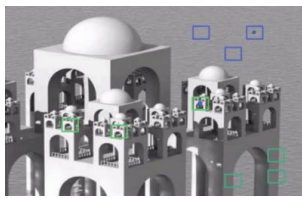


Image courtesy of online course at Coursera  
(<https://www.coursera.org/learn/image-processing>)

## Theory




- The **blue patches** in the image looks the similar.
- **Green patches** looks similar.
- So, we take a pixel.
  - Take small window around it. *จุดที่เลือก*
  - **Search for similar windows** in the image. *หาพื้นที่ในภาพที่คล้ายกัน*
  - **Average all the windows** and replace the pixel with the **result** we got. *เอาค่าเฉลี่ยมา Avg*
- This method is **Non-Local Means Denoising**. *เอาที่คล้ายมาใช้*
  - It **takes more time** compared to blurring techniques we saw earlier.
  - But its **result** is very **good**.
  - More details and online demo can be found at [http://www.ipol.im/pub/art/2011/bcm\\_nlm/](http://www.ipol.im/pub/art/2011/bcm_nlm/)  
(It has the details, online demo, etc. Highly recommended to visit.).
- For **color images**, image is converted to **CIELAB colorspace** and then it **separately denoise** L and AB components.

## Image Denoising in OpenCV

- OpenCV provides **four variations** of this technique.
  - **cv2.fastNlMeansDenoising()** <sup>Non-local</sup>  
works with a **single** **grayscale** image
  - **cv2.fastNlMeansDenoisingColored()**  
works with a **color** image. **single**
  - **cv2.fastNlMeansDenoisingMulti()** <sup>many frame</sup>  
works with **image sequence** (grayscale images)
  - **cv2.fastNlMeansDenoisingColoredMulti()**  
same as above, but for **color** images.
- We will demonstrate 2 and 3 here.
  - Rest is left for you.

# Image Denoising in OpenCV

- Common arguments are:
  - h**: parameter deciding filter strength.  
Higher h value removes noise better,  
but removes details of image also. (10 is ok)
  - hForColorComponents**: same as h, but for color images only.  
(normally same as h)
  - templateWindowSize**: should be odd. (recommended 7)
  - searchWindowSize**: should be odd. (recommended 21)



A diagram showing the parameters of the `cv2.fastNlMeansDenoisingColored` function. The function signature is: `dst = cv2.fastNlMeansDenoisingColored(img, None, 10, 10, 7, 21)`. The parameters are mapped to handwritten notes in Burmese:

- `img`: points to the input image.
- `None`: points to the `h` parameter, with a note "အရွယ်ခံမရှိ" (No size).
- `10`: points to the `h` parameter, with a note "အရွယ်ခံမရှိ" (No size).
- `10`: points to the `h` parameter, with a note "အရွယ်ခံမရှိ" (No size).
- `7`: points to the `h` parameter, with a note "အရွယ်ခံမရှိ" (No size).
- `21`: points to the `h` parameter, with a note "အရွယ်ခံမရှိ" (No size).
- `dst`: points to the output image, with a note "အရွယ်ခံမရှိ" (No size).

Demonstration of `cv2.fastNlMeansDenoisingColored()`  
It is used to **remove noise** from color images.  
**Noise** is expected to be **gaussian**.

```
import numpy as np
import cv2
from matplotlib import pyplot as plt
```

input image  
has a gaussian noise  
of  $\sigma = 25$ .

```
img = cv2.imread('die.jpg')
```

```
gaussian = cv2.GaussianBlur(img, (5, 5), 0)
```

```
median = cv2.medianBlur(img, 5)
```

```
dst = cv2.fastNlMeansDenoisingColored(img, None, 10, 10, 7, 21)
```

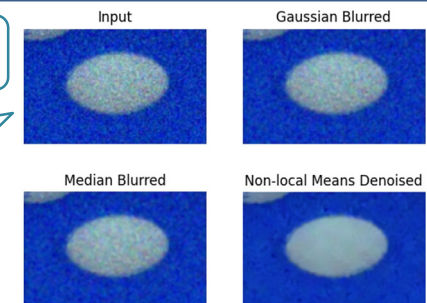
```
plt.subplot(221), plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.title('Input'), plt.axis('off')
```

```
plt.subplot(222),plt.imshow(cv2.cvtColor(gaussian,cv2.COLOR_BGR2RGB))
plt.title('Gaussian Blurred'), plt.axis('off')
```

```
plt.subplot(223),plt.imshow(cv2.cvtColor(median,cv2.COLOR_BGR2RGB))
plt.title('Median Blurred'), plt.axis('off')
```

```
plt.subplot(224), plt.imshow(cv2.cvtColor(dst, cv2.COLOR_BGR2RGB))
plt.title('Non-local Means Denoised'), plt.axis('off')-
```

```
plt.show()
```



Filter strength

### hForColorComponents

Output image

searchWindowSize

templateWindowSize

Demonstration of `cv2.fastNlMeansDenoisingMulti()`  
- Now we will apply the same method to a video.

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

cap = cv2.VideoCapture('vtest.avi')

# create a list of first 5 frames
img = [cap.read()[1] for i in range(5)]

# convert all to grayscale
gray = [cv2.cvtColor(i, cv2.COLOR_BGR2GRAY) for i in img]

# convert all to float64
gray = [np.float64(i) for i in gray]

# create a noise of variance 25
noise = np.random.randn(*gray[1].shape)*10

# Add this noise to images
noisy = [i+noise for i in gray]

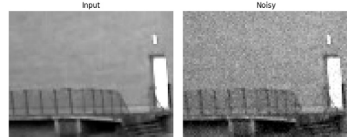
# Convert back to uint8
noisy = [np.uint8(np.clip(i,0,255)) for i in noisy]
```

វីដេអូ noisy

gray[2]



noisy[2]



Demonstration of `cv2.fastNlMeansDenoisingMulti()`

- The **first argument** is the list of **noisy frames**.
- **Second argument** `imgToDenoiseIndex` specifies which **frame** we need to **denoise**. For that we pass the **index** of frame in our input list.
- **Third** is the **temporalWindowSize** which specifies the **number of nearby frames** to be used for **denoising**. It should be odd.

```
# Median filter
median = cv2.medianBlur(noisy[2],5)
```

```
# Denoise 3rd frame
# considering all the 5 frames
dst = cv2.fastNlMeansDenoisingMulti(noisy, 2, 5, None, 7, 7, 35)
```

list of noisy frames

frame to be denoised

number of nearby frames used for dennoising

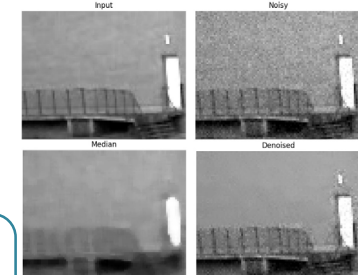
Filter strength, h

templateWindowSize

Output image

```
plt.subplot(221),plt.imshow(gray[2],'gray')
plt.title('Input'), plt.axis('off')
plt.subplot(222),plt.imshow(noisy[2],'gray')
plt.title('Noisy'), plt.axis('off')
plt.subplot(223),plt.imshow(median,'gray')
plt.title('Median'), plt.axis('off')
plt.subplot(224),plt.imshow(dst,'gray')
plt.title('Denoised'), plt.axis('off')
plt.show()
```

A total of `temporalWindowSize` frames are used where central frame is the frame to be denoised.  
For example, you passed a list of 5 frames as input.  
Let `imgToDenoiseIndex = 2` and `temporalWindowSize = 5`.  
Then **frame-0**, **frame-1**, **frame-2**, **frame-3**, and **frame-4** are used to denoise frame-2.



It takes considerable amount of time for computation.

## Image Inpainting

អាក្រក់ ដ៏ស្អាត

- Goals
  - You will **know** vision not deep learning
  - Learn how to remove small noises, strokes, etc. in old photographs by a method called **inpainting**.
  - See inpainting functionalities in OpenCV.



## Image Inpainting

ឧបករណ៍កែតម្រូវរូបភាព

- Most of you will have some **old degraded photos** at your home with some **black spots**, some **strokes**, etc. on it.
  - Have you ever thought of restoring it back?
- We **can't simply erase them** in a paint tool because it will simply replace black structures with white structures which is of no use.
- In these cases, a technique called **image inpainting** is used.
- The basic idea is simple:
  - Replace those bad marks **with its neighbouring pixels** so that it looks like the neighbourhood.
  - Consider the image shown below (taken from Wikipedia):



## Image Inpainting



- Several algorithms were designed for this purpose.

- OpenCV provides two of them.
- Both can be accessed by the same function, `cv2.inpaint()`.

1. First algorithm is based on the paper “An Image Inpainting Technique Based on the Fast Marching Method” by Alexandru Telea in 2004.

- It is based on Fast Marching Method.

- Consider a region in the image to be inpainted.

- Algorithm starts from the boundary of this region and goes inside the region gradually filling everything in the boundary first.
- It takes a small neighbourhood around the pixel on the neighbourhood to be inpainted.

## Image Inpainting



- This pixel is replaced by normalized weighted sum of all the known pixels in the neighbourhood.

- Selection of the weights is an important matter.
- More weightage is given to those pixels lying near to the point, near to the normal of the boundary and those lying on the boundary contours.

- Once a pixel is inpainted, it moves to next nearest pixel using Fast Marching Method.

- FMM ensures those pixels near the known pixels are inpainted first, so that it just works like a manual heuristic operation.
- This algorithm is enabled by using the flag, `cv2.INPAINT_TELEA`.

In physics, the Navier–Stokes equations are certain partial differential equations which describe the motion of viscous fluid substances, named after French engineer and physicist Claude-Louis Navier and Anglo-Irish physicist and mathematician George Gabriel Stokes.

## Image Inpainting



2. Second algorithm is based on the paper “Navier-Stokes, Fluid Dynamics, and Image and Video Inpainting” by Bertalmio, Marcelo, Andrea L. Bertozzi, and Guillermo Sapiro in 2001.

- This is based on fluid dynamics and utilizes partial differential equations.

- Basic principle is heuristic.

- It first travels along the edges from known regions to unknown regions (because edges are meant to be continuous).
- It continues isophotes (lines joining points with same intensity, just like contours joins points with same elevation) while matching gradient vectors at the boundary of the inpainting region.

- For this, some methods from fluid dynamics are used.

- Once they are obtained, color is filled to reduce minimum variance in that area.

- This algorithm is enabled by using the flag, `cv2.INPAINT_NS`.

### Demonstration of `cv2.inpaint()`

- Input image is degraded with some black strokes (added manually).
- The black strokes can be created with Paint tool.
- We need to create a mask of same size as that of input image, where
- non-zero pixels corresponds to the area which is to be inpainted.

```
import numpy as np
import cv2
from matplotlib import pyplot as plt
```

```
img = cv2.imread('messi 2.jpg')
mask = cv2.imread('mask2.png', 0)
```

```
# FMM based image inpainting by Telea
dst1 = cv2.inpaint(img, mask, 3, cv2.INPAINT_TELEA)
```

```
# Navier-Stokes based image inpainting
dst2 = cv2.inpaint(img, mask, 3, cv2.INPAINT_NS)
```

```
plt.subplot(221), plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.title('Input'), plt.axis('off')

plt.subplot(222), plt.imshow(cv2.cvtColor(mask, cv2.COLOR_BGR2RGB))
plt.title('Mask'), plt.axis('off')

plt.subplot(223), plt.imshow(cv2.cvtColor(dst1, cv2.COLOR_BGR2RGB))
plt.title('TELEA'), plt.axis('off')

plt.subplot(224), plt.imshow(cv2.cvtColor(dst2, cv2.COLOR_BGR2RGB))
plt.title('NS'), plt.axis('off')

plt.show()
```



inpaintRadius  
(Neighborhood around a pixel to inpaint)  
If the regions to be inpainted are thin,  
smaller values produce better results (less blurry).

INPAINT\_NS (Navier-Stokes based method)  
or  
INPAINT\_TELEA (Fast marching based method)



## Additional Resources

- Bertalmio, Marcelo, Andrea L. Bertozzi, and Guillermo Sapiro.  
“Navier-stokes, fluid dynamics, and image and video inpainting.”  
In Computer Vision and Pattern Recognition, 2001. CVPR 2001.  
Proceedings of the 2001 IEEE Computer Society Conference on,  
vol. 1, pp. I-355. IEEE, 2001.
- Telea, Alexandru.  
“An image inpainting technique based on the fast marching method.”  
Journal of graphics tools 9.1 (2004): 23-34.

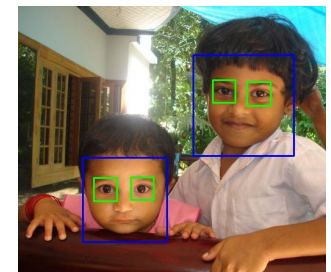
# BREAK

## OpenCV

## Object Detection

## Face Detection using Haar Cascades

- Goals
  - You will ...
    - See the basics of **face detection**  
using Haar Feature-based Cascade Classifiers
    - Extend the same for **eye detection**, etc.



## Face Detection using Haar Cascades

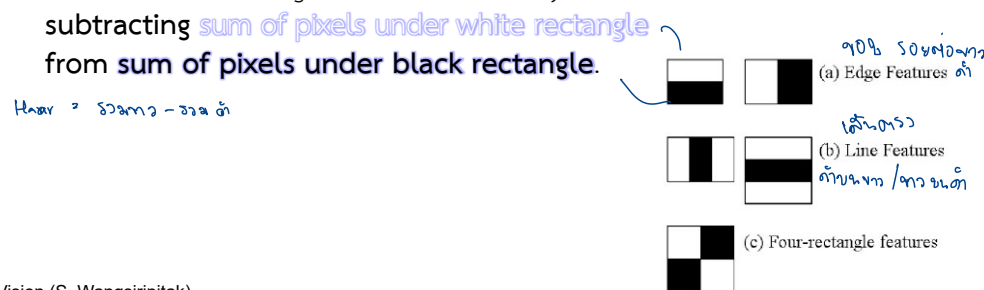
### Basics

- Object Detection using **Haar feature-based cascade classifiers** is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, “**Rapid Object Detection using a Boosted Cascade of Simple Features**” in 2001.
  - It is a **machine learning based approach** where a **cascade function** is **trained** from a lot of **positive** and **negative images**.
  - It is then used to detect objects in other images.

## Face Detection using Haar Cascades

### Basics

- Here we will work with **face detection**. *ໃຈກິດຂອງຜູ້ສະໜອງ ແລະ ບໍ່ສະໜອງ*
- Initially, the algorithm needs a lot of **positive images** (images of **faces**) and **negative images** (images **without faces**) to **train the classifier**.
- Then we need to **extract features** from it.
  - For this, **haar features** shown in below image are used.
  - They are just like our convolutional kernel.
  - Each feature is a single value obtained by **subtracting sum of pixels under white rectangle from sum of pixels under black rectangle**.



## Face Detection using Haar Cascades

### Basics

- Now **all possible sizes and locations** of **each kernel** is used to calculate plenty of features.
  - Just imagine how much **computation** it needs?
  - Even a 24x24 window results over 160000 features.
  - For **each feature calculation**, we need to find **sum of pixels under white and black rectangles**. *ສະໜອງ*
  - To **solve** this, they introduced the **integral images**. *ອັດຕະໂນມັດສະໜອງ*
    - It simplifies calculation of sum of pixels, how large may be the number of pixels, to an **operation involving just four pixels**.
    - It makes things **super-fast**.

## Face Detection using Haar Cascades

### Basics

- But among all these features we calculated, **most** of them are **irrelevant**.
  - For example, consider the image below.
  - Top row shows **two good features**.
  - The **first feature** selected seems to focus on the property that the region of the **eyes** is often **darker than** the region of the **nose and cheeks**.
  - The **second feature** selected relies on the property that the **eyes** are **darker than** the **bridge of the nose**.
  - But the **same windows** applying on **cheeks** or any **other place** is **irrelevant**.



## Face Detection using Haar Cascades

### Basics

- For this, we **apply** each and every **feature** on all the **training images**.
  - For **each feature**, it finds the **best threshold** which will **classify** the **faces** to **positive** and **negative**.
  - But obviously, there will be errors or misclassifications.
  - We select the **features** with **minimum error rate**, which means they are the features that **best** classifies the face and non-face images.
    - The process is **not as simple as this**.
    - Each image is given an **equal weight** in the **beginning**.
    - After each classification, **weights of misclassified images** are **increased**.  
*เพิ่มน้ำหนัก weight ของภาพที่ misclassified*
    - Then again same process is done.
    - New error rates are calculated. Also new weights.
    - The process is **continued until** required **accuracy** or **error rate** is achieved or required **number of features** are found.

## Face Detection using Haar Cascades

### Basics

- Final classifier is a weighted sum of these **weak classifiers**.  
*อาจสามารถนำ weak classifiers มาหาค่ารวม*
  - It is called weak because **it alone can't classify the image**, but together with others forms a strong classifier.
  - The paper says even **200** features provide detection with **95%** accuracy.
  - Their final setup had around 6000 features. *ไม่จำเป็นต้องใช้ทั้งหมด*  
(A reduction from 160000+ features to 6000 features. A big gain).
- So now you take an image.
- Take each 24x24 window.
- Apply 6000 features to it.
- Check if it is face or not.
  - A little **inefficient** and **time consuming**.
  - Authors have a good **solution** for that.

## Face Detection using Haar Cascades

### Basics

- In an image, **most** of the image **region** is **non-face** region.
  - So, it is a better idea to have a **simple method** to **check if a window is not a face region**.
  - If it is not, discard it in a single shot. Don't process it again.
  - Instead focus on region where there can be a face. *ถ้า 1 ภาพมี 3x3x4*
  - This way, we can find more time to check a possible face region.
- For this they introduced the concept of **Cascade of Classifiers**. *step 1 ผ่าน 1 → 2 → 3*
- Instead of applying all the 6000 features on a window, **group the features** into **different stages of classifiers** and apply one-by-one.  
(Normally **first few stages** will contain very **less number of features**).
  - If a window fails the first stage, discard it.  
We don't consider remaining features on it.
  - If it passes, apply the second stage of features and continue the process.
- The window which passes all stages is a face region.

## Face Detection using Haar Cascades

### Basics

- Authors' detector had **6000+ features** with **38 stages** with 1, 10, 25, 25 and 50 features in first five stages.  
(Two features in the above image is actually obtained as the best two features from Adaboost). *พบ 10 feature*
  - According to authors, on an average, **10 features** out of 6000+ are **evaluated per sub-window**.
- So, this is a simple intuitive explanation of how **Viola-Jones face detection** works. (Read paper for more details)



## Haar-cascade Detection in OpenCV

- OpenCV comes with a **trainer** as well as **detector**.
  - If you want to train your own classifier for any object like car, planes, etc. you can use OpenCV to create one.
- Here we will deal with **detection**.
- OpenCV already contains many **pre-trained classifiers** for face, eyes, smile, etc.
  - Those XML files are stored in `opencv/data/haarcascades/` folder.
- Let's create face and eye detector with OpenCV.

*MinNeighbors*  
 จำนวนเพื่อนบ้านแต่ละรูปสี่เหลี่ยมตัวเลือกควรเก็บไว้ ส่งผลต่อคุณภาพของ  
 ใบหน้าที่ตรวจพบ ค่าที่สูงขึ้นส่งผลให้ตรวจจับได้น้อยลงแต่มีคุณภาพสูงขึ้น

### Demonstration of `cv2.CascadeClassifier()` Face and eye detector

*premodel*  
 load the required XML classifiers

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')
```

load the input image (or video) in grayscale mode

```
img = cv2.imread('human and phone.jpeg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

**scaleFactor** : Image size reduction factor at each image scale.  
 E.g., 1.03 reduces size by 3 % (a small step).  
 → Increase the chance of a matching size (but expensive).  
*เลือกขนาดที่น้อยลงเรื่อยๆ*

**minNeighbors** : How many neighbors each candidate rectangle should have to retain it.  
 Affect the quality of the detected faces  
 Higher value results in less detections but with higher quality.  
*จาก 0-2 ในหน้า  
 มาก = 1 หน้าน้อย  
 ไม่ได้นอกจากที่อื่น*

```
# find the faces in the image
faces = face_cascade.detectMultiScale(gray, 1.3, 5)
```

**Rect(x,y,w,h)**

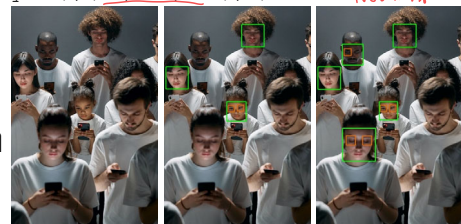
```
for (x,y,w,h) in faces:
    img = cv2.rectangle(img, (x,y), (x+w,y+h), (0,255,0), 2)
    roi_gray = gray[y:y+h, x:x+w]
    roi_color = img[y:y+h, x:x+w]
    eyes = eye_cascade.detectMultiScale(roi_gray)
    for (ex,ey,ew,eh) in eyes:
        cv2.rectangle(roi_color, (ex,ey), (ex+ew,ey+eh), (0,0,255), 2)
```

create a ROI for the face

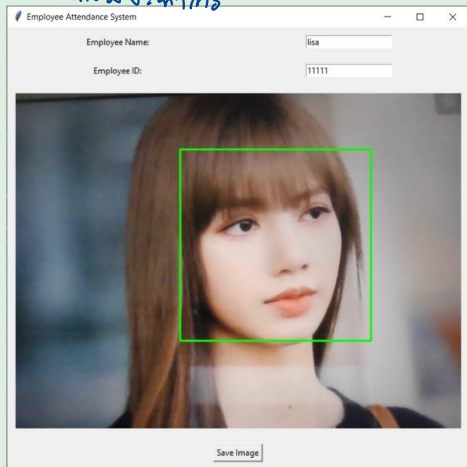
detectMultiScale(gray, 1.04, 5)

apply eye detection on this ROI (since eyes are always on the face)

*color*



๑) ให้ผู้ใช้ทำเครื่องหมายชื่อ และรหัสพนักงาน จากหน้าต่าง save image  
โปรแกรมจะทำการ



๑) เพิ่มใบหน้าพนักงาน

๓) ถ่ายภาพใบหน้าพร้อมทั้งทำ roi เพื่อเก็บ  
บันทึกข้อมูลใบหน้าของพนักงานแต่ละคน



# FACE RECOGNITION

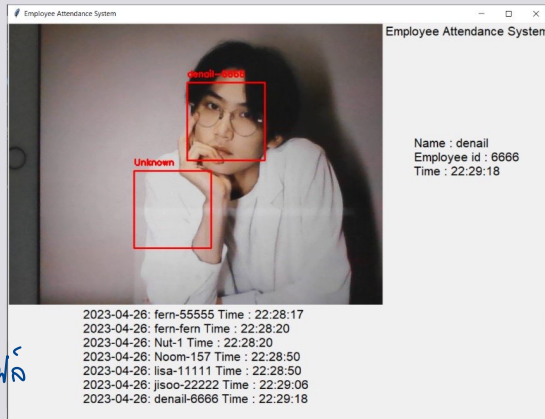
ตรวจจับใบหน้าของพนักงานที่เข้าทำงาน  
พร้อมเก็บ log เวลาที่เข้าทำงานในแต่ละวัน

ใช้ในกรณีที่ระบบจับใบหน้า

★ Local Binary Patterns Histograms (LBPH)

★ haarcascades ใช้ในกรณีค้นหาใบหน้า

★ pandas ใช้ในกรณีที่จัดเก็บข้อมูล ลงไฟล์



# LBPH

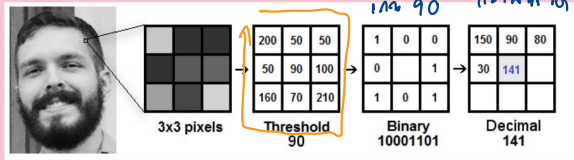
## LOCAL BINARY PATTERNS HISTOGRAMS

แบ่งภาพออกเป็นส่วนๆ ขนาด 3x3 พิกเซล และคำนวณค่าไบนารีความยาว 8 บิต จากค่าสีเทาของแต่ละพิกเซล โดยใช้ค่าสีเทาของพิกเซลตรงกลางเป็นค่า threshold หากค่าสีเทาของพิกเซลนั้นๆ มีค่าสูงกว่าพิกเซลตรงกลาง จะแทนบิตนั้นด้วยค่า 1 ถ้าน้อยกว่า จะแทนค่าบิตนั้นด้วย 0 แล้วนำค่าไบนารีที่ได้มาทำการสร้างเป็นเวกเตอร์เพื่อแทนใบหน้า

จากนั้น จะนำค่าที่ได้ไปสร้างเป็น Histogram แล้วนำมาเปรียบเทียบกัน  
จุดเด่น

สามารถทนต่อการเปลี่ยนแปลงระดับของแสงในภาพได้ดี เพราะ มาจากค่าสีเทา ผลรวมของแต่ละ kernel

เพื่อสร้างลักษณะที่เปลี่ยนแปลง ก็จะเกิดซ้ำ kernel ทำให้กราฟที่ได้ก็จะมีลักษณะคล้ายๆ กันอยู่



11100010

896

LBPB (อีลโดแกรมรูปแบบไบนารีท้องถิ่น)

**LBPB | อัลกอริทึม LBPB การจดจำใบหน้า**

เริ่มต้นด้วยการวิเคราะห์เมทริกซ์ที่แสดงถึงชั้นส่วนของภาพ และเมื่อคุณเรียนรู้ก่อนหน้านี้ รูปภาพจะแสดงในรูปแบบเหล่านี้ ในตัวอย่างนี้ เรามีสามแถวและสามคอลัมน์ และจำนวนพิกเซลทั้งหมดคือเก้า ให้เลือกพิกเซลกลางที่มี 1 สีดำแปด และใช้เงื่อนไข ถ้าค่ามากกว่าหรือเท่ากับ 8 ผลลัพธ์จะเป็น '1' มิฉะนั้น ถ้าค่าน้อยกว่า 8 ผลลัพธ์จะเป็นศูนย์ หลังจากใช้ครีมนวดผม เมทริกซ์จะมีลักษณะดังนี้

งานของ LBPB

การคำนวณพื้นฐานของอัลกอริทึมนี้คือการใช้เงื่อนไข โดยเลือกองค์ประกอบตรงกลางของเมทริกซ์ ตอนที่เรากำลังสร้างค่าไบนารี ค่าไบนารี = 11100010 อัลกอริทึมจะเริ่มใช้เงื่อนไขจากองค์ประกอบมุมบนซ้าย ขึ้นไปยังองค์ประกอบที่ 1 ของแถวที่ 2 คิดเหมือนการสร้างวงกลมแบบนี้

การคำนวณ | อัลกอริทึมการจดจำใบหน้า LBPB

หลังจากแปลงค่าไบนารีเป็นค่าทศนิยมแล้ว เราจะได้ค่าทศนิยม = 226 แสดงว่าพิกเซลเหล่านี้รอบค่ากลางเท่ากับ 226

อัลกอริทึมนี้มีประสิทธิภาพเมื่อเกิดปัญหาค่าคุณไสยของพิกเซลจะเพิ่มขึ้น ค่าที่สูง ภาพก็จะยิ่งสว่าง และเมื่อค่ายิ่งต่ำ ภาพก็จะยิ่งมืดลง ด้วยเหตุนี้ อัลกอริทึมนี้จึงให้ผลลัพธ์ที่ติดกับภาพที่สว่างและมืด เพราะเมื่อภาพสว่างขึ้นหรือมืดลง พิกเซลทั้งหมดในพื้นที่ใกล้เคียงจะเปลี่ยนไป หลังจากวางแสงบนภาพแล้วเมทริกซ์จะมีลักษณะดังนี้ หลังจากใช้เงื่อนไขข้างต้นแล้ว เราจะได้ค่าไบนารีเหมือนกับด้านบน นั่นคือ 11100010

พิกเซลเมตริก | อัลกอริทึม LBPB การจดจำใบหน้า

ลองพิจารณาภาพอื่นที่นี่ เพื่อให้เข้าใจได้ดียิ่งขึ้นว่าอัลกอริทึมจะจดจำใบหน้าของบุคคลได้อย่างไร

วิธีจดจำใบหน้า | อัลกอริทึม LBPB การจดจำใบหน้า

เรามีภาพใบหน้าอยู่ที่นี่ และสิ่งที่อัลกอริทึมจะทำคือสร้างสี่เหลี่ยมหลายช่อง ดังที่คุณเห็นที่นี่ และในแต่ละช่องสี่เหลี่ยมเหล่านี้ เรามีตัวแทนของรูปก่อนหน้าคือแสง ตัวอย่างเช่น สี่เหลี่ยมจัตุรัสนี้ไม่ได้แสดงเพียงพิกเซลเดียว แต่ถูกกำหนดด้วยหลายพิกเซลที่มีสามแถวและสี่คอลัมน์ สามคอลัมน์เท่ากับ 12 พิกเซลทั้งหมด ในสี่เหลี่ยมจัตุรัสเหล่านี้ ในแต่ละสี่เหลี่ยมที่มี 12 พิกเซล จากนั้นเราก็ใช้เงื่อนไขนั้นกับแต่ละเงื่อนไข พิจารณาพิกเซลกลาง

ขั้นตอนต่อไปคือการสร้างอีลโดแกรมซึ่งเป็นแนวคิดของสถิติที่จะนับจำนวนครั้งที่แต่ละสีปรากฏในตารางแต่ละช่อง นี่คือการแสดงอีลโดแกรม

สร้างอีลโดแกรม

ตัวอย่างเช่น หากค่า 110 ปรากฏขึ้น 50 ครั้ง แท่งแบบนี้จะถูกสร้างขึ้นด้วยขนาดนี้เท่ากับ 50 ถ้า 201 ปรากฏขึ้น 110 ครั้ง และแถบอื่นๆ จะถูกสร้างขึ้นในอีลโดแกรมนี้ด้วยขนาดเท่ากับ 100 จากการเปรียบเทียบ ของอีลโดแกรม อัลกอริทึมจะสามารถระบุขอบและมุมของภาพได้ ตัวอย่างเช่น ในสี่เหลี่ยมจัตุรัสแรกนี้ เราไม่มีข้อมูลเกี่ยวกับใบหน้าของบุคคลนั้น ดังนั้นอีลโดแกรมจะแตกต่างจากสี่เหลี่ยมจัตุรัสอื่นที่มีเส้นขอบของใบหน้า กล่าวโดยย่อ อัลกอริทึมรู้วาลูอีลโดแกรมใดแสดงถึงเส้นขอบ และอีลโดแกรมใดแสดงถึงลักษณะเด่นหลักของบุคคล เช่น สีของดวงตา รูปร่างของปาก และอื่นๆ

นี่คือทฤษฎีพื้นฐานของอัลกอริทึมนี้ ซึ่งมีพื้นฐานมาจากการสร้างและการเปรียบเทียบอีลโดแกรม



My Presenter is Mhor Lab Panda

Because of his image as a hard worker with black eyes, ~~our brand picked him to be the presenter~~. His life has been challenging, to work hard Sleep deprivation in a row but after using our goods, he was able to sleep well. able to completely rest and prepare for a new day It shows items to customers who can readily know them

I have to admit that he is a very influential person in the media. He will be able to make our beds sell very well. If you say that this product is guaranteed by him

pln