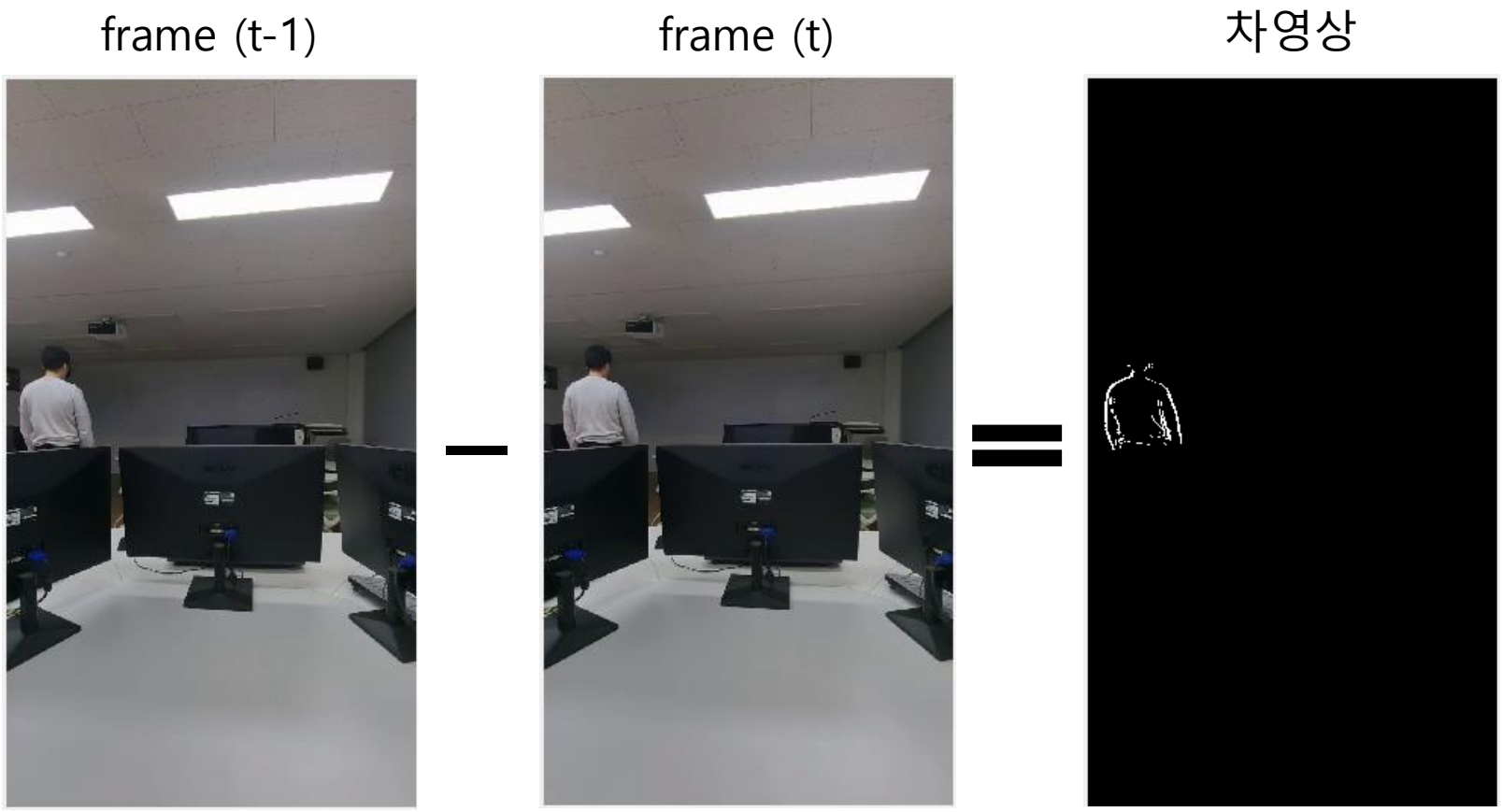
차영상(subtraction image)과
하르 캐스케이드(Haar Cascade)를 이용한 이상 검출

6.1 차영상

- 차영상
  - 두 프레임 또는 이미지 간의 화소 값 차이를 계산하여 이미지를 생성하는 법
  - 두 이미지 사이의 변화나 움직임을 감지하는데 사용됨



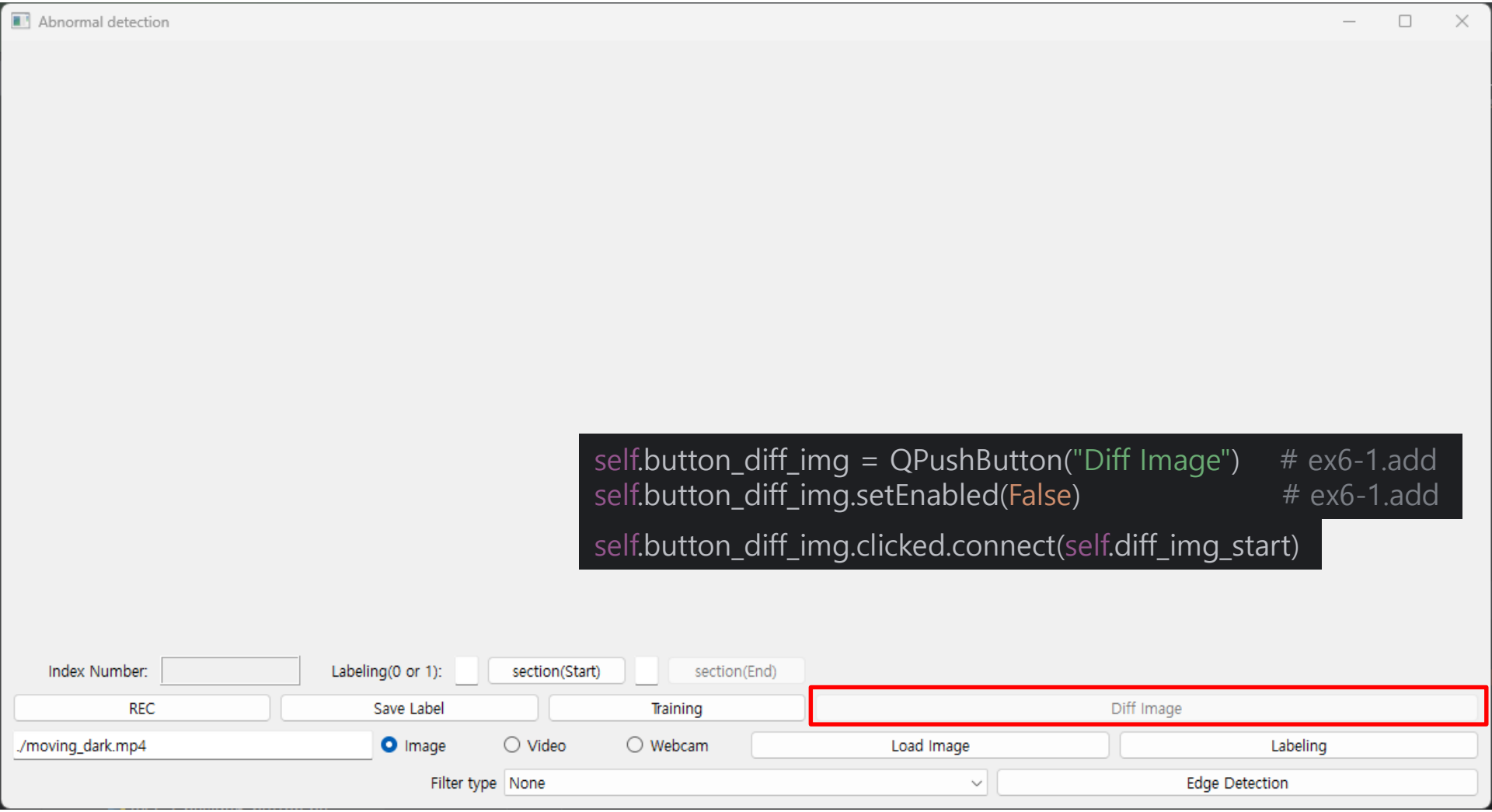
6.1 차영상

차영상의 특징

- **변화 감지:** 차영상은 두 이미지 간의 변화가 있는 영역을 쉽게 감지할 수 있도록 해주며, 배경이 고정된 상태에서 움직이는 물체가 있을 경우 그 움직임이 차영상에서 뚜렷하게 나타남
- **움직임 감지:**
  - 차영상을 통해 연속된 비디오 프레임 간의 움직임을 감지할 수 있음
  - 움직임 상황에서는 민감하게 테두리가 감지되나, 객체가 정지된 상황에서는 객체를 찾을 수 없음
  - 보안 감시 시스템, 객체 추적, 스포츠 분석 등에서 활용
- **조명 변화에 민감:** 차영상은 조명 변화에 민감하며, 조명이 급격히 변할 경우 실제 움직임이 없어도 차영상에서 변화가 감지

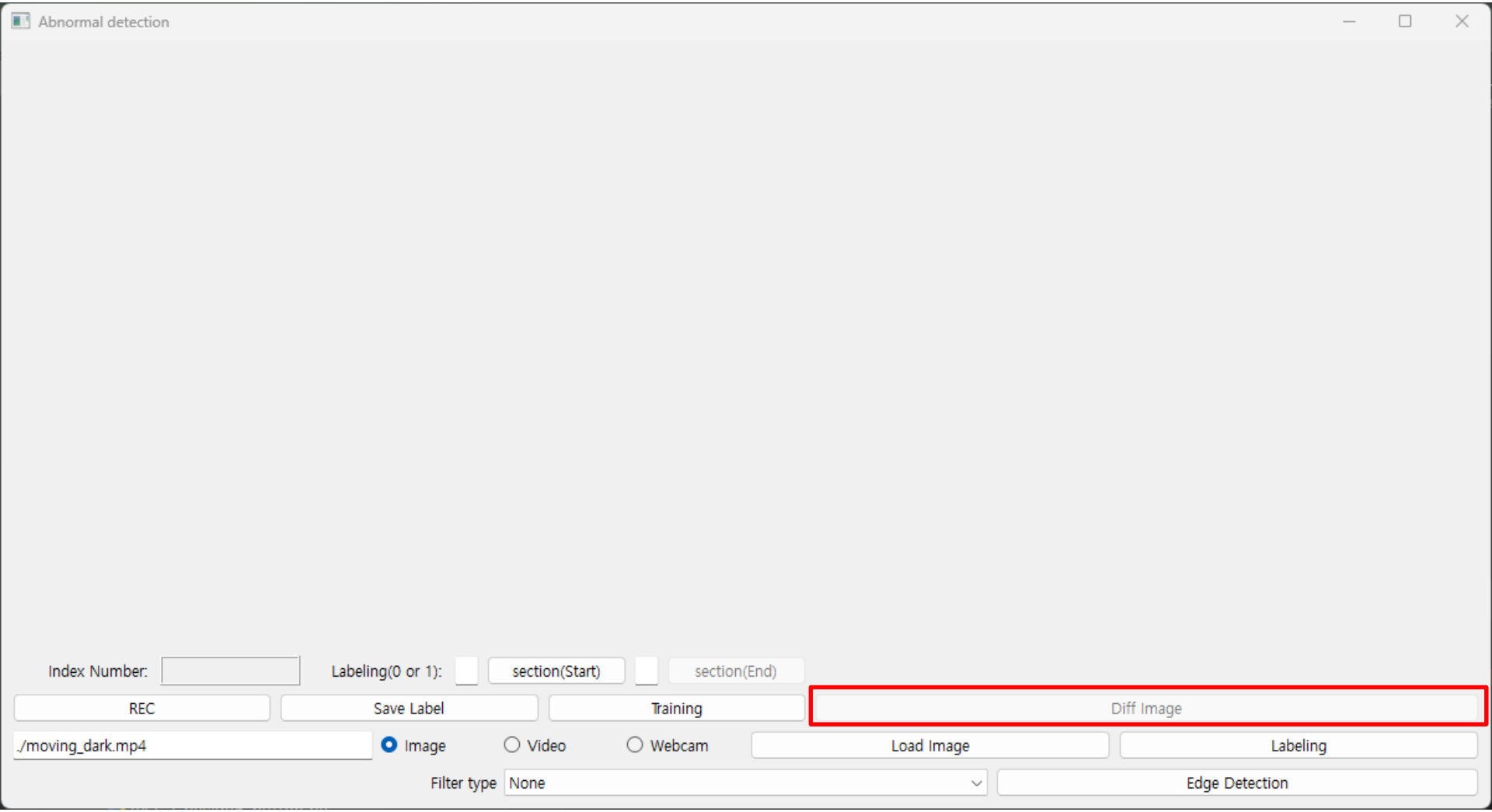
6.1 차영상

- Diff Image 버튼 배치
  - 빨강색 박스 영역 위치로 Diff Image 버튼 배치하기



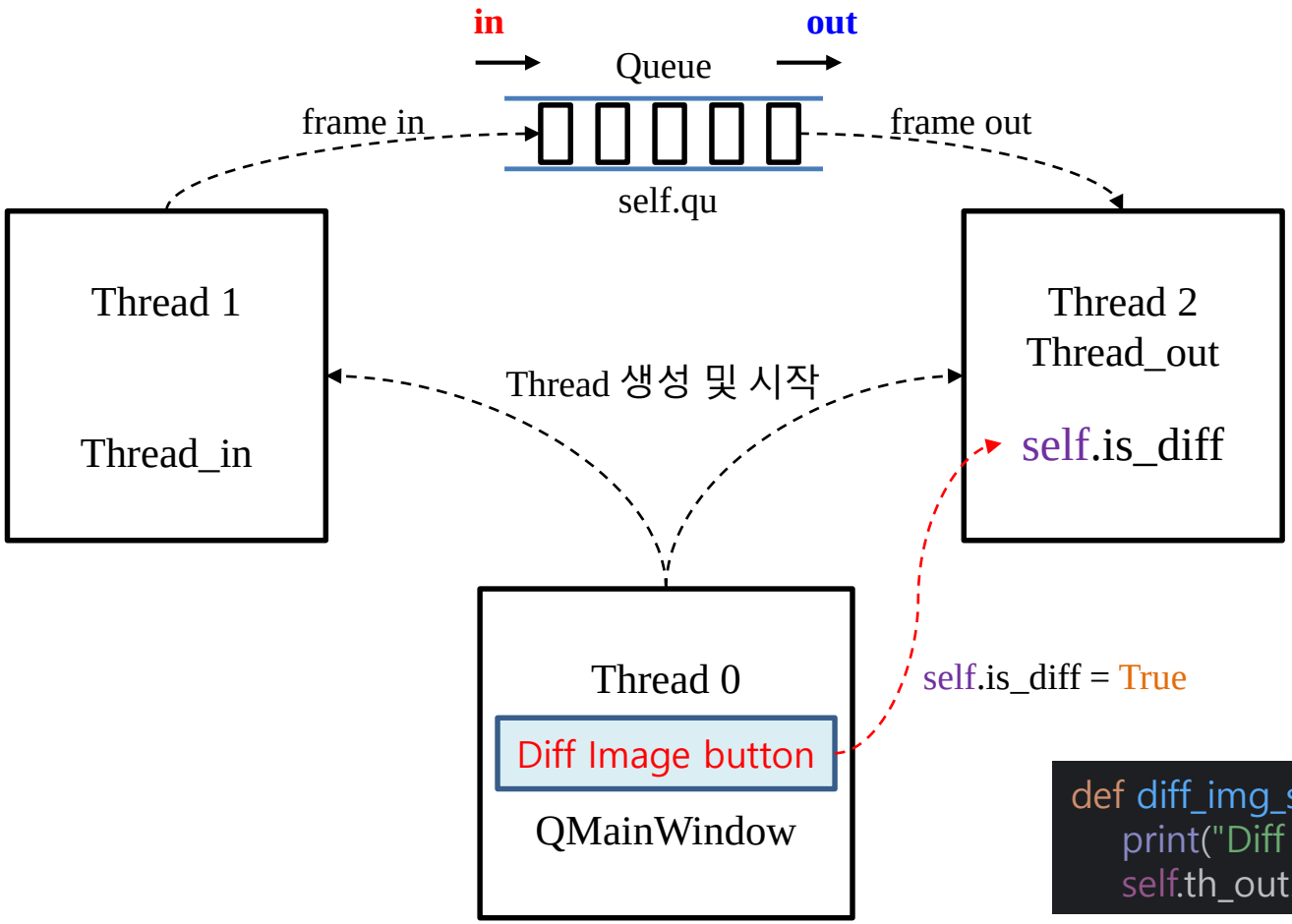
6.1 차영상

- Diff Image 버튼 동작 순서
  - video 또는 webcam 모드로 Load Image 버튼 실행 후 Diff Image 버튼 실행



6.1 차영상

- Diff Image 버튼의 동작 관계
  - Diff Image 버튼을 클릭하면 self.th_out.is_diff 변수를 제어하여 Thread_out 클래스 객체에서 차영상을 계산을 시작함




```
def diff_img_start(self):
    print("Diff image...")
    self.th_out.is_diff = True
```

6.1 차영상

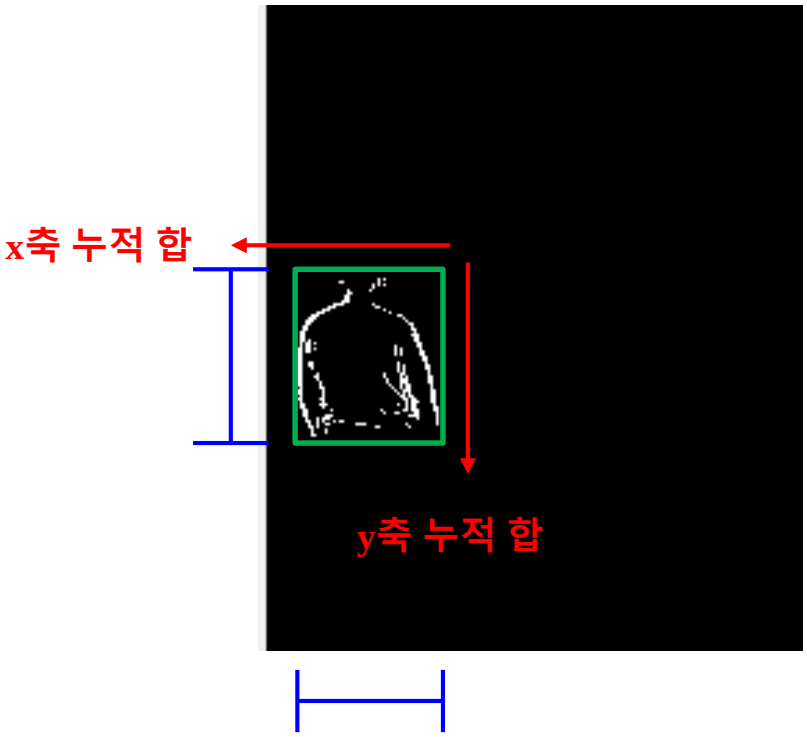
- Thread_out 클래스의 Diff Image 정의
  - def diff_img() 함수는 Thread_out 클래스의 매서드에 정의
  - Diff Image 버튼을 클릭하면 현재 frame과 이전 frame 간의 차영상을 계산하는 함수 구현
  - 차영상의 계산 결과를 threshold 함수를 통하여 이진 영상으로 보이기

```
def diff_img(self, frame):
    img = cv2.resize(frame, dsize=None, fx=0.375, fy=0.375)
    current_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    if self.pre_frame is None:
        # 현재 회색조 영상을 self.pre_frame에 저장하는 코드 작성
        return frame

    else:
        # 현재 회색조 영상과 이전 프레임간의 차영상 값 계산
        # Hint! cv2.absdiff() 함수 사용
        # cv2.threshold() 함수를 이용하여 차영상의 값이 30보다
        크면 화소를 흰색으로 표시하여 thresh 변수에 저장
        # 현재 회색조 영상을 self.pre_frame에 저장
        return thresh
```

6.1 차영상

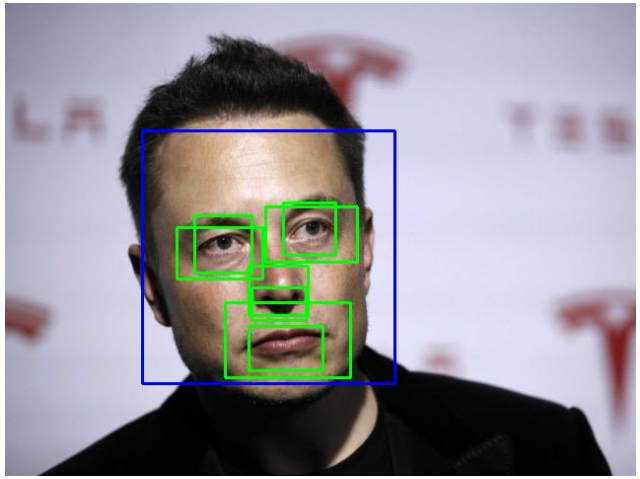
- 차영상의 객체 영역 표시하기
  - 차영상 결과의 x축 및 y축에 대한 누적 합산 값을 이용하여 객체 영역을 박스로 나타내기
  - Hint!) x축의 누적합: img.sum(axis=0)
  - Hint!) y축의 누적합: img.sum(axis=1)
  - Hint!) 특정 값의 인덱스를 찾는 방법: np.where(img == 1)



6.2 Haar cascade

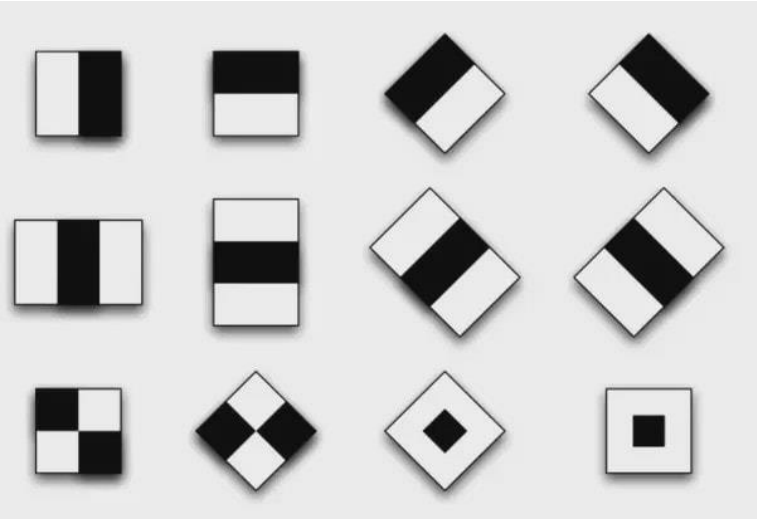
- 하르 캐스케이드(Haar cascade)
  - 객체 검출에 사용하는 기계학습 기반 검출 알고리즘으로, 얼굴 검출에 적용되어 속도와 정확도를 인정받은 기술
  - 비올라와 존스가 개발한 객체 검출 알고리즘에 기반하여 개발되었으며, 주로 얼굴, 눈, 자동차 등의 특정 객체를 이미지나 영상에서 탐지하는데 사용됨

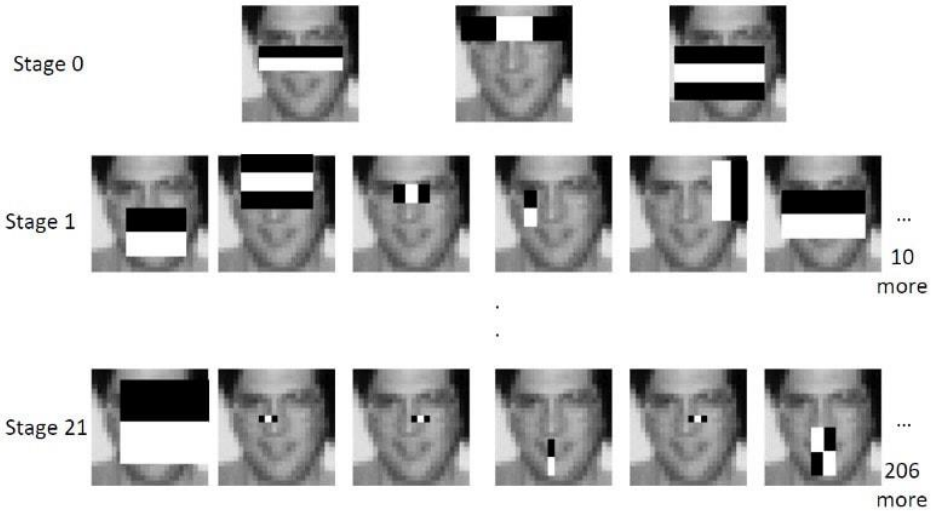
- Cascade
  - 계단식 구조 또는 연속적인 단계적 과정을 의미
  - Cascade 구조란 여러 단계의 분류기가 연속적으로 연결된 형태
    - 초기 단계에서는 객체가 아닐 가능성이 높은 영역을 빠르게 걸러내고, 후속 단계에서 더 정교하게 객체 여부를 검출하는 방식



6.2 Haar cascade

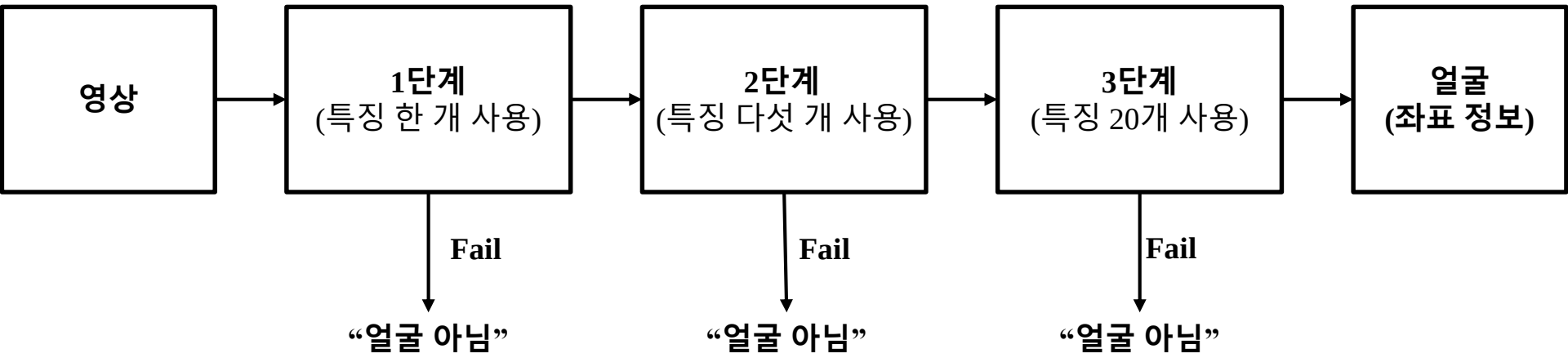
- 하르 캐스케이드(Haar cascade) 분류기
  - 여러 개의 검출기를 순차적으로 사용
  - 초기 단계에서 간단한 검출기를 적용하고, 진행할수록 복잡한 검출기를 적용
  - 단순 검출기를 통과한 후보에만 시간이 많이 걸리는 강력한 검출기가 적용
- 특징값
  - (흰색 영역의 화소값의 합) – (검은색 영역의 화소값 합)





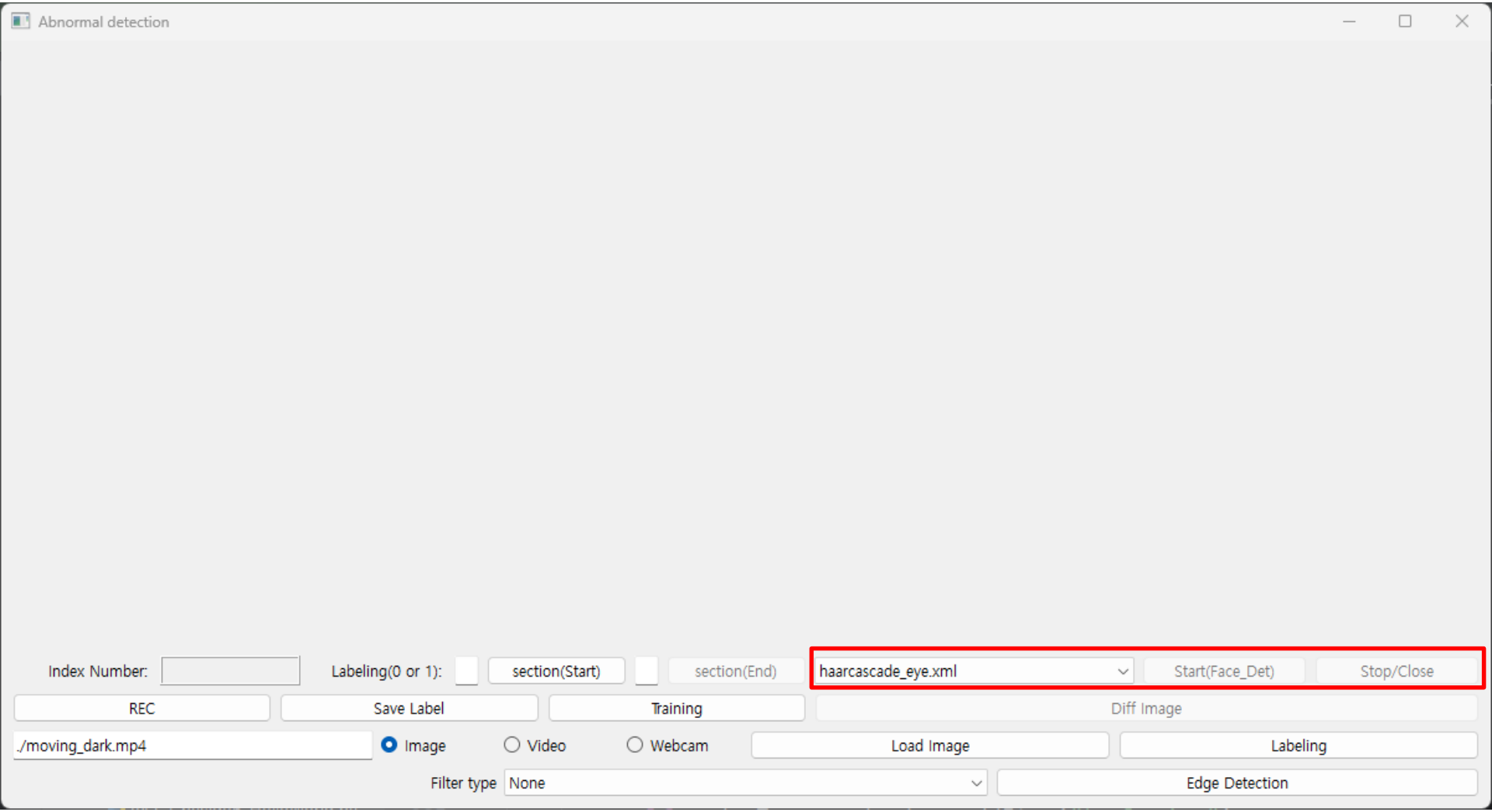
6.2 Haar cascade

- 하르 캐스케이드(Haar cascade) 분류기와 얼굴 검출



6.2 Haar cascade

- 하르 캐스케이드(Haar cascade) 분류기와 얼굴 검출 코드 적용하기



6.2 Haar cascade

- 하르 캐스케이드(Haar cascade) 분류기와 얼굴 검출 코드 적용하기
  - Window(or Form) 클래스에 콤보박스와 시작/종료 버튼 만들기
  - *layout 배치는 직접 구현해보기*

```
self.combobox_haar = QComboBox()
for xml_file in os.listdir(cv2.data.haarcascades):
    if xml_file.endswith(".xml"):
        self.combobox_haar.addItem(xml_file)
self.button_haar_start = QPushButton("Start(Face_Det)")
self.button_haar_stop = QPushButton("Stop/Close")

self.button_haar_start.setEnabled(False)
self.button_haar_stop.setEnabled(False)
```

6.2 Haar cascade

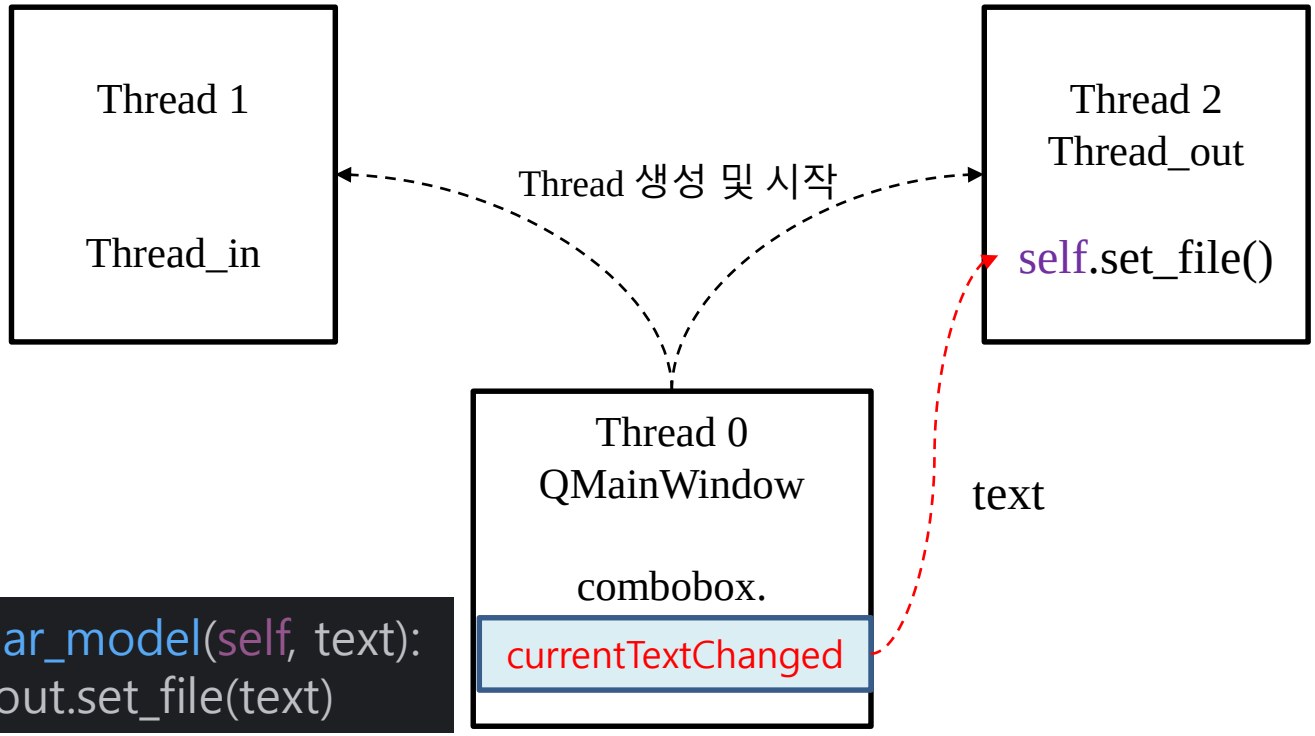
- 하르 캐스케이드(Haar cascade) 분류기와 얼굴 검출 코드 적용하기
  - Window(or Form) 클래스에 콤보박스와 시작/종료 버튼을 슬롯함수와 연결(connect)

```
self.combobox_haar.currentTextChanged.connect(self.set_haar_model)
self.button_haar_start.clicked.connect(self.start_haar)
self.button_haar_stop.clicked.connect(self.kill_thread)
```

6.2 Haar cascade

- 하르 캐스케이드(Haar cascade) 분류기와 얼굴 검출 코드 적용하기
  - Window(or Form) 클래스에 콤보박스와 시작/종료 버튼을 슬롯함수와 연결(connect)

```
def set_file(self, fname):
    self.trained_file = os.path.join(cv2.data.haarcascades, fname)
```




```
def set_haar_model(self, text):
    self.th_out.set_file(text)
```

6.2 Haar cascade

- Start(Face_Det) 버튼 구현
  - Start(Face_Det) 버튼을 비활성화
  - Stop/Close 버튼을 활성화
  - Thread_out 클래스의 set_file() 함수를 현재 콤보박스의 xml 파일로 실행
  - Thread_out 클래스의 is_haar 값을 True로 변환

```
def start_haar(self):
    self.button_haar_start.setEnabled(False)
    self.button_haar_stop.setEnabled(True)
    self.th_out.set_file(self.combobox_haar.currentText())
    self.th_out.is_haar = True
```

6.2 Haar cascade

- Stop/Close 버튼 구현
  - 기존의 kill_thread() 함수 활용
  - kill_thread() 함수 마지막에 Start(Face_Det), Stop/Close, Diff Image 버튼을 비활성화

```
self.button_haar_stop.clicked.connect(self.kill_thread)    # ex6-1.add
```

```
def kill_thread(self):
    # 기존의 kill_thread 코드는 그대로 남겨두세요
    # 밑에서부터 세 줄만 추가
    ...
    if self.labeling_capture is not None:
        self.labeling_capture.release()

    self.button_haar_start.setEnabled(False)      # ex6-1.add
    self.button_haar_stop.setEnabled(False)       # ex6-1.add
    self.button_diff_img.setEnabled(False)        # ex6-1.add
```

6.3 임계값 학습을 통한 화면 전환 구현

구현 및 실습

- (필수) 기존 코드에 다음 요소들 추가하기
  - "Diff Image", "Start(Face_Det)", "Stop/Close" 버튼 추가
  - haar cascade xml 종류를 선택하는 콤보박스 추가
- (필수) diff_img() 함수 완성 (7번 slide)
- (필수) 차영상에서 객체 영역 박스로 표시(8번 slide)
- (필수) Restart 버튼 만들기
  - 화면 전환 깜박임 문제 해결을 위한 버튼
  - 화면전환이 발생하면 더 이상 화면전환이 발생하지 않도록 하며, Restart 버튼을 클릭하면 다시 화면전환이 발생될 수 있도록 하는 버튼
- (선택1) mouse 점을 2개를 찍어서 화면에 관심영역(ROI) 사각형 그리기
  - Hint! cv2.rectangle(입력영상, 좌상단 점의 좌표, 우하단 점의좌표, 컬러표, 두께) 함수 사용
- (선택2) haar 객체탐지 결과의 중심좌표에 빨간색 점 찍기
  - Hint! cv2.circle() 함수 사용
- (선택3) ROI 영역에 객체의 중심좌표가 들어오면 화면 전환 시키기