BIGDATA & AI ANALYTICS
EXPERT COMPANY

OpenCV in Python

BIGDATA & AI ANALYTICS
EXPERT COMPANY

1. OpenCV 개요

Python 이미지 처리 라이브러리

PIL (Python Image Library)

- 주로 이미지 처리만을 위해 사용.
- 처리 성능이 상대적으로 느림

Scikit Image

- 파이썬 기반의 전반적인 컴퓨터 비전 기능 제공
- 사이파이(Scipy) 기반

OpenCV

- 오픈소스 기반의 최고 인기 컴퓨터 비전 라이브러리
- 컴퓨터 비전 기능 일반화에 크게 기여(어려운 기능도 API 몇줄로 간단하게 구현)
- C++ 기반이나 Python 도 지원(Java, C# 등 다양한 언어 지원)

OpenCV 개요

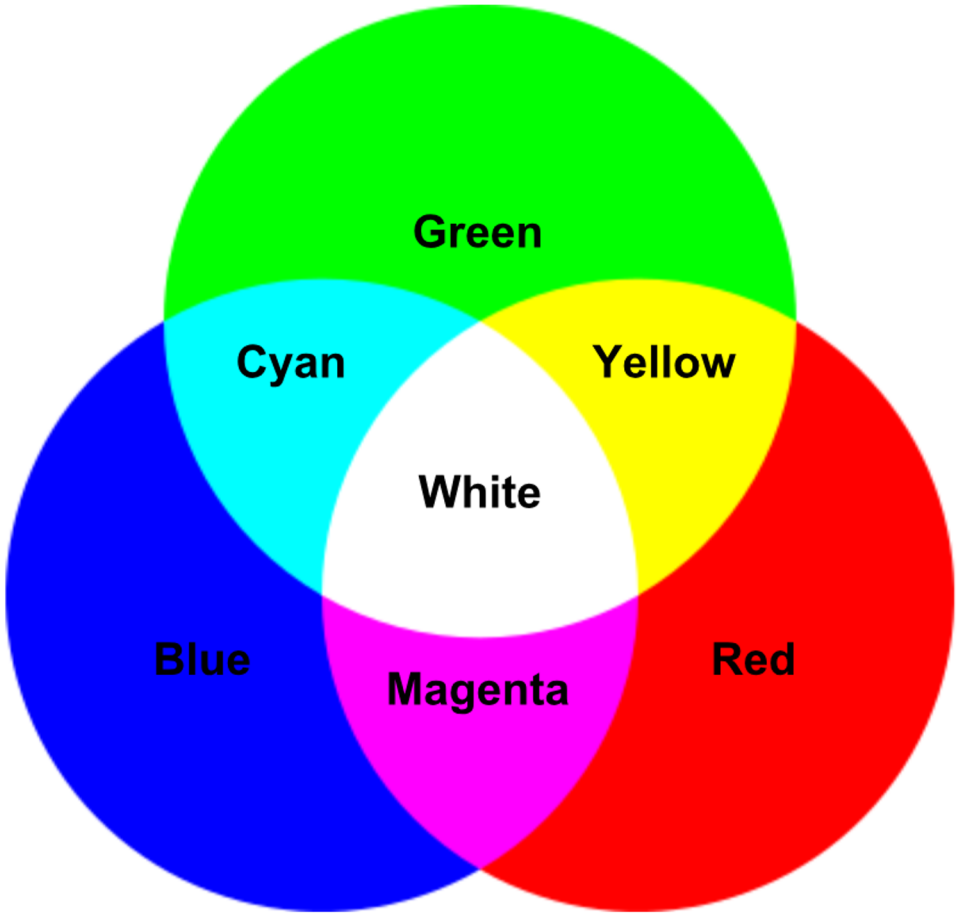
1.1 OpenCV 개요

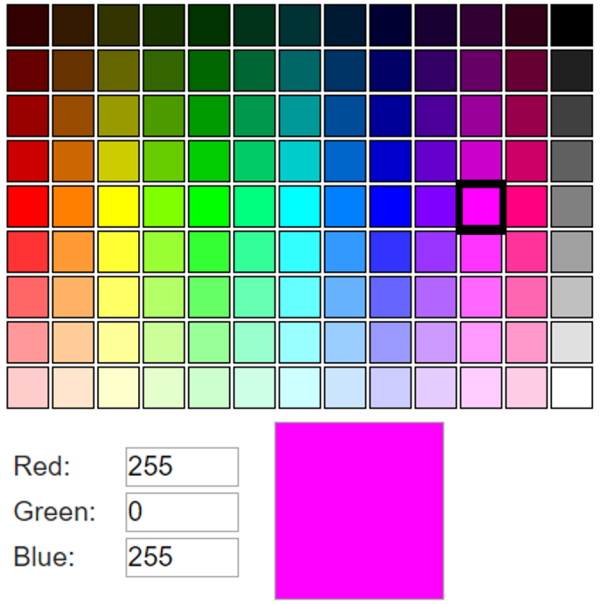
- OpenCV는 Open Source Computer Vision의 약어
- OpenCV는 2500개가 넘는 최신 컴퓨터 비전 알고리즘과 머신 러닝 알고리즘을 포함
- OpenCV는 기본적으로 C/C++ 언어로 작성되었지만, 현재 널리 사용되고 있는 python, java, matlab, javascript 등 인터페이스 제공
- OpenCV의 공식 웹 사이트 주소는 https://opencv.Org/

1.2 OpenCV 특징

- 이미지 처리와 분석에 최적화 된 필수 패키지
- 기본적인 영상 파일 입출력, 영상의 화질 향상, 객체 검출과 인식, 추적, 3차원 비전 문제 해결 등 기능을 제공
- 실시간 처리를 고려하여 만들어졌기 때문에 다양한 하드웨어 플랫폼에서 매우 빠르게 동작

Image Concept



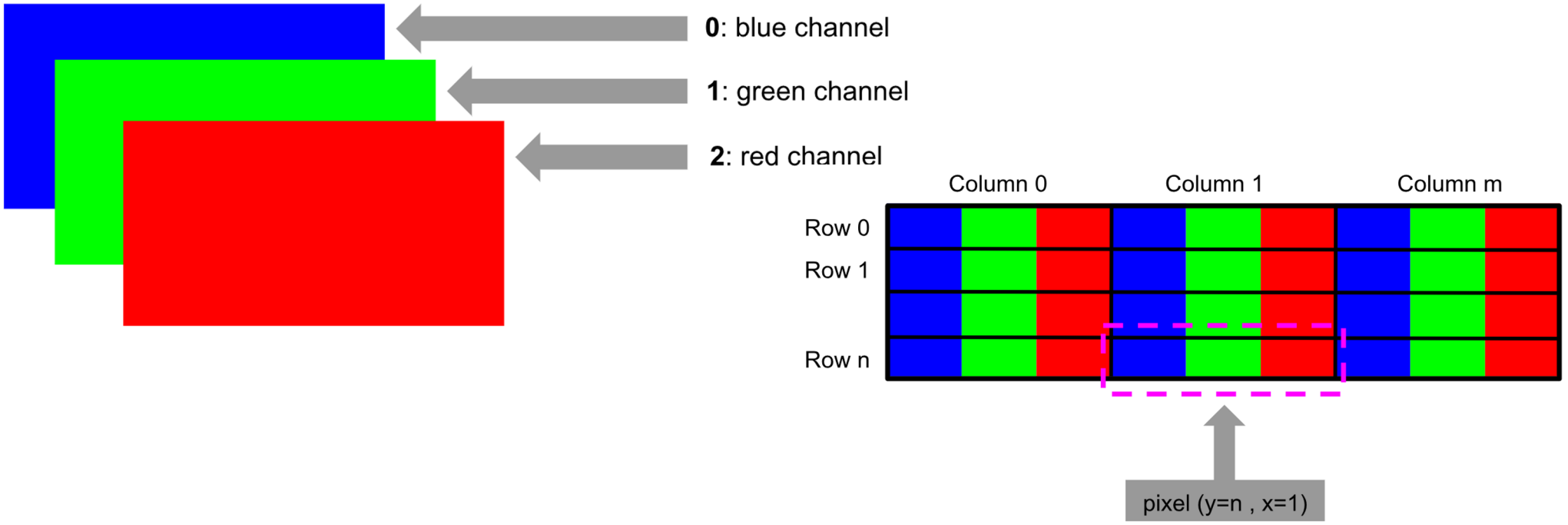


R, G, B
[0, 255] range.

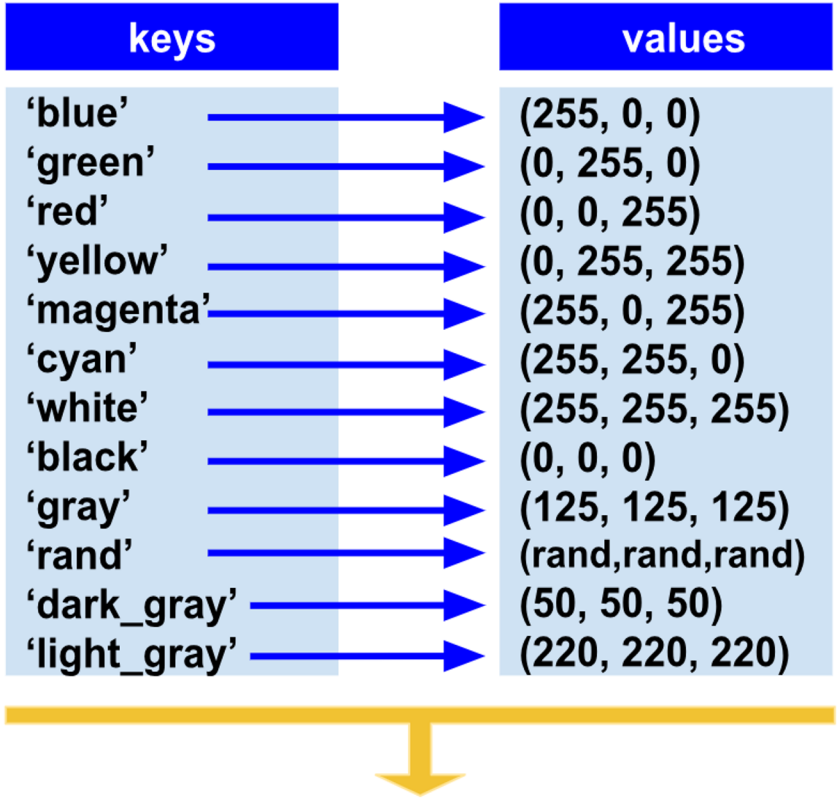
2^8 = 256
24-bit color depth:

OpenCV BGR Colors

- Python은 기본적으로 RGB
- OpenCV는 RGB 대신에 BGR 컬러 포맷 사용



OpenCV BGR Colors



BGR 컬러 크기

BLUE = (255, 0, 0)
GREEN = (0, 255, 0)
RED = (0, 0, 255)
YELLOW = (0, 255, 255)
MAGENTA = (255, 0, 255)
CYAN = (255, 255, 0)
DARK_GRAY = (50, 50, 50)
...

```
get one color:
colors['light_gray']
```

```
colors = {'blue': (255, 0, 0), 'green': (0, 255, 0), 'red': (0, 0, 255), 'yellow': (0, 255,
255), 'magenta': (255, 0, 255), 'cyan': (255, 255, 0), 'white':  (255, 255, 255), 'black':
(0, 0, 0), 'gray': (125, 125, 125), 'rand': np.random.randint(0,high=256,size=(3,)).tolist(),
'dark_gray': (50, 50, 50), 'light_gray': (220, 220, 220)}
```

OpenCV 개요

1.3 OpenCV 패키지 설치

cmd, 터미널에서 설치 시

```
pip install opencv-python
```

Jupyter Notebook에서 설치 시

```
!pip install opencv-python
```

1.4 OpenCV 패키지 import

이미지 처리를 사용하기 위해 OpenCV 패키지를 임포트
현재 OpenCV4가 최신 버전

```
import cv2
print(cv2.__version__)
```

BIGDATA & AI ANALYTICS
EXPERT COMPANY

2. Image 불러오기, 저장하기

이미지 불러오기

imread( ) 함수

```
# 라이브러리를 임포트
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

```
# 흑백 이미지로 로드
image = cv2.imread("images/cybertruck.jpg", cv2.IMREAD_GRAYSCALE)
```

이미지 불러오기


```
# 이미지를 출력
plt.imshow(image, cmap="gray"), plt.axis("off")
plt.show()
```

이미지 로드


```
# 데이터 타입을 확인
type(image)
```

>numpy.ndarray

```
# 이미지 데이터를 확인
image
```

>array([[225, 225, 225, ..., 207, 206, 206],
[226, 226, 226, ..., 209, 208, 207],
[226, 226, 226, ..., 208, 207, 206], ...,
[137, 137, 138, ..., 110, 109, 108],
[135, 135, 135, ..., 107, 106, 105],
[132, 132, 132, ..., 107, 106, 105]], dtype=uint8)

```
# 데이터 차원을 확인
image.shape
```

>(492, 875)

이미지 불러오기

컬러를 표현할 때 기본적으로

- OpenCV는 BGR을 사용
- Python 기본은 RGB 사용

```
# 컬러로 이미지를 로드
image_bgr = cv2.imread("images/ cybertruck.jpg ", cv2.IMREAD_COLOR)
```

```
# 픽셀을 확인
image_bgr[0,0]
```

>array([246, 234, 198], dtype=uint8)

이미지 불러오기


```
# RGB로 변환
image_rgb = cv2.cvtColor(image_bgr, cv2.COLOR_BGR2RGB)
```

```
# 이미지를 출력
plt.imshow(image_rgb), plt.axis("off")
plt.show( )
```

이미지 저장하기

imwrite( ) 함수

파일 확장자는 jpg. png 등으로 저장

```
# 흑백 이미지로 로드
image = cv2.imread("images/cybertruck.jpg", cv2.IMREAD_GRAYSCALE)
```

```
# 흑백 이미지로 로드
cv2.imwrite("images/cybertruck_grayscale.jpg", image)
```

BIGDATA & AI ANALYTICS
EXPERT COMPANY

3. Image 크기 변경, 자르기

이미지 크기 변경

resize( ) 함수

32x32, 64x64, 96x96, 256x256

```
# 흑백 이미지로 로드
image = cv2.imread("images/cybertruck.jpg", cv2.IMREAD_GRAYSCALE)
```

```
# 이미지 크기를 50x50 픽셀로 변경
image_50x50 = cv2.resize(image, (50, 50))
```

이미지 크기 변경


```
#이미지를 출력
plt.imshow(image_50x50, cmap="gray"), plt.axis("off")
plt.show()
```

이미지 자르기

kubwa

2차원 NumPy의 배열로 저장되기 때문에
배열(Arrary)의 슬라이싱으로 자르기

```
# 흑백 이미지로 로드
image = cv2.imread("images/cybertruck.jpg", cv2.IMREAD_GRAYSCALE)
```

```
# 열의 처음 절반과 모든 행을 선택
image_cropped = image[:, :438]
```

이미지 자르기


```
#이미지를 출력
plt.imshow(image_cropped, cmap="gray"), plt.axis("off")
plt.show()
```

BIGDATA & AI ANALYTICS
EXPERT COMPANY

4. Image 명도 조절

이미지 흐리게 하기

blur( ), filter2D( ), GaussianBlur( )

- 이미지를 흐리게 하려면 각 픽셀을 주변 픽셀의 평균값으로 변환
- 커널(Kernel)은 주변 픽셀에 주행하는 연산으로,
  - 커널이 크면 이미지가 부드러움
  - 커널이 작으면 임지가 선명함

```
# 흑백 이미지로 로드
image = cv2.imread("images/plane_256x256.jpg", cv2.IMREAD_GRAYSCALE)
```

```
# 이미지를 흐리게
image_blurry = cv2.blur(image, (5,5))
```

이미지 흐리게

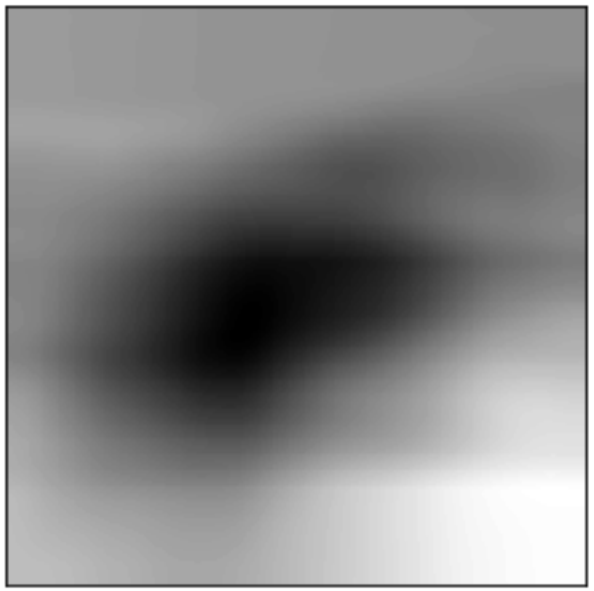
```
#이미지를 출력
plt.imshow(image_blurry, cmap="gray"), plt.axis("off")
plt.show()
```

이미지 흐리게

kubwa

```
# 이미지를 흐리게
image_very_blurry = cv2.blur(image, (100,100))
```

```
#이미지를 출력
plt.imshow(image_very_blurry, cmap="gray"), plt.axis("off")
plt.show( )
```



이미지 흐리게


```
# numpy로 커널을 생성
kernel = np.ones((5,5)) / 25.0
kernel
```

>array([[0.04, 0.04, 0.04, 0.04, 0.04],
[0.04, 0.04, 0.04, 0.04, 0.04],
[0.04, 0.04, 0.04, 0.04, 0.04],
[0.04, 0.04, 0.04, 0.04, 0.04],
[0.04, 0.04, 0.04, 0.04, 0.04]])

이미지 흐리게

```
# 이미지를 흐리게
image_kernel = cv2.filter2D(image, -1, kernel)
```

```
#이미지를 출력
plt.imshow(image_kernel, cmap="gray"), plt.xticks( [] ), plt.yticks( [] )
plt.show()
```

이미지 흐리게


```
# 가우시안 블러
image_very_blurry = cv2.GaussianBlur(image, (5,5), 0)
```

```
#이미지를 출력
plt.imshow(image_very_blurry , cmap="gray"), plt.xticks( [] ), plt.yticks( [] )
plt.show()
```

이미지 흐리게


```
gaus_vector = cv2.getGaussianKernel(5, 0)
gaus_vector
```

```
>array([[0.0625],
 [0.25 ],
 [0.375 ],
 [0.25 ],
 [0.0625]])
```

```
# 벡터를 외적하여 커널 생성
gaus_kernel = np.outer(gaus_vector, gaus_vector)
gaus_kernel
```

```
>array([[0.00390625, 0.015625 , 0.0234375 , 0.015625 , 0.00390625],
[0.015625 , 0.0625 , 0.09375 , 0.0625 , 0.015625 ],
[0.0234375 , 0.09375 , 0.140625 , 0.09375 , 0.0234375 ],
[0.015625 , 0.0625 , 0.09375 , 0.0625 , 0.015625 ],
[0.00390625, 0.015625 , 0.0234375 , 0.015625 , 0.00390625]])
```

이미지 흐리게


```
# 커널을 적용
image_kernel = cv2.filter2D(image, -1, gaus_kernel)
```

```
# 이미지를 출력
plt.imshow(image_kernel, cmap="gray"), plt.xticks([]), plt.yticks([])
plt.show()
```

이미지 선명하게

filter2D( )

대상을 강조하는 커널 생성으로 중앙 픽셀을 부각

```
# 흑백 이미지로 로드
image = cv2.imread("images/plane_256x256.jpg", cv2.IMREAD_GRAYSCALE)
```

```
# 커널을 생성
kernel =   np.array([[0, -1, 0],
                     [-1, 5,-1],
                     [0, -1, 0]])
```

이미지 선명하게


```
# 이미지를 선명하게 변경
image_sharp = cv2.filter2D(image, -1, kernel)
```

```
# 이미지를 출력
plt.imshow(image_sharp, cmap="gray"), plt.axis("off")
plt.show()
```

이미지 대비 향상

equalizeHist( )

히스토그램 평활화(Histogram Equalization)으로 객체의 형태를 두드러지게 만들어 줌

```
# 흑백 이미지로 로드
image = cv2.imread("images/plane_256x256.jpg", cv2.IMREAD_GRAYSCALE)
```

```
# 이미지 대비를 향상
image_enhanced = cv2.equalizeHist(image)
```

이미지 대비 향상

kubwa

```
# 이미지를 출력
plt.imshow(image_enhanced, cmap="gray"), plt.axis("off")
plt.show( )
```

이미지 대비 향상

cvtColor( )

- BRG을 YUV 컬러 포맷으로 변환
- Y는 루마(luma) 밝기, U와 V는 컬러

```
# 이미지 로드
image_bgr = cv2.imread("images/plane.jpg")
```

```
# YUV로 컬러 포맷 변환
image_yuv = cv2.cvtColor(image_bgr, cv2.COLOR_BGR2YUV)
```

```
# 히스토그램 평활화 적용
image_yuv[:, :, 0] = cv2.equalizeHist(image_yuv[:, :, 0])
```

이미지 대비 향상


```
# RGB로 다시 변환
image_rgb = cv2.cvtColor(image_yuv, cv2.COLOR_YUV2RGB)
```

```
# 이미지를 출력
plt.imshow(image_rgb), plt.axis("off")
plt.show( )
```

BIGDATA & AI ANALYTICS
EXPERT COMPANY

5. Image 구분 하기

색깔 구분하기

inRage( ), bitwise_and( )

색 범위 정의 후에 마스크(Mask)를 적용

```
# 이미지를 로드합니다.
image_bgr = cv2.imread('images/plane_256x256.jpg')
```

```
# BGR에서 HSV로 변환합니다.
image_hsv = cv2.cvtColor(image_bgr, cv2.COLOR_BGR2HSV)
```

```
# HSV에서 파랑 값의 범위를 정의합니다.
lower_blue = np.array([50,100,50])
upper_blue = np.array([130,255,255])
```

색깔 구분하기


```
# 마스크 만들기
mask = cv2.inRange(image_hsv, lower_blue, upper_blue)
```

```
# 이미지에 마스크 적용
image_bgr_masked = cv2.bitwise_and(image_bgr, image_bgr, mask=mask)
```

```
# BGR에서 RGB로 변환
image_rgb = cv2.cvtColor(image_bgr_masked, cv2.COLOR_BGR2RGB)
```

색깔 구분하기


```
# 이미지를 출력
plt.imshow(image_rgb), plt.axis("off")
plt.show()
```

```
# 마스크를 출력
plt.imshow(mask, cmap='gray'), plt.axis("off")
plt.show()
```

이미지 이진화

adaptiveThreshold( )

이미지 이진화는 임계처리(Threshold)로
어떤 값보다 큰값의 픽셀을 흰색으로 만들고, 작은 값의 픽셀은 검은색으로 만드는 과정

```
# 흑백 이미지를 로드
image_grey = cv2.imread("images/plane_256x256.jpg", cv2.IMREAD_GRAYSCALE)
```

```
# 적응적 임계처리 파라미터 입력

max_output_value = 255  # 출력 픽셀 강도의 최대값
neighborhood_size = 99 # 픽셀의 임계값 결정에 사용하는 주변 영역의 블록 크기
subtract_from_mean = 10 # 계산된 임계값에서 뺄 상수
```

이미지 이진화


```
# 적응적 임계처리를 적용
image_binarized = cv2.adaptiveThreshold(image_grey,
                                        max_output_value,
                                        cv2.ADAPTIVE_THRESH_GAUSSIAN_C, # 주변 픽셀 강도의 가중치 합
                                        cv2.THRESH_BINARY,
                                        neighborhood_size,
                                        subtract_from_mean)
```

```
plt.imshow(image_binarized, cmap="gray"), plt.axis("off")
plt.show()
```

이미지 이진화


```
# 주변 픽셀의 평균을 임계값으로 설정위해 cv2.ADAPTIVE_THRESH_MEAN_C를 적용
image_mean_threshold = cv2.adaptiveThreshold(image_grey,
                                             max_output_value,
                                             cv2.ADAPTIVE_THRESH_MEAN_C,
                                             cv2.THRESH_BINARY,
                                             neighborhood_size,
                                             subtract_from_mean)
```

```
plt.imshow(image_mean_threshold, cmap="gray"), plt.axis("off")
plt.show()
```

kubwa®

BIGDATA & AI ANALYTICS
EXPERT COMPANY

6. Image 마스킹

배경 제거

grabCut( )

원하는 피사체 주위에 사각형 박스를 그리고 grabCut()을 실행

```
# 이미지를 로드하고 RGB로 변환
image_bgr = cv2.imread('images/plane_256x256.jpg')
image_rgb = cv2.cvtColor(image_bgr, cv2.COLOR_BGR2RGB)
```

```
# 사각형 좌표: 시작점의 x, 시작점의 y, 너비, 높이
rectangle = (0, 56, 256, 150)
```

```
# 초기 마스크를  생성
mask = np.zeros(image_rgb.shape[:2], np.uint8)
```

배경 제거


```
# grabCut에 사용할 임시 배열을 생성
bgdModel = np.zeros((1, 65), np.float64)
fgdModel = np.zeros((1, 65), np.float64)
```

```
# grabCut을 실행
cv2.grabCut(image_rgb, # 원본 이미지
            mask, # 마스크
            rectangle, # 사각형
            bgdModel, # 배경을 위한 임시 배열
            fgdModel, # 전경을 위한 임시 배열
            5, # 반복 횟수
            cv2.GC_INIT_WITH_RECT) # 사각형을 사용한 초기화
```

```
# 배경인 곳은 0, 그외에는 1로 설정한 마스크를 생성
mask_2 = np.where((mask==2) | (mask==0), 0, 1).astype('uint8')
```

배경 제거


```
# 이미지에 새로운 마스크를 곱해 배경 제외
image_rgb_nobg = image_rgb * mask_2[:, :, np.newaxis]
```

```
# 이미지를 출력
plt.imshow(image_rgb_nobg), plt.axis("off")
plt.show()
```

배경 제거

```
# 마스크를 출력
plt.imshow(mask, cmap='gray'), plt.axis("off")
plt.show()
```

배경 제거

```
# 마스크2 를 출력
plt.imshow(mask_2, cmap='gray'), plt.axis("off")
plt.show()
```

경계선 감지

Canny( )

낮은 임계값(lower_threshold)과 높은 임계값(upper_threshold) 을 지정 후,
Canny( ) 함수로 경계선 감지 사용

```
# 흑백 이미지로 로드합니다.
image_gray = cv2.imread("images/plane_256x256.jpg", cv2.IMREAD_GRAYSCALE)
```

```
# 픽셀 강도의 중간값을 계산
median_intensity = np.median(image_gray)
```

```
# 중간 픽셀 강도에서 위아래 1 표준 편차 떨어진 값을 임계값 지정
lower_threshold = int(max(0, (1.0 - 0.33) * median_intensity))
upper_threshold = int(min(255, (1.0 + 0.33) * median_intensity))
```

경계선 감지


```
# 캐니 경계선 감지기를 적용
image_canny = cv2.Canny(image_gray, lower_threshold, upper_threshold)
```

```
# 이미지를 출력
plt.imshow(image_canny, cmap="gray"), plt.axis("off")
plt.show()
```

모서리 감지 I

cornerHarris( ), dialate( )

Harris 모서리 감지를 사용

```
# 흑백 이미지 로드
image_bgr = cv2.imread("images/plane_256x256.jpg")
image_gray = cv2.cvtColor(image_bgr, cv2.COLOR_BGR2GRAY)
image_gray = np.float32(image_gray)
```

```
# 모서리 감지 매개변수를 설정

block_size = 2 # 각 픽셀에서 모서리 감지에 사용하는 이웃의 픽셀 크기
aperture = 29 # 소벨 커널의 크기
free_parameter = 0.04 # 부드러운 모서리 구별
```

모서리 감지


```
# 모서리를 감지
detector_responses = cv2.cornerHarris(image_gray,
                                      block_size,
                                      aperture,
                                      free_parameter)
```

```
# 모서리 표시를 부각
detector_responses = cv2.dilate(detector_responses, None)
```

```
# 임계값보다 큰 감지 결과만 남기고 흰색으로 표시
threshold = 0.02
image_bgr[detector_responses >
          threshold *
          detector_responses.max()] = [255,255,255]
```

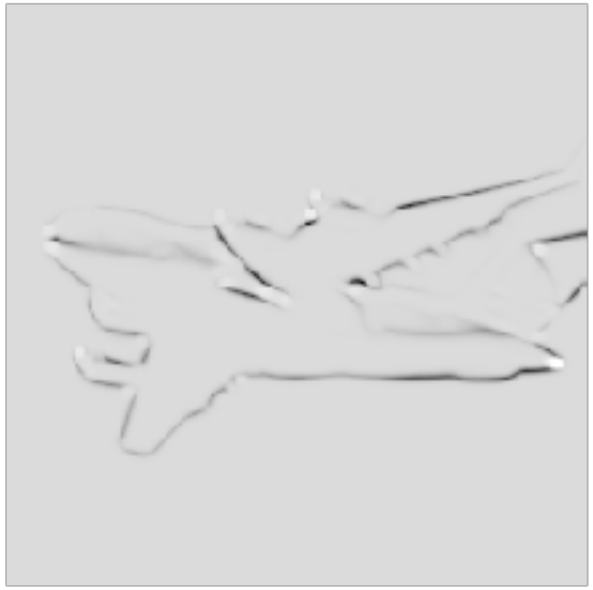
모서리 감지


```
# 흑백으로 변환
image_gray = cv2.cvtColor(image_bgr, cv2.COLOR_BGR2GRAY)
```

```
# 이미지를 출력
plt.imshow(image_gray, cmap="gray"), plt.axis("off")
plt.show()
```

모서리 감지


```
# 가능성 높은 모서리를 출력
plt.imshow(detector_responses, cmap='gray'), plt.axis("off")
plt.show()
```



모서리 감지 II

goodFeaturesToTrack( )

Shi-Tomasi 모서리 감지기(goodFeaturesToTrack)을 사용하여 뚜렷하게
나타난 모서리를 지정

```
# 이미지를 로드
image_bgr = cv2.imread('images/plane_256x256.jpg')
image_gray = cv2.cvtColor(image_bgr, cv2.COLOR_BGR2GRAY)
```

```
# 감지할 모서리 개수 매개변수 Parmeters 설정

corners_to_detect = 10 # 감지할 모서리 개수
minimum_quality_score = 0.05 # 모서리가 될 최소 품질(0~1 사이)
minimum_distance = 25 # 모서리 사이의 유클리드 거리
```

모서리 감지


```
# 모서리를 감지
corners = cv2.goodFeaturesToTrack(image_gray,
                                  corners_to_detect,
                                  minimum_quality_score,
                                  minimum_distance)
```

```
# numpy의 실수로 변환
corners = np.float32(corners)
```

```
# 모서리마다 흰 원을 그려준다
for corner in corners:
    x, y = corner[0]
    cv2.circle(image_bgr, (x,y), 10, (255,255,255), -1)
```

모서리 감지


```
# 흑백 이미지로 변환
image_rgb = cv2.cvtColor(image_bgr, cv2.COLOR_BGR2GRAY)
```

```
# 이미지를 출력
plt.imshow(image_rgb, cmap='gray'), plt.axis("off")
plt.show()
```

BIGDATA & AI ANALYTICS
EXPERT COMPANY

7. Image 인코딩

평균색을 특성으로 인코딩

mean( )

RGB 채널의 평균값을 계산하여 이미지의 평균 컬러를 나타내는 컬러 특성을 만듬

```
# BGR 이미지로 로드
image_bgr = cv2.imread("images/plane_256x256.jpg", cv2.IMREAD_COLOR)
```

```
# 각 채널의 평균을 계산
channels = cv2.mean(image_bgr)
```

```
# 파랑과 빨강을 변환(BGR 에서 RGB로 변환)
observation = np.array([(channels[2], channels[1], channels[0])])
```

평균색을 특성으로 인코딩


```
# 채널 평균 값을 확인
observation
```

>array([[90.53204346, 133.11735535, 169.03074646]])

```
# 이미지를 출력
plt.imshow(observation), plt.axis("off")
plt.show( )
```

컬러 히스토그램을 특성으로 인코딩

kubwa®

calcHist( )

- RBG 각 채널에서 히스토그램을 계산하여 적용
- 각 채널별로 256개의 특성이 있어 이미지에서 컬러 분포는 768개의 특성을 만들 수 있다

```
# BGR 이미지로 로드
image_bgr = cv2.imread("images/plane_256x256.jpg", cv2.IMREAD_COLOR)
```

```
# RGB로 변환
image_rgb = cv2.cvtColor(image_bgr, cv2.COLOR_BGR2RGB)
```

```
# 특성 값을 담을 리스트를 생성
features = [ ]

# 각 컬러 채널에 대해 히스토그램을 계산
colors = ("r", "g", "b")
```

컬러 히스토그램을 특성으로 인코딩

kubwa

```
# 각 채널을 반복하면서 히스토그램을 계산하고 리스트에 추가
for i, channel in enumerate(colors):
    histogram = cv2.calcHist([image_rgb], # 이미지
                             [i], # 채널 인덱스
                             None, # 마스크 없음
                             [256], # 히스토그램 크기
                             [0,256]) # 범위
    features.extend(histogram)
```

```
# 샘플의 특성 값으로 벡터를 만듬
observation = np.array(features).flatten( )

# 처음 다섯 개의 특성을 출력
observation[0:5]
```

>array([1008., 217., 184., 165., 116.], dtype=float32)

컬러 히스토그램을 특성으로 인코딩

kubwa

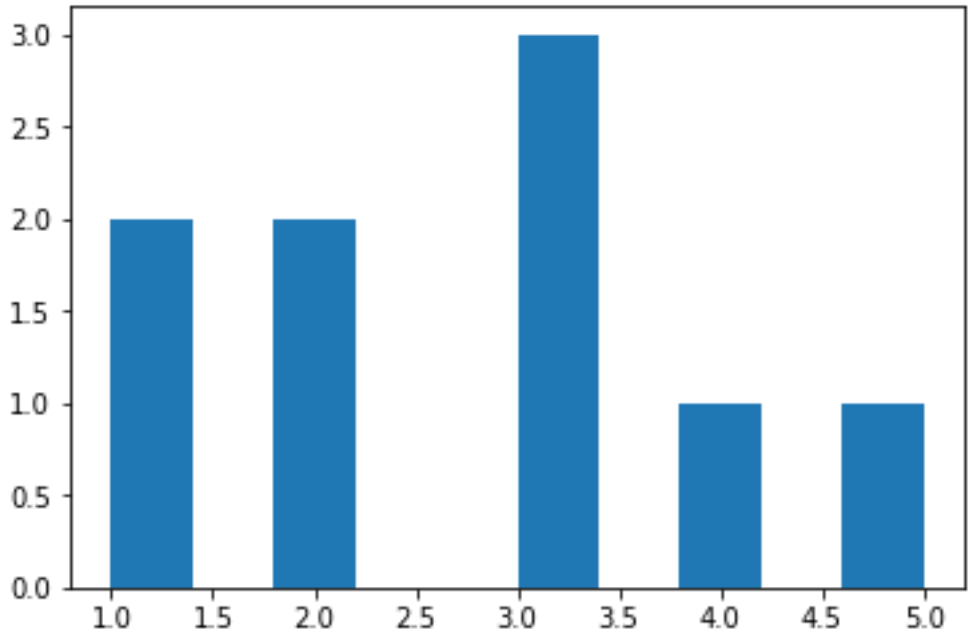
```
# RGB 채널 값을 확인
image_rgb[0,0]
```

>array([107, 163, 212], dtype=uint8)

```
# 판다스 임포트 불러오기
import pandas as pd

# 예시 데이터를  시리즈로 생성
data = pd.Series([1, 1, 2, 2, 3, 3, 3, 4, 5])
```

컬러 히스토그램을 특성으로 인코딩

```
# 히스토그램을 출력
data.hist(grid=False)
plt.show( )
```



컬러 히스토그램을 특성으로 인코딩

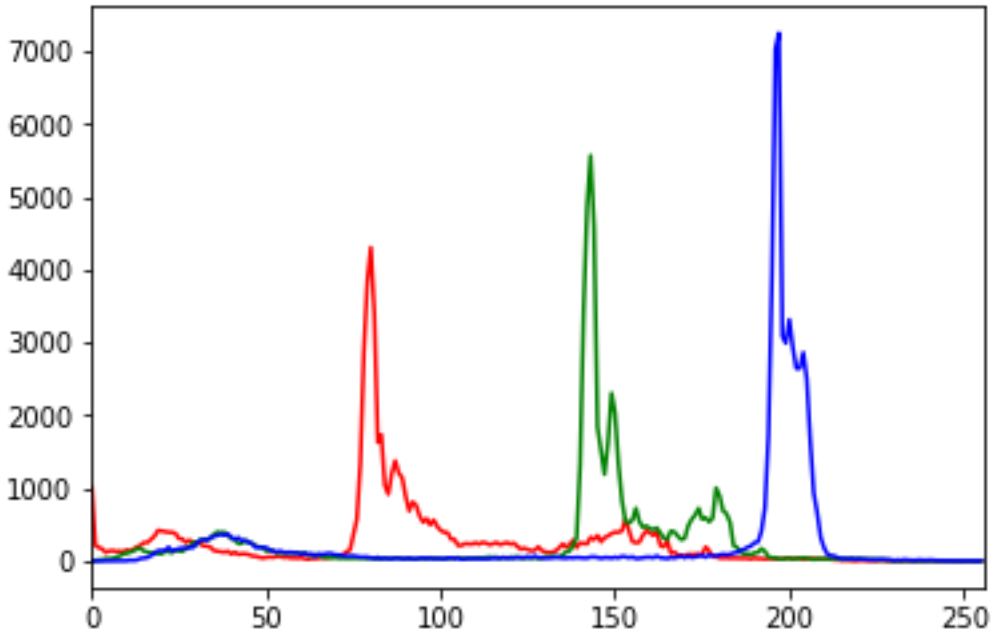
kubwa

```
# 각 컬러 채널에 대한 히스토그램을 계산
colors = ("r","g","b")
```

```
# 컬러 채널을 반복하면서 히스토그램을 계산하고 그래프를 그리기

for i, channel in enumerate(colors):
    histogram = cv2.calcHist([image_rgb], # 이미지
                             [i], # 채널 인덱스
                             None, # 마스크 없음
                             [256], # 히스토그램 크기
                             [0,256]) # 범위
    plt.plot(histogram, color = channel)
    plt.xlim([0,256])
```

컬러 히스토그램을 특성으로 인코딩

```
# 그래프를 출력
plt.show( )
```



BIGDATA & AI ANALYTICS
EXPERT COMPANY

8. 머신러닝 특성 만들기

머신러닝 특성 만들기

NumPy의 flatten( ) 사용하여 다차원 배열을 샘플값이 담긴 벡터로 변환

```
# 흑백 이미지로 로드
image = cv2.imread("images/plane_256x256.jpg", cv2.IMREAD_GRAYSCALE)
```

```
# 이미지를 10x10 픽셀 크기로 변환
image_10x10 = cv2.resize(image, (10, 10))
```

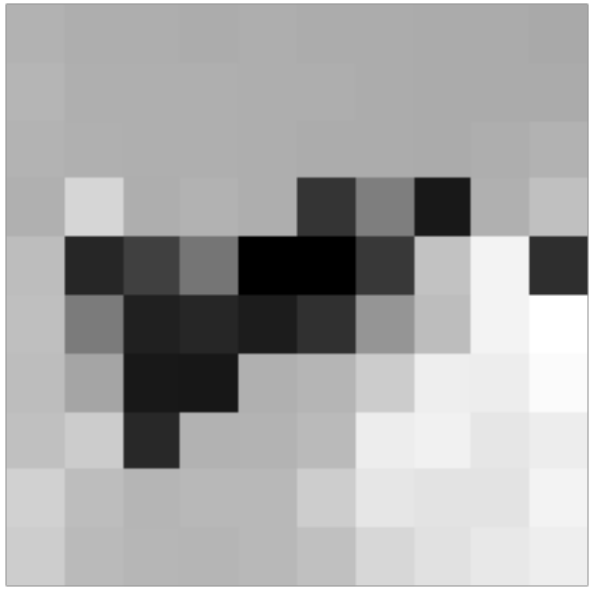
```
# 이미지 데이터를 1차원 벡터로 변환
image_10x10.flatten( )
```

>array([133, 130, 130, 129, 130, 129, 129, 128, 128, 127, 135, 131, 131, 131, 130, 130, 129, 128, 128, 128, 134, 132, 131, 131, 130, 129, 129, 128, 130, 133, 132, 158, 130, 133, 130, 46, 97, 26, 132, 143, 141, 36, 54, 91, 9, 9, 49, 144, 179, 41, 142, 95, 32, 36, 29, 43, 113, 141, 179, 187, 141, 124, 26, 25, 132, 135, 151, 175, 174, 184, 143, 151, 38, 133, 134, 139, 174, 177, 169, 174, 155, 141, 135, 137, 137, 152, 169, 168, 168, 179, 152, 139, 136, 135, 137, 143, 159, 166, 171, 175], dtype=uint8)

머신러닝 특성 만들기

kubwa

```
plt.imshow(image_10x10, cmap="gray"), plt.axis("off")
plt.show()
```



머신러닝 특성 만들기

```
image_10x10.shape
```

>(10, 10)

```
image_10x10.flatten().shape
```

>(100,)

```
# 컬러 이미지로 로드
image_color = cv2.imread("images/plane_256x256.jpg", cv2.IMREAD_COLOR)
```

```
# 이미지를 10 × 10 픽셀 크기로 변환
image_color_10x10 = cv2.resize(image_color, (10, 10))
```

```
# 이미지 데이터를 1차원 벡터로 변환하고 차원을 출력
image_color_10x10.flatten().shape
```

>(300,)

머신러닝 특성 만들기

```
# 흑백 이미지로 로드
image_256x256_gray = cv2.imread("images/plane_256x256.jpg",
cv2.IMREAD_GRAYSCALE)
```

```
# 이미지 데이터를 1차원 벡터로 변환하고 차원을 출력
image_256x256_gray.flatten().shape
```

>(65536,)

```
# 컬러 이미지로 로드
image_256x256_color = cv2.imread("images/plane_256x256.jpg",
cv2.IMREAD_COLOR)
```

```
# 이미지 데이터를 1차원 벡터로 변환하고 차원을 출력
image_256x256_color.flatten().shape
```

>(196608,)

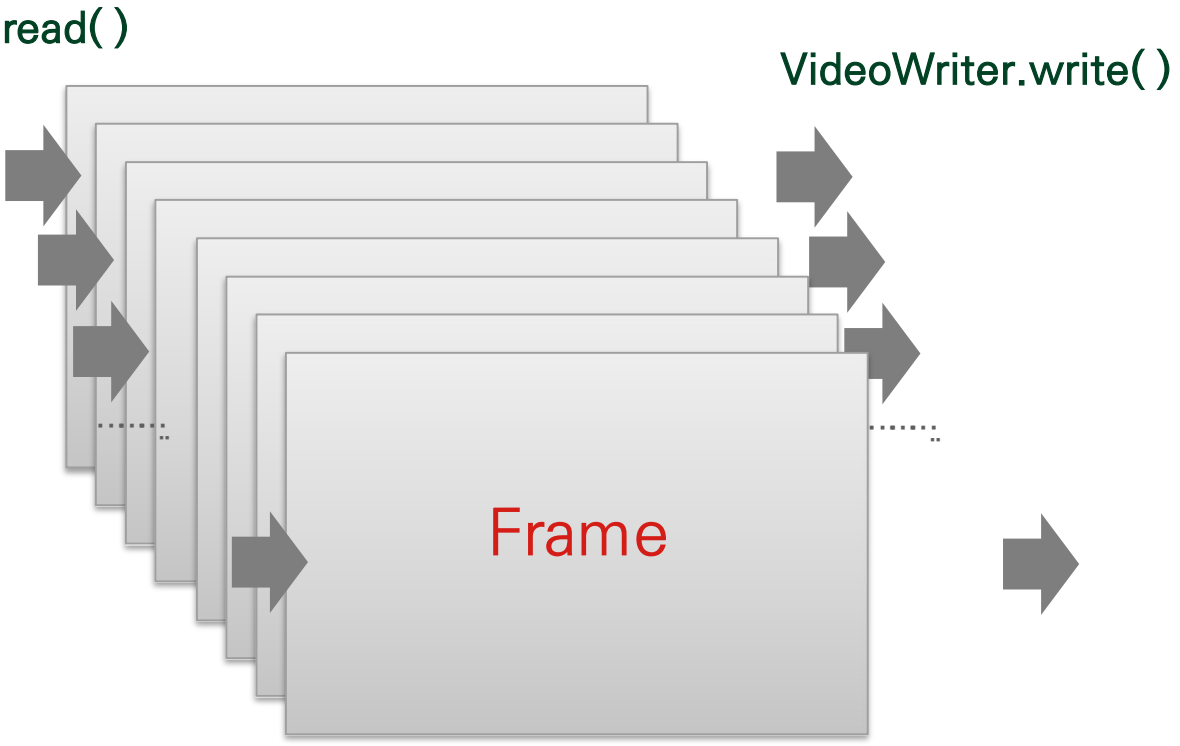
BIGDATA & AI ANALYTICS
EXPERT COMPANY

9. 딥러닝 DNN Inference

OpenCV VideoCapture

- OpenCV의 VideoCapture클래스는 동영상을 개별 Frame으로 하나씩 읽어(Read)들이는 기능을 제공
- VideoWriter는 VideoCapture로 읽어들인 개별 Frame을 동영상 파일로 Write 수행.

```
cap = cv2.VideoCapture(video_input_path)
......
......
vid_writer = cv2.VideoWriter(video_output_path, ….)

while True:
    hasFrame, img_frame = cap.read()
    if not hasFrame:
        print('더 이상 처리할 frame이 없습니다.')
        break

    vid_writer.write(img_frame))
```



OpenCV VideoWriter

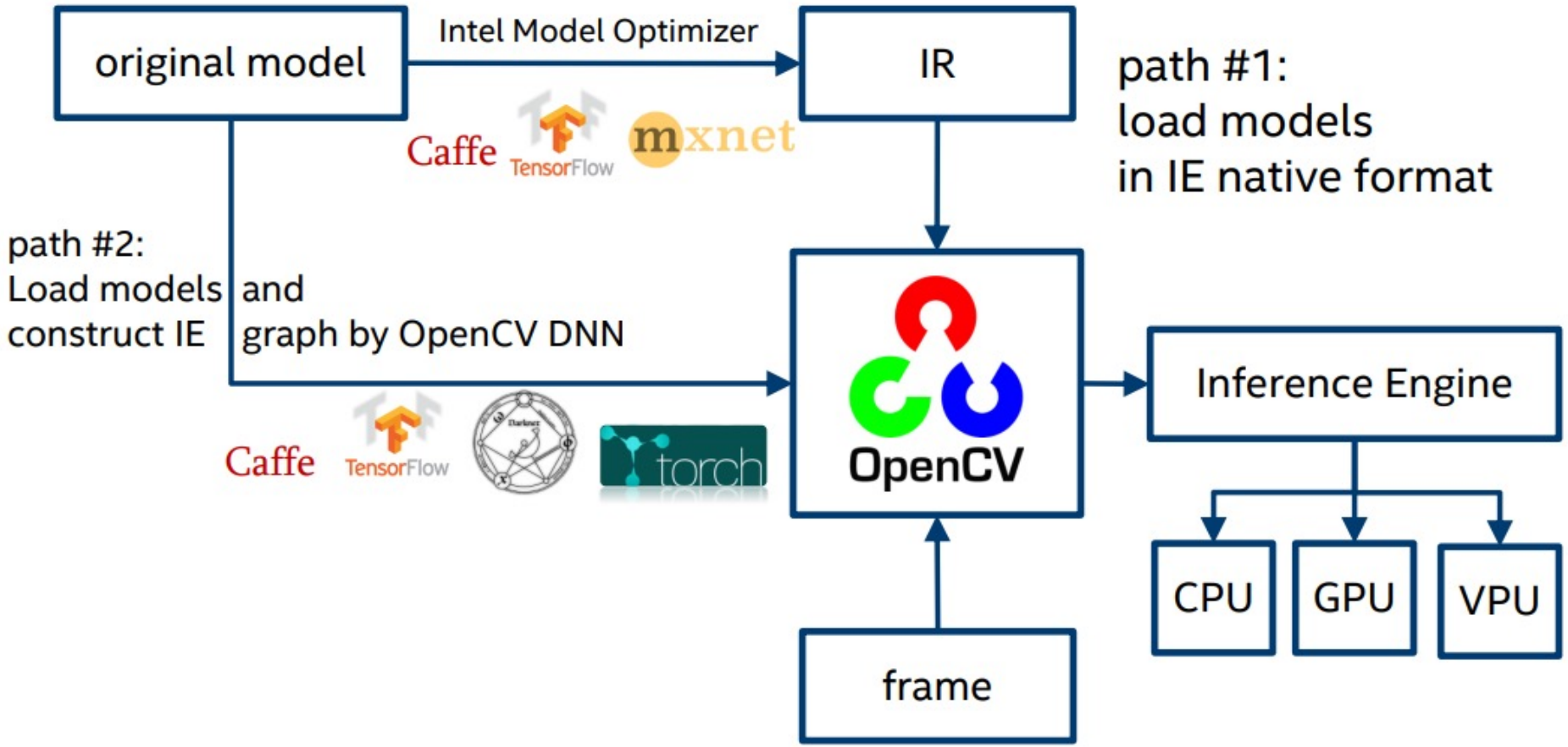
kubwa®

```
cap = cv2.VideoCapture(video_input_path)
codec = cv2.VideoWriter_fourcc(*'XVID')

vid_size = (round(cap.get(cv2.CAP_PROP_FRAME_WIDTH)),
            round(cap.get(cv2.CAP_PROP_FRAME_HEIGHT)))
vid_fps = cap.get(cv2.CAP_PROP_FPS )

vid_writer = cv2.VideoWriter(video_output_path, codec, vid_fps, vid_size)
```

- VideoWriter 객체는 write할 동영상 파일 위치, Encoding코덱 유형, write fps 수치, frame 크기를 생성자로 입력 받아 이들 값에 따른 동영상 Write 수행
- VideoWriter는 write시 특정 포맷으로 동영상을 Encoding 할 수 있음(DIVX, XVID, MJPG, X264, WMV1, WMV2)
- 리눅스 서버에서 반드시 XVID 형태로만 Encoding 적용하고 write시 동영상의 확장자는 반드시 avi 만 적용

OpenCV DNN Inference

- OpenCV C++, OpenCV Python
- DL Library Support: Tensorflow, Pytorch, Caffe2, theano
- Supports almost Topologies



OpenCV Inference in python

1. Architecture and weight files for the model

```
textGraph= "./mask_rcnn_inception_v2_coco_2018_01_28.pbtxt"
modelWeights= "./frozen_inference_graph.pb”

# Load the network
net = cv.dnn.readNetFromTensorflow(modelWeights, textGraph);
```

2. Set Backend & Target

```
net.setPreferableBackend(cv.dnn.DNN_BACKEND_OPENCV)
net.setPreferableTarget(cv.dnn.DNN_TARGET_CPU)
```

Backendrefers to the implementation
1.DNN_BACKEND_OPENCV
2.DNN_BACKEND_HALIDE
3.DNN_BACKEND_INFERENCE_ENGINE

Target refers to the processor
1.DNN_TARGET_CPU
2.DNN_TARGET_OPENCL
3.DNN_TARGET_OPENCL_FP16
4.DNN_TARGET_MYRIAD
5.DNN_TARGET_FPGA
6.DNN_TARGET_CUDA

OpenCV Inference in python

kubwa

3. Input image and videos with blob

```
hasFrame, frame = cap.read( )

# Create a 4D blob from a frame
blob = cv.dnn.blobFromImage(frame, swapRB=True, crop=False)

# Set the input to the network
net.setInput(blob)
```

4. Run the forward pass to get output from the output layers

```
boxes, masks = net.forward(['detection_out_final','detection_masks'])
```

OpenCV Inference 최적화

- **손쉬운 Inference**
  - Tensorflow, Caffe, Pytorch 등 각 Deep Learning Library에서 학습 추론 모델중
  - *.pb, *.pbtxt 파일만으로 아래 Object Detection과 Segmentaiton의 Inference 지원
- **영상(Video) Inference 지원**
  - OpenCV에서 제공하는 VideorWriter( ) API로 영상 Inference 가능
- **OpenVINO, CUDA 지원**
  - OpenVINO와 연계하여 xml, bin 인텔의 하드웨어 기본 지원 및
  - 2019년부터 NIVIDA CUDA의 벡엔드 지원

영상 검출 (Object Detection)

YOLOv3
SSD VGG
MobileNet-SSD
Faster-RCNN
R-FCN
OpenCVface detector
TinyYolov3

영상 분할 (Segmentation)

FCN
ENet
ResNet101_DUC_HDC
Mask-RCNN