

5주차 요약본

작성자



이의진(엘텍공과대학 휴먼기계바이오공학부)

9.4 분할

영상 분할

고전 알고리즘: 영역이 어떤 물체인지 알 수 없음. 의미를 모름 ➡ 딥러닝 등장

딥러닝을 이용한 영상 분할

◆ 종류



(a) 예제 영상

(b) 의미 분할

(c) 사례 분할

(d) 총괄 분할

그림 9-28 세 종류의 영상 분할 문제

thing-셀 수 있는 물체, stuff-셀 수 없는 물체

의미 분할(semantic segmentation):

→ 모든 pixel에 thing과 stuff 할당. 같은 부류의 thing이 여러 개면 같은 부류 번호 할당

사례 분할(instance segmentation):

→ thing만 분할. 같은 부류의 thing이 여러 개면 고유한 번호를 할당해 구분

총괄 분할(panoptic segmentation): semantic + panoptic

→ 모든 pixel에 thing과 stuff 할당. thing은 고유 번호까지 할당

9.4.1 성능 척도와 데이터셋

분류: 한 영상 → 하나의 부류 확률 벡터

분할: 한 영상 → pixel별로 부류를 지정

➡ 밀집 분류(dense classification) : 분할도 분류 문제 → 정확률로 성능 측정 가능

◆ PA(Pixel Accuracy)

화소 정확률: 분할이 사용하는 정확률

$$PA = \frac{\sum_{i=0,C} p_{ii}}{\sum_{i=0,C} \sum_{j=0,C} p_{ij}}$$

1) p_{ij} : 참값이 i 인데, j로 예측한 pixel 수

2) C : 물체 부류 개수

0은 배경을 나타냄. 총 $C+1$ 개 부류

분모는 전체 pixel수, 분자는 맞힌 pixel수

♦MPA(Mean Pixel Accuracy)

평균 화소 정확률: 부류별로 PA 계산하고 결과를 평균한 식

$$MPA = \frac{1}{C+1} \sum_{i=0,C} \frac{p_{ii}}{\sum_{j=0,C} p_{ij}}$$

♦IoU

물체 검출에 사용한 IoU를 분할에 적용한 식

부류별 AP(average precision) 계산하고, 모든 부류에 대해 평균하여 mAP를 구한다.

$$IoU = \frac{|A \cap B|}{|A \cup B|}$$

♦Dice 계수(Dice coefficient)

$$Dice = \frac{2|A \cap B|}{|A| + |B|}$$

[예시 9-2] PA, MPA, IoU, Dice 계수 계산

[그림 9-29]는 $C=2$ 인 간단한 상황을 가지고 PA, MPA, IoU 계산을 설명한다. PA와 MPA는 다음과 같다.

$$PA = \frac{29}{36} = 0.8056, MPA = \frac{1}{3} \left(\frac{14}{15} + \frac{5}{7} + \frac{10}{14} \right) = 0.7873$$

2번 물체에 대해 참값 영역을 A , 예측 영역을 B 라 하면 $A \cup B$ 와 $A \cap B$ 는 [그림 9-29(b)]와 같고 IoU와 Dice 계수는 다음과 같다.

$$IoU = \frac{10}{17} = 0.5882, \text{ Dice} = \frac{2 \times 10}{14 + 13} = 0.7407$$

참값						예측값					
0	0	0	0	0	0	0	0	0	0	0	0
0	2	2	2	0	0	0	2	2	2	2	0
2	2	2	2	1	0	0	2	2	1	2	0
2	2	2	1	1	0	0	2	2	1	1	0
0	2	2	1	1	0	0	2	2	1	1	0
0	2	2	1	1	0	0	0	2	2	1	0

(a) 참값과 예측값

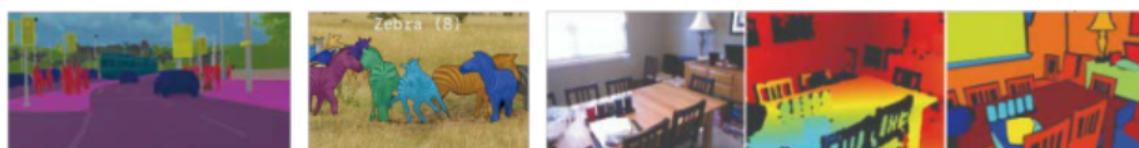
$A \cup B$						$A \cap B$					
	2	2	2	2	2						
2	2	2	2	2	2						
2	2	2									
	2	2									
	2	2									
		2									

(b) 영역 2의 IoU와 Dice 계수 계산 과정

그림 9-29 분할 성능을 측정하는 예시

→ 분할에 쓰는 데이터셋

- PASCAL VOC, ImageNet, COCO, OpenImages → 분할에 필요한 레이블 포함
- 특수한 목적으로 개발된 데이터셋



(a) Cityscapes(취리히) (b) LVIS (c) NYU-Depth V2

그림 9-30 분할에 쓰는 다양한 데이터셋

Cityscapes: 도심 도로 장면 분할

Youtube-Objects: 유튜브 비디오 분할

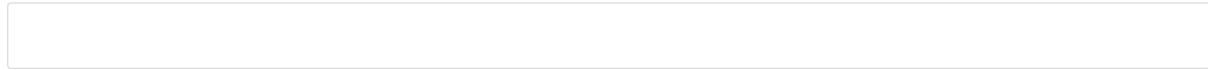
KITTI: 자율주행을 위한

LVIS: COCO를 확장한 데이터셋. 사례분할 위한 레이블링 잘 되어있음.

9.4.2 의미 분할을 위한 FCN

FCN: 의미 분할을 처음 시도한 CNN

초기 CNN: 앞 부분은 Conv-layer, 뒷부분은 FC-layer → 분류 문제



◆FCN(Fully Convolutional Network)의 구조와 동작

완전연결층 제거, 컨볼루션층과 풀링층으로만 구성

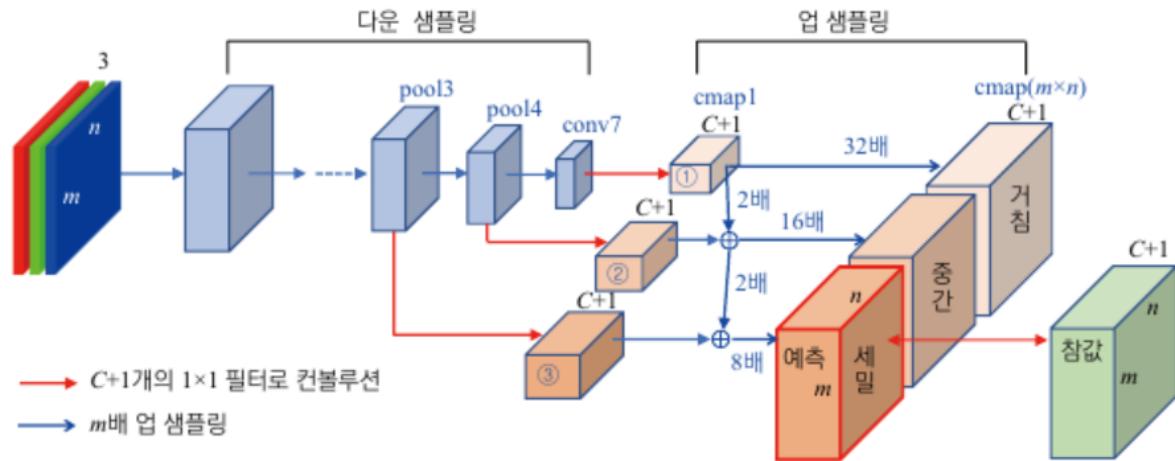


그림 9-31 FCN의 구조

1) 입력: $m \times n$ 컬러 영상

2) 출력: $m \times n \times (C+1)$ 텐서 $\rightarrow m \times n$ 맵이 $C+1$ 장

- 출력 텐서의 각각의 맵($1 \sim C+1$)은 각각의 부류를 책임진다.
- 출력층에서 (j, i) 위치의 $C+1$ 개 픽셀에 softmax 적용
→ 한 pixel은 $C+1$ 개 부류의 확률을 표현한다.

3) 참값: 예측 텐서(출력)와 마찬 가지로 $m \times n \times (C+1)$ 텐서

- (j, i) 위치가 k 번째 부류에 속하면 참값텐서의 (j, i, k)만 1로
- 다른 부류에 해당하는 나머지는 모두 0

4) 학습 알고리즘: 오류(예측 텐서와 참값의 차이)를 줄이는 방향으로 가중치 갱신

5) 학습 마친 모델로 예측: 예측 텐서에서 pixel별로 최댓값을 취함 → 예측맵 출력

[예시 9-3] FCN의 참값 텐서와 예측 텐서

[그림 9-32]는 [그림 9-29(a)]를 [그림 9-31]의 FCN에 맞추어 표현한 것이다. 참값 텐서에서 같은 위치의 모든 화소는 0~2번 맵 중 하나만 1을 가진다. 예측 텐서에서 같은 위치의 모든 화소는 0~2번 맵의 값을 더하여 1.0이 된다. 예측 텐서를 [그림 9-29(a)]의 맵으로 변환해 화면에 디스플레이할 수 있다. 예를 들어 (0, 0) 위치는 (0.8, 0.1, 0.1) 확률을 가지는데 0번 맵(부류 0)이 가장 크므로 0을 해당 위치에 기록한다.

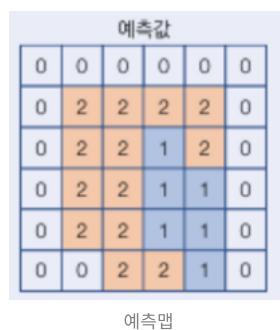
0						1						2					
1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	1	0	0	0	0	0	0	0	1	1	1	1	0
0	0	0	0	0	1	0	0	0	0	1	0	1	1	1	1	0	0
0	0	0	0	0	1	0	0	0	1	1	0	1	1	1	0	0	0
1	0	0	0	0	1	0	0	0	1	1	0	0	1	1	0	0	0
1	0	0	0	0	1	0	0	0	1	1	0	0	1	1	0	0	0

(a) 참값 텐서

0						1						2					
.8	.6	.7	.9	.9	.8	.1	.1	.1	.1	.1	0	.1	.3	.2	0	0	.2
1	.1	.1	0	0	.8	0	0	.1	0	.1	.1	0	.9	.8	1	.9	.1
1	.1	0	0	0	.5	0	0	0	.9	.1	.3	0	.9	1	.1	.9	.2
.8	.1	0	0	.1	.7	0	0	0	1	.9	.2	.2	.9	1	0	0	.1
.9	.1	0	0	0	.9	.1	.1	0	1	1	0	0	.8	1	0	0	.1
1	1	.1	0	.1	.9	0	0	0	0	.9	0	0	0	.9	1	0	.1

(b) 예측 텐서

그림 9-32 FCN의 참값 텐서와 예측 텐서



◆전치 컨볼루션으로 업 샘플링

FCN

- 1) 다운 샘플 : Conv-layer, Pooling-layer의 포복 조절
- 2) 업 샘플링: 전치 컨볼루션(업 컨볼루션)

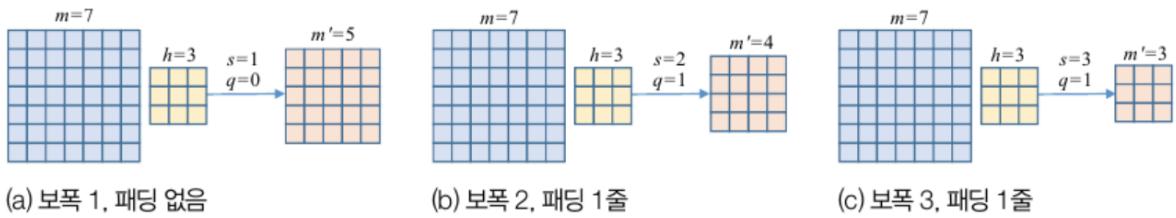


그림 9-33 컨볼루션에 따른 특징 맵의 크기 변화

→ 컨볼루션 적용해 얻은 특징 맵

$$\text{Output Size} = \left(\frac{H + 2P - F}{S} + 1 \right) \times \left(\frac{W + 2P - F}{S} + 1 \right)$$

(b) $m' = (7 + 2 - 3) / 2 + 1 = 4$

→ 전치 컨볼루션

: $m' \times m'$ 특징 맵에 전치 컨볼루션 적용해서 $m \times m$ 특징 맵으로 업 샘플링

<과정>

1. 행 사이에 $s'=s-1$ 개의 0행을 삽입. 열 사이에 $s'=s-1$ 개의 0열을 삽입

새로운 영상은 $(s(m'-1)+1) \times (s(m'-1)+1)$

2. $q'=h-q-1$ 만큼 0을 덧대기

3. stride=1인 컨볼루션 적용

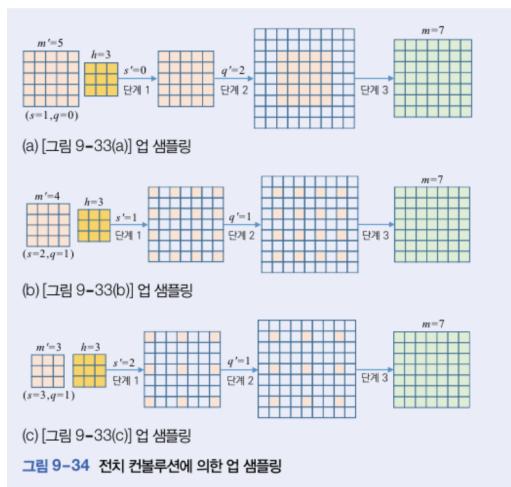


그림 9-34 전치 컨볼루션에 의한 업 샘플링

⇒ 전치 컨볼루션은 맵의 크기만 복원, 값은 복원하지 않음(필터는 학습을 통해 알아내므로)

FCN구조



- : 컨볼루션 층이 출력한 특징 맵에 1×1 커널을 $(C+1)$ 개 적용 → 부류별로 분할 맵을 가진 텐서

- : 텐서를 32배 업 샘플링 → 원본 영상크기의 $m \times n \times (C+1)$ 텐서 만들

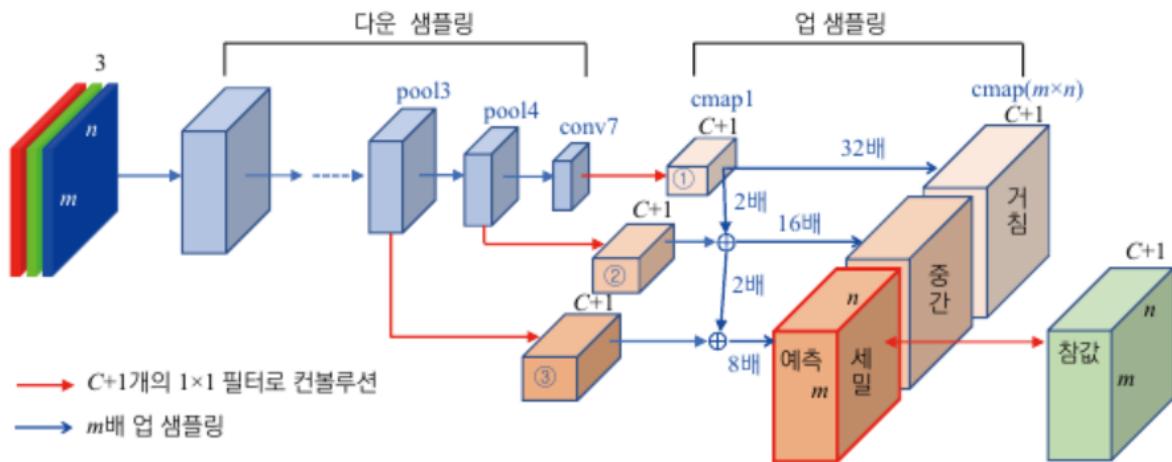


그림 9-31 FCN의 구조

1. Conv-layer로 Feature추출
2. **1x1 커널을 $(C+1)$ 개 적용(채널 수=깊이) → 부류별로 분할 맵을 가지고도록 함.**
3. **Up-sampling(전치 컨볼루션) 한 뒤, 원본 영상크기와 같은 Map 생성**
 - cmap1(①): 전역 정보 가짐 / 세밀함 부족
앞쪽의 특징맵: 전역 정보 부족 / 세밀한 정도
 - 여러 스케일의 특징 맵 결합해 분할 성능을 최고로 유지하는 전략
4. 최종 피처 맵과 라벨 피처 맵의 차이를 이용해 네트워크 학습

💡 Skip architecture

디테일한 segmentation map을 얻기 위함.
→ 피처 추출 단계의 피처맵도 업샘플링에 포함(Sum)하여 정보 손실을 막음

9.4.3 FCN을 개선한 신경망

♦DeConvNet: 오토인코더+FCN

▼ 오토인코더

auto-encoder:

- 입력 영상을 그대로 출력으로 내놓음.
- 인코더는 특징 맵을 점점 작게하고 디코더는 다시 키움
- 인코더와 디코더는 대칭을 이루도록 설계

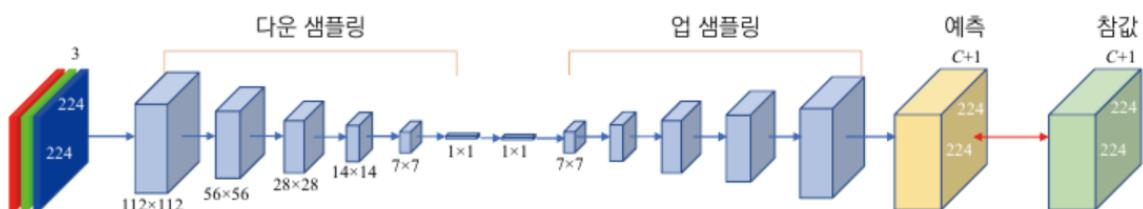


그림 9-35 DeConvNet의 구조

대칭구조의 표준 오토인코더 사용함. 업 샘플링은 전치 컨볼루션으로 달성 → 신경망 구조와 학습이 훨씬 세련되고 성능 우월함

◆U-net

의료 영상 분할을 목적으로 개발됨(2015 MICCAI 학술대회에서 발표됨)

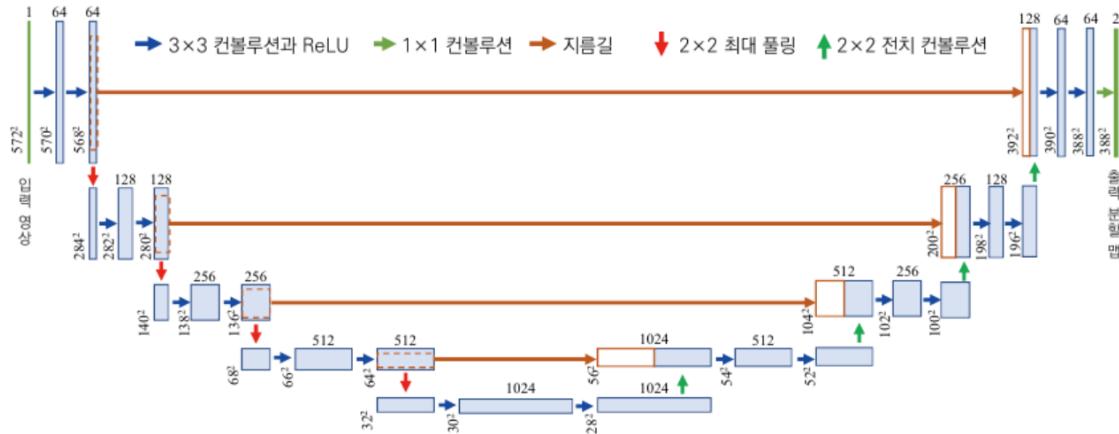


그림 9-36 U-net의 구조

입력 영상: 572 x 572 x 1 명암 영상

출력 영상: 388 x 388 x 2 맵 / 두 채널: 물체(장기), 배경

1. 다운 샘플링(축소 경로): \rightarrow , \downarrow 과정 4번 반복 \rightarrow 32 x 32 x 512 특징 맵
spatial resolution 줄고, 깊이는 늘어남
2. 업 샘플링(확대 경로): \uparrow 전치 컨볼루션을 통해 영상 크기 회복
3. Skip-connection \rightarrow :
 - a. 왼쪽의 568 x 568 x 64 특징 맵의 중앙에서 392 x 392 x 64만큼 잘라내 오른쪽의 확대 경로로 전달
 - b. 지름길 연결로 받은 애(392 x 392 x 64)랑 밑에서 올라온 애(392 x 392 x 64)랑 이어 붙어 392 x 392 x 128 텐서 만듦
 - c. 이 텐서를 컨볼루션층 통과시켜 388 x 388 x 2 분할 맵 출력(각각 물체 부류 확률, 배경 부류 확률)

인코더-디코더 구조 기반 신경망은 상당히 작게 축소했다가 복원하므로 상세 내용을 잃어버릴 가능성 크다

→ 팽창 컨볼루션을 이용한 DeepLabv3로 해결

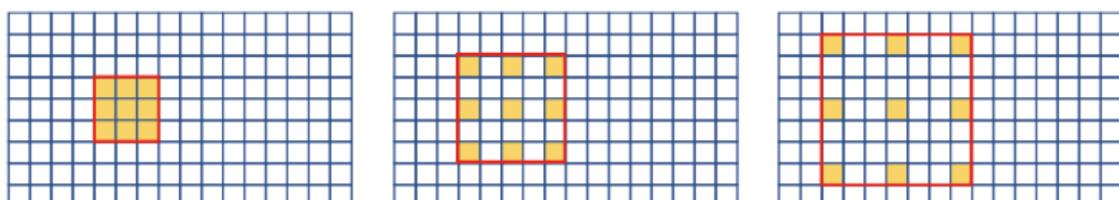


그림 9-37 팽창 컨볼루션(3x3 필터)

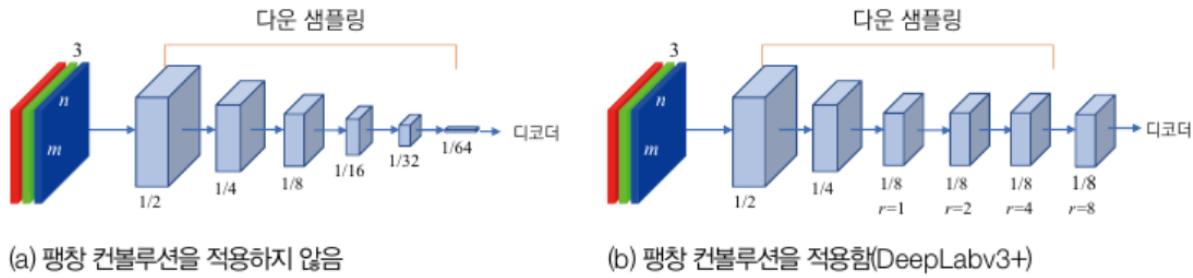
팽창 컨볼루션

receptive field(수용장): 출력 레이어의 픽셀 하나에 영향을 미치는 입력 픽셀들의 공간 크기

팽창계수 $r=1 \rightarrow$ 보통 컨볼루션

$r=2 \rightarrow$ 2-dilated 3x3 convolution / receptive field 는 5x5 로 15

$r=3 \rightarrow$ 3-dilated 3x3 convolution / receptive field 는 7x7 로 49



(a) 팽창 컨볼루션을 적용하지 않음

(b) 팽창 컨볼루션을 적용함(DeepLabv3+)

그림 9-38 팽창 컨볼루션 적용 여부에 따른 신경망 구조

(a) conv, pooling 거치며 특징 맵의 해상도 낮아짐

(b) 세 번째 특징 맵부터 수용장 늘리면서 해상도 유지

9.4.4 프로그래밍 실습: U-net을 이용한 분할 학습

- Dataset

Oxford IIIT Pet dataset

개와 고양이의 품종 37개(품종마다 200여 장씩 총 7349장)

샘플마다 머리 검출 위한 박스 레이블과 분할을 위한 레이블 제공

분할 레이블: 배경(blue), 물체(yellow), 경계(red)로 구분됨.

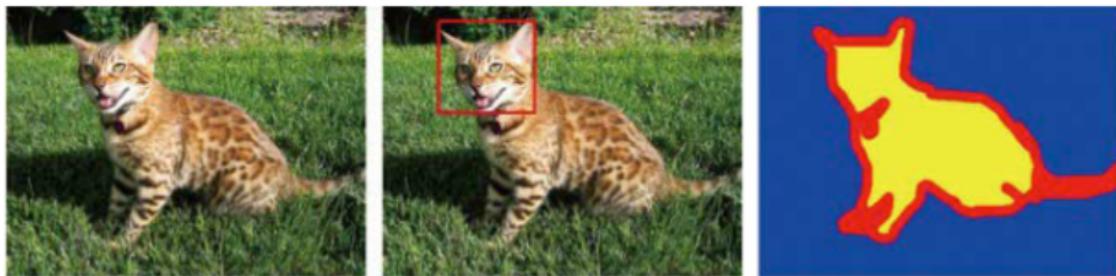


그림 9-39 Oxford IIIT Pet 데이터셋의 예제 영상과 레이블

◆[프로그램 9-4] Oxford pets dataset으로 U-net학습하기

▼ 전체 코드

```
from tensorflow import keras
import numpy as np
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras import layers
import os
import random
import cv2 as cv

input_dir='./datasets/oxford_pets/images/images/' #영상 폴더 경로 지정
target_dir='./datasets/oxford_pets/annotations/annotations/trimaps/' #레이블 폴더 경로 지정
img_size=(160,160) # 모델에 입력되는 영상 크기(원래는 다양한데 정규화해 신경망에 입력하겠다)
n_class=3 # 분할 레이블 (1:물체, 2:배경, 3:경계)
batch_size=32 # 미니 배치 크기

img_paths=sorted([os.path.join(input_dir,f) for f in os.listdir(input_dir) if f.endswith('.jpg')]) #영상 폴더에서 영상 이름을 모음
label_paths=sorted([os.path.join(target_dir,f) for f in os.listdir(target_dir) if f.endswith('.png') and not f.startswith('.')]) #
```

class OxfordPets(keras.utils.Sequence): #학습과 예측에 사용할 데이터를 준비하는 클래스
 def __init__(self, batch_size,img_size,img_paths,label_paths):

```

        self.batch_size=batch_size
        self.img_size=img_size
        self.img_paths=img_paths
        self.label_paths=label_paths

    def __len__(self):
        return len(self.label_paths)//self.batch_size

    def __getitem__(self,idx):
        i=idx*self.batch_size
        batch_img_paths=self.img_paths[i:i+self.batch_size]
        batch_label_paths=self.label_paths[i:i+self.batch_size]
        x=np.zeros((self.batch_size,) + self.img_size+(3,),dtype="float32")
        for j, path in enumerate(batch_img_paths):
            img=load_img(path,target_size=self.img_size)
            x[j]=img
        y=np.zeros((self.batch_size,) + self.img_size+(1,),dtype="uint8")
        for j, path in enumerate(batch_label_paths):
            img=load_img(path,target_size=self.img_size,color_mode="grayscale")
            y[j]=np.expand_dims(img,2)
            y[j]-=1 # 부류 번호를 1,2,3에서 0,1,2로 변환
        return x,y

    def make_model(img_size,num_classes):
        inputs=keras.Input(shape=img_size+(3,))

        # U-net의 다운 셀플링(축소 경로)
        x=layers.Conv2D(32,3,strides=2,padding='same')(inputs)
        x=layers.BatchNormalization()(x)
        x=layers.Activation('relu')(x)
        previous_block_activation=x # 지름길 연결을 위해

        for filters in [64,128,256]:
            x=layers.Activation('relu')(x)
            x=layers.SeparableConv2D(filters,3,padding='same')(x)
            x=layers.BatchNormalization()(x)
            x=layers.Activation('relu')(x)
            x=layers.SeparableConv2D(filters,3,padding='same')(x)
            x=layers.BatchNormalization()(x)
            x=layers.MaxPooling2D(3,strides=2,padding='same')(x)
            residual=layers.Conv2D(filters,1,strides=2,padding='same')(previous_block_activation)
            x=layers.add([x,residual]) # 지름길 연결
            previous_block_activation=x # 지름길 연결을 위해

        # U-net의 업 셀플링(확대 경로)
        for filters in [256, 128, 64, 32]:
            x=layers.Activation('relu')(x)
            x=layers.Conv2DTranspose(filters,3,padding='same')(x)
            x=layers.BatchNormalization()(x)
            x=layers.Activation('relu')(x)
            x=layers.Conv2DTranspose(filters,3,padding='same')(x)
            x=layers.BatchNormalization()(x)
            x=layers.UpSampling2D(2)(x)
            residual=layers.UpSampling2D(2)(previous_block_activation)
            residual=layers.Conv2D(filters,1,padding='same')(residual)
            x=layers.add([x,residual]) # 지름길 연결
            previous_block_activation=x # 지름길 연결을 위해

        outputs=layers.Conv2D(num_classes,3,activation='softmax',padding='same')(x)
        model=keras.Model(inputs, outputs) # 모델 생성
        return model

    model=make_model(img_size,n_class) # 모델 생성

    random.Random(1).shuffle(img_paths)
    random.Random(1).shuffle(label_paths)
    test_samples=int(len(img_paths)*0.1) # 10%를 테스트 집합으로 사용
    train_img_paths=img_paths[:-test_samples]
    train_label_paths=label_paths[:-test_samples]
    test_img_paths=img_paths[-test_samples:]
    test_label_paths=label_paths[-test_samples:]

    train_gen=OxfordPets(batch_siz,img_size,train_img_paths,train_label_paths) # 훈련 집합
    test_gen=OxfordPets(batch_siz,img_size,test_img_paths,test_label_paths) # 검증 집합

    model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
    cb=[keras.callbacks.ModelCheckpoint('oxford_seg.h5',save_best_only=True)] # 학습 결과 자동 저장
    model.fit(train_gen,epochs=30,validation_data=test_gen,callbacks=cb)

    preds=model.predict(test_gen) # 예측

    cv.imshow('Sample image',cv.imread(test_img_paths[0]))# 0번 영상 디스플레이
    cv.imshow('Segmentation label',cv.imread(test_label_paths[0])*64)
    cv.imshow('Segmentation prediction',preds[0]) # 0번 영상 예측 결과 디스플레이

    cv.waitKey()
    cv.destroyAllWindows()

```

1. 데이터 가져오기

```
input_dir='./datasets/oxford_pets/images/images/' #영상 폴더 경로 지정
target_dir='./datasets/oxford_pets/annotations/annotations/trimaps/' #레이블 폴더 경로 지정
img_size=(160,160) # 모델에 입력되는 영상 크기(원래는 다양한데 정규화해 신경망에 입력하겠다)
n_class=3 # 분할 레이블 (1:물체, 2:배경, 3:경계)
batch_size=32 # 미니 배치 크기

img_paths=sorted([os.path.join(input_dir,f) for f in os.listdir(input_dir) if f.endswith('.jpg')]) #영상 폴더에서 영상 이름을 모음
label_paths=sorted([os.path.join(target_dir,f) for f in os.listdir(target_dir) if f.endswith('.png') and not f.startswith('.')]) #
```

2. 커스텀 데이터셋 만들기

```
...
class CustomDataset(torch.utils.data.Dataset):
    def __init__(self):
        데이터셋의 전처리를 해주는 부분
        데이터셋 준비에 필요한 여러 가지 멤버 변수값 설정

    def __len__(self):
        데이터셋의 길이. 즉, 총 샘플의 수를 적어주는 부분

    def __getitem__(self, idx):
        데이터셋에서 특정 1개의 샘플을 가져오는 함수(인덱싱 위해)
    ...

class OxfordPets(keras.utils.Sequence): #학습과 예측에 사용할 데이터를 준비하는 클래스
    # 데이터셋 준비에 필요한 여러 가지 멤버 변수값 설정
    def __init__(self, batch_size,img_size,img_paths,label_paths):
        self.batch_size=batch_size
        self.img_size=img_size
        self.img_paths=img_paths
        self.label_paths=label_paths
        # 미니 배치의 개수 계산
    def __len__(self):
        return len(self.label_paths)//self.batch_size
    # 학습을 수행하는 fit함수가 원활 때마다 미니 배치를 만들어 제공
    def __getitem__(self,idx):
        i=idx*self.batch_size
        batch_img_paths=self.img_paths[i:i+self.batch_size]
        batch_label_paths=self.label_paths[i:i+self.batch_size]
        x=np.zeros((self.batch_size,),)+self.img_size+(3,),dtype="float32")
        for j, path in enumerate(batch_img_paths):
            img=load_img(path,target_size=self.img_size)
            x[j]=img #미니 배치만큼 입력 영상을 읽어와 x에 저장
        y=np.zeros((self.batch_size,),)+self.img_size+(1,),dtype="uint8")
        for j, path in enumerate(batch_label_paths):
            img=load_img(path,target_size=self.img_size,color_mode="grayscale")
            y[j]=np.expand_dims(img,2) #미니 배치만큼 레이블 영상을 읽어와 y에 저장
            y[j]-=1 # 부류 번호를 1,2,3에서 0,1,2로 변환
        return x,y
```

3. U-net에 기반한 모델 생성

```
def make_model(img_size,num_classes):
    inputs=keras.Input(shape=img_size+(3,)) #정규 크기 160x160x3영상을 입력하는 층 추가

    # U-net의 다운 샘플링(축소 경로)
    x=layers.Conv2D(32,3,strides=2,padding='same')(inputs)
    x=layers.BatchNormalization()(x)
    x=layers.Activation('relu')(x) # 첫 컨볼루션 층
    previous_block_activation=x # 지름길 연결을 위해

    for filters in [64,128,256]: # 특징 맵 크기 줄이며 깊이를 늘리기 위해 반복
        x=layers.Activation('relu')(x)
        x=layers.SeparableConv2D(filters,3,padding='same')(x)
        x=layers.BatchNormalization()(x)
        x=layers.Activation('relu')(x)
        x=layers.SeparableConv2D(filters,3,padding='same')(x)
        x=layers.BatchNormalization()(x)
        x=layers.MaxPooling2D(3,strides=2,padding='same')(x)
        # 컨볼루션 2번, 최대 풀링 적용해 x에 저장

    #이전 특징 맵에 컨볼루션 적용해 x와 크기 맞춤.
    residual=layers.Conv2D(filters,1,strides=2,padding='same')(previous_block_activation)
    x=layers.add([x,residual]) # 두 특징맵 더하여 지름길 연결
    previous_block_activation=x # 지름길 연결을 위해

    # U-net의 업 샘플링(확대 경로)
```

```

for filters in [256, 128, 64, 32]: # 특징 맵 크기 키우며 깊이를 줄이기 위해 반복
    x=layers.Activation('relu')(x)
    x=layers.Conv2DTranspose(filters,3,padding='same')(x)
    x=layers.BatchNormalization()(x)
    x=layers.Activation('relu')(x)
    x=layers.Conv2DTranspose(filters,3,padding='same')(x)
    x=layers.BatchNormalization()(x)
    x=layers.UpSampling2D(2)(x)
    # 전치 컨볼루션 2번과 업 샘플링 적용

    # 이전 특징 맵에 업 샘플링과 컨볼루션 적용해 x와 크기 맞춤
    residual=layers.UpSampling2D(2)(previous_block_activation)
    residual=layers.Conv2D(filters,1,padding='same')(residual)
    x=layers.add([x,residual]) # 두 특징맵 더하여 짜름길 연결
    previous_block_activation=x # 짜름길 연결을 위해

#마지막 컨볼루션층 쌓아 신경망 완성. 활성함수는 softmax, 출력 특징 맵의 채널 수는 3
outputs=layers.Conv2D(num_classes,3,activation='softmax',padding='same')(x)
model=keras.Model(inputs, outputs) # 모델 생성
return model

```

4. 모델 생성과 학습 실행, 예측

```

# 모델 생성
model=make_model(img_siz,n_class)

# 데이터셋을 9:1 비율로 훈련 집합과 테스트 집합으로 분할.
random.Random(1).shuffle(img_paths)
random.Random(1).shuffle(label_paths) #분할 전 영상 순서 섞기
test_samples=int(len(img_paths)*0.1) # 10%를 테스트 집합으로 사용
train_img_paths=img_paths[:-test_samples]
train_label_paths=label_paths[:-test_samples]
test_img_paths=img_paths[-test_samples:]
test_label_paths=label_paths[-test_samples:]

#OxfordPets 클래스로 훈련 집합과 테스트 집합을 구성
train_gen=OxfordPets(batch_siz,img_siz,train_img_paths,train_label_paths) # 훈련 집합
test_gen=OxfordPets(batch_siz,img_siz,test_img_paths,test_label_paths) # 검증 집합

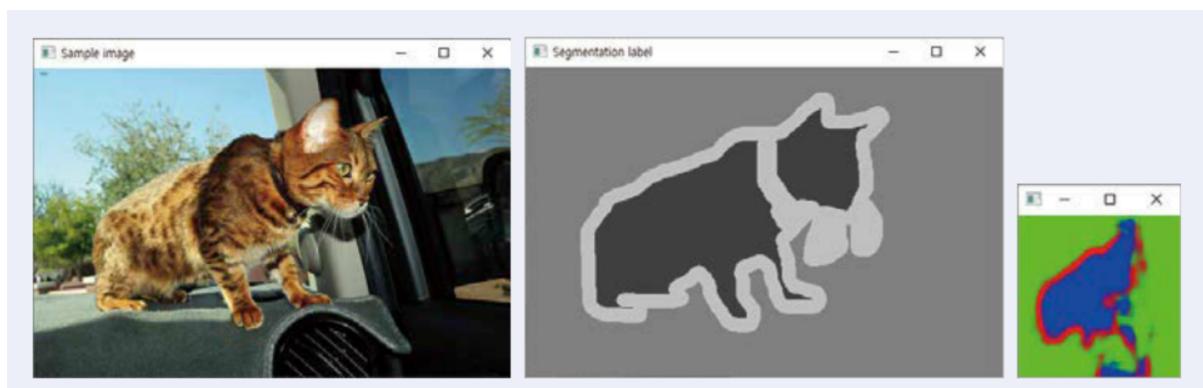
#compile과 fit으로 학습 실행
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',metrics=['accuracy'])
cb=[keras.callbacks.ModelCheckpoint('oxford_seg.h5',save_best_only=True)]
# callbacks 매개변수: 학습 도중 발생하는 가중치 값을 파일에 자동 저장
# save_best_only=True : 학습 도중 발생하는 가장 높은 성능의 가중치를 저장
model.fit(train_gen,epochs=30,validation_data=test_gen,callbacks=cb)

preds=model.predict(test_gen) # 테스트 집합으로 예측

cv.imshow('Sample image',cv.imread(test_img_paths[0]))# 0번 영상 디스플레이
cv.imshow('Segmentation label',cv.imread(test_label_paths[0])*64)
cv.imshow('Segmentation prediction',preds[0]) # 0번 영상 예측 결과 디스플레이

cv.waitKey()
cv.destroyAllWindows()

```

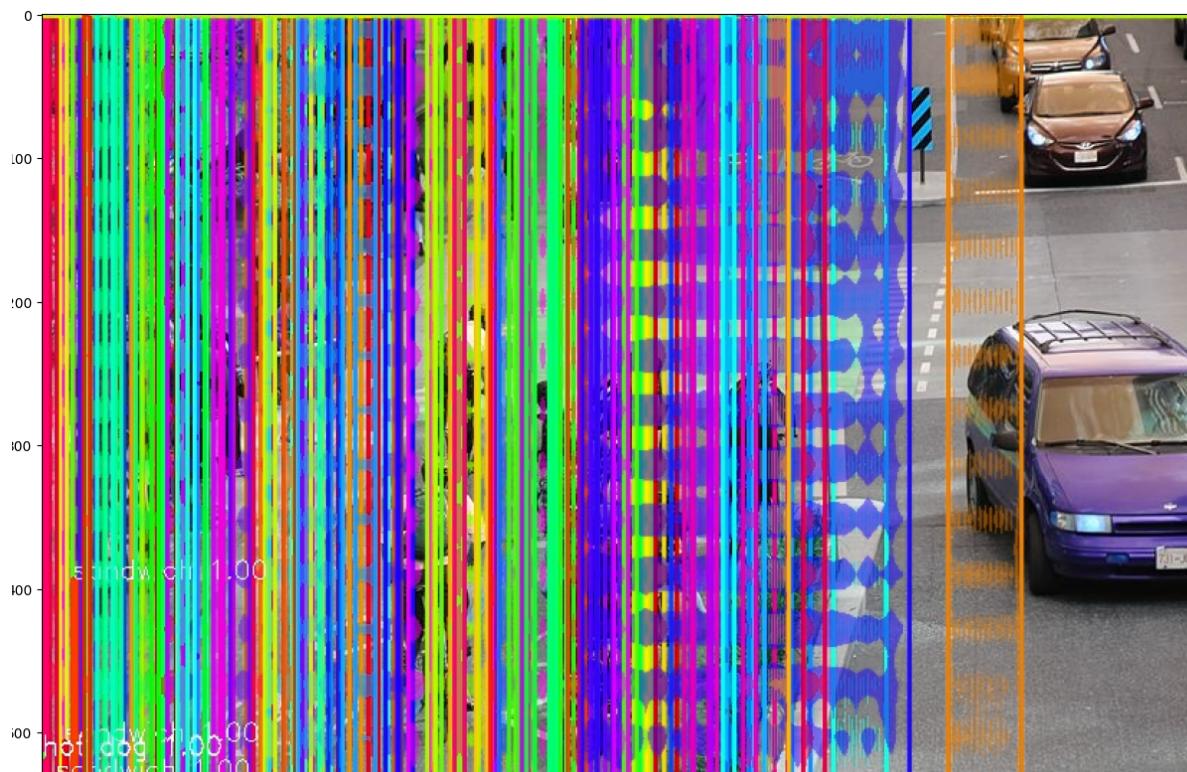


원: 입력 영상

가운데: 참값

오른쪽: 예측값

9.4.4 pixellib 라이브러리를 이용한 프로그래밍 실습



◆[프로그램 9-5] pixellib 라이브러리로 정지 영상을 의미 분할하기

```
from pixellib.semantic import semantic_segmentation  
#의미 분할에 쓸 클래스 불러옴  
import cv2 as cv  
  
seg=semantic_segmentation()
```

```

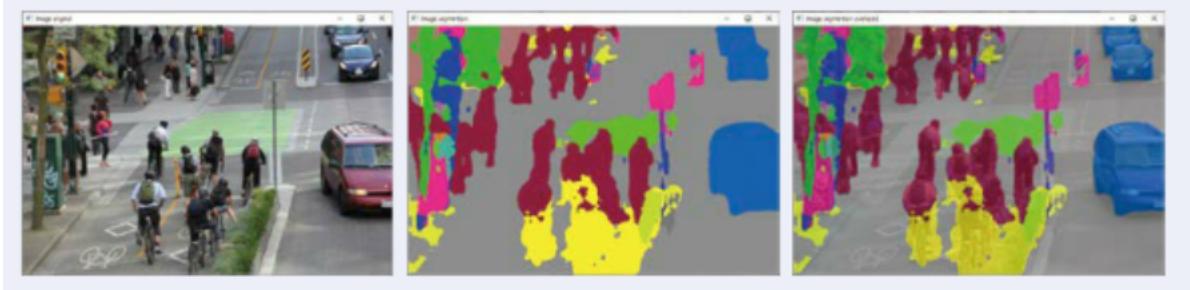
seg.load_ade20k_model('deeplabv3_xception65_ade20k.h5')
# Ade20K 데이터셋으로 학습한 모델 읽기

img_fname='busy_street.jpg'
# 의미 분할을 수행하는 세 가지 방법
seg.segmentAsAde20k(img_fname,output_image_name='image_new.jpg')
info1,img_segmented1=seg.segmentAsAde20k(img_fname) #분할된 영상은 img_segmented1 메타 정보는 info에 저장
info2,img_segmented2=seg.segmentAsAde20k(img_fname,overlay=True) #원래 영상에 분할 결과를 투명하게 겹쳐 표시

cv.imshow('Image original',cv.imread(img_fname))
cv.imshow('Image segmentation',img_segmented1)
cv.imshow('Image segmentation overlayed',img_segmented2)

cv.waitKey()
cv.destroyAllWindows()

```



◆[프로그램 9-6] pixellib 라이브러리로 비디오를 의미 분할하기

```

from pixellib.semantic import semantic_segmentation
#의미 분할에 쓸 클래스 불러옴
import cv2 as cv

cap=cv.VideoCapture(0) #웹 캠 연결. 결과는 cap에 저장

seg_video=semantic_segmentation()
seg_video.load_ade20k_model('deeplabv3_xception65_ade20k.h5')
# Ade20K 데이터셋으로 학습한 모델 읽기

seg_video.process_camera_ade20k(cap,overlay=True,frames_per_second=2,output_video_name='output_video.mp4',show_frames=True,frame_name=
# 웹 캠을 통해 들어오는 비디오 분할
# 두 번째 인수: 원래 영상에 색상 겹쳐 표시
# 세 번째 인수: 초당 2프레임을 동영상에 저장
# 네 번째 인수: 폴더에 저장할 mp4파일 이름 지정
# 다섯 번째 인수: 원도우 열어 분할 결과를 실시간으로 디스플레이 하라
# 여섯 번째 인수: 원도우 이름
cap.release()
cv.destroyAllWindows()

```

◆[프로그램 9-7] pixellib 라이브러리로 정지 영상을 의미 분할하기

```

from pixellib.instance import instance_segmentation
#사례 분할에 쓸 클래스 불러옴
import cv2 as cv

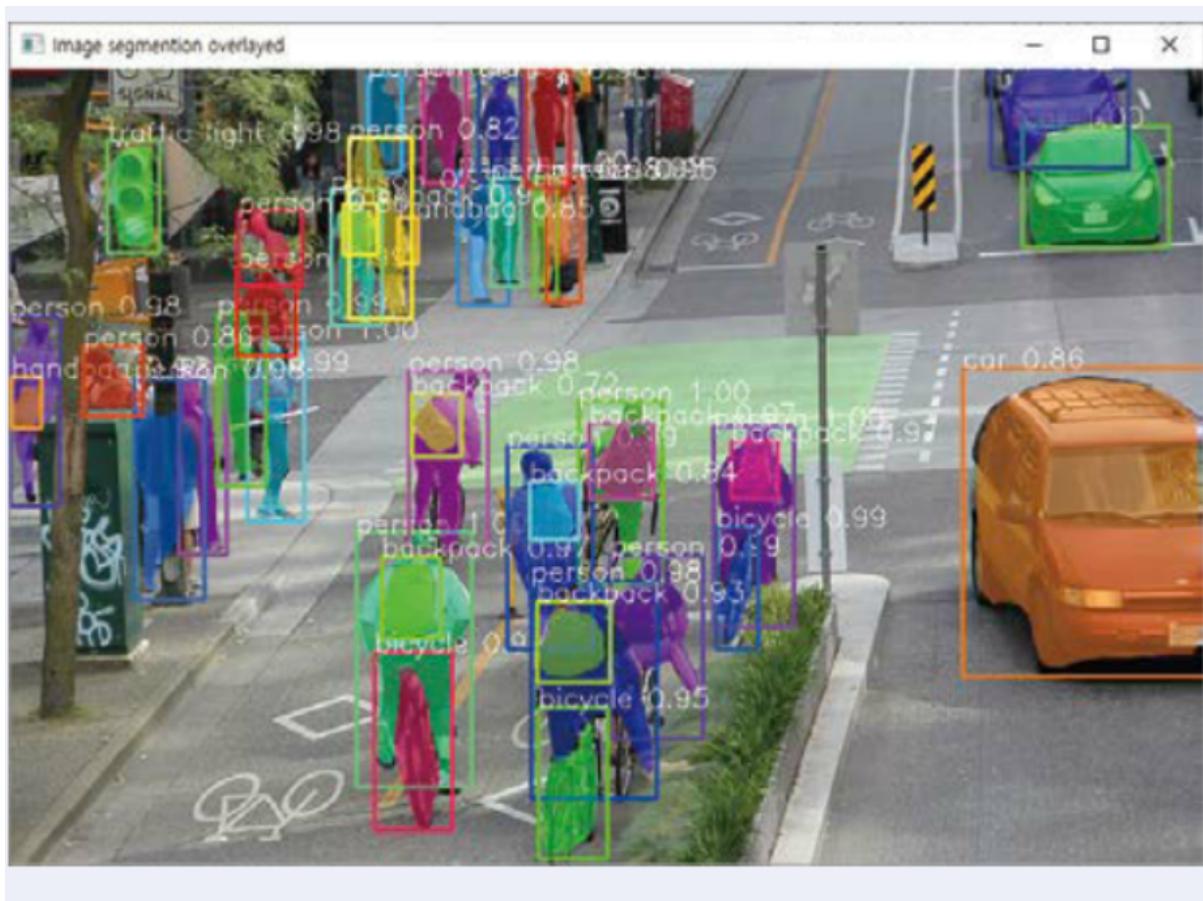
seg=instance_segmentation()
seg.load_model("mask_rcnn_coco.h5")

img_fname='busy_street.jpg'
info,img_segmented=seg.segmentImage(img_fname,show_bboxes=True)

cv.imshow('Image segmentation overlayed',img_segmented)

cv.waitKey()
cv.destroyAllWindows()

```



자동차를 서로 다른 색깔로 칠함 → 같은 부류의 물체가 여럿이면 고유 번호로 구별함.

◆[프로그램 9-8] pixellib 라이브러리로 비디오를 의미 분할하기

```
from pixellib.instance import instance_segmentation
import cv2 as cv

cap=cv.VideoCapture(0) #웹 캠 연결

seg_video=instance_segmentation()
seg_video.load_model("mask_rcnn_coco.h5")
#관심 부류 지정 가능
target_class=seg_video.select_target_classes(person=True,book=True)
# process_camera 함수로 비디오를 사례분할
seg_video.process_camera(cap,segment_target_classes=target_class,frames_per_second=2,show_frames=True,frame_name='Pixellib',show_bboxe
# 웹 캠을 통해 들어오는 비디오 분할
# 두 번째 인수: 관심 부류
# 세 번째 인수: 초당 프레임 수
# 네 번째 인수: 원도우 열어 분할 결과를 실시간으로 디스플레이 하라

cap.release()
cv.destroyAllWindows()
```

pixellib라이브러리는 10여 행의 코드로 분할 결과를 얻을 수 있어 프로그래밍 추상화 수준이 아주 높다.

추상화 수준을 낮추어 자체 데이터셋으로 학습하는 기능도 제공한다.

9.5 [비전 에이전트8] 배경을 내 맘대로 바꾸기

◆[프로그램 9-9] pixellib 라이브러리를 활용해 내 맘대로 배경 바꾸기

```
import cv2 as cv
import numpy as np
from PyQt5.QtWidgets import *
import sys
from pixellib.tune_bg import alter_bg
```

```

class VideoSpecialEffect(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('배경을 내 맘대로')
        self.setGeometry(200, 200, 400, 100)

        videoButton=QPushButton('배경 내 맘대로 켜기', self)
        self.pickCombo=QComboBox(self)
        self.pickCombo.addItems(['원래 영상', '흐릿(조금)', '흐릿(중간)', '흐릿(많이)', '빨강', '녹색', '파랑'])
        # 콤보 박스 메뉴 지정
        quitButton=QPushButton('나가기', self)

        videoButton.setGeometry(10,10,140,30)
        self.pickCombo.setGeometry(150,10,110,30)
        quitButton.setGeometry(280,10,100,30)

        videoButton.clicked.connect(self.videoSpecialEffectFunction)
        quitButton.clicked.connect(self.quitFunction)

    def videoSpecialEffectFunction(self):
        self.cap=cv.VideoCapture(0,cv.CAP_DSHOW)
        if not self.cap.isOpened(): sys.exit('카메라 연결 실패')

        while True:
            ret,frame=self.cap.read()
            if not ret: break

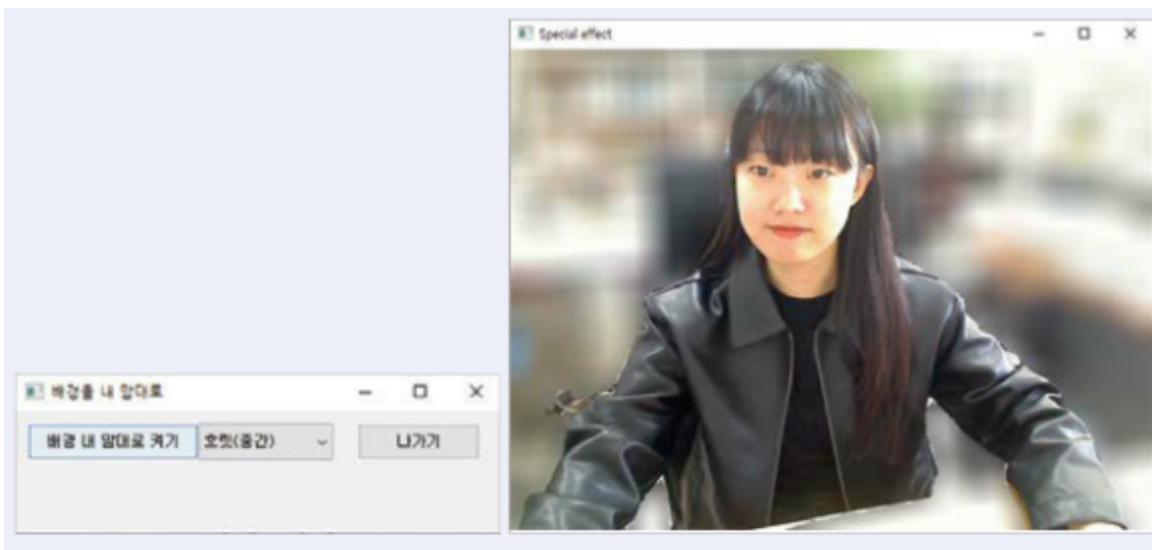
            pick_effect=self.pickCombo.currentIndex()          # 콤보박스에서 선택한 메뉴 알아내고 7가지 경우 처리
            if pick_effect==0:#0은 그대로 디스플레이
                special_img=frame
            elif pick_effect==1:#1-3:배경 흐릿하게하는 세 단계 적용
                special_img=change_bg.blur_frame(frame, low=True, detect='person')
            elif pick_effect==2:
                special_img=change_bg.blur_frame(frame, moderate=True, detect='person')
            elif pick_effect==3:
                special_img=change_bg.blur_frame(frame, extreme=True, detect='person')
            elif pick_effect==4: # 4-6: 배경 빨,초,파로 바꿈. detect= 'person'인수 설정해 사람 이외는 모두 배경으로 간주
                special_img=change_bg.color_frame(frame, colors=(255, 0, 0), detect='person')
            elif pick_effect==5:
                special_img=change_bg.color_frame(frame, colors=(0, 255, 0), detect='person')
            elif pick_effect==6:
                special_img=change_bg.color_frame(frame, colors=(0, 0, 255), detect='person')

            cv.imshow('Special effect',special_img)
            cv.waitKey(1)

    def quitFunction(self):
        self.cap.release()
        cv.destroyAllWindows()
        self.close()

change_bg=alter_bg(model_type="pb") # alter_bg 클래스로 객체 만들
change_bg.load_pascalvoc_model('xception_pascalvoc.pb')
# 파일을 읽어 물체 분할하고 배경 바꾸는데 쓸 모델 준비
app=QApplication(sys.argv)
win=VideoSpecialEffect()
win.show()
app.exec_()

```



9.6 사람 인식

생체 인식(biometrics): 사람의 생리학적 또는 행동학적 특성을 측정하고 유용한 응용에 활용하는 분야



그림 9-40 생체 인식(주황색 표시는 컴퓨터 비전 기술을 사용해야 하는 특성)

TIP 이 책은 얼굴 표정 인식(facial expression recognition)은 다루지 않는데, 이에 관심이 있는 독자는 서베이 논문으로 [Li2022a]과 현재 최고 성능을 한 눈에 파악할 수 있는 PapersWithCode 사이트(<https://paperswithcode.com/task/facial-expression-recognition>)를 참고한다.

얼굴 인식

성별과 나이 추정

Chapter 10 동적 비전

10.1 모션분석

- 비디오(동영상): 시간 순서에 따라 정지 영상을 나열한 구조
- 프레임: 비디오를 구성하는 영상 한 장

→ 프레임(2차원 공간) → 비디오(시간 축이 추가되어 3차원 시공간)

EX) 컬러 영상: 채널 3장 → $m \times n \times 3$ 텐서

T장의 프레임 담은 비디오: $m \times n \times 3 \times T$ (4차원 구조 텐서)

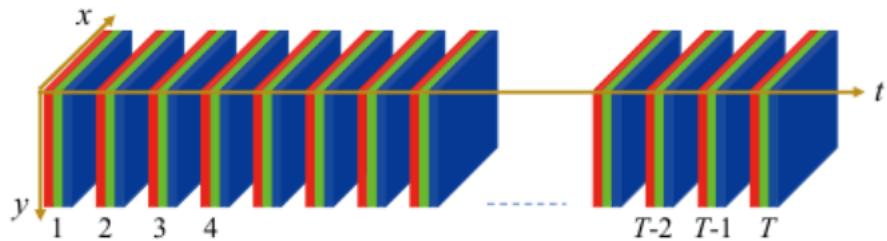


그림 10-2 3차원 공간에 표현되는 비디오

비디오 처리

→ 광류 추정 방법(고전 알고리즘) → 딥러닝 알고리즘

10.1.1 광류 추정 방법 : 모션 벡터와 광류

움직이는 물체는 연속 프레임에 명암 변화를 일으킨다. → 명암 변화를 분석하면 물체의 모션 정보를 알 수 있다.

광류(optical flow): 픽셀별로 모션 벡터를 추정해 기록한 맵

t 순간								t+1 순간							
0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	2	0	0	0	0	0	0	0	1	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	6	6	0	0	0	0	0	0	0	0	0	0
5	0	0	0	5	6	8	0	0	0	0	0	0	6	5	0
6	0	0	0	5	5	6	5	0	0	0	1	7	6	6	0
7	0	0	0	0	0	0	0	0	0	0	0	0	6	6	5

그림 10-3 연속 영상에서 모션 벡터를 추정할 때 발생하는 애매함

어디로 이동했는지 확정이 어려움.

따라서, 모션 벡터 추정 알고리즘은 → 밝기 항상성 조건을 가정한다.

밝기 항상성 : 연속한 두 영상에서 같은 물체는 같은 명암으로 나타난다.

1. dt 가 충분히 작으면, 테일러 급수에 의해 아래 식 성립. 2차 이상 무시

$$f(y + dy, x + dx, t + dt) = f(y, x, t) + \frac{\partial f}{\partial y} dy + \frac{\partial f}{\partial x} dx + \frac{\partial f}{\partial t} dt + 2차 이상의 합 \quad (10.2)$$

2. 밝기 항상성 가정에 의해 $f(y + dy, x + dx, t + dt) = f(y, x, t)$ 성립 →

$$\frac{\partial f}{\partial y} \frac{dy}{dt} + \frac{\partial f}{\partial x} \frac{dx}{dt} + \frac{\partial f}{\partial t} = 0$$

3. 모션 벡터 (v, u) 로 바꾸면,

$$\frac{\partial f}{\partial y}v + \frac{\partial f}{\partial x}u + \frac{\partial f}{\partial t} = 0$$

→ 광류 조건식

$$\frac{\partial f}{\partial y}$$

$$\frac{\partial f}{\partial x}$$

→ y와 x 방향으로의 명암 변화

[예시 10-1] 광류 조건식

계산 편의상 $\frac{\partial f}{\partial y}$ 와 $\frac{\partial f}{\partial x}$ 는 바로 이웃에 있는 화소와 명암 차이로 계산한다. 실제로는 [그림 4-6]의 소벨 연산자처럼 더 큰 연산자를 사용한다.

$$\frac{\partial f}{\partial y} = f(y+1, x, t) - f(y, x, t), \quad \frac{\partial f}{\partial x} = f(y, x+1, t) - f(y, x, t), \quad \frac{\partial f}{\partial t} = f(y, x, t+1) - f(y, x, t)$$

[그림 10-3]에서 노란색으로 표시된 (5,4) 위치의 화소에 위 식을 적용하면 다음과 같다.

$$\frac{\partial f}{\partial y} = f(6, 4, t) - f(5, 4, t) = -1, \quad \frac{\partial f}{\partial x} = f(5, 5, t) - f(5, 4, t) = 2, \quad \frac{\partial f}{\partial t} = f(5, 4, t+1) - f(5, 4, t) = 1$$

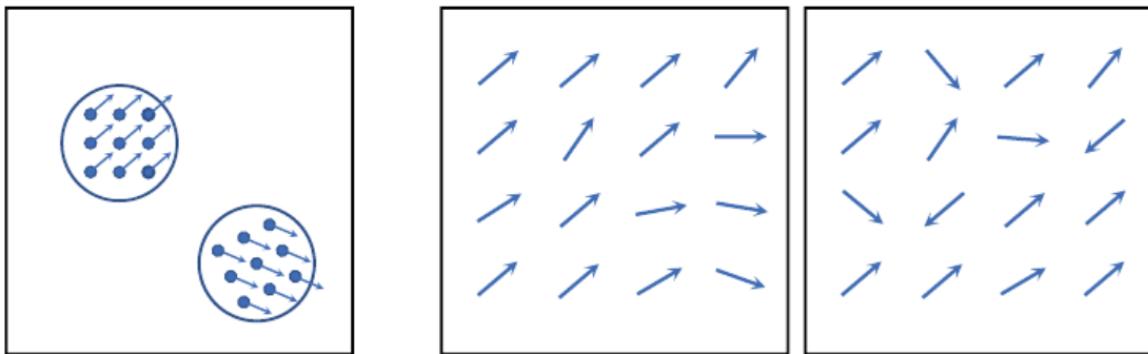
이들 값을 식 (10.4)의 광류 조건식에 대입하면 다음 식을 얻는다. 이 식을 풀면 노란색 점에서 모션 벡터를 얻는다. 그런데 식은 하나인데 변수는 2개이므로 유일한 해를 확정할 수 없다.

$$-v + 2u + 1 = 0$$

한계: 광류 조건식은 해를 유일하게 확정하지 못한다.

광류 추정 알고리즘의 아이디어

: 이웃 화소와 관계를 고려하면 정확한 해 구할 수 있다 .



(a) Lucas-Kanade 알고리즘의 지역 조건 (b) Horn-Schunck 알고리즘의 광역 조건

그림 10-4 광류 추정 알고리즘이 사용하는 가정

- (a) 어떤 화소는 이웃 화소와 유사한 모션 벡터 가진다
- (b) 영상 전체에 걸쳐 모션 벡터는 서서히 변한다.

→ 광류 조건식에 이들 조건 추가로 적용하고 최적화 과정 수행하기

◆프로그램 [10-1]Farneback 알고리즘으로 광류 추정(앞의 두 알고리즘 개선)

```

import numpy as np
import cv2 as cv
import sys

def draw_OpticalFlow(img,flow,step=16):
    for y in range(step//2,frame.shape[0],step):
        for x in range(step//2,frame.shape[1],step):
            dx,dy=fflow[y,x].astype(np.int)
            if(dx*dx+dy*dy)>1:
                cv.line(img,(x,y),(x+dx,y+dy),(0,0,255),2) # 큰 모션 있는 곳은 빨간색
            else:
                cv.line(img,(x,y),(x+dx,y+dy),(0,255,0),2)

cap=cv.VideoCapture(0,cv.CAP_DSHOW) # 카메라와 연결 시도
if not cap.isOpened(): sys.exit('카메라 연결 실패')

prev=None

while(1):
    ret,frame=cap.read() # 비디오를 구성하는 프레임 획득
    if not ret: sys('프레임 획득에 실패하여 루프를 나갑니다.')
    
    if prev is None: # 첫 프레임이면 광류 계산 없이 prev만 설정
        prev=cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
        continue

    curr=cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
    flow=cv.calcOpticalFlowFarneback(prev,curr,None,0.5,3,15,3,5,1.2,0)

    draw_OpticalFlow(frame,flow)
    cv.imshow('Optical flow',frame)

    prev=curr

    key=cv.waitKey(1) # 1밀리초 동안 키보드 입력 기다림
    if key==ord('q'): # 'q' 키가 들어오면 루프를 빠져나감
        break

cap.release() # 카메라와 연결을 끊음
cv.destroyAllWindows()

```

희소 광류 추정을 이용한 KLT 추적

지역 특징을 추적에 유리하도록 개조한 특징을 추출한 다음 이들 특징을 광류정보를 이용해 추적함

지역 특징으로 추출된 점에서만 모션 벡터를 추정 → 희소 광류

◆프로그램 [10-2] KLT 추적 알고리즘으로 물체 추적하기

```

import numpy as np
import cv2 as cv

cap=cv.VideoCapture('slow_traffic_small.mp4')

feature_params=dict(maxCorners=100,qualityLevel=0.3,minDistance=7,blockSize=7)
lk_params=dict(winSize=(15,15),maxLevel=2,criteria=(cv.TERM_CRITERIA_EPS|cv.TERM_CRITERIA_COUNT,10,0.03))

color=np.random.randint(0,255,(100,3))

ret,old_frame=cap.read()      # 첫 프레임
old_gray=cv.cvtColor(old_frame,cv.COLOR_BGR2GRAY)
p0=cv.goodFeaturesToTrack(old_gray,mask=None,**feature_params)

mask=np.zeros_like(old_frame) # 물체의 이동 궤적을 그릴 영상

while(1):
    ret,frame=cap.read()
    if not ret: break

    new_gray=cv.cvtColor(frame,cv.COLOR_BGR2GRAY)
    p1,match,err=cv.calcOpticalFlowPyrLK(old_gray,new_gray,p0,None,**lk_params) # 광류 계산

    if p1 is not None:      # 양호한 쌍 선택
        good_new=p1[match==1]
        good_old=p0[match==1]

    for i in range(len(good_new)): # 이동 궤적 그리기
        a,b=int(good_new[i][0]),int(good_new[i][1])
        c,d=int(good_old[i][0]),int(good_old[i][1])
        mask=cv.line(mask,(a,b),(c,d),color[i].tolist(),2)
        frame=cv.circle(frame,(a,b),5,color[i].tolist(),-1)

    img=cv.add(frame,mask)
    cv.imshow('LTK tracker',img)
    cv.waitKey(30)

    old_gray=new_gray.copy() # 이번 것이 이전 것이 됨
    p0=good_new.reshape(-1,1,2)

cv.destroyAllWindows()

```

10.1.2 딥러닝 기반 광류 추정

◆FlowNet : 광류 추정에 CNN을 적용한 최초 논문

DeConvNet 같은 신경망을 광류에 적용한다.

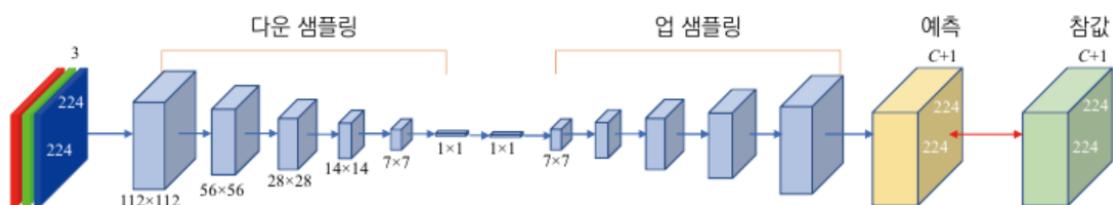


그림 9-35 DeConvNet의 구조

- 입력: 컬러 영상 두 장 쌍은 $384 \times 512 \times 6$ 텐서
- 다운 샘플링: 점점 크기 줄여 $6 \times 8 \times 1024$ 의 특징 맵 만들
- 업 샘플링: 점점 키워 136×320 의 광류 맵 출력

◆또 다른 신경망

- $384 \times 512 \times 3$ 영상 두장을 별도의 컨볼루션층으로 다운 샘플링
- 2개의 특징 맵을 결합하여 업 샘플링

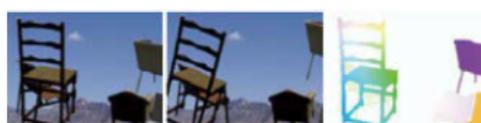
→ 광류는 매 pixel마다 모션 벡터 지정해야하므로 참값 만들기가 매우 어렵다.



(a) KITTI 데이터셋



(b) Sintel 데이터셋



(c) Flying chairs 데이터셋

그림 10-5 광류 데이터셋[Dosovitskiy2015]

◆ 광류 추정의 성능을 개선하려는 시도

1. 입력 영상을 의미 분할한 다음, 움직이지 않는 부류, 움직이는 부류, 살랑살랑 움직이는 부류로 구분하고 서로 다른 수식으로 광류 추정하고 결합.[Sevilla-Lara2016]
2. 자율주행 영상을 대상으로 알고리즘 구상. 자동차는 가림이 발생하므로 서로 다른 자동차를 구분해주는 사례분할 적용함[Bai2016]



그림 10-6 사례 분할을 활용한 광류 추정[Bai2016]

3. RAFT: t 순간의 특징맵, t+1순간의 영상을 1,2,4,8로 줄인 특징맵 각각에 대해 화소별 유사도를 계산한 비용 볼륨을 만듦[Tedd 2020]. 얘를 순환신경망에 입력해 광류 맵 추정
4. 트랜스토머는 컨볼루션층을 전혀 사용하지 않고 주목만으로 높은 성능 발휘했다 [Vaswani2017]
주목: 특징 사이의 관련성 분석하여 성능을 향상할 목적으로 컴퓨터 비전이 애용한다.
- 광류 추정에 트랜스포머를 적용한 최초 논문은 2021년에 발표됨.
5. FlowFormer는 RAFT가 사용했던 비용 볼륨을 트랜스포머에 적용하여 추가적인 성능향상을 얻는다 [Huang2022]

10.2 추적

KLT(지역 특징을 추적, 고전 추적 알고리즘의 표준)의 한계:

뚜렷하게 특징점이 나타나지 않는 물체는 추적하지 못한다.

추적하는 물체가 뭔지 알 수 없다.

여러 점이 같은 물체에 속하는지에 대한 정보가 없다.

10.2.1 문제의 이해

물체 추적 문제는 추적할 물체의 개수에 따라

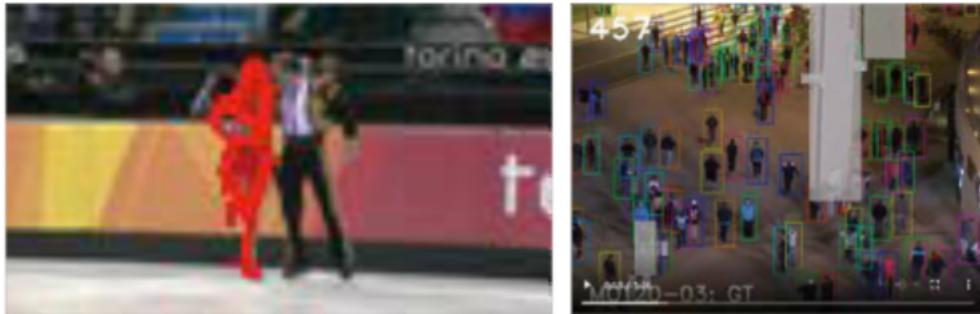
VOT(시각 물체 추적), **MOT**(다중 물체 추적)으로 나눈다.

- **VOT(시각 물체 추적)**

: 첫 프레임에서 물체 하나 지정 → 이후 프레임에서 그 물체를 추적하는 문제

- **MOT(다중 물체 추적)**

: 첫 프레임에서 추적할 물체의 부류 지정 → 단일 카메라로 수집한 비디오에서 여러 물체를 검출



(a) VOT(iceskater2 비디오)

(b) MOT(MOT20-03 비디오의 457번째 프레임)

그림 10-7 VOT 대회와 MOT 대회가 제공하는 데이터셋

TIP 각 데이터셋의 공식 사이트는 <https://votchallenge.net>와 <https://motchallenge.net>이다.

추적 문제는 배치 방식과 온라인 방식으로 나눌 수 있다.

- 배치방식: t 순간을 처리할 때 $t+1, t+2, \dots, T$ 순간의 프레임 활용
ex. 이미 녹화한 경기 분석해 선수의 장단점 파악
- 온라인 방식: 지난 순간의 프레임만 활용할 수 있다.
ex. 실시간 감시 시스템에서는 온라인 방식만 활용

→ VOT, MOT는 한 대의 카메라에서 입력된 비디오 한정

◆다중 카메라 추적(MCT : Multi-Camera Tracking): 장기 재식별이 중요

AI City 대회

- 도시 규모의 비디오 데이터 셋을 가지고 성능 겨루
- 차량 개수 세기, 차량 재식별, 다중 목표 다중 카메라 환경에서 자동차 추적, 교통 이상 상황 검출, 자연어 부문

<https://www.aicitychallenge.org/>

→ 현재 연구는 주로 사람이나 자동차를 추적 대상으로 함. 다른 종류 물체 추적해야하는 응용 분야도 있음.

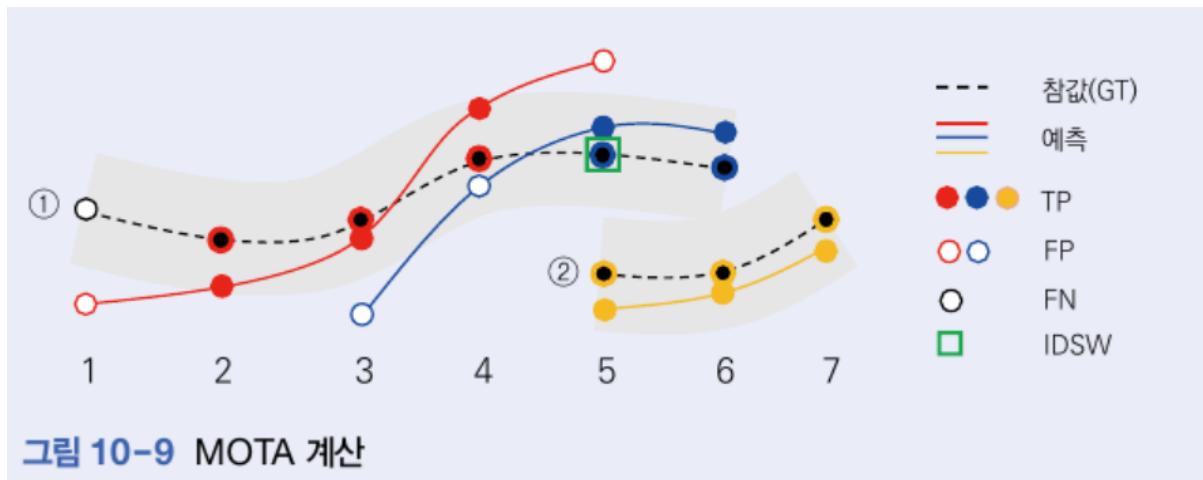
10.2.2 MOT의 성능척도

◆ MOTA(MOT Accuracy)

$$\text{MOTA} = 1 - \frac{\sum_{t=1,T} (FN_t + FP_t + IDSW_t)}{\sum_{t=1,T} GT_t}$$

- $GT_t, FN_t, FP_t, IDSW_t$ 은 t 순간에서 참값, 거짓 부정, 거짓 긍정, 번호 교환 오류의 개수
- 모든 프레임에서 FN, FP, IDSW를 세야한다.
- IoU가 임곗값을 넘으면 매칭으로 판단
- 이전 프레임 박스를 현재 프레임 박스에 할당하는 문제는 헝가리안 알고리즘으로 해결

→ 매칭, 할당이 일어났다고 가정하고 MOTA를 계산하는 과정



[그림 10-9]는 비디오의 길이 $T=7$ 인 경우다. 점선 궤적은 참값을 나타내는데, 1~6 순간을 지속하는 궤적 ①과 5~7 순간을 지속하는 궤적 ②가 있다. 회색 띠는 IoU 임곗값 0.5를 넘는 범위를 뜻한다. 추적 알고리즘은 빨간색, 파란색, 주황색의 궤적 3개를 찾았다.

- 1: IoU 0.5 넘지 못해 매칭X
FN, FP하나씩 발생
- 2 : 매칭
TP 하나 발생
- 3: **빨간색 매칭 → TP**
파란색 → FP
- 4: **빨간색이 매칭 → TP**(파란색과 더 가깝지만 이전 순간을 고려해 내린 결정)
파란색 → FP
- 5: **빨간색 매칭X → FP**
파란색 → TP
궤적1 입장에서 다른 물체와 쌍이 맞아졌으므로 **IDSW(번호교환)오류** 발생
궤적2 새로 등장해 TP추가 발생
- 6: TP 2개 발생
- 7: TP 1개 발생

$$MOTA = 1 - \frac{(1+1+0)+(0+0+0)+(0+1+0)+(0+1+0)+(0+1+1)+(0+0+0)+(0+0+0)}{(1+1+1+1+2+2+1)} = 1 - \frac{6}{9} = 0.3333$$

(단점) 쌍 맺기 성공률 낮더라도 검출 성공률 높으면 좋은 점수를 부여

→ HOTA : 쌍 맺기, 검출 성능 균형있게 고려함

10.2.3 딥러닝 기반 추적

딥러닝 추적 알고리즘

딥러닝 모델로 물체 검출 → 이웃한 프레임에서 검출된 물체 집합을 매칭해 쌍을 맺는다.

◆ 일반적인 처리절차

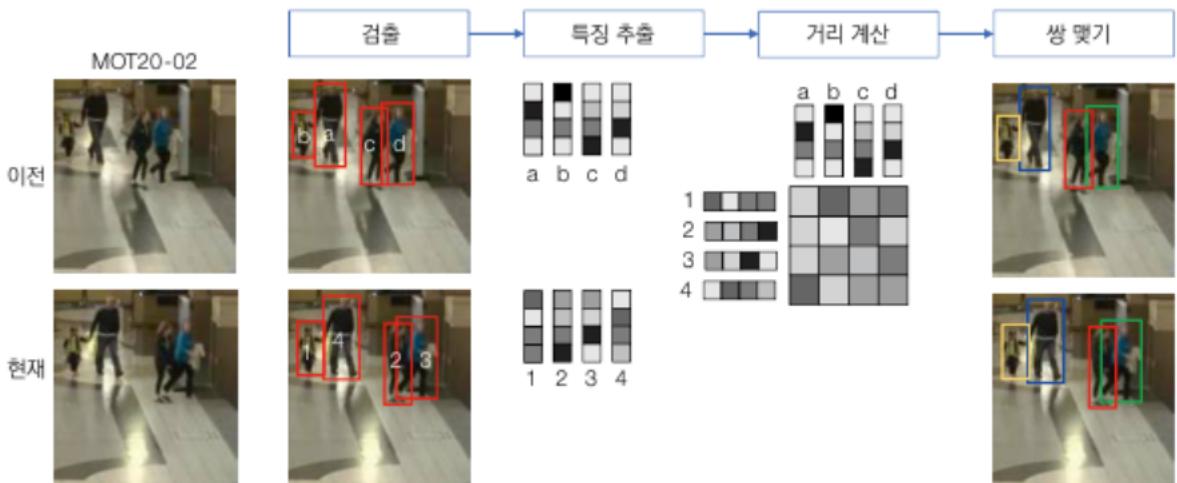


그림 10-10 물체 추적의 네 단계 처리 과정

- 물체 검출 알고리즘 적용해 박스 찾기(RCNN, YOLO)
- 박스에서 특징 추출(위치 정보활용, CNN으로 특징 추출)
- 이전 프레임 박스와 현재 프레임 박스의 거리 행렬 구하기
특징으로 박스 위치 \rightarrow 1-IoU를 거리로 사용
CNN으로 특징 추출 \rightarrow 특징 벡터 사이의 거리 사용
- 거리행렬 분석해 이전 프레임 박스와 현재 프레임의 박스를 쌍으로 맺어 이동 궤적을 구상 (헝가리안 알고리즘)

♦SORT

- 물체 검출 : 현재 순간 t 프레임에 faster RCNN 적용해 물체 검출 \rightarrow 사람 박스 남기고 나머지 버림($B_{detection}$)
- 특징 추출: 이전 순간의 목표물 위치정보와 이동 이력 정보 사용

$$\mathbf{b} = (x, y, s, r, \dot{x}, \dot{y}, \dot{s})$$

- x, y 는 목표물의 중심위치, s 는 크기, r 은 높이와 너비 비율(고정된 값)
 - 나머지는 이전에 이동했던 정보를 누적한 이력정보
- 이동량 나타내는 x', y', s' 를 각각 x, y, s 에 더해 t순간의 박스 예측 $\rightarrow B_{predict}$ 예
- 거리행렬: detection box와 predict box의 IoU계산하고 거리변환(1-IoU)
 - 매칭 쌍 찾기(헝가리안 알고리즘)

[예시 10-3] 헝가리안 알고리즘으로 최적의 매칭 쌍 구하기

헝가리안 알고리즘은 최소 비용이 되도록 작업자에게 과업을 할당하는 최적화 알고리즘이다. 예를 들어 1~3 작업자에 a~c 작업을 할당하는데 [그림 10-11(a)]의 비용 행렬이 주어졌다면 1-c, 2-a, 3-b 할당이 최적이다.

SORT에서는 행에 현재 프레임에 해당하는 $B_{detection}$, 열에 이전 프레임에서 예측된 $B_{predict}$ 을 배치한다. $B_{detection}=\{1,2,3,4\}$ 가 있고, $B_{predict}=\{a,b,c\}$ 가 있다고 가정한다. 박스 쌍의 IoU를 계산하고 1-IoU를 해당 요소에 기록하여 [그림 10-11(b)]의 거리 행렬을 얻었다고 가정한다. 4열에는 가상의 박스 d를 배치하였다. 이 행렬에 헝가리안 알고리즘을 적용하면 1-b, 2-d, 3-a, 4-c 쌍을 얻는데 2-d 쌍은 버린다.

TIP 헝가리안 알고리즘의 구체적인 동작 원리는 다음 문서를 참고한다.

https://web.archive.org/web/20120105112913/http://www.math.harvard.edu/archive/20_spring_05/handouts/assignment_overheads.pdf

	a	b	c
1	2	5	1
2	1	3	4
3	2	3	6

(a) 작업자에게 과업을 할당하는 문제

	a	b	c	d
1	0.9	0.2	0.7	1.0
2	0.7	0.4	0.8	1.0
3	0.1	0.3	1.0	1.0
4	1.0	0.7	0.3	1.0

(b) $B_{predict}$ 박스를 $B_{detection}$ 박스에 할당하는 문제

그림 10-11 헝가리안 알고리즘

SORT: 거리행렬 구할 때 박스의 IoU만 사용하여 IDSW 오류 많음

→ DeepSORT : 박스의 IoU + CNN으로 구한 특징 같이 사용

10.2.4 프로그래밍 실습: SORT로 사람 추적

```
import numpy as np
import cv2 as cv
import sys

def construct_yolo_v3():
    f=open('coco_names.txt', 'r')
    class_names=[line.strip() for line in f.readlines()]

    model=cv.dnn.readNet('yolov3.weights','yolov3.cfg')
    layer_names=model.getLayerNames()
    out_layers=[layer_names[i-1] for i in model.getUnconnectedOutLayers()]

    return model,out_layers,class_names

def yolo_detect(img,yolo_model,out_layers):
    height,width=img.shape[0],img.shape[1]
    test_img=cv.dnn.blobFromImage(img,1.0/256,(448,448),(0,0,0),swapRB=True)

    yolo_model.setInput(test_img)
    output3=yolo_model.forward(out_layers)

    box,conf,id=[[[],[],[]], # 박스, 신뢰도, 부류 번호
    for output in output3:
```

```

for vec85 in output:
    scores=vec85[5:]
    class_id=np.argmax(scores)
    confidence=scores[class_id]
    if confidence>0.5: # 신뢰도가 50% 이상인 경우만 취함
        centerx,centery=int(vec85[0]*width),int(vec85[1]*height)
        w,h=int(vec85[2]*width),int(vec85[3]*height)
        x,y=int(centerx-w/2),int(centery-h/2)
        box.append([x,y,x+w,y+h])
        conf.append(float(confidence))
        id.append(class_id)

ind=cv.dnn.NMSBoxes(box,conf,0.5,0.4)
objects=[box[i]+[conf[i]]+[id[i]] for i in range(len(box)) if i in ind]
return objects

# 여기까지 사전 학습된 YOLO v3를 읽어 모델 구성하는 construct_yolo_v3함수와 모델 검출하는 yolo_detect함수가 들어있다.

model,out_layers,class_names=construct_yolo_v3() # YOLO 모델 구성하고 객체 저장
colors=np.random.uniform(0,255,size=(100,3)) # 100개 색 만들어 저장(추적하는 물체를 서로 다른 색으로 표시하기 위해)

from sort import Sort

sort=Sort()

cap=cv.VideoCapture(0,cv.CAP_DSHOW) #웹 캠 연결
if not cap.isOpened(): sys.exit('카메라 연결 실패')

while True:
    ret,frame=cap.read() #웹 캠으로 비디오에서 프레임 읽어 frame에 저장
    if not ret: sys.exit('프레임 획득에 실패하여 루프를 나갑니다.')

    res=yolo_detect(frame,model,out_layers)#yolo_detect: frame에서 물체 검출해 res에 저장
    persons=[res[i] for i in range(len(res)) if res[i][5]==0] # 부류 0은 사람, 사람만 골라 저장

    if len(persons)==0: #검출된 사람 X
        tracks=sort.update()
    else:#검출된 사람 O
        tracks=sort.update(np.array(persons))

        # update 함수 호출: 새로 검출된 persons정보와 이전 이력 정보 보고 객체 내부나 추적 정보 갱신, 현재 프레임의 물체 번호
        for i in range(len(tracks)): #추적 번호에 따라 색 달리해 직사각형과 물체 번호 표시
            x1,y1,x2,y2,track_id=tracks[i].astype(int)
            cv.rectangle(frame,(x1,y1),(x2,y2),colors[track_id],2)
            cv.putText(frame,str(track_id),(x1+10,y1+40),cv.FONT_HERSHEY_PLAIN,3,colors[track_id],2)

    cv.imshow('Person tracking by SORT',frame)
    # 추적 물체 표시한 영상을 디스플레이
    key=cv.waitKey(1)
    if key==ord('q'): break

cap.release() # 카메라와 연결을 끊음
cv.destroyAllWindows()

```



카메라에서 벗어났다 들어오면 다른 번호가 부여됨
→ SORT는 재식별을 하지 않으므로 다른 물체로 간주

10.3 MediaPipe를 이용해 비디오에서 사람 인식

10.3.1 얼굴 검출(BlazeFace)

BlazeFace는 SSD(faster RCNN을 개조한 모델)를 얼굴 검출에 맞게 개조한 것.

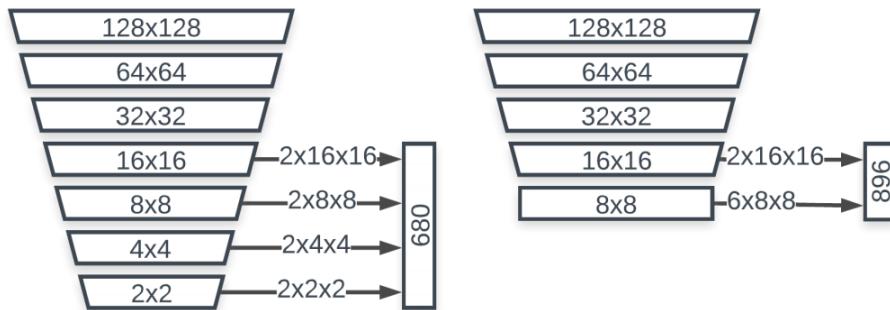


Figure 2. Anchor computation: SSD (left) vs. BlazeFace

- 가로세로 비율을 1로 고정하고 크기만 다른 박스를 ○렷 예측
- 크기는 8x8 까지만 줄임으로써 속도 향상 이룸
- SSD: 예측 박스가 여러 개 겹치는 문제를 비최대 억제로 해결 → BlazeFace는 여러 박스의 정보를 가중 평균하여 정확률 향상
- 얼굴 표시 박스 + 랜드마크 6개(눈 중심, 입 중심 등) 같이 출력

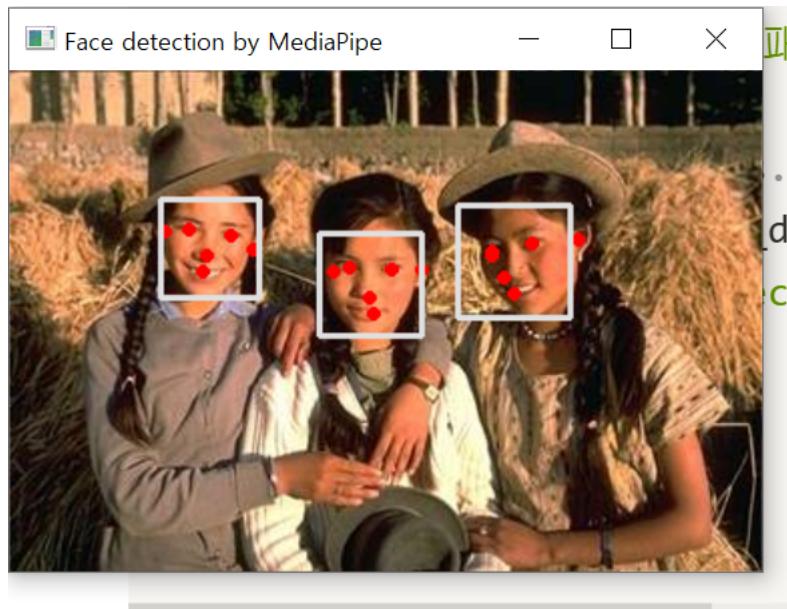
◆[프로그램10-4] BlazeFace로 얼굴 검출

```
import cv2 as cv
import mediapipe as mp
#영상 읽음
img=cv.imread('BSDS_376001.jpg')
#mediapipe모듈의 solution에서 얼굴 검출을 담당하는 모듈
mp_face_detection=mp.solutions.face_detection
#검출 결과를 그리는 모듈
mp_drawing=mp.solutions.drawing_utils

face_detection=mp_face_detection.FaceDetection(model_selection=1,min_detection_confidence=0.5)
```
FaceDetection 클래스
model_selection: 0(카메라로부터 2미터 이내) / 1(5미터 이내)
min_detection_confidence: 0~1 사이의 실수 설정, 검출 신뢰도가 설정 값보다 높으면 검출 성공
```
res=face_detection.process(cv.cvtColor(img, cv.COLOR_BGR2RGB))
#실제 검출 수행하고 결과 저장. Mediapipe는 RGB채널 순서 사용하므로 변환해줌.
#res: 검출한 얼굴 정보 들어있음

if not res.detections:
    print('얼굴 검출에 실패했습니다. 다시 시도하세요.')
else: #검출한 얼굴이 있으면?
    for detection in res.detections:
        mp_drawing.draw_detection(img,detection)
    cv.imshow('Face detection by MediaPipe',img)

cv.waitKey()
cv.destroyAllWindows()
```



→ 얼굴의 랜드마크를 잘 검출한 모습

```
In [1]: print(res.detections)
[{"label_id": 0,
 "score": 0.8932861089706421,
 "location_data": {
     "format": "RELATIVE_BOUNDING_BOX",
     "relative_bounding_box": { "xmin": 0.1996, "ymin": 0.2552, "width": 0.1337, "height": 0.2004},
     "relative_keypoints": { "x": 0.2414, "y": 0.3165},
     "relative_keypoints": { "x": 0.2967, "y": 0.3340}
     ...
 },
 "label_id": 0,
 "score": 0.8440857529640198
     ...
 },
 {"label_id": 0,
 "score": 0.813656747341156
 1}
```

score: 검출 신뢰도, relative_bounding_box: 박스

6개의 relative_keypoints

⇒ 얼굴 3개 검출했으므로 리스트 요소 3개

◆[프로그램10-5] 비디오에서 얼굴 검출

```
import cv2 as cv
import mediapipe as mp

mp_face_detection=mp.solutions.face_detection
mp_drawing=mp.solutions.drawing_utils

face_detection=mp_face_detection.FaceDetection(model_selection=1,min_detection_confidence=0.5)

cap=cv.VideoCapture(0,cv.CAP_DSHOW) #웹 캠 연결 시도

while True:
    ret,frame=cap.read() #웹 캠에서 프레임 읽음
    if not ret:
```

```

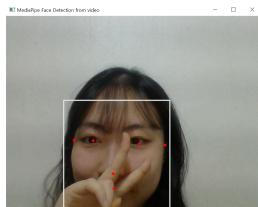
print('프레임 획득에 실패하여 루프를 나갑니다.')
break

res=face_detection.process(cv.cvtColor(frame, cv.COLOR_BGR2RGB))
#process로 검출 실행해 res에 저장
if res.detections:
    for detection in res.detections:
        mp_drawing.draw_detection(frame,detection)

cv.imshow('MediaPipe Face Detection from video',cv.flip(frame,1))
#flip함수로 좌우반전한 모습을 보여줌
if cv.waitKey(5)==ord('q'):
    break

cap.release()
cv.destroyAllWindows()

```



→ 증강현실 기능 추가(6개 랜드마크에 장신구 붙임)

```

import cv2 as cv
import mediapipe as mp

dice=cv.imread('dice.png',cv.IMREAD_UNCHANGED) # 증강 현실에 쓸 장신구
dice=cv.resize(dice,dsizes=(0,0),fx=0.1,fy=0.1)
w,h=dice.shape[1],dice.shape[0]

mp_face_detection=mp.solutions.face_detection
mp_drawing=mp.solutions.drawing_utils

face_detection=mp_face_detection.FaceDetection(model_selection=1,min_detection_confidence=0.5)

cap=cv.VideoCapture(0,cv.CAP_DSHOW)

while True:
    ret,frame=cap.read()
    if not ret:
        print('프레임 획득에 실패하여 루프를 나갑니다.')
        break

    res=face_detection.process(cv.cvtColor(frame, cv.COLOR_BGR2RGB))

    if res.detections:
        for det in res.detections:
            # 검출된 얼굴에 장신구 달고 디스플레이
            p=mp_face_detection.get_key_point(det,mp_face_detection.FaceKeyPoint.RIGHT_EYE)
            # get_key_point: 검출된 얼굴 정보를 담고있는 det에서 오른쪽 눈 위치를 꺼내 p에 저장
            x1,x2=int(p.x*frame.shape[1]-w//2),int(p.x*frame.shape[1]+w//2)
            y1,y2=int(p.y*frame.shape[0]-h//2),int(p.y*frame.shape[0]+h//2)
            # 장신구 배치를 위한 장신구 영상의 원쪽위와 오른쪽 아래 좌표 계산
            if x1>0 and y1>0 and x2<frame.shape[1] and y2<frame.shape[0]: # 장신구가 원본영상안에 머무는지 확인
                alpha=dice[:, :, 3]/255 # 투명도를 나타내는 알파값
                frame[y1:y2,x1:x2]=frame[y1:y2,x1:x2]*(1-alpha)+dice[:, :, 3]*alpha
                #원본영상의 해당위치에 장신구 혼합

cv.imshow('MediaPipe Face AR',cv.flip(frame,1))
if cv.waitKey(5)==ord('q'):
    break

cap.release()
cv.destroyAllWindows()

```



10.3.2 얼굴 그물망 검출

얼굴 영상 인식 & 응용

얼굴 검출 + 얼굴 정렬

- 얼굴 정렬: 얼굴에서 랜드마크 검출

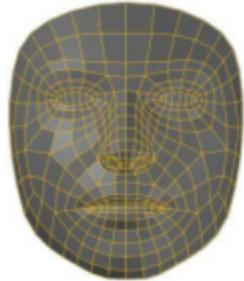


그림 10-12 FaceMesh가 사용하는 468개의 얼굴 랜드마크[Kartynnik2019]

◆[프로그램10-7] FaceMesh로 얼굴 그물망 검출하기

```
import cv2 as cv
import mediapipe as mp

mp_mesh=mp.solutions.face_mesh #얼굴 그물망 검출 담당
mp_drawing=mp.solutions.drawing_utils # 검출 결과 그림
mp_styles=mp.solutions.drawing_styles # 그리는 유형 지정

mesh=mp_mesh.FaceMesh(max_num_faces=2,refine_landmarks=True,min_detection_confidence=0.5,min_tracking_confidence=0.5)
#mp_mesh.FaceMesh(최대 얼굴 처리 개수, 눈,입에 있는 랜드마크를 더 정교하게 검출하라,검출 신뢰도가 0.5이상일때 성공으로 간주,랜드마크 추적 신뢰도가 0.5보다 작으면

cap=cv.VideoCapture(0,cv.CAP_DSHOW)

while True:
    ret,frame=cap.read()
    if not ret:
        print('프레임 획득에 실패하여 루프를 나갑니다.')
        break

    res=mesh.process(cv.cvtColor(frame,cv.COLOR_BGR2RGB))
    #검출 수행
    if res.multi_face_landmarks: #검출된 얼굴 있는지 확인
        for landmarks in res.multi_face_landmarks: #검출한 얼굴 각각에 대해 그물망 그리는 일 반복
            mp_drawing.draw_landmarks(image=frame,landmark_list=landmarks,connections=mp_mesh.FACEMESH_TESSELATION,landmark_drawing_spec=
            #그물망 그림
            mp_drawing.draw_landmarks(image=frame,landmark_list=landmarks,connections=mp_mesh.FACEMESH_CONTOURS,landmark_drawing_spec=
            #얼굴경계와 눈썹 그림
            mp_drawing.draw_landmarks(image=frame,landmark_list=landmarks,connections=mp_mesh.FACEMESH_IRISES,landmark_drawing_spec=
            #눈동자 그림
            cv.imshow('MediaPipe Face Mesh',cv.flip(frame,1)) # 좌우반전
            if cv.waitKey(5)==ord('q'):
                break

cap.release()
cv.destroyAllWindows()
```





(a) 22~24행에서 24행만 남기고 실행한 결과

(b) 23행만 남긴 뒤 수정 후 실행한 결과

그림 10-13 다양한 얼굴 그물망을 적용하기 위해 [프로그램 10-7]의 22~24행을 변형한 경우

10.3.3 손 랜드마크 검출

BlazeHand : 프레임마다 손 검출 적용하지 않고 모션 정보를 이용해 이전 프레임에서 검출한 랜드마크를 현재 프레임에서 예측하는 방법

얼굴 그물망, 손 랜드마크 모두 3차원 좌표로 표현

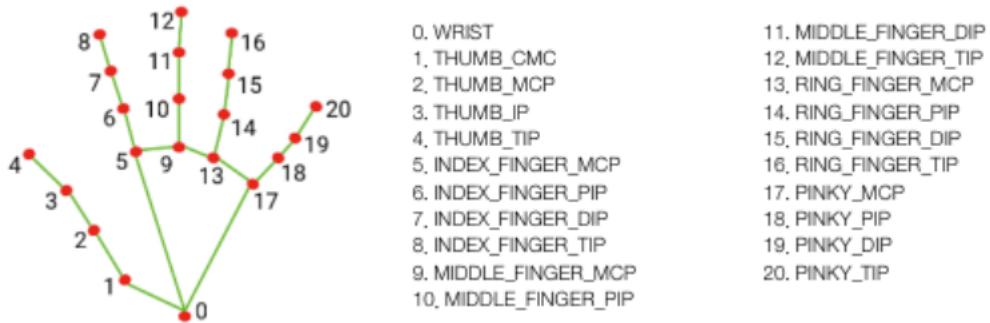


그림 10-14 손을 위한 21개의 랜드마크[Zhang2020]

◆[프로그램10-8] 손 랜드마크 검출하기

```

import cv2 as cv
import mediapipe as mp

mp_hand=mp.solutions.hands # 손 검출 담당하는 모듈
mp_drawing=mp.solutions.drawing_utils
mp_styles=mp.solutions.drawing_styles

hand=mp_hand.Hands(max_num_hands=2, static_image_mode=False, min_detection_confidence=0.5, min_tracking_confidence=0.5)

cap=cv.VideoCapture(0,cv.CAP_DSHOW)

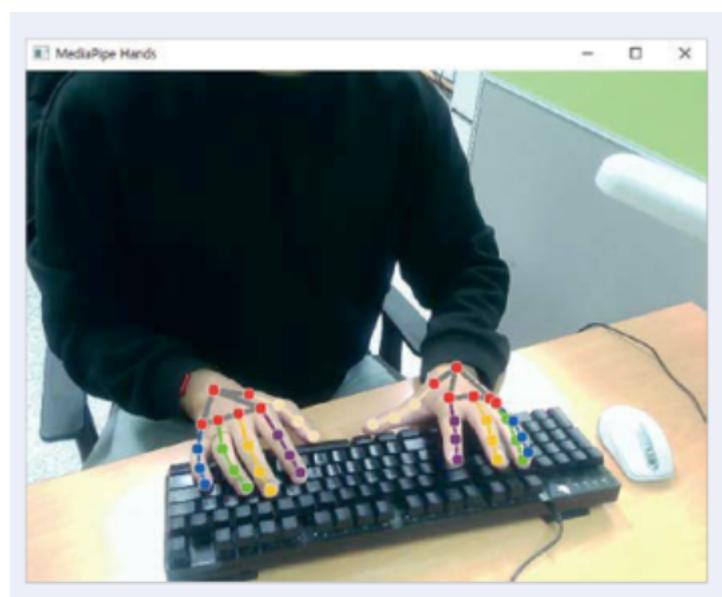
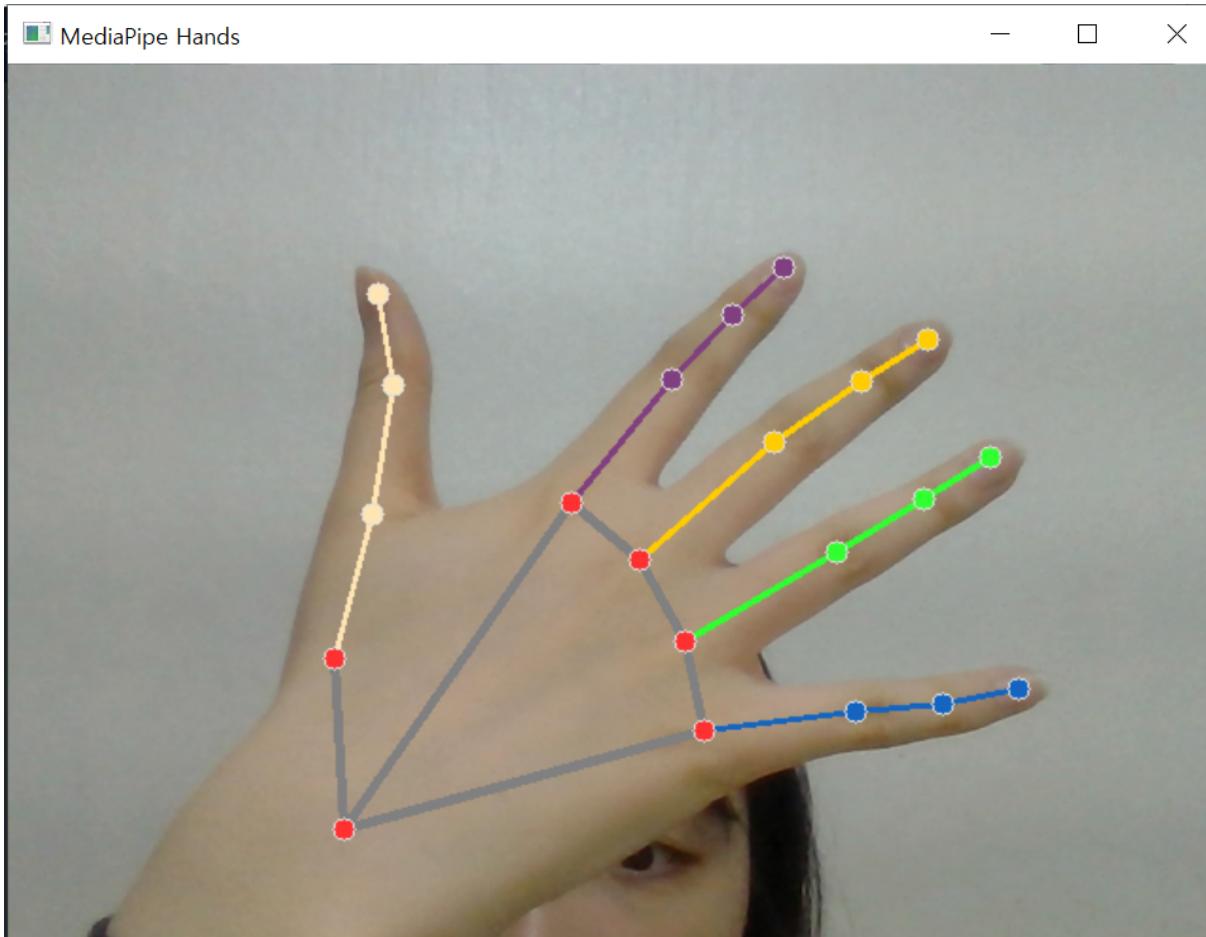
while True:
    ret,frame=cap.read()
    if not ret:
        print('프레임 획득에 실패하여 루프를 나갑니다.')
        break

    res=hand.process(cv.cvtColor(frame, cv.COLOR_BGR2RGB))

    if res.multi_hand_landmarks:
        for landmarks in res.multi_hand_landmarks:

```

```
mp_drawing.draw_landmarks(frame, landmarks, mp_hand.HAND_CONNECTIONS, mp_styles.get_default_hand_landmarks_style(), mp_styles.  
cv.imshow('MediaPipe Hands',cv.flip(frame,1)) # 좌우반전  
if cv.waitKey(5)==ord('q'):  
    break  
  
cap.release()  
cv.destroyAllWindows()
```



10.4 자세 추정과 행동 분류

자세 추정한 결과로 행동 분류 수행

10.4.1 자세추정

정지 영상 또는 비디오를 분석해 전신에 있는 관절(랜드마크, 키포인트) 위치를 알아냄.

10.4.2 BlazePose를 이용한 자세 추정

```
import cv2 as cv
import mediapipe as mp

mp_pose=mp.solutions.pose #자세 추정 담당
mp_drawing=mp.solutions.drawing_utils
mp_styles=mp.solutions.drawing_styles

pose=mp_pose.Pose(static_image_mode=False,enable_segmentation=True,min_detection_confidence=0.5,min_tracking_confidence=0.5)
# 첫 번째 인수: 첫 프레임에 ROI검출 적용하고 이후에는 추적을 사용하라
# 두 번째 인수: 전경, 배경 분할
cap=cv.VideoCapture(0,cv.CAP_DSHOW)

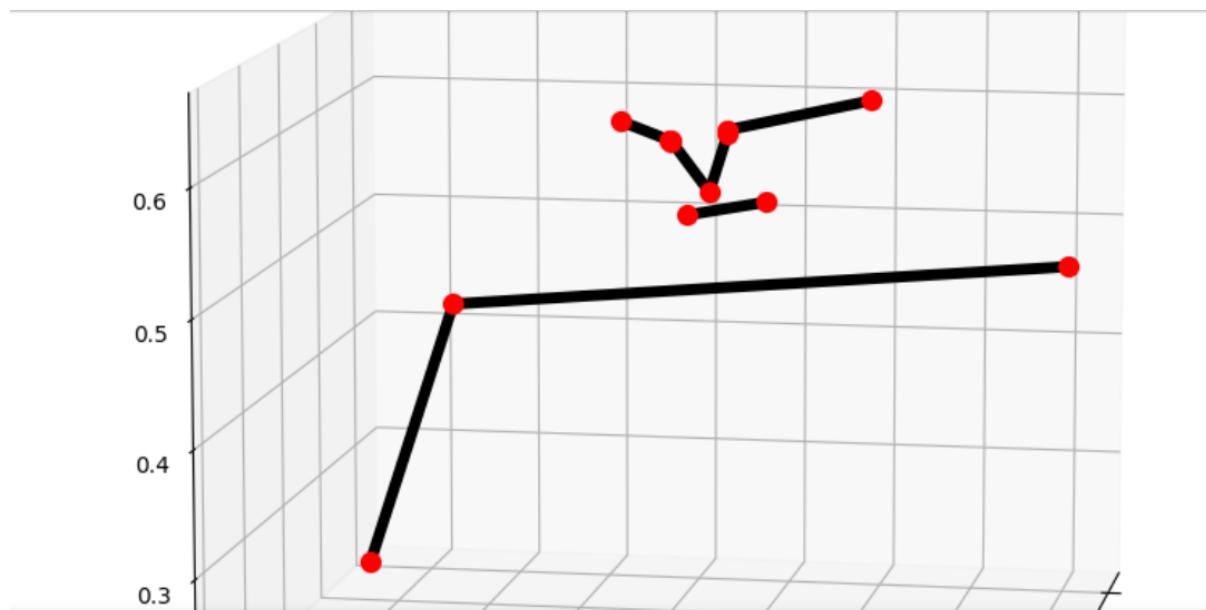
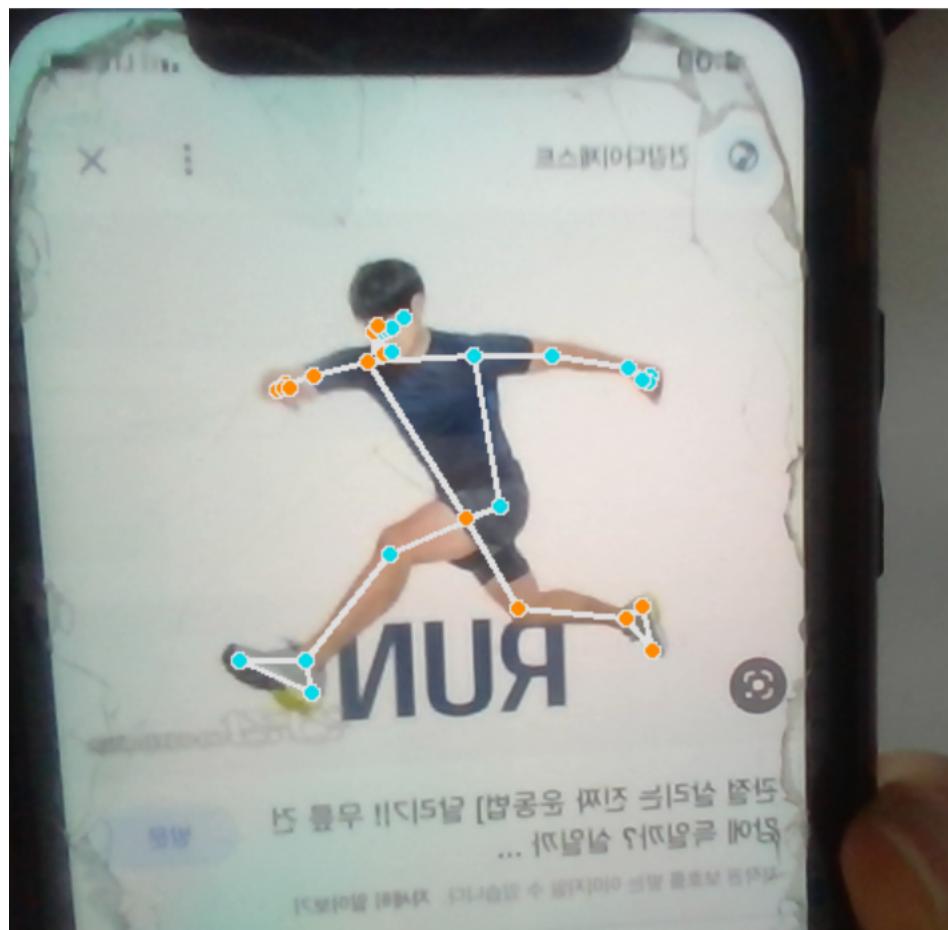
while True:
    ret,frame=cap.read()
    if not ret:
        print('프레임 획득에 실패하여 루프를 나갑니다.')
        break

    res=pose.process(cv.cvtColor(frame,cv.COLOR_BGR2RGB))

    mp_drawing.draw_landmarks(frame,res.pose_landmarks,mp_pose.POSE_CONNECTIONS,landmark_drawing_spec=mp_styles.get_default_pose_landmark_drawing_spec())
    cv.imshow('MediaPipe pose',cv.flip(frame,1)) # 좌우반전
    if cv.waitKey(5)==ord('q'):
        mp_drawing.plot_landmarks(res.pose_world_landmarks,mp_pose.POSE_CONNECTIONS)
        break

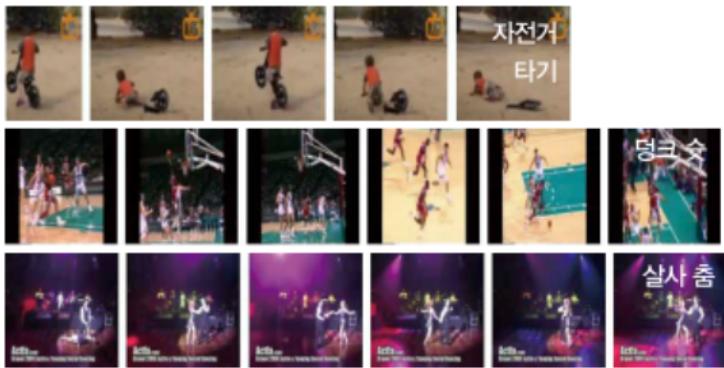
cap.release()
cv.destroyAllWindows()
```

 MediaPipe pose



10.4.3 행동 분류

현재 연구는 행동 분류까지,, 행동 이해는 ㄴㄴ



(a) Kinetics 데이터셋



(b) HAA500 데이터셋

그림 10-20 행동 분류를 위한 데이터셋

비디오 분류의 여러가지 방법

- 비디오는 3차원 공간 → 3차원 컨볼루션 수행

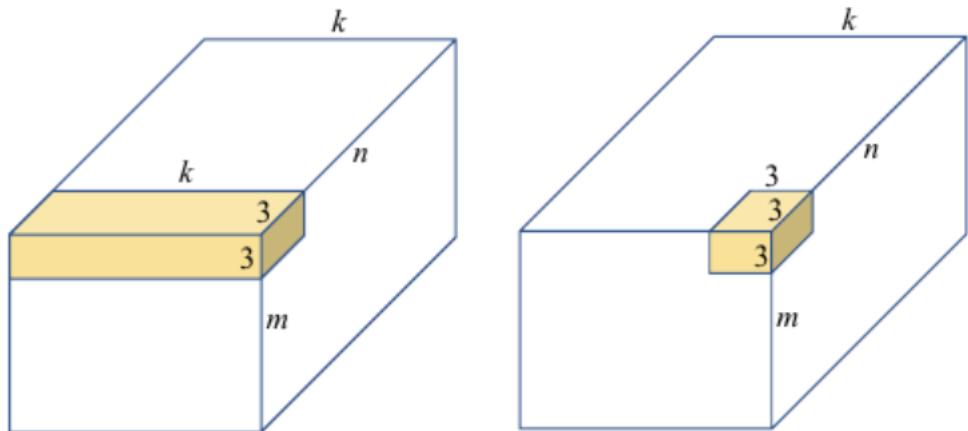


그림 10-21 2D 컨볼루션(왼쪽)과 3D 컨볼루션(오른쪽)

3x3x3 커널이 세 방향으로 이동

- 2Dconv 사용해 프레임별로 특징 추출하고 시계열 데이터 처리하는 LSTM 모델로 특징을 결합[Donahue2015]
- 비디오: 한 장의 프레임에서 추출한 특징 맵 / 인접한 여러 프레임에서 추출한 광류
위 두 흐름으로 부류 확률 벡터를 조정하고 결합해 최종 부류 결정하는 딥러닝 모델[Simonyan2014]
- 두 흐름은 마지막 컨볼루션층에서 결합해야 좋다[Feichtenhofer2016]
- 비디오를 몇 개 구간으로 나누고 각 구간에서 두 흐름의 컨볼루션을 수행→각 구간에서 얻은 특징 맵을 최대 출력 같은 단순한 연산으로 결합[Wang2019a]