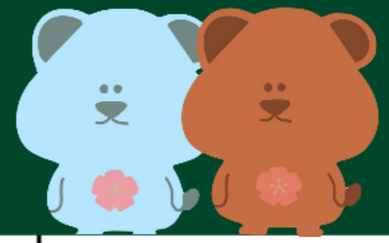
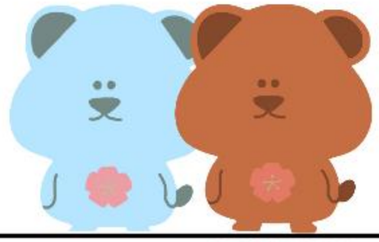


Week12: Deepfake Detection Challenge

발표자: 이지호, 김희숙

EWHA EURON

목차



EWHA EURON

대회 소개

EWHA EURON

#01 대회 소개 Deepfake Detection Challenge

대회 개요

- 콘텐츠 생성 및 수정 기술, 특히 딥페이크는 잘못된 정보, 조작, 괴롭힘 및 설득의 원천이 되어 악의적으로 사용될 수 있다. 조작된 미디어를 식별하는 작업의 기술적 요구가 크다.
- AWS, Facebook, Microsoft, the Partnership on AI's Media Integrity Steering Committee, and academics have come together to build the Deepfake Detection Challenge (DFDC).
- Task: 주어진 동영상이 딥페이크로 만들어진 동영상인지 아닌지를 예측한다.

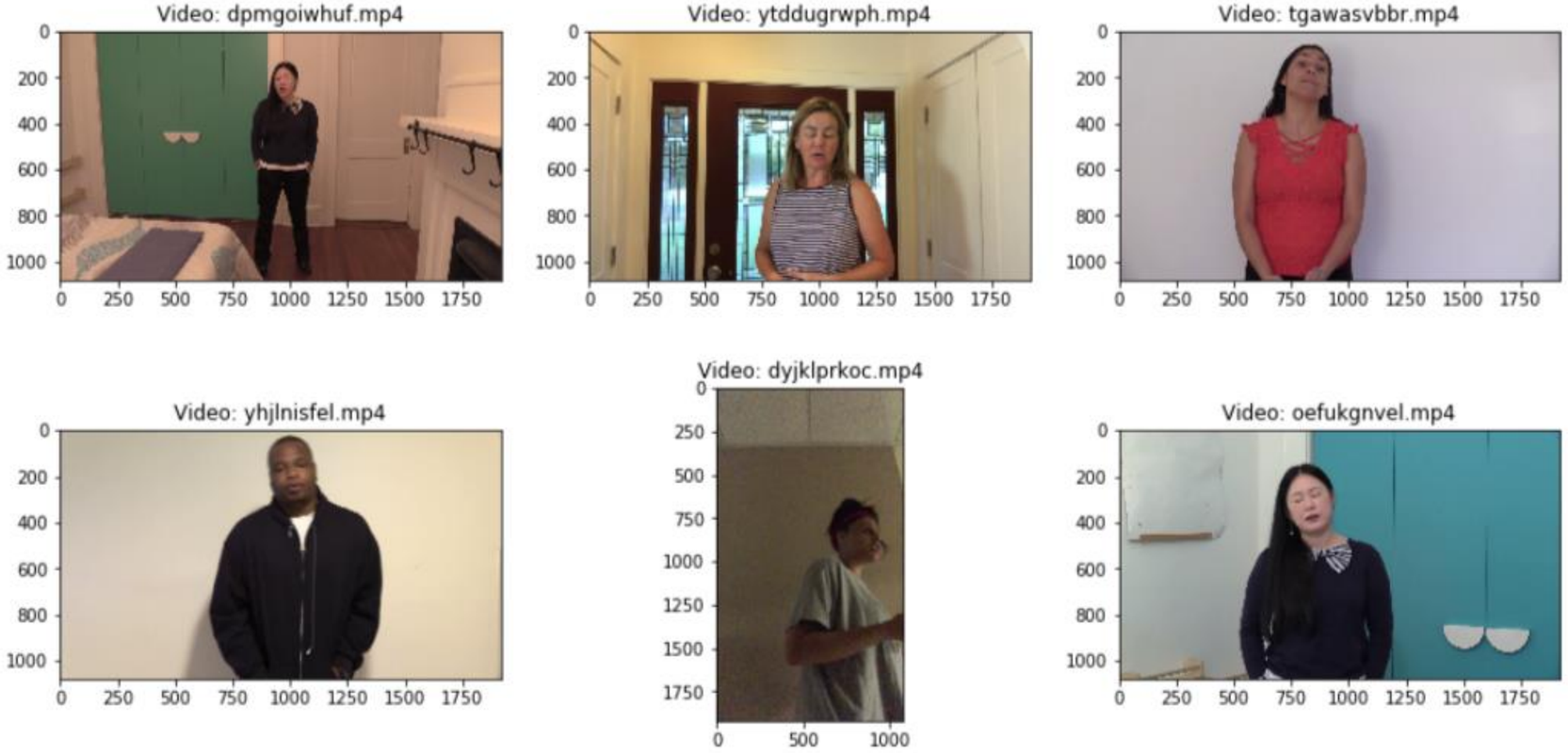
데이터 셋

- Train과 test 데이터셋으로 mp4영상 파일이 주어짐
  - Train_samle_videos.zip: 학습 영상과 메타 데이터
  - Test_test.zip
- (submission) Columns:
  - (1) video data file name
  - (2) label:비디오가 진짜인지 가짜진지 여부
  - (3) original: 학습 데이터가 fake인 경우, 실제 원본 데이터 목록

	label	split	original
aagfhgtpmv.mp4	FAKE	train	vudstovrck.mp4
aapnvogymq.mp4	FAKE	train	jdubbvfswz.mp4
abarnvbtwb.mp4	REAL	train	None
abofeumbvv.mp4	FAKE	train	atvmxvwyns.mp4
abqwwspghj.mp4	FAKE	train	qzimuostzz.mp4

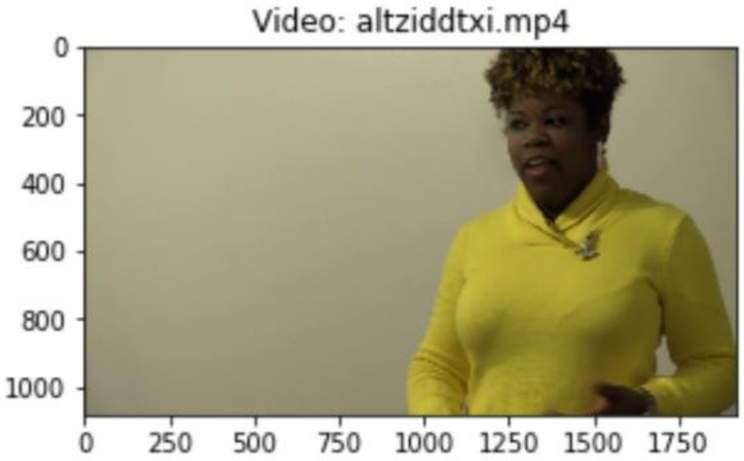
#01 대회 소개 Deepfake Detection Challenge

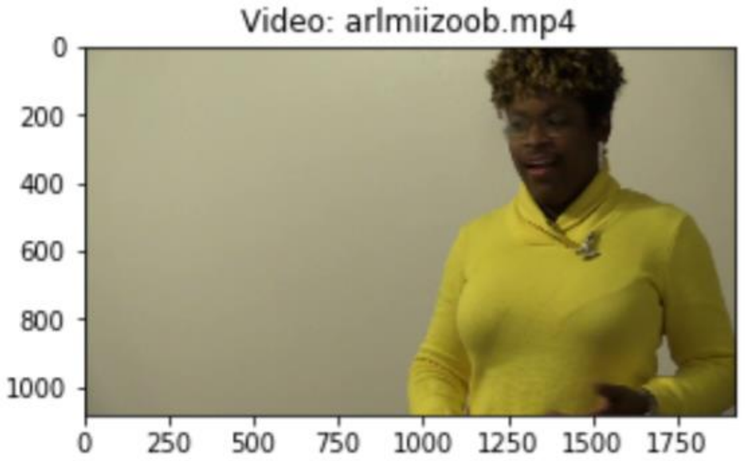
Data EDA

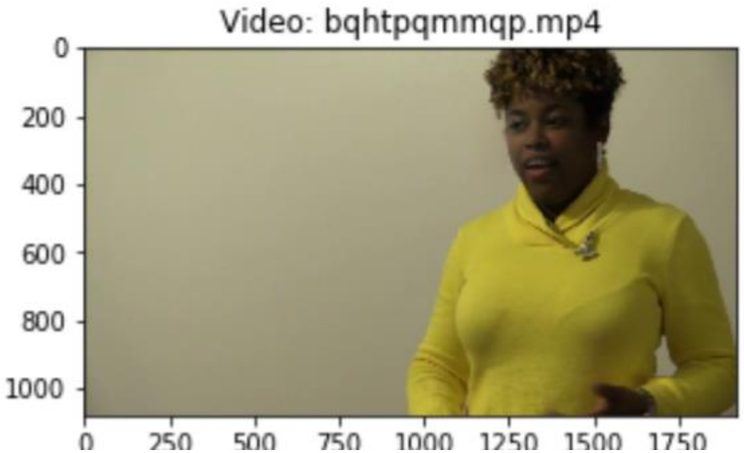


#01 대회 소개 Deepfake Detection Challenge

Data EDA







#01 대회 소개 Deepfake Detection Challenge

1등 전략

- Frame-by-frame Classification Approach
- Data Preperation
  - extracted boxes and landmarks with MTCNN and saved them as json
  - extracted crops in original size and saved them as png
  - extracted SSIM masks with difference between real and fake and saved them as png
- Face-Detector

```
detector = MTCNN(margin=0, thresholds=[0.65, 0.75, 0.75], device="cpu")
```

I used simple MTCNN detector.
Input size for face detector was caluclated for each video depending on video resolution.

  - 2x resize for videos with less than 300 pixels wider side
  - no resize for videos with wider side between 300 and 1000
  - 0.5x resize for videos with wider side > 1000 pixels
  - 0.33x resize for videos with wider side > 1900 pixels

#01 대회 소개 Deepfake Detection Challenge

```python
def save_landmarks(ori_id, root_dir):
    ori_id = ori_id[:-4]
    ori_dir = os.path.join(root_dir, "crops", ori_id)
    landmark_dir = os.path.join(root_dir, "landmarks", ori_id)
    os.makedirs(landmark_dir, exist_ok=True)
    for frame in range(320):
        if frame % 10 != 0:
            continue
        for actor in range(2):
            image_id = "{}_{}.png".format(frame, actor)
            landmarks_id = "{}_{}".format(frame, actor)
            ori_path = os.path.join(ori_dir, image_id)
            landmark_path = os.path.join(landmark_dir, landmarks_id)

            if os.path.exists(ori_path):
                try:
                    image_ori = cv2.imread(ori_path, cv2.IMREAD_COLOR)[...,::-1]
                    frame_img = Image.fromarray(image_ori)
                    batch_boxes, conf, landmarks = detector.detect(frame_img, landmarks=True
                    if landmarks is not None:
                        landmarks = np.around(landmarks[0]).astype(np.int16)
                        np.save(landmark_path, landmarks)
                except Exception as e:
                    print(e)
                    pass
```

Data preparation→ landmark detector 이용(generate_landmarks.py)

```python
def save_diffs(pair, root_dir):
    ori_id, fake_id = pair
    ori_dir = os.path.join(root_dir, "crops", ori_id)
    fake_dir = os.path.join(root_dir, "crops", fake_id)
    diff_dir = os.path.join(root_dir, "diffs", fake_id)
    os.makedirs(diff_dir, exist_ok=True)
    for frame in range(320):
        if frame % 10 != 0:
            continue
        for actor in range(2):
            image_id = "{}_{}.png".format(frame, actor)
            diff_image_id = "{}_{}_diff.png".format(frame, actor)
            ori_path = os.path.join(ori_dir, image_id)
            fake_path = os.path.join(fake_dir, image_id)
            diff_path = os.path.join(diff_dir, diff_image_id)
            if os.path.exists(ori_path) and os.path.exists(fake_path):
                img1 = cv2.imread(ori_path, cv2.IMREAD_COLOR)
                img2 = cv2.imread(fake_path, cv2.IMREAD_COLOR)
                try:
                    d, a = compare_ssim(img1, img2, multichannel=True, full=True)
                    a = 1 - a
                    diff = (a * 255).astype(np.uint8)
                    diff = cv2.cvtColor(diff, cv2.COLOR_BGR2GRAY)
                    cv2.imwrite(diff_path, diff)
                except:
                    pass
```

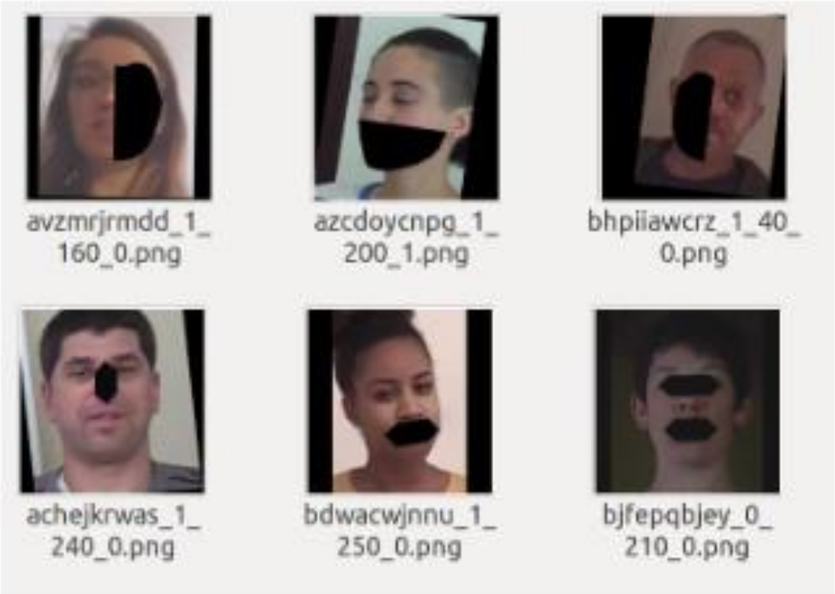
Original/fake의 차이 계산, SIMM detector 이용(generate_diffs.py)

EWHA EURON

#01 대회 소개 Deepfake Detection Challenge

1등 전략

- Frame-by-frame Classification Approach
- Data Preparation – Augmentation(with Albumentation library)
  - ➢ Resizing 외에 별다른 전처리는 없었지만 augmentation을 많이 했음
  - ➢ Generalization을 위해서 GridMask를 이용해 이미지 일부를 drop out → alignment를 사용하지 않는 facenet을 robust 하게 만드는데 역할



FaceNet & MTCNN

EWHA EURON

#02 FaceNet

Face Detection

➢Face detection Task (object detection + α)

- Region proposal
- Face or background classification → face classification, Bbox regression, face landmark localization **: Rough estimation**
- Face Landmark detect → face classification, Bbox regression, face landmark localization **: Refine**

→ **MTCNN Detector**

➢FaceNet

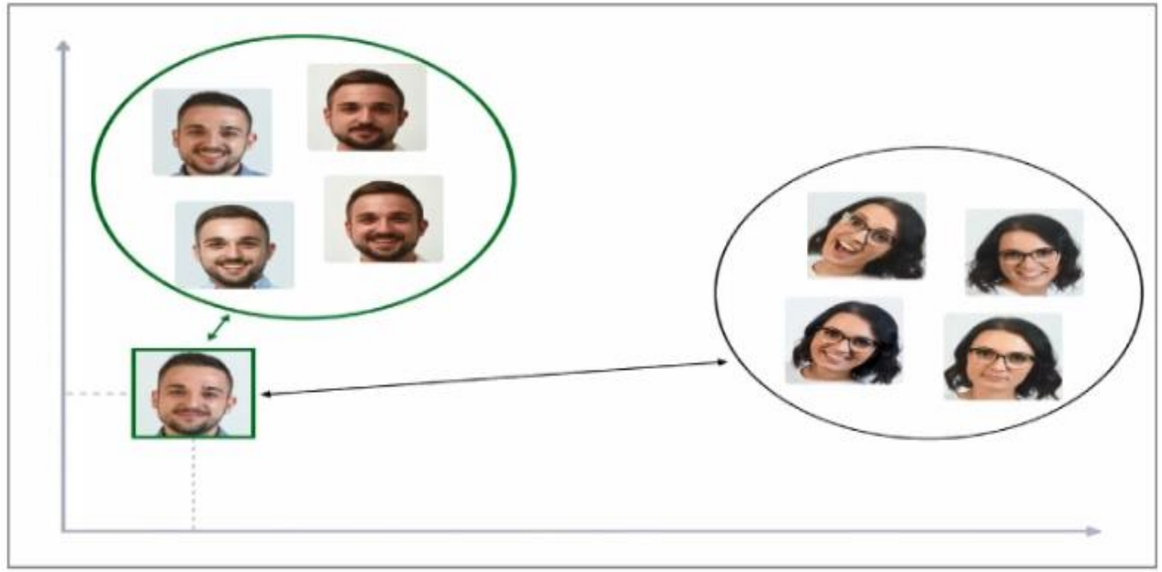
- 얼굴 사진에서 특정 사람에 대한 특징 값을 추출하는 모델.
- 모델의 결과 feature값을 활용하여 image identification, verification, clustering을 수행.
- Face detection task에서 breakthrough가 된 **Triplet loss**를 처음 제안한 논문.
- Contributions: Deep learning based approach, SOTA

[A unified embedding for face recognition, *f schoroff et al, 2015*]

#02 FaceNet

FaceNet

➢ Triplet loss로 얼굴 feature의 유사도를 구하자



<model structure>

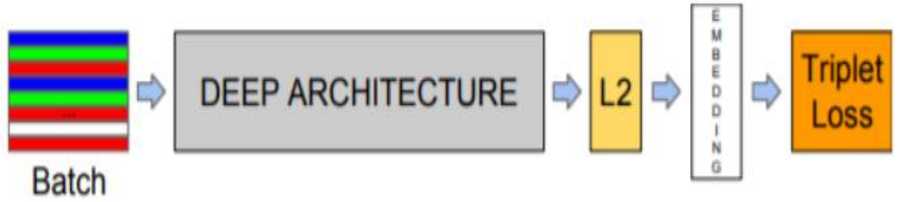


Figure 2. **Model structure.** Our network consists of a batch input layer and a deep CNN followed by L_2 normalization, which results in the face embedding. This is followed by the triplet loss during training.

➢ Triplet Loss
- 이미지의 유사도를 계산
- Inputs – 임베딩 테이블을 통과한 결과
  ① Anchor embedding: 인풋 이미지에 해당하는 특정 사람의 얼굴 사진을 나타내는 벡터 값
  ② Positive embedding: Anchor와 동일한 인물의 얼굴 사진임을 나타내는 벡터 값
  ③ Negative embedding: Anchor와 다른 인물의 얼굴 사진임을 나타내는 벡터 값

[A unified embedding for face recognition, *f schoroff et al, 2015*]

#02 FaceNet

FaceNet

➢ Triplet Loss

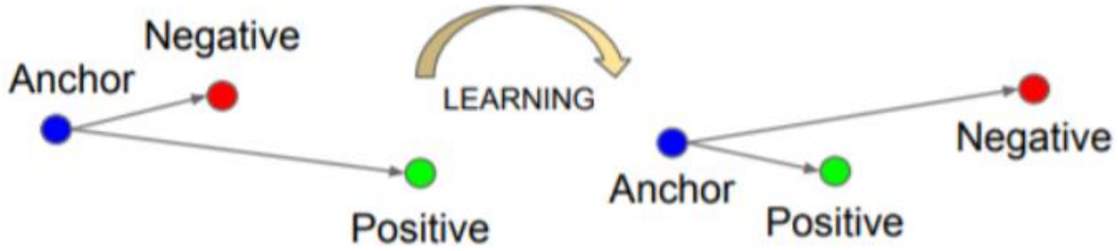


Figure 3. The **Triplet Loss** minimizes the distance between an *anchor* and a *positive*, both of which have the same identity, and maximizes the distance between the *anchor* and a *negative* of a different identity.

The loss that is being minimized is then $L =$

$$\sum_{i}^{N} \left[\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha \right]_+ . \quad (3)$$

- L2 distance loss
- Green box: anchor와 positive사이의 거리
- Red box: anchor와 negative사이의 거리, - term
- Loss를 최적화(minimizing)하는 것은 positive embeddin과의 거리는 가깝게, negative embedding과의 거리는 멀게 학습하는 것.

[A unified embedding for face recognition, *f schoroff et al, 2015*]

#02 FaceNet

FaceNet

➢ Network & Results

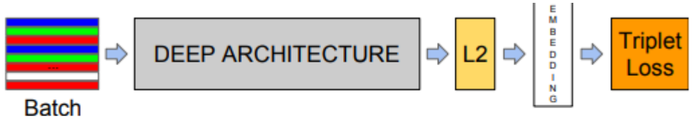


Figure 2. **Model structure.** Our network consists of a batch input layer and a deep CNN followed by L_2 normalization, which results in the face embedding. This is followed by the triplet loss during training.

layer	size-in	size-out	kernel	param	FLPS
conv1	220×220×3	110×110×64	7×7×3, 2	9K	115M
pool1	110×110×64	55×55×64	3×3×64, 2	0	
rnorm1	55×55×64	55×55×64		0	
conv2a	55×55×64	55×55×64	1×1×64, 1	4K	13M
conv2	55×55×64	55×55×192	3×3×64, 1	111K	335M
rnorm2	55×55×192	55×55×192		0	
pool2	55×55×192	28×28×192	3×3×192, 2	0	
conv3a	28×28×192	28×28×192	1×1×192, 1	37K	29M
conv3	28×28×192	28×28×384	3×3×192, 1	664K	521M
pool3	28×28×384	14×14×384	3×3×384, 2	0	
conv4a	14×14×384	14×14×384	1×1×384, 1	148K	29M
conv4	14×14×384	14×14×256	3×3×384, 1	885K	173M
conv5a	14×14×256	14×14×256	1×1×256, 1	66K	13M
conv5	14×14×256	14×14×256	3×3×256, 1	590K	116M
conv6a	14×14×256	14×14×256	1×1×256, 1	66K	13M
conv6	14×14×256	14×14×256	3×3×256, 1	590K	116M
pool4	14×14×256	7×7×256	3×3×256, 2	0	
concat	7×7×256	7×7×256		0	
fc1	7×7×256	1×32×128	maxout p=2	103M	103M
fc2	1×32×128	1×32×128	maxout p=2	34M	34M
fc7128	1×32×128	1×1×128		524K	0.5M
L2	1×1×128	1×1×128		0	
total				140M	1.6B

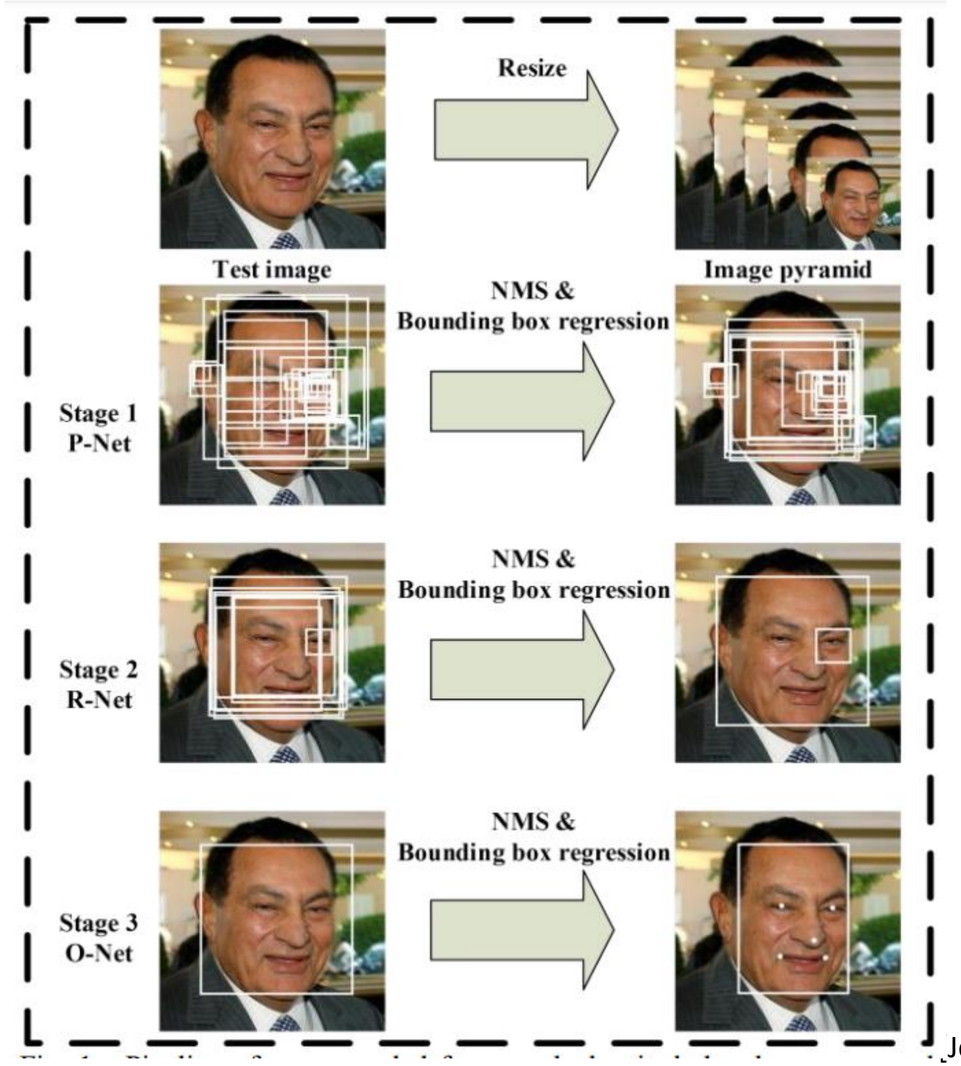
98.87%±0.15 accuracy

[A unified embedding for face recognition, *f schoroff et al, 2015*]

#02 MTCNN

MTCNN

- Multi-task: joint learning of face detection, landmark detection, Bbox regression



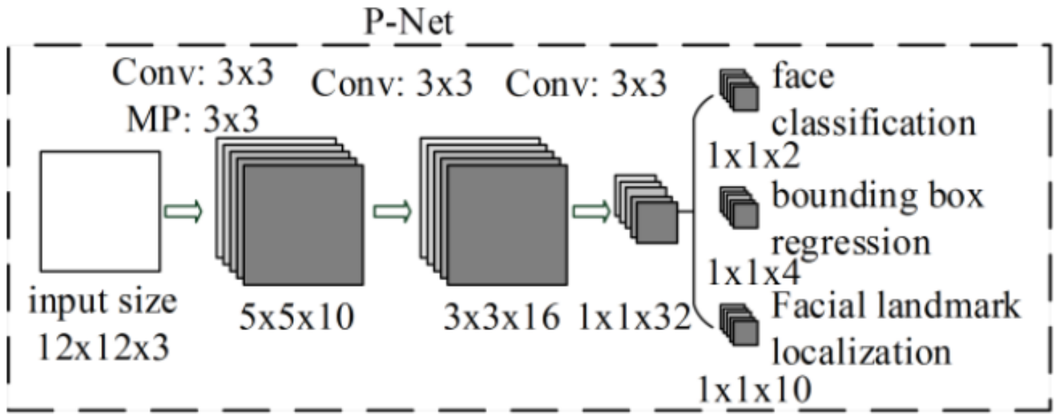
- Stage 0. Image pyramid
  - (object detection에서 multi size sliding window와 비슷한 역할). 다양한 사이즈의 얼굴을 더 잘 detection하기 위해서 인풋 이미지를 각기 다른 스케일로 resizing한다.
- Stage 1. P-Net(Proposal Network)
  - 이미지에서 얼굴 영역을 찾아낸다.
- Stage 2. R-Net(Refine Network)
  - P-Net에서 찾아낸 region proposal을 추리는데 중점을 둔다.
- Stage 3. O-Net(Output Network)
  - P-Net에서 찾아낸 region proposal에서 face landmark를 찾아내는데 중점을 둔다.

[1] Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks, *kaipeng et al, 2016*

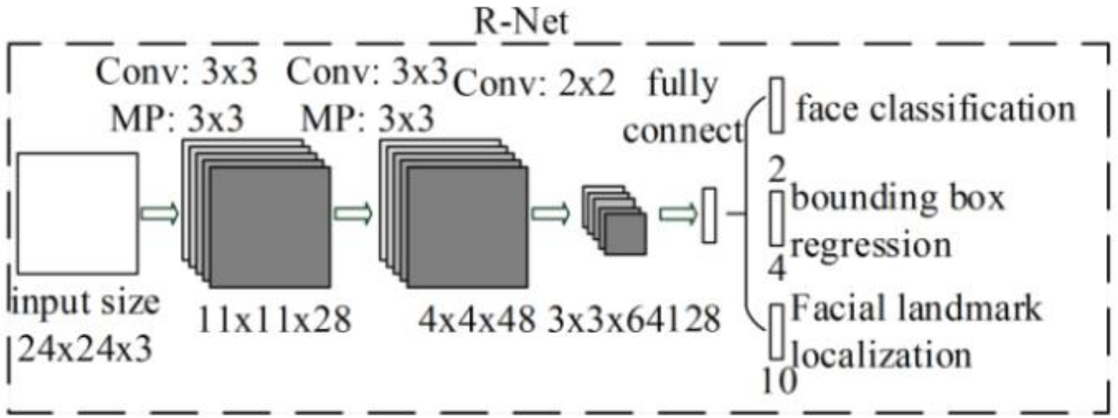
#02 MTCNN

MTCNN

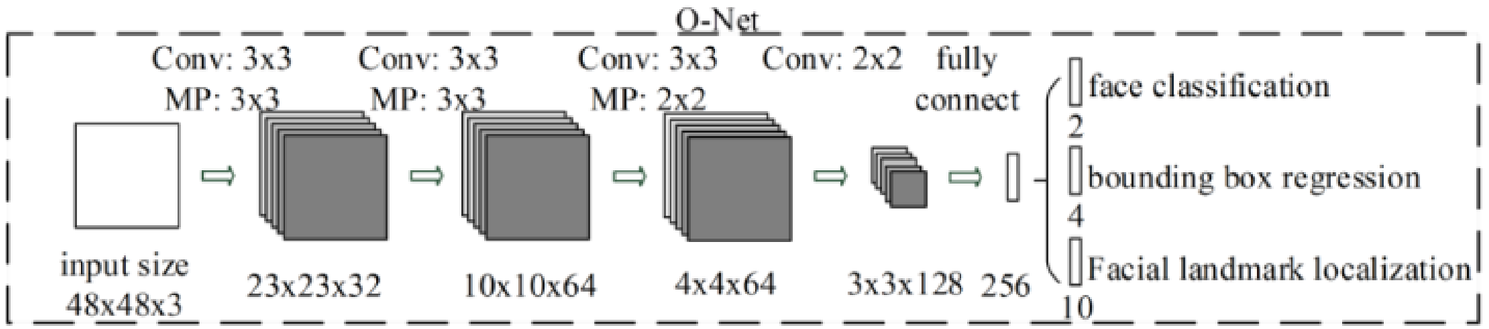
1단계. P-Net(Proposal Network)



2단계. R-Net(Refine Network)



3단계. O-Net(Output Network)

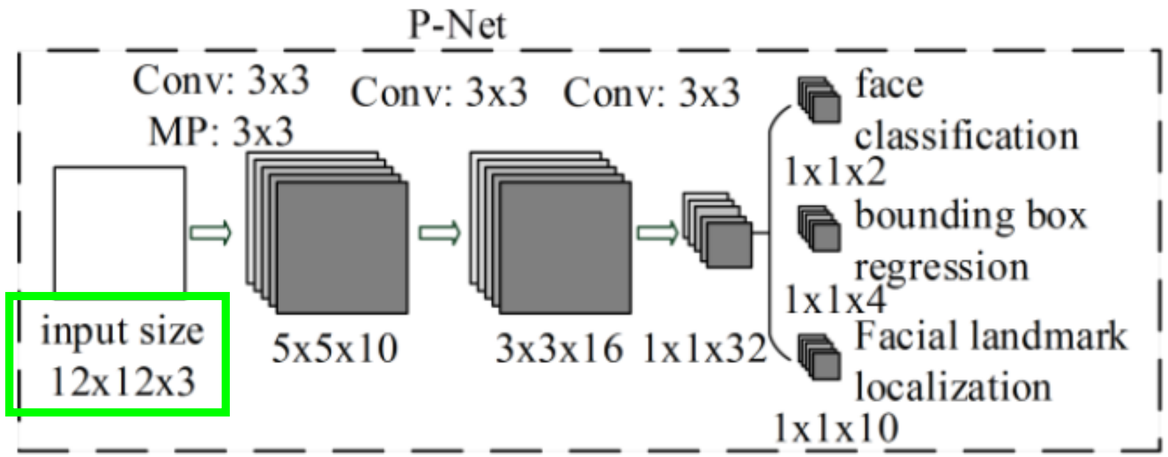


[Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks, *kaipeng et al, 2016*]

#02 MTCNN

MTCNN

1단계. P-Net(Proposal Network)



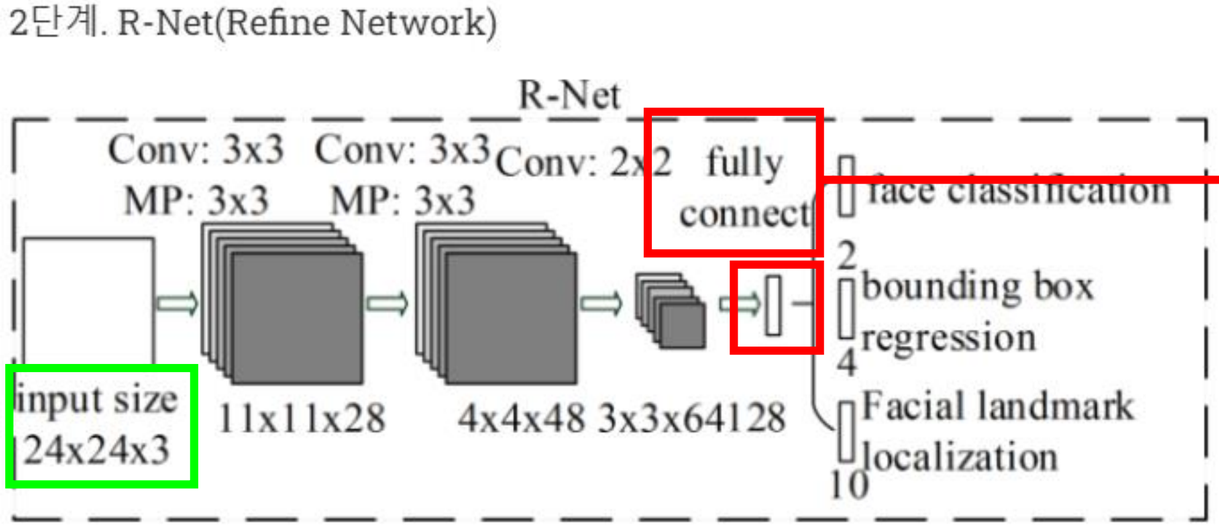
- face classification (2개)
  y^{det} = GT에서 얼굴이 있는지 여부(있을때 1, 없을때 0)
  p = 얼굴이 있을 확률
- bbox regression (4개)
  예측한 bbox의 왼쪽상단 x,y좌표
  예측한 bbox의 너비와 높이
- face landmark localization (10개)
  왼쪽 눈의 x,y 좌표
  오른쪽 눈의 x,y 좌표
  코의 x,y 좌표
  입의 왼쪽 끝 부분의 x,y 좌표
  입의 오른쪽 끝 부분의 x,y 좌표

- **P-Net**
  - 이미지에서 얼굴을 찾아내는데 중점을 둔 network
  - Convolution layer로만 이루어짐
  - input size: 12x12
  - NMS(non-maximum suppression)알고리즘으로 높은 정확도의 후보 영역만 추려내고, 남은 후보 영역의 face classification, bounding box regression, face landmark localization 값을 2단계로 보냄

[Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks, *kaipeng et al, 2016*]

#02 MTCNN

MTCNN



더 정확한 값 추려 낼 수 있음

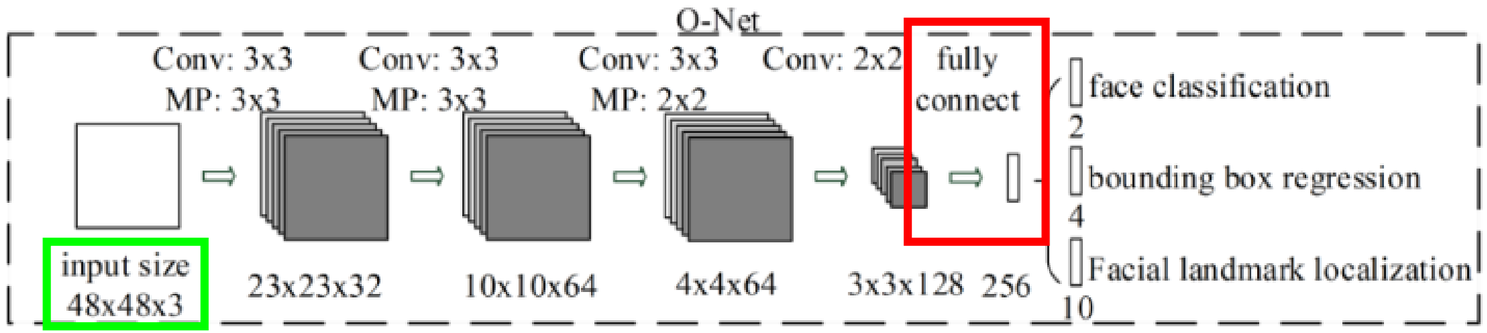
- **R-Net**
  - P-Net에서 찾아낸 후보 영역을 추려내는데 중점을 둔 network
  - P-Net의 구조와 유사하나, input size 24x24로 resize
  - P-Net에서 얻은 결과를 사용
  - Convolution layer 와 끝에 fully connected layer 로 이루어짐
  - NMS(non-maximum suppression)알고리즘으로 높은 정확도의 후보 영역만 추려내고, 남은 후보 영역의 face classification, bounding box regression, face landmark localization 값을 3단계로 보냄

[Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks, *kaipeng et al, 2016*]

#02 MTCNN

MTCNN

3단계. O-Net(Output Network)



- **O-Net**
  - R-Net에서 찾아낸 후보 영역에서 face landmark를 찾아내는데 중점을 둔 network
  - input size 48x48로 resize
  - P-Net과 R-Net에서 얻은 결과를 모두 사용
  - 최종적인 face classification, bbox regression, face landmark localization값 추출

[Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks, *kaipeng et al, 2016*]

#02 MTCNN

MTCNN

➢Multi Task Loss

- classification, regression, localization 세 가지 테스크에 대하여 각각 loss를 구한 뒤, 이를 가중치를 두어 합치는 방식

$$L_i^{det} = -(y_i^{det} \log(p_i) + (1 - y_i^{det})(1 - \log(p_i)))$$

Classification Loss

$$L_i^{box} = \left\| \hat{y}_i^{box} - y_i^{box} \right\|_2^2$$

Bounding Box Regression Loss

$$L_i^{landmark} = \left\| \hat{y}_i^{landmark} - y_i^{landmark} \right\|_2^2$$

localization Loss

$$\min \sum_{i=1}^{N} \sum_{j \in \{det, box, landmark\}} \alpha_j \beta_i^j L_i^j$$

P−Net: (α_{det}: 1, α_{box}: 0.5, $\alpha_{landmark}$: 0.5)

R−Net: (α_{det}: 1, α_{box}: 0.5, $\alpha_{landmark}$: 0.5)

O−Net: (α_{det}: 1, α_{box}: 0.5, $\alpha_{landmark}$: 1)

최종 Loss

[Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks, *kaipeng et al, 2016*]

#Summary

Remark

Input size

As soon as I discovered that EfficientNets significantly outperform other encoders I used only them in my solution. As I started with B4 I decided to use "native" size for that network (380x380). Due to memory costraints I did not increase input size even for B7 encoder.

Encoders

The winning encoder is current state-of-the-art model (EfficientNet B7) pretrained with ImageNet and noisy student Self-training with Noisy Student improves ImageNet classification

I tried multiple variants of efficient nets at the beginning:

- solo B3 (300×300) - 0.29 public
- solo B4 (380×380) - 0.27 public
- solo B5 (380×380) - 0.25 public
- solo B6 (380×380) - 0.27 public (surprisingly it was worse than B5 and I have not tried B7 until competition last week)
- solo B7 (380×380) - 0.24 public
  In the end I used two submits with:
- 15xB5 (different seeds) with heursitic overfitted for Public LB which was trained with standard augmentations - 10th place on private
- 7xB7 (different seeds) with more conservative avergaing heursitic and trained with hardcore augmentations - 3rd place on private

1. Find face bboxes

To extract face bboxes I used facenet library, basically only MTCNN. `python preprocessing/detect_original_faces.py --root-dir DATA_ROOT` This script will detect faces in real videos and store them as jsons in DATA_ROOT/bboxes directory

2. Extract crops from videos

To extract image crops I used bboxes saved before. It will use bounding boxes from original videos for face videos as well. `python preprocessing/extract_crops.py --root-dir DATA_ROOT --crops-dir crops` This script will extract face crops from videos and save them in DATA_ROOT/crops directory

3. Generate landmarks

From the saved crops it is quite fast to process crops with MTCNN and extract landmarks
`python preprocessing/generate_landmarks.py --root-dir DATA_ROOT` This script will extract landmarks and save them in DATA_ROOT/landmarks directory

4. Generate diff SSIM masks

`python preprocessing/generate_diffs.py --root-dir DATA_ROOT` This script will extract SSIM difference masks between real and fake images and save them in DATA_ROOT/diffs directory

5. Generate folds

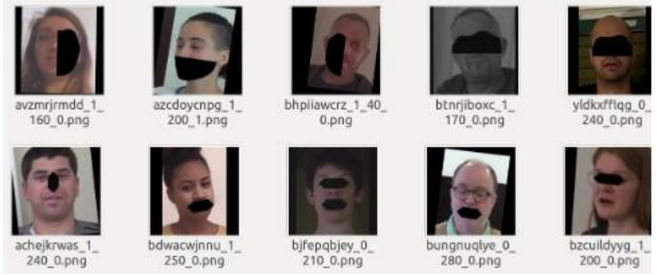
`python preprocessing/generate_folds.py --root-dir DATA_ROOT --out folds.csv`
By default it will use 16 splits to have 0-2 folders as a holdout set. Though only 400 videos can be used for validation as well.

Generalization approach

I expected it won't be enough for generalization and spent a few weeks working on domain specific augmentations.
I wanted to push models to learn the following properties:

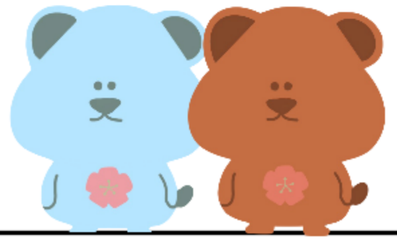
- visual artifacts (models learn that easily even without any augmentations)
- different encoding of face from other part of the image. Big margin helps with that.
- face warping artifacts. Big marging helps with that as well. Related article https://arxiv.org/abs/1811.00656 to catch FWA
- blending artifacts - here we need either to predict blending mask (https://arxiv.org/abs/1912.13458) but it's not possible to obtain ground truth mask as there is no big difference in pixels/SSIM on the edge due to blurring and other techniques used to reduce blending artifacts or come up with augmentations that **destroy** visual artifacts.
  To catch face blending artefacts:

1. removed half face horisontally or vertically. Used dlib face convex hulls.
2. blacked out landmarks (eyes, nose or mouth). Used MTCNN landmarks for that.
3. blacked out half of the image. To be safe I checked that it will not delete highly confident difference from masks generated with SSIM.



https://github.com/selimsef/dfdc_deepfake_challenge
https://www.kaggle.com/competitions/deepfake-detection-challenge/discussion/145721

THANK YOU



EWHA EURON