TrustNet

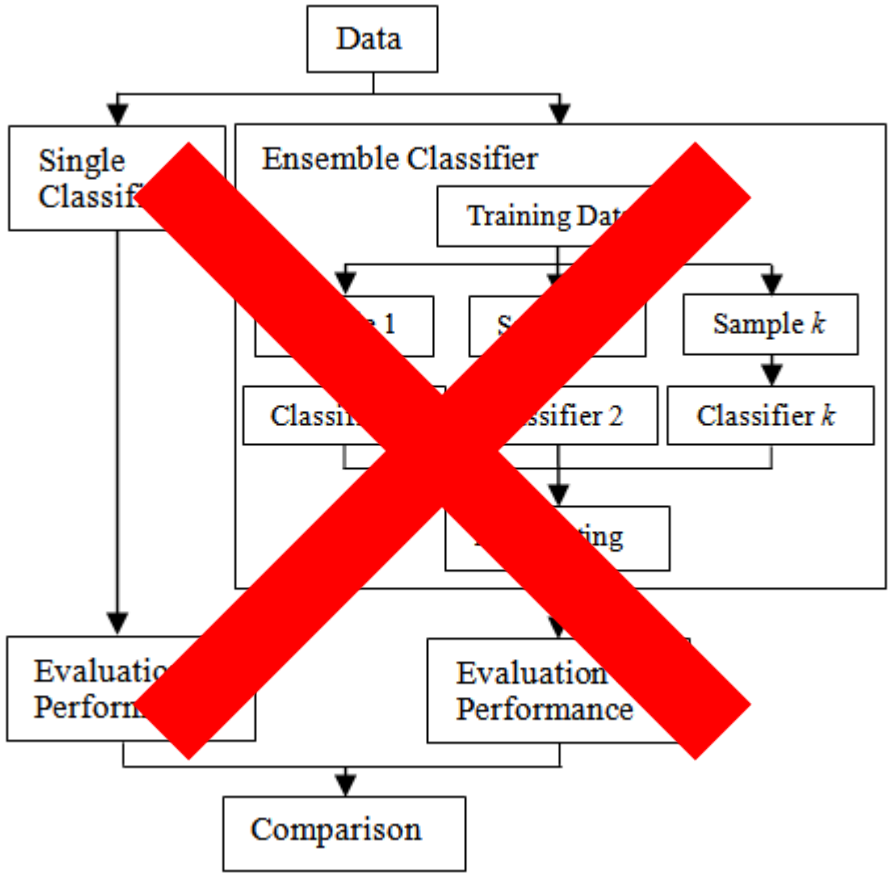
DeepFake Detector

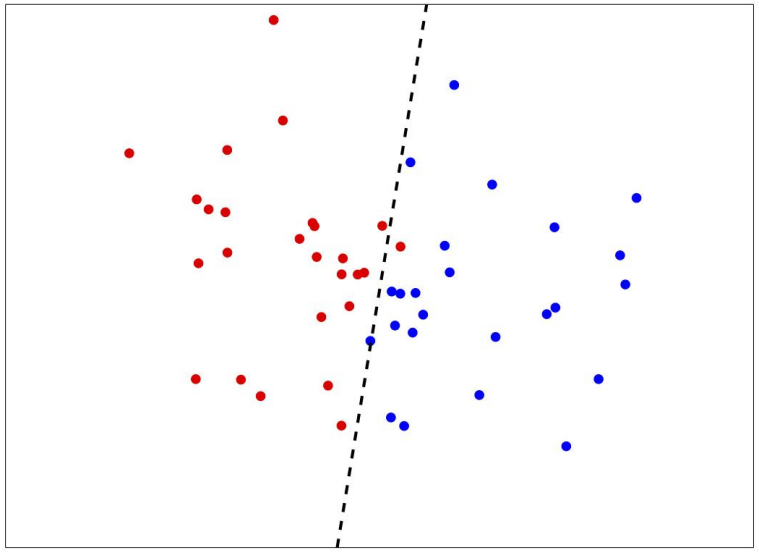
# DFDC- 1st Place Solution



SW_Maestro_

# Simple is the Best!

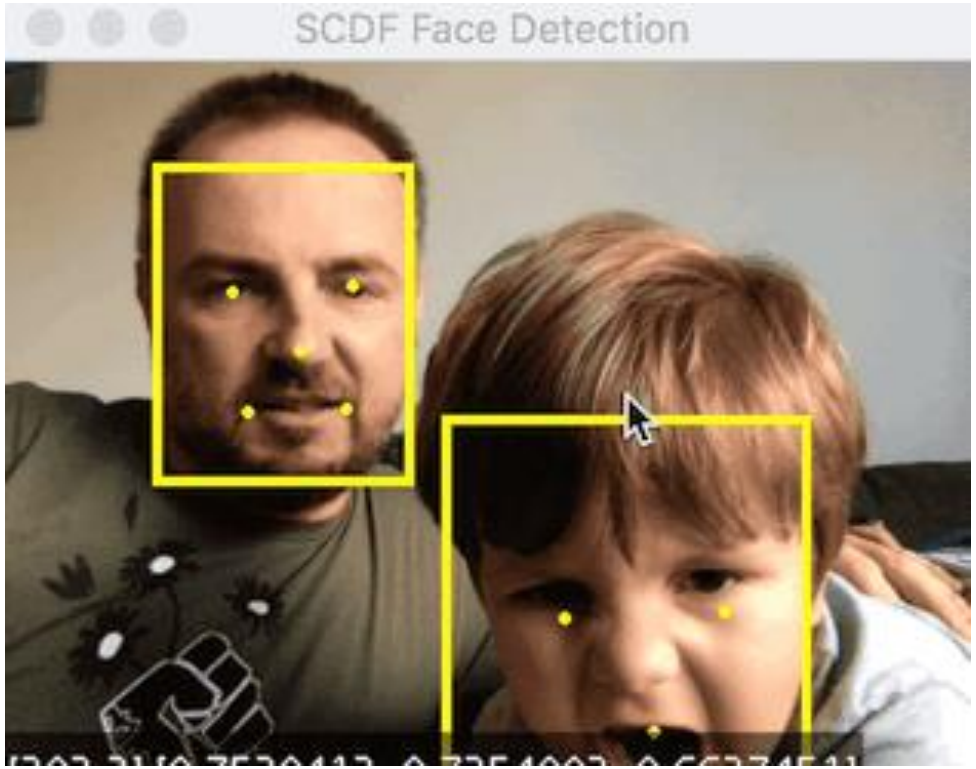


다수의 Classifier 결합한 복잡한 모델 < Single Classifier들의 조화

# 1. Data Preparation

# MT-CNN으로 얼굴 인식

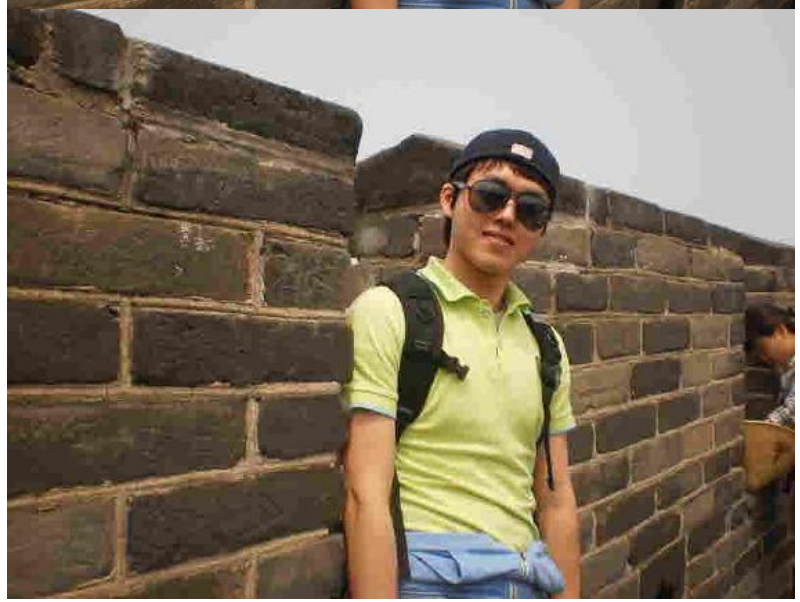


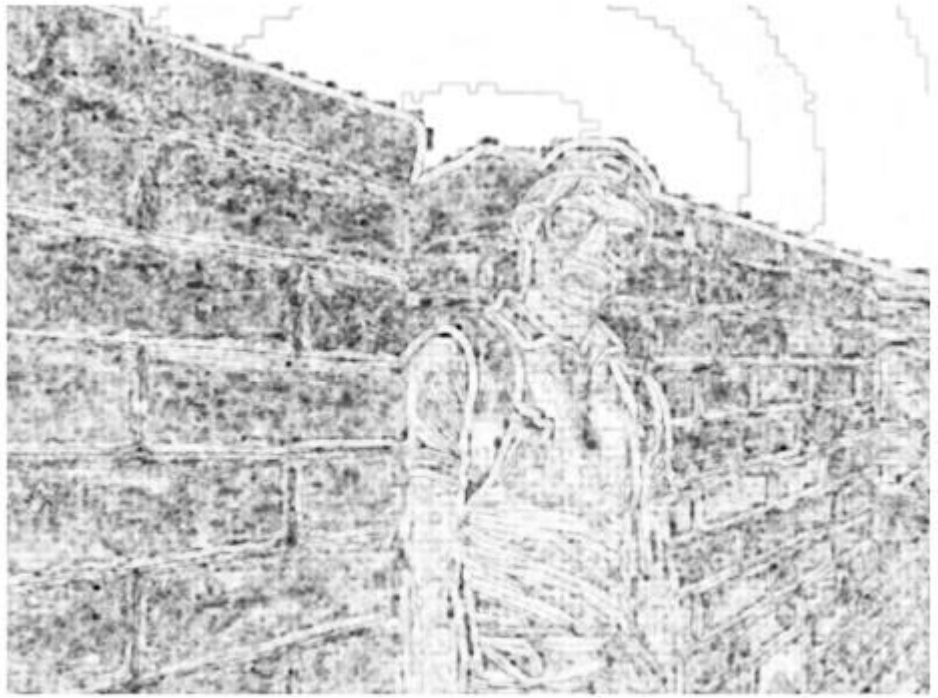

MT-CNN을 통해 얼굴 부분의 Bound Box와 눈 코 입 등의 Face Landmark를 Json 형식으로 저장한다.

# Cropping 과정 진행


원본 이미지에서, DeepFake 검출에 필요한 부분만 Cropping 작업을 진행하고, 이를 png파일로 저장한다. 이 때 Crop은 얼굴 BBOX의 20% 정도의 Margin을 두고 저장한다.

# SSIM Mask 도출



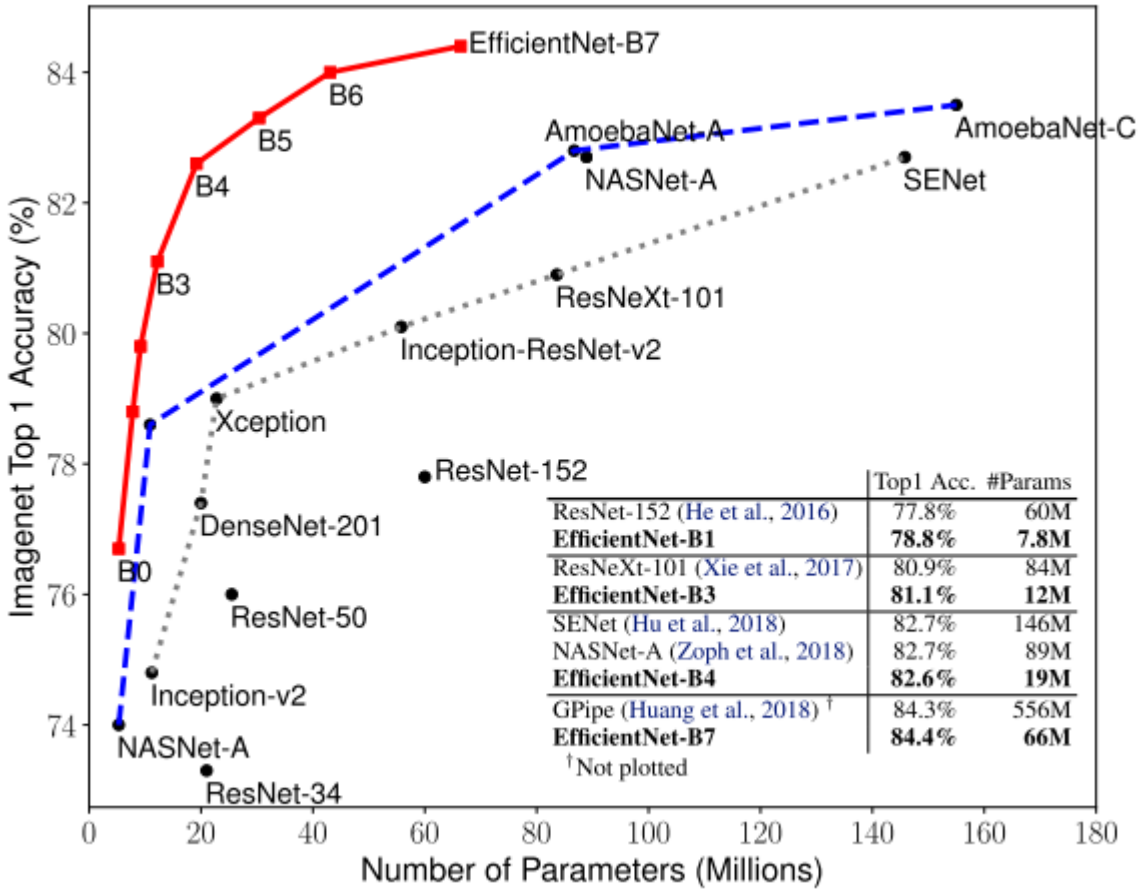


# 2. Face Detector

# MT-CNN Resizing

- 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

# Encoder : EfficientNets




| | Top1 Acc. | #Params |
|---|---|---|
| ResNet-152 (He et al., 2016) | 77.8% | 60M |
| **EfficientNet-B1** | **78.8%** | **7.8M** |
| ResNeXt-101 (Xie et al., 2017) | 80.9% | 84M |
| **EfficientNet-B3** | **81.1%** | **12M** |
| SENet (Hu et al., 2018) | 82.7% | 146M |
| NASNet-A (Zoph et al., 2018) | 82.7% | 89M |
| **EfficientNet-B4** | **82.6%** | **19M** |
| GPipe (Huang et al., 2018) † | 84.3% | 556M |
| **EfficientNet-B7** | **84.4%** | **66M** |

†Not plotted

대회 당시, EfficientNets이 SOTA 알고리즘이었고, 다른 Encoder들과 비교했을 때, 압도적인 성능차를 보여줬다고 함

Insight => 지금의 SOTA인 ResNest와 EfficientNet B7을 비교하면 뭐가 더 좋을까? 실험 필요

이미지 크기는 B4에서 사용하는 기본 사양인 380 x 380을 사용했다고 함.

메모리 문제로 인해 B7에서도 380 x 380을 유지했었다고 한다.

# 3. Test & Result

# Hyper-parameters

- sampled fake crops based on number of real crops i.e. `fakes.sample(n=num_real, replace=False, random_state=seed)`
- 2500 iterations per epoch
- SGD, momentum=0.9, weight decay=1e-4
- PolyLR with 0.01 starting LR
- 75k iterations
- used Apex with mixed precision
- trained on 4 GPUs with SyncBN and DDP. Batch size 16×4 for B5, 12×4 for B7.
- label smoothing - for me label smoothing with 0.01 eps was optimal on public leaderboard. Also it allowed not to use clipping at all.

# EfficientNet 성능 비교

Solo B3 (300x300) – Score : 0.29
Solo B4 (380x380) – Score : 0.27
Solo B5 (380x380) – Score : 0.25
Solo B6 (380x380) – Score : 0.27 (왜 B5보다 오히려 점수가 낮은지 이해가 되지 않았다고 함, 그래서 B7은 한참이 지난 후에야 테스트를 진행)
Solo B7 (380x380) – Score : 0.24

15xB5 (Different Seeds Ensemble) : 기본적인 Augmentation을 진행 -> 10위 진입
7x B7 (Different Seeds Ensemble) : Heuristic 하게 얻은 Averaging 수식을 활용하여 Ensemble 진행, 굉장히 하드한 Augmentation을 사용했다. (3위 달성) (1,2위 짤려서 지금은 1위인듯)

# Averaging Predictions

한 비디오당 32 프레임을 사용하였으며, 단순히 평균을 구하는 방식이 아니라, 여러 실험을 해 본 결과 아래 코드를 사용하는 것이 가장 좋은 정확도를 얻었다고 함(0.25 -> 0.22 Solo B5)

```
import numpy as np

def confident_strategy(pred, t=0.8):
    pred = np.array(pred)
    sz = len(pred)
    fakes = np.count_nonzero(pred > t)
    # 11 frames are detected as fakes with high probability
    if fakes > sz // 2.5 and fakes > 11:
        return np.mean(pred[pred > t])
    elif np.count_nonzero(pred < 0.2) > 0.9 * sz:
        return np.mean(pred[pred < 0.2])
    else:
        return np.mean(pred)
```

# 4. Augmentation

# Augmentation 종류

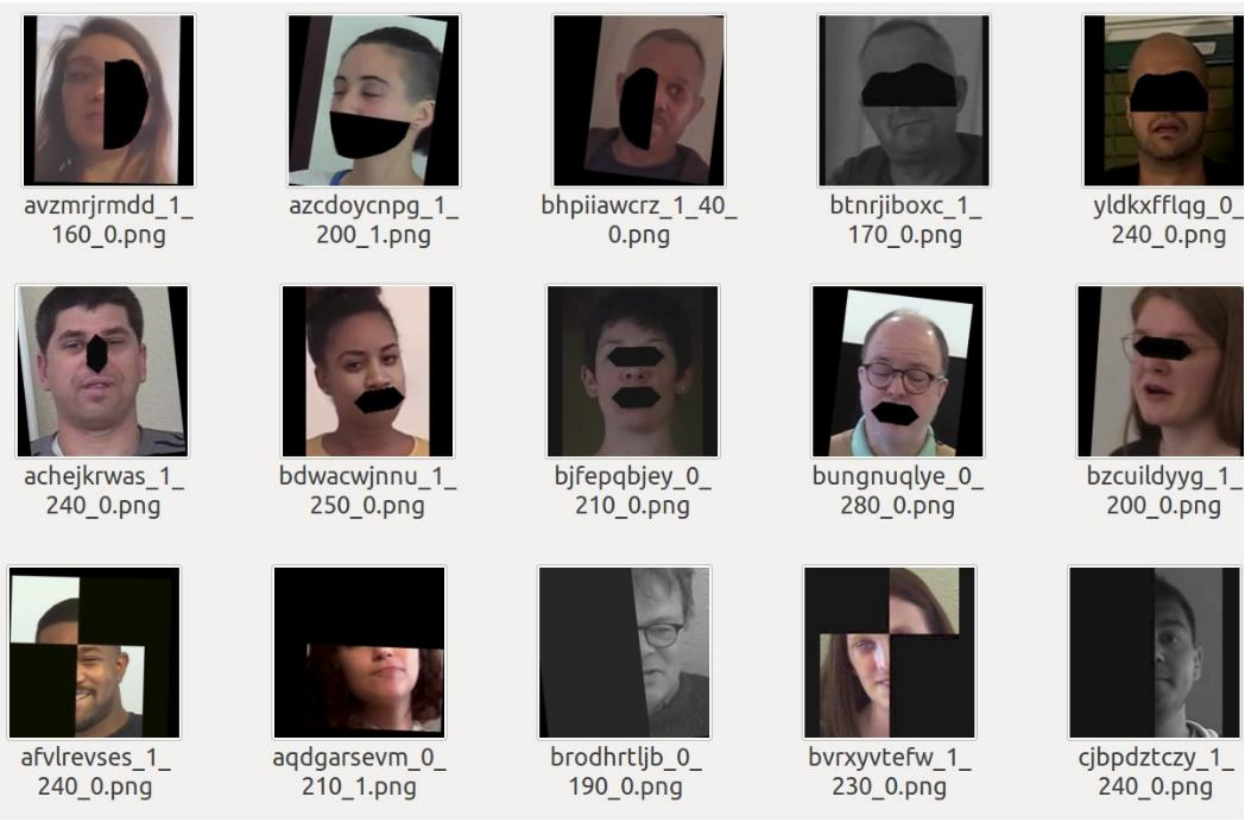
이미지 압축, 노이즈, 흐리게 만들기, 해상도 변경, 색상 깨트리기, 회전, Scaling등등

Albumentations 라이브러리 활용

```
eate_train_transforms(size=300):
turn Compose([
  ImageCompression(quality_lower=60, quality_upper=100, p=0.5),
  GaussNoise(p=0.1),
  GaussianBlur(blur_limit=3, p=0.05),
  HorizontalFlip(),
  OneOf([
      IsotropicResize(max_side=size, interpolation_down=cv2.INTER_AREA, interpolation_up=cv2.INTER_CUBIC),
      IsotropicResize(max_side=size, interpolation_down=cv2.INTER_AREA, interpolation_up=cv2.INTER_LINEAR),
      IsotropicResize(max_side=size, interpolation_down=cv2.INTER_LINEAR, interpolation_up=cv2.INTER_LINEAR),
  ], p=1),
  PadIfNeeded(min_height=size, min_width=size, border_mode=cv2.BORDER_CONSTANT),
  OneOf([RandomBrightnessContrast(), FancyPCA(), HueSaturationValue()], p=0.7),
  ToGray(p=0.2),
  ShiftScaleRotate(shift_limit=0.1, scale_limit=0.2, rotate_limit=10, border_mode=cv2.BORDER_CONSTANT, p=0.5),
```

# Augmentation 종류

이미지 압축, 노이즈, 흐리게 만들기, 해상도 변경, 색상 깨트리기, 회전, Scaling등등
Albumentations 라이브러리 활용



# No External Data

애초에 제공된 영상이 너무 많기도 하고, Augmentation을 통해 외부 데이터는 일절 사용하지 않았음!

# 5. Generalization Approach

# 학습시키고자 한 것

Visual Artifacts ( Augmentation 없이도 시각적으로 눈에 띄는 인공적인 모습은 잘 잡아냄)

기존 얼굴과 눈코입(조작된 부분)간의 인코딩 타입 차이를 검출하는 것 ( Big Margin을 통해 극대화 가능)

Face Warping Artifacts 검출 : Big Margin을 통해 극대화 가능
https://arxiv.org/abs/1811.00656

Blending Artifacts : https://arxiv.org/abs/1912.13458https://arxiv.org/abs/1912.13458

# 6. Reference

SSIM : https://bskyvision.com/m/396

Blending Artifacts : https://arxiv.org/abs/1912.13458https://arxiv.org/abs/1912.13458

Face Warping Artifacts : https://arxiv.org/abs/1811.00656

Augmentation : https://github.com/albumentations-team/albumentations