연구 노트 2주차

▼ 2팀 : 20191467 이재형

✅연구 주제 : Adative data enginering Learning framework

✅2주차 Topic : Baseline model인 Face Transformer 파악

---

1. Face Transformer

2103.14803v2.pdf

- 위 논문 'Face Transformer for Recognition'은 트랜스포머 모델을 얼굴 인식에 적용하는 가능성을 탐구하고, 전통적인 합성곱 신경망 CNN과의 성능을 비교한다.
- 기존의 비전 트랜스포머(ViT)의 토큰 생성 과정을 수정하여 이미지 패치가 겹치도록 함으로써 중요한 얼굴 특징을 더 잘 포착할 수 있도록 설계되었다.
- 위 **모델의 구조**와 **ViT와의 차별점**, **성능**에 대해 알아보았다.

1-1. Modify tokens generations

- 기존의 비전 트랜스포머**[1]**는 이미지를 여러 개의 작은 패치로 분할하고, 이 패치들을 각각의 토큰으로 변환하여 트랜스포머 모델의 입력으로 사용한다.
- 하지만 이 방식은 각 패치 간의 정보가 분리되어 인접한 패치들 사이의 중요한 정보가 누락될 수 있는 문제가 존재한다.

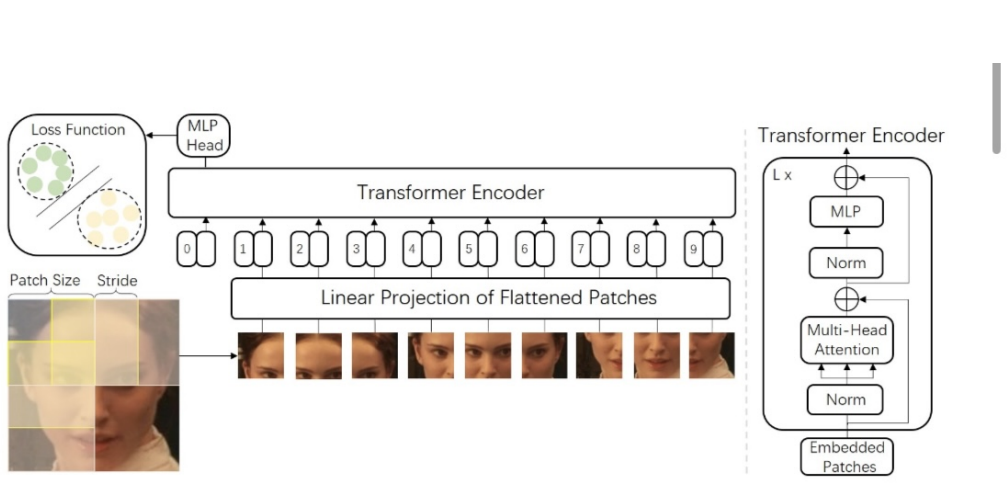


Fig. 1. The overall of Face Transformer. The face images are split into multiple patches and input as tokens to the transformer encoder. To better describe the inter-patch information, we modify the tokens generation method of ViT [1], to make the image patch overlaps slightly, which can improve the performance compared with original ViT. The Transformer encoder is basically a standard Transformer model [17]. Eventually, the face image embeddings can be used for loss functions [9], [10]. The illustration is inspired by ViT [1].

- 이를 해결하기 위해 저자들은 이미지 패치가 서로 겹치도록 토큰 생성 방식을 수정했다.
  - 위의 그림을 보면 얼굴 이미지는 다수의 작은 패치들로 분할하고, 이 패치들을 각각의 토큰으로 변환하여 트랜스포머 모델의 입력으로 사용한다.

적으로, 토큰을 생성할 때 이미지 패치들이 서로 약간 겹치게 하여 각 토큰이 인접한 패치의 정보 일부를 포함하도록 설계하였다.

- sliding patches를 통해 이미지 패치들이 겹치게 한다. 이는 왼쪽 여성의 얼굴 이미지의 겹쳐진 패치를 통해 직관적으로 파악할 수 있다.
- 이렇게 하면, 인접 패치 간(inter-patch)의 정보를 더 잘 통합할 수 있어 얼굴의 중요 특징들을 더 효과적으로 인식할 수 있게 된다.

1-2. Network Architecture

▼ Face transformer 모델은 기존의 Vision Transformer의 구조를 따르며, 주요 차이점은 이미지 패치 생성 방식임을 위에서 확인했다. 슬라이딩 패치 방식을 사용하여 토큰을 생성하는 과정을 살펴보자.

- **패치 생성 과정**
  - 이미지 $X \in R^{H*W*C}$는 $N * N$ 크기의 패치로 나뉘며, 각 패치는 겹치는 영역을 포함하여 생성된다.
  - 각 패치는 flatten되어 2D 패치 배열 $p \in R^{N*(P^2*C)}$로 변환된다.
    - P : 패치의 크기, C : 채널 수
  - 이 패치들은 학습 가능한 선형 투사를 통해 모델 차원 D로 매핑되어 패치 임베딩 p_k를 생성한다.
  - 클래스 토큰은 패치 임베딩과 연결되고, 트랜스포머 인코더를 통과하여 최종 얼굴 이미지 임베딩으로 사용된다.

- **위치 임베딩** : 위치 정보를 유지하기 위해 위치 임베딩이 각 패치 임베딩에 추가된다. 이는 모델이 패치의 원래 위치를 인식하도록 도와준다.

- **Transformer Encoder**
  - **Multihead Self-Attention, MSA** : 각 헤드에서 입력 시퀀스의 다른 부분에 주의를 기울이게 한다. Self-Attention 메커니즘은 query, key, value를 이용하여 입력 시퀀스의 각 위치에서 정보를 집계한다.
  - **Position-wise Feed-Forward Network, FFN** : 각 위치에 독립적으로 적용되는 FC-layer이다.

1-3. Loss function

- Face Transformer는 **CosFace** 손실 함수를 사용하여 학습된다.
  - CosFace 손실 함수는 클래스 간 거리를 최대화하고 클래스 내 거리를 최소화한다.
  - 이를 통해, 더 좋은 판별력을 가진 특징을 학습도록 설계된다.

$$L = -log\frac{e^{s*cos(\theta_y)}}{e^{s*cos(\theta_y)} + \sum_{j \neq y} e^{s*cos(\theta_j)}}$$

- 위의 수식에서 θ_y는 올바른 클래스에 대한 각도, θ_j는 잘못된 클래스에 대한 각도이고, s는 스케일링 파라미터이다.
- 위 손실 함수는 클래스 레이블 y에 해당하는 출력과 다른 클래스에 대한 출력 간의 각도 차이를 활용하여, 결과적으로 더 정확한 얼굴 인식을 위한 임베딩을 생성한다.

1-4. Performance

- 논문에서 제시한 Face Transformer와 CNN기반 모델의 성능 비교는 크게 **CASIA-WebFace**와 **MS-Celeb-1M** 이 2가지 데이터셋을 통해 이루어졌다.

[데이터셋]

1. **CASIA-WebFace**

- 상대적으로 작은 규모의 데이터셋.
- 트랜스포머 모델은 CNN과 비교하여 낮은 성능을 보임.

→ FT 모델은 충분한 양의 데이터로부터 학습될 때 효과적임을 시사함.

2. **MS-Celeb-1M**

- 대규모 데이터셋.
- 이 데이터셋에서 CNN과 비슷하거나 약간의 우수한 성능을 보임.

→ 예상한대로 대규모 데이터셋에서 잠재력을 충분히 발휘할 수 있음.

[성능 비교 결과]

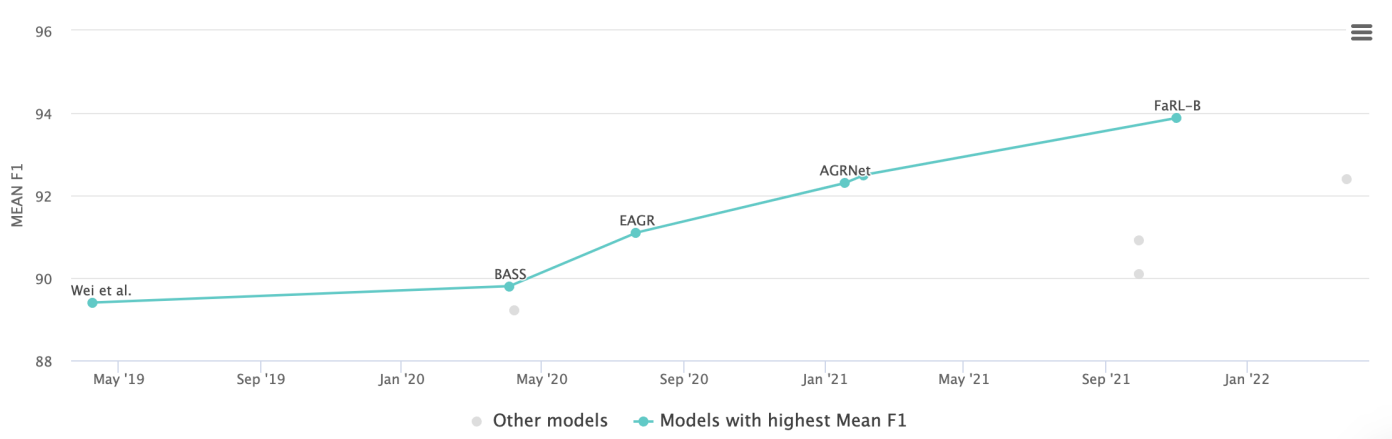
- **LFW (Labeled Faces in the Wild)**: Face Transformer는 MS-Celeb-1M 데이터셋을 사용할 때 CNN 모델과 유사한 정확도를 보여주었다. 특히, ViT 모델의 변형인 ViT-P12S8이 가장 높은 정확도를 달성했다.
- **CFP-FP (Celebrities in Frontal-Profile)**: 트랜스포머 모델은 다양한 포즈의 얼굴 인식에서도 경쟁력 있는 성능을 보여주었다.
- **IJB-C**: 국제 얼굴 인식 벤치마크에서도 트랜스포머 모델은 높은 성능을 유지했다. ResNet-100과 비교해도 1:1 검증에서 TAR@FAR에서 비슷하거나 약간 낮은 성능을 보였다.

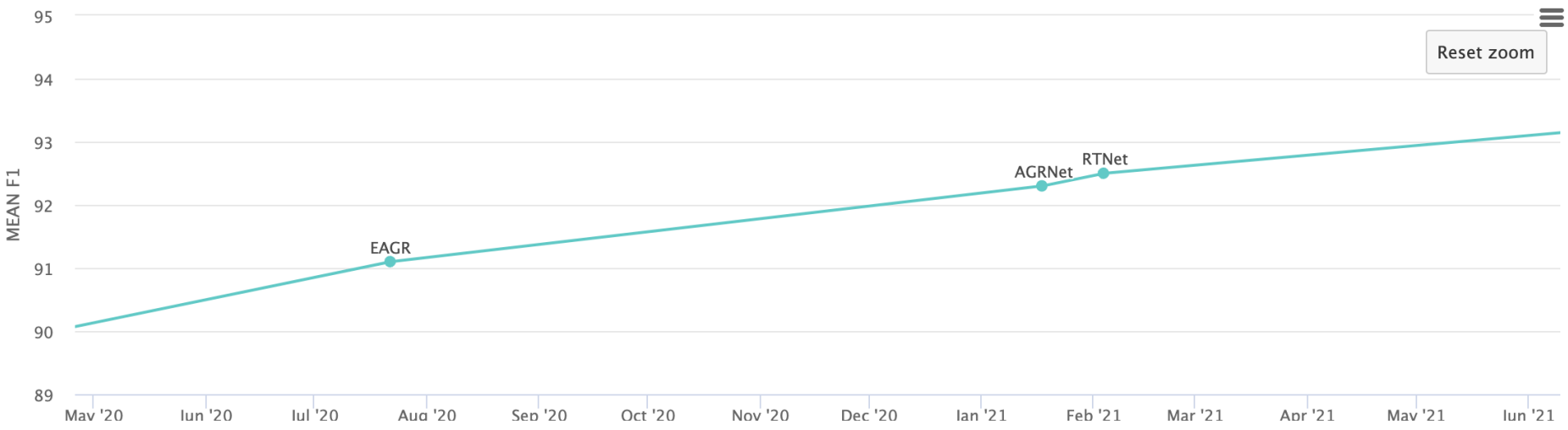
[추가 관찰]

- **비교 모델**: ViT-P8S8, ViT-P10S8, ViT-P12S8 등의 모델 변형들을 통해 패치의 겹침이 모델 성능에 미치는 영향을 관찰했다. 겹치는 패치가 있는 모델이 겹치지 않는 패치 모델보다 일반적으로 더 나은 성능을 보임.
- **Advantages and Disadvantages**: 트랜스포머 모델은 얼굴 영역에 집중하는 능력이 뛰어나며, 이로 인해 일부 벤치마크에서는 CNN 모델보다 우수한 결과를 보여 주기도 했다. 그러나 일부 조건에서는 CNN 모델이 더 나은 강인성을 보이기도 했다.

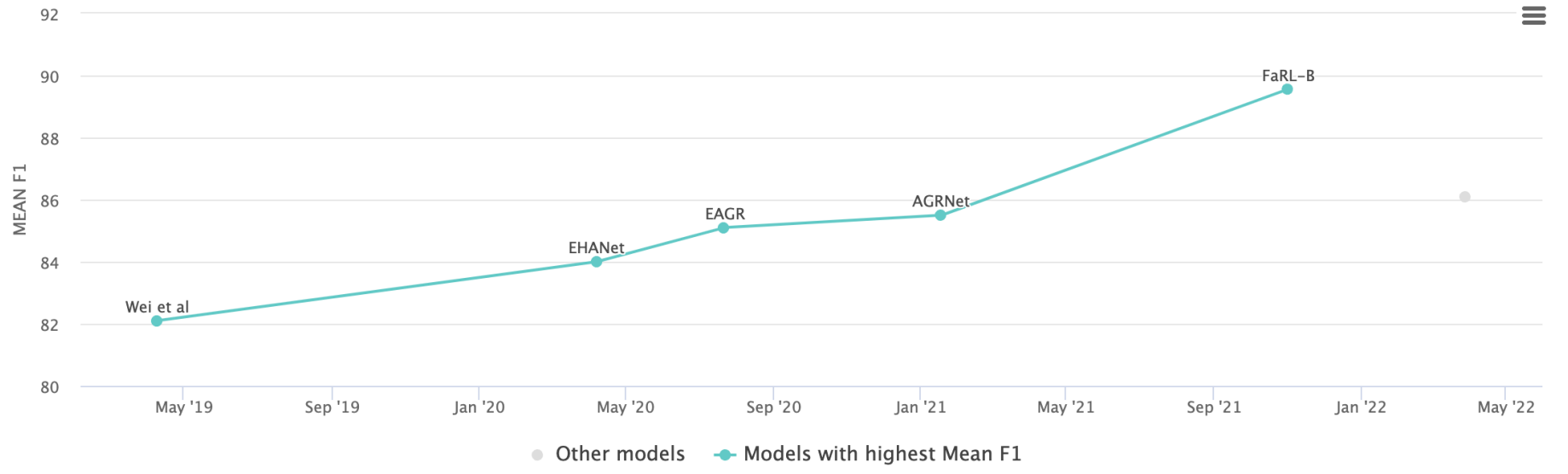
Model BenchMark(Mean- f1)

LaPa





CelebAMask-HQ



- 위의 두 성능 벤치마크에서는 FaRL-B가 가장 높은 성능을 보이나 학습을 할 때 추가 데이터셋을 활용하여 모델의 성능을 올린 경우이므로 이를 배제하면 현재로서 가장 높은 성능을 보이는 모델인 AGRNet과 RTNet은[3] 현재 추가 데이터 없이 가장 높은 성능을 보인다.

Mean- f1

f1 -score

- F1 점수는 정밀도와 재현율의 조화평균으로, 모델의 분류 성능을 평가하는 데 사용됩니다. 이 점수는 0에서 1 사이의 값을 가지며, 1은 최고의 성능을 나타냅니다.
- f1스코어는 Segmentation작업에서 F1-점수는 모델이 예측한 픽셀이 실제로 얼마나 정확한지를 측정하는 지표입니다.
- 이는 정밀도(Precision)와 재현율(Recall)의 조화평균을 기반으로 하며, 각각 모델이 예측한 픽셀 중 실제와 일치하는 픽셀의 비율과 실제 픽셀 중 모델이 정확하게 예측한 픽셀의 비율을 나타냅니다. 이렇게 F1-점수는 세그멘테이션 모델의 성능을 종합적으로 평가하는 데 사용됩니다.

Select-reward- function

$$PQ = \underset{k \in K}{\text{mean}} \frac{\sum_{(p,g) \in TP_k} IoU(p,g)}{|TP_k| + \frac{1}{2}|FP_k| + \frac{1}{2}|FN_k|}$$

https://arxiv.org/pdf/2302.08242

위의 그림은 Panomatic(Semantic + instance) Segmentation에서 사용하는 Eval Function으로 이를 Task reward로 활용하여 아래의 식으로 리워드를 부여한다.

$$reward(y,k) = \left[\sum_{(p,g)\in TP_k} IoU(p,g)\right] - w|FP_k|$$

https://arxiv.org/pdf/2302.08242

Reward function으로는 PQ 평가 지표를 최대로 하기 위해 IOU가 최대가 되어야 하므로 IOU를 최대로 하고 각자 개인의 원하는 상황에 따라 다르지만 거짓 양성을 억제하기 위해 벌칙 항으로 -wIFPI항이 붙어 있음을 볼 수 있음.
우리와 같은 경우에는 f1-score를 지표로 삼으므로
$f1-score = \frac{1}{precision} + \frac{1}{recall}$이고 이를 우리의 상황에 대입하면 적용하고자 하는 테스크에 적합한 리워드를 지정할 수 있다.

$$reward(y,k) = f1 - w * FP_k$$
$$(0 < w < 1)$$

위의 리워드는 우리의 목적 함수에 부합하는 리워드임을 알 수 있음.

```
    def binary_f1_score(preds, labels, w = 0.3):
         # preds와 labels는 이진 텐서여야 함
         tp = (preds * labels).sum().float()
         tn = ((1 - preds) * (1 - labels)).sum().float()
         fp = (preds * (1 - labels)).sum().float()
         fn = ((1 - preds) * labels).sum().float()

         precision = tp / (tp + fp + 1e-6)
         recall = tp / (tp + fn + 1e-6)
         f1 = 2 * (precision * recall) / (precision + recall + 1e-6)
         return f1, fp
    def reinforce_step(self, batch, mode ='train'):
         x, y = batch

         # Pass through network
         pred = self.net(x).squeeze(1)

         # Calculate MLE loss
         # loss = F.binary_cross_entropy_with_logits(pred, y)

         # Generate sample and baseline by sampling from predicted probs
         dist = Bernoulli(logits=pred)
         sample = dist.sample()
         baseline = dist.sample()
                 reward_train, f_1 = binary_f1_score(sample, y)
                 reward_base, f_2 =  binary_f1_score(baseline, y)

        # Calculate reward (F1) with baseline
                 reward = (reward_train -w * f_1 ) - (reward_base - w *f_2)
        # Get the log prob of sampling each prediction in batch
        log_prob = torch.mean(dist.log_prob(sample), dim=[1, 2])

        # Calculate reinforce loss
        loss = torch.mean(-log_prob * reward)

        # Get metrics
        metrics = getattr(self, f"{mode}_metrics")(pred, y.long())

        # Log
        self.log(f"{mode}_loss", loss, on_epoch=True, prog_bar=True)
        for k, v in metrics.items():
            self.log(f"{mode}_{k.lower()}", v, on_epoch=True)
```

```
        return loss
```

Reference

[1] https://velog.io/@leehyuna/Vision-TransformerViT

[2]https://github.com/google-research/big_vision/blob/main/big_vision/train.py

[3]Paper with code