
텍스트 감성분석과 이미지 데이터 기반의 이모티콘 추천 시스템

권정혁, 김수연, 노명은, 문지원, 윤성준

Table of contents

01**연구 배경**

카카오톡 이모티콘 플러스, 사용자 경험, Content-based Filtering, Service Model

02**이미지 데이터**

Style Transfer, CLIP 으로 그림체 유사도 계산하기

03**텍스트 데이터**

KoBERT로 다중 분류 모델 만들기

04**결론**

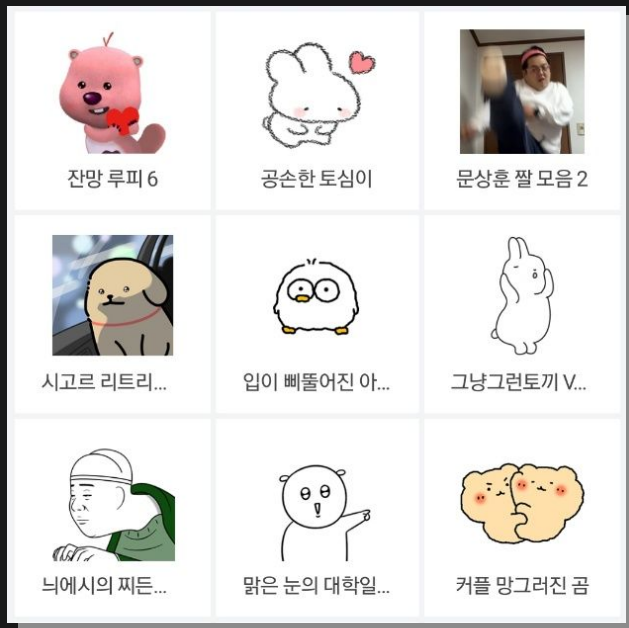
이모티콘 추천 시스템 결과

01

연구 배경

카카오톡 이모티콘 플러스, 사용자 경험, Content-based Filtering, Service Model

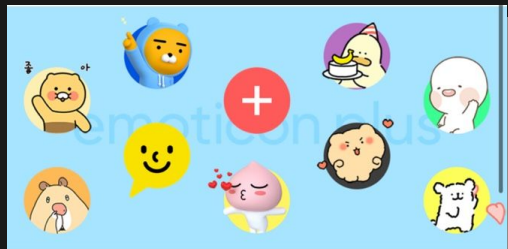
카카오톡 이모티콘



이모티콘은?

- ✓ 중요한 감정표현의 수단
- ✓ 수많은 이모티콘이 존재
- ✓ 역설적으로 이모티콘 선택 과정에 어려움 발생

카카오톡 이모티콘 플러스



**50만개 이모티콘을
무제한으로**

상황에 딱 맞는 이모티콘 추천까지

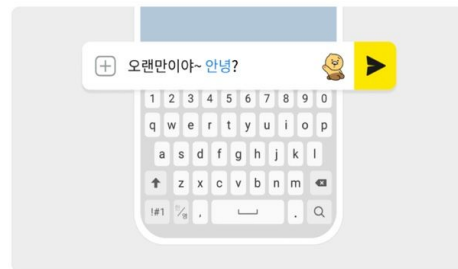
이모티콘 플러스

6,900원/월
앱스토어 전용가

특 답장, 어려우세요?

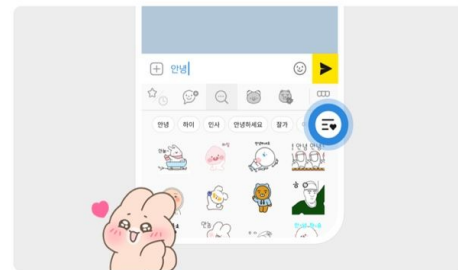
채팅창에 단어만 입력하면, 상황에 딱 맞는 이모티콘을 추천해 드려요.

더 빠르고 센스 있게 답장해 보세요.



내 취향 이모티콘만 쓱쓱!

이모티콘 취향이 확실한 분이라면, '마이취향 기능'으로 취향 저격 이모티콘만 모아서 사용할 수 있어요.



이모티콘 플러스 사용자 경험

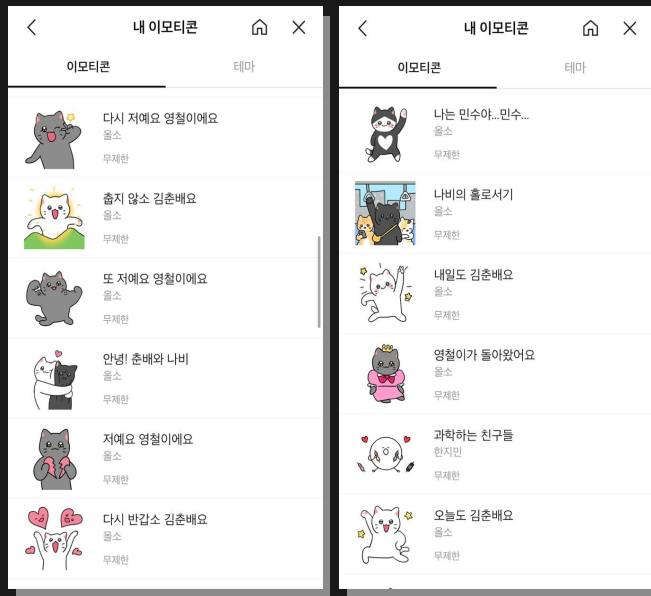
표 1. 자주 사용하는 이모티콘 선호 이유

순위	응답	비율
1	감성이 좋아서	39%
2	답변 대신	15%
3	감정 전달	12.9%
4	범용성	10.8%
5	대화 분위기 조성	10.8%
6	아이덴티티	6.4%
7	기타	4.3%

표 2. 이모티콘 플러스 내 검색 방식

순위	응답	비율
1	채팅창에 키워드 입력 후 추천 창에서 선택.	60%
2	서비스 페이지에서 검색한다.	20%
3	즐거찾기에 저장해 두고 사용한다.	6.6%
4	채팅 치다가 추천 받으면 즉흥적으로.	13.3%

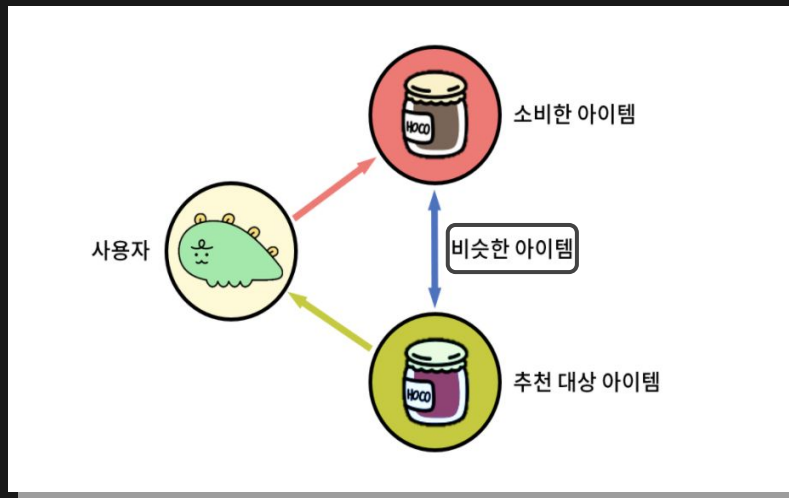
A씨의 실제 이모티콘 보유창



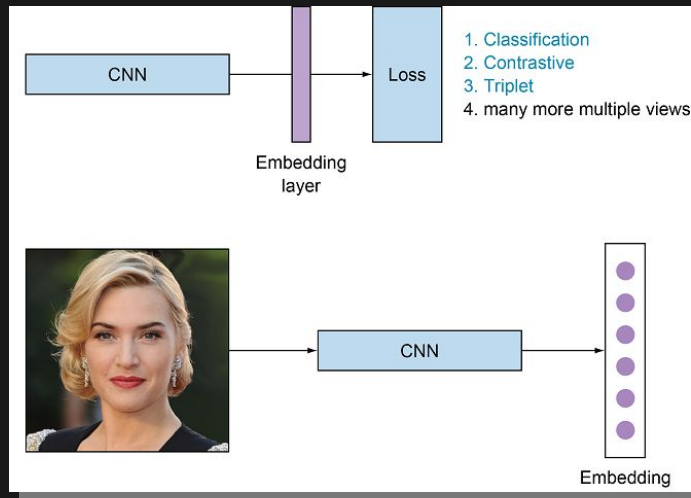
✓ 사용자의 취향과 상황에 맞는 이모티콘을 추천하는 것이 중요

Content-based Filtering

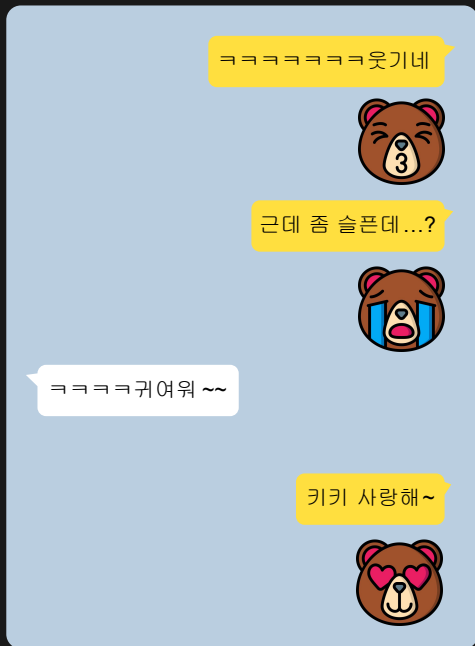
사용자가 소비한 아이템에 대해 아이템의 내용(content)이 비슷하거나 특별한 관계가 있는 다른 아이템을 추천하는 방법



Embedding 방법 사용

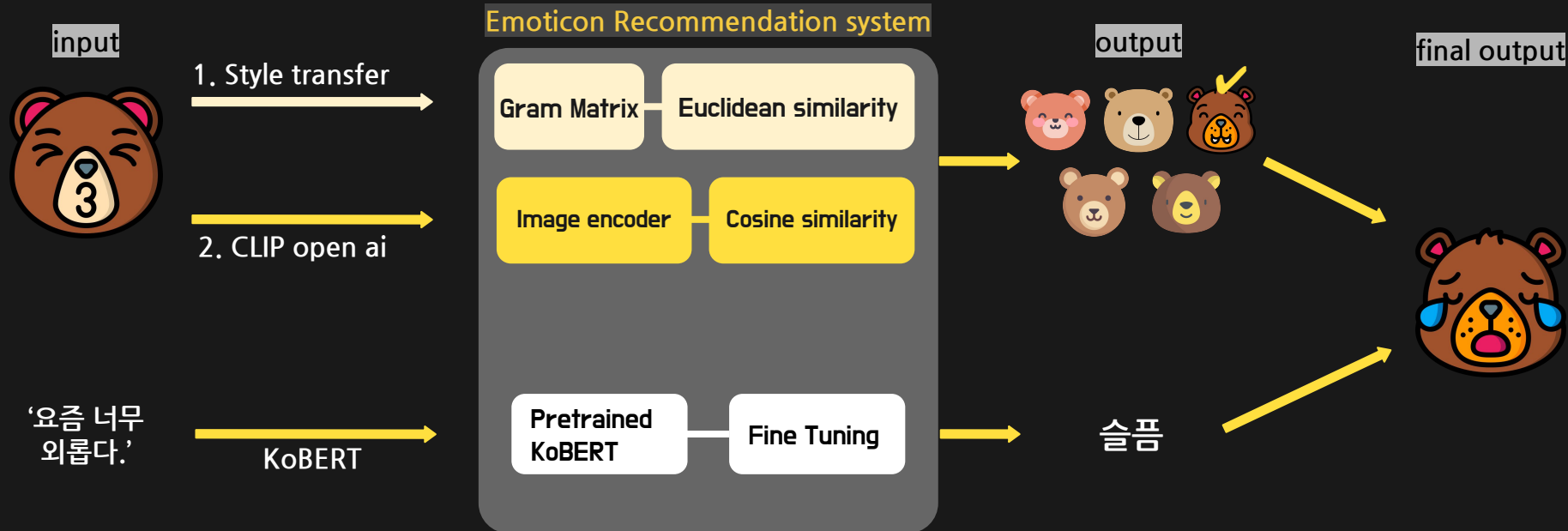


Service Model - Intro



사용자가 기존에 자주 사용한 이모티콘을 이용해
그림체가 유사한 이모티콘을 선별

Service Model



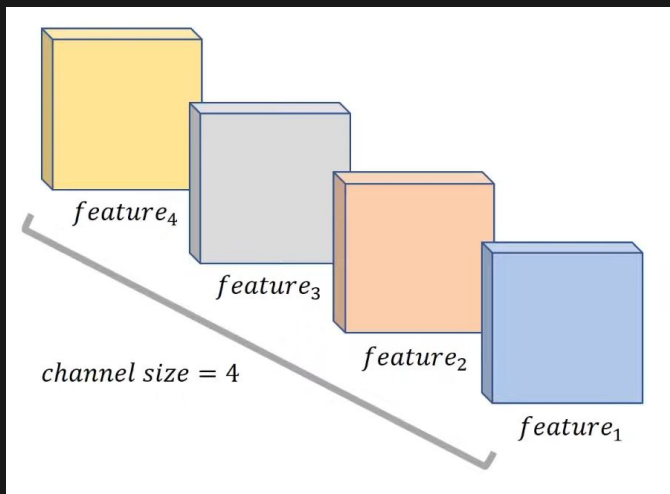
02

이미지 테이터

Style Transfer, CLIP으로 그림체 유사도 계산하기

방법 1. Style Transfer의 Gram Matrix

- ✓ 스타일(Style)은 서로 다른 특징(feature)간의 상관관계(correlation)를 의미
- ✓ G_{ij} = 특징 i와 특징 j의 상관관계(correlation)

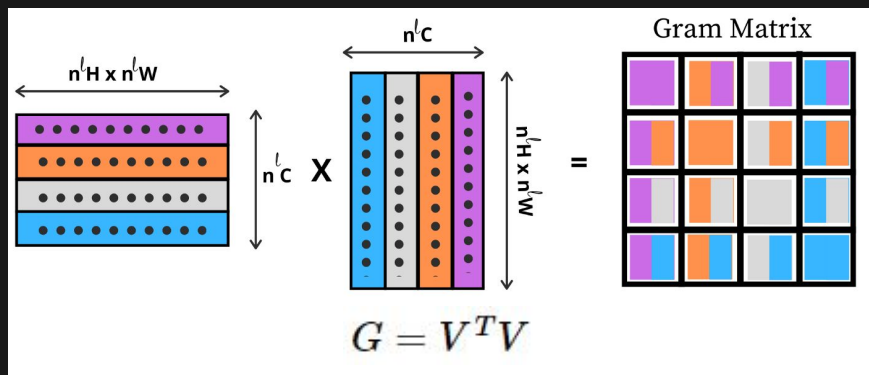
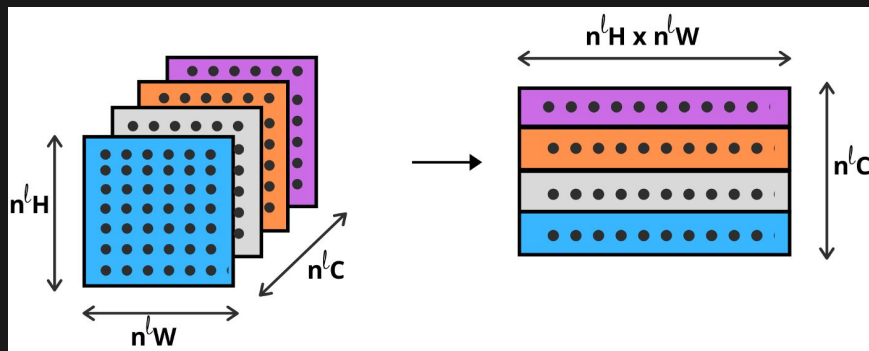


Gram Matrix (= Style Representation)

G_{11}	G_{12}	G_{13}	G_{14}
G_{21}	G_{22}	G_{23}	G_{24}
G_{31}	G_{32}	G_{33}	G_{34}
G_{41}	G_{42}	G_{43}	G_{44}

$$G_{ij} = \sum_k F_{ik} F_{jk} \quad \begin{array}{l} \text{(feature i와 feature j의 내적)} \\ \text{k는 활성화 값 위치} \end{array}$$

Gram Matrix



Code

```
class GramMatrix(nn.Module):
    def forward(self, input):
        b,c,h,w = input.size()
        F = input.view(b, c, h*w)
        G = torch.bmm(F, F.transpose(1,2))
        return G
```

parameters

- ✓ b : batch size
- ✓ c : channel
- ✓ h : height
- ✓ w : weight

Code

✓ pre-trained된 ResNet 사용

```
def style_extract(data):  
    total_arr = []  
    label_arr = []  
    resnet=Resnet().cuda()
```

ResNet(Residual neural network)?

Skip connection 방식 사용

=> 깊이(layer)가 깊어졌을 때 발생하는 문제 해결

```
    for idx,(image,label) in enumerate(data):  
        i = image.cuda()  
        i = i.view(-1,i.size()[0],i.size()[1],i.size()[2])
```

```
        style_target = list(GramMatrix().cuda()(i) for i in resnet(i))
```

✓ 각 layer로 부터 feature map을 추출하여 Gram Matrix 계산

✓ shape 조절 후 list에 array 저장

```
        arr = torch.cat([style_target[0].view(-1),style_target[1].view(-1),style_target[2].view(-1),style_target[3].view(-1)],0)  
        gram = arr.cpu().data.numpy().reshape(1,-1)  
        total_arr.append(gram.reshape(-1))  
        label_arr.append(label)
```

```
        if idx % 50 == 0 and idx != 0:  
            print(f'{idx} images style feature extracted..[{round(idx / len(data), 2) * 100}%]')  
print("\nImage style feature extraction done.\n")
```

```
    return total_arr, label_arr
```

Code

```
def main():
```

- ✓ configuration & data load

```
    config = configInfo('config.json')
    data = WebtoonLoader().data
    img_list, img_list2 = imgList(data)
```

- ✓ pre-trained된 모델을 가져와 feature 정보 추출

```
    resnet = Resnet().cuda()
    for param in resnet.parameters():
        param.requires_grad = False # 파라미터 업데이트 안 하도록 설정
    total_arr, label_arr = style_extract(data)
```

- ✓ feature 행렬 차원 축소 (고차원 -> 2차원 => 시각화 가능)

```
    result = tsne(total_arr, 2, 100)
    for i in range(len(result)) :
        result[i,0],result[i,1]= np.atleast_1d(result[i,0], result[i,1])
```

t-SNE?

차원 축소 및 시각화에 널리 쓰이는 방법

Code

✓ 가장 가까운 이웃 3개 계산

```
NN=NearestNeighbors(n_neighbors=4)
NN.fit(result)
distance, emoticon=NN.kneighbors([result[0]])
for idx in emoticon[0].tolist() :
    if idx != 0 :
        idx-=1
        print(config["idx_to_class"][str(idx)])
```



Test image

출력 결과

왼쪽 이모티콘 이미지(잔망루피 6)를 넣었을 때 가장 가까운 이웃 3개 출력

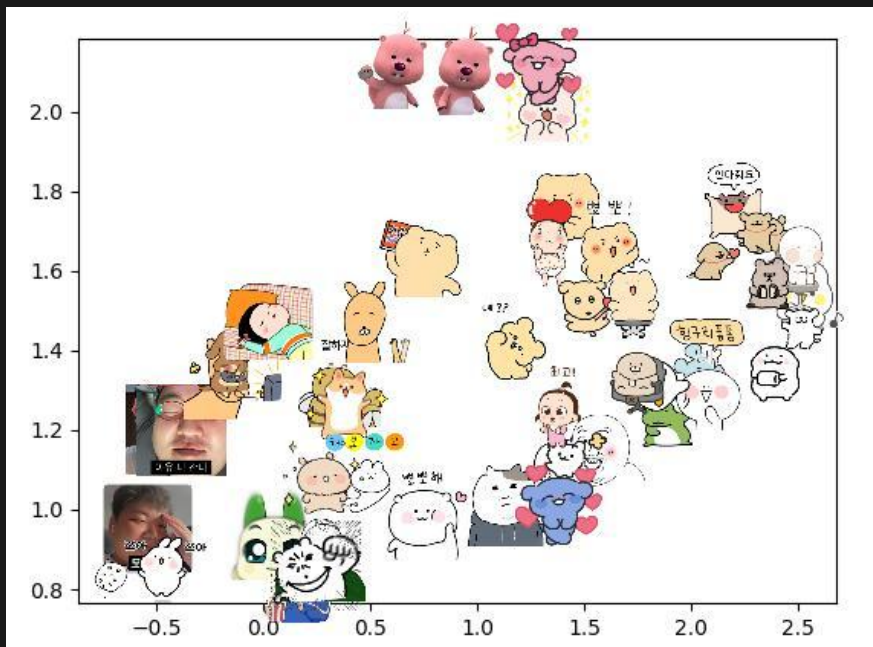
잔망	루피	5
잔망	루피	4
잔망	루피	
--

✓ 시각화

```
for i in range(len(result)):
    img_path = img_list[i]
    imscatter(result[i, 0], result[i, 1], image=img_path, zoom=0.1) # zoom= 0.2 였음
plt.savefig(config["visualization"]["title"])
```

```
if __name__ == '__main__':
    main()
```

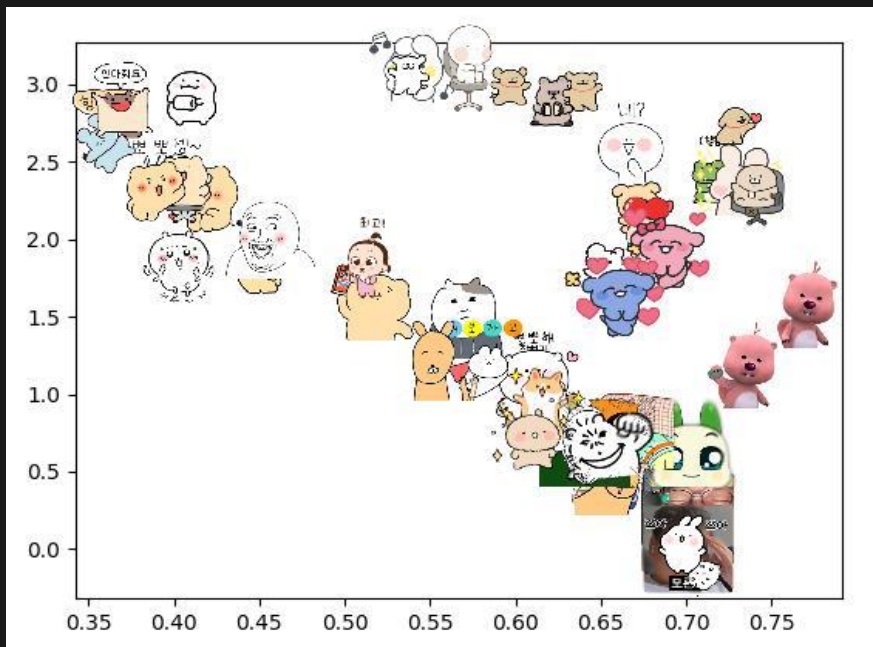
결과



RGB Scale

- ✓ 색깔이 더 중요하게 반영되는 경향이 있음.
- ✓ 비슷한 그림체이지만 색깔이 다르면 잡아내지 못함.
- ✓ 직관적으로 납득이 가지 않는 부분이 존재함.

결과

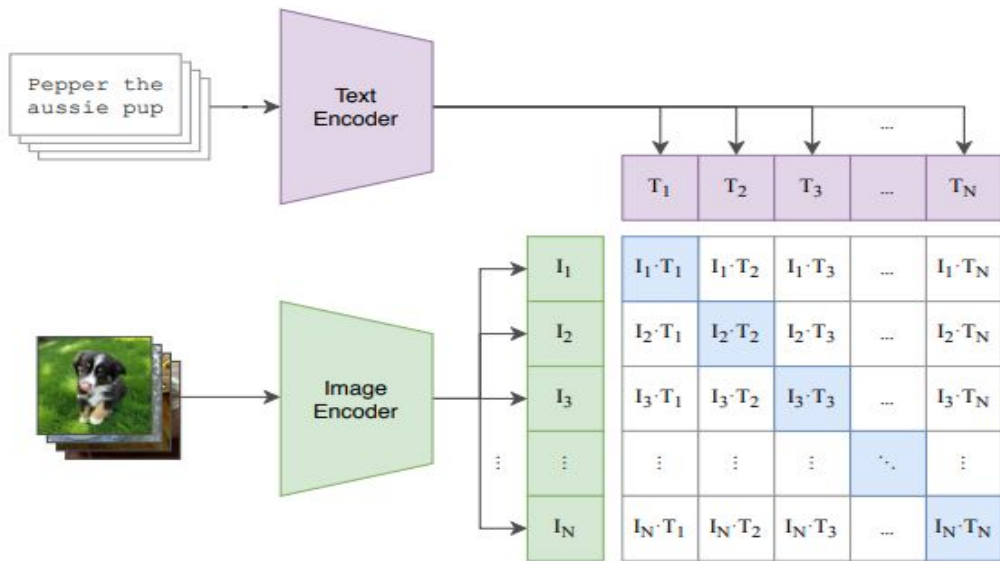


Gray Scale

- ✓ 색깔을 더 중요하게 반영하는 문제를 해결함.
- ✓ 하지만, 그림체의 특징을 추출하는 성능이 저하함.
- ✓ 여전히 직관적으로 납득이 가지 않는 부분이 존재함.

방법 2. CLIP openai

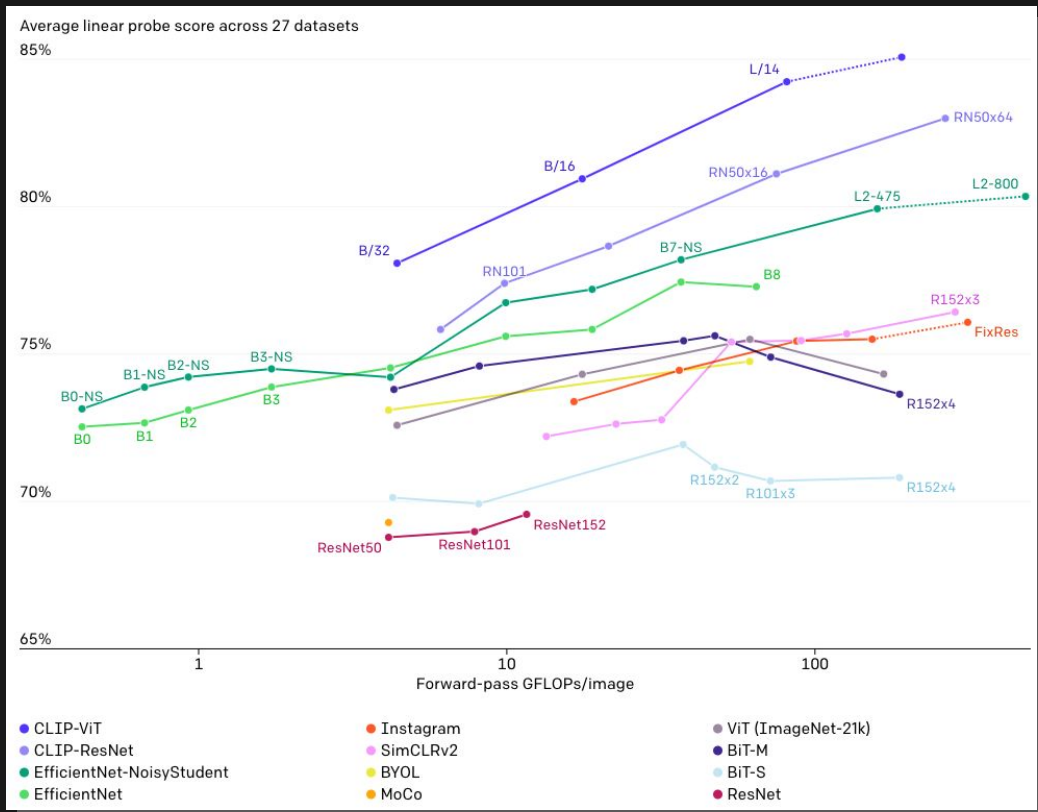
(1) Contrastive pre-training



CLIP

- ✓ 4억개의 (이미지, 텍스트) 쌍으로 학습한 모델
- ✓ Natural Language Supervision
- ✓ labeling 작업이 필요 없음.

CLIP



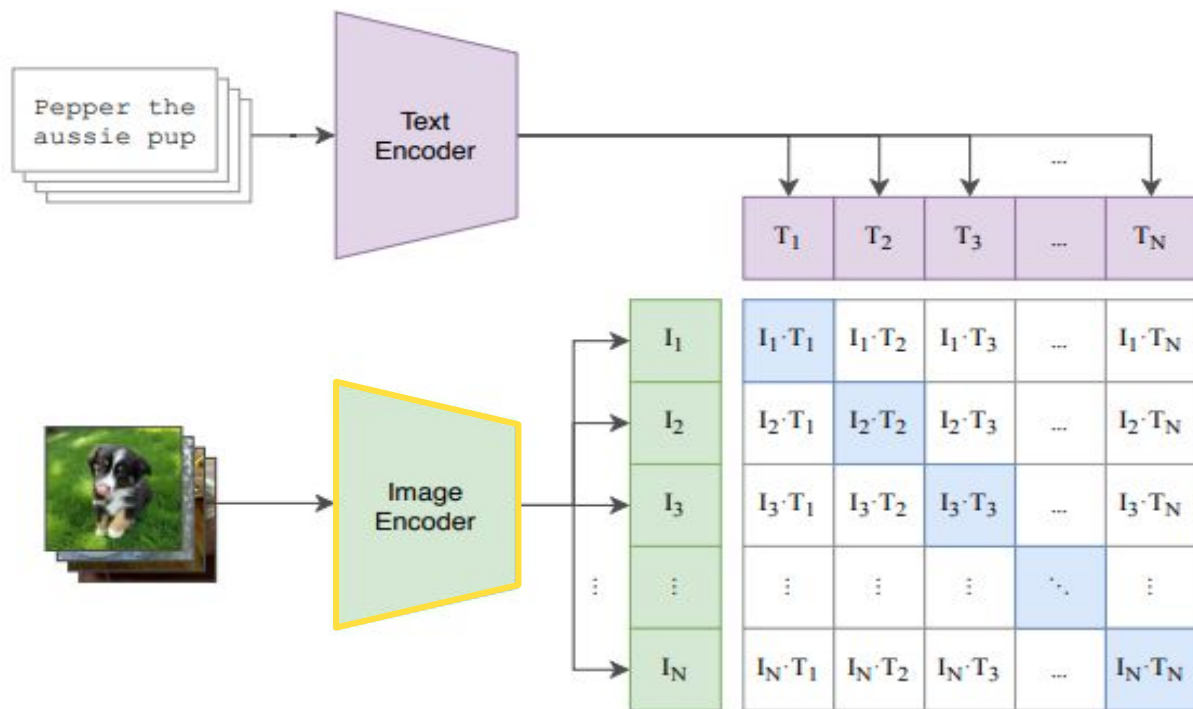
CLIP

- ✓ 4억개의 (이미지, 텍스트) 쌍으로 학습한 모델
- ✓ Natural Language Supervision
- ✓ labeling 작업이 필요 없음.
- ✓ CLIP 모델의 성능이 가장 높음.

CLIP

(1) Contrastive pre-training

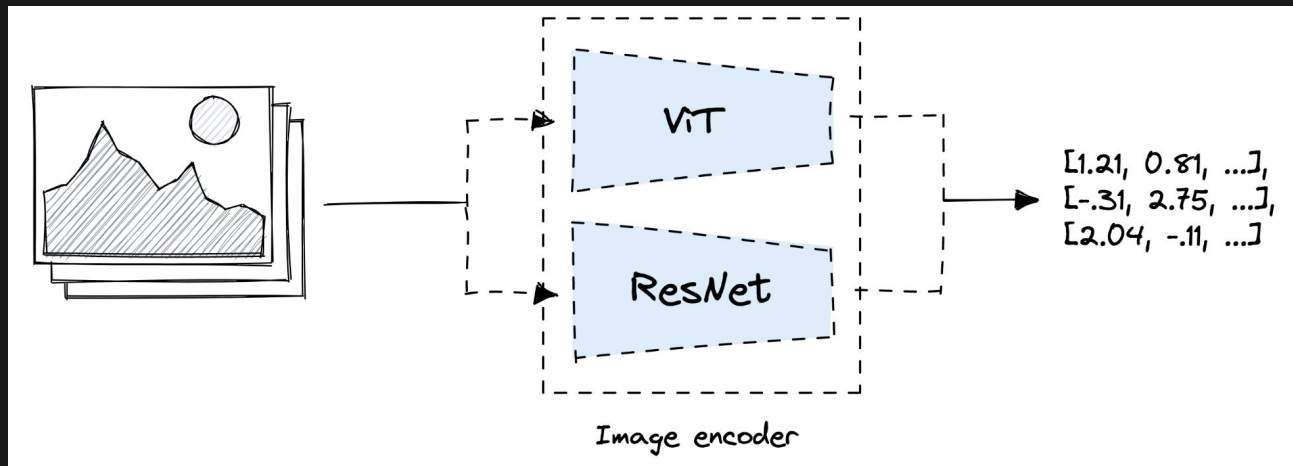
Input: (이미지, 텍스트)



CLIP_Image Encoder

VIT(Vision Transformer)?

NLP 분야에서 사용하는 Transformer를 Image Classification 분야에 맞게 약간 변형



인코딩 결과 = 512차원 벡터

이미지를 Patch로 분할 후 Sequence로 입력 -> NLP에서 단어가 입력되는 방식과 동일함.

Code

✓ load model

```
# 모델 불러오기 : Vision Transformer
model, preprocess = clip.load("ViT-B/32")
```

✓ 기존 이미지 특징 추출

```
image_input = torch.tensor(np.stack(images)).cuda()
with torch.no_grad():
    image_features = model.encode_image(image_input).float()
```

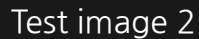
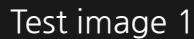
✓ 테스트 이미지 특징 추출

```
test_image_input = torch.tensor(np.stack(test_images)).cuda()
with torch.no_grad():
    test_image_features = model.encode_image(test_image_input).float()
```

✓ 코사인 유사도 계산

```
image_features /= image_features.norm(dim=-1, keepdim=True)
similarity = test_image_features.cpu().numpy() @ image_features.cpu().numpy().T
```

Cosine Similarity between image feature



Code 및 결과

```
# 투표용 리스트 생성
```

```
vote_list=[0]*50
```

```
# 각 사진에 대하여 유사도 값이 가장 높은 이모티콘 종류 K개 추출
```

```
for i in range(similarity.shape[0]):
```

```
    max_list = []
```

```
    thumbnails_list = []
```

```
    similarity_list = similarity[i].reshape(-1).tolist()
```

```
    maxidx_list = heapq.nlargest(5, similarity_list)
```

```
    for k in maxidx_list:
```

```
        max_list.append(similarity_list.index(k))
```

```
    for j in max_list:
```

```
        vote_list[j]+=1
```

```
        thumbnails_list.append(thumbnails_name[j])
```

```
print("추천", i+1, ":", thumbnails_list)
```

```
# 각 사진에서 가장 많은 표를 받은 이모티콘 선별
```

```
result_list=[]
```

```
max_vote=max(vote_list)
```

```
boundary=0
```

✓ 3개보다 적으면 추가

```
while len(result_list)< 3:
```

```
    limit=max(vote_list)-boundary
```

```
    for i in range(50):
```

```
        if vote_list[i]==max_vote-boundary:
```

```
            result_list.append(thumbnails_name[i])
```

```
            boundary+=1
```

```
print(result_list)
```

[Output]

추천 1 : ['문상훈 짤 모음', '문상훈 짤 모음 2', '곽튜브 짤 종합선물세트', '으른의 삶 다테시', '베고미의 사회생활']

추천 2 : ['문상훈 짤 모음', '문상훈 짤 모음 2', '꾸버는 열일중', '으른의 삶 다테시', '해피 땡땡이 커플 (물티즈)']

추천 3 : ['문상훈 짤 모음', '문상훈 짤 모음 2', '곽튜브 짤 종합선물세트', '유용한 긍정멘트 32독', '으른의 삶 다테시']

['문상훈 짤 모음', '문상훈 짤 모음 2', '으른의 삶 다테시']

이미지 데이터 : 제언

✓ 방법 1 : t-SNE 차원 축소 전에 **고차원** feature map에서 유사도 계산하기

✓ 방법 1 : 유사도 계산 방식을 유클리드에서 **코사인**으로 계산하기

✓ 유사도 기반으로 이모티콘을 선별할 때 반영했으면 좋았을 것들

- 사용자의 이모티콘 **데이터 기록 세분화** (얼마나 자주 사용 -> 가중치 적용 가능)
- 이모티콘의 유사도 **가중치** 반영 (유사도에 따라 추천 투표에 어떤 가중치로 반영할지)

✓ 다른 방법 시도

: EfficientNetV2와 같은 ImageNet 데이터의 카테고리 분류를 위해 미리 학습된 모델을 바탕으로 실제 사용할 데이터에 대해 **파라미터 미세 조정(Fine Tuning)**하여 수행한 다음, **분류 레이어의 입력**으로 들어가는 보틀넥 피처를 이미지 임베딩으로 사용하는 방법

03

텍스트 데이터

KoBERT로 다중 분류 모델 만들기

감성분석 (Sentiment analysis)

- ✓ 감정에 맞게 이모티콘을 추천해주는 것이 주제이므로 다중 분류 모델로 학습
- ✓ 충분한 대화 데이터 셋 확보가 어려워 사전 훈련이 된 모델 사용
- ✓ 사전 학습 모델 중 한국어로 학습한 koBERT 모델이 성능이 가장 좋다.

표 1. NSMC를 활용한 성능비교

Table 1. Performance comparison using NSMC

Model	Test Acc(%)
LSTM(128 Layer)	85.79
Bidirectional LSTM(128 Layer)	84.67
1D-CNN(128 Layer)	84.72
BERT(Multilingual-cased)	87.00
KoBERT(SKTBrain)	89.59

데이터 수집(AI HUB)

✓ 1. 한국어 감정 정보가 포함된 단발성 대화 데이터셋

- (<https://aihub.or.kr/aihubdata/data/view.do?currMenu=115&topMenu=100>)

✓ 2. 한국어 감정 정보가 포함된 연속적 대화 데이터셋

- (<https://aihub.or.kr/aihubdata/data/view.do?currMenu=115&topMenu=100>)

✓ 3. 감성대화 말뭉치

- (<https://aihub.or.kr/aihubdata/data/view.do?currMenu=115&topMenu=100>)

✓ 4. 온라인 구어체 말뭉치 데이터

- (<https://aihub.or.kr/aihubdata/data/view.do?currMenu=115&topMenu=100&aihubDataSe=realm&dataSetSn=625>)

=> 총 84MB 가량의 데이터를 확보

데이터 전처리

```
# "놀람"에 해당하는 텍스트들이 "중립", "혐오", "당황" 등을 넘나들기 때문에 "놀람"에 해당하는 데이터들을 삭제한다.
```

```
keti_single=keti_single[keti_single["감정"]!="놀람"]
```

- ✓ 중복되는 감정 통합
- ✓ Json 파일에서 필요한 내용만 추출하여 정리
- ✓ 필요없는 column은 제거

데이터 증강

- ✓ 라벨링이 되지 않은 데이터를 ChatGPT API를 통해 라벨링 진행

➡ Data augmentation

```
"I want you to classify this text to '불안, 당황, 분노, 슬픔, 중립, 행복, 혐오',  
the text is '%sentence_input"
```

✓
0초

```
[49] print(len(dataset_train))  
      print(len(dataset_test))
```

127513

19054

=> 1.1MB의 데이터 추가 확보

학습과정 - code

✓ pre-trained된 KoBERT + Fine tuning

 SKTBrain / KoBERT Public



#깃허브에서 KoBERT 파일 로드

```
!pip install git+https://git@github.com/SKTBrain/KoBERT.git@master
```

Code

✓ kobert 학습모델 만들기

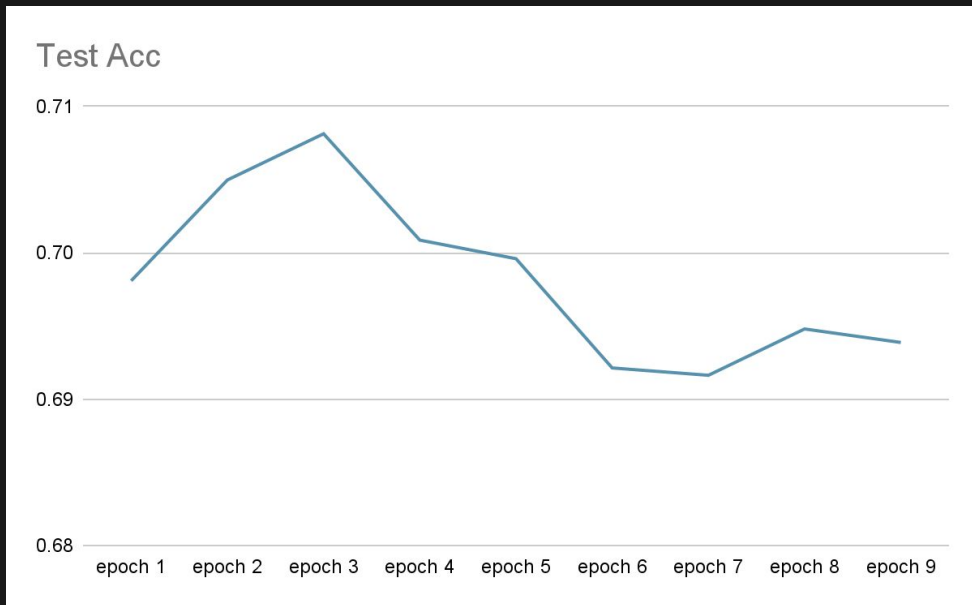
```
class BERTClassifier(nn.Module):
    def __init__(self,
                 bert,
                 hidden_size = 768,
                 num_classes=6,
                 dr_rate=None,
                 params=None):
        super(BERTClassifier, self).__init__()
        self.bert = bert
        self.dr_rate = dr_rate

        self.classifier = nn.Linear(hidden_size , num_classes)
        if dr_rate:
            self.dropout = nn.Dropout(p=dr_rate)
```

✓ 분류하려는 감정 갯수: 6개 => 6개 클래스로 분류

학습 과정 및 결과 - 1

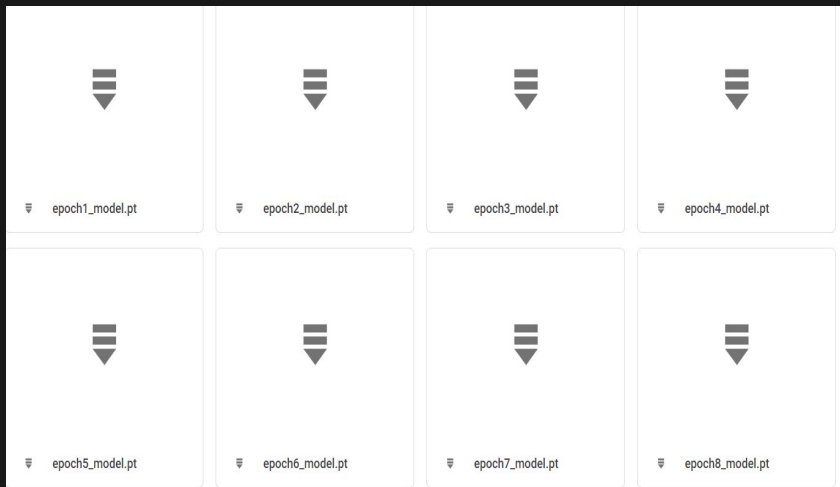
- ✓ 매 epoch마다 모델 학습 결과를 저장함.
- ✓ Epoch 3 이후 test acc 감소 -> 학습 중단



학습 결과 - 1

✓ 저장된 모델로 추론하는 것 결과 도출

➡ 가장 acc가 높게 나온 epoch3의 모델 활용



하고싶은 말을 입력해주세요 : 피곤해
>> 입력하신 내용에서 슬픔이 느껴집니다.

하고싶은 말을 입력해주세요 : 학습이 잘된 것 같아
>> 입력하신 내용에서 행복이 느껴집니다.

하고싶은 말을 입력해주세요 : 이 카페의 노래가 좋아
>> 입력하신 내용에서 행복이 느껴집니다.

하고싶은 말을 입력해주세요 : 난 더운게 싫어
>> 입력하신 내용에서 **중립**이 느껴집니다.

하고싶은 말을 입력해주세요 : 난 모기가 싫어
>> 입력하신 내용에서 **중립**이 느껴집니다.

하고싶은 말을 입력해주세요 : 쓰레기 치우는 건 귀찮아
>> 입력하신 내용에서 분노가 느껴집니다.

학습 방법 보안

- ✓ '혐오'의 경우 잘 드러나지 않았다. -> 분노에 통합
- ✓ test acc가 0.7정도로 train acc에 비해 낮았다
- ✓ 행복데이터를 기준으로 데이터 비율 재수정 후 2차학습 진행

1차 학습 데이터

중립	48616
슬픔	27517
분노	19710
불안	15999
당황	14670
행복	14406
혐오	5649

Name: 감정, dtype: int64



2차 학습 데이터

중립	25000
분노	15332
불안	15000
슬픔	15000
당황	14667
행복	14098

Name: 감정, dtype: int64

학습 과정 및 결과 - 2

- ✓ Epoch 3이후 test acc 감소
- ✓ 매 Epoch마다 모델 학습 결과를 저장함

Epoch 1 test acc 0.7000545662416852

Epoch 2 test acc 0.7397681451612903

Epoch 3 test acc 0.7405745967741936

Epoch 4 test acc 0.7316853005865103

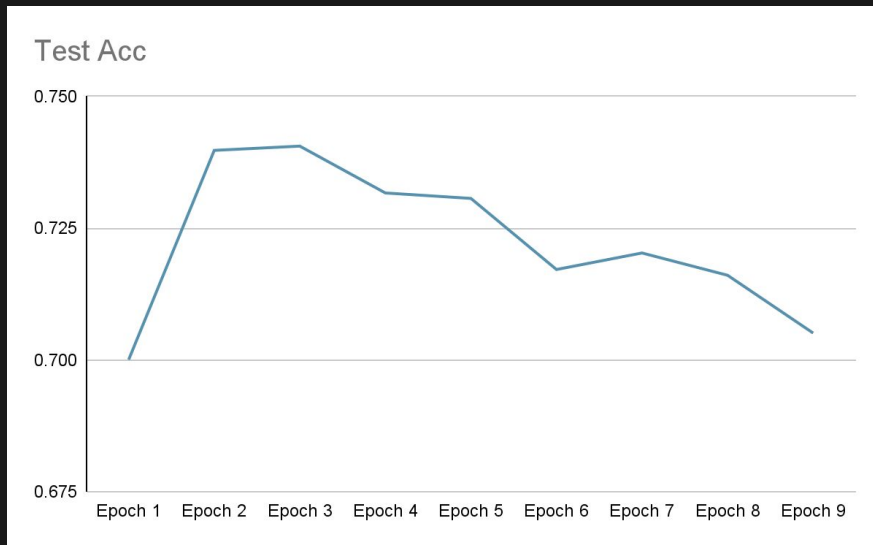
Epoch 5 test acc 0.7306525301023401

Epoch 6 test acc 0.7171611696230599

Epoch 7 test acc 0.7202965631929047

Epoch 8 test acc 0.7160660338137472

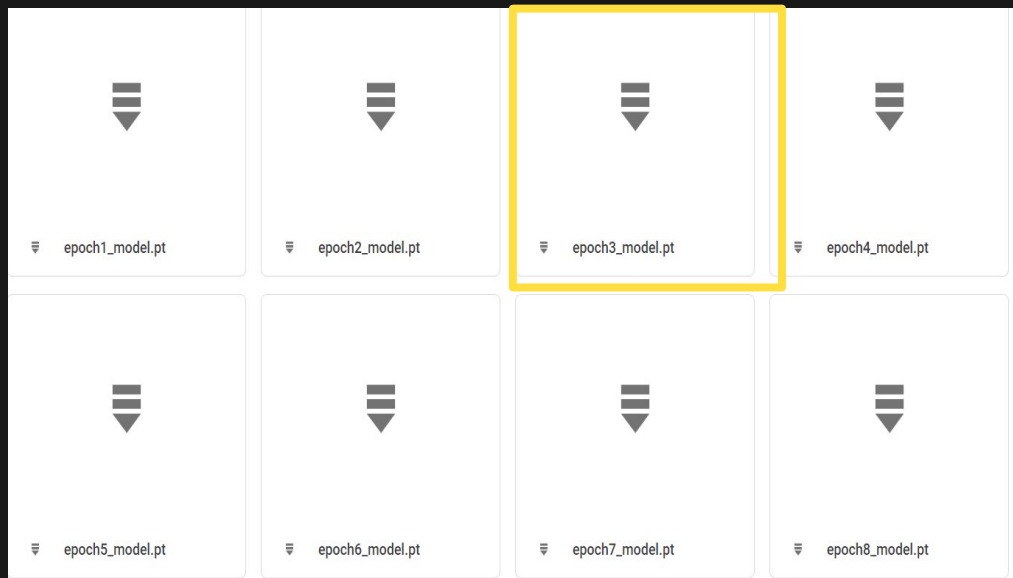
Epoch 9 test acc 0.7051041089246121



학습 결과 - 2

✓ 저장된 모델로 추론하는 것 결과 보이기

-> 가장 acc가 높게 나온 epoch3의 모델 활용



하고싶은 말을 입력해주세요 : 어제 저녁에 먹은 치킨 e o 개맛있었음 존맛탱 행복

하고싶은 말을 입력해주세요 : 과제 증발해버림;; 내 학점도 같이 증발할 듯 불안

하고싶은 말을 입력해주세요 : 미친 교수님이 과제를 또 주셨어 개오바 분노

하고싶은 말을 입력해주세요 : 어제 부모님과 싸웠어.. 너무 속상하다 슬픔

하고싶은 말을 입력해주세요 : 밥 먹었어? 중립

한계점

- ✓ 데이터 비율이 고르지 못했다.
- ✓ ChatGPT로 데이터 증강을 시도했으나, 과연 잘 된 것인지 검증을 하지 못했던 점.
- ✓ 이모티콘 라벨링 과정에서 부정적인 감정이 별로 없었다.

04

결론

이모티콘 추천 시스템 결과

결론 : 이모티콘 추천 결과 예시1

사용자의 이모티콘 사용 경향 : **실사 인물 형태**의 이모티콘

Test image 1



Test image 2



Test image 3



1. 그림체 기반 필터링

추천 1 : ['문상훈 짤 모음', '문상훈 짤 모음 2', '곽튜브 짤 종합선물세트', '으른의 삶 다테시', '베고미의 사회생활']

추천 2 : ['문상훈 짤 모음', '문상훈 짤 모음 2', '꾸버는 열일중', '으른의 삶 다테시', '해피 댕댕이 커플 (물티즈)']

추천 3 : ['문상훈 짤 모음', '문상훈 짤 모음 2', '곽튜브 짤 종합선물세트', '유용한 긍정멘트 32쪽', '으른의 삶 다테시']

['문상훈 짤 모음', '문상훈 짤 모음 2', '으른의 삶 다테시']

결론 : 이모티콘 추천 결과 예시1

input : “너 너무 사람 킹받게 하는 것 같아.”

```
sentence = input("하고싶은 말을 입력해주세요 : ")  
sentiment=predict(sentence)  
print(sentiment)  
sentiment_filtering=style_filtering[style_filtering['감정']==sentiment]
```

하고싶은 말을 입력해주세요 : 너 너무 사람 킹받게 하는 것 같아
분노

output :



결론 : 이모티콘 추천 결과 예시2

사용자의 이모티콘 사용 경향 : **실사 인물 형태**의 이모티콘

Test image 1



Test image 2



Test image 3



1. 그림체 기반 필터링

추천 1 : ['으른의 삶 다테시', 'GO라니', 'GO라니 2', '먼작귀2탄~먼가작고귀여운녀석~(치이카와)', '업티콘 존댓말로 업업업!']

추천 2 : ['으른의 삶 다테시', 'GO라니 2', '해피 땡땡이 커플 (몰티즈)', 'GO라니', '꾸버는 열일중']

추천 3 : ['오늘의 짤 #히노애짤', '으른의 삶 다테시', 'GO라니 2', '꾸버는 열일중', 'GO라니']

['GO라니', '으른의 삶 다테시', 'GO라니 2']

결론 : 이모티콘 추천 결과 예시2

input : “ㅋㅋㅋ 문상훈 품 미쳤다”

```
하고싶은 말을 입력해주세요 : ㅋㅋㅋ 문상훈 품 미쳤다  
/usr/local/lib/python3.9/dist-packages/torch/utils/data/dataloader.py:55  
assert self._num_workers == 0  
행복
```

output :



결론 : 이모티콘 추천 결과 예시3

사용자의 이모티콘 사용 경향 : 동물 형태의 이모티콘

Test image 1



Test image 2



Test image 3



1. 그림체 기반 필터링

추천 1 : ['자신감 급상승 ! 남남이의 하루!', '꼬물꼬물 따랑해 2 (곰식이 ver)', '망그러진 곰 5', '안녕! 나는 익명이고 조금 쭈글해', '해피 댕댕이 커플 (리트리버)']

추천 2 : ['안녕! 나는 익명이고 조금 쭈글해', '망그러진 곰 5', '시고르 리트리버 댕댕라이프', '빈티지 고양이 3', '꾸버는 열일중']

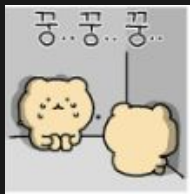
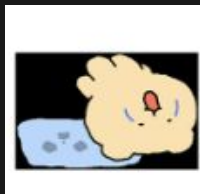
추천 3 : ['댕잘어울리는 댕댕이 (분댕이 ver)', '깜자라고해', '망그러진 곰 5', '댕잘어울리는 댕댕이 (파댕이 ver)', '자신감 급상승 ! 남남이의 하루!']
['망그러진 곰 5', '안녕! 나는 익명이고 조금 쭈글해', '자신감 급상승 ! 남남이의 하루!']

결론 : 이모티콘 추천 결과 예시3

input : “나 오늘 남자친구한테 차였어. 나는 아직 사랑하는데 내가 싫대. 눈물이 안멈춰”

하고싶은 말을 입력해주세요 : 나 오늘 남자친구한테 차였어, 나는 아직 사랑하는데 내가 싫대, 눈물이 안 멈춰 슬픔

output :



감사합니다