국내 관광 활성화를 위한 카테고리 분류

김예찬 / 박시언 / 박윤아 / 정승민 / 김민

목차

STEP 01

주제선정 및 배경

STEP 02

EDA 및 전처리

STEP 03

Transformer
모델

STEP 04

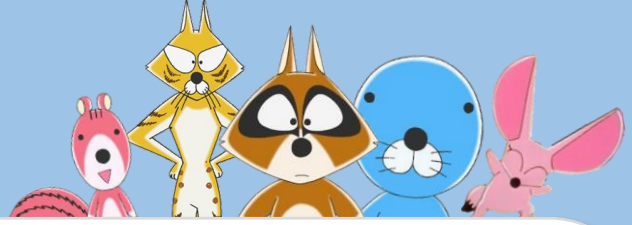
카테고리 분류 모델

STEP 05

3주차 계획

01 주제선정 및 배경

01 주제선정 및 배경

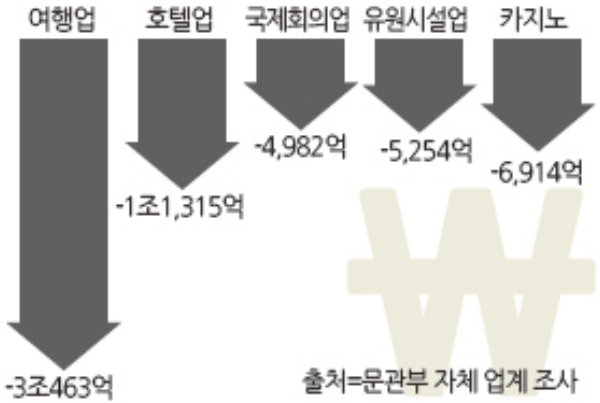


1. 분석 배경

상반기 한국 인·아웃바운드 현황
출처=법무부 출입국 통계

	인바운드	아웃바운드
인원수(명)	213만명	381만명
감소율(%)	-74.7	-74.6

상반기 관광업계 피해규모 단위=원
전체 피해액 5조8,928억원



출처=문관부 자체 업계 조사

뉴시스 PICK | 7일 전 | 네이버뉴스

문체부 "관광업계 고통 실감하고 있다"...비자·출입국 제도 등 개...

등 외국인 **국내** 관광(인바운드) 회복을 위한 여건이 차차 갖춰지고 있다"며 "하지만 **코로나**19 여파로 관광업계가 받은 **타격**이 워낙 커 회복을 실감하기 어려운 만...

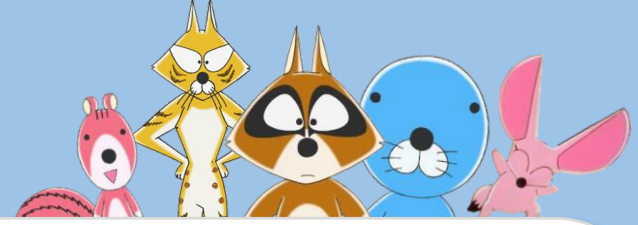
www.traveltimes.co.kr › news

[**코로나**19 **첫 확진 후** 9개월, 현황과 전망] **여행산업** 초토화...터널의 끝은 어디쯤에? <...

2020년 1월20일 한국에서 **첫 코로나**19 **확진자**가 발생한 이후 꼬박 9개월이 흘렀다. 불과 9개월 만에 **여행산업**은 전대미문의 극심한 침체에 빠졌고, 여전히 터널의 끝은 보이지 않고 있다. **코로나**19 9개월이 남긴 상처와 향후 전망을 살폈다. ●일상이 된 휴·폐업 그리고 휴·퇴직아웃바운드와 인바운드 부문은 **코로나**19로 인해 한마디로 ...

2020년 코로나19 발생으로 관광업계가 큰 타격을 입음

01 주제선정 및 배경



1. 분석 배경

그러나 최근 코로나 확산세가 감소하면서

콘텐츠 포화 지수

500%이상	500%이상	500%이상
매우 높음	매우 높음	매우 높음
블로그	카페	VIEW

▲ 관광 관련 검색량 증가, 콘텐츠 수 증가

뉴스홈 | 최신기사

문체부 '코로나 이후 방한관광 재도약' 토론회

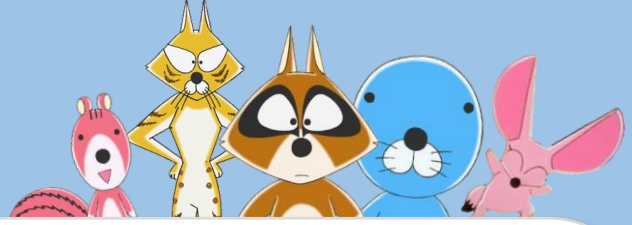
송고시간 | 2022-10-27 09:37

관광진흥기본계획 수립에 반영

문화체육관광부

▲ 정부의 관광산업 회복 및 재도약 발표

01 주제선정 및 배경



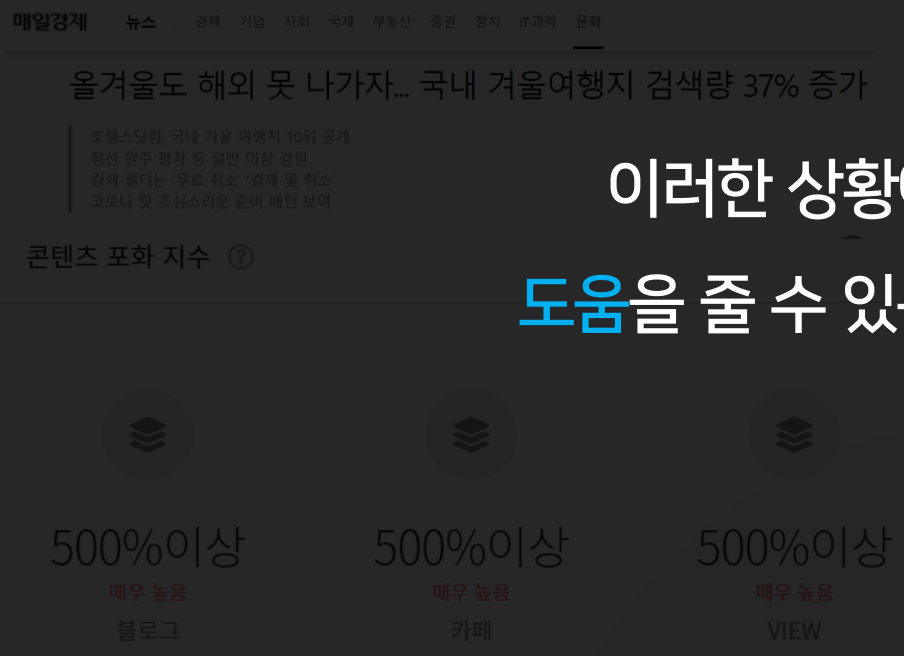
1. 분석 배경

따라서 관광 산업의 회복에 대해 국내 관광업계의 관심도 증가하고 있음

01 주제선정 및 배경

1. 분석 배경

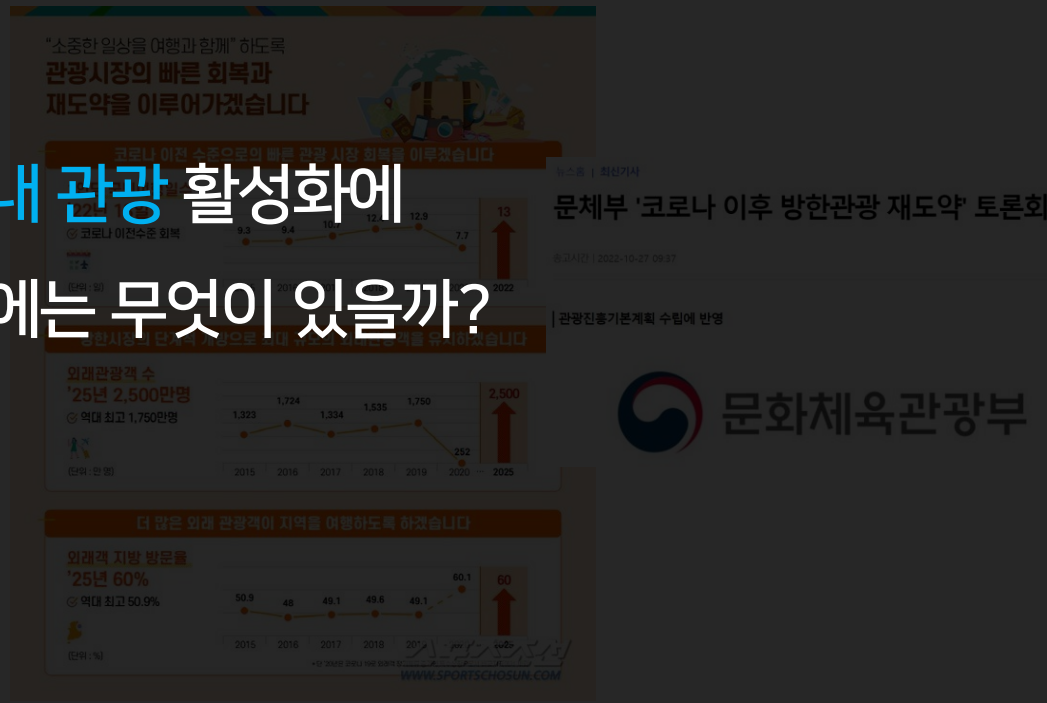
그러나 최근 코로나 [illegible]가 감소하면서



▲ 국내 관광 관련 검색량 증가, 콘텐츠 수 증가

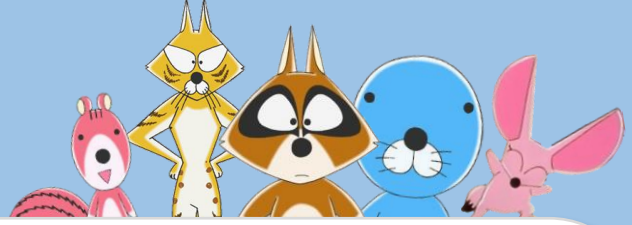
이러한 상황에서 국내 관광 활성화에

도움을 줄 수 있는 방안에는 무엇이 있을까?

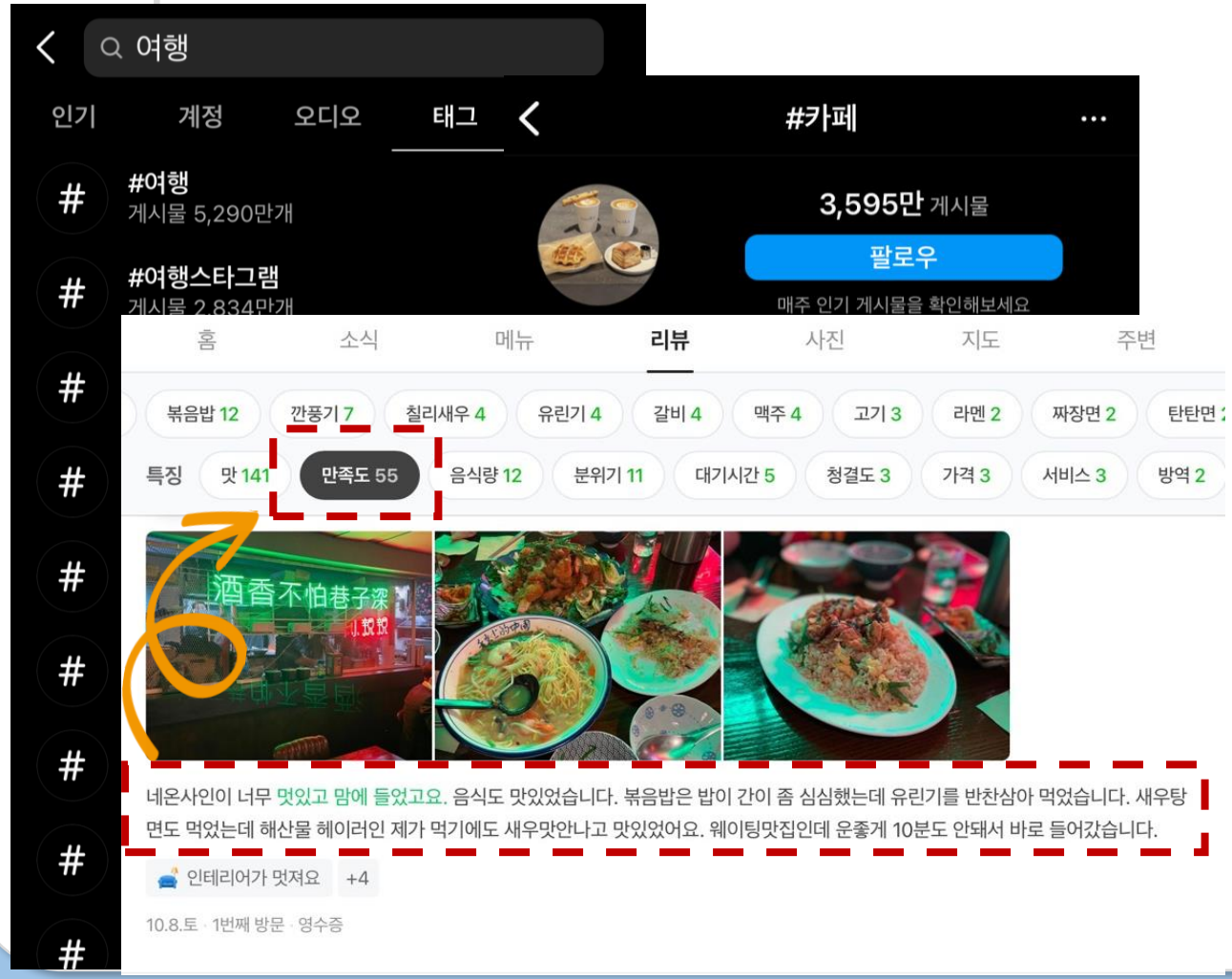


▲ 정부의 관광산업 회복 및 재도약 발표

01 주제선정 및 배경



1. 분석 배경

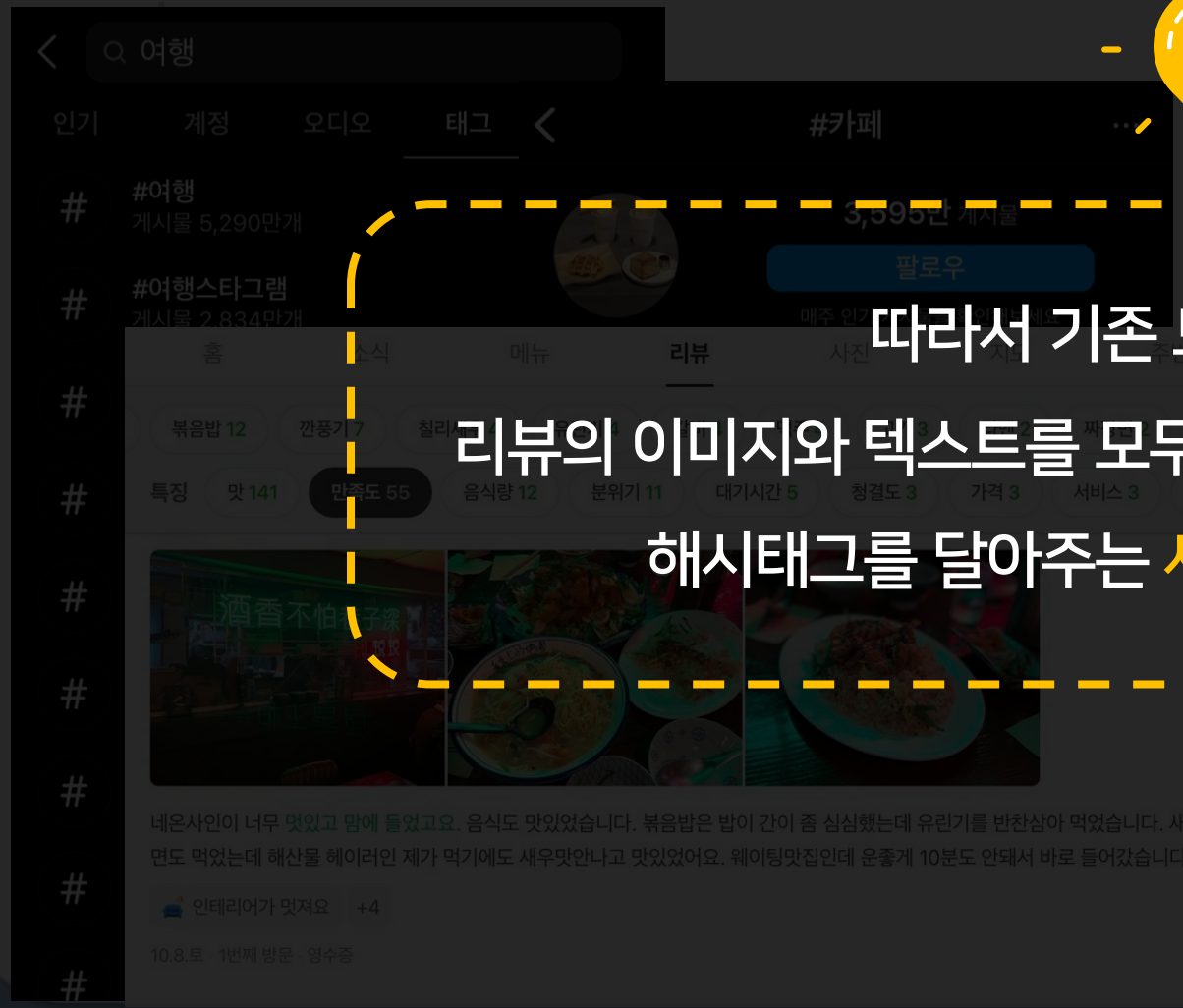


최근 # 해시태그를 통한 장소의 검색과 홍보가 활발하게 이루어지고 있음

그러나 기존모델은 리뷰 텍스트 중 키워드만 추출하여 해시태그 반영하기 때문에 정보가 한정되어 있음

01 주제선정 및 배경

1. 분석 배경



따라서 기존 모델을 개선하여

리뷰의 이미지와 텍스트를 모두 활용해서 **카테고리를 분류** 한 후

해시태그를 달아주는 **새로운 모델**을 만들어보자

최근 # 해시태그를 통한 장소의 검색과 홍보가

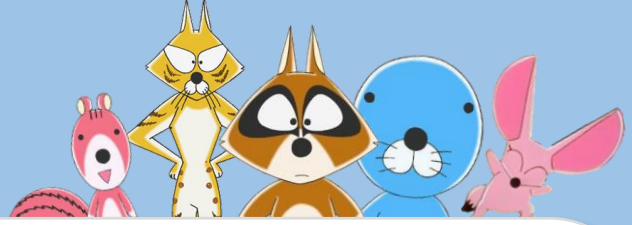
활발하게 이루어지고 있음

그러나 기존모델은 리뷰 텍스트 중

키워드만 추출하여 해시태그 반영하기 때문에

정보가 한정되어 있음

01 주제선정 및 배경



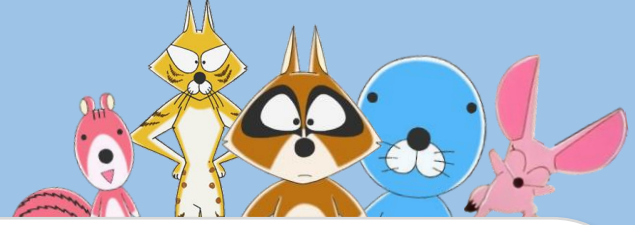
2. 기대 효과

해시태그를 통해 카테고리의 단계 별로 검색이 가능하다면
해당 장소에 대한 접근성이 높아짐

국내 관광 활성화에 도움 ↑

02 EDA 및 전처리

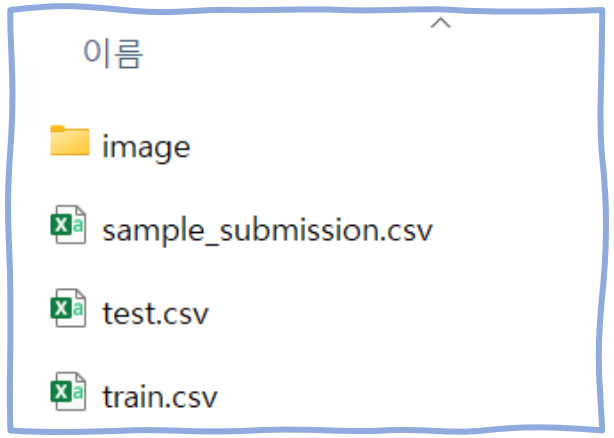
02 EDA 및 전처리



0. 데이터 소개

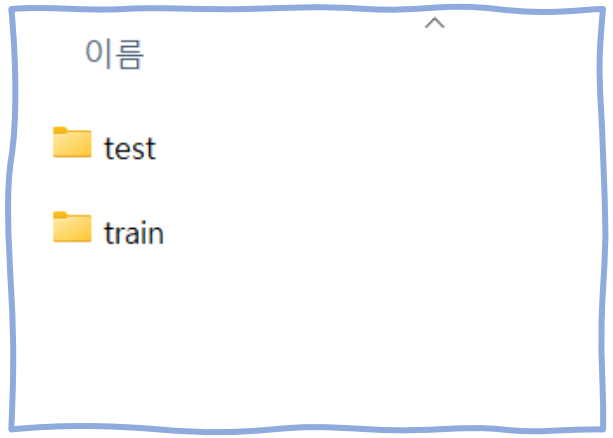
한국관광공사가 제공하는 국문관광 데이터

국내 관광지에 대한 이미지 데이터와 텍스트 데이터 제공



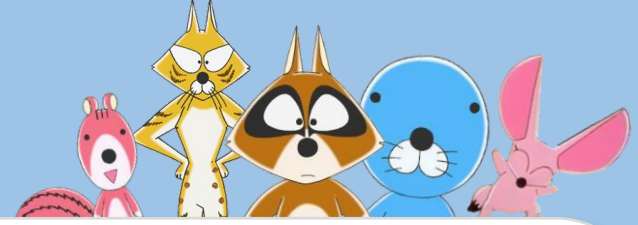
- CSV 파일로 존재

- train/ test/ sample_submission 3개의 파일



- 이미지 파일 안에 test/ train 존재

02 EDA 및 전처리



0. 데이터 소개

한국관광공사가 제공하는 국문관광 데이터 train. csv

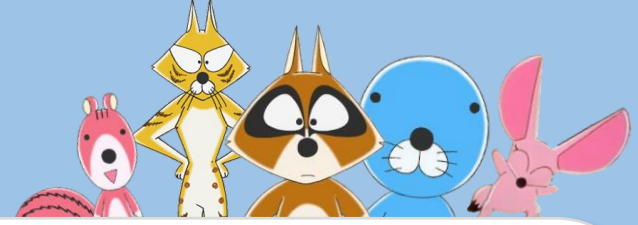
16987 X 6

id	Img_path	Overview	Cat1	Cat2	cat3
TRAIN_00000	./image/train/TRAIN_00000.jpg	소안항은 조용한 섬으로 인근해안이 청정해역으로 일찍이 김 양식을 …	자연	자연관광지	항구/포구
TRAIN_00001	./image/train/TRAIN_0001.jpg	기도 이천시 모가면에 있는 골프장으로 대중제 18홀이다. 회원제로 개장을 …	레포츠	육상레포츠	골프
TRAIN_00003	./image/train/TRAIN_00002.jpg	금오산성숯불갈비는 한우고기만을 전문적으로 취급하고 사용하는 부식 자재 또한 유기농법으로 …	음식	음식점	한식
…			…	…	…

이미지

텍스트

02 EDA 및 전처리



0. 데이터 소개

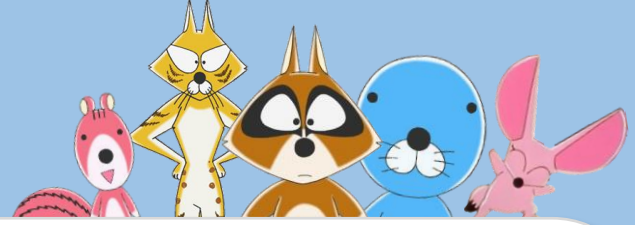
한국관광공사가 제공하는 국문관광 데이터 train. csv

16987 X 6

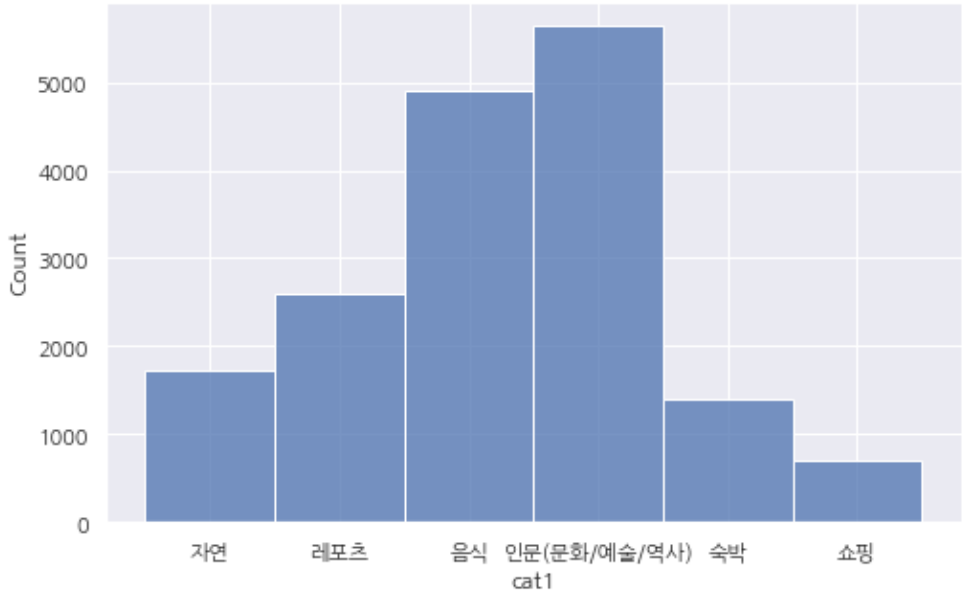
id	Img_path	Overview	Cat1	Cat2	cat3
TRAIN_00000	./image/train/TRAIN_00000.jpg	소안항은 조용한 섬으로 인근해안이 청정해역으로 일찍이 김 양식을 …	자연	자연관광지	항구/포구
TRAIN_00001	./image/train/TRAIN_0001.jpg	기도 이천시 모가면에 있는 골프장으로 대중제 18홀이다. 회원제로 개장을 …	레포츠	육상레포츠	골프
TRAIN_00003	./image/train/TRAIN_00002.jpg	금오산성숯불갈비는 한우고기만을 전문적으로 취급하고 사용하는 부식 자재 또한 유기농법으로 …	음식	음식점	한식
…	…		…	…	…

이미지 데이터와 텍스트 데이터를 이용해 자동으로 카테고리를 분류하는 것이 목표

02 EDA 및 전처리



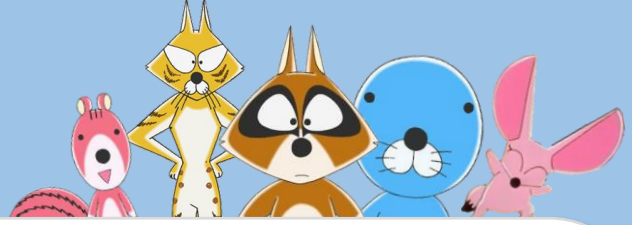
1. DATA 증강 전



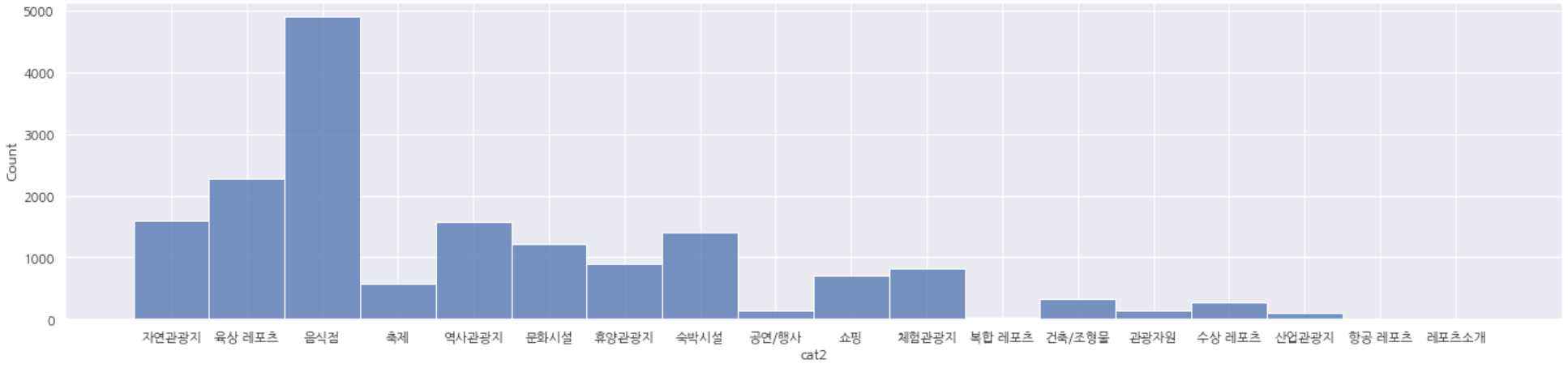
Category 1

총 분류는 6개, 가장 적은 분류와 많은 분류는 약 6배정도 차이가 남

02 EDA 및 전처리



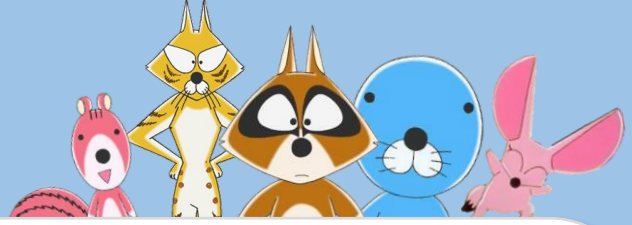
1. DATA 증강 전



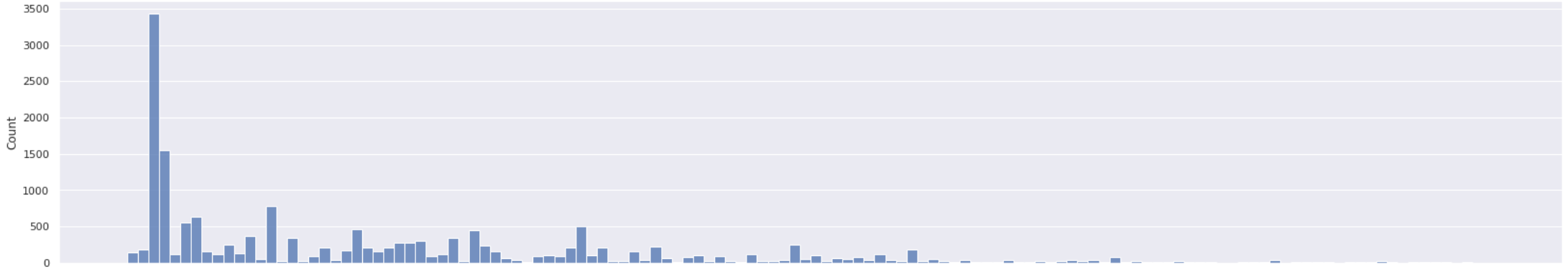
Category 2

총 분류는 18개, 매우 불균형한 분포를 보임

02 EDA 및 전처리



1. DATA 증강 전



Category 3

총 분류는 128개, 매우 불균형한 분포를 보임

02 EDA 및 전처리

1. DATA 증강 전

이후에 데이터 증강을 통해

불균형한 분포를 완화시킬 예정!

Category 3

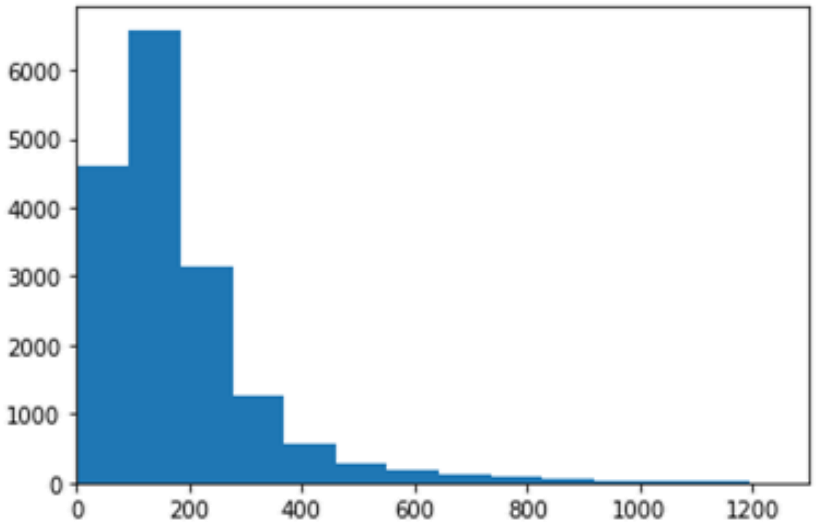
총 분류는 128개, 매우 불균형한 분포를 보인다

02 EDA 및 전처리

2. Text Data 길이 분포 시각화

텍스트 데이터를 전처리 후,

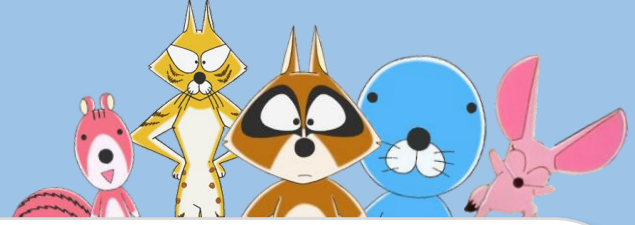
토큰화된 단어 개수 별 데이터의 분포를 살펴봄



대체로 0에서 300정도의

분포를 보임

02 EDA 및 전처리



3. 라벨이 잘못 분류된 sample 수정

'한식'이 '서양식'으로 잘못 표기됨

id	Img_path	Overview	Cat1	Cat2	Cat3
TRAIN_00000	./image/train/TRAIN_00000.jpg	한국 미식 프로그램에 자주 소개된 매장이다 대표메뉴는 꽃게찜이다. 서울시 성동구에 있는 한식전문점이다.	음식	음식점	서양식
TRAIN_00001	./image/train/TRAIN_0001.jpg	기도 이천시 모가면에 있는 골프장으로 대중제 18홀이다. 회원제로 개장을 …	레포츠	육상레포츠	골프
TRAIN_00003	./image/train/TRAIN_00002.jpg	금오산성숯불갈비는 한우고기만을 전문적으로 취급하고 사용하는 부식 자재 또한 유기농법으로 …	음식	음식점	한식
…	…		…	…	…

잘못 라벨링된 경우가 존재하는 것을 발견하고

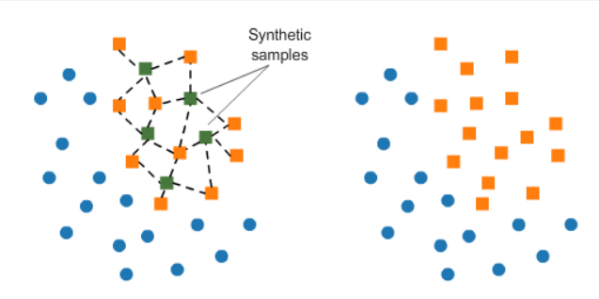
직접 모든 행을 확인하여 라벨을 수정함

02 EDA 및 전처리

4. 데이터 증강 – 기존의 방법

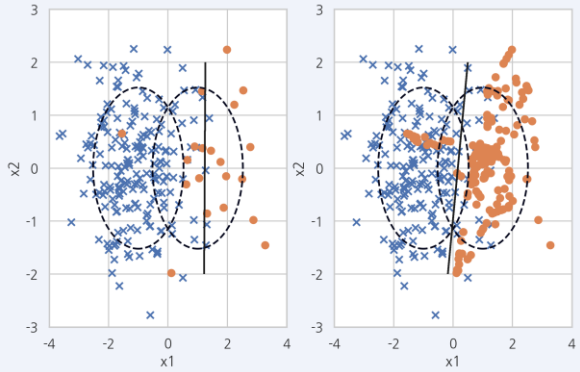
SMOTE

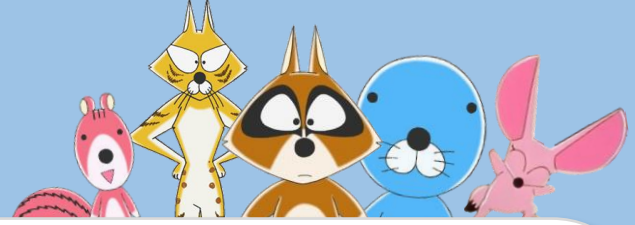
✓ 적은 클래스 데이터들을 KNN을 이용하여 새로운 데이터를 생성하는 오버샘플링 기법



ADASYN

✓ SMOTE과 동일 과정을 진행 후 데이터의 임의의 작은 값을 더해 주며 데이터가 더 분산되게 표현

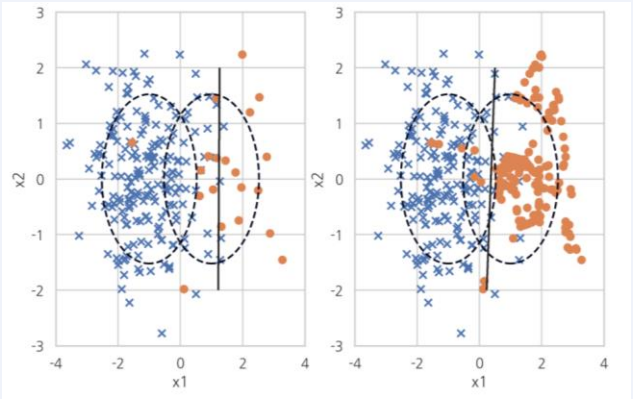




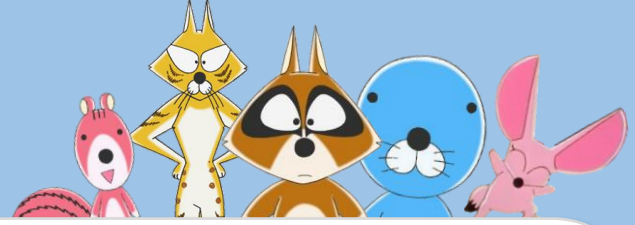
4. 데이터 증강 – 기존의 방법

SMOTETomek

✓ 오버 샘플링인 SMOTE와 언더 샘플링인 Tomek를 융합한 방법



02 EDA 및 전처리



4. 데이터 증강 – 기존의 방법

SMOTETomek

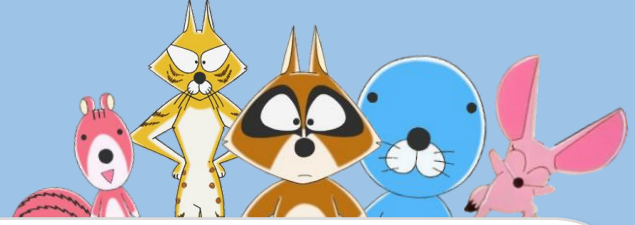
✓ 오버 샘플링인 SMOTE와 언더 샘
플링인 Tomek를 융합한 방법

기존의 증강방식은 주로 정형데이터를 다룸

그러나 관광 데이터는 비정형데이터로 구성되어 있기 때문에

새로운 증강방식을 사용해야 함

02 EDA 및 전처리



4. 데이터 증강

Easy Data Augmentation

현재 보유하고 있는 데이터를 변형하여 증강하는 방법

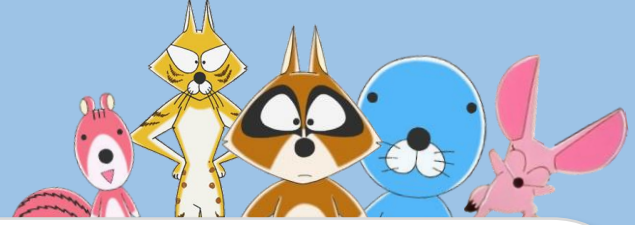
SR Synonym Replacement,
특정 단어를 유의어로 교체

RI Random Insertion,
임의의 단어를 삽입

RS Random Swap,
문장 내 단어 위치를 임의로 바꿈

RD Random Deletion,
문장 내 단어를 임의로 삭제

02 EDA 및 전처리



4. 데이터 증강

Easy Data Augmentation

현재 보유하고 있는 데이터를 변형하여 증강하는 방법

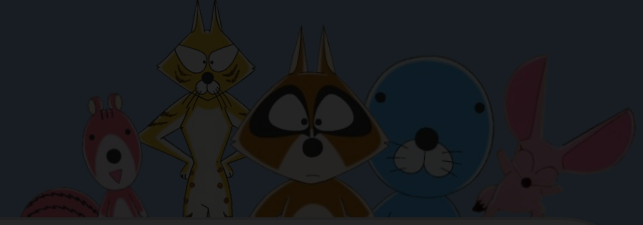
SR Synonym Replacement,
특정 단어를 유의어로 교체

RI Random Insertion,
임의의 단어를 삽입

RS Random Swap,
문장 내 단어 위치를 임의로 바꿈

RD Random Deletion,
문장 내 단어를 임의로 삭제

SR, RI 방법을 적용할 경우 문장의 의미가 변형될 가능성이 높기 때문에,
RS, RD 방법을 통해서만 증강을 하기로 결정



4. 데이터 증강

Easy Data Augmentation

현재 보유하고 있는 데이터를 변형하여 증강하는 방법

데이터 역전 현상

고유값 개수 기준으로 증강할 경우 특정 값보다
조금 큰 데이터는 증강되고, 특정 값보다 조금 작은
데이터는 증강되지 않는 문제 발생

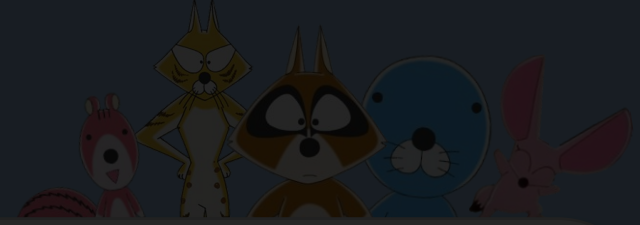
SR Synonym Replacement,
특정 단어를 유의어로 교체

RI Random Insertion,
임의의 단어를 삽입

RS Random Swap,
문장 내 단어 위치를 임의로 바꿈

RD Random Deletion,
문장 내 단어를 임의로 삭제

SR, RI 방법을 적용할 경우 문장의 의미가 변형될 가능성이 높기 때문에,
RS, RD 방법을 통해서만 증강을 하기로 결정



4. 데이터 증강

Easy Data Augmentation

현재 보유하고 있는 데이터를 변형하여 증강하는 방법

데이터 역전 현상

고유값 개수 기준으로 증강할 경우 특정 값보다
조금 큰 데이터는 증강되고, 특정 값보다 조금 작은
데이터는 증강되지 않는 문제 발생

SR Synonym Replacement,
특정 단어를 유의어로 교체

RI Random Insertion,
임의의 단어를 삽입

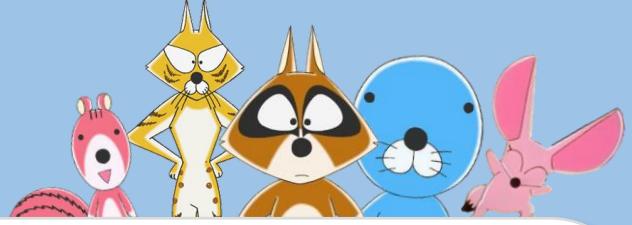
RS Random Swap,
문장 내 단어 위치를 임의로 바꿈

RD Random Deletion,
문장 내 단어를 임의로 삭제

확률적인 방법으로 데이터 증강

SR, RI 방법을 적용할 경우 문장의 의미가 변형될 가능성이 높기 때문에,
RS, RD 방법을 통해서만 증강을 하기로 결정

02 EDA 및 전처리



4. 데이터 증강 - 확률적인 접근

Step 1. $\frac{\text{전체 데이터의 개수}}{\text{해당 소분류(cat3)에 속하는 데이터 개수}}$ 구하기

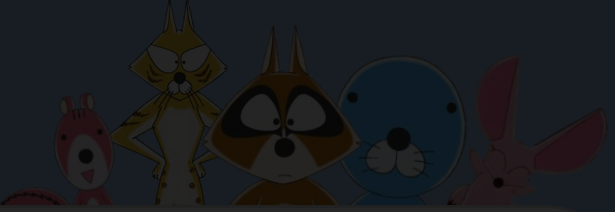
Step 2. 분산을 줄이기 위해 루트를 씌어 줌

Step 3. 해당 값을 0~1 사이의 확률 값으로 만들기 위해 **min-max scailing 수행**

Step 4. 각 관찰 값에 대해 binomial distribution을 통해 0, 1 값 추출

Step 5. 0이면 데이터 증강을 하지 않고, 1이면 데이터를 증강

02 EDA 및 전처리



*데이터 증강

확률적인 접근

소분류(cat3)에 속하는 데이터 수가

적을 수록 증강될 확률이 높아지기 때문에

전체 데이터의 분포를 해치지 않고 증강할 수 있다는 장점이 존재함

Step 1. $\frac{\text{전체 데이터의 개수}}{\text{해당 소분류(cat3)개수}}$ 구하기

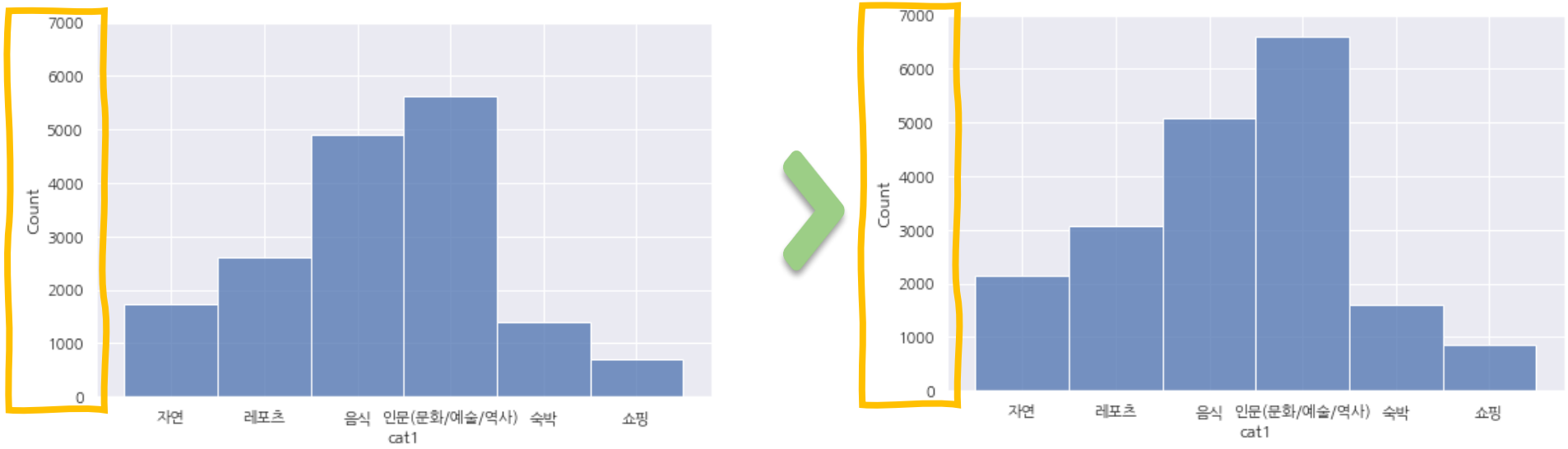
Step 2. 해당 값을 0~1 사이의 값으로 만들기 위해 min-max scailing

Step 3. 각 관찰값에 대해 binomial distribution을 통해 0,1 값 추출

Step 4. 0이면 데이터 증강을 하지 않고, 1이면 데이터를 증강

02 EDA 및 전처리

5. DATA 증강 후

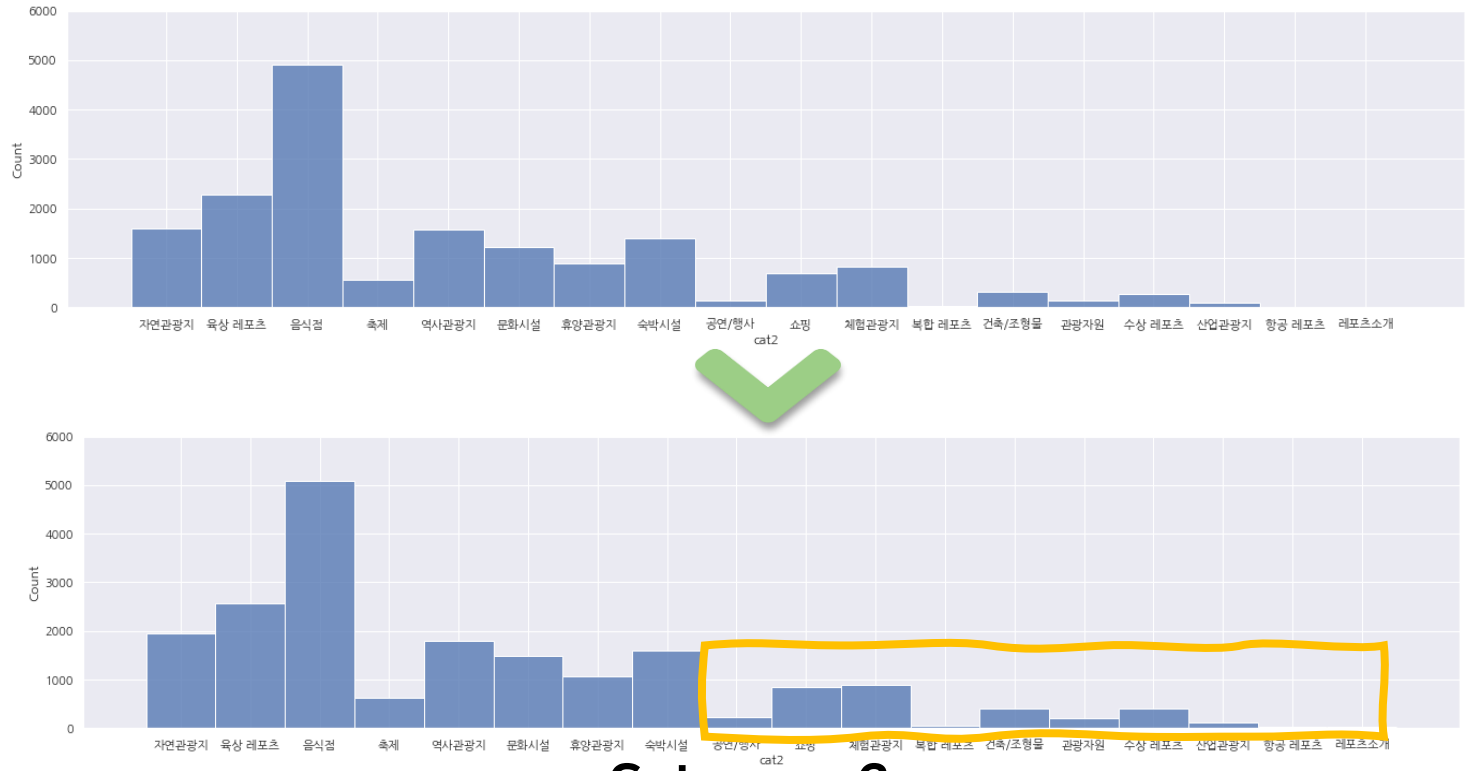


Category 1

데이터 증강 전과 비교해, 데이터가 3000개 정도 증가 했으며 불균형 문제가 다소 완화됨

02 EDA 및 전처리

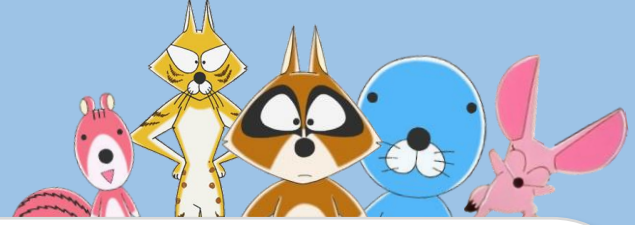
5. DATA 증강 후



Category 2

거의 없던 데이터가 증강되었음을 확인할 수 있음

02 EDA 및 전처리



5. DATA 증강 후

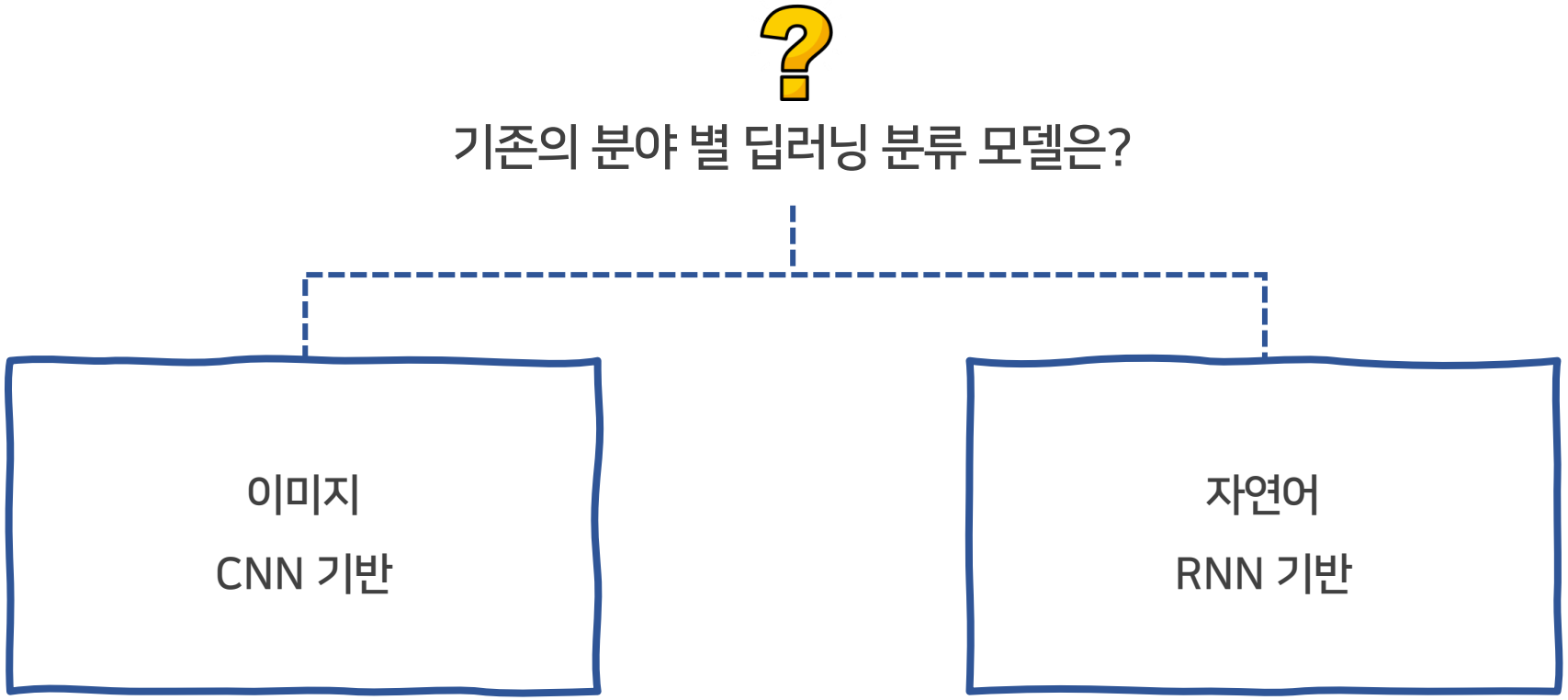
Category 3

데이터 수가 적은 카테고리를 시각화한 결과, 상당히 증강된 것을 알 수 있음

03 Transformer 모델

03 Transformer 모델

1. Transformer란?



03 Transformer 모델

1. 모델 선정

서기 2017년..

기존의 분야별 딥러닝 분류 모델은?

Attention is all you need라는

딥러닝 역사상 가장 획기적인 논문 중 하나가 등장

이미지

CNN 기반

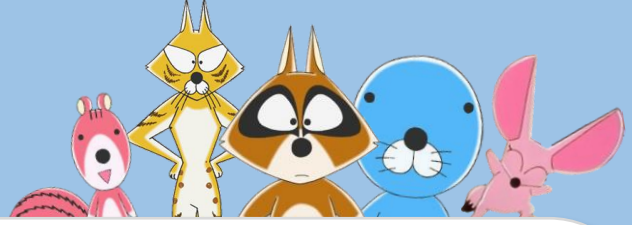
자연어

RNN 기반

Transformer가 이 세상에

자신의 존재를 드러냈다.

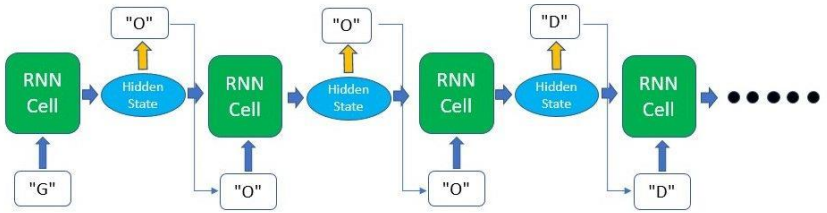
03 Transformer 모델



1. Transformer 모델의 등장 배경

기존 RNN

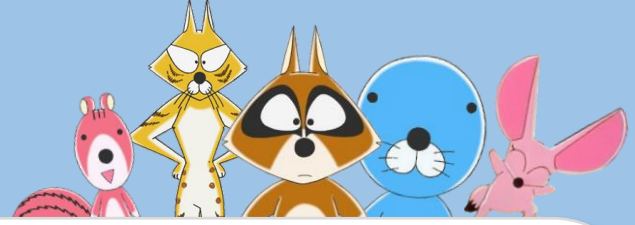
- ✓ 기존 cell의 출력 값이 다음 cell의 입력 값이 되는 직렬 연결
- ✓ 직렬 연결로 인한 많은 연산량과 낮은 연산 속도



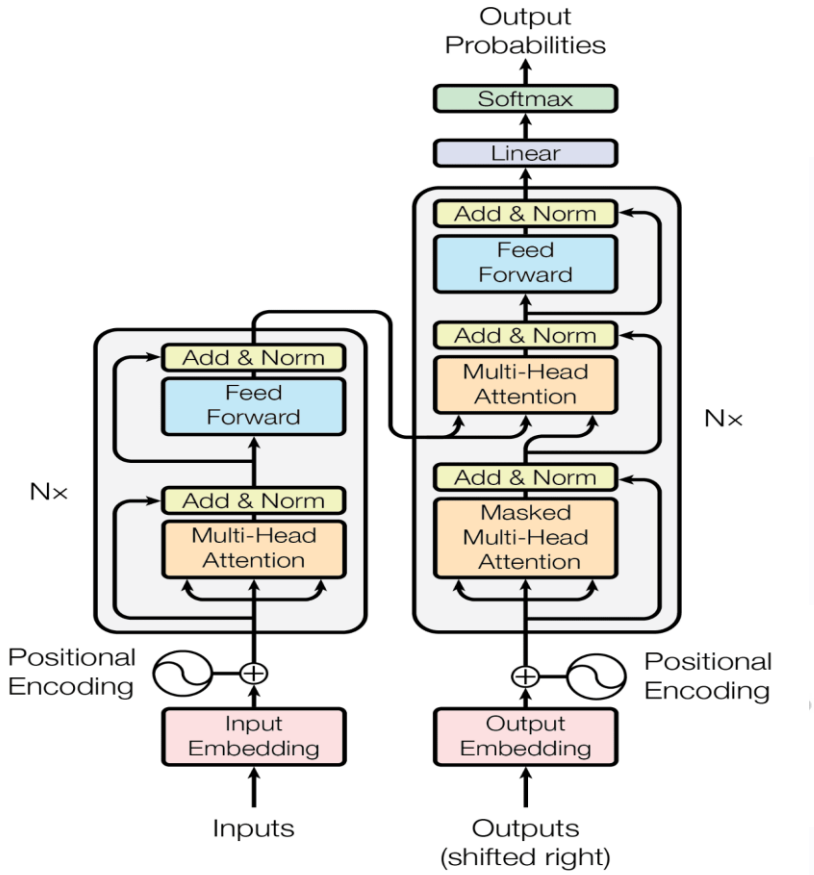
Transformer

- ✓ 한번에 데이터를 입력하면서도 Encoder-Decoder의 형태 유지
- ✓ 또한 Self-Attention 기법을 활용하여 기존 RNN 모델에서 성능을 끌어올림

03 Transformer 모델



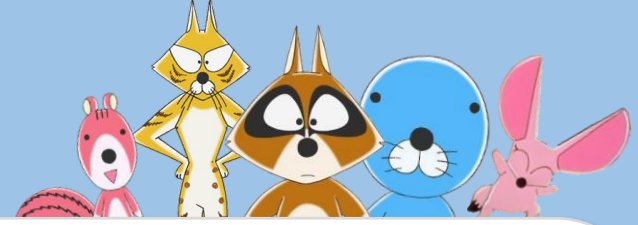
1. Transformer 모델의 등장 배경



Transformer

- ✓ 한번에 데이터를 입력하면서도 Encoder-Decoder의 형태 유지
- ✓ 또한 Self-Attention 기법을 활용하여 기존 RNN 모델에서 성능을 끌어올림

03 Transformer 모델

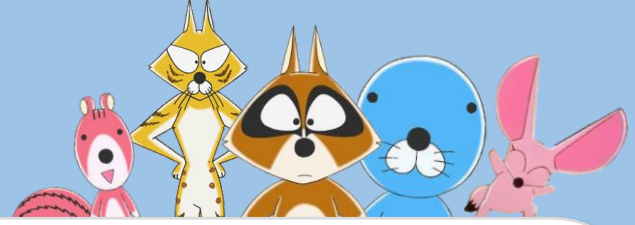


1. Transformer 모델의 등장 배경

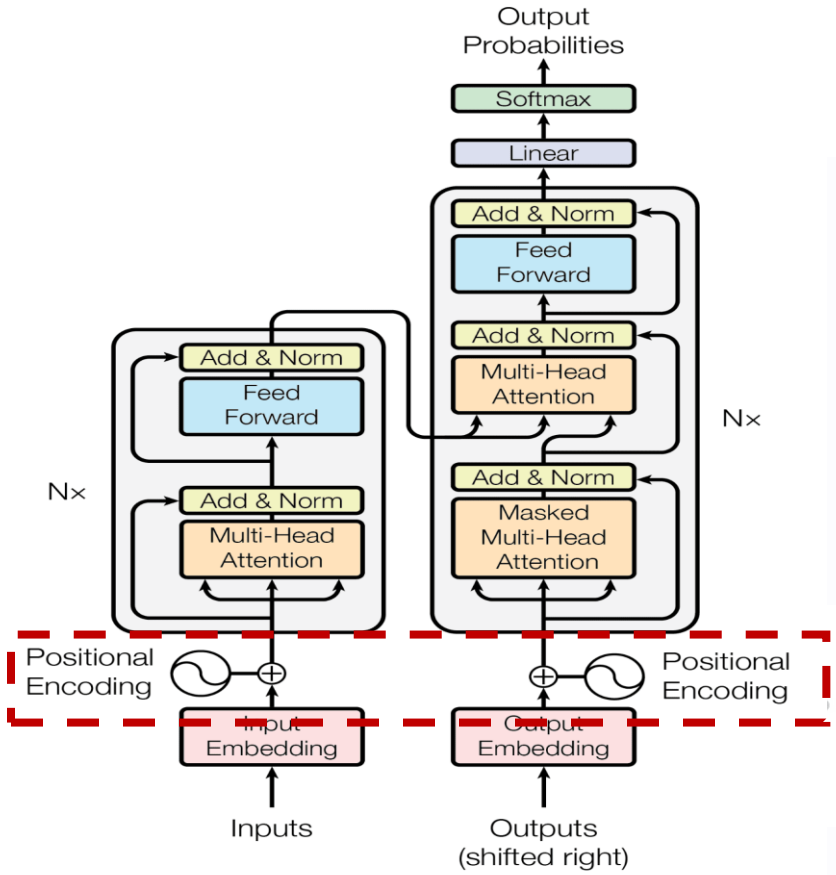
Transformer에 대한 개념을

간단하게 알아보자!

03 Transformer 모델



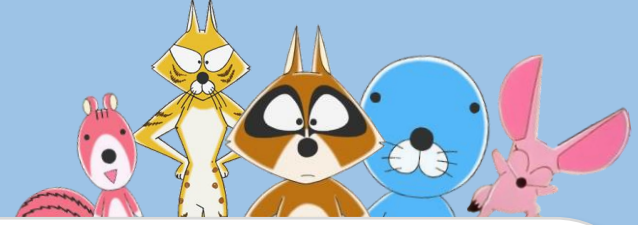
2. Transformer 모델의 구조



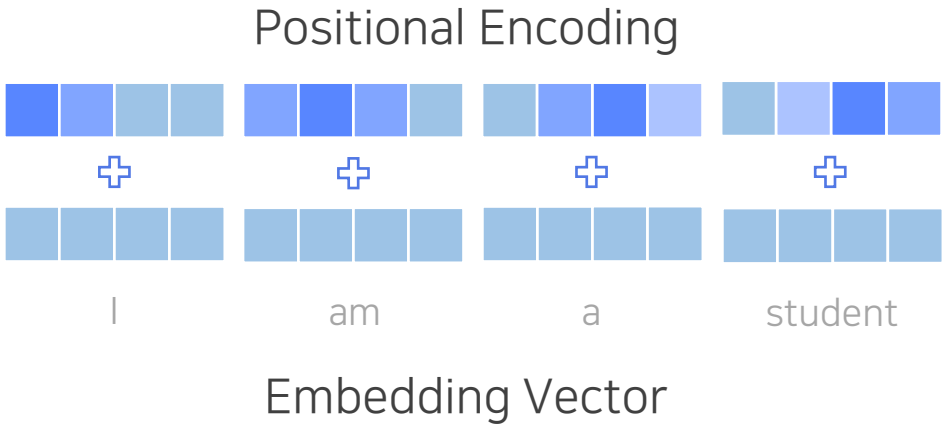
Positional Encoding

- ✓ Transformer 모델은 데이터를 순서대로 읽어오지 않기 때문에 데이터에 상대적/절대적인 위치 정보가 필요함
- ✓ 이런 위치 정보를 input값에 따로 부여하는 것이 Positional Encoding

03 Transformer 모델



2. Transformer 모델의 구조

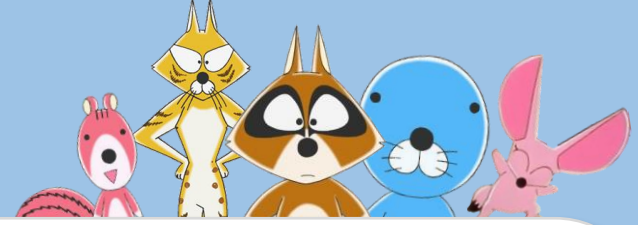


Positional Encoding

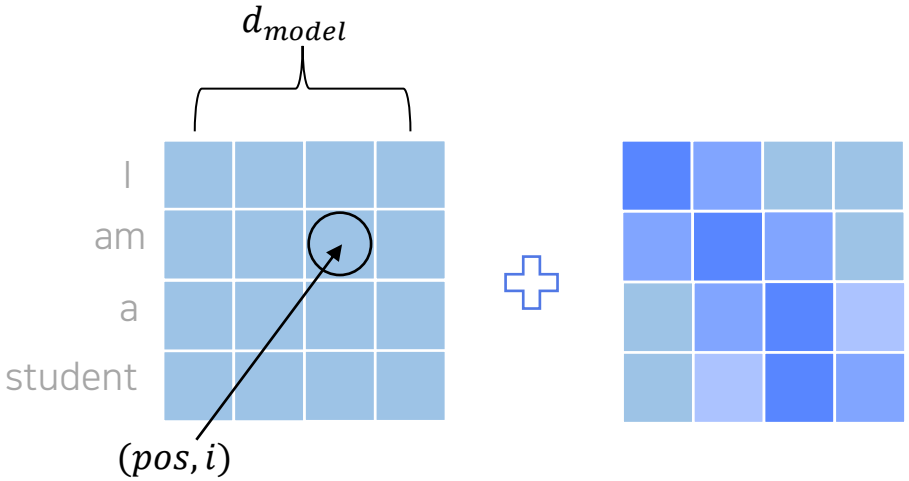
- ✓ 왼쪽 그림의 색은 값을 의미
- ✓ 실제로는 어떤 기준에 의해서 값이 매겨질까?

sin함수와 cos함수 사용

03 Transformer 모델



2. Transformer 모델의 구조



Positional Encoding

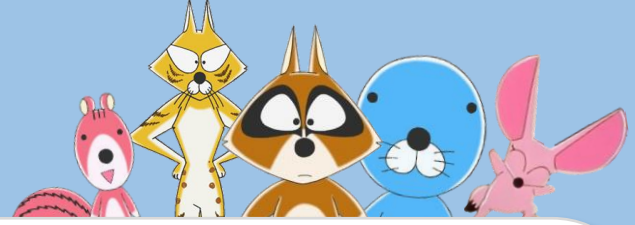
- ✓ 왼쪽 그림의 색은 값을 의미
- ✓ 실제로는 어떤 기준에 의해서 값이 매겨질까?

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

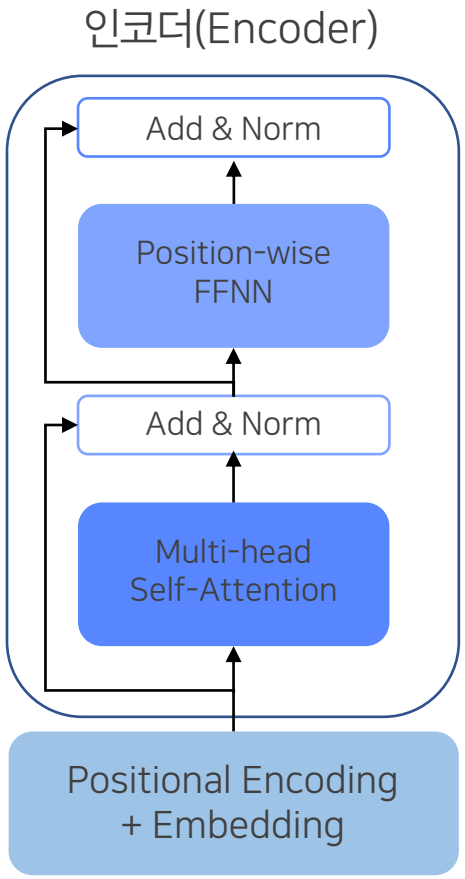
$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

sin함수와 cos함수 사용

03 Transformer 모델



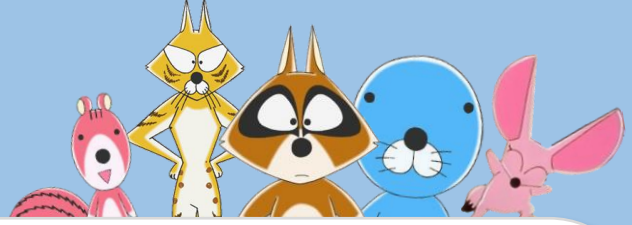
2. Transformer 모델의 구조



Encoder

- ✓ Transformer의 Encoder는 num_layers만큼 인코더 층을 쌓음
- ✓ 하나의 인코더 층은 Multi-head Self-Attention 층과 Position-wise FFNN으로 구성

03 Transformer 모델



2. Transformer 모델의 구조 (Multi-head Self-Attention)

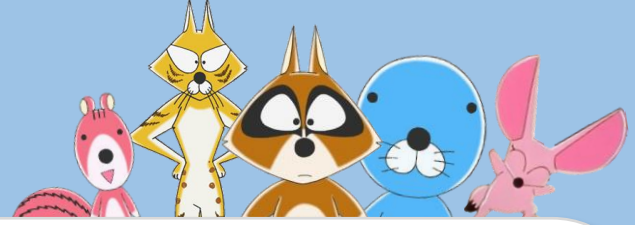
Attention

- ✓ Query와 Key의 유사도를 바탕으로 각 Key에 대응하는 Value를 가중치로 적용하여 Attention Value를 얻음
- ✓ 딥러닝 클린업 3주차 참고!

Self-Attention

- ✓ 자기 자신에게 Attention을 수행하는 것
- ✓ Sequential한 데이터를 처리할 때, CNN이나 RNN보다 복잡도를 훨씬 더 줄일 수 있음

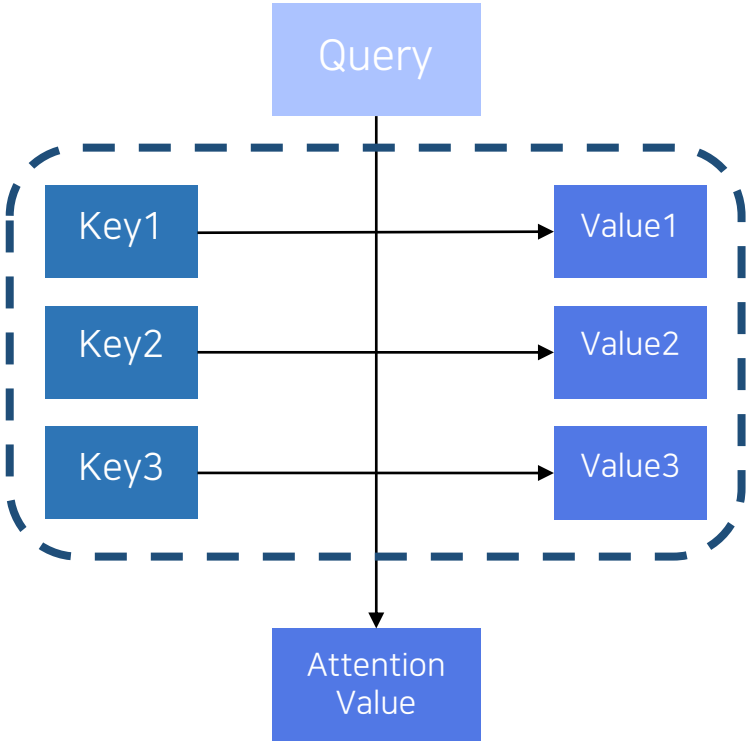
03 Transformer 모델



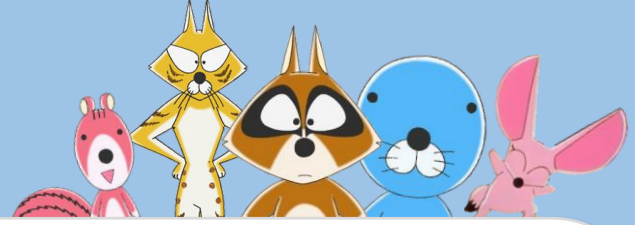
2. Transformer 모델의 구조 (Multi-head Self-Attention)

Attention

- ✓ Query와 Key의 유사도를 바탕으로 각 Key에 대응하는 Value를 가중치로 적용하여 Attention Value를 얻음
- ✓ 딥러닝 클린업 3주차 참고!



03 Transformer 모델

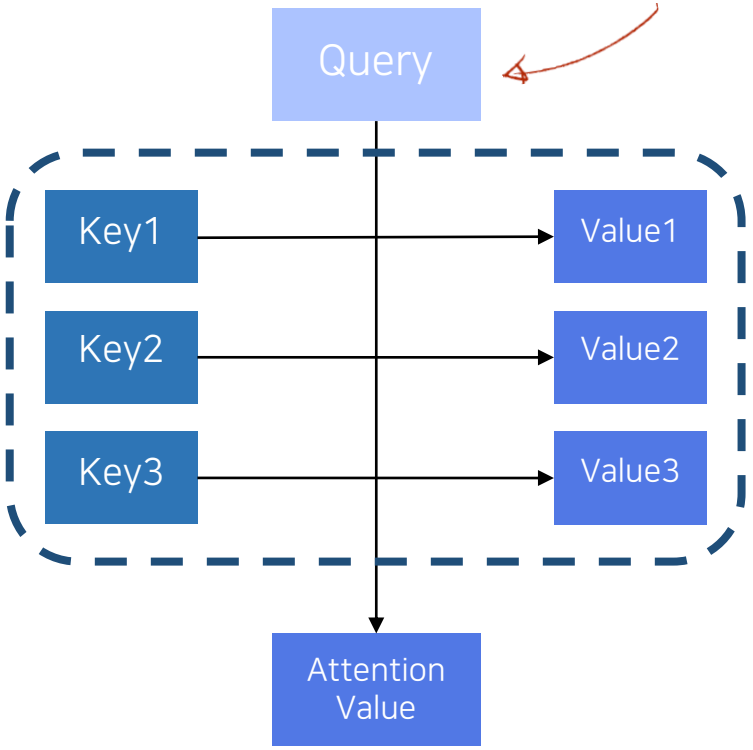


2. Transformer 모델의 구조 (Multi-head Self-Attention)

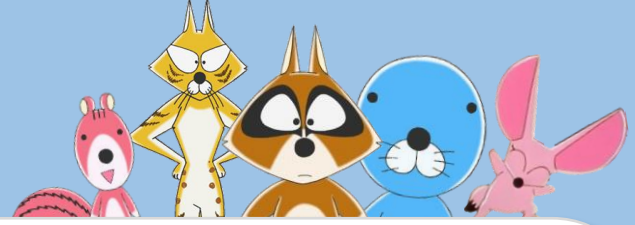
Attention

- ✓ Query와 Key의 유사도를 바탕으로 각 Key에 대응하는 Value를 가중치로 적용하여 Attention Value를 얻음
- ✓ 딥러닝 클린업 3주차 참고!

Self-Attention을 적용하기 위해선
Query, Key, Value를 구해야 함



03 Transformer 모델



2. Transformer 모델의 구조 (Multi-head Self-Attention)

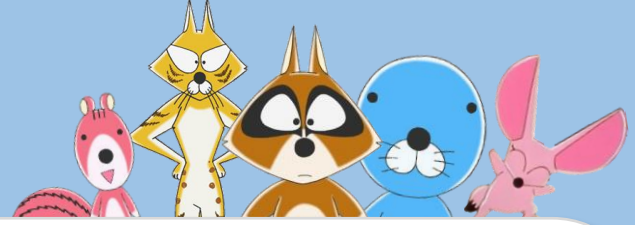
Self-Attention

- ✓ Q=Query: 입력 문장의 모든 단어 벡터
- ✓ K=Key: 입력 문장의 모든 단어 벡터
- ✓ V=Value: 입력 문장의 모든 단어 벡터

즉, 입력 데이터를 바탕으로 Q, K, V를 찾음

Self-Attention을 적용하기 위해선
Query, Key, Value를 구해야 함

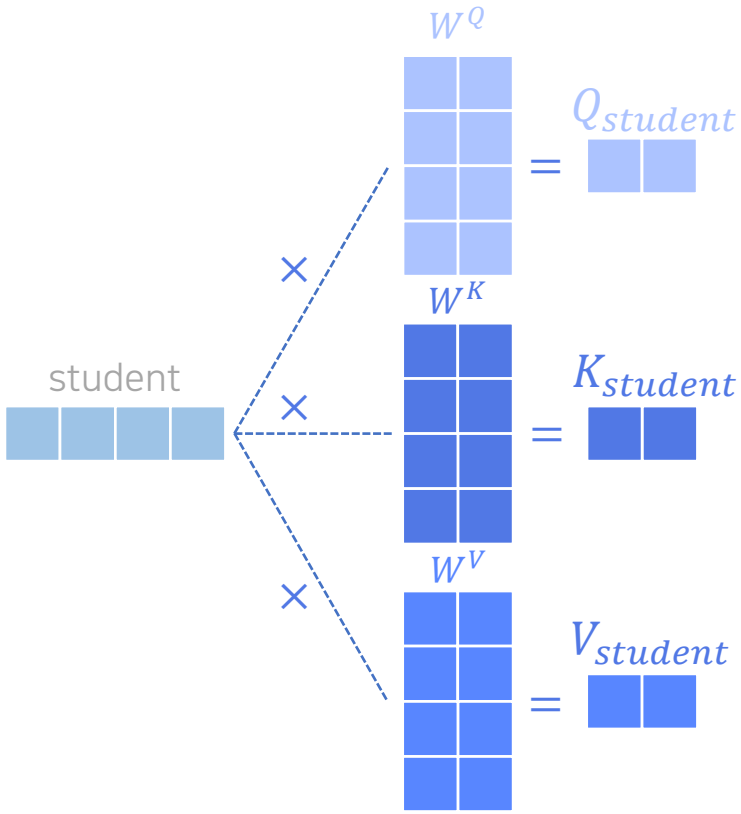
03 Transformer 모델



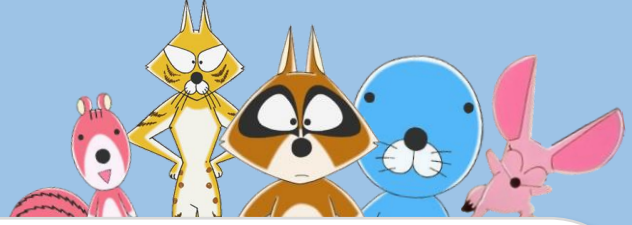
2. Transformer 모델의 구조 (Multi-head Self-Attention)

Self-Attention

- ✓ Q, K, V는 각 단어 벡터에 가중치 W^Q, W^k, W^v를 곱하여 얻어낸다.
- ✓ 이때 가중치는 각 단어행렬 벡터의 차원보다 항상 작고, num_head에 의해 결정된다.
- ✓ $\dim(W^Q) = d_{model} \times (\frac{d_{model}}{num\_head})$



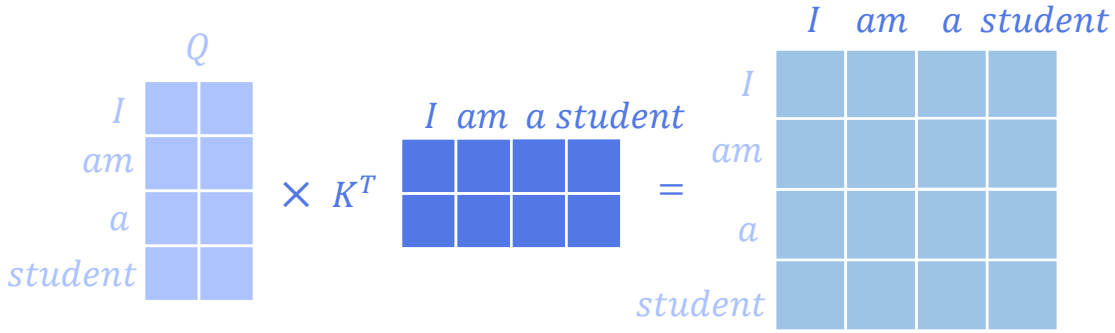
03 Transformer 모델

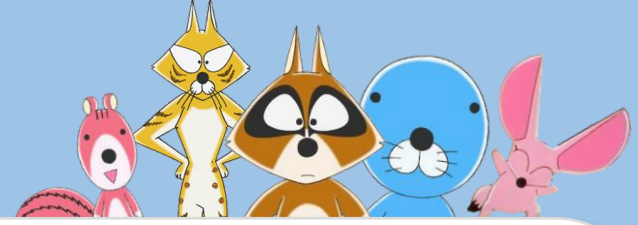


2. Transformer 모델의 구조 (Multi-head Self-Attention)

Self-Attention

- ✓ 각 단어들로 만든 Q, K를 결합하여 matrix로 만들어 Attention Value 행렬을 얻음
- ✓ $Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$

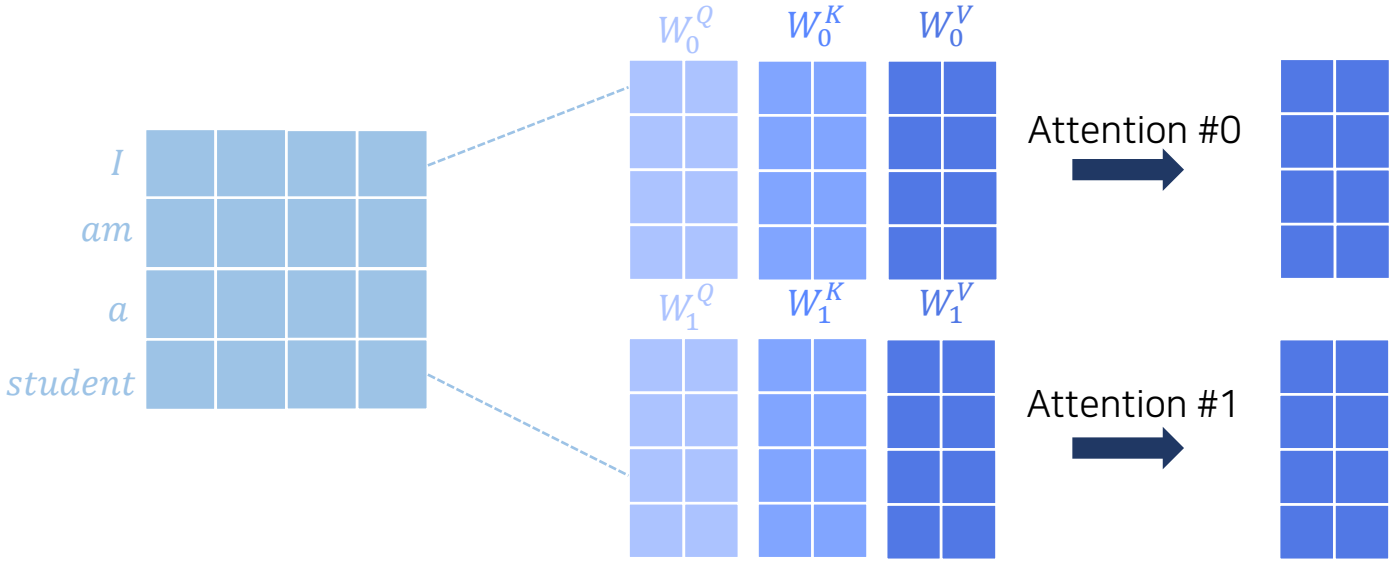


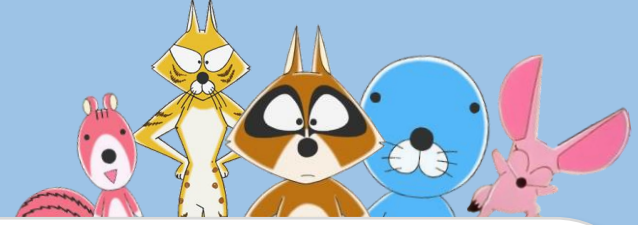


2. Transformer 모델의 구조 (Multi-head Self-Attention)

Multi-Head Self-Attention

Self-Attention을 d_{model}차원 K, Q, V 벡터로 한번 Attention하는 것이
아닌 h번 수행하여 Attention Value를 얻어내는 것

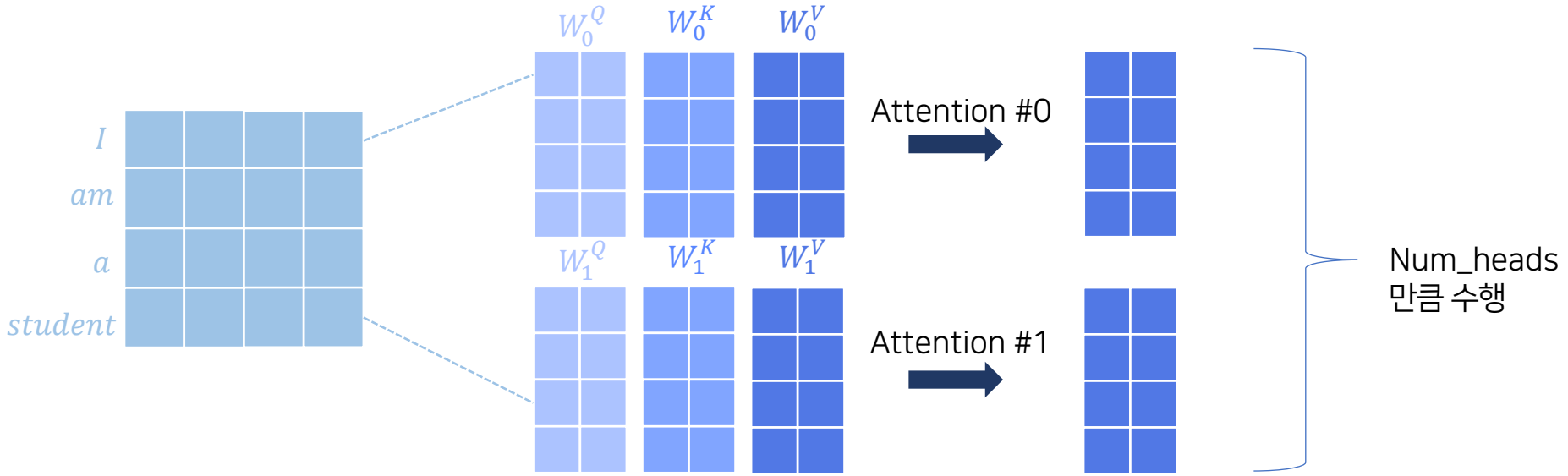




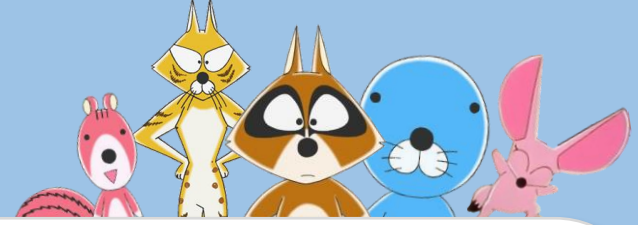
2. Transformer 모델의 구조 (Multi-head Self-Attention)

Multi-Head Self-Attention

Self-Attention을 d_{model}차원 K, Q, V 벡터로 한번 Attention하는 것이
아닌 h번 수행하여 Attention Value를 얻어내는 것



03 Transformer 모델



2. Transformer 모델의 구조 (Position-wise FFNN)

Position-wise FFNN

Position-Wise FFNN은 Encoder와 Decoder에서 모두 사용되는 sublayer

Multi-Head Attention 층을 통과한 벡터들이 지나가는 sublayer

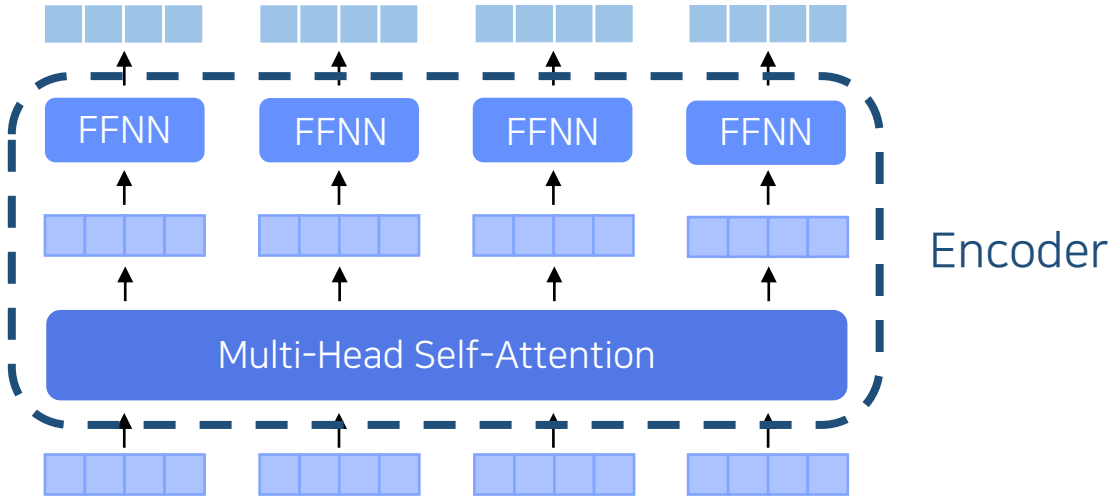
FFNN

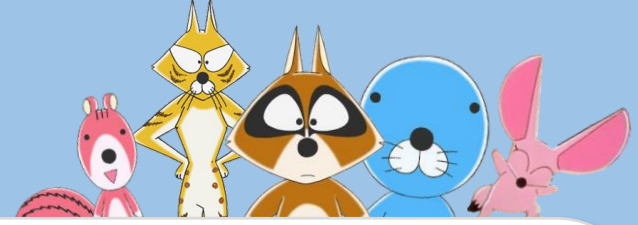
x

$F_1 = xW_1 + b_1$

$F_2 = \max(0, F_1)$

$F_3 = F_2W_2 + b_2$





2. Transformer 모델의 구조 (Position-wise FFNN)

Position-wise FFNN

Position-Wise FFNN을 지난 각 출력 벡터들은

다음 Encoder의 입력 벡터로 들어감

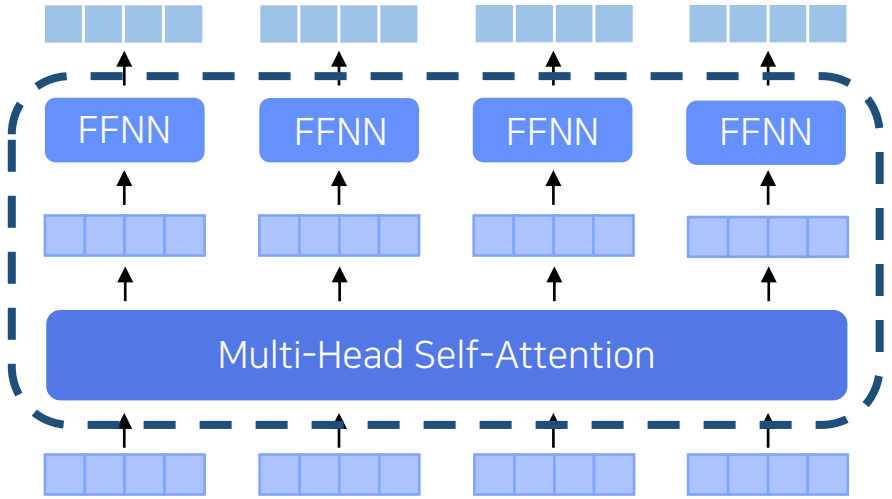
FFNN

x

$F_1 = xW_1 + b_1$

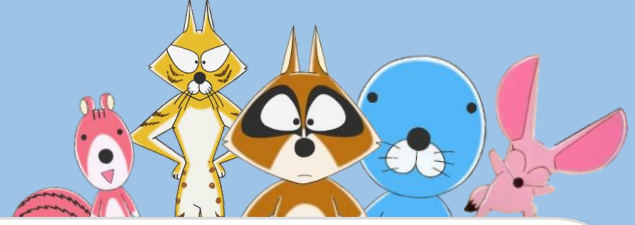
$F_2 = \max(0, F_1)$

$F_3 = F_2W_2 + b_2$

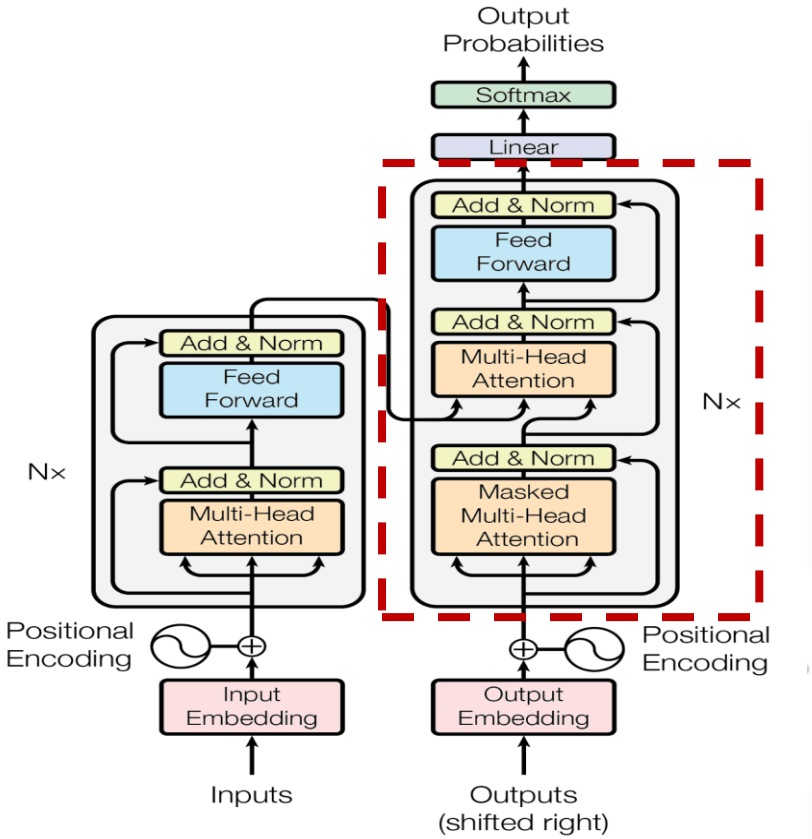


Encoder

03 Transformer 모델



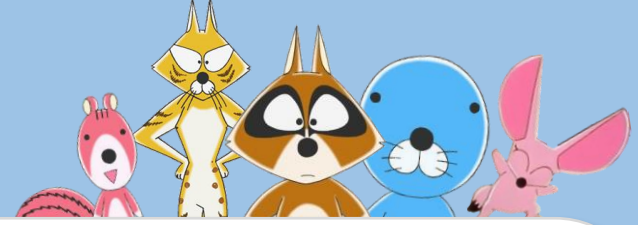
2. Transformer 모델의 구조 (Decoder)



Decoder

✓ Masked Multi-Head Attention과 Encoder-Decoder attention을 진행함

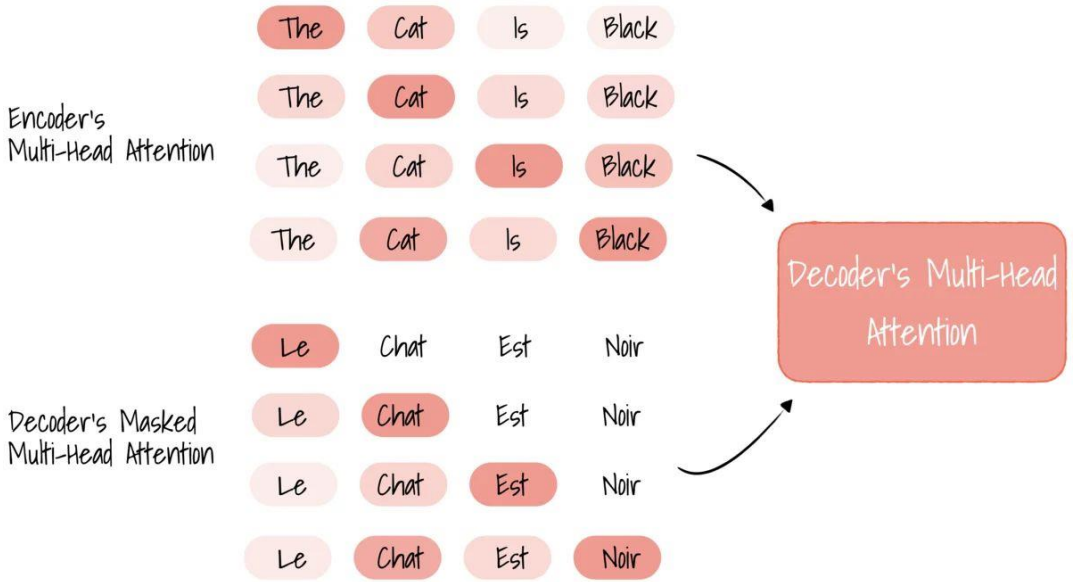
03 Transformer 모델



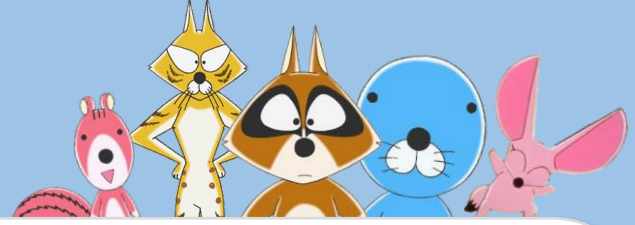
2. Transformer 모델의 구조 (Masked Multi-Head Attention)

Masked Multi-Head Attention

- ✓ 자신 결과물의 앞 단에 위치한 토큰들만의 attention score를 참고
- ✓ 오른쪽 그림의 'Le'를 attention할 때 'chat', 'Est', 'Noir'을 사용하지 않음



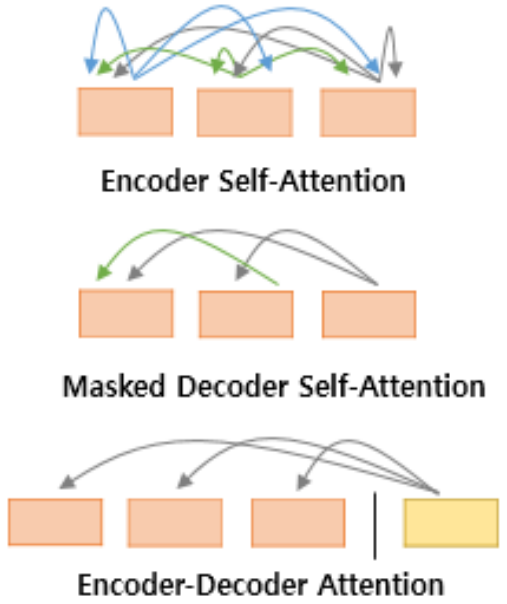
03 Transformer 모델



2. Transformer 모델의 구조 (Encoder-Decoder attention)

Encoder-Decoder attention

- ✓ Encoder Output의 Key, Value로 attention을 진행
- ✓ 이 과정을 통해 input embedding과 out embedding 사이에 위치적 관계가 만들어 짐

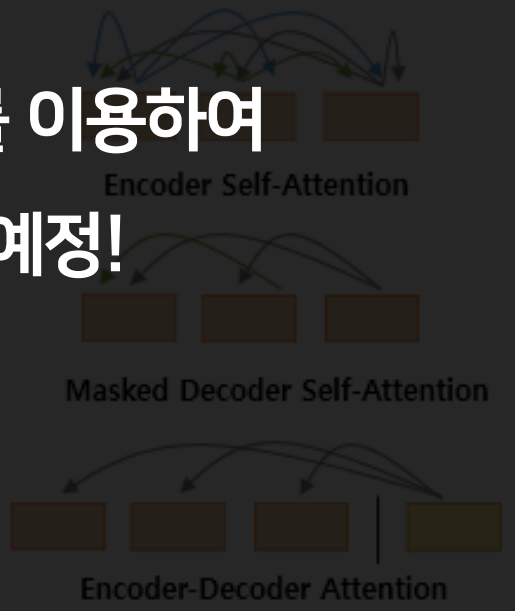


3. Transfomer 모델의 구조 (Encoder-Decoder attention)

Encoder-Decoder attention

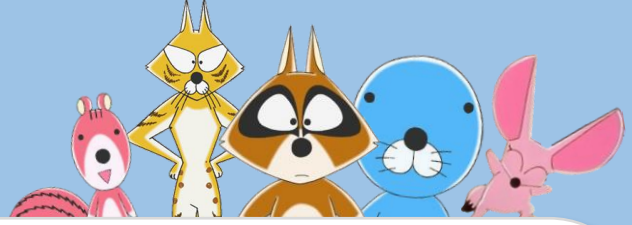
- ✓ Encoder Output으로 Key, Value로 attention을 진행
- ✓ 이 과정을 통해 input embedding과 out embedding 사이에 위치적 관계가 만들어짐

이러한 Transformer 구조를 이용하여 최종적인 모델을 구성할 예정!



04 카테고리 분류 모델

04 카테고리 분류 모델



1. ViT

Vision Transformer

NLP에서 사용되던 transformer 모델을 이미지 분류 문제에 맞게 변형하여

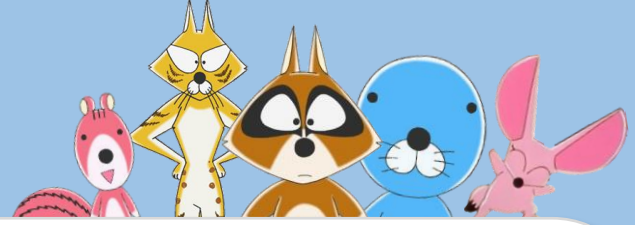
CNN과 비슷하거나 더 나은 성능 기록

이것도 구글에서 발표...

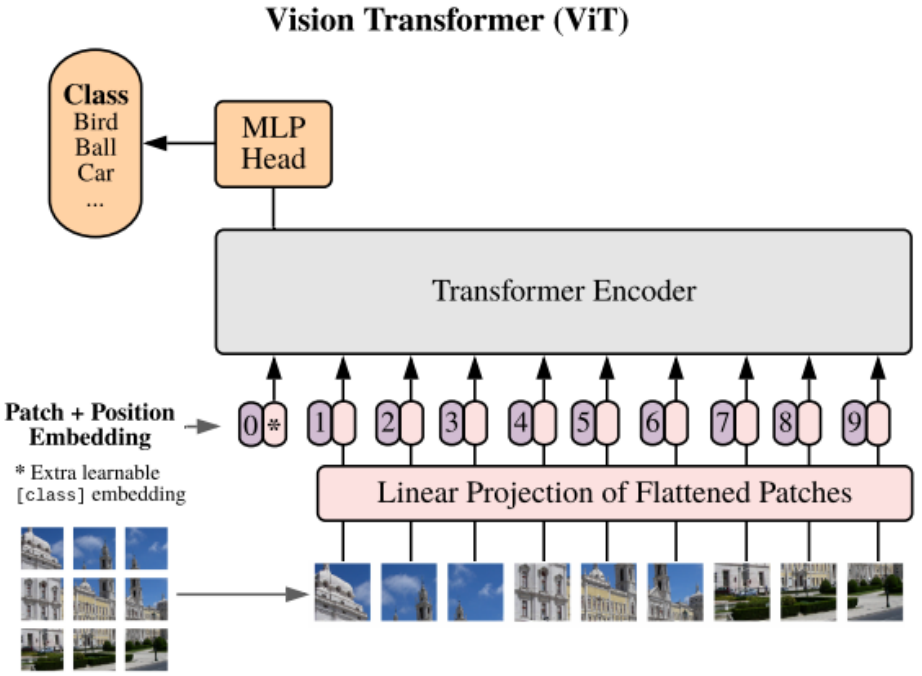
사랑해요 구글 날 가져요 구글

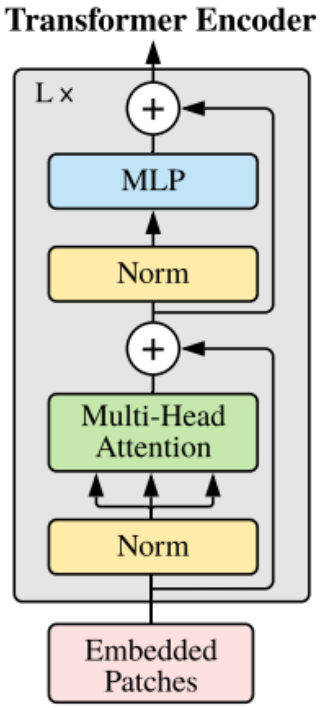
제발 절 가져주세요 구글..

04 카테고리 분류 모델



1. ViT





이미지를 패치로 분할

▼

Positional embedding vector와

CLS token을 concat

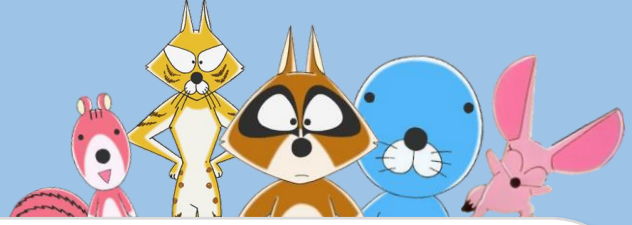
▼

위 1차원 embedding vector를 input

▼

인코더, FC layer를 거쳐 최종적으로 분류

04 카테고리 분류 모델



2. BERT

BERT

Bidirectional Encoder Representations from Transformers

NLP 분야에서 기록적인 성능을 보이는 transformer 기반 모델

레이블이 없는 방대한 양의 데이터를 사전 학습한 모델

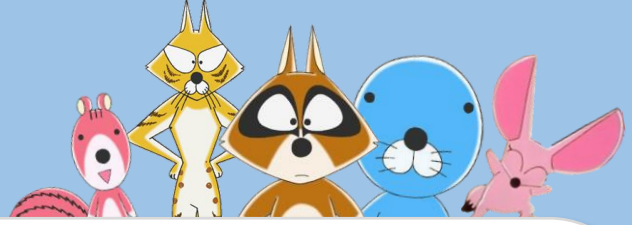
사전 학습한 모델을 레이블이 있는 데이터에 대해 파인 튜닝하여 사용

기존의 모델처럼 문장을 왼쪽에서 오른쪽으로만 읽어 문맥을 파악하지 않고,

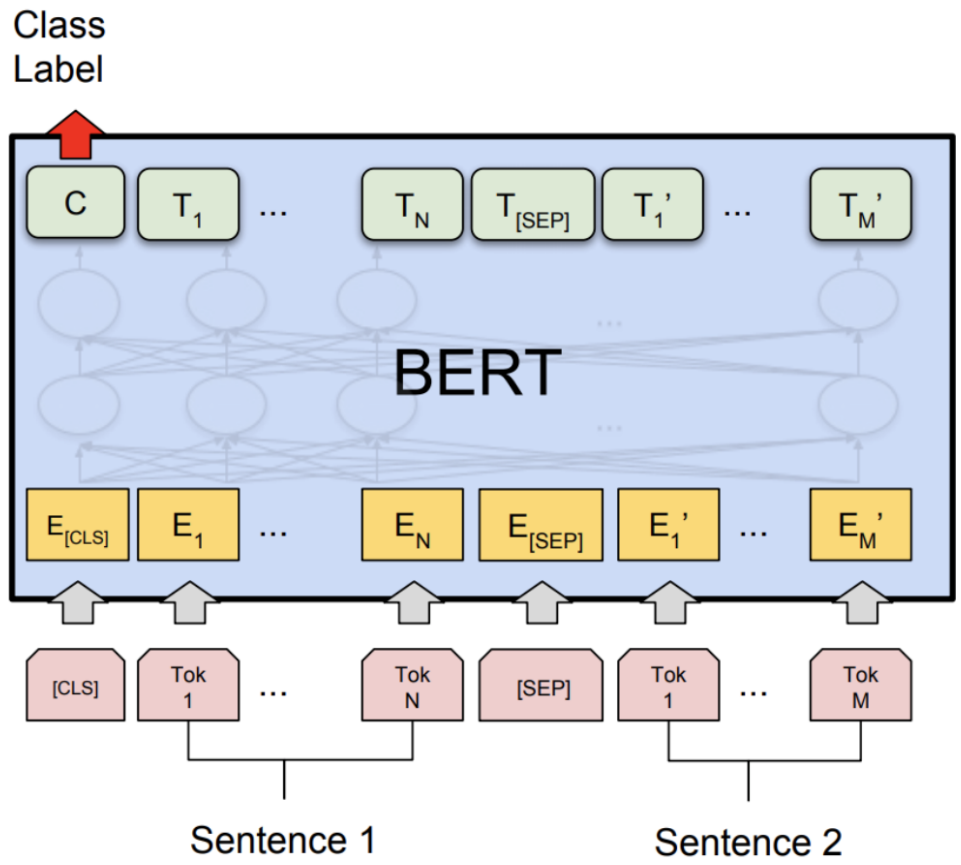
양방향에서 전체 맥락을 파악하는 것이 특징

→ 의미 파악 성능 향상

04 카테고리 분류 모델

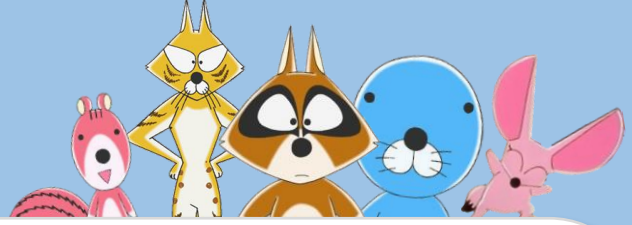


2. BERT



- ✓ Self attention layer를 여러 개 사용하여 의미 관계 추출
- ✓ 인코더-디코더 구조의 인코더만 사용
- ✓ Token, segment, position embedding을 사용하여 문장 표현

04 카테고리 분류 모델



2. BERT

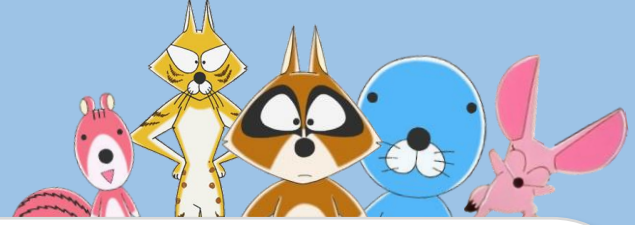
- RoBERTa

- BERT 기반 모델
- BERT가 under fit 되어있다고 판단하여 추가적인 튜닝 진행
- BERT보다 큰 batch size로 더 많은 데이터를 학습
- Dynamic masking 적용
- BERT보다 더 긴 sequence를 입력으로 사용

- KoBERT

- BERT 모델은 영단어로 사전 학습
- BERT 모델을 한국어 자연어 처리에도 사용 가능하도록
  한국어 데이터를 추가로 학습시킨 것

04 카테고리 분류 모델



3. ELECTRA

ELECTRA

Efficiently Learning an Encoder that Classifies Token Replacements Accurately

학습의 효율성 (computational cost)에 주목한 모델

기존의 마스킹 방식이 아닌 RTD (Replaced Token Detection) pre-training task를 제안

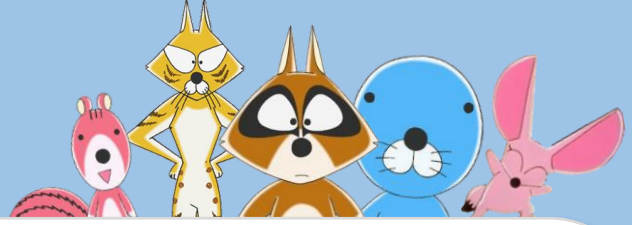
모델 크기, 데이터, 컴퓨팅 리소스가 동일한 조건에서 BERT의 성능 능가

기존의 마스킹 방식은 마스킹 된 토큰을 원본 토큰으로 복원하는 작업

RTD는 generator가 실제 토큰을 가짜 토큰으로 치환하고

해당 토큰의 진위 여부를 discriminator가 맞히는 이진 분류 문제

04 카테고리 분류 모델

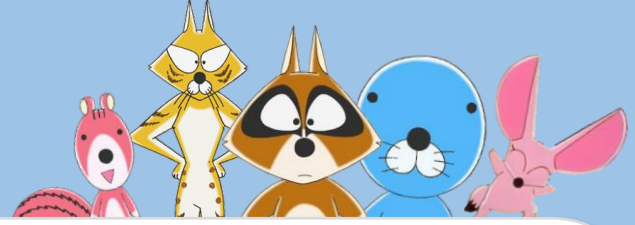


3. ELECTRA

- KcELECTRA

- ELECTRA 기반 모델
- 구어체, 신조어 등 노이즈가 많은 정제되지 않은 텍스트 데이터에 사용하기 위해 네이버 뉴스의 댓글을 수집해 사전 학습한 모델
- 한글, 영어, 특수문자, 이모지까지 학습 대상에 포함

04 카테고리 분류 모델



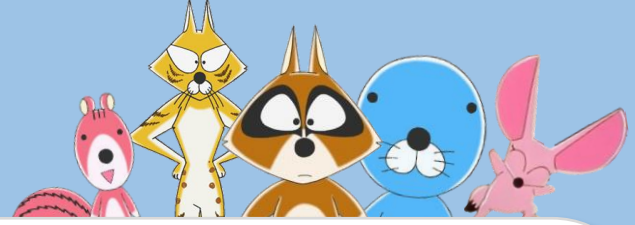
4. Baseline model

Baseline model : ViT + RoBERTa 로 구성된 multi modal

```
tokenizer = AutoTokenizer.from_pretrained("klue/roberta-large")
feature_extractor = ViTFeatureExtractor.from_pretrained('google/vit-large-patch32-384')
train_data_loader = create_data_loader(train, tokenizer, feature_extractor, 256, 16, shuffle_=True)
valid_data_loader = create_data_loader(valid, tokenizer, feature_extractor, 256, 16)


EPOCHS = 30
model = TourClassifier(n_classes1 = 6, n_classes2 = 18, n_classes3 = 128, text_model_name = "klue/roberta-large",
                       image_model_name = "google/vit-large-patch32-384").to(device)
optimizer = optim.AdamW(model.parameters(), lr= 3e-5)
total_steps = len(train_data_loader) * EPOCHS
scheduler = get_cosine_schedule_with_warmup(
optimizer,
num_warmup_steps=int(total_steps*0.1),
num_training_steps=total_steps
)
loss_fn = nn.CrossEntropyLoss().to(device)
```

04 카테고리 분류 모델



4. Baseline model

Baseline model : ViT + RoBERTa 로 구성된 multi modal

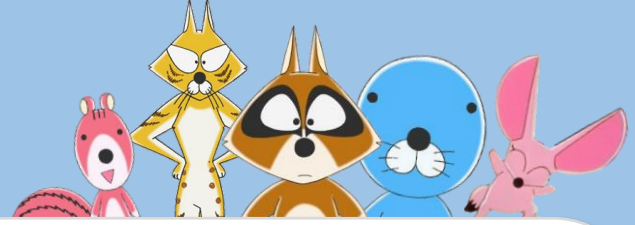
텍스트 데이터 처리를 담당하는
RoBERTa (BERT 기반 모델)

사전학습 모델 불러오기

```
tokenizer = AutoTokenizer.from_pretrained("klue/roberta-large")
feature_extractor = ViTFeatureExtractor.from_pretrained('google/vit-large-patch32-384')
train_data_loader = create_data_loader(train, tokenizer, feature_extractor, 256, 16, shuffle_=True)
valid_data_loader = create_data_loader(valid, tokenizer, feature_extractor, 256, 16)


EPOCHS = 30
model = TourClassifier(n_classes1 = 6, n_classes2 = 18, n_classes3 = 128, text_model_name = "klue/roberta-large",
                       image_model_name = "google/vit-large-patch32-384").to(device)
optimizer = optim.AdamW(model.parameters(), lr= 3e-5)
total_steps = len(train_data_loader) * EPOCHS
scheduler = get_cosine_schedule_with_warmup(
optimizer,
num_warmup_steps=int(total_steps*0.1),
num_training_steps=total_steps
)
loss_fn = nn.CrossEntropyLoss().to(device)
```

04 카테고리 분류 모델



4. Baseline model

Baseline model : ViT + RoBERTa 로 구성된 multi-modal

이미지 데이터 처리를 담당하는

ViT

사전학습 모델 불러오기

```
tokenizer = AutoTokenizer.from_pretrained("klue/roberta-large")
feature_extractor = ViTFeatureExtractor.from_pretrained('google/vit-large-patch32-384')
train_data_loader = create_data_loader(train, tokenizer, feature_extractor, 256, 16, shuffle_=True)
valid_data_loader = create_data_loader(valid, tokenizer, feature_extractor, 256, 16)


EPOCHS = 30
model = TourClassifier(n_classes1 = 6, n_classes2 = 18, n_classes3 = 128, text_model_name = "klue/roberta-large",
                       image_model_name = "google/vit-large-patch32-384").to(device)
optimizer = optim.AdamW(model.parameters(), lr= 3e-5)
total_steps = len(train_data_loader) * EPOCHS
scheduler = get_cosine_schedule_with_warmup(
optimizer,
num_warmup_steps=int(total_steps*0.1),
num_training_steps=total_steps
)
loss_fn = nn.CrossEntropyLoss().to(device)
```

4. Baseline model

사전 학습? 전이 학습?

이미지 데이터 처리를 담당하는

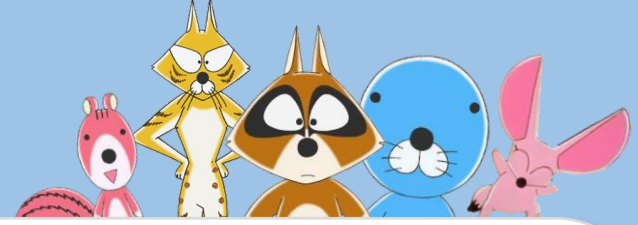
ViT

전이 학습이란 특정 task에 대하여 학습된 딥러닝 모델을
다른 task로 전이하여 학습에 사용하는 개념

사전 학습된 모델을 전이 학습으로 사용하면
적은 양의 데이터에서도 빠르고 우수한 성능을 보임 !

= 효율성 증가

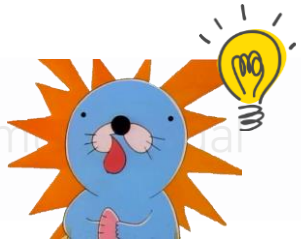
04 카테고리 분류 모델



4. Baseline model

Baseline model : ViT + RoBERTa 크로스 attention multimodal

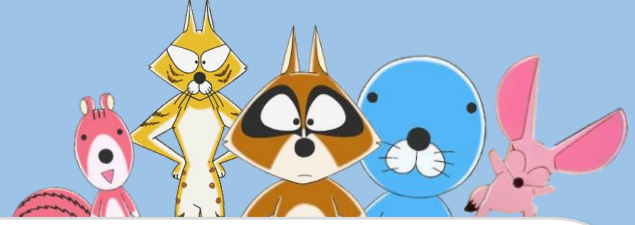
ViT + RoBERTa 멀티모달 클래스



```
tokenizer = AutoTokenizer.from_pretrained("klue/roberta-large")
feature_extractor = ViTFeatureExtractor.from_pretrained('google/vit-large-patch32-384')
train_data_loader = create_data_loader(train, tokenizer, feature_extractor, 256, 16, shuffle_=True)
valid_data_loader = create_data_loader(valid, tokenizer, feature_extractor, 256, 16)


EPOCHS = 30
model = TourClassifier(n_classes1 = 6, n_classes2 = 18, n_classes3 = 128, text_model_name = "klue/roberta-large",
                       image_model_name = "google/vit-large-patch32-384").to(device)
optimizer = optim.AdamW(model.parameters(), lr= 3e-5)
total_steps = len(train_data_loader) * EPOCHS
scheduler = get_cosine_schedule_with_warmup(
optimizer,
num_warmup_steps=int(total_steps*0.1),
num_training_steps=total_steps
)
loss_fn = nn.CrossEntropyLoss().to(device)
```

04 카테고리 분류 모델



4. Baseline model

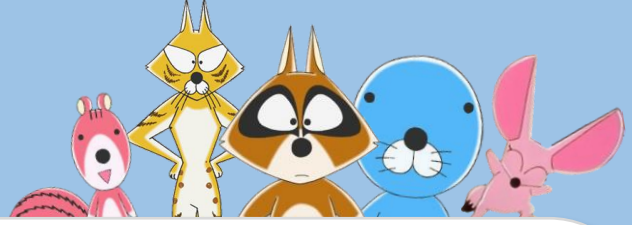
멀티모달의 순전파는 어떻게 이뤄지는 거징??

```python
def forward(self, input_ids, attention_mask,pixel_values):
  text_output = self.text_model(input_ids=input_ids, attention_mask=attention_mask)
  image_output = self.image_model(pixel_values = pixel_values)
  concat_outputs = torch.cat([text_output.last_hidden_state, image_output.last_hidden_state],1)
  #config hidden size 일치해야함
  encoder_layer = nn.TransformerEncoderLayer(d_model=self.text_model.config.hidden_size, nhead=8).to(device)
  transformer_encoder = nn.TransformerEncoder(encoder_layer, num_layers=2).to(device)

  outputs = transformer_encoder(concat_outputs)
  #cls token
  outputs = outputs[:,0]
  output = self.drop(outputs)

  out1 = self.cls(output)
  out2 = self.cls2(output)
  out3 = self.cls3(output)
  return out1,out2,out3
```

04 카테고리 분류 모델



4. Baseline model

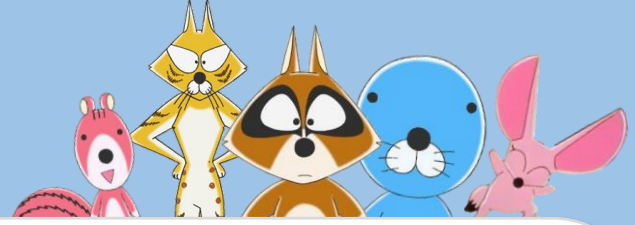
각 모델의 학습 결과를 저장

```
def forward(self, input_ids, attention_mask,pixel_values):
  text_output = self.text_model(input_ids=input_ids, attention_mask=attention_mask)
  image_output = self.image_model(pixel_values = pixel_values)
  concat_outputs = torch.cat([text_output.last_hidden_state, image_output.last_hidden_state],1)
  #config hidden size 일치해야함
  encoder_layer = nn.TransformerEncoderLayer(d_model=self.text_model.config.hidden_size, nhead=8).to(device)
  transformer_encoder = nn.TransformerEncoder(encoder_layer, num_layers=2).to(device)

  outputs = transformer_encoder(concat_outputs)
  #cls token
  outputs = outputs[:,0]
  output = self.drop(outputs)

  out1 = self.cls(output)
  out2 = self.cls2(output)
  out3 = self.cls3(output)
  return out1,out2,out3
```

04 카테고리 분류 모델



4. Baseline model

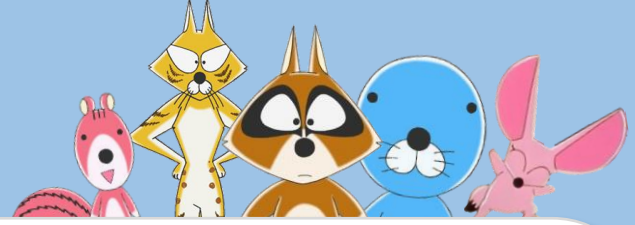
텍스트, 이미지 데이터의 결과값을
하나의 array로 합치기

```
def forward(self, input_ids, attention_mask,pixel_values):
  text_output = self.text_model(input_ids=input_ids, attention_mask=attention_mask)
  image_output = self.image_model(pixel_values =pixel_values)
  concat_outputs = torch.cat([text_output.last_hidden_state, image_output.last_hidden_state],1)
  #config hidden size 일치해야함
  encoder_layer = nn.TransformerEncoderLayer(d_model=self.text_model.config.hidden_size, nhead=8).to(device)
  transformer_encoder = nn.TransformerEncoder(encoder_layer, num_layers=2).to(device)

  outputs = transformer_encoder(concat_outputs)
  #cls token
  outputs = outputs[:,0]
  output = self.drop(outputs)

  out1 = self.cls(output)
  out2 = self.cls2(output)
  out3 = self.cls3(output)
  return out1,out2,out3
```

04 카테고리 분류 모델



4. Baseline model

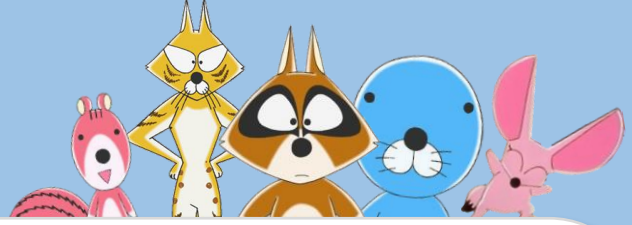
concat_output을 인코더에 입력하여 학습

```
def forward(self, input_ids, attention_mask,pixel_values):
  text_output = self.text_model(input_ids=input_ids, attention_mask=attention_mask)
  image_output = self.image_model(pixel_values = pixel_values)
  concat_outputs = torch.cat([text_output.last_hidden_state, image_output.last_hidden_state],1)
  #config hidden_size 일치해야함
  encoder_layer = nn.TransformerEncoderLayer(d_model=self.text_model.config.hidden_size, nhead=8).to(device)
  transformer_encoder = nn.TransformerEncoder(encoder_layer, num_layers=2).to(device)

  outputs = transformer_encoder(concat_outputs)
  #cls token
  outputs = outputs[:,0]
  output = self.drop(outputs)

  out1 = self.cls(output)
  out2 = self.cls2(output)
  out3 = self.cls3(output)
  return out1,out2,out3
```

04 카테고리 분류 모델



4. Baseline model

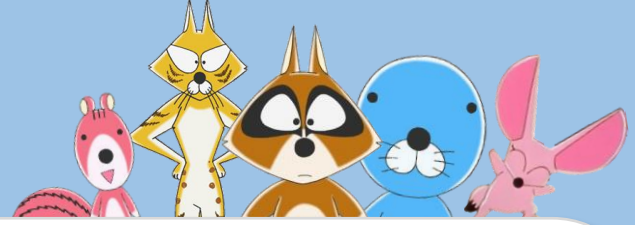
FC layer를 거쳐 cat1, cat2, cat3 최종 분류

```
def forward(self, input_ids, attention_mask,pixel_values):
  text_output = self.text_model(input_ids=input_ids, attention_mask=attention_mask)
  image_output = self.image_model(pixel_values = pixel_values)
  concat_outputs = torch.cat([text_output.last_hidden_state, image_output.last_hidden_state],1)
  #config hidden size 일치해야함
  encoder_layer = nn.TransformerEncoderLayer(d_model=self.text_model.config.hidden_size, nhead=8).to(device)
  transformer_encoder = nn.TransformerEncoder(encoder_layer, num_layers=2).to(device)

  outputs = transformer_encoder(concat_outputs)
  #cls token
  outputs = outputs[:,0]
  output = self.drop(outputs)

  out1 = self.cls(output)
  out2 = self.cls2(output)
  out3 = self.cls3(output)
  return out1,out2,out3
```

04 카테고리 분류 모델



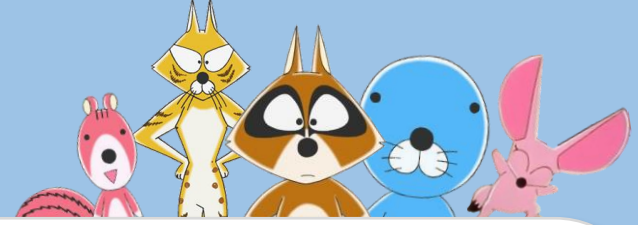
4. Baseline model

```
loss1 = loss_fn(outputs, cats1)
loss2 = loss_fn(outputs2, cats2)
loss3 = loss_fn(outputs3, cats3)

loss = loss1 * 0.1 + loss2 * 0.1 + loss3 * 0.8
```

대중소 분류에 대한 loss를 가중치를 부여하여
전체 모델의 loss 결정

04 카테고리 분류 모델



5. Multimodal

Concat

[Image Processing]

- ViT
- Custom CNN layer

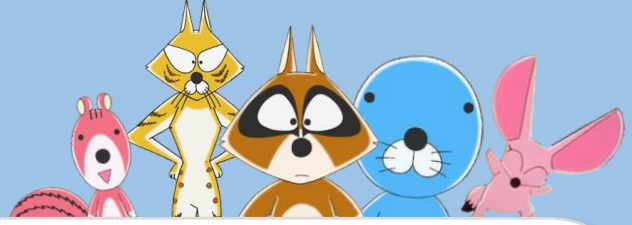
[NLP]

- RoBERTa
- KoBERT
- KcELECTRA

"편백골 관광농원 캠핑장은 글램핑과
야영장이 함께 운영되는 캠핑장이다.
캠핑장 옆으로 계곡물이…"

05 3주차 계획

05 3주차 계획



3주차 계획

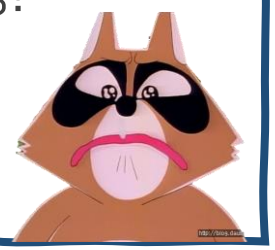
최종적인 카테고리 분류 모델을 구축

↓

기존 키워드 추출 모델과 결합

↓

최종 해시태그 모델 생성!



감사합니당