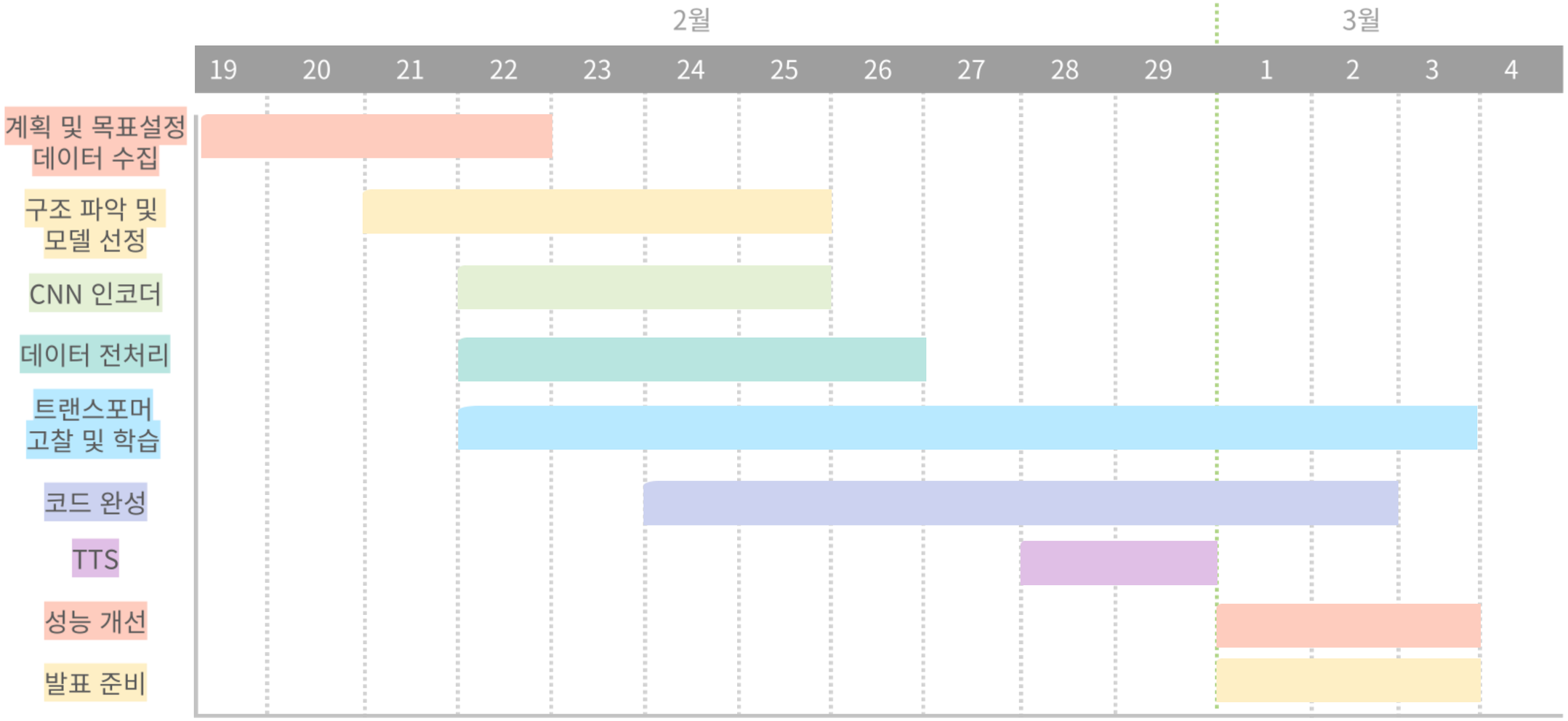
Image Captioning

Image to Caption + TTS

이상치제거팀 이정훈 박준혁 이민형

목차

타임 테이블


2월
3월
19
20
21
22
23
24
25
26
27
28
29
1
2
3
4
계획 및 목표설정
데이터 수집
구조 파악 및
모델 선정
CNN 인코더
데이터 전처리
트랜스포머
고찰 및 학습
코드 완성
TTS
성능 개선
발표 준비

활용 데이터

마이크로 소프트 MS COCO

마이크로 소프트에서 만든 이미지 데이터셋으로
디텍션, 이미지 분류, 이미지 캡셔닝 등
다양한 AI분야에서 활용됨

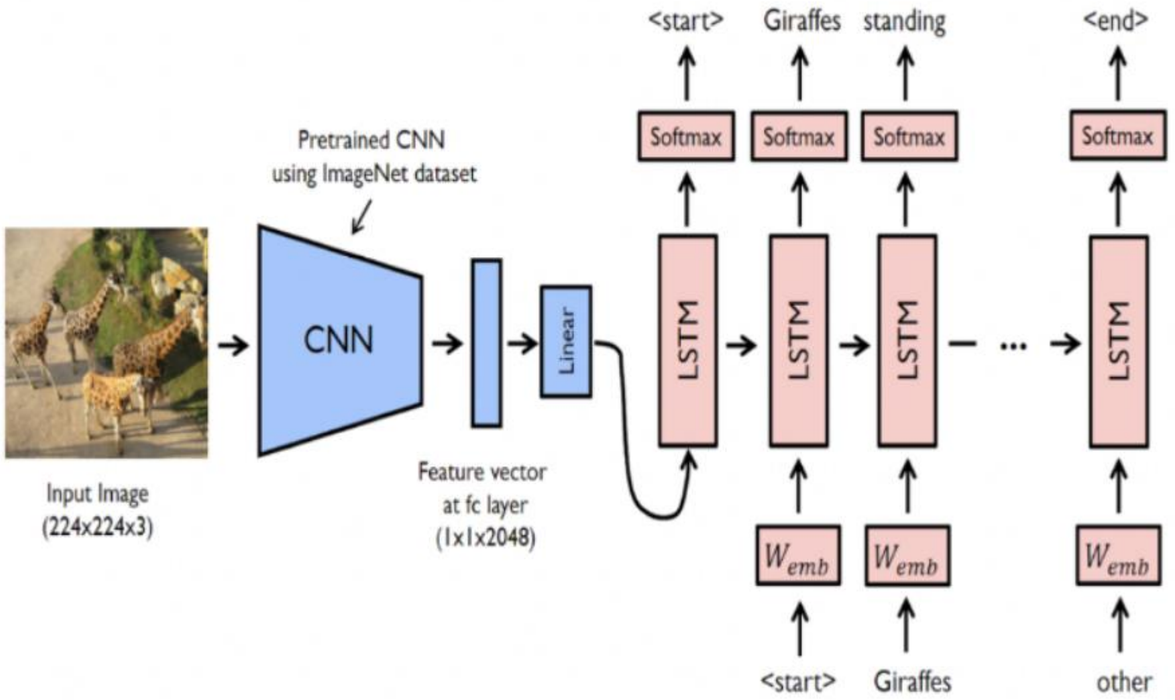
약 12만장의 train 데이터

5000 장의 validation, 약 4만장의 test 데이터

이미지 별로 평균 5개의 캡션

이미지 출처 : https://cocodataset.org/#home

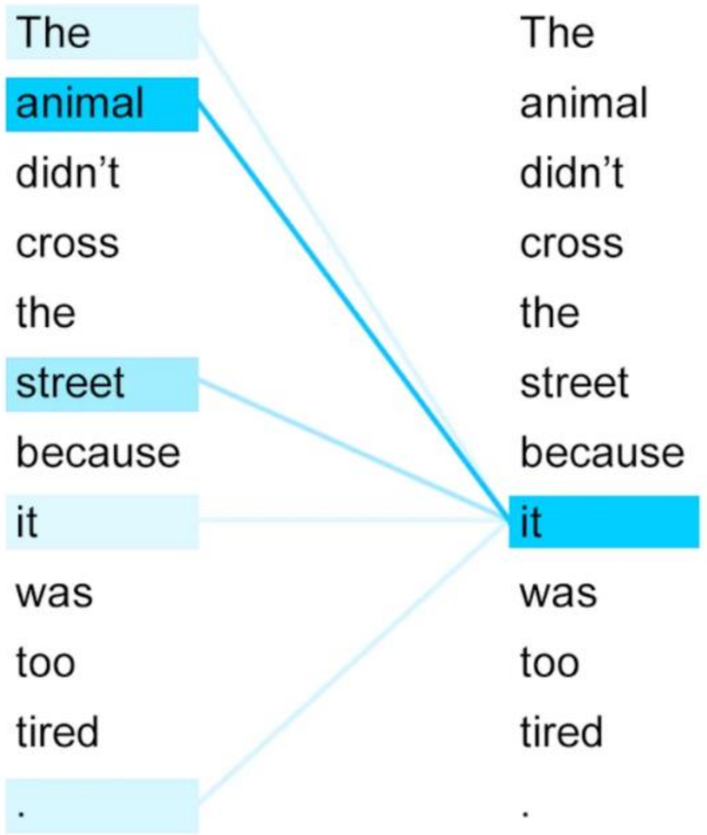
CNN과 RNN을 이용한 이미지 캡셔닝



CNN	이미지의 특징맵 추출, RNN에 전달
RNN	특징맵을 이용해 자연어 처리 및 캡션 학습
RNN의 단점	기울기 소실, 폭주(gradient vasnishing, exploding) 메모리 사용 과다, 시퀀스가 길때의 문제
LSTM의 단점	모델의 복잡성, 여전한 메모리 사용량

이미지 출처 : https://bsnn.tistory.com/45

Attention 메커니즘

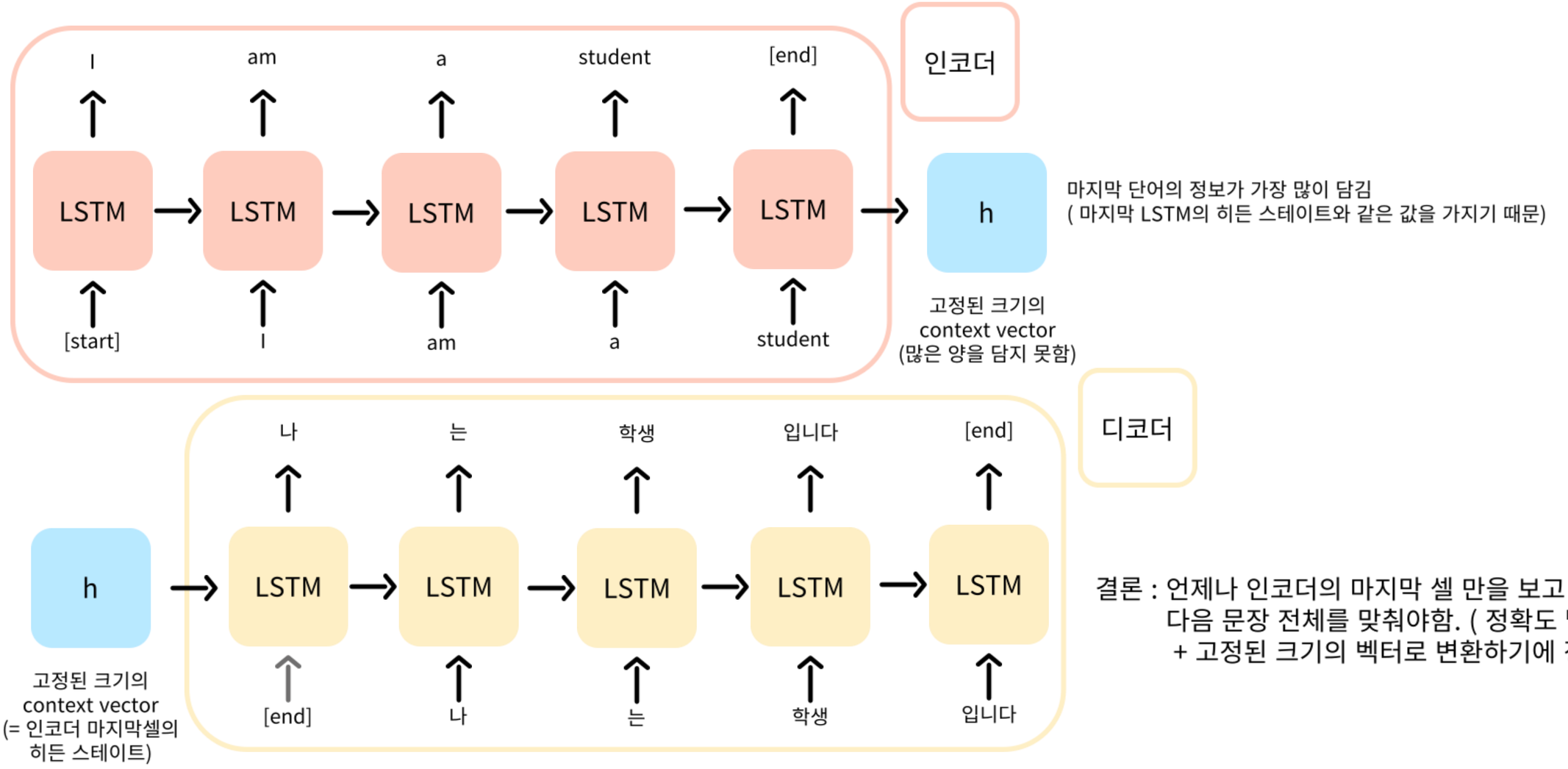


어텐션	입력에 대해서 특정 부분에 주의를 기울이는 메커니즘
Query	다른 모든 단어에 대한 사용되는 현재 단어로, 현재 처리 중인 토큰을 의미
Key	문장의 모든 단어를 Key로 사용함
Value	각 단어가 가지고 있는 의미나 정보를 나타냄 이를 통해 단어들 간의 관계를 이해하고 문장을 이해할 수 있게 됨

좌측 이미지 출처 : https://wikidocs.net/31379

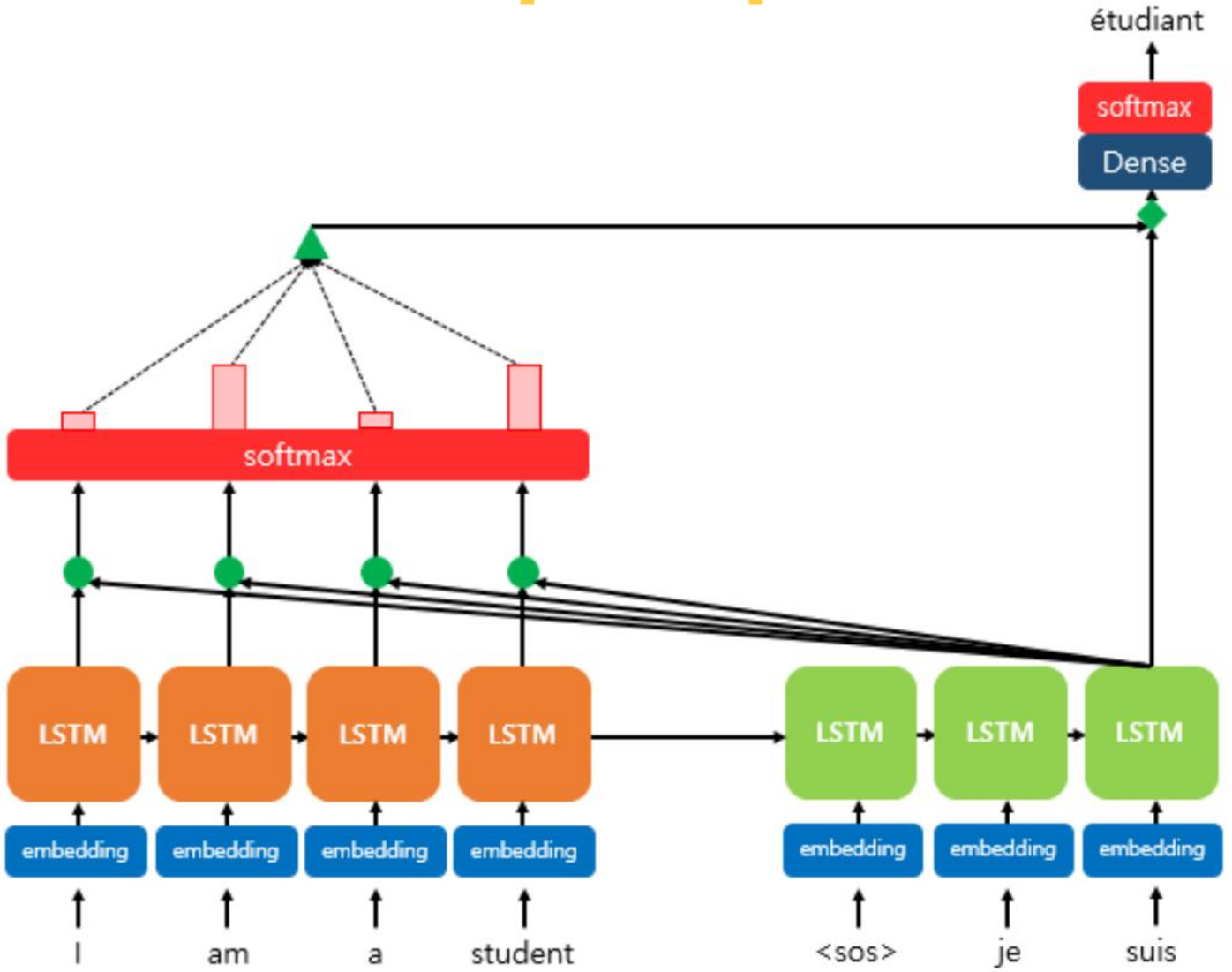
우측 이미지 출처 : https://codingopera.tistory.com/41

Seqence-to-Sequence



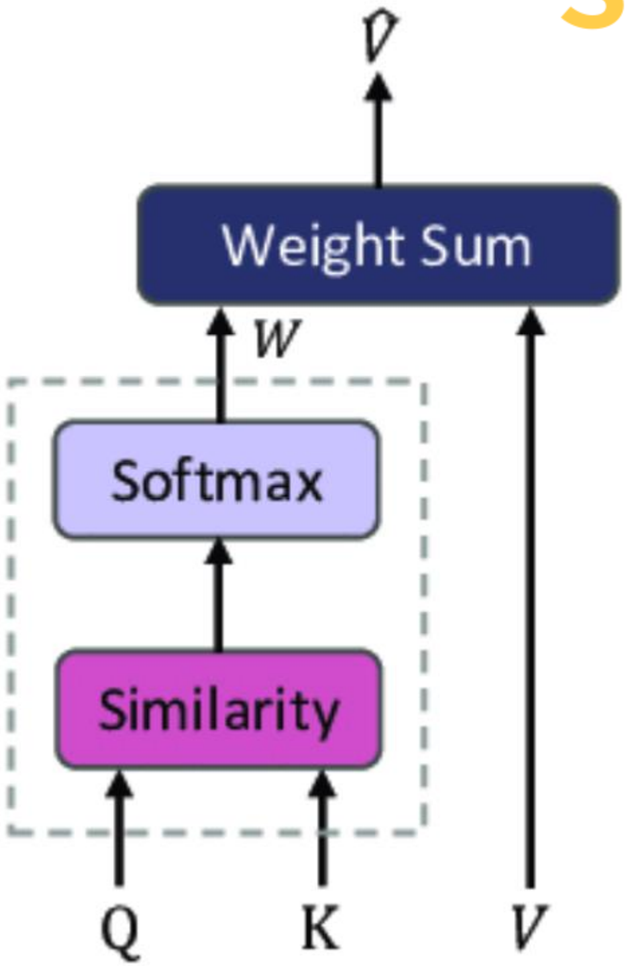
결론 : 언제나 인코더의 마지막 셀 만을 보고
다음 문장 전체를 맞춰야함. (정확도 떨어짐)
+ 고정된 크기의 벡터로 변환하기에 정보 소실

Seq2Seq + Attention



이미지 출처 : https://moondol-ai.tistory.com/316

Self Attention

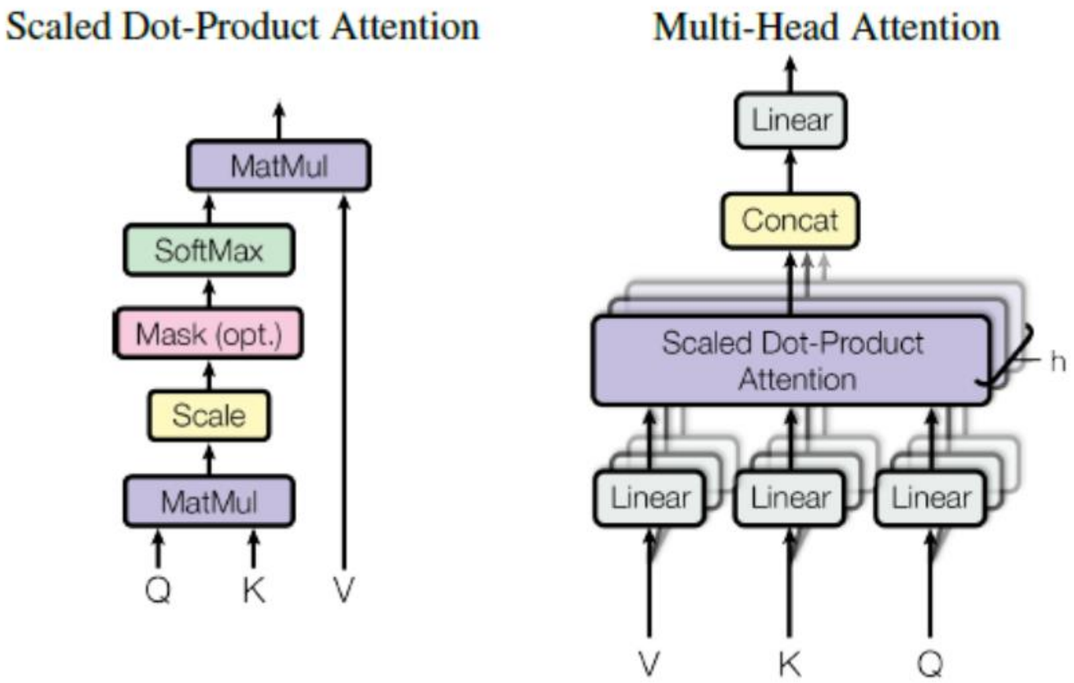


(a) Self-Attention

어텐션	이미지나 문장에서 어떤 부분에 집중을 해야 할지
Q, K, V	각각 Query, Key, Value를 의미
왜 Self 인가	입력 시퀀스의 각 단어나 토큰이 자신과 다른 단어나 토큰 간의 관계를 계산하는 데 사용됨
장점	의미파악, 위치정보 포착에 뛰어남
단점	두가지의 정보를 결합하기 힘듬 ex) 이미지 + 텍스트
결과	Attention Value Matrics (2차원)

이미지 출처 : https://www.researchgate.net/figure/Comparison-between-self-attention-mechanism-and-self-attention-with-AWG-The-AWG-module_fig2_350322794

Multi-Head Attention

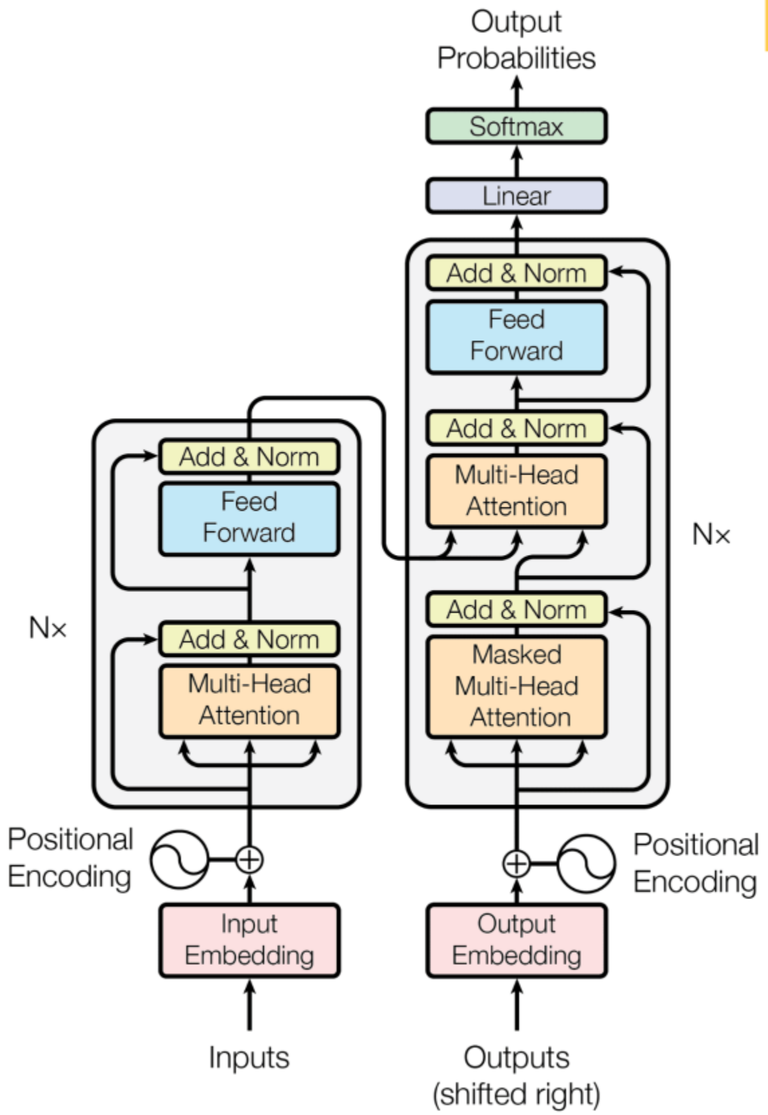


어텐션	이미지나 문장에서 어떤 부분에 집중을 해야 할지
Q, K, V	각각 Query, Key, Value를 의미
MatMul 행렬곱	Query와 Key의 행렬곱 Query가 Key와 얼마나 비슷한지를 나타내는 수치
Mask	사용해선 안될 미래 데이터, 패딩된 부분에 마스크를 시켜 어텐션 가중치를 제거함.
Multi-Head Attention	Scaled Dot-Product Attention을 h만큼 병렬 수행
결과	Attention Value Matrics (2차원)

* Scaled Dot-Porduct Attention은 self-attention의 한 종류

이미지 출처 : Attention Is All You Need 논문 4 페이지

Transformer



트랜스 포머	어텐션 메커니즘을 기반으로한 새로운 Architecture 자연어 처리에 뛰어남
트랜스포머의 구조	어텐션 메커니즘을 기반으로 Encoder와 Decoder가 존재
장점	기존 RNN계열의 문제점을 해결하고 훨씬 자연스러운 문장 생성 가능
Embedding	이미지, 텍스트 -> 벡터
Skip Connection	어텐션 연산하기 전의 데이터와 후의 데이터를 더함
결과	Attention기반의 자연스러운 문장 생성

이미지 출처 : Attention Is All You Need 논문 3 페이지

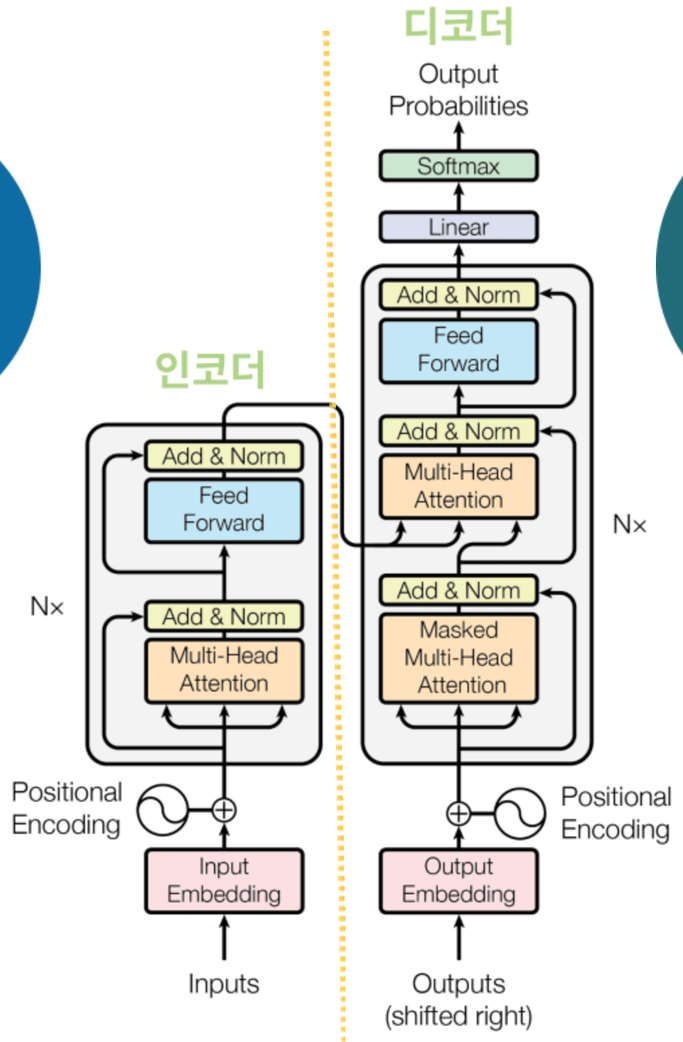
Transformer에 사용되는 3가지 어텐션

1
인코더와 디코더

좌, 우를 반으로 나눴을때를 기준으로
왼쪽이 인코더, 오른쪽이 디코더

2
인코더의 어텐션

인코더의 입력들에 대한
어텐션을 구함
예) 한국어, 영어, 이미지



3
디코더의 첫번째 어텐션

디코더의 입력에 대한
어텐션을 구함
예) 한국어, 영어

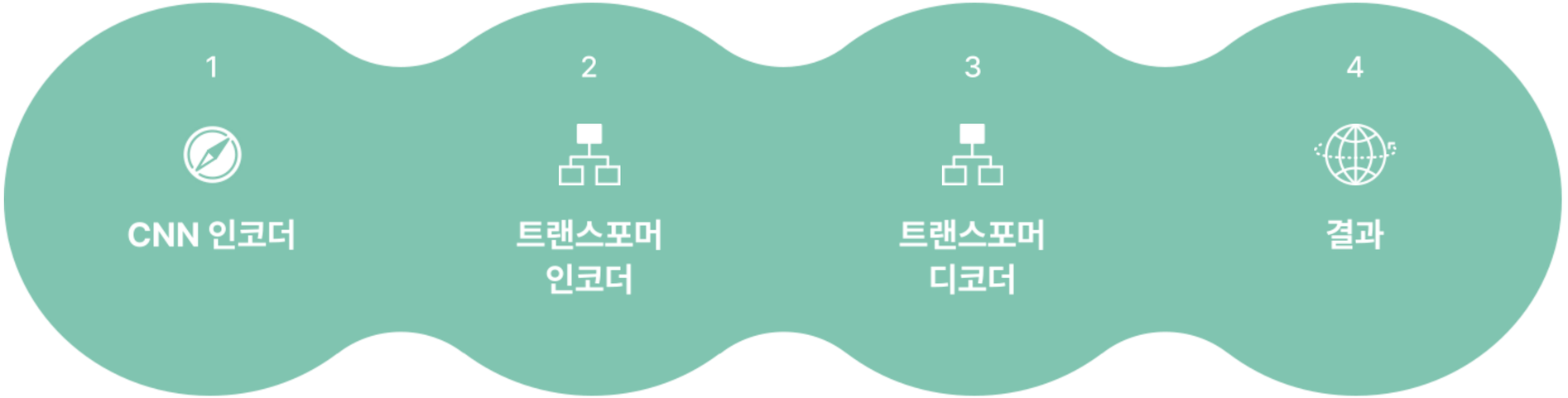
4
디코더의 두번째 어텐션

디코더 어텐션(Q)과
인코더 어텐션(K,V)의
출력끼리 어텐션을 구해서
둘 사이의 관계를 구함

이미지 출처 : Attention Is All You Need 논문 3 페이지

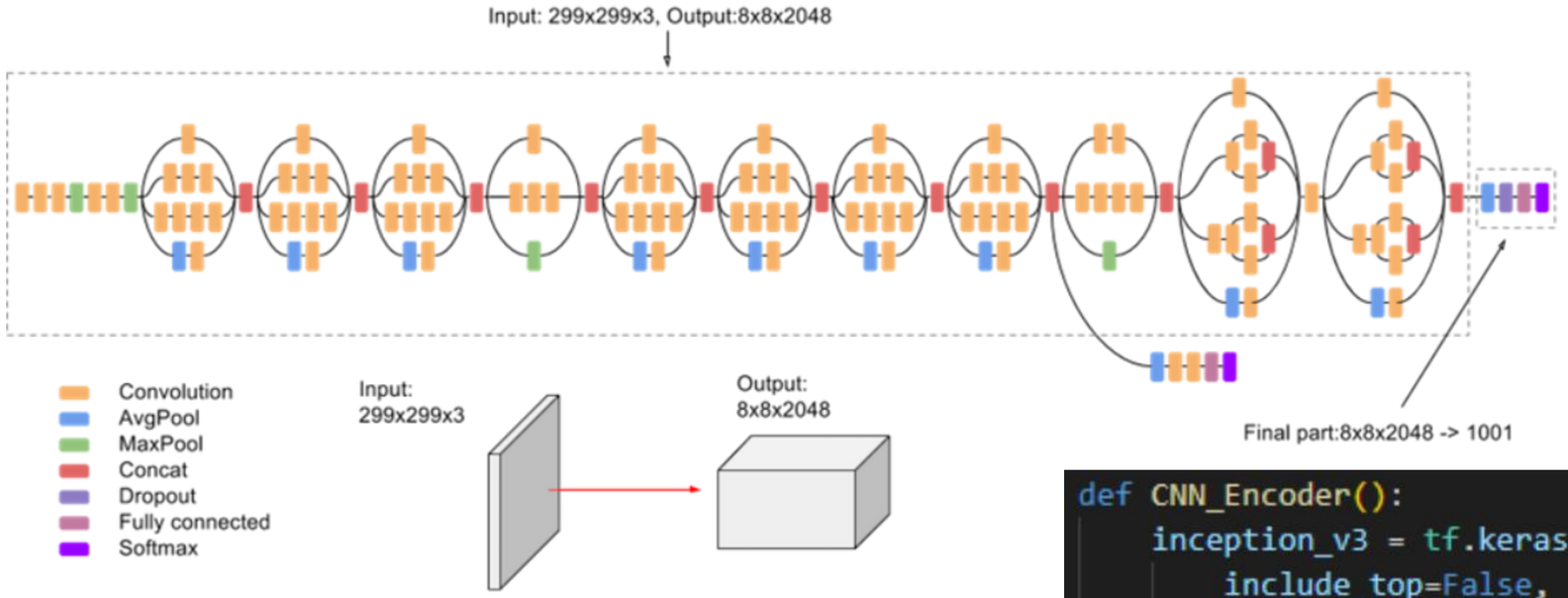
트랜스포머를 사용한 이미지 캡셔닝

+ 핵심코드



코드 출처 : https://www.kaggle.com/code/pritishmishra/image-captioning-on-coco-dataset

CNN Encoder

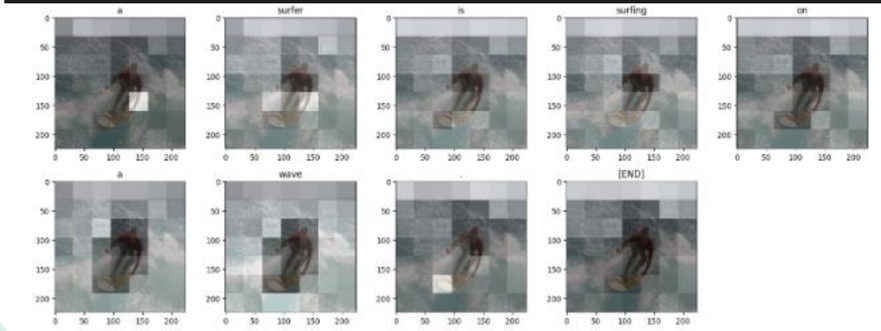


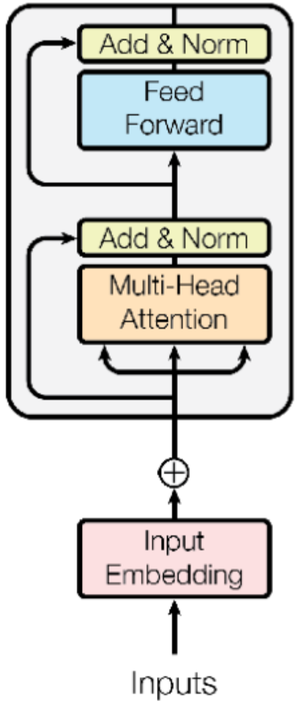

```python
def CNN_Encoder():
    inception_v3 = tf.keras.applications.InceptionV3(
        include_top=False,
        weights='imagenet'
    )

    output = inception_v3.output
    output = tf.keras.layers.Reshape(
        (-1, output.shape[-1]))(output)

    cnn_model = tf.keras.models.Model(inception_v3.input, output)
    return cnn_model
```

이미지 출처 : https://cloud.google.com/tpu/docs/inception-v3-advanced?hl=ko

Transformer Encoder

```python
class TransformerEncoderLayer(tf.keras.layers.Layer):

    def __init__(self, embed_dim, num_heads):
        super().__init__()
        self.layer_norm_1 = tf.keras.layers.LayerNormalization()
        self.layer_norm_2 = tf.keras.layers.LayerNormalization()
        self.attention = tf.keras.layers.MultiHeadAttention(
            num_heads=num_heads, key_dim=embed_dim)
        self.dense = tf.keras.layers.Dense(embed_dim, activation="relu")


    def call(self, x, training):
        x = self.layer_norm_1(x)
        x = self.dense(x)

        attn_output = self.attention(
            query=x,
            value=x,
            key=x,
            attention_mask=None,
            training=training
        )

        x = self.layer_norm_2(x + attn_output)
        return x
```





Inputs	이미지 특성맵 벡터
Input Embedding	이미지 특성맵 벡터를 시퀀스와 같은 크기로 변환
Multi-Head Attention	이미지 특성맵 내에서 어디가 중요한지 파악
Feed Forward	Activation에 Relu를 사용한 Dense 레이어
Add & Norm	어텐션 연산하기 전의 데이터와 후의 데이터를 더하고 Normalization
결과	기존 이미지 특성맵 벡터 + 어텐션

이미지 출처 : Attention Is All You Need 논문 3 페이지

Transformer Decoder

```python
def call(self, input_ids, encoder_output, training, mask=None):
    embeddings = self.embedding(input_ids)

    combined_mask = None
    padding_mask = None

    if mask is not None:
        causal_mask = self.get_causal_attention_mask(embeddings)
        padding_mask = tf.cast(mask[:, :, tf.newaxis], dtype=tf.int32)
        combined_mask = tf.cast(mask[:, tf.newaxis, :], dtype=tf.int32)
        combined_mask = tf.minimum(combined_mask, causal_mask)

    attn_output_1 = self.attention_1(
        query=embeddings,
        value=embeddings,
        key=embeddings,
        attention_mask=combined_mask,
        training=training
    )

    out_1 = self.layernorm_1(embeddings + attn_output_1)

    attn_output_2 = self.attention_2(
        query=out_1,
        value=encoder_output,
        key=encoder_output,
        attention_mask=padding_mask,
        training=training
    )

    out_2 = self.layernorm_2(out_1 + attn_output_2)

    ffn_out = self.ffn_layer_1(out_2)
    ffn_out = self.dropout_1(ffn_out, training=training)
    ffn_out = self.ffn_layer_2(ffn_out)

    ffn_out = self.layernorm_3(ffn_out + out_2)
    ffn_out = self.dropout_2(ffn_out, training=training)
    preds = self.out(ffn_out)
    return preds
```

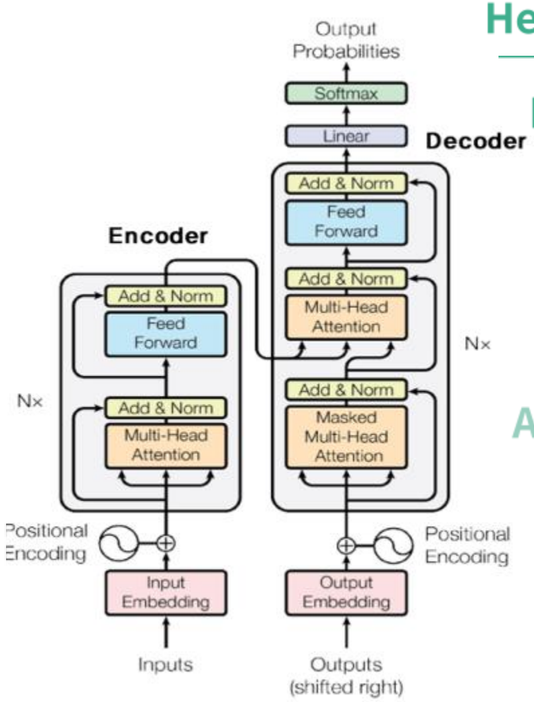
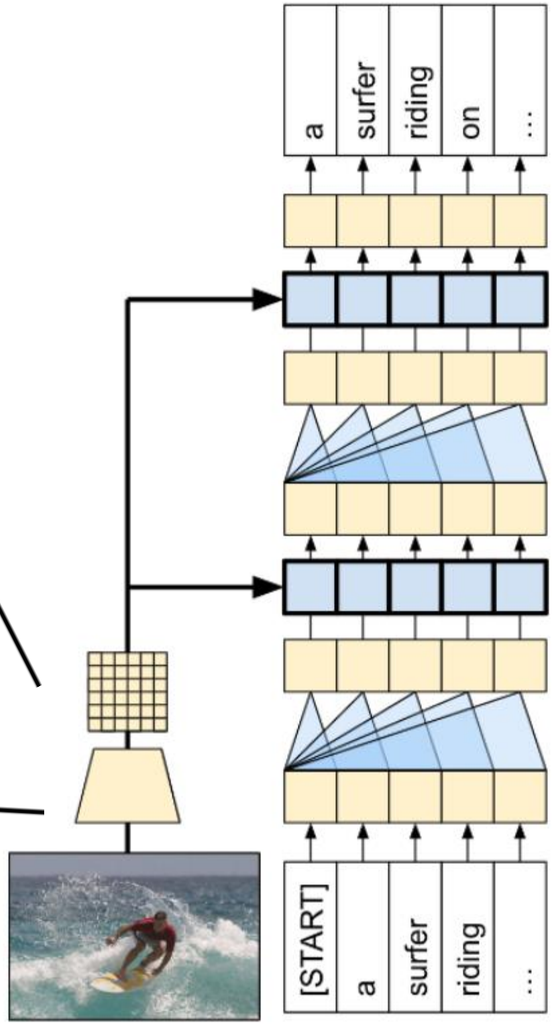


Figure 1: The Transformer - model architecture.

Outputs	이미지에 해당하는 캡션
Output Embedding	단어 토큰에 대한 정수화 (Vectorization) 토큰 임베딩과 위치 임베딩 둘 다함
Masked Multi-Head Attention	패딩된 토큰과 사용하지 말아야할 미래의 토큰까지 사용하는것을 막아줌
Multi-Head Attention	이미지와 캡션을 매칭하며 학습
Feed Forward	Activation에 Relu를 사용한 Dense 레이어
Add & Norm	어텐션 연산하기 전의 데이터와 후의 데이터를 더하고 Normalization
결과	이미지를 보고 이미지의 중요한 부분을 파악, 그에 맞는 알맞은 단어를 선택해 문장을 생성

이미지 출처 : Attention Is All You Need 논문 3 페이지

이미지 캡셔닝의 구조



트랜스포머 인코더

이미지의 중요한 부분 파악

CNN 인코더

이미지의 특성 벡터 추출

트랜스포머 디코더의 출력

이미지의 중요한 부분에 집중해 그에 맞는 문장 생성

디코더의 두번째 어텐션

이미지와 단어를 매칭하며 학습

디코더의 첫번째 어텐션

단어 사이의 연관성 파악

이미지 출처 : https://www.tensorflow.org/text/tutorials/image_captioning?hl=en

결과

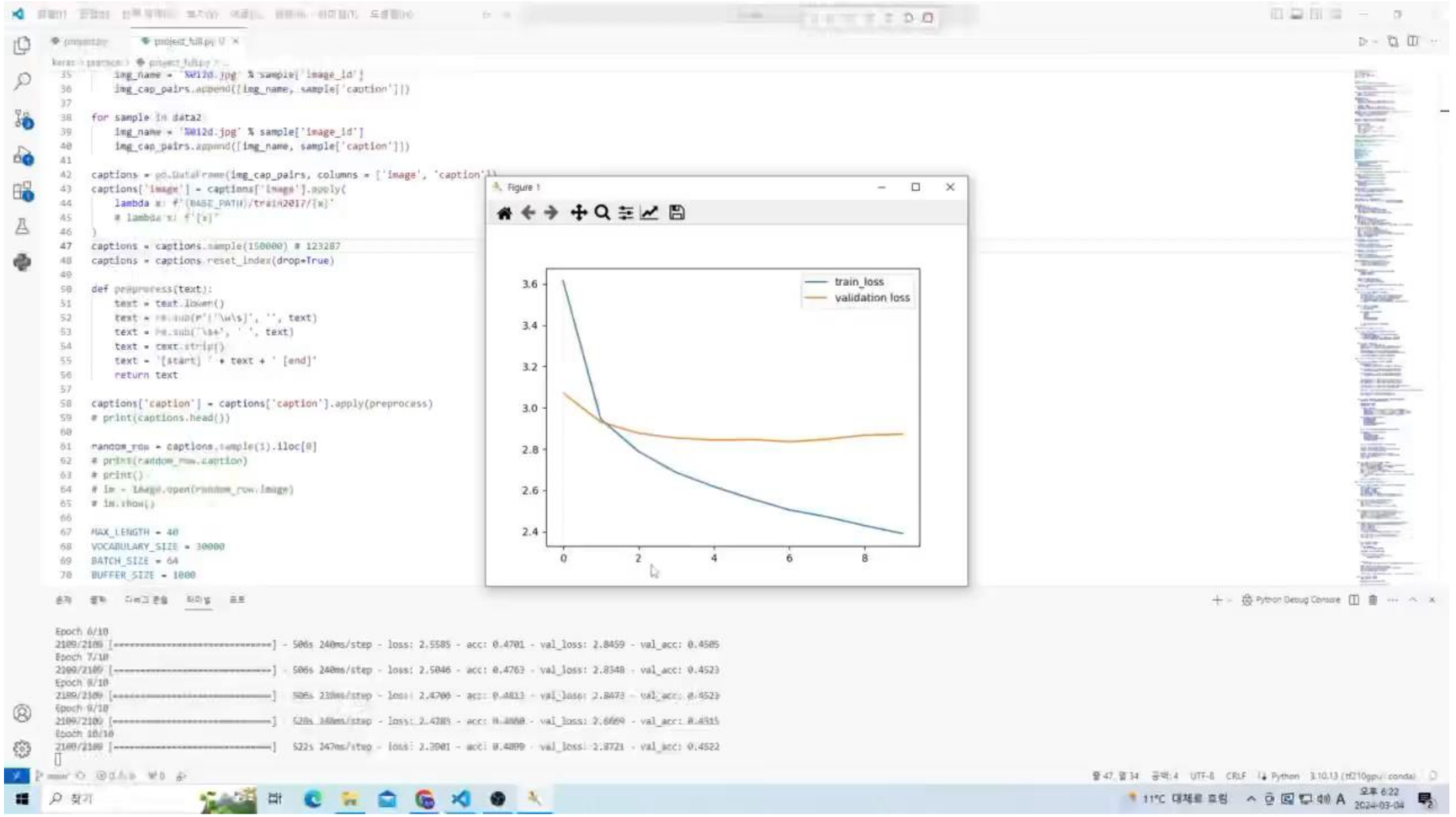


Figure 1
train_loss
validation loss
3.6
3.4
3.2
3.0
2.8
2.6
2.4
0
2
4
6
8
Epoch 6/10
2100/2100 [==============================] - 506s 240ms/step - loss: 2.5585 - acc: 0.4701 - val_loss: 2.8459 - val_acc: 0.4505
Epoch 7/10
2100/2100 [==============================] - 506s 240ms/step - loss: 2.5046 - acc: 0.4763 - val_loss: 2.8348 - val_acc: 0.4523
Epoch 8/10
2100/2100 [==============================] - 505s 240ms/step - loss: 2.4700 - acc: 0.4813 - val_loss: 2.8473 - val_acc: 0.4523
Epoch 9/10
2100/2100 [==============================] - 520s 248ms/step - loss: 2.4385 - acc: 0.4860 - val_loss: 2.8669 - val_acc: 0.4515
Epoch 10/10
2100/2100 [==============================] - 522s 247ms/step - loss: 2.3981 - acc: 0.4899 - val_loss: 2.8721 - val_acc: 0.4522

감사합니다

「 이상치 제거팀 : 박준혁, 이민형, 이정훈 」