
Image Captioning

Image to Caption + TTS

「 이상치 제거팀 : 이민형, 박준혁, 이정훈 」

목차

1

타임테이블

2

CNN과 RNN을 이용한 이미지 캡셔닝

3

Attention

4

트랜스포머

5

트랜스포머에 사용되는 3가지 어텐션

6

트랜스포머를 활용한 이미지 캡셔닝

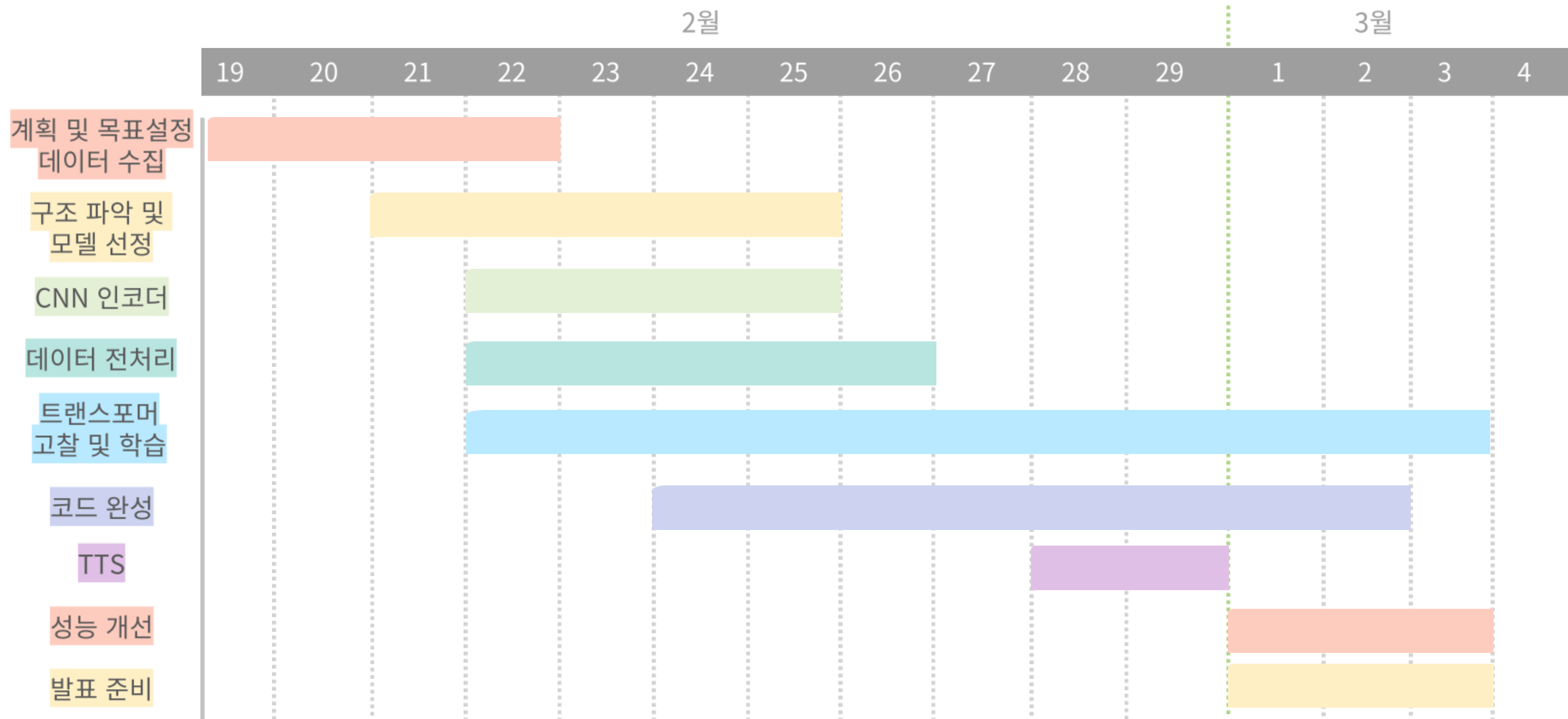
7

핵심 코드

8

결과

타임 테이블



활용 데이터

마이크로 소프트 MS COCO



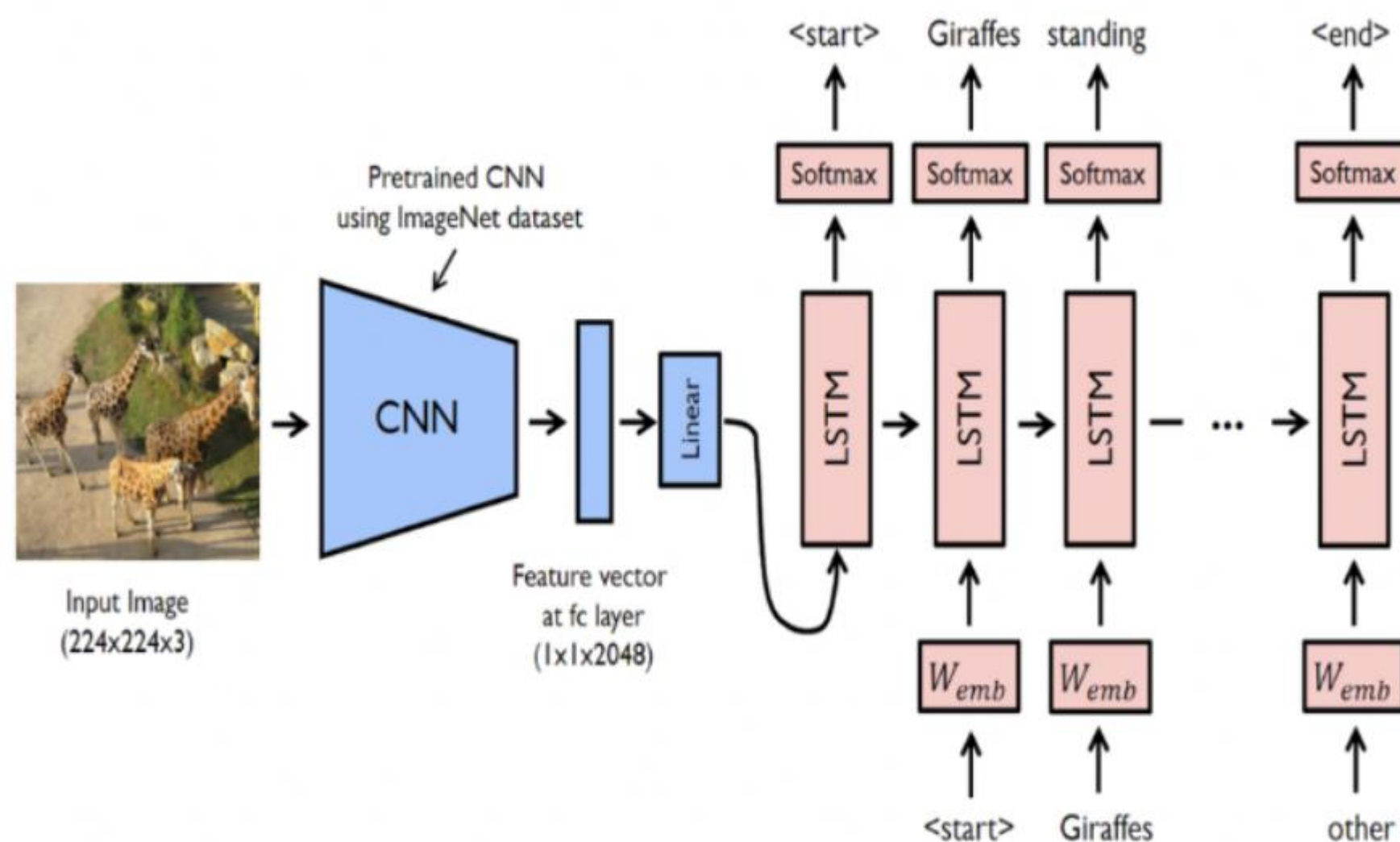
마이크로 소프트에서 만든 이미지 데이터셋으로
디텍션, 이미지 분류, 이미지 캡셔닝 등
다양한 AI분야에서 활용됨

약 12만장의 train 데이터

5000 장의 validation, 약 4만장의
test 데이터

이미지 별로 평균 5개의 캡션

CNN과 RNN을 이용한 이미지 캡셔닝



CNN

이미지의 특징맵 추출, RNN에 전달

RNN

특징맵을 이용해 자연어 처리 및 캡션 학습

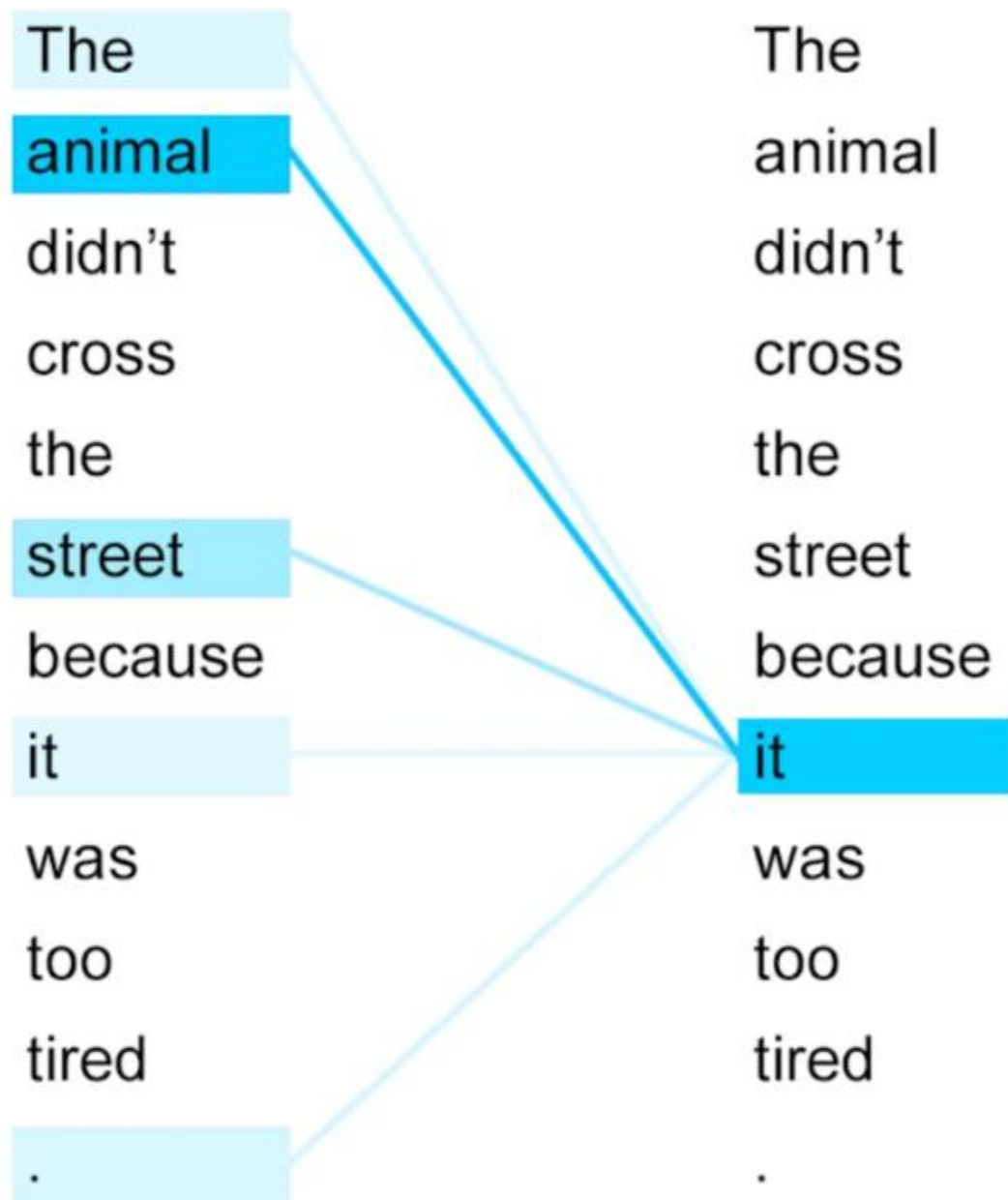
RNN의 단점

기울기 소실, 폭주 (gradient vanishing, exploding)
메모리 사용 과다, 시퀀스가 길때의 문제

LSTM의 단점

모델의 복잡성, 여전한 메모리 사용량

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

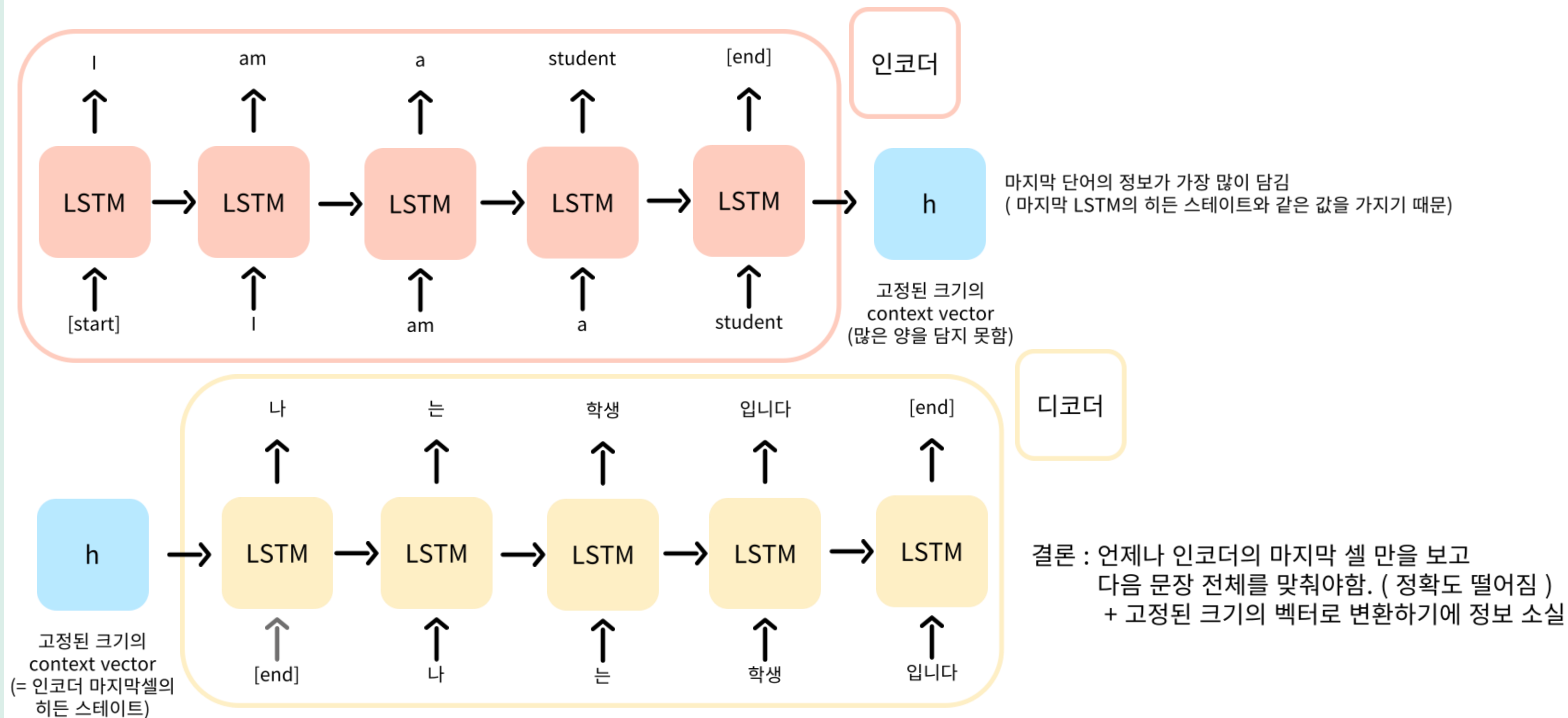
어제, 카페, 갔었어, 거기

Source sequence

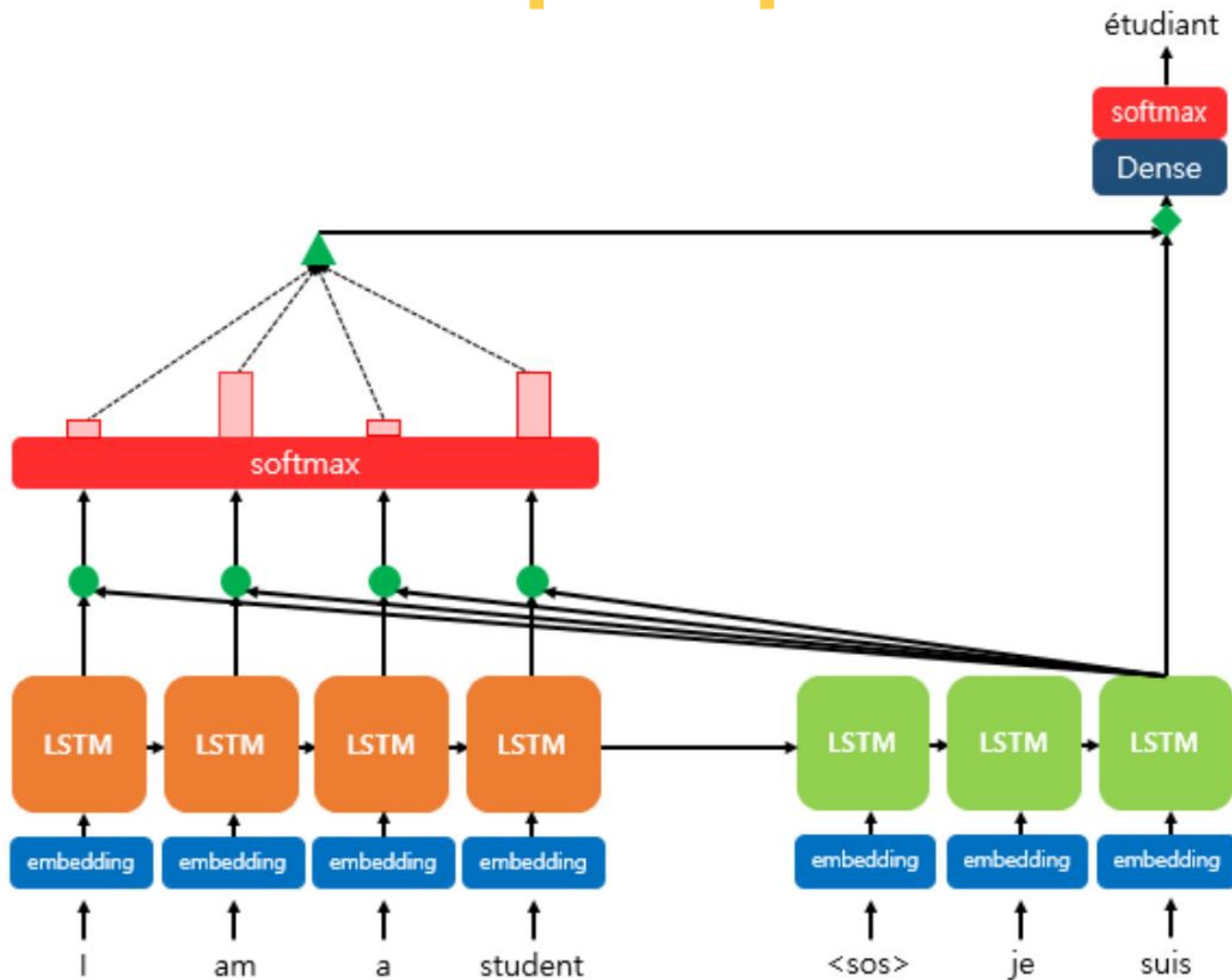
I, went, to, **cafe**

Target sequence

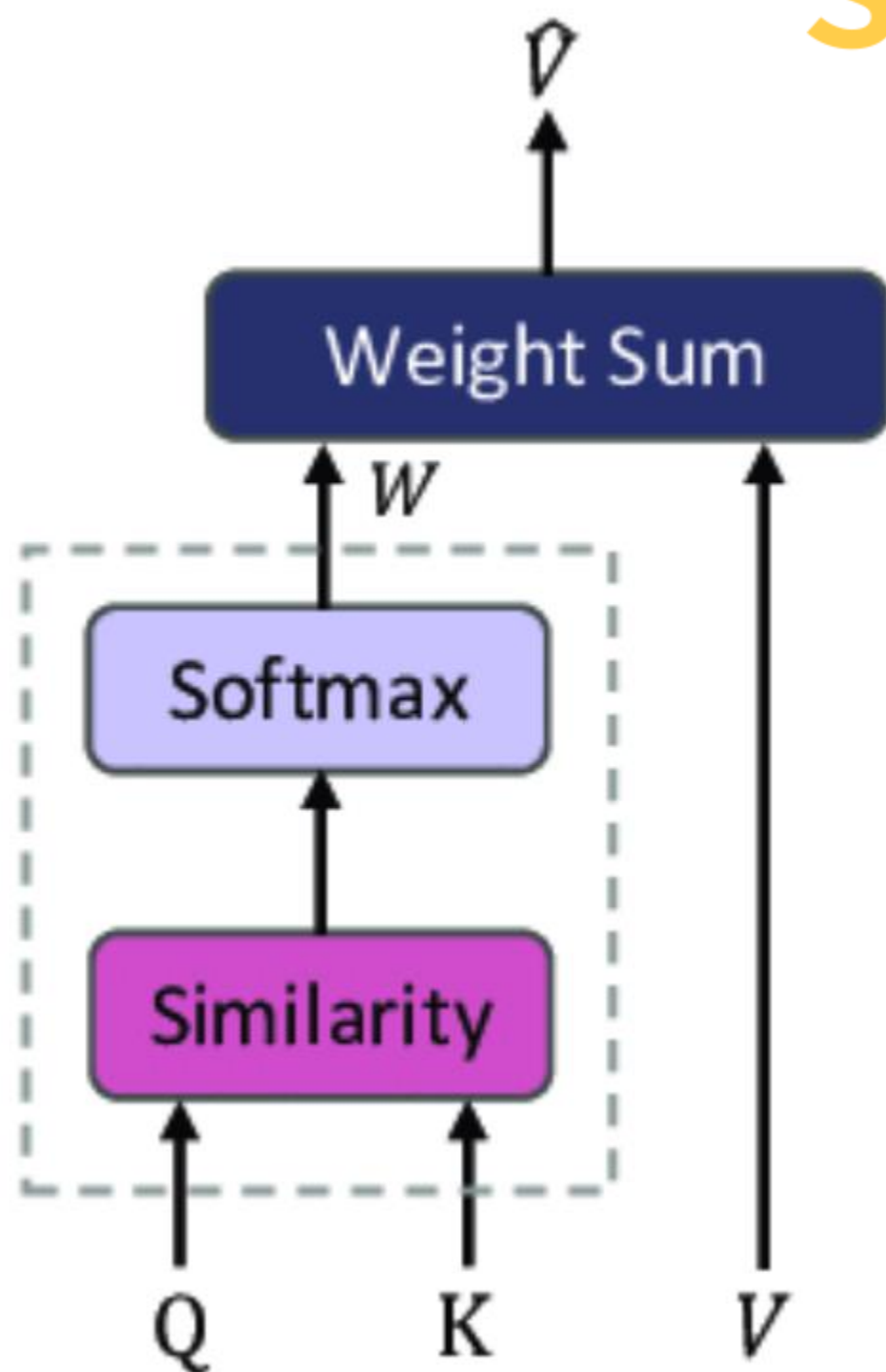
Sequence-to-Sequence



Seq2Seq + Attention



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

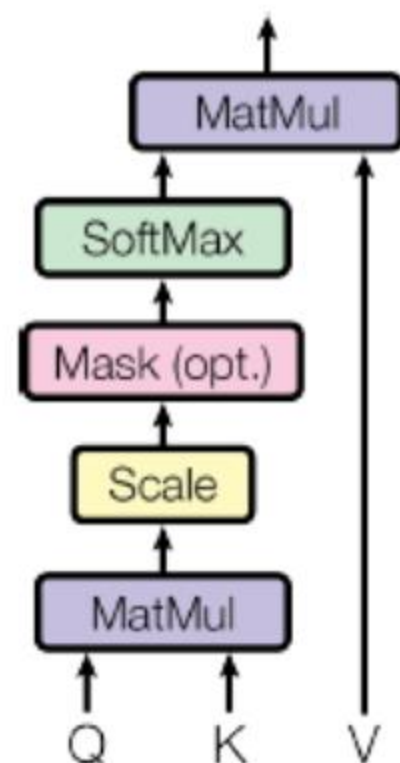
장점 의미파악, 위치정보 포착에 뛰어남

단점 두가지의 정보를 결합하기 힘들
ex) 이미지 + 텍스트

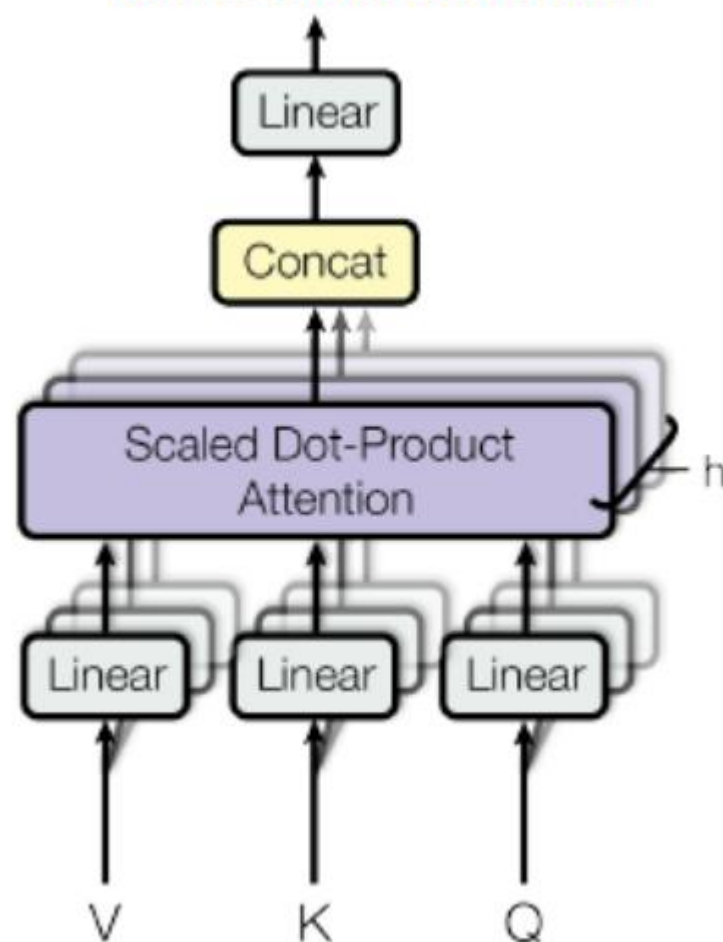
결과 Attention Value Matrics

Multi-Head Attention

Scaled Dot-Product Attention



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

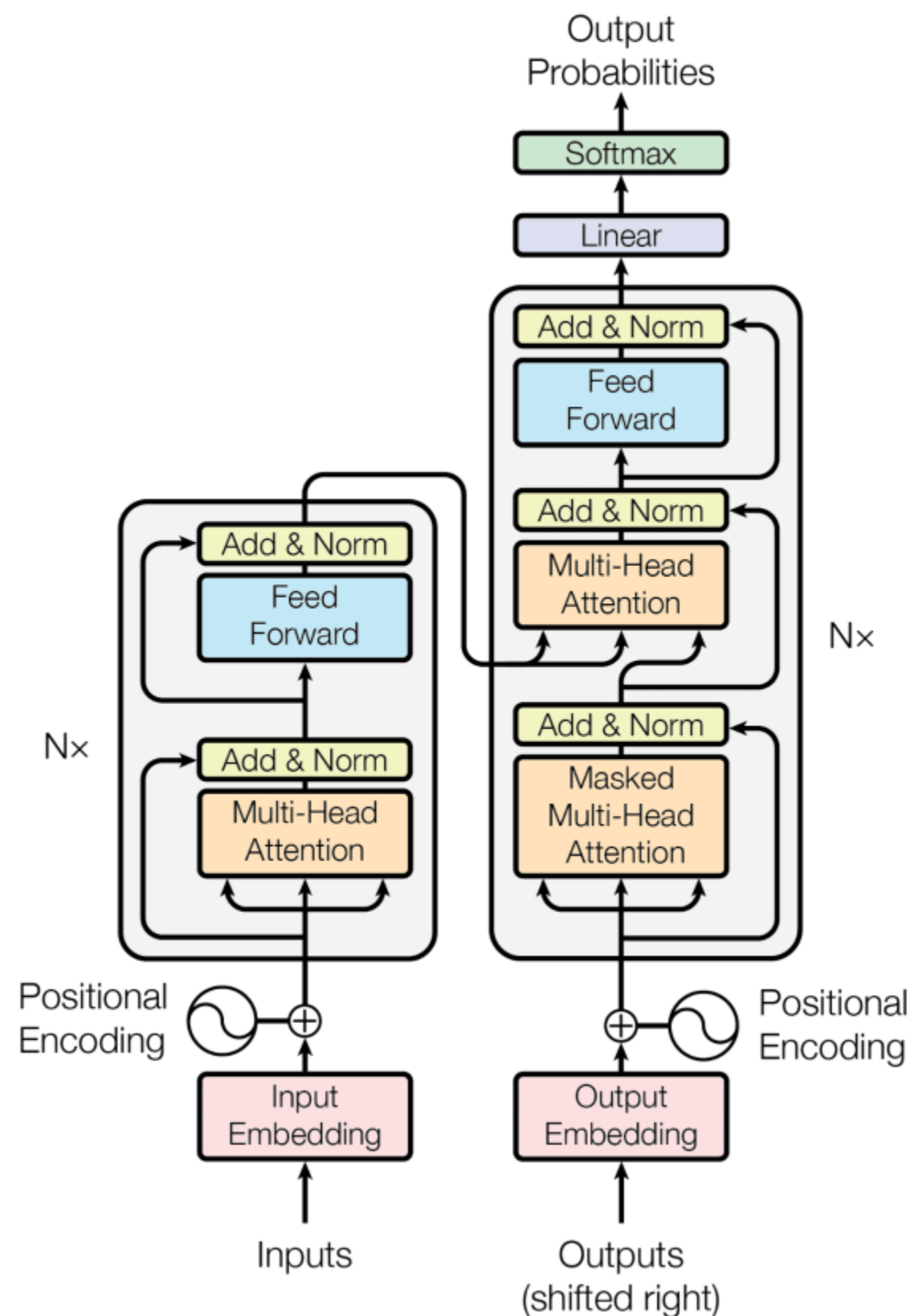
결과

Attention Value Matrics

* Scaled Dot-Product Attention은 self-attention의 한 종류

이미지 출처 : Attention Is All You Need 논문 4 페이지

Transformer



트랜스 포머

어텐션 메커니즘을 기반으로 한 새로운 Architecture
자연어 처리에 뛰어남

트랜스포머의 구조

어텐션 메커니즘을 기반으로
Encoder와 Decoder가 존재

장점

기존 RNN 계열의 문제점을 해결하고
훨씬 자연스러운 문장 생성 가능

Embedding

이미지, 텍스트 -> 벡터

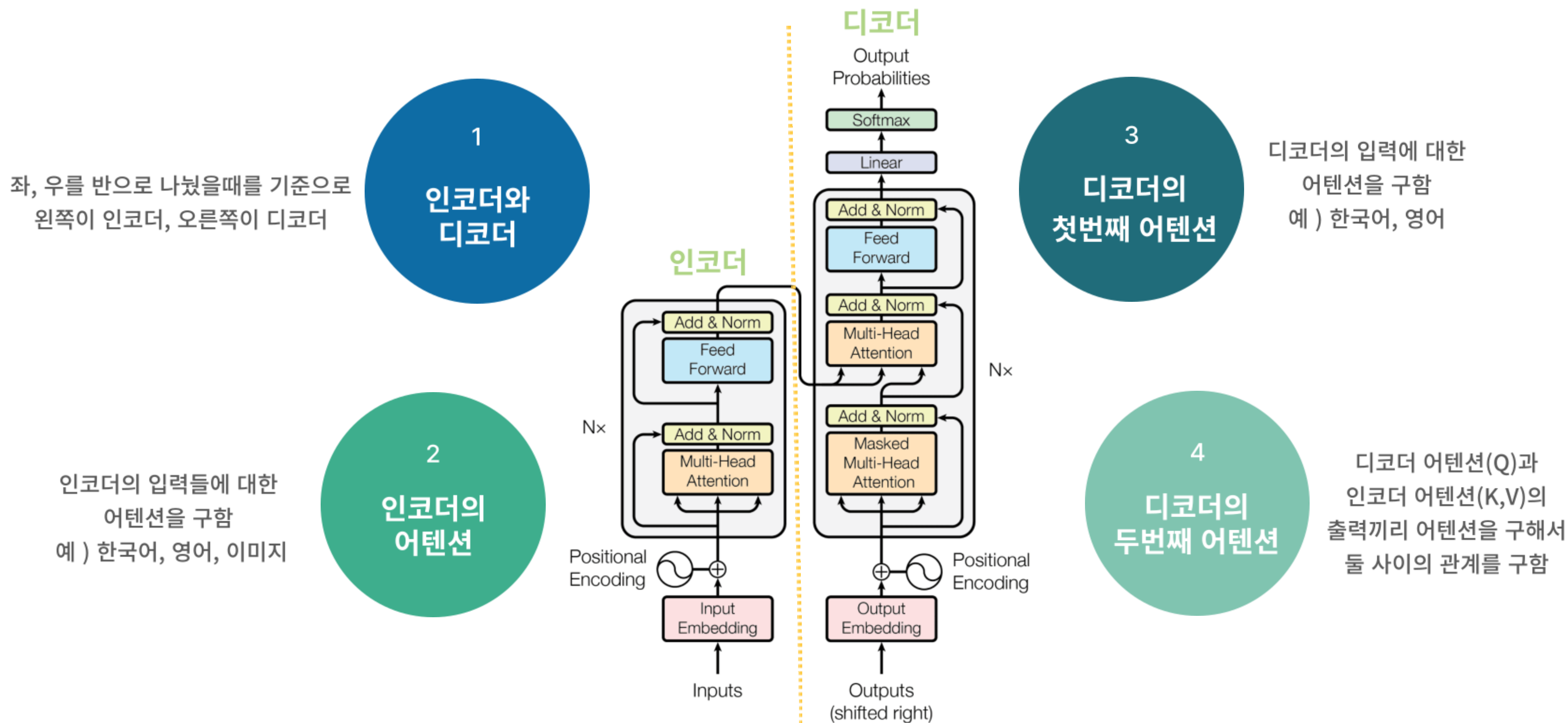
Skip Connection

어텐션 연산하기 전의 데이터와 후의 데이터를 더함

결과

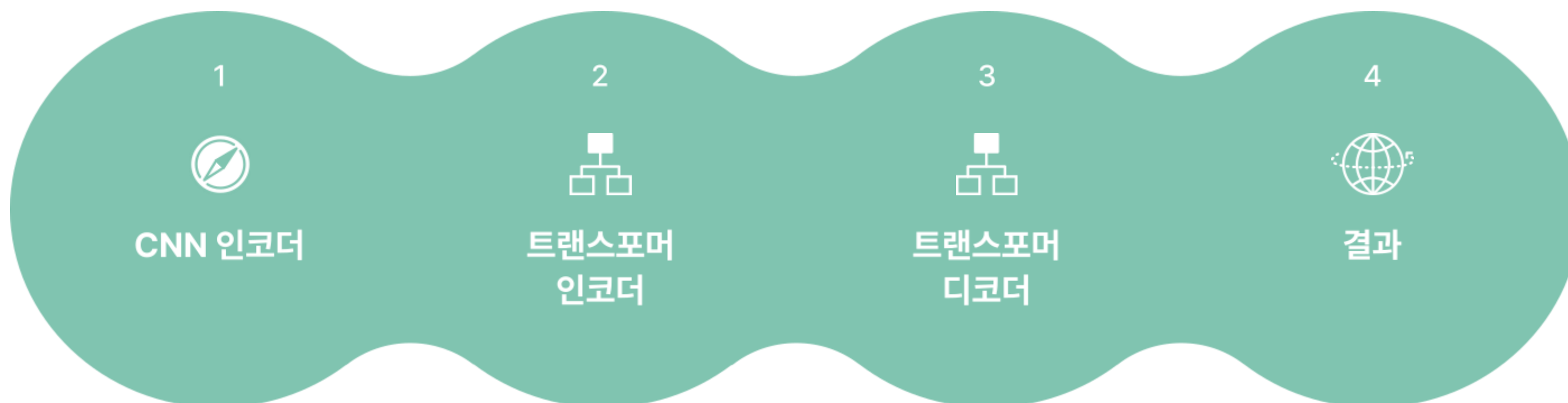
Attention 기반의 자연스러운 문장 생성

Transformer에 사용되는 3가지 어텐션

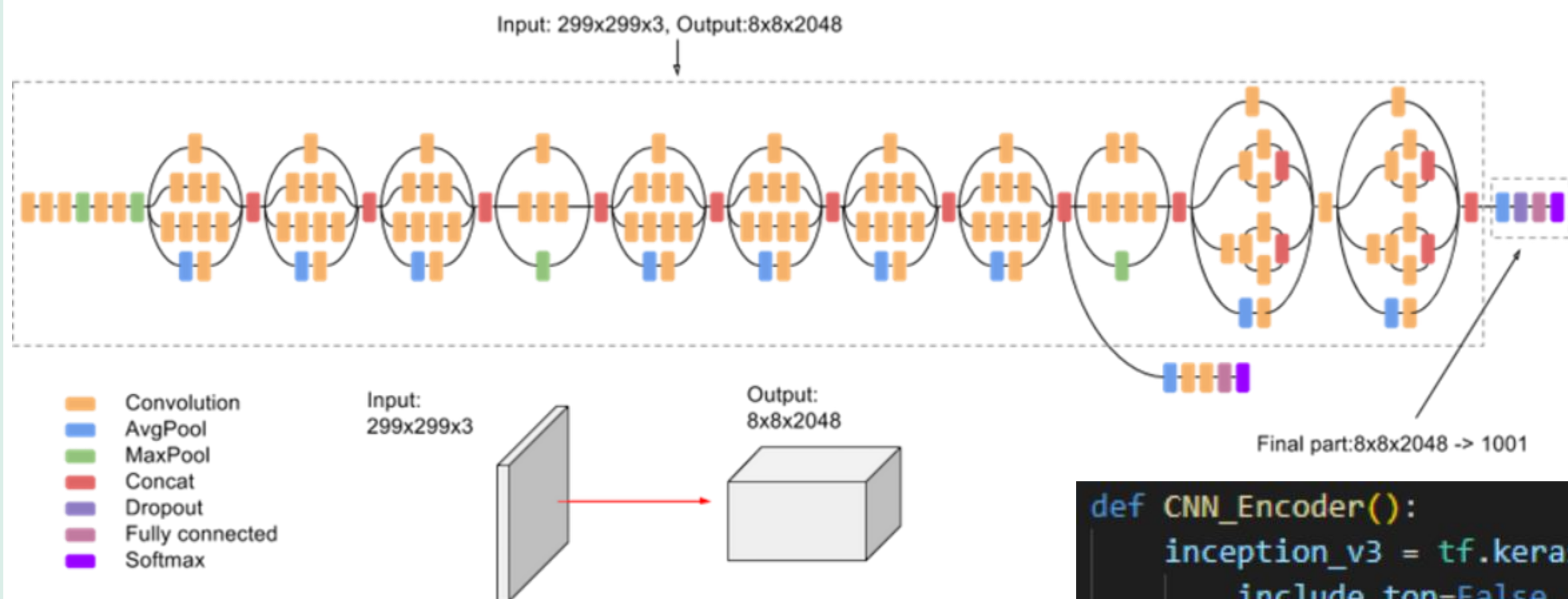


트랜스포머를 사용한 이미지 캡셔닝

+ 핵심코드



CNN Encoder



```
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
```

Transformer Encoder

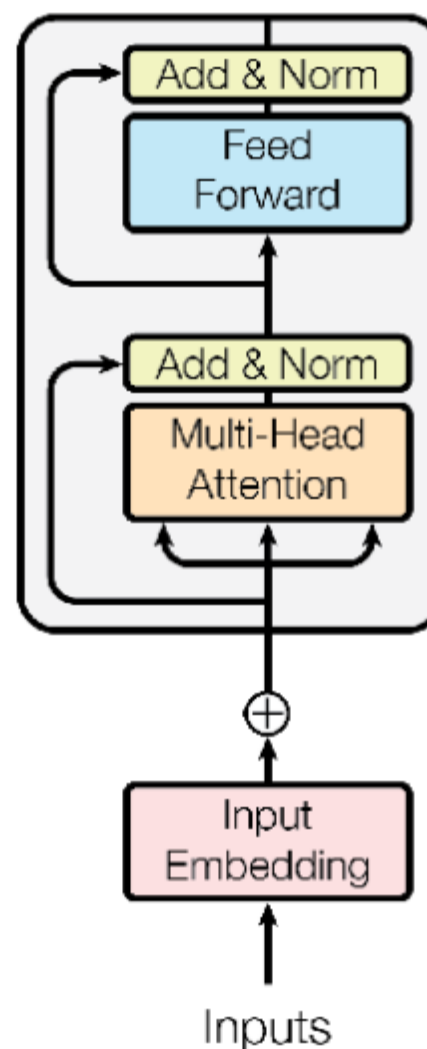
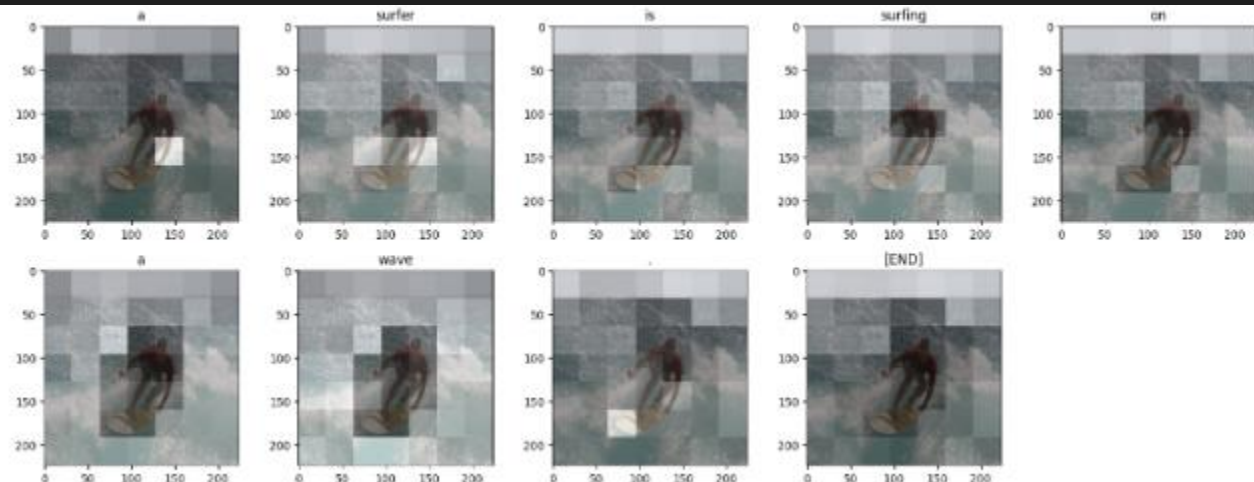
```
class TransformerEncoderLayer(tf.keras.layers.Layer):

    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

Transformer Decoder

```
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
```

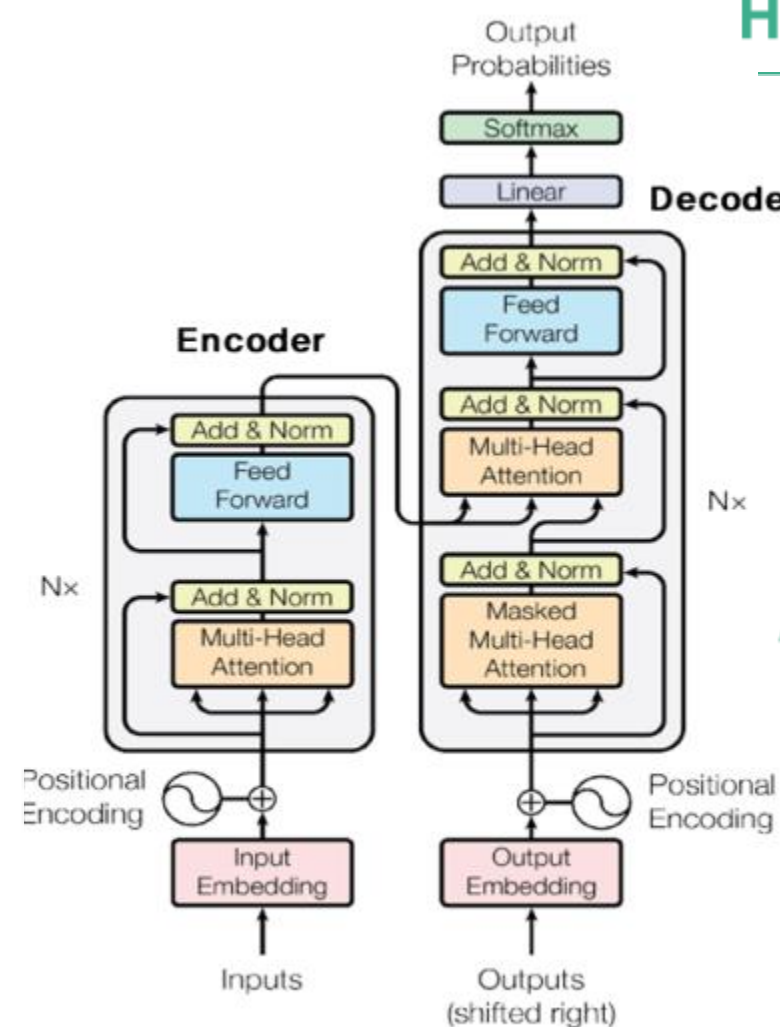


Figure 1: The Transformer - model architecture.

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

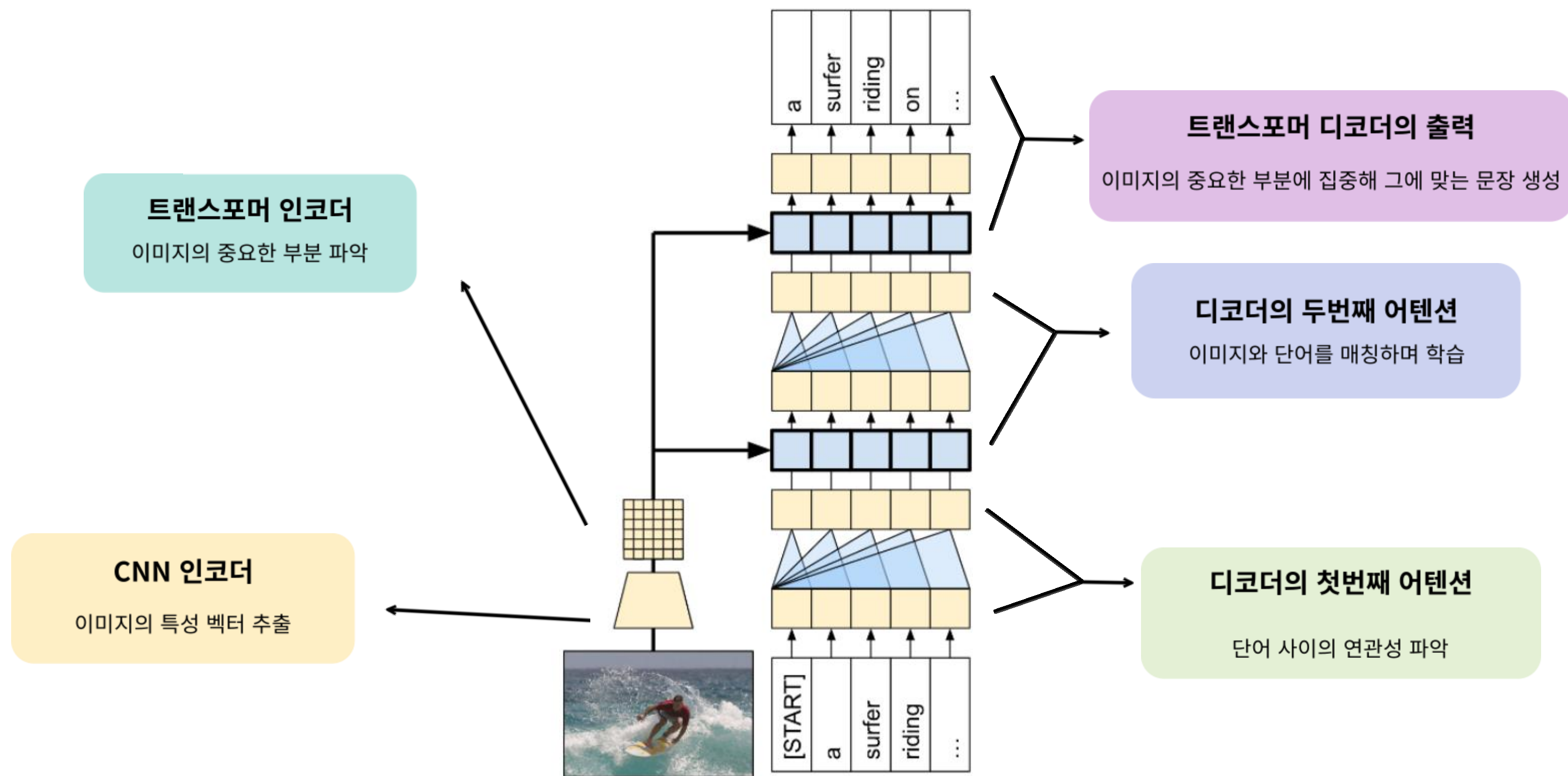
Add & Norm

어텐션 연산하기 전의 데이터와 후의 데이터를 더하고
Normalization

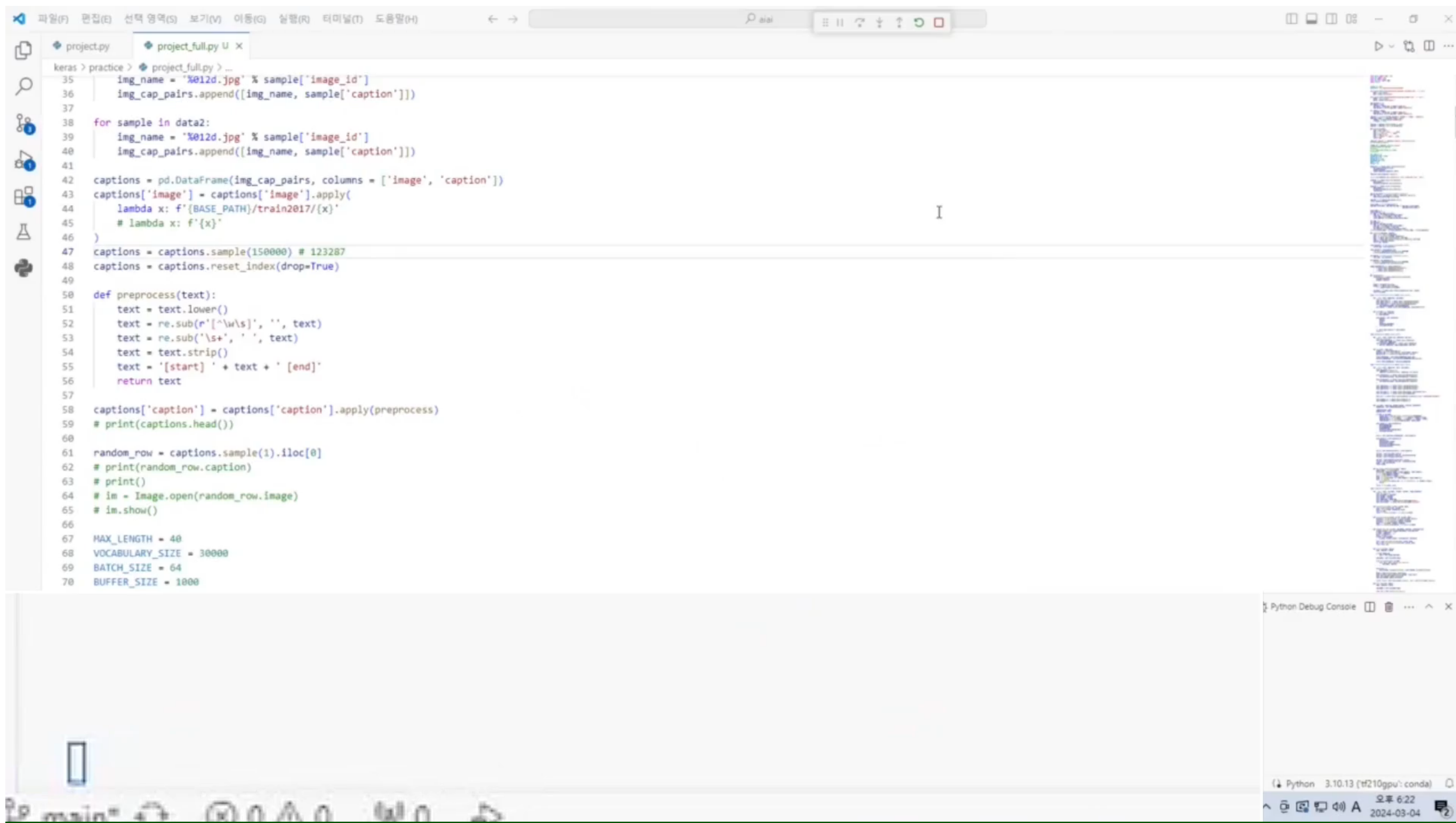
결과

이미지를 보고 이미지의 중요한 부분을 파악,
그에 맞는 알맞은 단어를 선택해 문장을 생성

이미지 캡셔닝의 구조



결과



```
keras > practice > project_full.py > ...
35 img_name = '%012d.jpg' % sample['image_id']
36 img_cap_pairs.append([img_name, sample['caption']])
37
38 for sample in data2:
39     img_name = '%012d.jpg' % sample['image_id']
40     img_cap_pairs.append([img_name, sample['caption']])
41
42 captions = pd.DataFrame(img_cap_pairs, columns = ['image', 'caption'])
43 captions['image'] = captions['image'].apply(
44     lambda x: f'{BASE_PATH}/train2017/{x}'
45     # lambda x: f'{x}'
46 )
47 captions = captions.sample(150000) # 123287
48 captions = captions.reset_index(drop=True)
49
50 def preprocess(text):
51     text = text.lower()
52     text = re.sub(r'[\^\\w\s]', '', text)
53     text = re.sub('\s+', ' ', text)
54     text = text.strip()
55     text = '[start] ' + text + ' [end]'
56     return text
57
58 captions['caption'] = captions['caption'].apply(preprocess)
59 # print(captions.head())
60
61 random_row = captions.sample(1).iloc[0]
62 # print(random_row.caption)
63 # print()
64 # im = Image.open(random_row.image)
65 # im.show()
66
67 MAX_LENGTH = 40
68 VOCABULARY_SIZE = 30000
69 BATCH_SIZE = 64
70 BUFFER_SIZE = 1000
```

Python Debug Console

Python 3.10.13 (tf210gpu: conda)

오후 6:22
2024-03-04

감사합니다

「 이상치 제거팀 : 이민형, 박준혁, 이정훈 」