week 4

1. **cnn**

cnn 모델은 3차원 데이터를 입력 데이터로 넣어야함

input.shape 때려본 다음에 2차원이면

```
input= np.expand_dims(input,axis=-1)
input.shape
```

이거 써서 차원 하나 더 들려준다.

Last-layer activation and loss function combinations

Problem type	Last-layer activation	Loss function	Example
Binary classification	sigmoid	binary_crossentropy	Dog vs cat, Sentiemnt analysis(pos/neg)
Multi-class, single-label classification	softmax	categorical_crossentropy	MNIST has 10 classes single label (one prediction is one digit)
Multi-class, multi-label classification	sigmoid	binary_crossentropy	News tags classification, one blog can have multiple tags
Regression to arbitrary values	None	mse	Predict house price(an integer/float point)
Regression to values between 0 and 1	sigmoid	mse or binary_crossentropy	Engine health assessment where 0 is broken, 1 is new

문제 해결 타입에 따른 손실함수와 활성화 함수의 조합

model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])

여기서 해당 손실함수를 쓰는 이유는?

다중 분류여서 손실함수로 categorical_crossentropy 이거 쓰는게 맞긴한데

sparse_categorical_crossentropy 얘는 레이블이 정수값으로 주어진 경우에 해당 손실함수를 시용하여 내부적으로 원 핫 인코딩을 수행한 후 categorical_crossentropy 이 손실함수와 같은 방식으로 작동함

따라서 해당 작업 즉 원핫인코딩을 위에서 진행했으면 손실함수를 categorical_crossentropy로 사용할 것.

위에서는 target 이 0 1 2의 정수형 데이터이므로 sparse_categorical_crossentropy 사용했음

```
# 새로운 층으로 Conv1D 층과 Dense층 추가하고 출력층 추가

new_model.add(Conv1D(64,kernel_size=3,activation='relu'))
new_model.add(MaxPooling1D(pool_size=2))
new_model.add(Flatten())
new_model.add(Dense(64,activation='relu'))
new_model.add(Dense(3,activation='softmax'))
```

여기서 각 층을 추가한 이유

Conc1D(1차원 합성곱 층)

일단 기존 train accurncy 점수가 낮았기 때문에 성능을 향상시킬 필요가 있음 그래서 Conc1D층을 하나 더 추가하여 예측점수를 올리고자 함.

MaxPooling1D (1차원 최대 풀링 층)

데이터 차원을 줄이고 중요한 특징들만 남기기 위해서 추가됨

모델이 복잡해서 차원 축소가 필요할 때, 또는

모델의 데이터가 너무 커서 더 높은 수준의 특징요약을 원할떄 사용함

Flatten (평탄화 층)

출력층에서는 1차원 벡터 형태의 입력을 필요로 하기 때문에 위에서 나온 다차원 데이터들을 1차원 벡터로 변환하기 위해 해당 층 추가

Dense (완전 연결 층)

입력 데이터 최종 분류

이렇게 층 추가해도 과대적합 문제 벗어나지 못함

해서 파인튜닝 ㄱㄱ

즉 파인 튜닝이란?

모델의 가중치 변경 예를 들어, 뉴련의 갯수 변경, 각 뉴런에 더해지는 상수값 등을 변경하거나 신경망 층을 추가해서 과대적합, 과소적합을 막아 성능을 향상시키고자 하는 행위

```
from tensorflow.keras.layersimport Dropout, BatchNormalizatio
n

new_model.add(Dropout(0.5))
new_model.add(Conv1D(64,kernel_size=3,activation='relu'))
new_model.add(BatchNormalization())
new_model.add(MaxPooling1D(pool_size=2))
new_model.add(Dropout(0.5))

new_model.add(Flatten())
new_model.add(Dense(64,activation='relu'))
new_model.add(BatchNormalization())
new_model.add(Dropout(0.5))
new_model.add(Dense(3,activation='softmax'))

new_model.summary()
```

여기서는 Dropout, BatchNormalization층을 추가함

Dropout

뉴런의 일부를 무작위로 비활성화해서 과적합을 막음

BatchNormalization Layer

입력 데이터의 분포를 정규화하여 훈련속도를 높이는데 사용됨 출력을 정규화하여 신경망이 너무 큰 가중치를 학습하지 않도록 함 이 역시 과적합을 막는데 사용됨

보통 Conv 층 다음에 배치되는 것이 일반적

해서 결국 모델의 과대적합 문제는 해결함

2. **lstm**

features 데이터에서 라벨(타깃)은 activity

타깃 데이터 레이블을 위해 labelencoder 사용

원핫 인코딩은 데이터를 0과 1로 구성된 벡터형태로 변환시키는 것

```
from collectionsimport Counter
import numpyas np

def split_sequences(sequences, n_steps):
    X, y = list(), list()# 빈 리스트를 생성하여 시퀀스 데이터와 레이블을 담을 공간을 만듦
for iin range(len(sequences)):# 전체 시퀀스 데이터를 순회
# find the end of this pattern
        end_ix = i + n_steps# 현재 인덱스(i)에서 n_steps만큼 떨어진 시퀀스의 끝을 계산
# check if we are beyond the dataset
if end_ix > len(sequences):# 시퀀스 끝이 데이터의 범위를 넘어서는지 확인
break# 범위를 넘으면 루프 종료
# gather input (X) and output parts (y)
        seq_x = sequences[i:end_ix, :-1]# 입력 데이터 (특징 데이터)
        seq_y_values = sequences[i:end_ix, -1]# 시퀀스 동안의 출력 데이터 (레이블들)
```

```
# 가장 빈번하게 나온 레이블 찾기
        most_common_label = Counter(seq_y_values).most_common
(1)[0][0]

        X.append(seq_x)# 입력 데이터 추가
        y.append(most_common_label)# 가장 많이 나온 레이블 추가

return np.array(X), np.array(y)# 리스트를 numpy 배열로 변환하여 반
환
```

이 데이터는 시계열 데이터로 슬라이딩 윈도우 방식으로 시퀀스별로 탐색해서 시퀀스동안에 가장 많이 나온 레이블을 대표 레이블로 채택

이 함수에서 리턴하는 x, y 에는 최종적으로

x → 시퀀스별로 묶인 특징 데이터들

y → 각 시퀀스를 데표하는 activity 값들

이 저장되어 리턴되게 된다.

```
# 기존 모델의 층을 동결 (학습되지 않도록 설정)
base_model.trainable = False
```

여기서 층을 동결하는 이유

우리가 지금 하려는게 뭐임? 바로 LA에 대한 가중치 파일을 불러와 RA 데이터를 평가하려 함

둘의 데이터셋 구조가 비슷하므로 이미 학습된 유용한 패턴을 그대로 활용하고자 함

또한 연산 효율성 증가, 과대적합 방지 등이 있음

```
from tensorflow.keras.applicationsimport VGG16
from tensorflow.kerasimport layers, models

def remove_last_layers(model, num_layers_to_remove):
# 모델의 레이어를 하나씩 슬라이스해서 앞의 레이어만 가져옴
    model_layers = model.layers[:-num_layers_to_remove]# 마지
```

```
막 레이어부터 지정된 갯수만큼 제외
    new_model = models.Sequential(model_layers)# 새로운 모델에
해당 레이어들만 추가
return new_model


# 출력층부터 1개의 레이어를 삭제
new_model = remove_last_layers(base_model, 1)

# 새로운 출력층 추가
new_model.add(layers.Dense(256, activation='relu'))
new_model.add(layers.Dense(19, activation='softmax'))# 예: 10
개의 클래스
new_model.compile(optimizer='adam', metrics=['accuracy'], los
s='categorical_crossentropy')

# 모델 요약 출력
new_model.summary()
```

`remove_last_layers`

이 함수가 하는 역할은?

새 모델을 만들기 위해 기존의 모델의 출력층을 지우는 작업

예를 들어

기존의 모델의 구조가

LSTM

dense

dense

이렇게 되어있다 치면

```
# 출력층부터 1개의 레이어를 삭제
new_model = remove_last_layers(base_model, 1)
```

```
# 새로운 출력층 추가
new_model.add(layers.Dense(256, activation='relu'))
new_model.add(layers.Dense(19, activation='softmax'))  # 예: 10개의 클래스
new_model.compile(optimizer='adam', metrics=['accuracy'], loss='categorical_crossentropy')

# 모델 요약 출력
new_model.summary()
```

이거 하면

1. 출력층부터 한개의 레이어(dense)를 지우고
2. 뉴런 256개 자리 dense레이어 하나와

    뉴런 19개 짜리 dense 레이어 하나를 추가

결국

lstm(false) ← 기존 레이어 동결

dense(false) ← 기존 레이어 동결

dense(256)

dense(19)

이렇게 바뀜

기존 모델의 마지막 층을 제거하는 이유는?

마지막층은 출력층이므로 다른 데이터셋에 맞게 훈련되어 있음

이 데이터를 예로 들면 RA에 맞춰져 있어서 LA 데이터로 학습결과를 확인하기위해 마지막 출력층을 제거

여기서 new.model 불러오면

`base_model.trainable = false`

이 상태인듯?

마지막이 이제

동결 해제

배치 정규화, 드롭아웃 층 추가