# Functional API 활용

최석재 *lingua@naver.com*

# 다중 입력 모델

*Multi-Input Model*

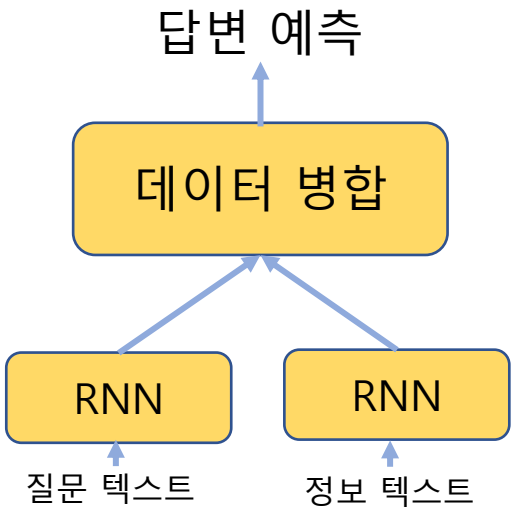
# 다중 입력 모델

- 다중 입력 모델은 둘 이상의 데이터를 받는 모델이다

- 예로 두 개의 텍스트를 받아 하나의 답변을 내놓는 모델 샘플을 만들어본다
- 첫 번째 텍스트는 사용자의 질문, 두 번째 텍스트는 답변에 필요한 정보
- 그러면 모델은 정보 텍스트에서 답변을 찾아 내놓는 경우이다
- 여기서는 하나의 단어만 출력하는 경우로 진행한다

- 전처리는 모두 되어 텍스트는 넘파이 배열이 된 것을 전제한다
- 넘파이 배열로 변환되었다면 텍스트와 이미지의 결합도 가능하다

- 다중 입력 상황에서는 데이터의 concatenate 과정이 중요하다



# 모델 설정

# 텍스트 크기 설정

- from keras.models import Model # 모델 클래스 임포트
- from keras import layers
- from keras import Input # Input 클래스 임포트

- text1_max_words = 10000 # text1의 단어 크기
- text2_max_words = 10000 # text2의 단어 크기
- answer_max_words = 500 # answer의 단어 크기

# text1 Input 설정

```
text1
```

maxlen

num_samples

```
array([[ 440, 7403, 6318, ..., 1043,  567, 2123],
       [5494, 8231, 7817, ..., 4748, 1219, 6650],
       [5966, 5704, 6211, ..., 2438, 4079, 8272],
       ...,
       [6689, 6020, 5743, ..., 9049, 1237, 8000],
       [1778, 7540, 6310, ..., 7988,  944, 5994],
       [9976, 6269, 4595, ..., 4371, 4351,  938]])
```

특성수 open
(maxlen == input_length)

층의 이름을 지정할 수 있음 (생략 가능)

- text1_input = Input(shape=(None,), dtype='int32', name='text1')
- embedded_text1 = layers.Embedding(text1_max_words, 64)(text1_input)
- encoded_text1 = layers.LSTM(32)(embedded_text1)

input_dim

output_dim

※ Input() 으로 입력층을 형성하고, 임베딩 층과 순환신경망 층을 연결

# text2 Input 설정

- text2_input = Input(shape=(None,), dtype='int32', name='text2')
- embedded_text2 = layers.Embedding(text2_max_words, 32)(text2_input)
- encoded_text2 = layers.LSTM(16)(embedded_text2)

※ 같은 방식으로 text2 Input 설정. Embedding의 output_dim과 LSTM의 노드수만 줄였다
(변경이 가능하다는 것을 보이기 위하여 줄임)

# 데이터 병합

- 데이터 병합을 수행하면 데이터 사이의 상관관계를 포착하여 학습할 수 있다

- concatenated = layers.Concatenate(axis=-1)([encoded_text1, encoded_text2])

※ concatenate은 두 개의 모듈을 리스트로 연결하여 병합한다
axis=-1 로서, 마지막 차원(열 쪽)을 기준으로 연결한다
concatenate 층은 병합되는 모듈이 갖는 노드의 개수를 더하여 필요한 노드를 생성한다

# concatenate 연습

- axis=-1 은 마지막 축을 기준으로 한다는 의미
- 파이썬에서 -1은 '뒤에서 첫 번째'를 가리킬 때가 있다
- axis=-1 이 기본값

- import numpy as np
- a = np.arange(10).reshape(2, 5)
- print("a", a, "₩n")

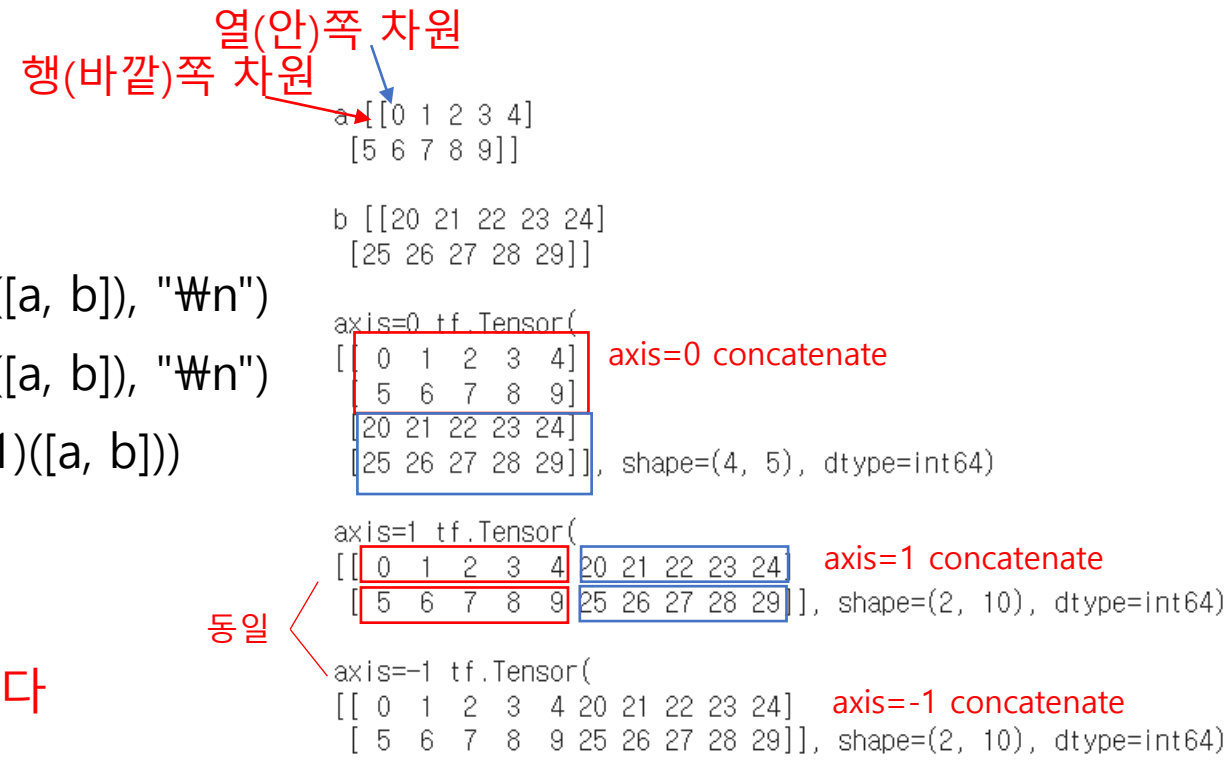
- b = np.arange(20, 30).reshape(2, 5)
- print("b", b, "₩n")

- import tensorflow as tf
- print("axis=0", tf.keras.layers.Concatenate(axis=0)([a, b]), "₩n")
- print("axis=1", tf.keras.layers.Concatenate(axis=1)([a, b]), "₩n")
- print("axis=-1", tf.keras.layers.Concatenate(axis=-1)([a, b]))

현재 1과 -1은 같은 의미

따라서 text1과 text2는 각 특성 개수만큼 나란히 붙는다

axis=0에서는 행 차원(바깥쪽 차원)을 기준으로 연결
axis=1에서는 열 차원(안쪽 차원)을 기준으로 연결
※ axis=0일 때는 두 데이터의 안쪽 차원(2nd 차원, 열) 수가 같아야 하고,
axis=1일 때는 두 데이터의 바깥쪽 차원(1st 차원, 행) 수가 같아야 한다
(현재 axis=-1은 axis=1과 같은 의미)



# 출력층 생성

- answer = layers.Dense(answer_max_words, activation='softmax')(concatenated)

출력층의 노드 수는 만들어질 수 있는 단어의 종류(500개)가 되어야 한다

※ answer의 후보 단어 중 어떤 것을 선택할 지를 알려주는 softmax 출력층 생성

# 모델 컴파일

# 2개의 입력 모듈을 리스트로 연결하여 Model의 inputs에 넣고,
# 1개의 출력 모듈을 Model의 outputs에 넣어 모델을 구성하고, 객체를 생성

- model = Model(inputs=[text1_input, text2_input], outputs=answer)
- model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['acc'])

# 모델 구조

- model.summary()

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 text1 (InputLayer)             [(None, None)]       0           []

 text2 (InputLayer)             [(None, None)]       0           []

 embedding (Embedding)          (None, None, 64)     640000      ['text1[0][0]']

 embedding_1 (Embedding)        (None, None, 32)     320000      ['text2[0][0]']

 lstm (LSTM)                    (None, 32)           12416       ['embedding[0][0]']

 lstm_1 (LSTM)                  (None, 16)           3136        ['embedding_1[0][0]']

 concatenate (Concatenate)      (None, 48)           0           ['lstm[0][0]',
                                                                  'lstm_1[0][0]']

 dense (Dense)                  (None, 500)          24500       ['concatenate[0][0]']

==================================================================================================
Total params: 1,000,052
Trainable params: 1,000,052
Non-trainable params: 0
__________________________________________________________________________________________________
```

(batch size, input_length)

(batch size, input_length, output_dim)

axis=1 연결이므로
행의 수만 같으면 됨

현재 행은 None이므로
입력시 동일한 개수의
샘플만 입력하면 됨

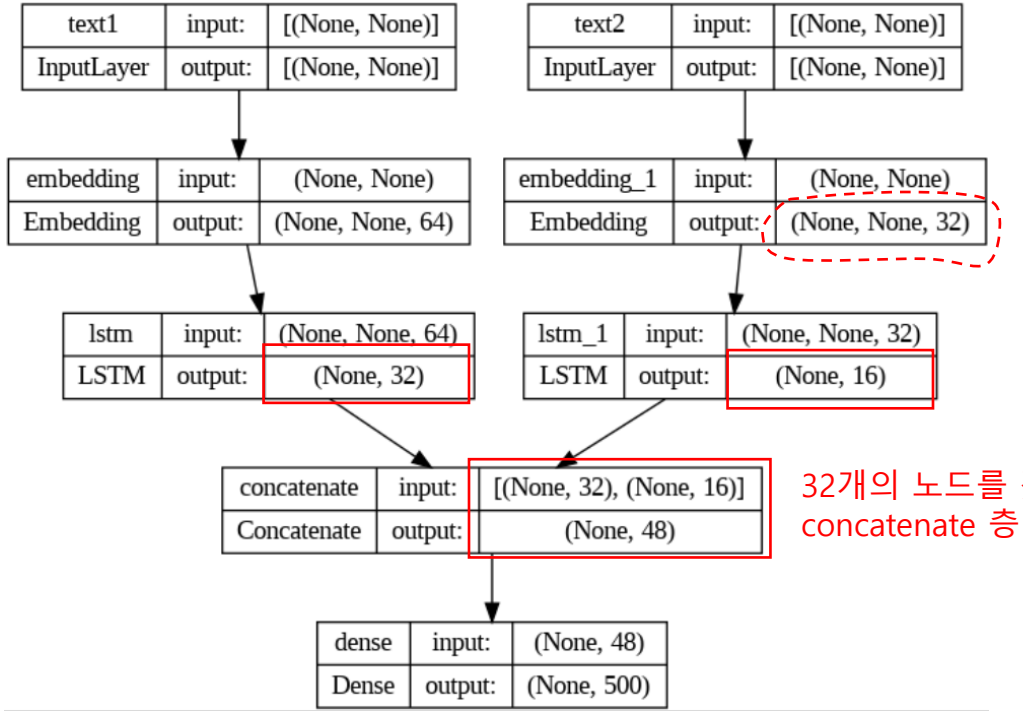
두 개의 입력 모듈과 연결됨

# 모델 그래프

# 다중 입력, 다중 출력 모델은 그래프를 그리는 것이 도움이 된다

※ Colab에서는 추가 설치 없이 그래프가 그려진다

- from tensorflow.keras.utils import plot_model
- plot_model(model, show_shapes=True, show_layer_names=True)




(None, None, 32)
(batch_size, input_length, feature_dim)을 의미
batch_size와 input_length는 open되었다

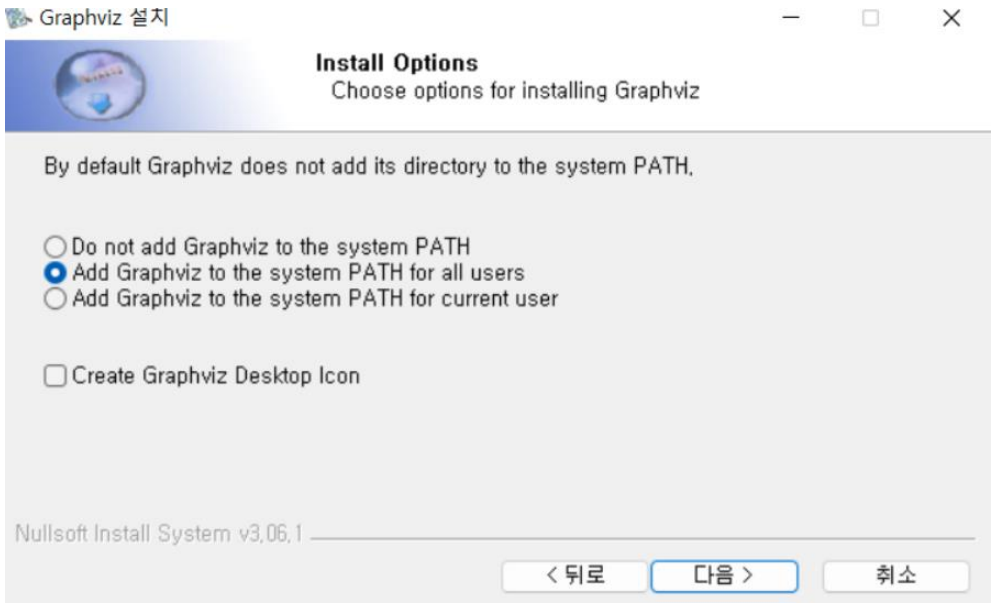
text1은 32개의 노드로, text2는 16개의 노드로 나온다

32개의 노드를 갖는 text1과, 16개의 노드를 갖는 text2가 병합되면,
concatenate 층이 필요한 노드는 두 모듈을 더한 48개로 **자동 확정**된다

# 모델 그래프 도구 설치 (Local PC)

- pip install pydot
- pip install graphviz
- https://gitlab.com/api/v4/projects/4207231/packages/generic/graphviz-releases/6.0.2/windows_10_cmake_Release_graphviz-install-6.0.2-win64.exe

※ 코드는 Jupyter 또는 VS code에서 실행한다




진행하면서 path를 all users가 볼 수 있도록 하는 옵션 선택

# 데이터 입력

- import numpy as np
- from keras.utils import to_categorical

- num_samples = 1000 # 전체 샘플의 수
- max_len = 100 # 각 문장의 길이

# 입력 데이터 생성

# 텍스트 데이터가 넘파이 배열로까지 변환되었다고 가정하고, 이를 랜덤함수로 생성한다

# 2차원 데이터로 만든다

# 두 데이터의 입력되는 샘플의 수는 같아야 한다

최솟값 최댓값

- text1 = np.random.randint(low=0, high=text1_max_words, size=(num_samples, max_len))

숫자개수

- text2 = np.random.randint(low=0, high=text2_max_words, size=(num_samples, max_len))

text1, text2 두 모듈은 각각 LSTM(32), LSTM(16)을 가지고 있어 그 결과가 (None, 32) (None, 16)로 나오는데, axis=1일 때는 바깥쪽(행쪽) 차원 수가 같으면 되므로 구조적으로 문제 없다.
입력되는 샘플의 양(행)인 num_samples의 개수는 같아야 한다
그러나 maxlen(열)은 달라도 되며, text2가 기사문이므로 이것의 maxlen은 200 정도도 좋겠다(현재는 동일)

```
text1
```

```
array([[ 440, 7403, 6318, ..., 1043,  567, 2123],
       [5494, 8231, 7817, ..., 4748, 1219, 6650],
       [5966, 5704, 6211, ..., 2438, 4079, 8272],
       ...,
       [6689, 6020, 5743, ..., 9049, 1237, 8000],
       [1778, 7540, 6310, ..., 7988,  944, 5994],
       [9976, 6269, 4595, ..., 4371, 4351,  938]])
```

```
text2
```

```
array([[8020, 3287, 2929, ..., 1634, 2655, 4446],
       [7327, 5966, 9255, ..., 7480, 7378, 9861],
       [7673,  489, 2454, ...,  496, 1441,  144],
       ...,
       [6769, 5928, 4049, ..., 4060, 5295,  966],
       [1551, 3281, 2082, ..., 8274, 8941, 5335],
       [6710, 6792, 4117, ..., 2863, 4357, 4715]])
```

# 답변 데이터 생성

# 생성된 배열을 다중 분류의 답변 데이터가 될 수 있도록 원-핫 인코딩한다

- answers = np.random.randint(low=0, high=answer_max_words, size=num_samples)
- answers = to_categorical(answers)

```
answers
```

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]], dtype=float32)
```

# Error!

- 랜덤 함수로 데이터 생성 중 다음과 같은 에러가 발생할 수 있다

```
ValueError: Shapes (None, 499) and (None, 500) are incompatible
```

- answers의 생성된 숫자가 500개에 미치지 못하여 발생하는 경우로서,
  이때는 앞의 코드를 다시 실행한다

high=answer_max_words가 500이 입력되었으나, 랜덤 종류가 500이 안 된 경우로서,
이렇게 되면 answers = to_categorical(answers) 에서 500개의 슬롯이 만들어지지 못한다

그런데 출력층 생성시 tf.keras.layers.Dense(answer_max_words, activation='softmax')(concatenated)
로 500개를 쓸 것이라고 설정해놓았기 때문에 문제가 된다

# 모델 훈련

- 현재와 같이 입력 모듈에 이름(name)을 붙인 경우는 아래 첫 번째 방식처럼 딕셔너리 형태로 모델에 입력할 수 있음
- 또는 주석 처리한 두 번째 방식처럼 두 배열을 리스트로 묶어서 입력할 수 있음

x y

'층의 이름':생성한 데이터

- model.fit({'text1':text1, 'text2':text2}, answers, epochs=3, batch_size=128)
- #model.fit([text1, text2], answers, epochs=3, batch_size=128)

# 예측

- prediction = model.predict({'text1':text1, 'text2':text2})
- label = np.argmax(prediction[0])
- print(label)

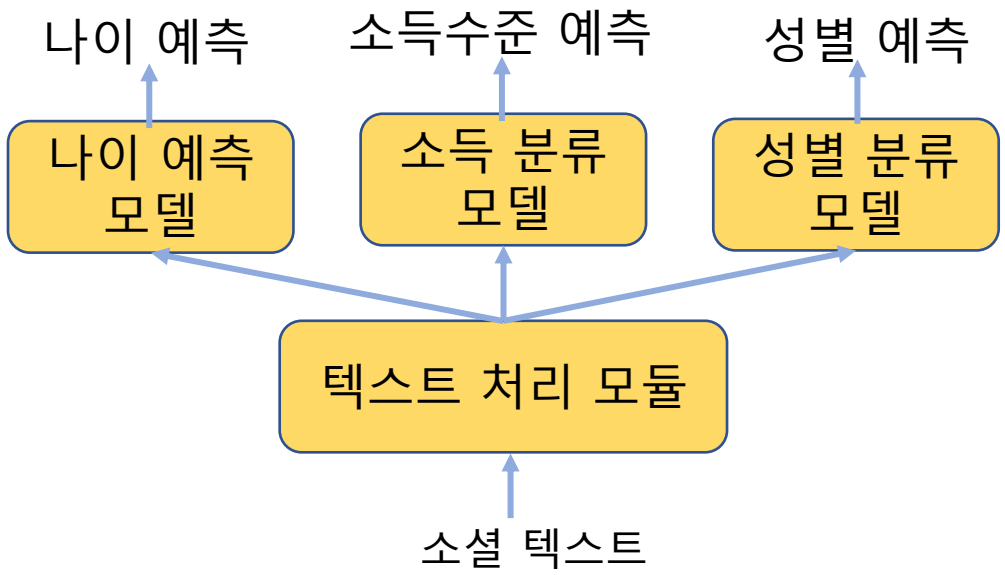
# 428. 428번 단어로 예측하였다

# 다중 출력 모델

*Multi-Output Model*

# 다중 출력 모델

- 다중 출력 모델은 둘 이상의 결과를 출력하는 모델이다
- 소셜 블로그 텍스트를 입력받아 나이, 소득수준, 성별을 예측하는 모델 샘플을 만들어본다
- 다중 출력 상황에서는 출력층 생성과 모델 컴파일 과정이 중요하다



# 모델 설정

- from keras import layers
- from keras import Input
- from keras.models import Model

- max_words = 10000 # 사용되는 어휘는 10,000개
- num_income_groups = 10 # 소득 수준 그룹은 총 10단계

# Input 설정

- posts_input = Input(shape=(None,), dtype='int32', name='posts')

※ shape=(샘플 특성수,)

현재 샘플 특성수는 None으로 열어두었다

# 중간 과정 설정

(input_dim, output_dim, input_length(생략가능))

- x = layers.Embedding(max_words, 256)(posts_input)
- x = layers.Conv1D(filters=128, kernel_size=5, activation='relu')(x)
- x = layers.MaxPooling1D(pool_size=5)(x)
- x = layers.Conv1D(filters=256, kernel_size=5, activation='relu')(x)
- x = layers.Conv1D(filters=256, kernel_size=5, activation='relu')(x)
- x = layers.MaxPooling1D(pool_size=5)(x)
- x = layers.Conv1D(filters=256, kernel_size=2, activation='relu')(x)
- x = layers.GlobalMaxPooling1D()(x)
- x = layers.Dense(128, activation='relu')(x)

GlobalMaxPooling1D는 입력된 행의 값 중 가장 큰 값 하나를 사용한다 (별도의 pool_size가 없음)
따라서 각 채널에서 최댓값 하나만 남게 되어 차원을 줄이는 효과가 있다 (예: (batch_size, 256, 32) → (batch_size, 32)
Flatten과는 다르다. Flatten은 batch_size를 제외하고는 항상 한 개의 차원만 남기며, 셀의 수는 보존된다 (위의 경우 batch_size, 256x32)

# 3개의 출력층 생성

# 출력값이 3개이므로, 출력층도 3개를 만들어야 한다
# 앞 단계의 모듈 x를 아래 세 개의 출력층에 각각 연결하여 예측값을 얻는다

# 출력층 생성
# 연령 예측
- pred_age = layers.Dense(1, name='age')(x)  # 다중 출력의 경우에는 출력층 모듈에 이름을 넣어준다

regression은 activation 함수를 사용하지 않고 출력값을 그대로 받는다

# 소득 수준 분류
- pred_income = layers.Dense(num_income_groups, activation='softmax', name='income')(x)

다중분류이므로 softmax

# 성별 예측
- pred_gender = layers.Dense(1, activation='sigmoid', name='gender')(x)

이진분류이므로 sigmoid

# 모델 객체 생성

# 하나의 input과 여러 개의 output을 갖는 모델 객체를 생성한다

- model = Model(posts_input, [pred_age, pred_income, pred_gender])

# 모델 컴파일

- model.compile(optimizer='rmsprop',
  loss={'age':'mse',
  'income':'sparse_categorical_crossentropy',
  'gender':'binary_crossentropy'})

- #model.compile(optimizer='rmsprop', loss=['mse', 'sparse_categorical_crossentropy', 'binary_crossentropy'])

- 출력이 3개이므로 손실값 설정도 3개 각각 설정해야 함
- 특히 연령은 회귀 모델, 소득은 다중 분류, 성별은 이진 분류로 훈련 방식이 서로 다름
- 손실값 설정 시 다중 입력에서와 마찬가지로 출력층 모듈에 이름을 붙였다면 딕셔너리 형태로 구성 가능

# 모델 구조

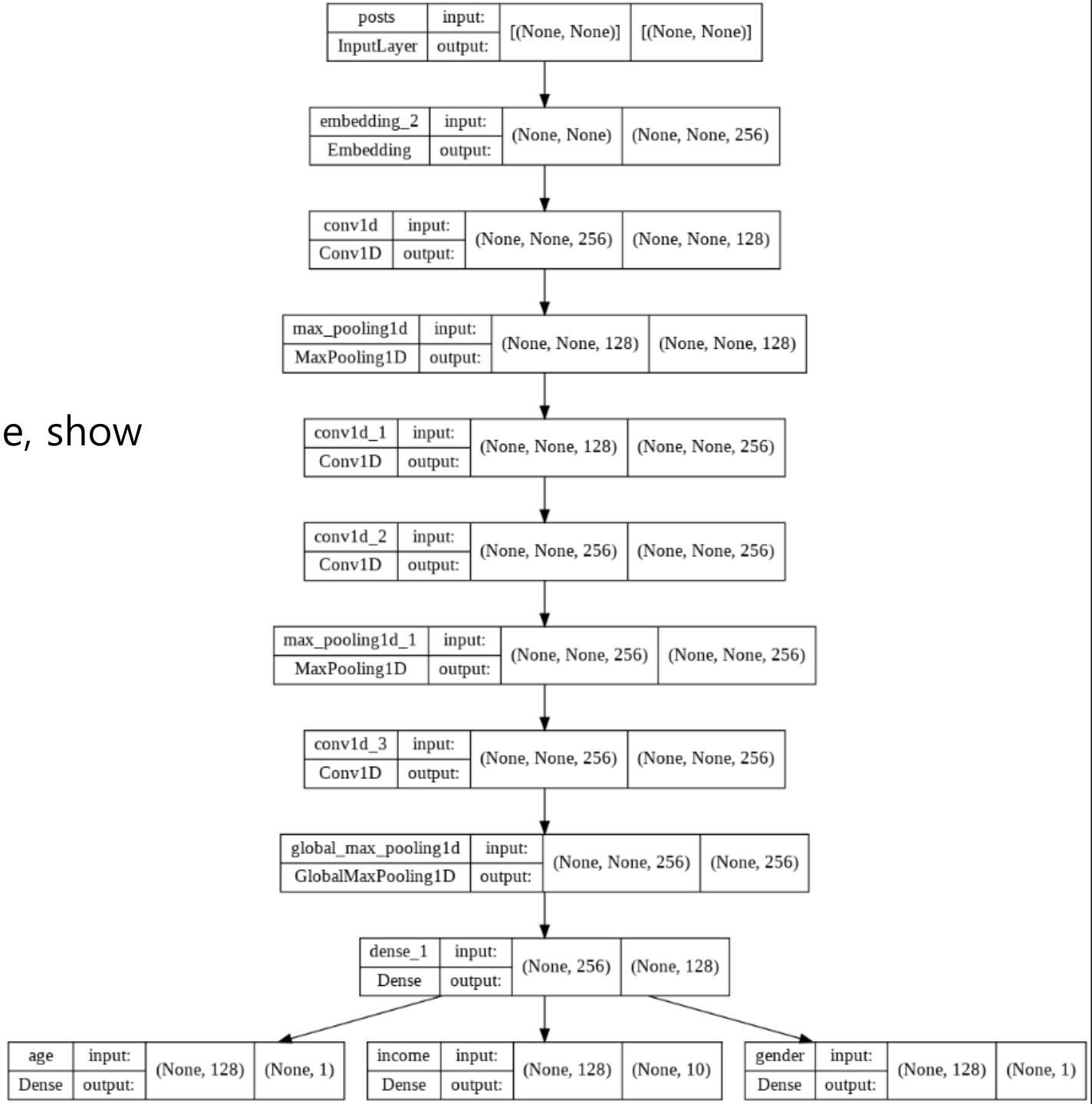
- model.summary()

3개의 출력층

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 posts (InputLayer)             [(None, None)]       0           []

 embedding (Embedding)          (None, None, 256)    2560000     ['posts[0][0]']

 conv1d (Conv1D)                (None, None, 128)    163968      ['embedding[0][0]']

 max_pooling1d (MaxPooling1D)   (None, None, 128)    0           ['conv1d[0][0]']

 conv1d_1 (Conv1D)              (None, None, 256)    164096      ['max_pooling1d[0][0]']

 conv1d_2 (Conv1D)              (None, None, 256)    327936      ['conv1d_1[0][0]']

 max_pooling1d_1 (MaxPooling1D)  (None, None, 256)   0           ['conv1d_2[0][0]']

 conv1d_3 (Conv1D)              (None, None, 256)    131328      ['max_pooling1d_1[0][0]']

 global_max_pooling1d (GlobalMa  (None, 256)         0           ['conv1d_3[0][0]']
 xPooling1D)

 dense (Dense)                  (None, 128)          32896       ['global_max_pooling1d[0][0]']

 age (Dense)                    (None, 1)            129         ['dense[0][0]']

 income (Dense)                 (None, 10)           1290        ['dense[0][0]']

 gender (Dense)                 (None, 1)            129         ['dense[0][0]']

==================================================================================================
Total params: 3,381,772
Trainable params: 3,381,772
Non-trainable params: 0
__________________________________________________________________________________________________
```

# 모델 그래프

- from tensorflow.keras.utils import plot_model
- plot_model(model, show_shapes=True, show_layer_names=True)



# 입력 데이터 생성

- import numpy as np
- num_samples = 1000
- max_len = 100

# posts는 입력되는 소셜 블로그. (행의 수 x 문장 길이)의 2차원 데이터
- posts = np.random.randint(low=0, high=max_words, size=(num_samples, max_len))

# 다음은 3개의 출력값
- target_age = np.random.randint(low=0, high=100, size=num_samples)  # 연령은 0~99
- target_income = np.random.randint(low=0, high=10, size=num_samples)  # 소득수준 그룹은 0~9
- target_gender = np.random.randint(low=0, high=2, size=num_samples)  # 성별은 0 또는 1

# 모델 훈련

# 데이터를 넣어 모델을 훈련한다

input

output

- model.fit(posts, {'age': target_age, 'income': target_income, 'gender': target_gender}, epochs=3, batch_size=64)

- # model.fit(posts, [target_age, target_income, target_gender], epochs=3, batch_size=64)

# 평가

# 모델에 예측할 데이터와 그 정답을 같이 넣어준다

- test_eval = model.evaluate(posts, {'age': target_age, 'income': target_income, 'gender': target_gender})

# 모델이 예측한 결과와 입력된 정답을 비교한 결과

- print('prediction model loss & acc:', test_eval)

```
32/32 [==============================] - 1s 7ms/step - loss: 142.3530 - age_loss: 139.2744 - income_loss: 2.3537 - gender_loss: 0.7249
prediction model loss & acc: [142.35302734375, 139.27439880371094, 2.3536946773529053, 0.7249264121055603]
```

앞에서부터 전체 손실값, 첫 번째 출력층 손실값, 두 번째 출력층 손실값, 세 번째 출력층 손실값

# 예측

- 여기서는 학습 데이터로 사용한 posts 를 다시 테스트 데이터로 사용한다
- result = model.predict(posts)

```
print("연령 예측 결과:₩n", result[0][:10])
```

```
연령 예측 결과:
 [[36.12848 ]
 [15.437461]
 [10.220898]
 [41.44453 ]
 [59.529785]
 [47.966187]
 [29.399721]
 [56.756836]
 [22.354914]
 [43.195774]]
```

```
print("소득 수준 예측 결과:₩n", result[1][:5])
```

```
소득 수준 예측 결과:
 [[1.08718827e-01 8.06921721e-02 9.23308283e-02 5.44497669e-02
  1.60899729e-01 1.18484266e-01 1.10405520e-01 8.79595063e-06
  1.24157481e-01 1.49852589e-01]
 [1.16228528e-01 8.96052942e-02 1.04653202e-01 8.40211734e-02
  1.31247655e-01 1.15503192e-01 1.12503558e-01 2.05997098e-03
  1.23983130e-01 1.20194271e-01]
 [1.14497706e-01 9.53648537e-02 1.06239177e-01 9.29552987e-02
  1.22997634e-01 1.13695912e-01 1.11781046e-01 7.91061483e-03
  1.19152211e-01 1.15405448e-01]
 [1.10498928e-01 7.10083172e-02 8.98717940e-02 4.88607511e-02
  1.69033125e-01 1.19179785e-01 1.10329457e-01 2.19586059e-06
  1.30647570e-01 1.50568098e-01]
 [1.03367247e-01 6.00656755e-02 7.80604407e-02 3.33039314e-02
  1.95190489e-01 1.18780650e-01 1.06210098e-01 1.86243234e-08
  1.29496485e-01 1.75524995e-01]]
```

```
print("성별 예측 결과:₩n", result[2][:10])
```

```
성별 예측 결과:
 [[0.13831967]
 [0.33755055]
 [0.39178884]
 [0.11592087]
 [0.04826087]
 [0.0886676 ]
 [0.16734406]
 [0.05087802]
 [0.2613928 ]
 [0.10740474]]
```

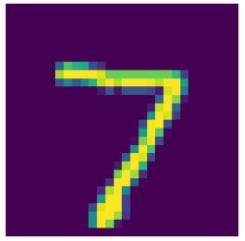
※ 1이 될 확률

# 연습문제

```
print(x_train_4d.shape, x_test_4d.shape)
```

```
(60000, 28, 28, 1) (10000, 28, 28, 1)
```

```
예상 숫자: 7
홀수 1, 짝수 0: 1

정답: 7
```



- mnist 데이터를 이용하여 이미지의 숫자와 홀짝 여부를 판별하는 다중출력 모델을 만들고자 한다
- Functional API를 이용하여 모델을 만드시오 (모델 아키텍처는 간단하게 구성)
- 전처리와 1개 데이터 구성에 유의
    - from keras.datasets import mnist
    - (train_images, train_labels), (test_images, test_labels) = mnist.load_data()

① 현재 mnist 데이터는 3D이나, 본래 이미지는 4D이므로, 현재 데이터를 4D로 만드시오

② _labels 값을 이용하여 홀수와 짝수 구분만 하는 동일한 순서의 배열을 생성하시오

③ _images 데이터를 입력으로 하고, 이미지의 숫자와 홀짝 여부를 판별하는 모델을 만드시오

④ model.evaluate() 함수를 사용하여 test 데이터를 평가하면 5개의 결과가 나오는데 각 수치가 무엇을 의미하는지 텐서플로 문서를 확인하여 파악하시오

⑤ 4D로 형변환된 test 데이터 1개를 넣어, 예측된 숫자와 홀짝 여부를 출력하시오

⑥ 입력된 이미지를 그림으로 그려 정답을 직접 확인할 수 있게 하시오

# 데이터 로딩

- import tensorflow as tf
- from keras.datasets import mnist
- import numpy as np

- (train_images, train_labels), (test_images, test_labels) = mnist.load_data()

- print(train_images.shape) # (60000, 28, 28)
- print(test_images.shape) # (10000, 28, 28)

# 홀짝 배열 생성 – train

```
# train: 홀수(1)와 짝수(0) 배열 만들기
y_train_odd = []
for y in train_labels:                    # label을 홀수와 짝수로 구분
    if y % 2 == 0:
        y_train_odd.append(0)
    else:
        y_train_odd.append(1)

y_train_odd = np.array(y_train_odd)
print("y_train_odd.shape:", y_train_odd.shape)
```

# 홀짝 배열 생성 – test

# test: 홀수(1)와 짝수(0) 배열 만들기

- y_test_odd = []
- for y in test_labels:

```
    if y % 2 == 0:
      y_test_odd.append(0)
    else:
      y_test_odd.append(1)
```

- y_test_odd = np.array(y_test_odd)
- print("y_test_odd.shape:", y_test_odd.shape)

# 전처리 결과

# 값 비교

- print("원 데이터:", train_labels[:10])
- print("홀짝 데이터", y_train_odd[:10])

```
y_train_odd.shape: (60000,)
y_test_odd.shape: (10000,)
원 데이터: [5 0 4 1 9 2 1 3 1 4]
홀짝 데이터 [1 0 0 1 1 0 1 1 1 0]
```

# 데이터 정규화

- x_train = train_images/255.0
- x_test = test_images/255.0

# 채널 추가

# 색상 데이터이므로 RGB를 표현하는 축이 별도로 필요하다
# 색상은 단색이므로 새로운 축을 추가하고, 채널 개수는 1개로 지정한다
# np.expand_dims와 tf.expand_dims는 사용법이 같다

- x_train_4d = tf.expand_dims(x_train, axis=-1)  # 새로운 축을 마지막에 추가
- x_test_4d = tf.expand_dims(x_test, -1)

- print(x_train_4d.shape, x_test_4d.shape)

```
(60000, 28, 28, 1) (10000, 28, 28, 1)
```

- from keras.models import Model
- from keras import layers
- from keras import Input

- inputs = Input(shape=(28, 28, 1), dtype='float32', name='inputs’) # input

- conv = layers.Conv2D(32, kernel_size=(3, 3), activation='relu', name='conv2d')(inputs) # hidden
- pool = layers.MaxPooling2D(pool_size=2, name='maxpooling2d')(conv)
- flat = layers.Flatten(name='flatten')(pool) # 2D 변환

- digit_outputs = layers.Dense(10, activation='softmax', name='digit_dense')(flat) # output1 (숫자)
- odd_outputs = layers.Dense(1, activation= ' sigmoid', name='odd_dense')(flat) # output2 (홀짝)

- model = Model(inputs=inputs, outputs=[digit_outputs, odd_outputs]) # Model

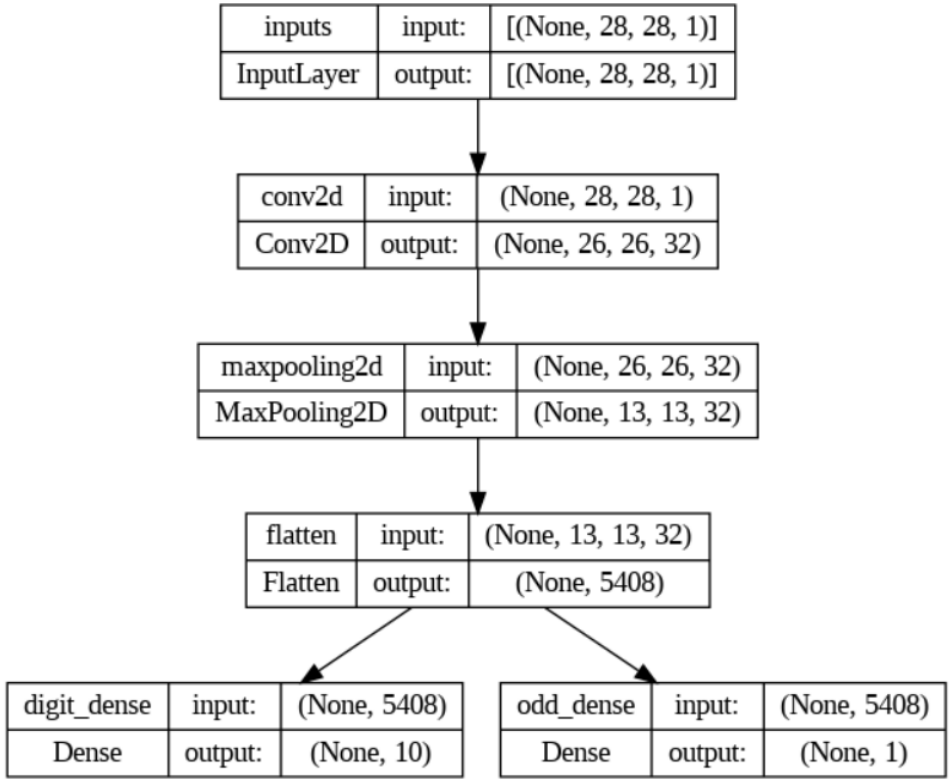
# 모델 요약

- model.summary()

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
=================================================================
 inputs (InputLayer)         [(None, 28, 28, 1)]       0         []

 conv2d (Conv2D)             (None, 26, 26, 32)        320       ['inputs[0][0]']

 maxpooling2d (MaxPooling2D  (None, 13, 13, 32)        0         ['conv2d[0][0]']
 )

 flatten (Flatten)           (None, 5408)              0         ['maxpooling2d[0][0]']

 digit_dense (Dense)         (None, 10)                54090     ['flatten[0][0]']

 odd_dense (Dense)           (None, 1)                 5409      ['flatten[0][0]']

=================================================================
Total params: 59819 (233.67 KB)
Trainable params: 59819 (233.67 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

# 모델 그래프

- from tensorflow.keras.utils import plot_model
- plot_model(model, show_shapes=True, show_layer_names=True)



# 모델 컴파일 및 훈련

# 정수를 바로 종속변수로 사용했으므로, 손실함수는 sparse_categorical_crossentropy

- model.compile(optimizer='adam', loss={'digit_dense':'sparse_categorical_crossentropy', 'odd_dense':'binary_crossentropy'}, metrics=['accuracy'])

# 모델 훈련

- history = model.fit(x_train_4d, [train_labels, y_train_odd], validation_data=(x_test_4d, [test_labels, y_test_odd]), epochs=10, batch_size=64)

```
Epoch 1/10
938/938 [==============================] - 7s 7ms/step - loss: 0.4526 - digit_dense_loss: 0.2540 - odd_dense_loss: 0.1987 - digit_dense_accuracy: 0.9302 - odd_dense_accuracy: 0.9224 - val_loss: 0
Epoch 2/10
938/938 [==============================] - 5s 5ms/step - loss: 0.1890 - digit_dense_loss: 0.0882 - odd_dense_loss: 0.1008 - digit_dense_accuracy: 0.9750 - odd_dense_accuracy: 0.9655 - val_loss: 0
Epoch 3/10
938/938 [==============================] - 5s 5ms/step - loss: 0.1525 - digit_dense_loss: 0.0669 - odd_dense_loss: 0.0856 - digit_dense_accuracy: 0.9801 - odd_dense_accuracy: 0.9703 - val_loss: 0
Epoch 4/10
938/938 [==============================] - 5s 6ms/step - loss: 0.1339 - digit_dense_loss: 0.0558 - odd_dense_loss: 0.0780 - digit_dense_accuracy: 0.9837 - odd_dense_accuracy: 0.9732 - val_loss: 0
Epoch 5/10
938/938 [==============================] - 5s 6ms/step - loss: 0.1214 - digit_dense_loss: 0.0489 - odd_dense_loss: 0.0725 - digit_dense_accuracy: 0.9853 - odd_dense_accuracy: 0.9756 - val_loss: 0
Epoch 6/10
938/938 [==============================] - 9s 9ms/step - loss: 0.1106 - digit_dense_loss: 0.0425 - odd_dense_loss: 0.0681 - digit_dense_accuracy: 0.9877 - odd_dense_accuracy: 0.9764 - val_loss: 0
Epoch 7/10
938/938 [==============================] - 6s 6ms/step - loss: 0.1009 - digit_dense_loss: 0.0376 - odd_dense_loss: 0.0633 - digit_dense_accuracy: 0.9883 - odd_dense_accuracy: 0.9783 - val_loss: 0
Epoch 8/10
938/938 [==============================] - 6s 7ms/step - loss: 0.0930 - digit_dense_loss: 0.0334 - odd_dense_loss: 0.0596 - digit_dense_accuracy: 0.9901 - odd_dense_accuracy: 0.9797 - val_loss: 0
Epoch 9/10
938/938 [==============================] - 4s 5ms/step - loss: 0.0874 - digit_dense_loss: 0.0301 - odd_dense_loss: 0.0573 - digit_dense_accuracy: 0.9912 - odd_dense_accuracy: 0.9804 - val_loss: 0
Epoch 10/10
938/938 [==============================] - 5s 6ms/step - loss: 0.0809 - digit_dense_loss: 0.0259 - odd_dense_loss: 0.0550 - digit_dense_accuracy: 0.9921 - odd_dense_accuracy: 0.9811 - val_loss: 0
```

# 성능 평가

- result = model.evaluate(x_test_4d, [test_labels, y_test_odd])

```
313/313 [==============================] - 1s 3ms/step - loss: 0.1053 - digit_dense_loss: 0.0496 - odd_dense_loss: 0.0557 - digit_dense_accuracy: 0.9852 - odd_dense_accuracy: 0.9814
```

# 성능 확인

- print("전체 loss:", result[0])
- print("digit_dense_loss:", result[1])
- print("odd_dense_loss:", result[2])
- print("digit_dense_accuracy:", result[3])
- print("odd_dense_accuracy:", result[4])

```
전체 loss: 0.10526323318481445
digit_dense_loss: 0.04960251599550247
odd_dense_loss: 0.05566073581576347 4
digit_dense_accuracy: 0.9851999878883362
odd_dense_accuracy: 0.9814000129699707
```

# 성능 확인

- print(model.metrics_names)
- print(history.history['odd_dense_accuracy'])

```
['loss', 'digit_dense_loss', 'odd_dense_loss', 'digit_dense_accuracy', 'odd_dense_accuracy']
[0.92236667871475222, 0.9655333161354065, 0.9703166484832764, 0.9732166528701782, 0.975600004196167, 0.9763833284378052, 0.9782834
```

# 1개 데이터 예측

- print("x_test_in[0]의 Dimension:", x_test_4d[0].ndim)
- print("x_test_in[0:1]의 Dimension:", x_test_4d[0:1].ndim)  # 4D 데이터 접근 (include:exclude)

- result = model.predict(x_test_4d[0:1])
- print("예상 숫자:", np.argmax(result[0]))
- print("홀수 1, 짝수 0:", (result[1][0] > 0.5).astype(np.int64)[0])

- print("₩n정답:", test_labels[0])

```
x_test_in[0]의 Dimension: 3
x_test_in[0:1]의 Dimension: 4
1/1 [==============================] - 0s 133ms/step
예상 숫자: 7
홀수 1, 짝수 0: 1

정답: 7
```

# 숫자 확인

- import matplotlib.pyplot as plt

- plt.figure(figsize=(5,5))
- plt.imshow(tf.squeeze(x_test_4d[0], axis=-1))  # 차원 축소 후 그리기
- plt.axis('off')
- plt.show()