AI Programming

Lab 04: LAB_04_AlexNet_dist

소속: 컴퓨터정보공학부

학번: 2019202103

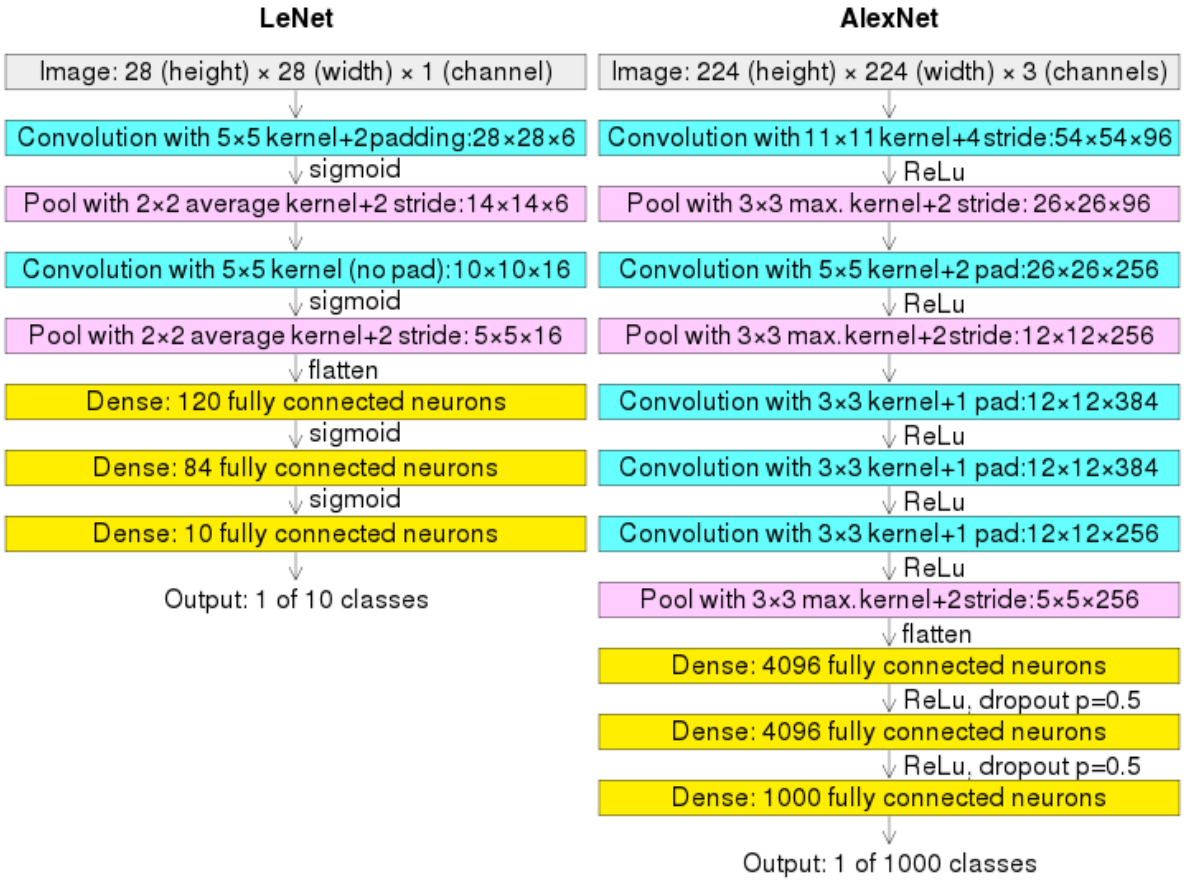
이름: 이은비

제출 날짜: 2023/10/19

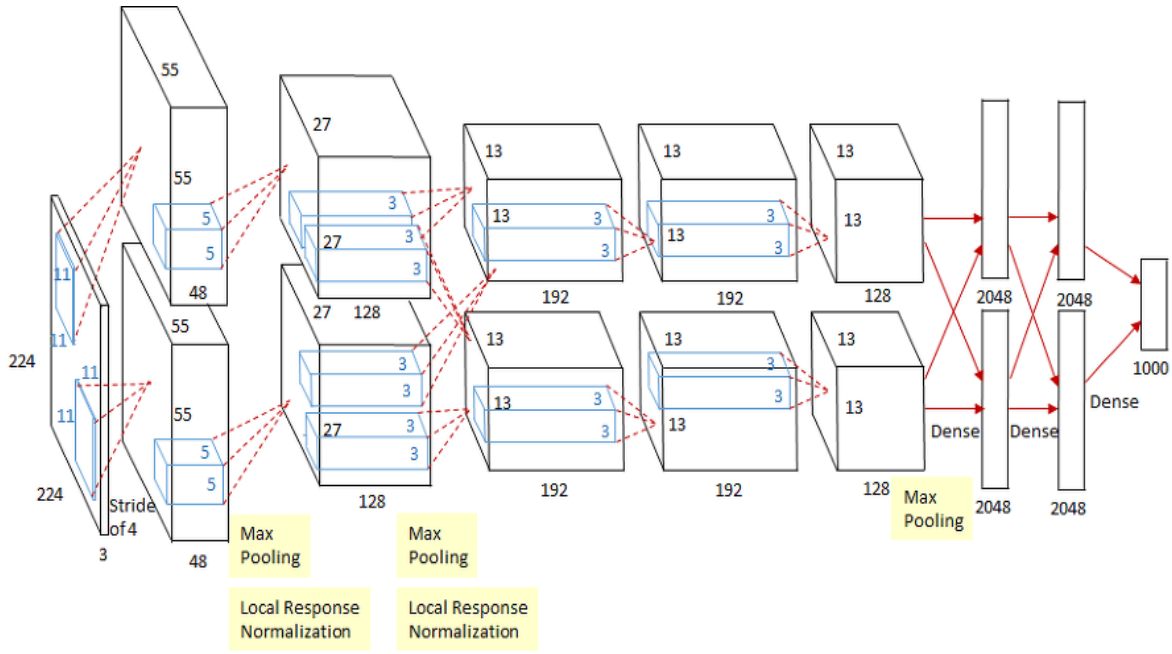
<Lab Objective>

‘cifar10’ dataset을 이용하며 AlexNet이라는 CNN구조의 layer를 AlexNet에 맞게 전체 network를 definition하고 Training합니다. 그리고 Convergence Graph을 plot하는 등 loss와 accuracy를 확인합니다. 그리고 위와 같은 과정에서 쓰이는 AlexNet은 기본적으로 CNN구조를 갖추고 있으며 CNN의 가장 간단한 구조 중의 하나인 LeNet-5와 비교할 수 있으므로 두가지의 구조 모두 알아보면 다음과 같습니다. LeNet에 비교해서 AlexNet은 layer가 전체적으로 증가한 것과 더불어 input으로 들어오는 dataset이 다르며 Image의 사이즈는 물론 그에 따른 convolution에 쓰이는 kernel등의 parameters 또한 다른 것을 알 수 있습니다.

그리고 activation 또한 ReLU로 다르게 적용하고 있으며 LeNet과 다르게 flatten과 dropout도 적용하여서 Output을 도출하는 것을 알 수 있습니다.



그리고 AlexNet의 각 layer마다의 작업과 특징을 보면 AlexNet은 8개의 레이어로 구성되어 있으며 5개의 convolution과 3개의 full-connected layer로 구성되어 있습니다.



2번째, 4번째, 5번째 convolution layers는 전 단계의 같은 채널의 feature-map과 연결되어 있지만 위의 그림과 같이 3번째 convolution layer는 전 단계의 두 채널의 feature-map들과 모두 연결되어 있다는 것을 알 수 있습니다. 그리고 각 layer마다의 각 작업이 수행되는지 살펴보면 input으로는 width x height x channel 이 227 x 227 x 3 으로 구성되어 있으며 output으로 1000개의 class를 도출하는 것을 알 수 있습니다. 이처럼 AlexNet구조를 확인하고 layer를 쌓는 과정을 잘 처리하도록 합니다.

<Whole simulation program flow>

'cifar10' dataset을 이용하며 AlexNet이라는 CNN구조의 layer를 AlexNet에 맞게 Conv2D, batch_normalization, max_pooling2d, flatten, dense와 dropout을 이용하여 각 layer에 적절하게 배치해서 전체 network를 definition하고, Training합니다. 그리고 그 과정을 좀 더 구체적으로 설명하면 1) 첫번째 레이어(컨볼루션 레이어): 96개의 11 x 11사이즈 필터커널로 입력 영상을 컨볼루션해준다. 컨볼루션 보폭(stride)를 4로 설정했고, zero-padding은 사용하지 않았습니다. zero-padding은 컨볼루션으로 인해 특성맵의 사이즈가 축소되는 것을 방지하기 위해, 또는 축소되는 정도를 줄이기 위해 영상의 가장자리 부분에 0을 추가하는 padding과정을 거칩니다. 그 결과 55 x 55 x 96 특성맵(96장의 55 x 55 사이즈 특성맵들)이 산출된 것을 알 수 있습니다. 그 다음에 ReLU 함수로 활성화해줍니다. 이어서 3 x 3 overlapping max pooling이 stride 2로 시행됩니다. 그 결과 27 x 27 x 96 특성맵을 갖게 되는 것을 알 수 있습니다. 그 다음에는 수렴 속도를 높이기 위해 local response normalization이 시행됩니다. local

response normalization은 특성맵의 차원을 변화시키지 않으므로, 특성맵의 크기는 27 x 27 x 96으로 유지됩니다.

2) 두번째 레이어(컨볼루션 레이어): 256개의 5 x 5 커널을 사용하여 전 단계의 특성맵을 컨볼루션 해줍니다. 그리고, ReLU 함수로 활성화합니다. 그 다음에 3 x 3 overlapping max pooling을 stride 2로 시행합니다. 그 결과 13 x 13 x 256 특성맵을 얻게 됩니다. 그 후 local response normalization이 시행되고, 특성맵의 크기는 13 x 13 x 256으로 그대로 유지됩니다.

3) 세번째 레이어(컨볼루션 레이어): 384개의 3 x 3 커널을 사용하여 전 단계의 특성맵을 컨볼루션 해줍니다. 이 또한 역시 ReLU 함수로 활성화합니다.

4) 네번째 레이어(컨볼루션 레이어): 384개의 3 x 3 커널을 사용해서 전 단계의 특성맵을 컨볼루션 해줍니다.이 역시 ReLU 함수로 활성화한다.

5) 다섯번째 레이어(컨볼루션 레이어): 256개의 3 x 3 커널을 사용해서 전 단계의 특성맵을 컨볼루션해줍니다. 그리고 13 x 13 x 256 특성맵(256장의 13 x 13 사이즈 특성맵들)을 얻게 됩니다. 역시 ReLU 함수로 활성화합니다. 그 다음에 3 x 3 overlapping max pooling을 stride 2로 시행합니다. 그 결과 6 x 6 x 256 특성맵을 얻게 됩니다.

6) 여섯번째 레이어(Fully connected layer): 6 x 6 x 256 feature map을 flatten해줘서 6 x 6 x 256 = 9216차원의 벡터로 만들어줍니다. 그것을 여섯번째 레이어의 4096개의 뉴런과 fully connected 해줍니다. 그 결과를 ReLU 함수로 활성화합니다.

7) 일곱번째 레이어(Fully connected layer): 4096개의 뉴런으로 구성되어 있다. 전 단계의 4096개 뉴런과 fully connected되어 있습니다. 출력 값은 ReLU 함수로 활성화됩니다.

8) 여덟번째 레이어(Fully connected layer): 1000개의 뉴런으로 구성되어 있습니다. 전 단계의 4096개 뉴런과 fully connected되어 있습니다. 1000개 뉴런의 출력 값에 softmax 함수를 적용해 1000개 클래스 각각에 속할 확률을 비교해서 그 중 1개를 predict한 결과를 출력할 수 있습니다. 그리고 Convergence Graph을 plot하여서 epoch에 따라서 loss와 val_loss를 비교하고, accuracy 와 val_accuracy 또한 비교합니다. 그리고 최종적으로 수치상의 Model Performance를 평가하고, random sample를 통한 Model 또한 Test합니다.

<Simulation results (screenshots)>

이후 학습에 사용될 병렬process를 위한 gpu를 available하게 한 뒤, gpu개수를 확인합니다.

```
# Load Libraries (set the gpu available)
import numpy as np
import tensorflow as tf
import tensorflow_datasets as tfds
import matplotlib.pyplot as plt

gpus = tf.config.list_physical_devices('GPU')
print("Num GPUs Available: ", len(gpus))
```

```
Num GPUs Available:  1
```

Training에 사용되는 dataset인 CIFAR10 dataset을 download하고, 준비된 상태가 되게 합니다.

```
Downloading and preparing dataset 162.17 MiB (download: 162.17 MiB, generated: 132.40 MiB, total: 294.58 MiB) to /root/tensorflow_datasets/cifar10/3.0.2...
Dl Completed...: 100% 1/1 [00:09<00:00, 5.84s/ url]
Dl Size...: 100% 162/162 [00:09<00:00, 33.44 MiB/s]
Extraction completed...: 100% 8/8 [00:09<00:00, 1.09 file/s]
Dataset cifar10 downloaded and prepared to /root/tensorflow_datasets/cifar10/3.0.2. Subsequent calls will reuse this data.
FeaturesDict({
    'id': Text(shape=(), dtype=string),
    'image': Image(shape=(32, 32, 3), dtype=uint8),
    'label': ClassLabel(shape=(), dtype=int64, num_classes=10),
})
{Split('train'): <SplitInfo num_examples=50000, num_shards=1>, Split('test'): <SplitInfo num_examples=10000, num_shards=1>}
50000
```

전체 dataset을 train,test,validation set으로 각각 분할한 뒤 그 개수를 확인하는 출력입니다.

```
n_train = len(ds_train)
n_test = len(ds_test)
n_val = len(ds_val)

print(n_train,n_test,n_val)
```

```
40000 10000 10000
```

Image dimension을 출력한 뒤 class의 label을 출력합니다. 그리고 이미지를 plot한 뒤 그 이미지의 predicted된 결과 text를 출력합니다.

각 layer 별로 convolution, batch_normalization, max_pooling2d, flatten, dense, dropout과 같은 함수를 이용하여 위에서 언급한 AlexNet의 구조에 맞는 layer를 작성하고 그에 대한 summary()를 통해 parmeters,shape등 정보들을 출력하여 확인 할 수 있도록 합니다.

```
# Print model summary
AlexNet.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 55, 55, 96)        34944

 batch_normalization (Batch  (None, 55, 55, 96)        384
 Normalization)

 max_pooling2d (MaxPooling2  (None, 27, 27, 96)        0
 D)

 conv2d_1 (Conv2D)           (None, 27, 27, 256)       614656

 batch_normalization_1 (Bat  (None, 27, 27, 256)       1024
 chNormalization)

 max_pooling2d_1 (MaxPoolin  (None, 13, 13, 256)       0
 g2D)

 conv2d_2 (Conv2D)           (None, 13, 13, 384)       885120

 conv2d_3 (Conv2D)           (None, 13, 13, 384)       1327488

 conv2d_4 (Conv2D)           (None, 13, 13, 256)       884992

 batch_normalization_2 (Bat  (None, 13, 13, 256)       1024
 chNormalization)

 max_pooling2d_2 (MaxPoolin  (None, 6, 6, 256)         0
 g2D)

 flatten (Flatten)           (None, 9216)              0

 dense (Dense)               (None, 4096)              37752832
```

```
 dropout_1 (Dropout)         (None, 4096)              0

 dense_2 (Dense)             (None, 10)                40970

=================================================================
Total params: 58324746 (222.49 MB)
Trainable params: 58323530 (222.49 MB)
Non-trainable params: 1216 (4.75 KB)
_________________________________________________________________
```

전체 epoch개수에 해당하는 만큼 model training을 거칩니다. 그리고 각 epoch마다의 loss와 accuracy,

그리고 각각의 val_loss와 val_accuracy에 대해서도 출력합니다.

```
n_epochs = 10

results = AlexNet.fit(dataset, epochs=n_epochs, batch_size=n_batch,
                      validation_data=valiset, validation_batch_size=n_batch,
                      verbose=1)
```

```
Epoch 1/10
625/625 [==============================] - 78s 103ms/step - loss: 2.3294 - acc: 0.2749 - val_loss: 1.9725 - val_acc: 0.2909
Epoch 2/10
625/625 [==============================] - 63s 101ms/step - loss: 1.6798 - acc: 0.3885 - val_loss: 1.6802 - val_acc: 0.4051
Epoch 3/10
625/625 [==============================] - 63s 100ms/step - loss: 1.5027 - acc: 0.4639 - val_loss: 1.7326 - val_acc: 0.3725
Epoch 4/10
625/625 [==============================] - 63s 101ms/step - loss: 1.3601 - acc: 0.5192 - val_loss: 1.4745 - val_acc: 0.4648
Epoch 5/10
625/625 [==============================] - 63s 101ms/step - loss: 1.2669 - acc: 0.5590 - val_loss: 1.3611 - val_acc: 0.5343
Epoch 6/10
625/625 [==============================] - 62s 99ms/step - loss: 1.1536 - acc: 0.6022 - val_loss: 1.2572 - val_acc: 0.5644
Epoch 7/10
625/625 [==============================] - 62s 100ms/step - loss: 1.0524 - acc: 0.6383 - val_loss: 1.1649 - val_acc: 0.5977
Epoch 8/10
625/625 [==============================] - 62s 99ms/step - loss: 0.9715 - acc: 0.6721 - val_loss: 0.9919 - val_acc: 0.6697
Epoch 9/10
625/625 [==============================] - 63s 100ms/step - loss: 0.9179 - acc: 0.6907 - val_loss: 0.9917 - val_acc: 0.6691
Epoch 10/10
625/625 [==============================] - 63s 101ms/step - loss: 0.8479 - acc: 0.7153 - val_loss: 0.9323 - val_acc: 0.6922
```

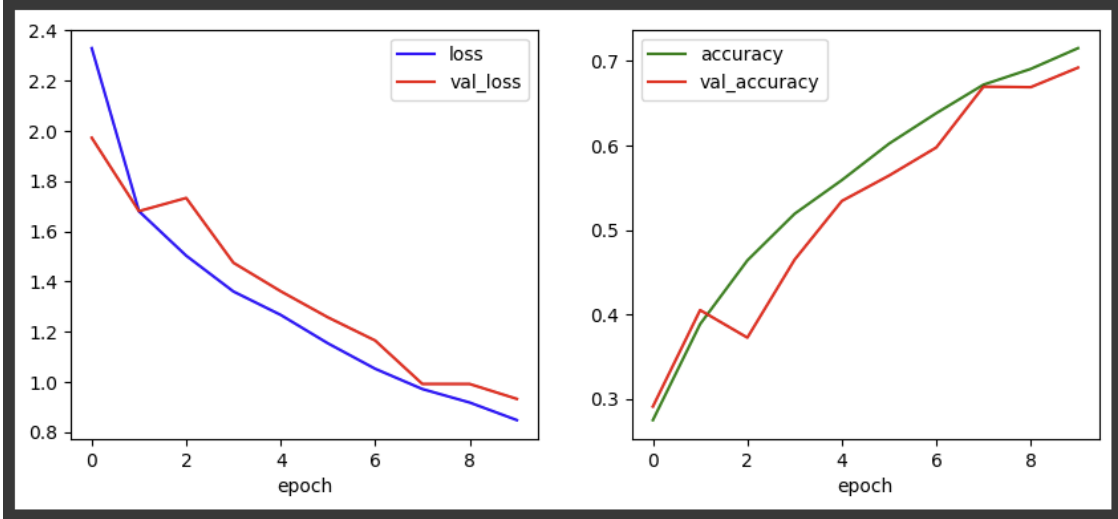
앞서 확인 및 출력한 내용인 각 epoch마다의 loss와 accuracy,

그리고 각각의 val_loss와 val_accuracy에 대해 convergence graph를 plot합니다.

```
# Plot Convergence Graph
# plot loss and accuracy
plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
plt.plot(results.history['loss'], 'b-', label='loss')
plt.plot(results.history['val_loss'], 'r-', label='val_loss')
plt.xlabel('epoch')
plt.legend()

plt.subplot(1,2,2)
plt.plot(results.history['acc'], 'g-', label='accuracy')
plt.plot(results.history['val_acc'], 'r-', label='val_accuracy')
plt.xlabel('epoch')
plt.legend()

plt.show()
```



AlexNet model의 performance 를 평가 및 출력합니다.

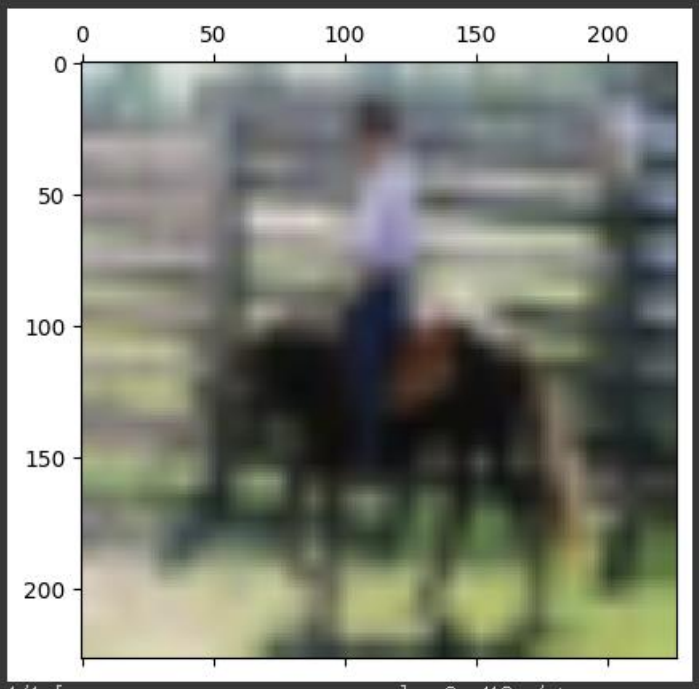
```
# Evaluate Model Performance
AlexNet.evaluate(testset)
```

```
157/157 [==============================] - 14s 85ms/step - loss: 0.9401 - acc: 0.6901
[0.9401073455810547, 0.6901000142097473]
```

Training된 model에 대해서 test하는 것으로 prediction과 actual 내용을 출력 및 확인합니다.

```
outt_4 = AlexNet.predict(X_test)
p_pred = np.argmax(outt_4, axis=-1)

print('My prediction is ' + classes[p_pred[0]])
print('Actual image is  ' + classes[tf.argmax(y_test, -1)])
```




```
1/1 [==============================] - 0s 418ms/step
My prediction is horse
Actual image is  horse
```

<Discussion>

Stride 와 padding 과 같은 parameter 에 따라서 layer 를 통과한 후의 결과가 다르게 출력되기 때문에 계산을 각각하는 과정이 필요했는데 패딩을 하지 않을 시에 합성곱의 결과 최종 이미지 크기는 (n - f + 1) x (n - f + 1) 이며(n: 이미지 한 축 크기, f: 필터 한 축 크기)

그러나 1 픽셀의 패딩을 적용하게 되면 합성곱의 결과 최종 이미지 크기는 (n + 2p - f + 1) x (n + 2p - f + 1) 가 되는 것을 알 수 있습니다. (n: 이미지 한 축 크기, p: 패딩 크기, f: 필터 한 축 크기)

그리고 이때 일반적으로 필터의 크기는 홀수를 이용하는데 홀수인 경우에만 합성곱에서 동일한 크기로 패딩을 더해줄 수 있고, 중심의 위치를 하나로 고정하는 효과가 있기 때문입니다. 그리고 스트라이드 합성곱은 합성곱 신경망의 기본구성요소로 필터 적용시 이동간격을 의미합니다. 일반적으로 필터(커널)가 input image 에서 1pixel 씩 이동하면서 연산하는데, stride 를 주게되면 필터 이동시 해당 pixel 만큼 이동하면서 연산하게 됩니다.

stride 적용시에 사용하는 식은 다음과 같습니다.

· 입력 크기=(H, W)
· 필터 크기=(FH, FW)
· 출력 크기=(OH, OW)
· 패딩=P
· 스트라이드=S

$$OH = \frac{H + 2P - FH}{S} + 1$$

$$OW = \frac{H + 2P - FW}{S} + 1$$

그리고 위와 같은 식들을 통해 각 layer 의 parameter 들을 조정하는 과정을 거쳤습니다. 그런데 이부분에서 layer 를 덜 추가해서 나중 결과에서 loss 가 끝에서 엄청 올라가고 accuracy 가 떨어지는 상황이 발생하였는데 과제로 나온 expected result 정보를 다시 확인해서 수정하여 최종적인 결과로 도출해냈습니다.