

Computer Graphics

실습 10.

2020. 11. 19

박 화 종

aqkrghkwhd@naver.com

실습 소개

- 과목 홈페이지

- 충남대학교 사이버 캠퍼스 (<http://e-learn.cnu.ac.kr>)

- TA 연락처

- 박화종
- 공대 5호관 506호 컴퓨터비전 연구실
- aqkrghkwhd@naver.com

- 실습 튜터

- 최수민(00반)
- eocjstnals12@naver.com
- 신준호(01반)
- wnsgh578@naver.com

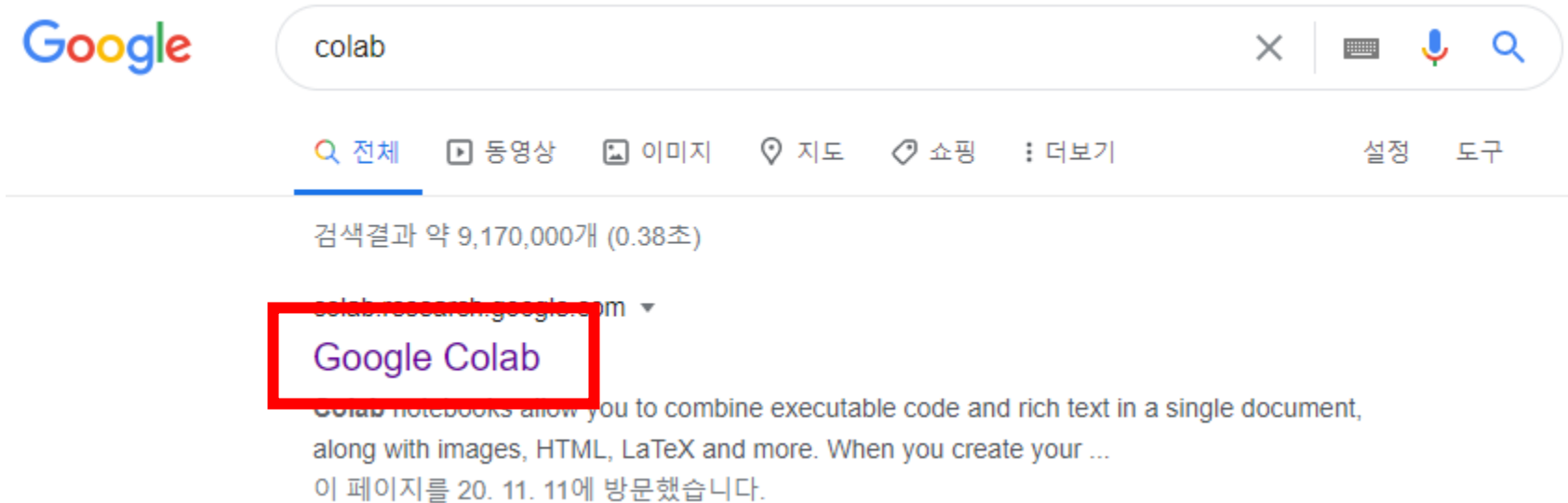
목 차

- **VGG 실습**
 - VGG16
- **ResNet 실습**
 - ResNet50
- **Inception 실습**
 - InceptionV3

VGG 실습

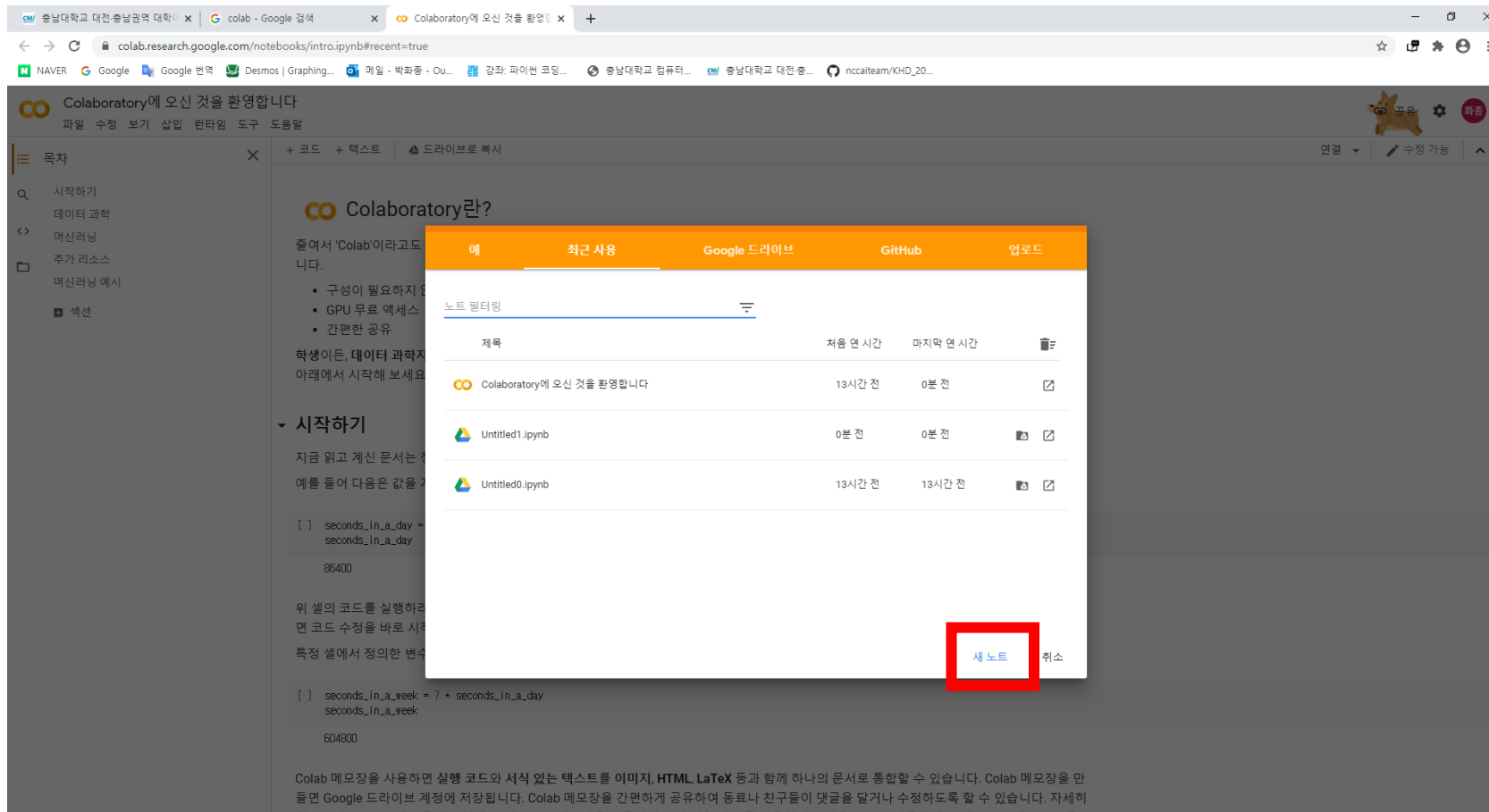
- Colab 사용하기

- 컴퓨터에 GPU가 있는 경우 Colab 사용 안해도 상관없음



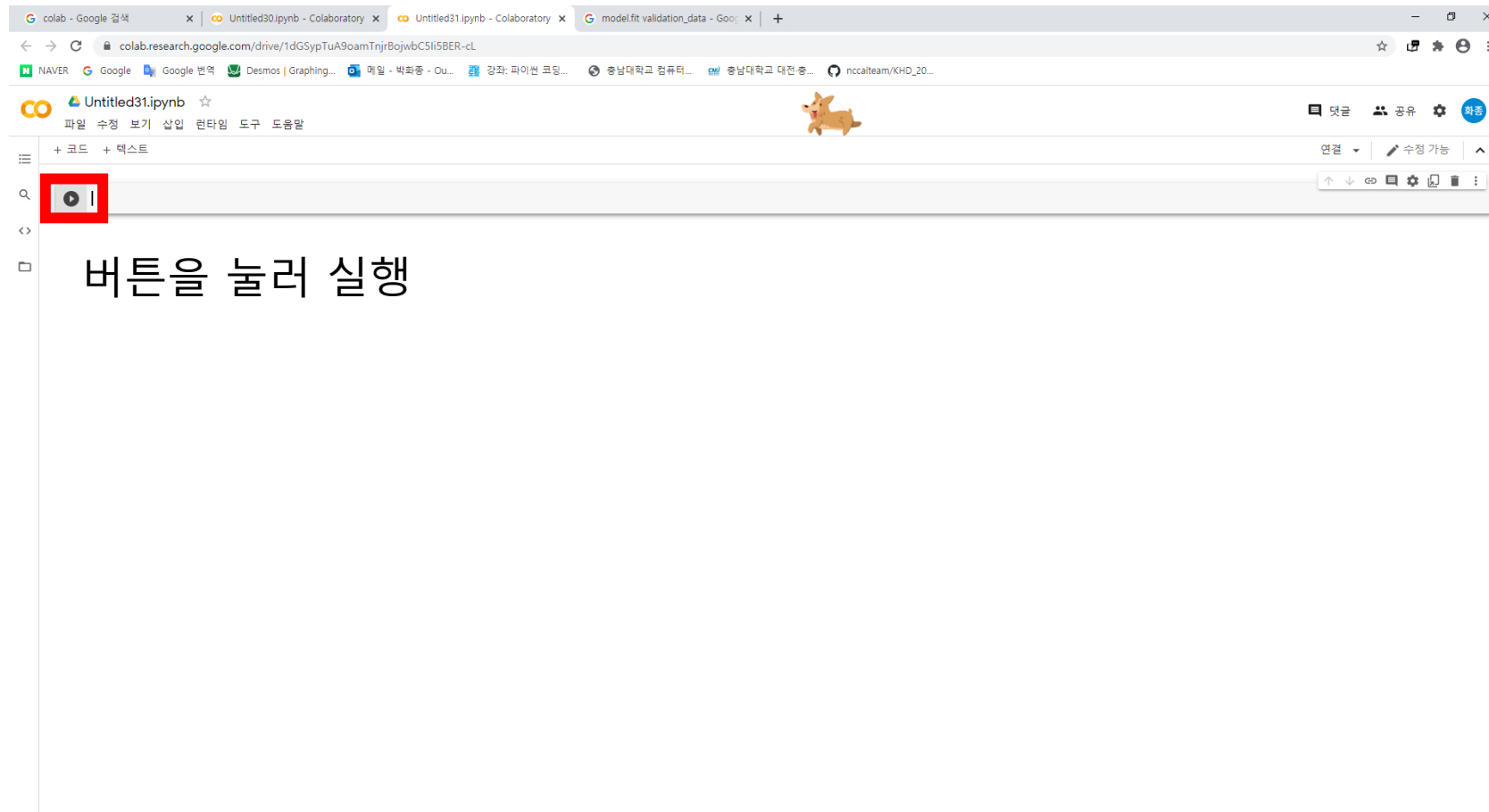
VGG 실습

• Colab 사용하기



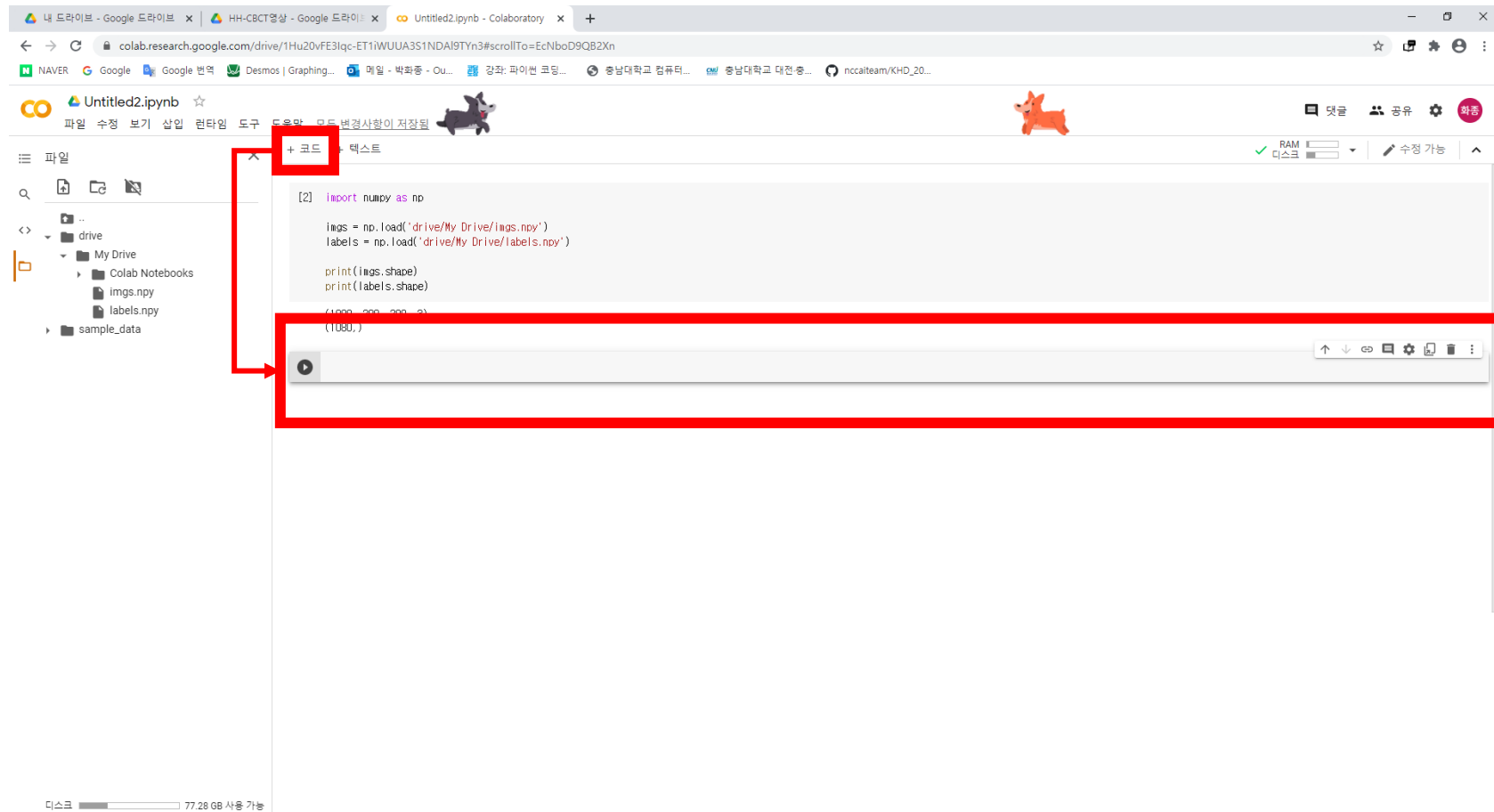
VGG 실습

- Colab 사용하기



VGG 실습

• Colab 사용하기



VGG 실습

- **Cifar-10 Dataset으로 실습**
 - Cifar-10 데이터 분류 하기

airplane



automobile



bird



cat



deer



dog



frog



horse



ship

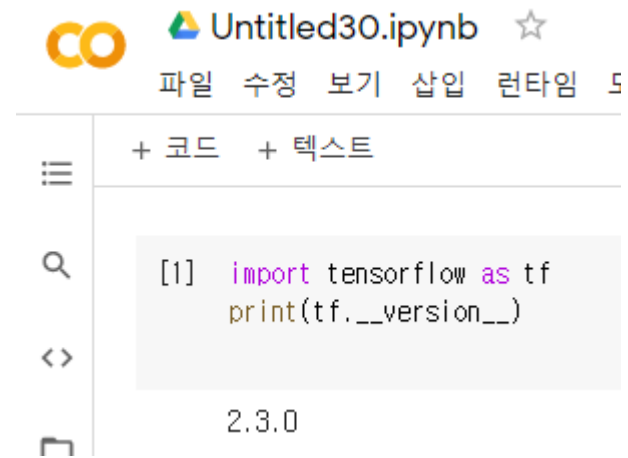


truck



VGG 실습

- Cifar-10 Dataset으로 실습



The screenshot shows a Jupyter Notebook titled 'Untitled30.ipynb'. The interface includes a top bar with the Google Colab logo and a star icon, and a menu bar with options like '파일', '수정', '보기', '삽입', '런타임', and '도움말'. Below the menu bar, there are tabs for '+ 코드' and '+ 텍스트'. The main area displays a code cell with the following code:

```
[1] import tensorflow as tf
    print(tf.__version__)
```

The output of the code cell is '2.3.0'.

이번 실습 및 과제에서 사용할
tensorflow 버전

VGG 실습

- **Cifar-10 Dataset으로 실습**
 - Dataset load

```
▶ print(y_test[:10, :])
```

```
[[3]  
[8]  
[8]  
[0]  
[6]  
[6]  
[1]  
[6]  
[3]  
[1]]
```

```
▶ (x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()  
  
print('train data')  
print(x_train.shape)  
print(y_train.shape)  
  
print('test data')  
print(x_test.shape)  
print(y_test.shape)
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz  
170500096/170498071 [=====] - 2s 0us/step  
train data  
(50000, 32, 32, 3)  
(50000, 1)  
test data  
(10000, 32, 32, 3)  
(10000, 1)
```

VGG 실습

- **Cifar-10 Dataset으로 실습**
 - VGG16 모델 만들기

```
[21] def vgg16(input_shape = (224,224,3)):
    model = tf.keras.models.Sequential()

    model.add(tf.keras.layers.Conv2D(input_shape = input_shape, filters = 64, kernel_size = (3,3), strides = (1, 1), activation = 'relu', padding = 'same'))
    model.add(tf.keras.layers.Conv2D(filters = 64, kernel_size = (3,3), strides = (1, 1), activation = 'relu', padding = 'same'))
    model.add(tf.keras.layers.MaxPooling2D(pool_size = (2,2)))

    model.add(tf.keras.layers.Conv2D(filters = 128, kernel_size = (3,3), strides = (1, 1), activation='relu', padding = 'same'))
    model.add(tf.keras.layers.Conv2D(filters = 128, kernel_size = (3,3), strides = (1, 1), activation='relu', padding = 'same'))
    model.add(tf.keras.layers.MaxPooling2D(pool_size = (2,2)))

    model.add(tf.keras.layers.Conv2D(filters = 256, kernel_size = (3,3), strides = (1, 1), activation='relu', padding = 'same'))
    model.add(tf.keras.layers.Conv2D(filters = 256, kernel_size = (3,3), strides = (1, 1), activation='relu', padding = 'same'))
    model.add(tf.keras.layers.Conv2D(filters = 256, kernel_size = (3,3), strides = (1, 1), activation='relu', padding = 'same'))
    model.add(tf.keras.layers.MaxPooling2D(pool_size = (2,2)))

    model.add(tf.keras.layers.Conv2D(filters = 512, kernel_size = (3,3), strides = (1, 1), activation='relu', padding = 'same'))
    model.add(tf.keras.layers.Conv2D(filters = 512, kernel_size = (3,3), strides = (1, 1), activation='relu', padding = 'same'))
    model.add(tf.keras.layers.Conv2D(filters = 512, kernel_size = (3,3), strides = (1, 1), activation='relu', padding = 'same'))
    model.add(tf.keras.layers.MaxPooling2D(pool_size = (2,2)))

    model.add(tf.keras.layers.Conv2D(filters = 512, kernel_size = (3,3), strides = (1, 1), activation='relu', padding = 'same'))
    model.add(tf.keras.layers.Conv2D(filters = 512, kernel_size = (3,3), strides = (1, 1), activation='relu', padding = 'same'))
    model.add(tf.keras.layers.Conv2D(filters = 512, kernel_size = (3,3), strides = (1, 1), activation='relu', padding = 'same'))
    model.add(tf.keras.layers.MaxPooling2D(pool_size = (2,2)))

    #fully connected
    model.add(tf.keras.layers.Flatten())
    model.add(tf.keras.layers.Dense(4096, activation='relu'))
    model.add(tf.keras.layers.Dense(4096, activation='relu'))
    model.add(tf.keras.layers.Dense(10, activation='softmax'))

    return model
```

원래는 output이 1000이지만
cifar-10 dataset은 10개의 class가 있으므로
10으로 변경



VGG 실습

- **Cifar-10 Dataset으로 실습**
 - VGG16 모델 만들기

```

▶ vgg_cifar10 = vgg16(input_shape = (32, 32, 3))

opt = tf.keras.optimizers.Adam(learning_rate=0.1)
vgg_cifar10.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['acc'])

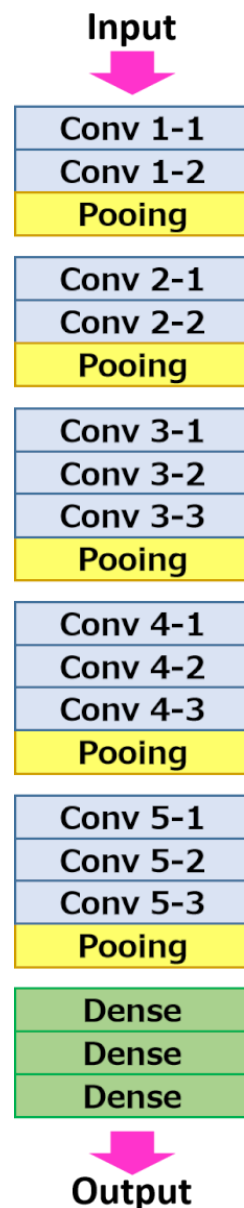
vgg_cifar10.summary()

```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
=====		
conv2d_26 (Conv2D)	(None, 32, 32, 64)	1792

conv2d_27 (Conv2D)	(None, 32, 32, 64)	36928



VGG 실습

- **Cifar-10 Dataset으로 실습**
 - VGG16 모델 만들기

Layer (type)	Output Shape	Param #
conv2d_26 (Conv2D)	(None, 32, 32, 64)	1792
conv2d_27 (Conv2D)	(None, 32, 32, 64)	36928
max_pooling2d_10 (MaxPooling)	(None, 16, 16, 64)	0
conv2d_28 (Conv2D)	(None, 16, 16, 128)	73856
conv2d_29 (Conv2D)	(None, 16, 16, 128)	147584
max_pooling2d_11 (MaxPooling)	(None, 8, 8, 128)	0
conv2d_30 (Conv2D)	(None, 8, 8, 256)	295168
conv2d_31 (Conv2D)	(None, 8, 8, 256)	590080
conv2d_32 (Conv2D)	(None, 8, 8, 256)	590080
max_pooling2d_12 (MaxPooling)	(None, 4, 4, 256)	0
conv2d_33 (Conv2D)	(None, 4, 4, 512)	1180160
conv2d_34 (Conv2D)	(None, 4, 4, 512)	2359808
conv2d_35 (Conv2D)	(None, 4, 4, 512)	2359808
max_pooling2d_13 (MaxPooling)	(None, 2, 2, 512)	0
conv2d_36 (Conv2D)	(None, 2, 2, 512)	2359808
conv2d_37 (Conv2D)	(None, 2, 2, 512)	2359808
conv2d_38 (Conv2D)	(None, 2, 2, 512)	2359808
max_pooling2d_14 (MaxPooling)	(None, 1, 1, 512)	0
flatten_2 (Flatten)	(None, 512)	0
dense_6 (Dense)	(None, 4096)	2101248
dense_7 (Dense)	(None, 4096)	16781312
dense_8 (Dense)	(None, 10)	40970



VGG 실습

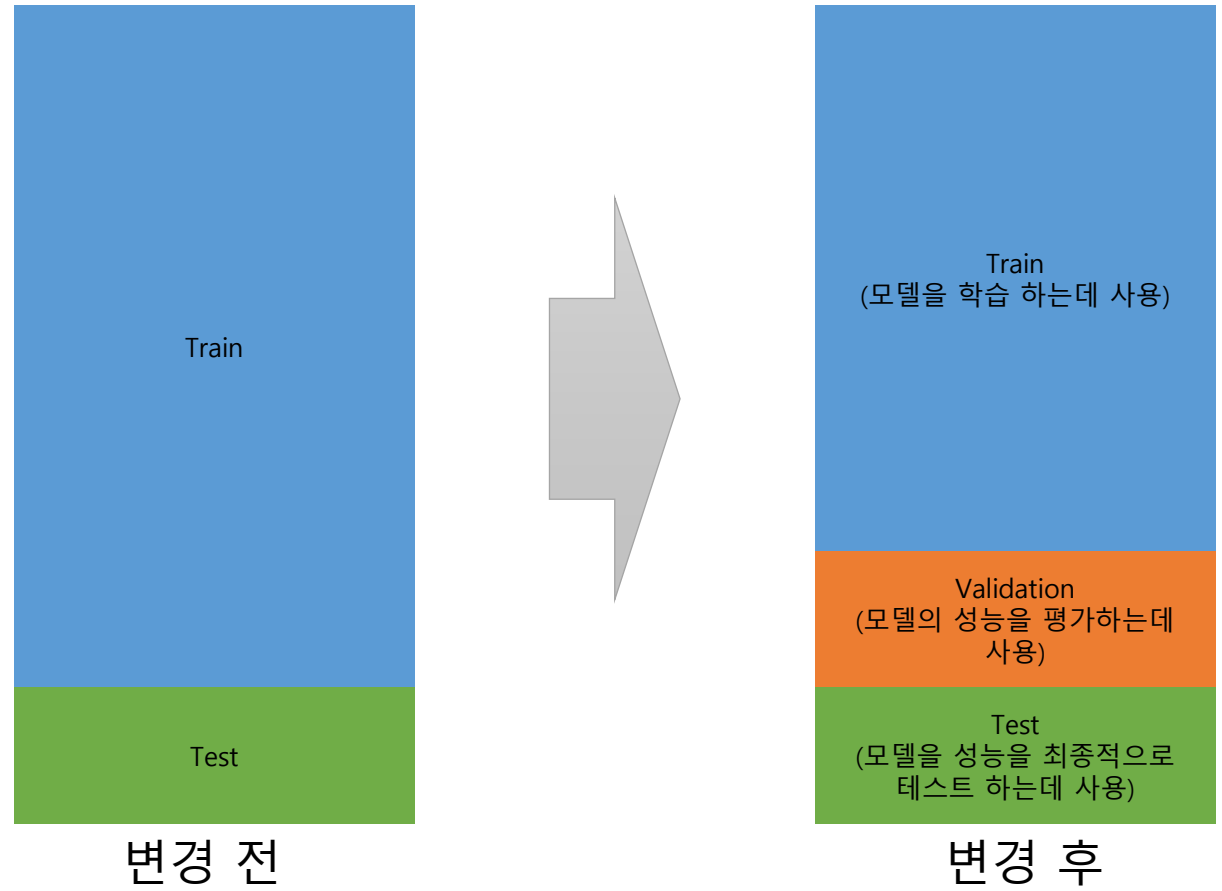
• Cifar-10 Dataset으로 실습

- 모델 학습하기
 - 데이터 나누기
 - Train
 - Validation
 - Test

```
print('train data')  
print(x_train.shape)  
print(y_train.shape)
```

```
print('test data')  
print(x_test.shape)  
print(y_test.shape)
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar10.html  
170500096/170498071 [=====]  
train data  
(50000, 32, 32, 3)  
(50000, 1)  
test data  
(10000, 32, 32, 3)  
(10000, 1)
```



Validation data를 만드는 이유는??? (이따가 설명)

VGG 실습

- **Cifar-10 Dataset으로 실습**

- 모델 학습하기
 - 데이터 나누기
 - Train
 - Validation
 - Test

Dataset
Train : 4만
Valid : 1만
Test : 1만

```
▶ from sklearn.model_selection import train_test_split

x_train, x_valid, y_train, y_valid = train_test_split(x_train, y_train, test_size=0.2, random_state=123)

print('train data')
print(x_train.shape)
print(y_train.shape)

print('validation data')
print(x_valid.shape)
print(y_valid.shape)
```

```
↳ train data
(40000, 32, 32, 3)
(40000, 1)
validation data
(10000, 32, 32, 3)
(10000, 1)
```

VGG 실습

• Cifar-10 Dataset으로 실습

- 모델 학습하기
 - 데이터 나누기
 - Train
 - Validation
 - Test

```
print(y_test[:10, :])
```

```
tf.Tensor(
[[0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]], shape=(10, 10), dtype=float32)
```

```
# scalar 형태의 레이블(0~9)을 One-hot Encoding 형태로 변환합니다.
y_train = tf.squeeze(tf.one_hot(y_train, 10),axis=1)
y_valid = tf.squeeze(tf.one_hot(y_valid, 10),axis=1)
y_test = tf.squeeze(tf.one_hot(y_test, 10),axis=1)
```

```
print('train data')
print(x_train.shape)
print(y_train.shape)
```

```
print('valid data')
print(x_valid.shape)
print(y_valid.shape)
```

```
print('test data')
print(x_test.shape)
print(y_test.shape)
```

```
train data
(40000, 32, 32, 3)
(40000, 10)
valid data
(10000, 32, 32, 3)
(10000, 10)
test data
(10000, 32, 32, 3)
(10000, 10)
```


VGG 실습

• Cifar-10 Dataset으로 실습

- 모델 학습하기
- GPU 사용하기

The screenshot shows a Google Colab notebook interface. A file menu is open, highlighting the '노트 설정' (Note Settings) option. The notebook contains Python code for loading and preprocessing CIFAR-10 data. The code includes comments in Korean and uses TensorFlow operations to load data from Google Drive, convert it to one-hot encoding, and print the shapes of the training, validation, and test datasets.

```

train_data = tf.read_file('train_data')
x_train = tf.cast(train_data, tf.float32)
y_train = tf.zeros([40000, 10])
validation_data = tf.read_file('validation_data')
x_valid = tf.cast(validation_data, tf.float32)
y_valid = tf.zeros([10000, 10])
test_data = tf.read_file('test_data')
x_test = tf.cast(test_data, tf.float32)
y_test = tf.zeros([10000, 10])

# scalar 형태의 레이블(0-9)을 one-hot Encoding 형태로 변환합니다.
y_train = tf.squeeze(tf.one_hot(y_train, 10), axis=1)
y_valid = tf.squeeze(tf.one_hot(y_valid, 10), axis=1)
y_test = tf.squeeze(tf.one_hot(y_test, 10), axis=1)

print('train data')
print(x_train.shape)
print(y_train.shape)

print('valid data')
print(x_valid.shape)
print(y_valid.shape)

print('test data')
print(x_test.shape)
print(y_test.shape)

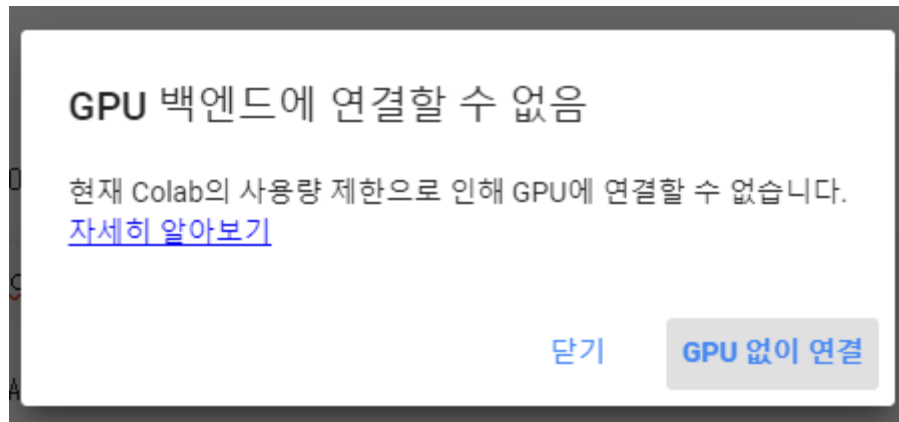
train_data
(40000, 32, 32, 3)
(40000, 10)
valid data
(10000, 32, 32, 3)
(10000, 10)
test data
(10000, 32, 32, 3)
(10000, 10)

[9] print(y_test[:10, :])

tf.Tensor(
[[0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
])
  
```


VGG 실습

- **Cifar-10 Dataset으로 실습**
 - 모델 학습하기
 - GPU 사용하기



다른 계정 이용하기
나중에 다시 하기

<중요!>
미리미리 실습 및 과제 하기
GPU 연결이 안될 수 도 있음

리소스 한도 관련

<https://research.google.com/colaboratory/faq.html#resource-limits>

VGG 실습

• Cifar-10 Dataset으로 실습

- 모델 학습하기
 - GPU 사용하기

Colab의 사용량 한도란 무엇인가요?
Colab에서는 보장된 리소스 또는 무제한 리소스를 제공하는 대신 경우에 따라 **사용량 한도를 동적으로 변경**함으로써 무료 리소스를 제공할 수 있습니다. 즉, **전체 사용량 한도, 유희 시간 제한 기간, 최대 VM 수명, 사용 가능한 GPU 유형 등의 요소가 시간에 따라 달라집니다.** Colab은 이러한 **한도를 공개하지 않는데**, 그 이유 중 하나는 한도가 빠르게 바뀔 수 있으며 실제로도 빠르게 바뀌는 경우가 있기 때문입니다.

장기간 연산을 실행하기보다는 Colab을 대화식으로 사용하거나 또는 **최근에 Colab 리소스 사용량이 비교적 적었던 사용자에게 GPU 및 TPU가 우선적으로 할당**되는 경우도 있습니다. 이로 인해 Colab을 장기적으로 실행되는 연산에 사용하거나 **최근에 Colab에서 리소스 사용량이 비교적 많았던 사용자**는 **사용량 한도에 도달하여 GPU 및 TPU 액세스가 일시적으로 제한**될 가능성이 높습니다. 많은 연산이 필요한 사용자는 Colab UI를 자체 하드웨어에서 실행하는 로컬 런타임으로 사용하는 것이 적합할 수 있습니다. 더 높으면서 안정적인 사용량 한도에 관심이 있다면 Colab Pro가 적합할 수 있습니다.

리소스 한도 관련

<https://research.google.com/colaboratory/faq.html#resource-limits>

VGG 실습

- **Cifar-10 Dataset으로 실습**

- 모델 학습하기
 - GPU 사용하기

Colab에서 메모장을 얼마나 오래 실행할 수 있나요?

메모장은 최대 수명이 12시간인 가상 머신에 연결되어 실행됩니다. 유휴 상태가 너무 오래 지속되면 메모장의 VM 연결이 해제됩니다. 최대 VM 수명 및 유휴 시간 제한 동작은 시간 또는 사용량에 따라 달라질 수 있습니다.

Colab에서는 메모리를 얼마나 사용할 수 있나요?

Colab 가상 머신에서 사용할 수 있는 메모리 양은 시간에 따라 달라집니다(VM 수명 내에는 안정적임). 시간에 따라 메모리를 조정하므로 Colab을 계속 무료로 제공할 수 있습니다. Colab에서 추가 메모리가 필요하다고 판단되면 메모리가 더 많은 VM에 자동으로 할당될 수도 있습니다. Colab에서 더 많은 메모리를 더 안정적으로 사용하는 데 관심이 있다면 Colab Pro가 적합할 수 있습니다.

리소스 한도 관련

<https://research.google.com/colaboratory/faq.html#resource-limits>

VGG 실습

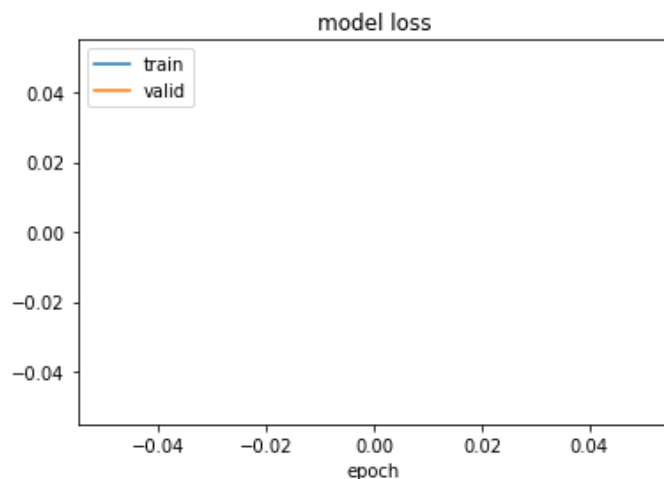
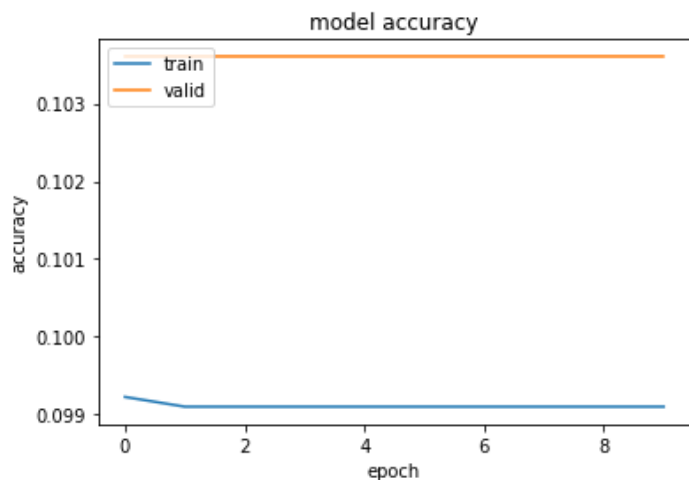
- **Cifar-10 Dataset으로 실습**
 - 모델 학습하기

```
[9] history = vgg_cifar10.fit(x_train, y_train, batch_size=32, epochs=10, validation_data=(x_valid, y_valid))
```

```
Epoch 1/10
  1/1250 [.....] - ETA: 0s - loss: 2.3023 - acc: 0.1562WARNING:tensorflow:Callbacks method `on_t
1250/1250 [=====] - 43s 35ms/step - loss: nan - acc: 0.0992 - val_loss: nan - val_acc: 0.1036
Epoch 2/10
1250/1250 [=====] - 44s 35ms/step - loss: nan - acc: 0.0991 - val_loss: nan - val_acc: 0.1036
Epoch 3/10
1250/1250 [=====] - 44s 35ms/step - loss: nan - acc: 0.0991 - val_loss: nan - val_acc: 0.1036
Epoch 4/10
1250/1250 [=====] - 44s 35ms/step - loss: nan - acc: 0.0991 - val_loss: nan - val_acc: 0.1036
Epoch 5/10
1250/1250 [=====] - 44s 35ms/step - loss: nan - acc: 0.0991 - val_loss: nan - val_acc: 0.1036
Epoch 6/10
1250/1250 [=====] - 44s 35ms/step - loss: nan - acc: 0.0991 - val_loss: nan - val_acc: 0.1036
Epoch 7/10
1250/1250 [=====] - 44s 35ms/step - loss: nan - acc: 0.0991 - val_loss: nan - val_acc: 0.1036
Epoch 8/10
1250/1250 [=====] - 44s 35ms/step - loss: nan - acc: 0.0991 - val_loss: nan - val_acc: 0.1036
Epoch 9/10
1250/1250 [=====] - 44s 35ms/step - loss: nan - acc: 0.0991 - val_loss: nan - val_acc: 0.1036
Epoch 10/10
1250/1250 [=====] - 44s 35ms/step - loss: nan - acc: 0.0991 - val_loss: nan - val_acc: 0.1036
```

VGG 실습

- **Cifar-10 Dataset으로 실습**
 - 모델 학습하기



학습이 제대로 진행되지 않음

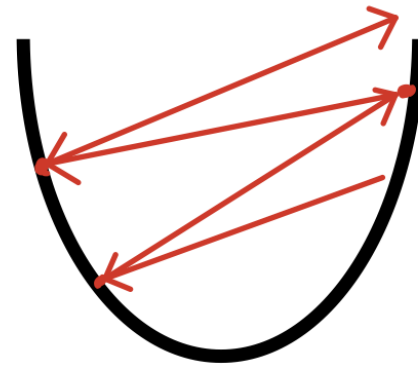
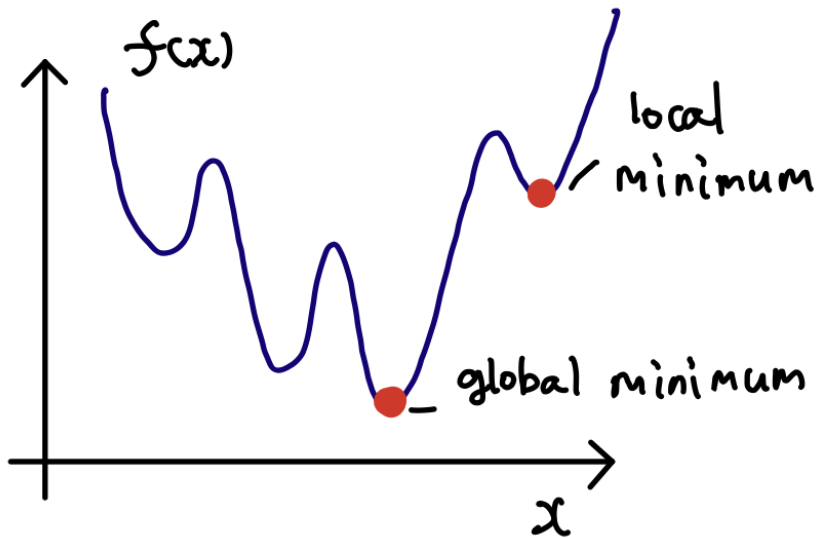
```
import matplotlib.pyplot as plt

plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'valid'], loc='upper left')
plt.show()

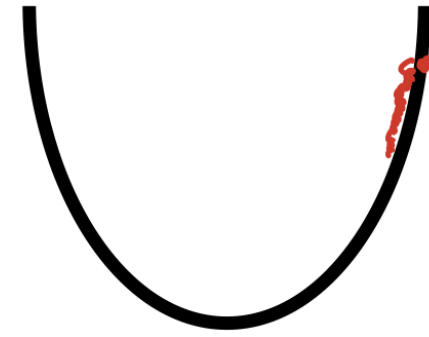
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.xlabel('epoch')
plt.legend(['train', 'valid'], loc='upper left')
plt.show()
```

VGG 실습

- **Cifar-10 Dataset으로 실습**
 - 모델 학습하기
 - learning rate란?



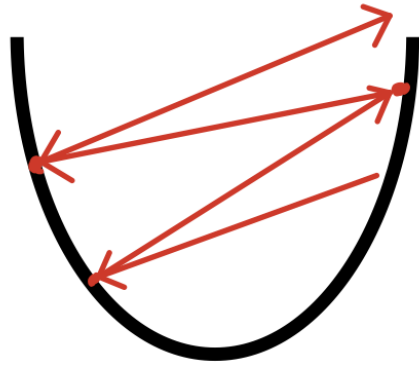
large learning rate



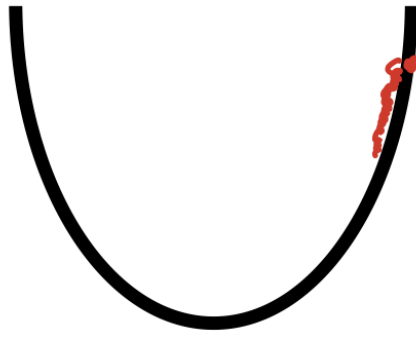
small learning rate

VGG 실습

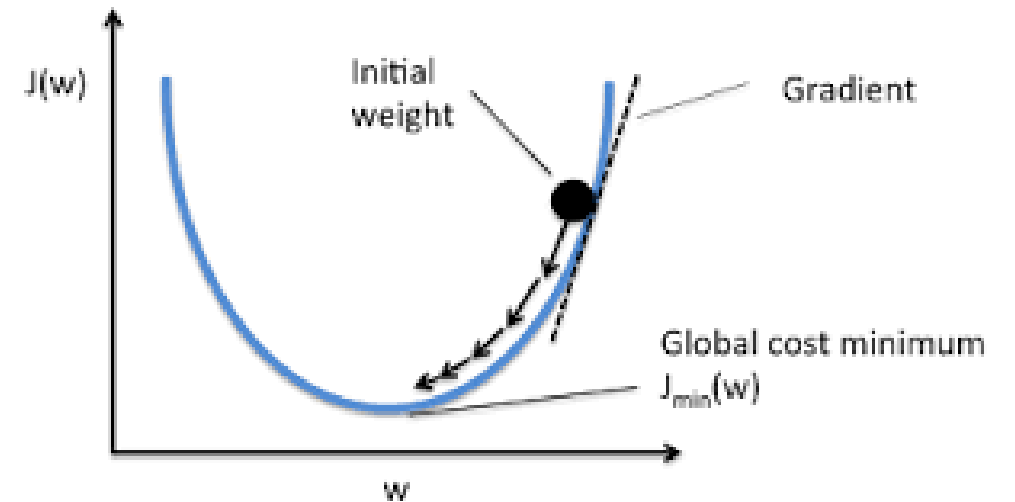
- **Cifar-10 Dataset으로 실습**
 - 모델 학습하기
 - learning rate란?



large learning rate



small learning rate



VGG 실습

- **Cifar-10 Dataset으로 실습**
 - 모델 학습하기
 - learning rate란?

```
▶ vgg_cifar10 = vgg16(input_shape = (32, 32, 3))
```

```
opt = tf.keras.optimizers.Adam(learning_rate=0.1)
```

```
vgg_cifar10.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['acc'])
```

```
vgg_cifar10.summary()
```

적절한 learning rate를 찾아서 한번 바꿔보기
참고로 default값은 0.001

Model: "sequential_3"

Layer (type)	Output Shape	Param #
conv2d_26 (Conv2D)	(None, 32, 32, 64)	1792
conv2d_27 (Conv2D)	(None, 32, 32, 64)	36928

VGG 실습

- **Cifar-10 Dataset으로 실습**

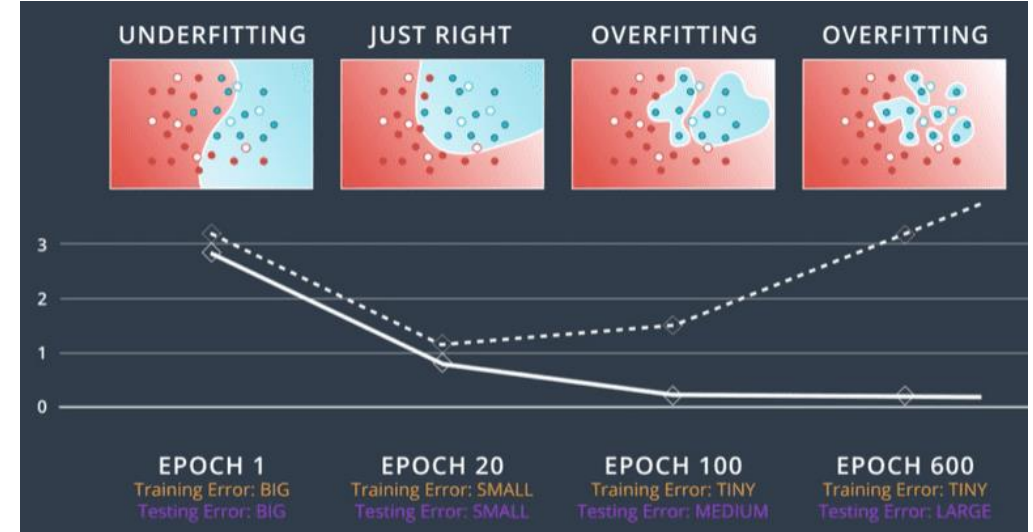
- 모델 학습하기
 - epoch란?

```
[9] history = vgg_cifar10.fit(x_train, y_train, batch_size=32, epochs=10, validation_data=(x_valid, y_valid))
```

```
Epoch 1/10
 1/1250 [.....] - ETA: 0s - loss: 2.3023 - acc: 0.1562WARNING:tensorflow:Callbacks method `on_t
1250/1250 [=====] - 43s 35ms/step - loss: nan - acc: 0.0992 - val_loss: nan - val_acc: 0.1036
Epoch 2/10
1250/1250 [=====] - 44s 35ms/step - loss: nan - acc: 0.0991 - val_loss: nan - val_acc: 0.1036
Epoch 3/10
1250/1250 [=====] - 44s 35ms/step - loss: nan - acc: 0.0991 - val_loss: nan - val_acc: 0.1036
Epoch 4/10
1250/1250 [=====] - 44s 35ms/step - loss: nan - acc: 0.0991 - val_loss: nan - val_acc: 0.1036
Epoch 5/10
1250/1250 [=====] - 44s 35ms/step - loss: nan - acc: 0.0991 - val_loss: nan - val_acc: 0.1036
Epoch 6/10
1250/1250 [=====] - 44s 35ms/step - loss: nan - acc: 0.0991 - val_loss: nan - val_acc: 0.1036
Epoch 7/10
1250/1250 [=====] - 44s 35ms/step - loss: nan - acc: 0.0991 - val_loss: nan - val_acc: 0.1036
Epoch 8/10
1250/1250 [=====] - 44s 35ms/step - loss: nan - acc: 0.0991 - val_loss: nan - val_acc: 0.1036
Epoch 9/10
1250/1250 [=====] - 44s 35ms/step - loss: nan - acc: 0.0991 - val_loss: nan - val_acc: 0.1036
Epoch 10/10
1250/1250 [=====] - 44s 35ms/step - loss: nan - acc: 0.0991 - val_loss: nan - val_acc: 0.1036
```

VGG 실습

- **Cifar-10 Dataset으로 실습**
 - 모델 학습하기
 - epoch란?



Iteration	Iteration	Iteration	Iteration	Iteration	Iteration	Iteration	Iteration	Iteration	Iteration
-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------

1 epoch = 전체 데이터셋을 한번 학습 완료

그렇다면 epoch를 iteration으로 나눠서 학습하는 이유는?? (조금 이따 설명)

VGG 실습

- **Cifar-10 Dataset으로 실습**

- 모델 학습하기
 - epoch란?

적절한 epoch를 찾아서 한번 바꿔보기

```
[9] history = vgg_cifar10.fit(x_train, y_train, batch_size=32, epochs=10, validation_data=(x_valid, y_valid))
```

```
Epoch 1/10
 1/1250 [.....] - ETA: 0s - loss: 2.3023 - acc: 0.1562WARNING:tensorflow:Callbacks method `c
1250/1250 [=====] - 43s 35ms/step - loss: nan - acc: 0.0992 - val_loss: nan - val_acc: 0.1036
Epoch 2/10
1250/1250 [=====] - 44s 35ms/step - loss: nan - acc: 0.0991 - val_loss: nan - val_acc: 0.1036
Epoch 3/10
1250/1250 [=====] - 44s 35ms/step - loss: nan - acc: 0.0991 - val_loss: nan - val_acc: 0.1036
Epoch 4/10
1250/1250 [=====] - 44s 35ms/step - loss: nan - acc: 0.0991 - val_loss: nan - val_acc: 0.1036
Epoch 5/10
1250/1250 [=====] - 44s 35ms/step - loss: nan - acc: 0.0991 - val_loss: nan - val_acc: 0.1036
Epoch 6/10
1250/1250 [=====] - 44s 35ms/step - loss: nan - acc: 0.0991 - val_loss: nan - val_acc: 0.1036
Epoch 7/10
1250/1250 [=====] - 44s 35ms/step - loss: nan - acc: 0.0991 - val_loss: nan - val_acc: 0.1036
Epoch 8/10
1250/1250 [=====] - 44s 35ms/step - loss: nan - acc: 0.0991 - val_loss: nan - val_acc: 0.1036
Epoch 9/10
1250/1250 [=====] - 44s 35ms/step - loss: nan - acc: 0.0991 - val_loss: nan - val_acc: 0.1036
Epoch 10/10
1250/1250 [=====] - 44s 35ms/step - loss: nan - acc: 0.0991 - val_loss: nan - val_acc: 0.1036
```

VGG 실습

- **Cifar-10 Dataset으로 실습**

- 모델 학습하기
 - batch size란?

```
[9] history = vgg_cifar10.fit(x_train, y_train, batch_size=32, epochs=10, validation_data=(x_valid, y_valid))
```

```
Epoch 1/10
 1/1250 [.....] - ETA: 0s - loss: 2.3023 - acc: 0.1562WARNING:tensorflow:Callbacks method `c
1250/1250 [=====] - 43s 35ms/step - loss: nan - acc: 0.0992 - val_loss: nan - val_acc: 0.1036
Epoch 2/10
1250/1250 [=====] - 44s 35ms/step - loss: nan - acc: 0.0991 - val_loss: nan - val_acc: 0.1036
Epoch 3/10
1250/1250 [=====] - 44s 35ms/step - loss: nan - acc: 0.0991 - val_loss: nan - val_acc: 0.1036
Epoch 4/10
1250/1250 [=====] - 44s 35ms/step - loss: nan - acc: 0.0991 - val_loss: nan - val_acc: 0.1036
Epoch 5/10
1250/1250 [=====] - 44s 35ms/step - loss: nan - acc: 0.0991 - val_loss: nan - val_acc: 0.1036
Epoch 6/10
1250/1250 [=====] - 44s 35ms/step - loss: nan - acc: 0.0991 - val_loss: nan - val_acc: 0.1036
Epoch 7/10
1250/1250 [=====] - 44s 35ms/step - loss: nan - acc: 0.0991 - val_loss: nan - val_acc: 0.1036
Epoch 8/10
1250/1250 [=====] - 44s 35ms/step - loss: nan - acc: 0.0991 - val_loss: nan - val_acc: 0.1036
Epoch 9/10
1250/1250 [=====] - 44s 35ms/step - loss: nan - acc: 0.0991 - val_loss: nan - val_acc: 0.1036
Epoch 10/10
1250/1250 [=====] - 44s 35ms/step - loss: nan - acc: 0.0991 - val_loss: nan - val_acc: 0.1036
```

VGG 실습

- **Cifar-10 Dataset으로 실습**

- 모델 학습하기
 - batch size란?

전체 데이터셋을 한번에 메모리에 올려놓고 학습을 할 수 없음.

memory <- 나의 메모리 크기



↔
batch size

만약 전체 데이터셋이 40,000개일 경우
batch size가 40일 때
1 epoch을 완료하기 위해선
1,000 iteration이 필요

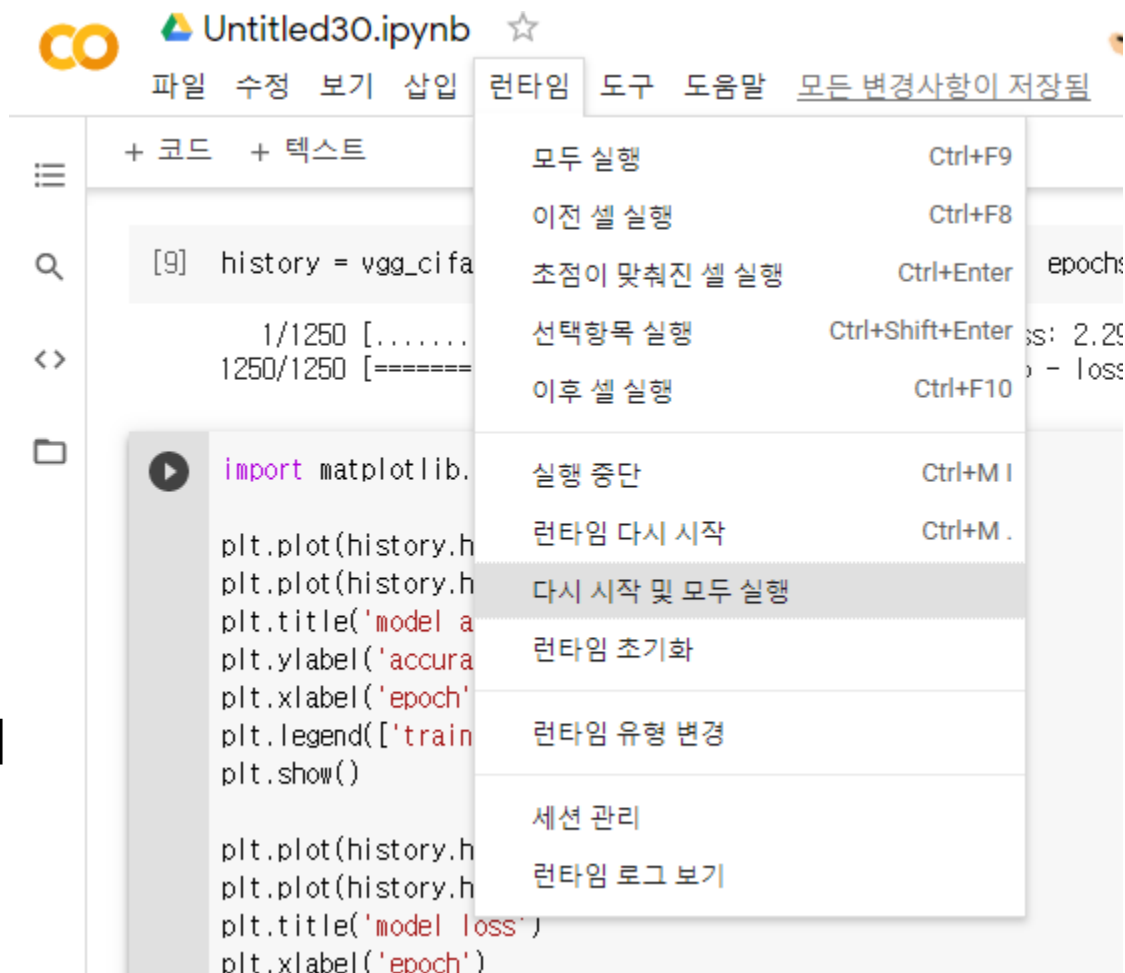
VGG 실습

- **Cifar-10 Dataset으로 실습**
 - 모델 학습하기

learning rate와 epochs, batch size 등 수정했으면 오른쪽과 같이 다시 시작

이렇게 하이퍼파라미터를 수정하는 기준이 Validation data이다.

즉, 하이퍼파라미터는 validation data의 정확도를 높이는 쪽으로 설정되어 있다.



VGG 실습

- **Cifar-10 Dataset으로 실습**
 - 모델 학습하기

```
history = vgg_cifar10.fit(x_train, y_train, batch_size=32, epochs=20, validation_data=(x_valid, y_valid))
```

```
Epoch 1/20
1/1250 [.....] - ETA: 49s - loss: 2.3046 - acc: 0.0938WARNING:tensorflow:Callbacks method `on_train_end`
1250/1250 [=====] - 44s 35ms/step - loss: 1.8338 - acc: 0.2554 - val_loss: 1.5577 - val_acc: 0.3925
Epoch 2/20
1250/1250 [=====] - 44s 35ms/step - loss: 1.3238 - acc: 0.5055 - val_loss: 1.1360 - val_acc: 0.5951
Epoch 3/20
1250/1250 [=====] - 44s 35ms/step - loss: 1.0041 - acc: 0.6389 - val_loss: 1.0167 - val_acc: 0.6464
Epoch 4/20
1250/1250 [=====] - 44s 35ms/step - loss: 0.7976 - acc: 0.7214 - val_loss: 0.8977 - val_acc: 0.6860
Epoch 5/20
1250/1250 [=====] - 43s 35ms/step - loss: 0.6310 - acc: 0.7799 - val_loss: 0.8644 - val_acc: 0.7113
Epoch 6/20
1250/1250 [=====] - 43s 35ms/step - loss: 0.4927 - acc: 0.8281 - val_loss: 0.9117 - val_acc: 0.7133
Epoch 7/20
1250/1250 [=====] - 43s 35ms/step - loss: 0.3747 - acc: 0.8711 - val_loss: 0.8555 - val_acc: 0.7364
Epoch 8/20
1250/1250 [=====] - 43s 35ms/step - loss: 0.2908 - acc: 0.9005 - val_loss: 0.9488 - val_acc: 0.7400
Epoch 9/20
1250/1250 [=====] - 43s 35ms/step - loss: 0.2298 - acc: 0.9216 - val_loss: 1.0726 - val_acc: 0.7229
Epoch 10/20
1250/1250 [=====] - 43s 35ms/step - loss: 0.1877 - acc: 0.9372 - val_loss: 1.0927 - val_acc: 0.7387
Epoch 11/20
1250/1250 [=====] - 44s 35ms/step - loss: 0.1522 - acc: 0.9490 - val_loss: 1.0864 - val_acc: 0.7437
Epoch 12/20
1250/1250 [=====] - 44s 35ms/step - loss: 0.1451 - acc: 0.9525 - val_loss: 1.1426 - val_acc: 0.7411
Epoch 13/20
1250/1250 [=====] - 43s 35ms/step - loss: 0.1253 - acc: 0.9590 - val_loss: 1.1194 - val_acc: 0.7439
Epoch 14/20
1250/1250 [=====] - 43s 35ms/step - loss: 0.1118 - acc: 0.9637 - val_loss: 1.1605 - val_acc: 0.7438
Epoch 15/20
1250/1250 [=====] - 43s 35ms/step - loss: 0.0991 - acc: 0.9689 - val_loss: 1.3363 - val_acc: 0.7446
Epoch 16/20
1250/1250 [=====] - 44s 35ms/step - loss: 0.0961 - acc: 0.9697 - val_loss: 1.2426 - val_acc: 0.7403
Epoch 17/20
1250/1250 [=====] - 43s 35ms/step - loss: 0.0923 - acc: 0.9702 - val_loss: 1.2839 - val_acc: 0.7424
Epoch 18/20
1250/1250 [=====] - 43s 35ms/step - loss: 0.0837 - acc: 0.9733 - val_loss: 1.3219 - val_acc: 0.7463
Epoch 19/20
1250/1250 [=====] - 44s 35ms/step - loss: 0.0818 - acc: 0.9742 - val_loss: 1.2160 - val_acc: 0.7404
Epoch 20/20
1250/1250 [=====] - 43s 35ms/step - loss: 0.0735 - acc: 0.9773 - val_loss: 1.4095 - val_acc: 0.7432
```

VGG 실습

- **Cifar-10 Dataset으로 실습**
 - 모델 학습하기



```
import numpy as np
```

```
print('validation accuracy')  
print(history.history['val_acc'][-1])  
print(np.max(history.history['val_acc']))
```

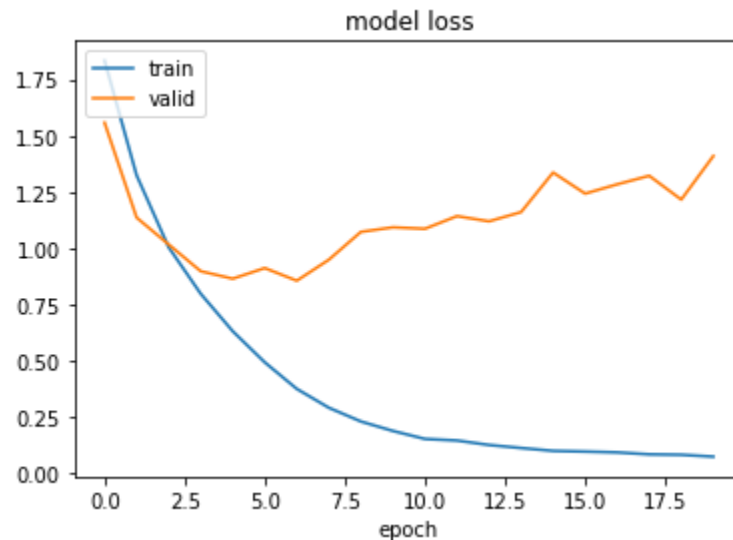
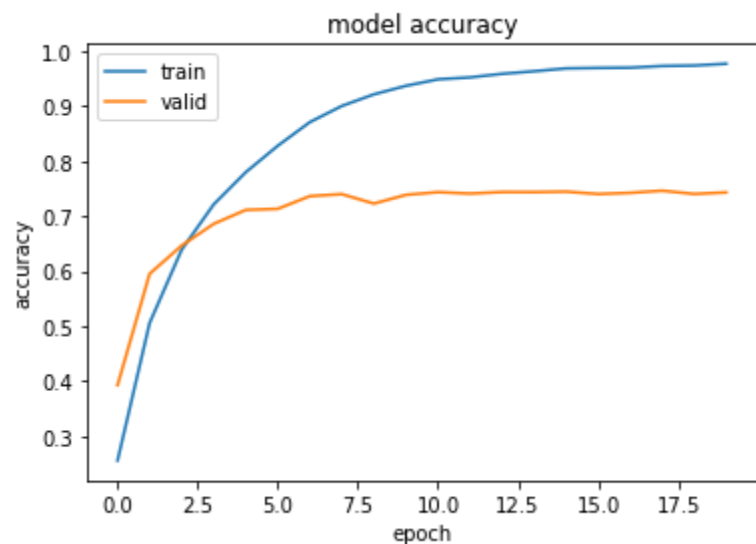
validation accuracy

0.7432000041007996

<- 최종 정확도

0.7462999820709229

<- best 정확도



VGG 실습

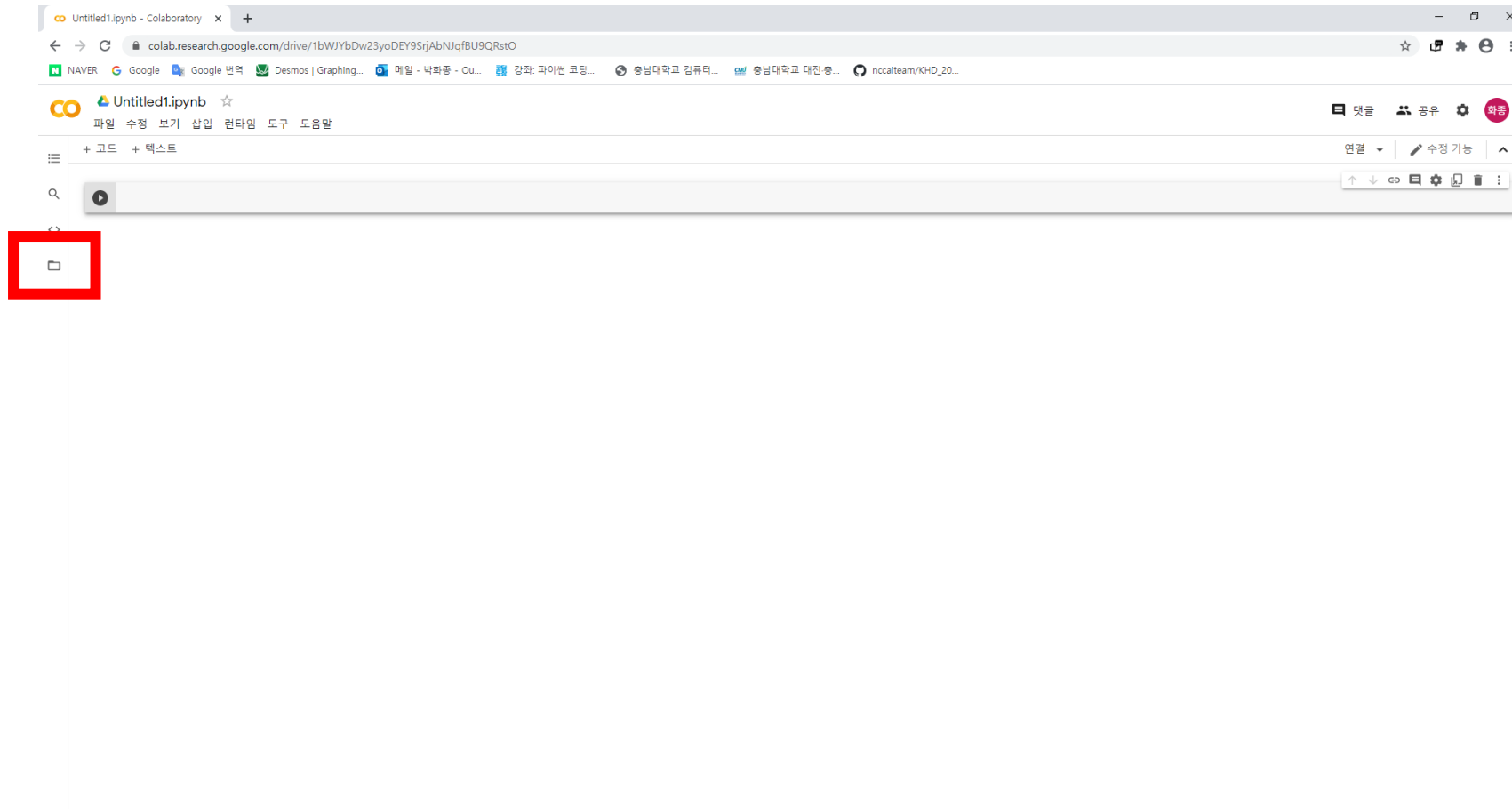
- **Cifar-10 Dataset으로 실습**
 - 모델 테스트하기

```
▶ results = vgg_cifar10.evaluate(x_test, y_test, batch_size=32)  
  
print('test accuracy')  
print(results[1])
```

```
313/313 [=====] - 3s 9ms/step - loss: 1.4189 - acc: 0.7476  
test accuracy  
0.7476000189781189
```

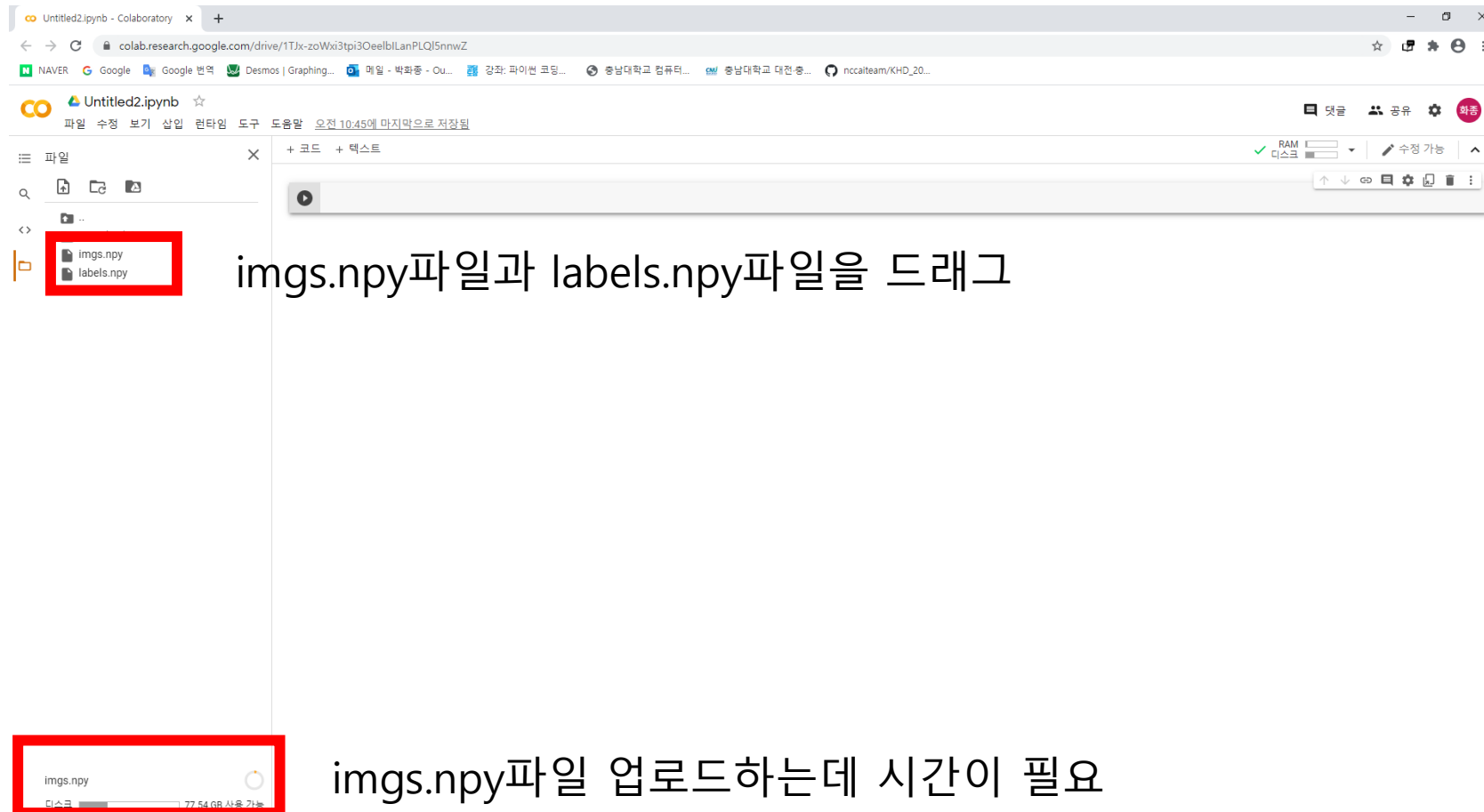
ResNet 실습

- Colab에 데이터 불러오기



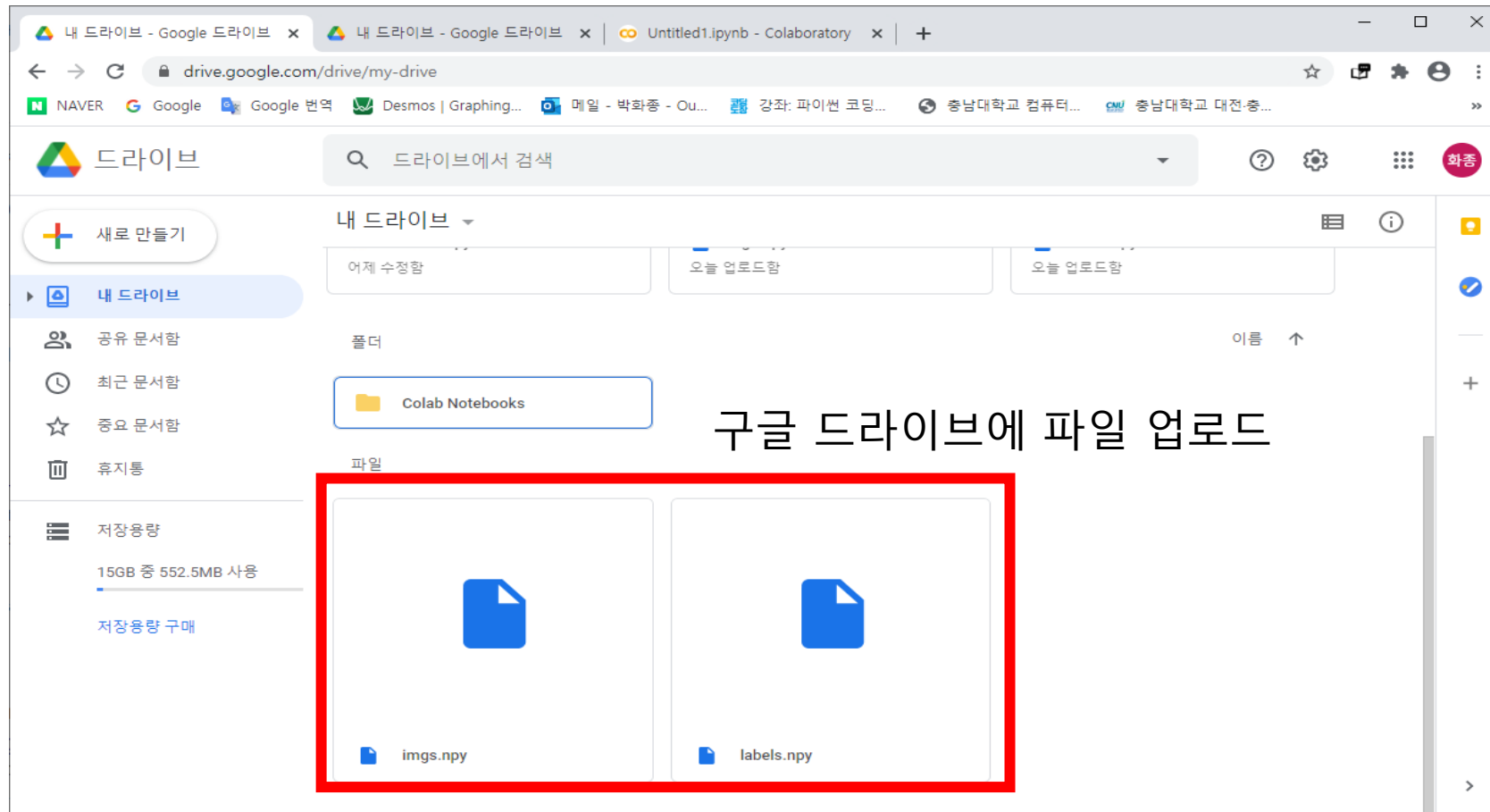
ResNet 실습

- Colab에 데이터 불러오기(방법1)



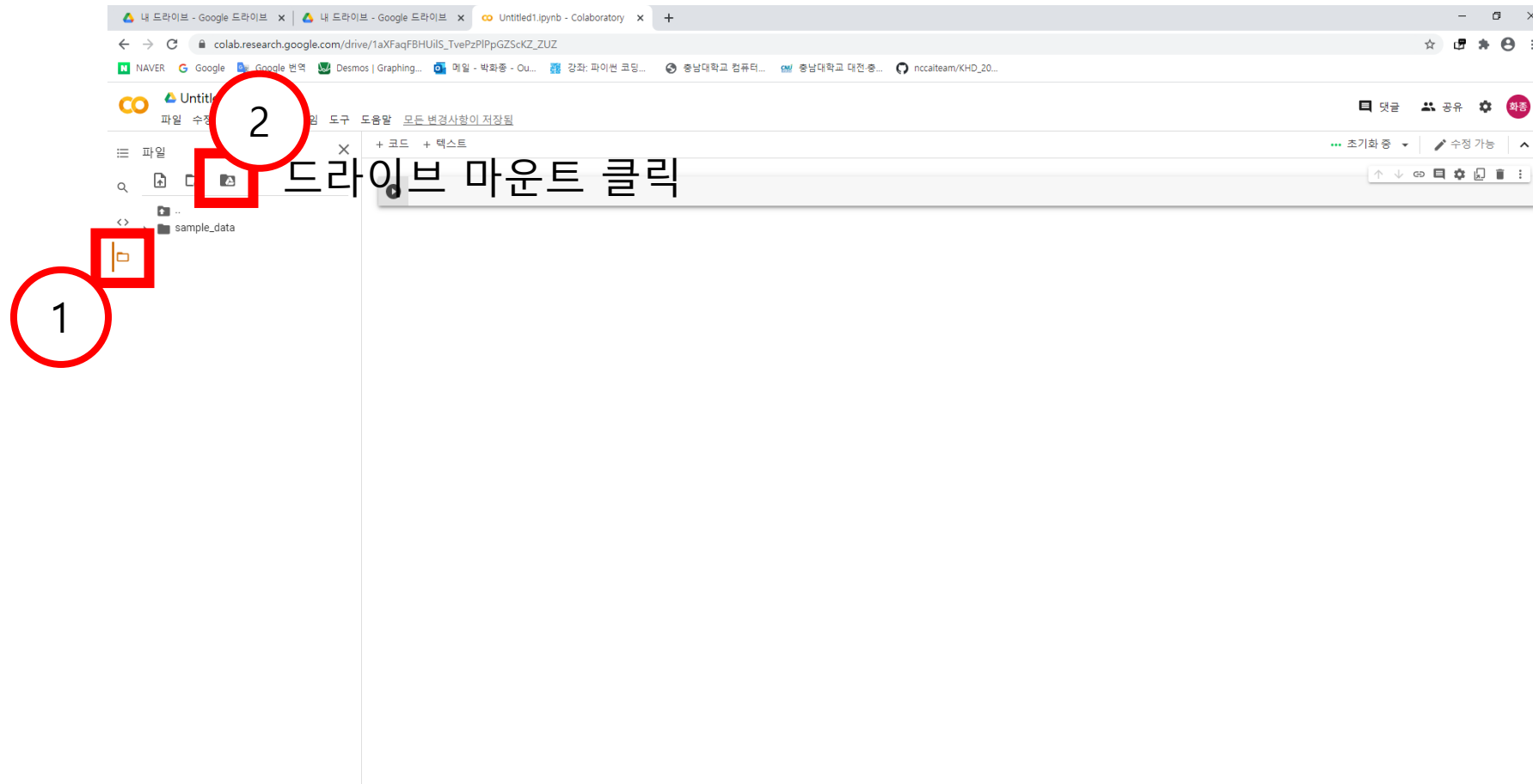
ResNet 실습

- Colab에 데이터 불러오기(방법2)



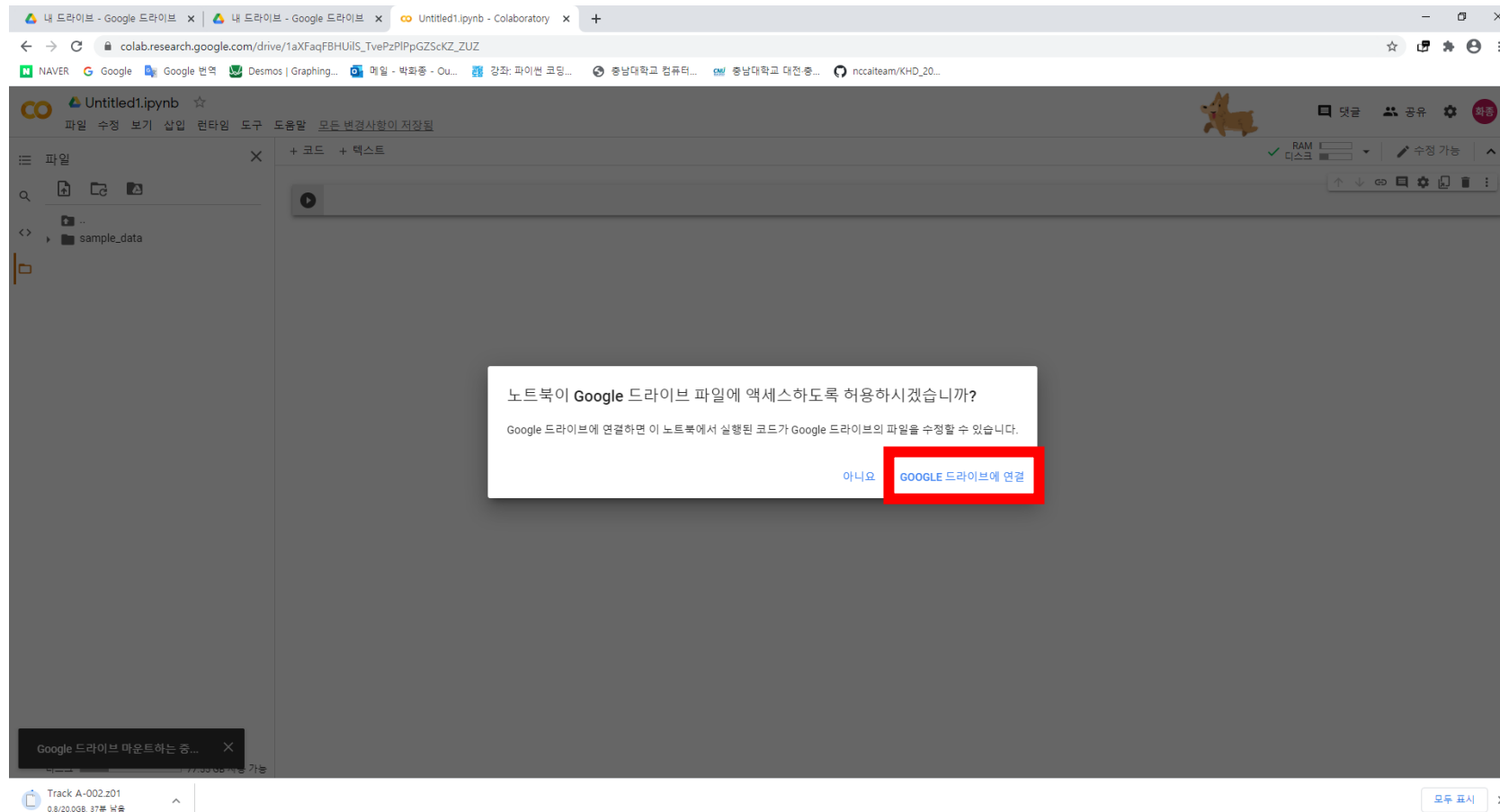
ResNet 실습

- Colab에 데이터 불러오기(방법2)



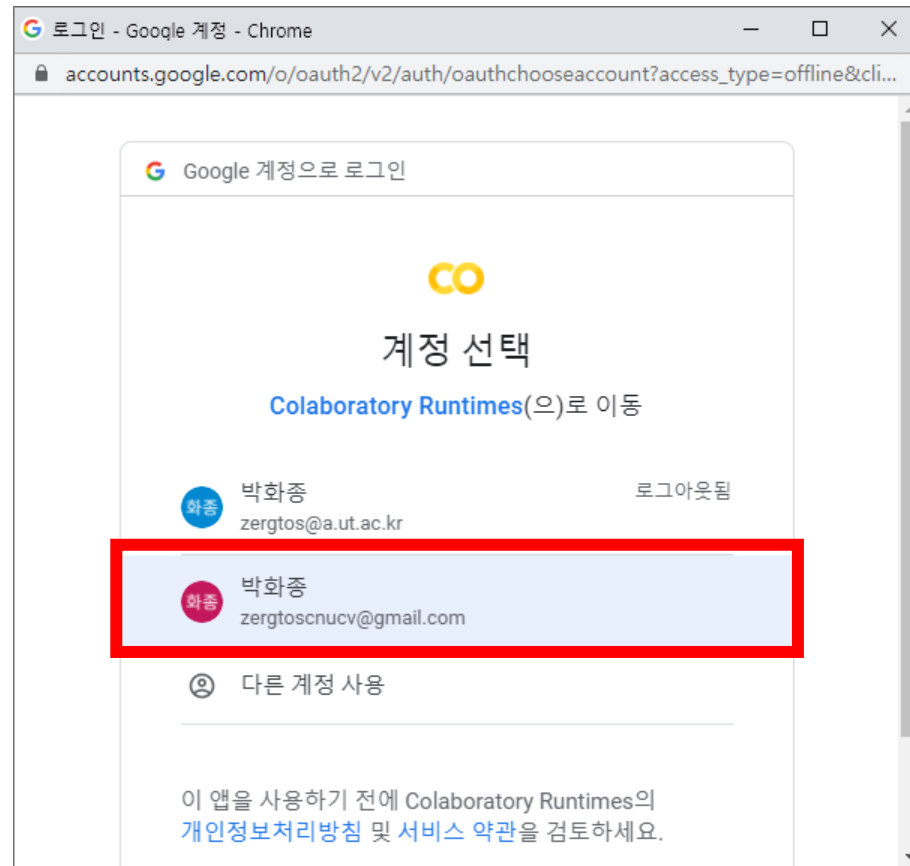
ResNet 실습

- Colab에 데이터 불러오기(방법2)



ResNet 실습

- Colab에 데이터 불러오기(방법2)



파일을 업로드한 계정 선택

ResNet 실습

- Colab에 데이터 불러오기(방법2)

링크 클릭

```
1 from google.colab import drive  
2 drive.mount('/content/drive')
```

... Go to this URL in a browser https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdg1

Enter your authorization code:

ResNet 실습

• Colab에 데이터 불러오기(방법2)

이렇게 하면 [Google Drive File Stream](#)에서 다음 작업을 할 수 있습니다.

-  Google 드라이브 파일 보기, 수정, 생성, 삭제 ⓘ
-  Google 포토의 사진, 동영상, 앨범을 봅니다. ⓘ
-  프로필 및 연락처와 같은 Google 사용자 정보 ⓘ
조회
-  Google 드라이브 문서 보기, 수정, 생성, 삭제 ⓘ

Google Drive File Stream 앱을 신뢰할 수 있는지 확인

민감한 정보가 이 사이트 또는 앱과 공유될 수 있습니다. Google Drive File Stream의 [서비스 약관](#) 및 [개인정보처리방침](#)을 검토하여 내 데이터가 어떻게 처리되는지 알아보세요. 언제든지 [Google 계정](#)에서 액세스 권한을 확인하고 삭제할 수 있습니다.

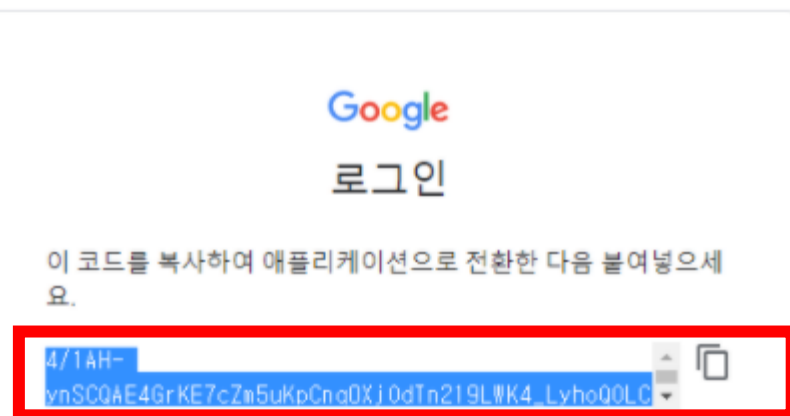
[타사 앱 권한 부여에 관한 위험 알아보기](#)

취소

허용

ResNet 실습

- Colab에 데이터 불러오기(방법2)



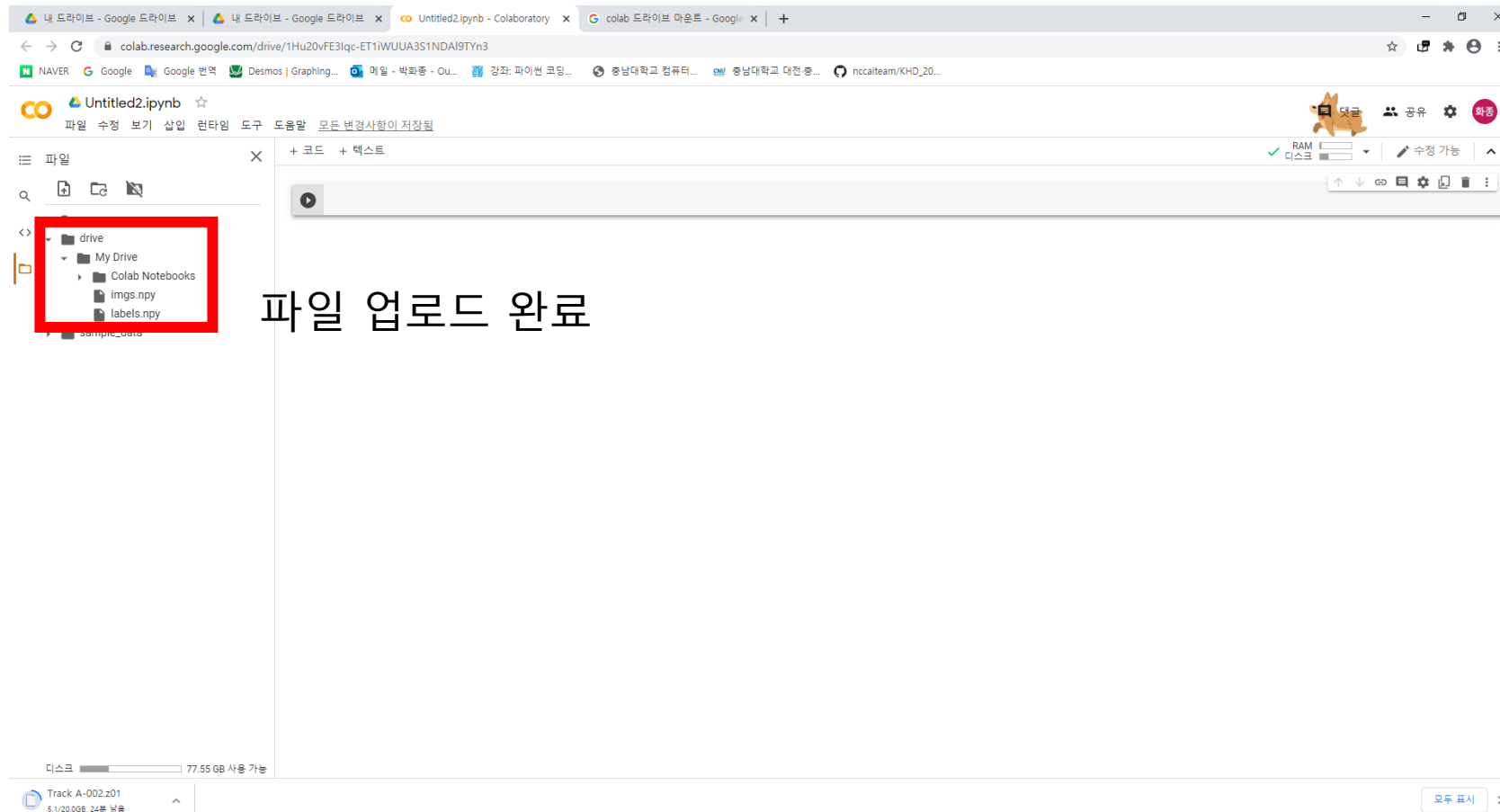
링크 복사



링크 붙여넣기 후 run

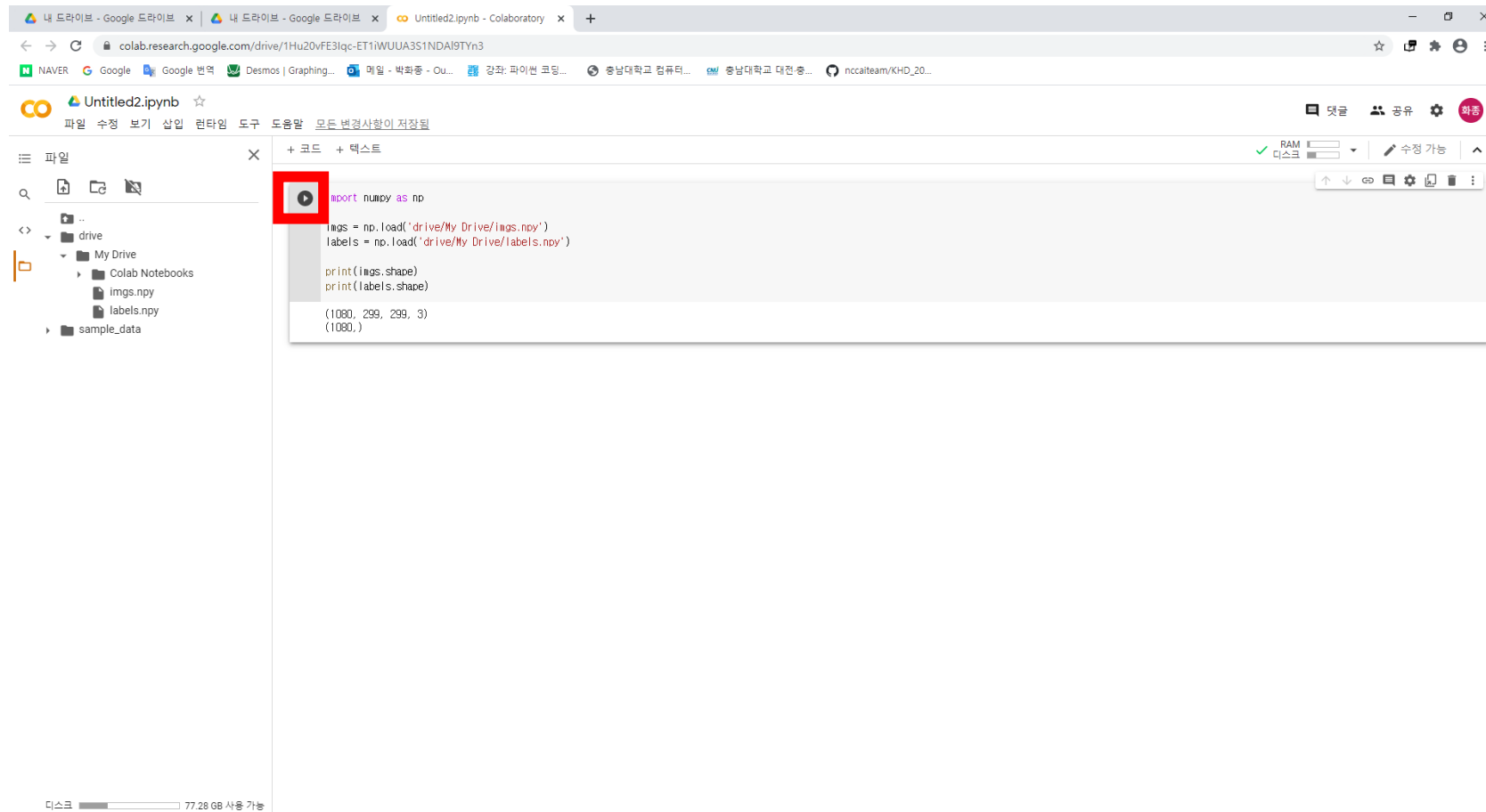
ResNet 실습

• Colab에 데이터 불러오기(방법2)



ResNet 실습

- Colab에 데이터 불러오기



ResNet 실습

• Dataset 설명

	지진	산불	폭설	산사태	낙뢰	폭우	쓰나미	태풍	화산
Img									
									
Data 개수	120	120	120	120	120	120	120	120	120

ResNet 실습

- Dataset 설명

- Disaster

- 지진
 - 산불
 - 폭설
 - 산사태
 - 낙뢰
 - 폭우
 - 쓰나미
 - 태풍
 - 화산

- Image

- shape : (229, 229, 3)
 - 데이터 수
 - 각 재난별 120개씩 총 1080개

ResNet 실습

- 코드
 - ResNet50

```
[1] import numpy as np

imgs = np.load('drive/My Drive/imgs.npy')
labels = np.load('drive/My Drive/labels.npy')

print(imgs.shape)
print(labels.shape)

(1080, 299, 299, 3)
(1080,)
```

```
[2] import cv2
print(cv2.__version__)

#resize (299, 299) -> (32, 32)
x_train = []
for idx in range(len(imgs)):
    print('\r start ', idx+1, '/', len(imgs), end='')
    img = imgs[idx]
    img = cv2.resize(img, (32, 32))
    x_train.append(img)
print()#end

x_train = np.array(x_train)
print(x_train.shape)

4.1.2
start 1080 / 1080
(1080, 32, 32, 3)
```

ResNet 실습

- 코드

```
import tensorflow as tf

base_model = tf.keras.applications.ResNet50(weights=None, input_shape=(32, 32, 3))
base_model = tf.keras.models.Model(base_model.inputs, base_model.layers[-2].output)
x = base_model.output
pred = tf.keras.layers.Dense(9, activation='softmax')(x)
model = tf.keras.models.Model(inputs=base_model.input, outputs=pred)

opt = tf.keras.optimizers.Adam(learning_rate=0.001)
model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['acc'])

model.summary()
```

ResNet 실습

• 코드

conv5_block2_add (Add)	(None, 1, 1, 2048)	0	conv5_block1_out [0] [0] conv5_block2_3_bn [0] [0]
conv5_block2_out (Activation)	(None, 1, 1, 2048)	0	conv5_block2_add [0] [0]
conv5_block3_1_conv (Conv2D)	(None, 1, 1, 512)	1049088	conv5_block2_out [0] [0]
conv5_block3_1_bn (BatchNormali	(None, 1, 1, 512)	2048	conv5_block3_1_conv [0] [0]
conv5_block3_1_relu (Activation	(None, 1, 1, 512)	0	conv5_block3_1_bn [0] [0]
conv5_block3_2_conv (Conv2D)	(None, 1, 1, 512)	2359808	conv5_block3_1_relu [0] [0]
conv5_block3_2_bn (BatchNormali	(None, 1, 1, 512)	2048	conv5_block3_2_conv [0] [0]
conv5_block3_2_relu (Activation	(None, 1, 1, 512)	0	conv5_block3_2_bn [0] [0]
conv5_block3_3_conv (Conv2D)	(None, 1, 1, 2048)	1050624	conv5_block3_2_relu [0] [0]
conv5_block3_3_bn (BatchNormali	(None, 1, 1, 2048)	8192	conv5_block3_3_conv [0] [0]
conv5_block3_add (Add)	(None, 1, 1, 2048)	0	conv5_block2_out [0] [0] conv5_block3_3_bn [0] [0]
conv5_block3_out (Activation)	(None, 1, 1, 2048)	0	conv5_block3_add [0] [0]
avg_pool (GlobalAveragePooling2	(None, 2048)	0	conv5_block3_out [0] [0]
dense (Dense)	(None, 9)	18441	avg_pool [0] [0]

Total params: 23,606,153
Trainable params: 23,553,033
Non-trainable params: 53,120

ResNet 실습

- 코드

```
[4] from sklearn.model_selection import train_test_split

y_train = np.reshape(labels, newshape=(len(labels), 1))
print(y_train.shape)
```

```
(1080, 1)
```

```
[5] x_train, x_valid, y_train, y_valid = train_test_split(x_train, y_train, test_size=0.2, random_state=123)

x_train, x_test, y_train, y_test = train_test_split(x_train, y_train, test_size=0.2, random_state=123)

print('train data')
print(x_train.shape)
print(y_train.shape)

print('validation data')
print(x_valid.shape)
print(y_valid.shape)

print('test data')
print(x_test.shape)
print(y_test.shape)
```

```
train data
(691, 32, 32, 3)
(691, 1)
validation data
(216, 32, 32, 3)
(216, 1)
test data
(173, 32, 32, 3)
(173, 1)
```

ResNet 실습

- 코드

```
[6] # scalar 형태의 레이블(0~8)을 One-hot Encoding 형태로 변환합니다.  
y_train = tf.squeeze(tf.one_hot(y_train, 9),axis=1)  
y_valid = tf.squeeze(tf.one_hot(y_valid, 9),axis=1)  
y_test = tf.squeeze(tf.one_hot(y_test, 9),axis=1)
```

```
print('train data')  
print(x_train.shape)  
print(y_train.shape)
```

```
print('valid data')  
print(x_valid.shape)  
print(y_valid.shape)
```

```
print('test data')  
print(x_test.shape)  
print(y_test.shape)
```

```
train data  
(691, 32, 32, 3)  
(691, 9)  
valid data  
(216, 32, 32, 3)  
(216, 9)  
test data  
(173, 32, 32, 3)  
(173, 9)
```

```
[7] print(np.sum(y_train, axis=0))  
print(np.sum(y_valid, axis=0))  
print(np.sum(y_test, axis=0))
```

```
[83. 79. 69. 77. 80. 76. 78. 71. 78.]  
[19. 27. 27. 15. 24. 23. 24. 29. 28.]  
[18. 14. 24. 28. 16. 21. 18. 20. 14.]
```

ResNet 실습

• 코드

```
▶ history = model.fit(x=x_train, y=y_train, batch_size=32, epochs=20, validation_data=(x_valid, y_valid))

▶ Epoch 1/20
22/22 [=====] - 2s 104ms/step - loss: 3.8725 - acc: 0.2084 - val_loss: 73.2270 - val_acc: 0.1250
Epoch 2/20
22/22 [=====] - 1s 38ms/step - loss: 2.0461 - acc: 0.3589 - val_loss: 44.8110 - val_acc: 0.0880
Epoch 3/20
22/22 [=====] - 1s 38ms/step - loss: 1.5703 - acc: 0.4588 - val_loss: 5.3259 - val_acc: 0.1389
Epoch 4/20
22/22 [=====] - 1s 38ms/step - loss: 1.1936 - acc: 0.6136 - val_loss: 2.5537 - val_acc: 0.1343
Epoch 5/20
22/22 [=====] - 1s 38ms/step - loss: 0.9895 - acc: 0.7438 - val_loss: 3.1189 - val_acc: 0.0880
Epoch 6/20
22/22 [=====] - 1s 38ms/step - loss: 0.9517 - acc: 0.7656 - val_loss: 3.5630 - val_acc: 0.1296
Epoch 7/20
22/22 [=====] - 1s 38ms/step - loss: 0.9287 - acc: 0.7496 - val_loss: 2.8116 - val_acc: 0.2130
Epoch 8/20
22/22 [=====] - 1s 38ms/step - loss: 0.6804 - acc: 0.8017 - val_loss: 4.0670 - val_acc: 0.1528
Epoch 9/20
22/22 [=====] - 1s 38ms/step - loss: 0.7761 - acc: 0.7916 - val_loss: 4.8960 - val_acc: 0.0880
Epoch 10/20
22/22 [=====] - 1s 39ms/step - loss: 0.5407 - acc: 0.8408 - val_loss: 4.1932 - val_acc: 0.1620
Epoch 11/20
22/22 [=====] - 1s 38ms/step - loss: 0.5839 - acc: 0.8770 - val_loss: 3.5121 - val_acc: 0.1667
Epoch 12/20
22/22 [=====] - 1s 38ms/step - loss: 0.6784 - acc: 0.8350 - val_loss: 3.5022 - val_acc: 0.2315
Epoch 13/20
22/22 [=====] - 1s 38ms/step - loss: 0.8102 - acc: 0.8017 - val_loss: 2.0090 - val_acc: 0.4167
Epoch 14/20
22/22 [=====] - 1s 39ms/step - loss: 0.2945 - acc: 0.9103 - val_loss: 2.9205 - val_acc: 0.3611
Epoch 15/20
22/22 [=====] - 1s 38ms/step - loss: 0.3294 - acc: 0.9030 - val_loss: 2.5826 - val_acc: 0.3889
Epoch 16/20
22/22 [=====] - 1s 38ms/step - loss: 0.4145 - acc: 0.9204 - val_loss: 3.3951 - val_acc: 0.3565
Epoch 17/20
22/22 [=====] - 1s 38ms/step - loss: 0.8663 - acc: 0.8162 - val_loss: 5.3431 - val_acc: 0.3148
Epoch 18/20
22/22 [=====] - 1s 38ms/step - loss: 0.9150 - acc: 0.7656 - val_loss: 20.7562 - val_acc: 0.3102
Epoch 19/20
22/22 [=====] - 1s 38ms/step - loss: 1.3382 - acc: 0.7250 - val_loss: 9.8198 - val_acc: 0.3657
Epoch 20/20
22/22 [=====] - 1s 38ms/step - loss: 1.0384 - acc: 0.7800 - val_loss: 21.5284 - val_acc: 0.3194
```

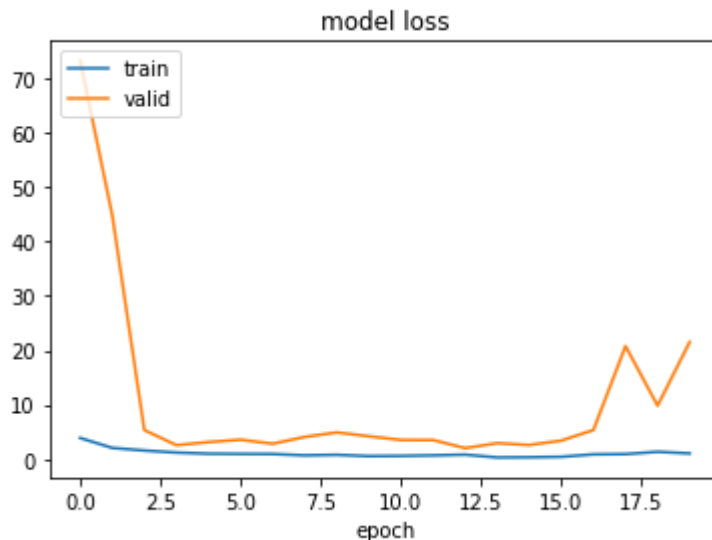
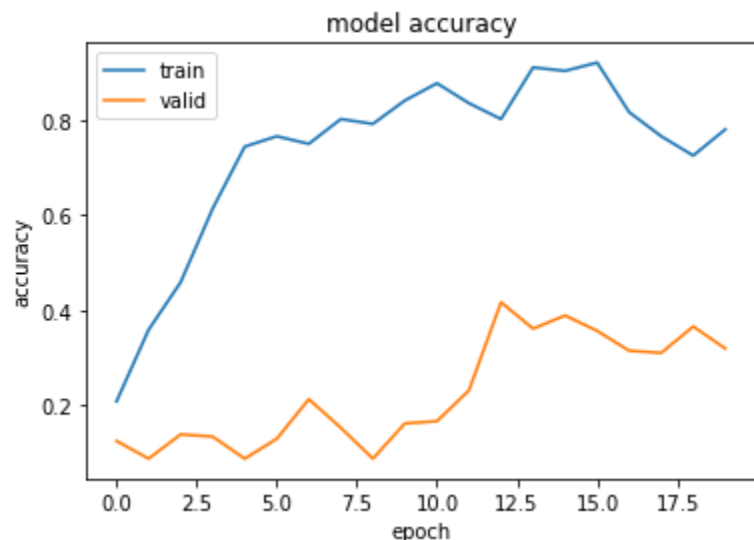
ResNet 실습

- 코드

```
[9] import matplotlib.pyplot as plt

plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'valid'], loc='upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.xlabel('epoch')
plt.legend(['train', 'valid'], loc='upper left')
plt.show()
```



ResNet 실습

- 코드

```
[10] print('validation accuracy')  
      print(history.history['val_acc'][-1])  
      print(np.max(history.history['val_acc']))
```

```
validation accuracy  
0.3194444477558136  
0.4166666567325592
```

```
[12] results = model.evaluate(x_test, y_test, batch_size=32)  
  
      print('test accuracy')  
      print(results[1])
```

```
6/6 [=====] - 0s 48ms/step - loss: 29.2589 - acc: 0.3815  
test accuracy  
0.3815028965473175
```


Inception 실습

- 코드
 - InceptionV3

```
[1] import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
```

```
[2] (x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()

x_train, x_valid, y_train, y_valid = train_test_split(x_train, y_train, test_size=0.2, random_state=123)
x_train, x_test, y_train, y_test = train_test_split(x_train, y_train, test_size=0.2, random_state=123)

y_train = tf.squeeze(tf.one_hot(y_train, 10), axis=1)
y_valid = tf.squeeze(tf.one_hot(y_valid, 10), axis=1)
y_test = tf.squeeze(tf.one_hot(y_test, 10), axis=1)

print('train data')
print(x_train.shape)
print(y_train.shape)

print('valid data')
print(x_valid.shape)
print(y_valid.shape)

print('test data')
print(x_test.shape)
print(y_test.shape)
```

할 필요 없음

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [=====] - 2s 0us/step
train data
(32000, 32, 32, 3)
(32000, 10)
valid data
(10000, 32, 32, 3)
(10000, 10)
test data
(8000, 32, 32, 3)
(8000, 10)
```

Inception 실습

• 코드

- InceptionV3



```
base_model = tf.keras.applications.InceptionV3(weights=None, input_shape=(32, 32, 3))
base_model = tf.keras.models.Model(base_model.inputs, base_model.layers[-2].output)
x = base_model.output
pred = tf.keras.layers.Dense(9, activation='softmax')(x)
model = tf.keras.models.Model(inputs=base_model.input, outputs=pred)

opt = tf.keras.optimizers.Adam(learning_rate=0.001)
model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['acc'])

model.summary()
```

```
-----
ValueError                                Traceback (most recent call last)
<ipython-input-3-20b01686d975> in <module>()
----> 1 base_model = tf.keras.applications.InceptionV3(weights=None, input_shape=(32, 32
      2 base_model = tf.keras.models.Model(base_model.inputs, base_model.layers[-2].outp
      3 x = base_model.output
      4 pred = tf.keras.layers.Dense(9, activation='softmax')(x)
      5 model = tf.keras.models.Model(inputs=base_model.input, outputs=pred)
```

1 frames

```
/usr/local/lib/python3.6/dist-packages/tensorflow/python/keras/applications/imagenet_utili
369         raise ValueError('Input size must be at least ' + str(min_size) +
370                             'x' + str(min_size) + '; got `input_shape=' +
--> 371                             str(input_shape) + '`')
372     else:
373         if require_flatten:
```

ValueError: Input size must be at least 75x75; got `input_shape=(32, 32, 3)`

SEARCH STACK OVERFLOW

Inception 실습

- 코드

- InceptionV3

```
import cv2

x_data = []
for i in range(len(x_train)):
    img = cv2.resize(x_train[i], (75, 75))
    x_data.append(img)

x_train = np.array(x_data)

x_data = []
for i in range(len(x_valid)):
    img = cv2.resize(x_valid[i], (75, 75))
    x_data.append(img)

x_valid = np.array(x_data)

x_data = []
for i in range(len(x_test)):
    img = cv2.resize(x_test[i], (75, 75))
    x_data.append(img)

x_test = np.array(x_data)

print(x_train.shape)
print(x_valid.shape)
print(x_test.shape)
```

```
(32000, 75, 75, 3)
(10000, 75, 75, 3)
(8000, 75, 75, 3)
```

```
[4] base_model = tf.keras.applications.InceptionV3(weights='imagenet')
base_model = tf.keras.models.Model(base_model.inputs, base_model.outputs)
```

Inception 실습

• 코드

• InceptionV3

concatenate_3 (Concatenate)	(None, 1, 1, 768)	0
activation_187 (Activation)	(None, 1, 1, 192)	0
mixed10 (Concatenate)	(None, 1, 1, 2048)	0
avg_pool (GlobalAveragePooling2)	(None, 2048)	0
dense_1 (Dense)	(None, 10)	20490
=====		
Total params: 21,823,274		
Trainable params: 21,788,842		
Non-trainable params: 34,432		

```

base_model = tf.keras.applications.InceptionV3(weights=None, input_shape=(75, 75, 3))
base_model = tf.keras.models.Model(base_model.inputs, base_model.layers[-2].output)
x = base_model.output
pred = tf.keras.layers.Dense(10, activation='softmax')(x)
model = tf.keras.models.Model(inputs=base_model.input, outputs=pred)

opt = tf.keras.optimizers.Adam(learning_rate=0.001)
model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['acc'])

model.summary()

```

batch_normalization_166 (Batch Normalization)	(None, 3, 3, 192)	576	conv2d_166[0][0]
activation_166 (Activation)	(None, 3, 3, 192)	0	batch_normalization_166[0][0]
conv2d_167 (Conv2D)	(None, 3, 3, 192)	258048	activation_166[0][0]
batch_normalization_167 (Batch Normalization)	(None, 3, 3, 192)	576	conv2d_167[0][0]
activation_167 (Activation)	(None, 3, 3, 192)	0	batch_normalization_167[0][0]
conv2d_164 (Conv2D)	(None, 3, 3, 192)	147456	mixed7[0][0]
conv2d_168 (Conv2D)	(None, 3, 3, 192)	258048	activation_167[0][0]
batch_normalization_164 (Batch Normalization)	(None, 3, 3, 192)	576	conv2d_164[0][0]
batch_normalization_168 (Batch Normalization)	(None, 3, 3, 192)	576	conv2d_168[0][0]
activation_164 (Activation)	(None, 3, 3, 192)	0	batch_normalization_164[0][0]

Inception 실습

• 코드

• InceptionV3

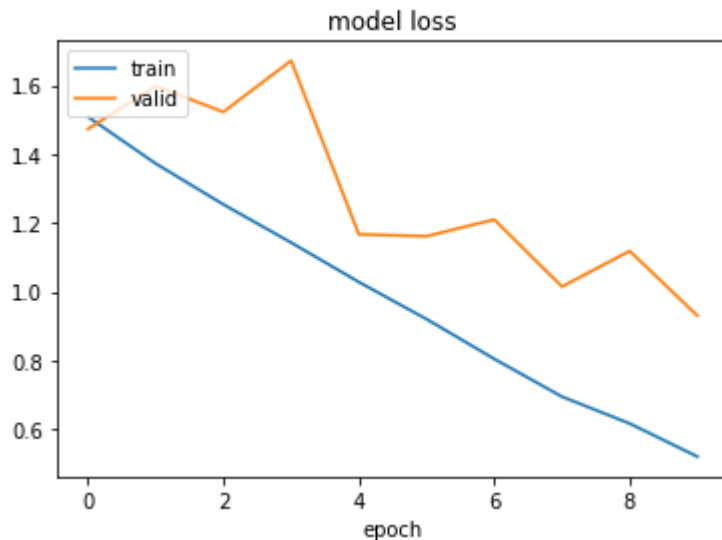
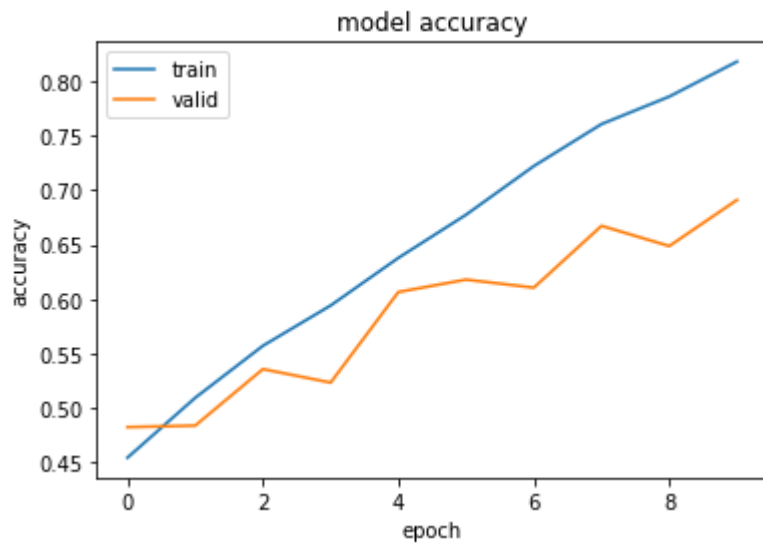
하이퍼파라미터는
알아서 설정하세요!

```
[8] history = model.fit(x=x_train, y=y_train, batch_size=32, epochs=10, validation_data=(x_valid, y_valid))
```

```
Epoch 1/10
1000/1000 [=====] - 54s 54ms/step - loss: 1.5088 - acc: 0.4543 - val_loss: 1.4732 - val_acc: 0.4824
Epoch 2/10
1000/1000 [=====] - 54s 54ms/step - loss: 1.3726 - acc: 0.5093 - val_loss: 1.5982 - val_acc: 0.4838
Epoch 3/10
1000/1000 [=====] - 54s 54ms/step - loss: 1.2536 - acc: 0.5571 - val_loss: 1.5229 - val_acc: 0.5357
Epoch 4/10
1000/1000 [=====] - 54s 54ms/step - loss: 1.1425 - acc: 0.5941 - val_loss: 1.6724 - val_acc: 0.5233
Epoch 5/10
1000/1000 [=====] - 54s 54ms/step - loss: 1.0277 - acc: 0.6378 - val_loss: 1.1666 - val_acc: 0.6065
Epoch 6/10
1000/1000 [=====] - 54s 54ms/step - loss: 0.9196 - acc: 0.6775 - val_loss: 1.1610 - val_acc: 0.6179
Epoch 7/10
1000/1000 [=====] - 54s 54ms/step - loss: 0.8038 - acc: 0.7222 - val_loss: 1.2096 - val_acc: 0.6105
Epoch 8/10
1000/1000 [=====] - 54s 54ms/step - loss: 0.6938 - acc: 0.7607 - val_loss: 1.0143 - val_acc: 0.6673
Epoch 9/10
1000/1000 [=====] - 54s 54ms/step - loss: 0.6156 - acc: 0.7860 - val_loss: 1.1180 - val_acc: 0.6487
Epoch 10/10
1000/1000 [=====] - 54s 54ms/step - loss: 0.5192 - acc: 0.8181 - val_loss: 0.9295 - val_acc: 0.6911
```

Inception 실습

- 코드
 - InceptionV3



```
[9] plt.plot(history.history['acc'])
    plt.plot(history.history['val_acc'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'valid'], loc='upper left')
    plt.show()

    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'valid'], loc='upper left')
    plt.show()
```

Inception 실습

- 코드
 - InceptionV3

```
[10] print('validation accuracy')  
      print(history.history['val_acc'][-1])  
      print(np.max(history.history['val_acc']))
```

```
validation accuracy  
0.691100001335144  
0.691100001335144
```

```
▶ results = model.evaluate(x_test, y_test, batch_size=32)  
  
print('test accuracy')  
print(results[1])
```

```
250/250 [=====] - 4s 15ms/step - loss: 0.9294 - acc: 0.6948  
test accuracy  
0.6947500109672546
```

CNN 학습하기

• ResNet50 네트워크 학습시키기

- 실습에서 진행한 것 처럼 보고서에 자세한 설명 필요(실습 참고)
- Cifar-100 데이터셋 학습시키기(테스트 데이터 정확도 30% 이상)

```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar100.load_data()

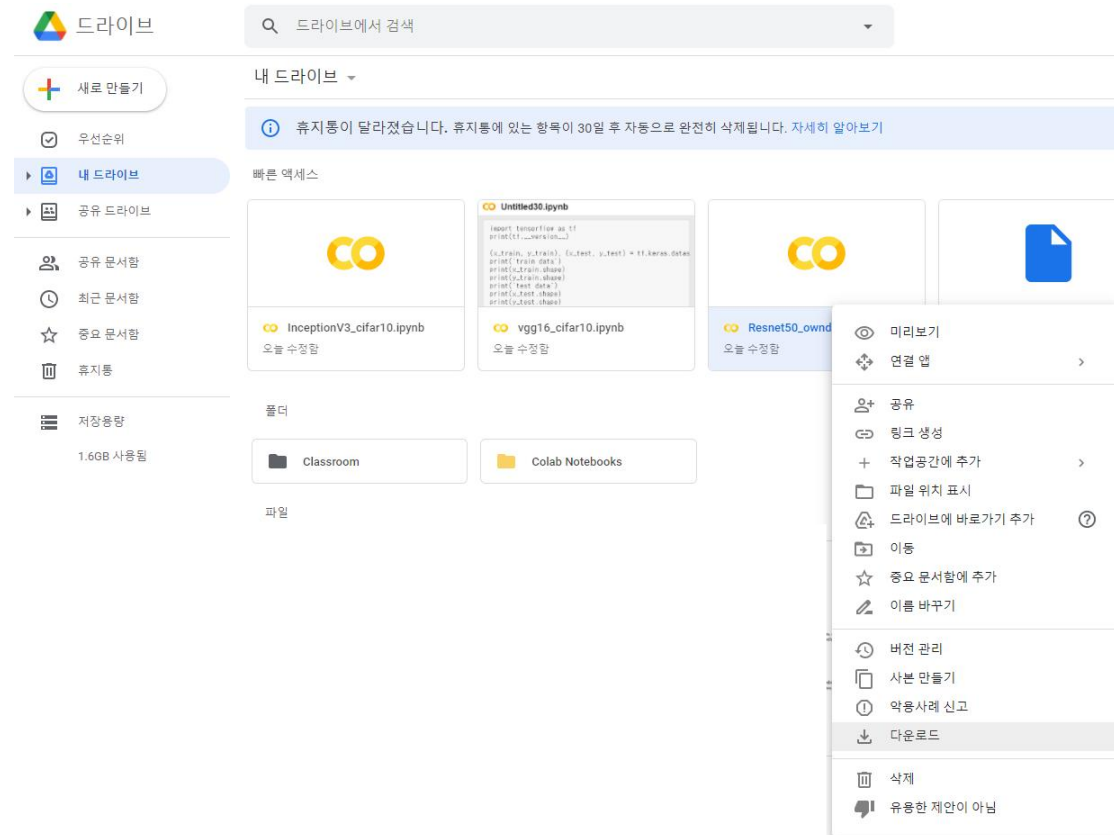
print('train data')
print(x_train.shape)
print(y_train.shape)

print('test data')
print(x_test.shape)
print(y_test.shape)
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-100-python.tar.gz
169009152/169001437 [=====] - 6s 0us/step
train data
(50000, 32, 32, 3)
(50000, 1)
test data
(10000, 32, 32, 3)
(10000, 1)
```


과제 - 코드

- 자신이 실행한 코드(.py파일 혹은 .ipynb) 제출
 - 구글 드라이브에서 다운 가능



과제 - 보고서

- 보고서

- 내용:

- 이름, 학번, 학과
 - 구현 코드: 구현한 코드(테스트 데이터 정확도 30% 이상)
 - 코드 설명: 구현한 코드에 대한 설명(실습에서 한 것 처럼 작성하기)
 - 느낀 점 : 결과를 보고 느낀 점, 혹은 과제를 하면서 어려웠던 점 등
 - 과제 난이도: 개인적으로 생각하는 난이도 (과제가 너무 쉬운 것 같다 등)

- .pdf 파일로 제출 (이 외의 파일 형식일 경우 감점)

- 파일 이름:

- [CG]20xxxxxxx_이름_n주차_과제.pdf

과제

- **제출 기한**

- 12월 02일 23시 59분까지 (최대 점수 10점)

- **추가 제출 기한**

- 12월 09일 23시 59분까지 (최대 점수 4점, 과제 총점 계산 후 -6점)
- 12월 10일 00시 00분 이후 (점수 0점)

- **채점**

- 구현을 못하거나(잘못 구현하거나) 보고서 내용이 빠진 경우 감점
- 아무것도 구현하지 못해도 과제 제출하면 기본점수 있음
- 다른 사람의 코드를 copy해서 제출시 보여준 사람, copy한 사람 둘 다 0점

- **제출 파일**

- 아래의 파일을 압축해서 [CG]20xxxxxxx_이름_n주차_과제.zip로 제출
 - .py 파일 전부 or .ipynb 파일 전부
 - .pdf 보고서 파일
 - 데이터셋은 첨부할 필요 없음

QnA