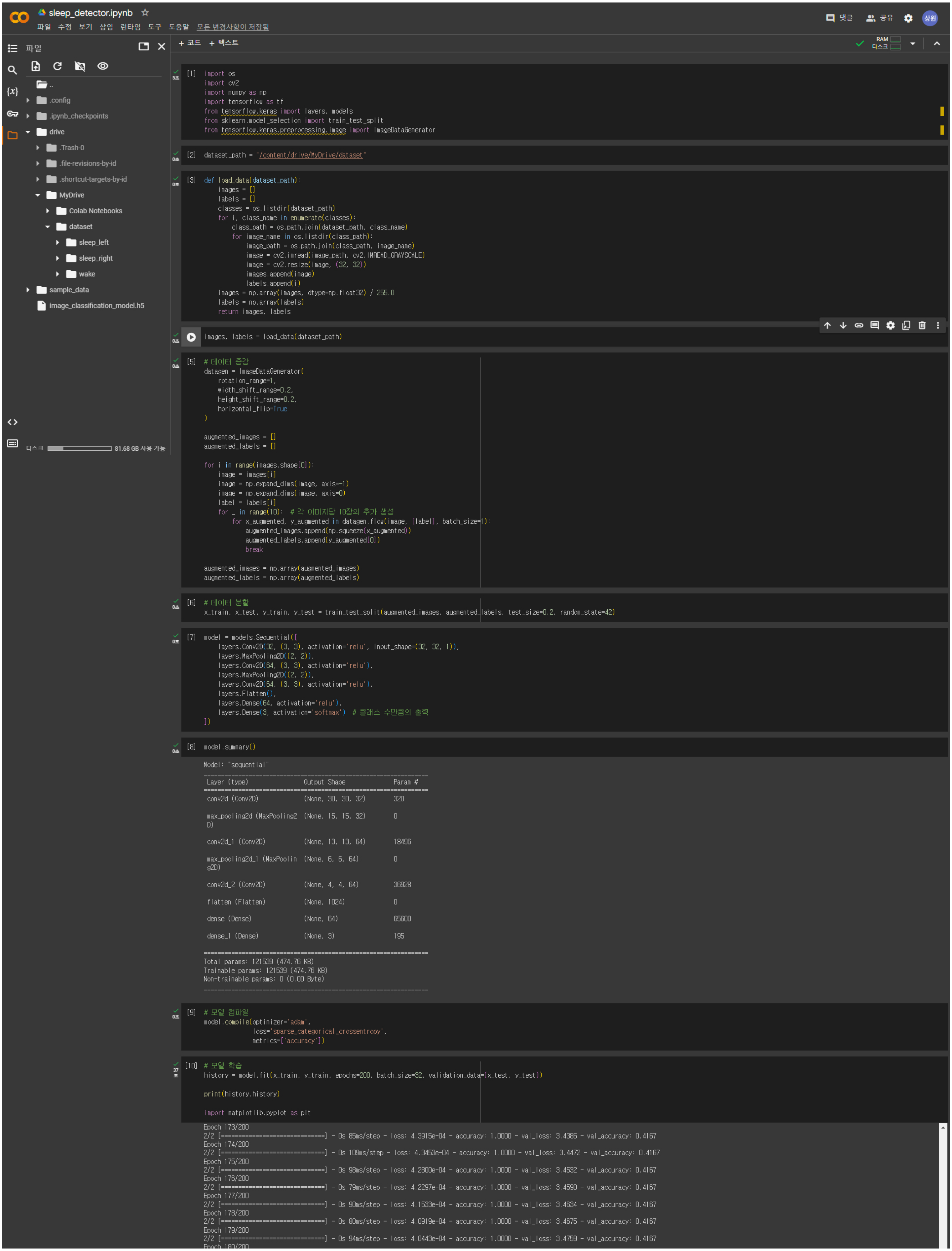


```
[1]  import os
     import cv2
     import numpy as np
     import tensorflow as tf
     from tensorflow.keras import layers, models
     from sklearn.model_selection import train_test_split
     from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
[2]  dataset_path = "/content/drive/MyDrive/dataset"
```

```
[3]  def load_data(dataset_path):
         images = []
         labels = []
         classes = os.listdir(dataset_path)
         for i, class_name in enumerate(classes):
             class_path = os.path.join(dataset_path, class_name)
             for image_name in os.listdir(class_path):
                 image_path = os.path.join(class_path, image_name)
                 image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
                 image = cv2.resize(image, (32, 32))
                 images.append(image)
                 labels.append(i)
         images = np.array(images, dtype=np.float32) / 255.0
         labels = np.array(labels)
         return images, labels
```

```
     images, labels = load_data(dataset_path)
```

```
[5]  # 데이터 증강
     datagen = ImageDataGenerator(
         rotation_range=1,
         width_shift_range=0.2,
         height_shift_range=0.2,
         horizontal_flip=True
     )

     augmented_images = []
     augmented_labels = []

     for i in range(images.shape[0]):
         image = images[i]
         image = np.expand_dims(image, axis=-1)
         image = np.expand_dims(image, axis=0)
         label = labels[i]
         for _ in range(10):  # 각 이미지당 10장의 추가 생성
             for x_augmented, y_augmented in datagen.flow(image, [label], batch_size=1):
                 augmented_images.append(np.squeeze(x_augmented))
                 augmented_labels.append(y_augmented[0])
                 break

     augmented_images = np.array(augmented_images)
     augmented_labels = np.array(augmented_labels)
```

```
[6]  # 데이터 분할
     x_train, x_test, y_train, y_test = train_test_split(augmented_images, augmented_labels, test_size=0.2, random_state=42)
```

```
[7]  model = models.Sequential([
         layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 1)),
         layers.MaxPooling2D((2, 2)),
         layers.Conv2D(64, (3, 3), activation='relu'),
         layers.MaxPooling2D((2, 2)),
         layers.Conv2D(64, (3, 3), activation='relu'),
         layers.Flatten(),
         layers.Dense(64, activation='relu'),
         layers.Dense(3, activation='softmax')  # 클래스 수만큼의 출력
     ])
```

```
[8]  model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        320

 max_pooling2d (MaxPooling2  (None, 15, 15, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 13, 13, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 6, 6, 64)          0
 g2D)

 conv2d_2 (Conv2D)           (None, 4, 4, 64)          36928

 flatten (Flatten)           (None, 1024)              0

 dense (Dense)               (None, 64)                65600

 dense_1 (Dense)             (None, 3)                 195

=================================================================
Total params: 121539 (474.76 KB)
Trainable params: 121539 (474.76 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

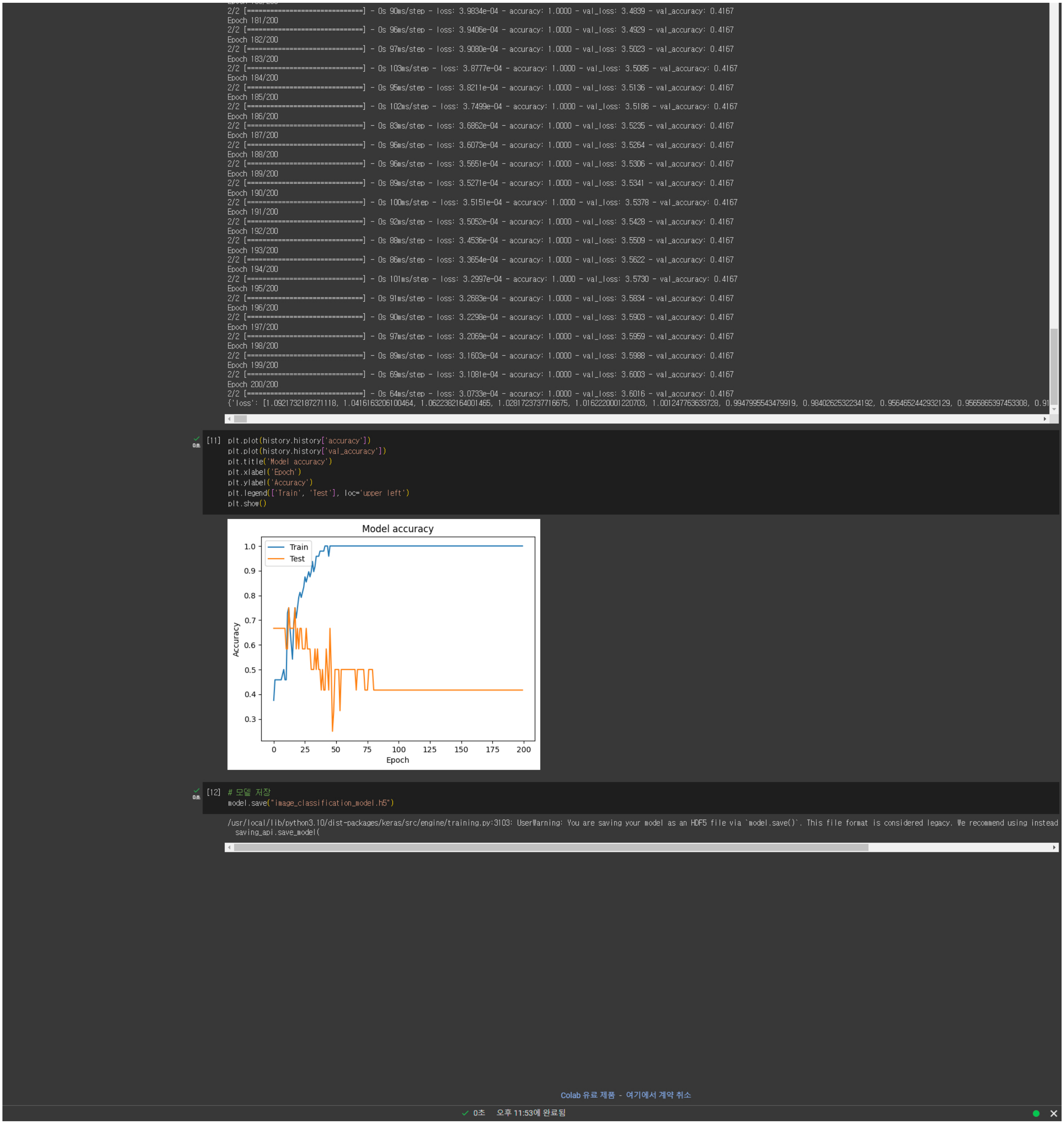
```
[9]  # 모델 컴파일
     model.compile(optimizer='adam',
                   loss='sparse_categorical_crossentropy',
                   metrics=['accuracy'])
```

```
[10] # 모델 학습
     history = model.fit(x_train, y_train, epochs=200, batch_size=32, validation_data=(x_test, y_test))

     print(history.history)

     import matplotlib.pyplot as plt
```

```
Epoch 173/200
2/2 [==============================] - 0s 85ms/step - loss: 4.3915e-04 - accuracy: 1.0000 - val_loss: 3.4386 - val_accuracy: 0.4167
Epoch 174/200
2/2 [==============================] - 0s 109ms/step - loss: 4.3453e-04 - accuracy: 1.0000 - val_loss: 3.4472 - val_accuracy: 0.4167
Epoch 175/200
2/2 [==============================] - 0s 98ms/step - loss: 4.2800e-04 - accuracy: 1.0000 - val_loss: 3.4532 - val_accuracy: 0.4167
Epoch 176/200
2/2 [==============================] - 0s 79ms/step - loss: 4.2297e-04 - accuracy: 1.0000 - val_loss: 3.4590 - val_accuracy: 0.4167
Epoch 177/200
2/2 [==============================] - 0s 90ms/step - loss: 4.1533e-04 - accuracy: 1.0000 - val_loss: 3.4634 - val_accuracy: 0.4167
Epoch 178/200
2/2 [==============================] - 0s 80ms/step - loss: 4.0919e-04 - accuracy: 1.0000 - val_loss: 3.4675 - val_accuracy: 0.4167
Epoch 179/200
2/2 [==============================] - 0s 94ms/step - loss: 4.0443e-04 - accuracy: 1.0000 - val_loss: 3.4759 - val_accuracy: 0.4167
Epoch 180/200
```



```
2/2 [==============================] - 0s 90ms/step - loss: 3.9834e-04 - accuracy: 1.0000 - val_loss: 3.4839 - val_accuracy: 0.4167
Epoch 181/200
2/2 [==============================] - 0s 96ms/step - loss: 3.9406e-04 - accuracy: 1.0000 - val_loss: 3.4929 - val_accuracy: 0.4167
Epoch 182/200
2/2 [==============================] - 0s 97ms/step - loss: 3.9080e-04 - accuracy: 1.0000 - val_loss: 3.5023 - val_accuracy: 0.4167
Epoch 183/200
2/2 [==============================] - 0s 103ms/step - loss: 3.8777e-04 - accuracy: 1.0000 - val_loss: 3.5085 - val_accuracy: 0.4167
Epoch 184/200
2/2 [==============================] - 0s 95ms/step - loss: 3.8211e-04 - accuracy: 1.0000 - val_loss: 3.5136 - val_accuracy: 0.4167
Epoch 185/200
2/2 [==============================] - 0s 102ms/step - loss: 3.7499e-04 - accuracy: 1.0000 - val_loss: 3.5186 - val_accuracy: 0.4167
Epoch 186/200
2/2 [==============================] - 0s 83ms/step - loss: 3.6862e-04 - accuracy: 1.0000 - val_loss: 3.5235 - val_accuracy: 0.4167
Epoch 187/200
2/2 [==============================] - 0s 96ms/step - loss: 3.6073e-04 - accuracy: 1.0000 - val_loss: 3.5264 - val_accuracy: 0.4167
Epoch 188/200
2/2 [==============================] - 0s 96ms/step - loss: 3.5651e-04 - accuracy: 1.0000 - val_loss: 3.5306 - val_accuracy: 0.4167
Epoch 189/200
2/2 [==============================] - 0s 89ms/step - loss: 3.5271e-04 - accuracy: 1.0000 - val_loss: 3.5341 - val_accuracy: 0.4167
Epoch 190/200
2/2 [==============================] - 0s 100ms/step - loss: 3.5151e-04 - accuracy: 1.0000 - val_loss: 3.5378 - val_accuracy: 0.4167
Epoch 191/200
2/2 [==============================] - 0s 92ms/step - loss: 3.5052e-04 - accuracy: 1.0000 - val_loss: 3.5428 - val_accuracy: 0.4167
Epoch 192/200
2/2 [==============================] - 0s 88ms/step - loss: 3.4536e-04 - accuracy: 1.0000 - val_loss: 3.5509 - val_accuracy: 0.4167
Epoch 193/200
2/2 [==============================] - 0s 86ms/step - loss: 3.3654e-04 - accuracy: 1.0000 - val_loss: 3.5622 - val_accuracy: 0.4167
Epoch 194/200
2/2 [==============================] - 0s 101ms/step - loss: 3.2997e-04 - accuracy: 1.0000 - val_loss: 3.5730 - val_accuracy: 0.4167
Epoch 195/200
2/2 [==============================] - 0s 91ms/step - loss: 3.2683e-04 - accuracy: 1.0000 - val_loss: 3.5834 - val_accuracy: 0.4167
Epoch 196/200
2/2 [==============================] - 0s 90ms/step - loss: 3.2298e-04 - accuracy: 1.0000 - val_loss: 3.5903 - val_accuracy: 0.4167
Epoch 197/200
2/2 [==============================] - 0s 97ms/step - loss: 3.2069e-04 - accuracy: 1.0000 - val_loss: 3.5959 - val_accuracy: 0.4167
Epoch 198/200
2/2 [==============================] - 0s 89ms/step - loss: 3.1603e-04 - accuracy: 1.0000 - val_loss: 3.5988 - val_accuracy: 0.4167
Epoch 199/200
2/2 [==============================] - 0s 69ms/step - loss: 3.1081e-04 - accuracy: 1.0000 - val_loss: 3.6003 - val_accuracy: 0.4167
Epoch 200/200
2/2 [==============================] - 0s 64ms/step - loss: 3.0733e-04 - accuracy: 1.0000 - val_loss: 3.6016 - val_accuracy: 0.4167
{'loss': [1.0921732187271118, 1.0416163206100464, 1.0622382164001465, 1.0281723737716675, 1.0162220001220703, 1.001247763633728, 0.9947995543479919, 0.9840262532234192, 0.9564652442932129, 0.9565865397453308, 0.91
```

```
[11] plt.plot(history.history['accuracy'])
     plt.plot(history.history['val_accuracy'])
     plt.title('Model accuracy')
     plt.xlabel('Epoch')
     plt.ylabel('Accuracy')
     plt.legend(['Train', 'Test'], loc='upper left')
     plt.show()
```

```
[12] # 모델 저장
     model.save("image_classification_model.h5")
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead
  saving_api.save_model(
```

Colab 유료 제품 - 여기에서 계약 취소
✓ 0초   오후 11:53에 완료됨