sleep_detector.ipynb

파일 수정 보기 삽입 런타임 도구 도움말

댓글 공유 상원

+ 코드 + 텍스트

RAM 디스크

파일

- ..
- .config
- .ipynb_checkpoints
- drive
  - .Trash-0
  - .file-revisions-by-id
  - .shortcut-targets-by-id
  - MyDrive
    - Colab Notebooks
    - dataset
      - sleep_left
      - sleep_right
      - wake
  - sample_data
  - image_classification_model.h5

디스크 81.66 GB 사용 가능

[1] 9초
```
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] 0초
```
dataset_path = "/content/drive/MyDrive/dataset"
```

[3] 0초
```
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
```

[4] 0초
```
images, labels = load_data(dataset_path)
```

[5] 0초
```
# 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] 0초
```
# 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] 0초
```
model = models.Sequential([
    layers.Conv2D(128, (3, 3), activation='relu', input_shape=(32, 32, 1)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(256, (3, 3), activation='relu'),
    layers.Flatten(),
    layers.Dense(256, activation='relu'),
    layers.Dense(3, activation='softmax')  # 클래스 수만큼의 출력
])
```

[8] 0초
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 128)       1280

 max_pooling2d (MaxPooling2  (None, 15, 15, 128)       0
 D)

 conv2d_1 (Conv2D)           (None, 13, 13, 128)       147584

 max_pooling2d_1 (MaxPoolin  (None, 6, 6, 128)         0
 g2D)

 conv2d_2 (Conv2D)           (None, 4, 4, 256)         295168

 flatten (Flatten)           (None, 4096)              0

 dense (Dense)               (None, 256)               1048832

 dense_1 (Dense)             (None, 3)                 771

=================================================================
Total params: 1493635 (5.70 MB)
Trainable params: 1493635 (5.70 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[9] 0초
```
# 모델 컴파일
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

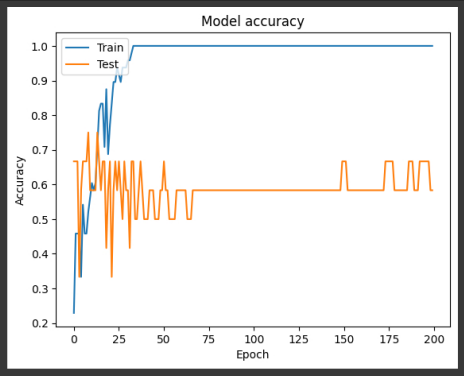
[10] 2분
```
# 모델 학습
history = model.fit(x_train, y_train, epochs=200, batch_size=32, validation_data=(x_test, y_test))

print(history.history)

import matplotlib.pyplot as plt
```

```
Epoch 161/200
2/2 [==============================] - 0s 146ms/step - loss: 2.0853e-04 - accuracy: 1.0000 - val_loss: 2.4497 - val_accuracy: 0.5833
Epoch 162/200
2/2 [==============================] - 0s 143ms/step - loss: 2.0724e-04 - accuracy: 1.0000 - val_loss: 2.4536 - val_accuracy: 0.5833
Epoch 163/200
2/2 [==============================] - 0s 159ms/step - loss: 2.0419e-04 - accuracy: 1.0000 - val_loss: 2.4499 - val_accuracy: 0.5833
Epoch 164/200
2/2 [==============================] - 0s 150ms/step - loss: 2.0002e-04 - accuracy: 1.0000 - val_loss: 2.4475 - val_accuracy: 0.5833
Epoch 165/200
2/2 [==============================] - 0s 167ms/step - loss: 1.9739e-04 - accuracy: 1.0000 - val_loss: 2.4397 - val_accuracy: 0.5833
Epoch 166/200
2/2 [==============================] - 0s 156ms/step - loss: 1.9440e-04 - accuracy: 1.0000 - val_loss: 2.4362 - val_accuracy: 0.5833
Epoch 167/200
2/2 [==============================] - 0s 154ms/step - loss: 1.9229e-04 - accuracy: 1.0000 - val_loss: 2.4346 - val_accuracy: 0.5833
```

```
Epoch 168/200
2/2 [==============================] - 0s 165ms/step - loss: 1.9032e-04 - accuracy: 1.0000 - val_loss: 2.4368 - val_accuracy: 0.5833
Epoch 169/200
2/2 [==============================] - 0s 145ms/step - loss: 1.8818e-04 - accuracy: 1.0000 - val_loss: 2.4426 - val_accuracy: 0.5833
Epoch 170/200
2/2 [==============================] - 0s 147ms/step - loss: 1.8618e-04 - accuracy: 1.0000 - val_loss: 2.4555 - val_accuracy: 0.5833
Epoch 171/200
2/2 [==============================] - 0s 162ms/step - loss: 1.8314e-04 - accuracy: 1.0000 - val_loss: 2.4635 - val_accuracy: 0.5833
Epoch 172/200
2/2 [==============================] - 0s 186ms/step - loss: 1.8045e-04 - accuracy: 1.0000 - val_loss: 2.4650 - val_accuracy: 0.5833
Epoch 173/200
2/2 [==============================] - 0s 140ms/step - loss: 1.7777e-04 - accuracy: 1.0000 - val_loss: 2.4618 - val_accuracy: 0.5833
Epoch 174/200
2/2 [==============================] - 0s 156ms/step - loss: 1.7577e-04 - accuracy: 1.0000 - val_loss: 2.4611 - val_accuracy: 0.6667
Epoch 175/200
2/2 [==============================] - 0s 153ms/step - loss: 1.7376e-04 - accuracy: 1.0000 - val_loss: 2.4621 - val_accuracy: 0.6667
Epoch 176/200
2/2 [==============================] - 0s 152ms/step - loss: 1.7186e-04 - accuracy: 1.0000 - val_loss: 2.4620 - val_accuracy: 0.6667
Epoch 177/200
2/2 [==============================] - 0s 165ms/step - loss: 1.6973e-04 - accuracy: 1.0000 - val_loss: 2.4653 - val_accuracy: 0.6667
Epoch 178/200
2/2 [==============================] - 0s 147ms/step - loss: 1.6768e-04 - accuracy: 1.0000 - val_loss: 2.4676 - val_accuracy: 0.6667
Epoch 179/200
2/2 [==============================] - 0s 140ms/step - loss: 1.6550e-04 - accuracy: 1.0000 - val_loss: 2.4726 - val_accuracy: 0.5833
Epoch 180/200
2/2 [==============================] - 0s 188ms/step - loss: 1.6341e-04 - accuracy: 1.0000 - val_loss: 2.4767 - val_accuracy: 0.5833
Epoch 181/200
2/2 [==============================] - 0s 152ms/step - loss: 1.6132e-04 - accuracy: 1.0000 - val_loss: 2.4809 - val_accuracy: 0.5833
Epoch 182/200
2/2 [==============================] - 0s 139ms/step - loss: 1.5983e-04 - accuracy: 1.0000 - val_loss: 2.4843 - val_accuracy: 0.5833
Epoch 183/200
2/2 [==============================] - 0s 154ms/step - loss: 1.5767e-04 - accuracy: 1.0000 - val_loss: 2.4828 - val_accuracy: 0.5833
Epoch 184/200
2/2 [==============================] - 0s 200ms/step - loss: 1.5698e-04 - accuracy: 1.0000 - val_loss: 2.4799 - val_accuracy: 0.5833
Epoch 185/200
2/2 [==============================] - 1s 260ms/step - loss: 1.5381e-04 - accuracy: 1.0000 - val_loss: 2.4868 - val_accuracy: 0.5833
Epoch 186/200
2/2 [==============================] - 1s 233ms/step - loss: 1.5234e-04 - accuracy: 1.0000 - val_loss: 2.4958 - val_accuracy: 0.5833
Epoch 187/200
2/2 [==============================] - 1s 237ms/step - loss: 1.5066e-04 - accuracy: 1.0000 - val_loss: 2.4991 - val_accuracy: 0.6667
Epoch 188/200
2/2 [==============================] - 1s 224ms/step - loss: 1.4919e-04 - accuracy: 1.0000 - val_loss: 2.5010 - val_accuracy: 0.6667
Epoch 189/200
```

0초

```
[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()
```



0초

```
[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(
```


✓ 0초 오전 12:09에 완료됨