```
#*******************************************************************************
# Image multi-classification using keras
#
# conda create -n recogvoice tensorflow keras pillow pydot  opencv
# conda activate recogvoice
# pip install matplotlib
#*******************************************************************************
# 아래 각 메쏘드(함수)의  설명은 http://keras.io 사이트에서 참조

import numpy as np
import os, cv2

from keras import models
from keras.models import Sequential, save_model, load_model
from keras.layers import Dense, Flatten, Dropout
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.preprocessing import image
from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array, load_img
from keras.callbacks import ModelCheckpoint
from keras.utils.vis_utils import plot_model

from PIL import Image           # PIL: Python Image Library (pillow 설치)
import matplotlib.pyplot as plt


os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'    # 텐서플로우 관련 warning 메세지 출력 방지
    # 프로그램 실행시 "Your CPU supports instructions that this TensorFlow binary
    #       was not compiled to use: AVX AVX2" 안보이게 텐서플로우 환경변수값 변
#np.random.seed(15)                          # 반복적 실행에도 같은 결과를 얻고 싶을 경우 사용

TRAIN_FOLDER = 'train'
TEST_FOLDER = 'test'
TARGET_SIZE = (64, 64)
INPUT_SHAPE = (64, 64, 3)

#==============================================================================
# [0] 폴더 갯수로 분류할 Class 갯수 확인
#==============================================================================
total_folder_no=0
for _, dirnames, filenames in os.walk(TRAIN_FOLDER):    # 디렉토리 리스트와 파일리스트의 튜플 반환
    #total_file_no += len(filenames)
    total_folder_no += len(dirnames)

total_class_no = total_folder_no
print("Wn 분류할 Class 수: ",total_class_no)

#==============================================================================
# [1] Data Preparation
#       학습과 검증, 테스트에 사용할 데이터 준비
#           참조: https://keras.io/api/preprocessing/image/ (영어)
#                 https://keras.io/ko/preprocessing/image/ (한글)
#==============================================================================
# ImageDataGenerator 생성
# 지정한 폴더에서 증강된(augmented) 데이터 배치(batch, 데이터 묶음)을 생성
# 입력 데이터 값에 관계없이 0~1 사이의 실수값으로 변경
train_datagen = ImageDataGenerator(
    rotation_range = 0,         # 정수값, random 회전 각도
    width_shift_range=0.2,      # (1/3)정수값: 예) 2일경우 [-2, -1, 0, 1, 2] 중 '임의' 정수 값
                                # (2/3)실수값: 예) 1.0보다 큰 2.0 일경우 [-2.0 ~ 2.0) 사이 임의 실수값
                                #               1.0보다 작은 실수값 : 이미지 가로길이의 임의 비율값만큼 shift
                                # (3/3)1차원 배열값: 예) [-3, -1, 1, 3] 에서 임의의 값
    height_shift_range=0.0,     # width_shift_range와 마찬가지 방식으로 방향만 세로로 적용
    brightness_range=None,      # 튜플이나, 두값을 가진 리스트: 주어진 범위내 임의 밝기값 만큼 shift
                                #           예) [-5, +5] : 읽은 밝기값에 -5에서 +5 사이의 임의 밝기값 만큼 더함
    shear_range=0.0,            # 실수값: 반시계 방향으로 임의 도(degree) 단위로 shearing 함 (shear변형)
                                # 사각형 object -> 평행사변형, 원->타원이 될 수 있음
    zoom_range=0.0,             # 실수값: [lower, upper] 범위내 임의값으로 확대
                                #         1보다 작은값이면 [1-실수값, 1+실수값] 사이의 임의 값 적용
    fill_mode='nearest',        # 입력이미지 바깥 테두리 경계값 설정 방법.
                                #       {"constant", "nearest", "reflect" 혹은 "wrap"} 중 하나
    horizontal_flip = False,    # 좌우 반전
    vertical_flip = False,      # 상하 반전
    rescale = 1/255.0,          # 입력값 재조정 (이미지 각 화소값에 곱하기 하는 값)
    data_format='channels_last' # 케라스 환경설정 파일(~/.keras/keras.json)에 지정한 대로 설정됨
```

```
                                #   기본설정은 'channels_last' 임 -> (samples, height, width, channels)
    #validation_split=          # 실수값: 입력데이터 중, validation용 이미지 데이터 비율 ( 0~1 사이값)
    #featurewise_std_normalization= # Boolean: 입력데이터를 '전체 데이터세트의 평균값'으로 나눔
    #samplewise_std_normalization:= # Boolean: 각 입력이미지를 '각 입력이미지 하나의 평균값'으로 나눔
    #.... 몇개 더 있음
)

test_datagen =  ImageDataGenerator(rescale = 1/255.)

# [1-1] 학습용 데이터 (Augmentation) 세트 생성
    # 실시간으로 데이터 증강시키면서 'tensor image data의 묶음(batch)' 생성
    # Gradient Descent : 에러 Gradient값을 낮추는 방향으로 반복해서 학습하는 최적화 방법
    # Batch Size : 신경망 내부 파라미터 값을 업데이트 하기 전에 계산에 참여시키는 샘플데이터의 수
    # Hyperparameter : 학습시 사용자가 직접 정해야 하는 변수 (batch_size, learning_rate...)
train_generator = train_datagen.flow_from_directory (
    TRAIN_FOLDER,               # 이미지 데이터가 들어있는 폴더명
    target_size=TARGET_SIZE,    # 이미지 크기 정규화 (세로x가로)
    batch_size=3,               # 정수값 : 학습시, Weight를 업데이트 하는 간격간의 데이터 갯수를 나타냄
                                #     Stochastic Gradient Descent(=1): 한 iteration에 하나의 샘플만 사용
                                #                         학습변화속도는 빠름/잦은 업데이트로 학습시간은 김/noisy update문제
                                #     Batch Gradient Descent(=N) : 전체 데이터(N)에 대한 에러 계산 후 업데이트
                                #                         학습변화 속도 느림/계산시간 적음/보다 안정된 학습/메모리 사용량 큼
                                #     Mini-Batch SGD(<N) : 일부 데이터에 대한 에러 계산 후 파라미터 업데이트 (일반적임)
                                #                         경험적으로 32(경험적으로 일반적 값), 64, 128 등이 사용됨
                                # https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/
    class_mode='categorical',   # 분류방식 : 출력층을 사용하는 방식 결정 (binary, categorical, sparse, None)
                                #   다중분류-categorical, 이진분류-binary, 라벨미반환: none
    shuffle = True,             # 데이터 생성 순서를 난수로 섞을지 결정 (train:True, test:False)
                                #--------------아래는 데이터 변형 관련 설정, 설정 안하면 기본값으로 설정됨---------
    )

    # [1-2] 테스트용 데이터에 대하여도 마찬가지로 데이터 세트 생성
test_generator = test_datagen.flow_from_directory (
    TEST_FOLDER, target_size=TARGET_SIZE, batch_size=1, shuffle = False, class_mode='categorical')

#=============================================================================
# [2] Model Construction
#       신경망의 구조 생성
#=============================================================================
#es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=100)
model = Sequential()            # kera가 제공하는 신경망 모델 종류
                                #   Sequential: layer간 공유가 없고, 입출력 laye가 각각하나씩 뿐임
                                #   Functional: 인접하지 않은 layer간 연결이 자유롭고, 복잡한 모델 생성시 사용
                                #     (https://machinelearningmastery.com/keras-functional-api-deep-learning/)
    # 입력층과 연결될 첫번째 layer 생성
model.add(Conv2D(32, kernel_size=(3,3), activation='relu', input_shape=INPUT_SHAPE))
    # 컨벌루션 필터의 갯수 : 32개, 컨벌루션 레이어 각 이미지 크기는 kernel_size에 의해 자동으로 결정됨
    # kernel_size : 2차원 컨벌루션 윈도(필터)의 가로,세로 크기
    # activation : 각 뉴런의 활성화 함수 (linear, sigmoid, softmax, retifier(relu), ...)
    # input_shape : 입력 영상의 크기(줄,열,채널수), 입력과 연결된 첫번째 레이어에서만 사용
    # 출력층은 3x3 kernel_size로 인해 30x26 크기의 이미지 32개가 됨
model.add(MaxPooling2D(pool_size=(2,2)))
    # 앞 레이어 영상에서 2x2 즉, 4개의 점 영역에서 가장 큰 값 하나만 다음 레이어로 전달
    # (효과1)인근픽셀(2x2 영역)들의 미세한 위치 이동에 무관한 학습이 가능
    # (효과2)네트웍의 크기를 줄여줌, 값이 상대적으로 큰 픽셀이 학습에 기여하는 바가 크다는 전제
    # 출력층은 2x2 pool_size값으로 인해 15x13 크기의 이미지 32개가 됨
model.add(Dropout(0.2))
    # 학습용 특정 데이터에 과적합(Overfitting) 되는 문제를 막기 위함
    # 지정한 비율(rate)의 데이터 만큼 난수로 선택된 노드들의 입력값을 강제적으로 0으로 세팅
    # 나머지 데이터는 대신 1/(1-rate) 값 만큼 증가 시켜서 전체 입력값의 합은 변하지 않게 한다
    # 학습시에만 사용되도록, keras에서 model.fit() 사용시 training 값이 True 일때만 적용됨
    # 네트워크 size 변화는 없음

model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
    # 출력층은 3x3 kernel_size값으로 인해 13x11 크기의 이미지 64개가 됨
model.add(MaxPooling2D(pool_size=(2,2)))
    # 출력층은 2x2 pool_size값으로 인해 6x5 크기의 이미지 64개가 됨
model.add(Dropout(0.2))

model.add(Flatten())
    # 1차원 레이어 구조 추가
    # 출력노드수는 6x5x64 = 1920
model.add(Dense(64, activation='relu'))
    # 출력노드수는 64개
model.add(Dense(total_class_no, activation='softmax'))
```

```
    #출력 노드수는 클래스 수인 10개
save_model(model, 'classify_model.tf')  # tensorflow 버젼 2.x 용 형식으로 저장
    # 모델 데이터 전체(모델 구조, weights, 모델의 최적화 상태) 저장
    # model = load_model('classify_model.tf') 학습한 모델정보와 weight를 불러올 때 사용
    # model.save_weights('myocr_model_weights.tf') 학습한 weight만 저장

# keras의 모델 관련 출력함수 2개로 모델 상황 출력
print(model.summary())
#plot_model(model, to_file='model_plot.png', show_shapes=True, show_layer_names=True)

#================================================================================
# [3] Configuring Training
#       학습 방식에 대한 환경(최적화방법, 손실함수, 정확성 측정 기준) 설정
#================================================================================
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

#================================================================================
# [4] Model Training
#       학습데이터를 이용하여 실제 학습
#================================================================================
# 최적 학습시의 model/weights를 저장하거나 중단 후 학스을 재개하기 위하여 체크포인트 설정
checkpointer = ModelCheckpoint(filepath="best_weights.hdf5",
                               monitor= 'val_accuracy', mode='max', verbose=1,
                               save_weights_only=True, save_best_only=True)
    # https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/ModelCheckpoint

step_size_train = train_generator.n // train_generator.batch_size
#step_size_valid = valid_generator.n // valid_generator.batch_size
step_size_test = test_generator.n // test_generator.batch_size

history= model.fit_generator(train_generator, steps_per_epoch=step_size_train,
    callbacks = [checkpointer], #callbacks = [es],
    validation_data=train_generator,validation_steps=step_size_train, epochs=200)
        # steps_per_epoch : 한번의 epoch에서 처리할 batch 묶음의 수
        # epochs : steps_per_epoch에서 정한 횟수만큼 학습 시키면 한번의 epoch가 됨
        # validation_data : 학습에 참가시키지 않고, 각 epoch 후마다 검증용으로 loss값 등 평가
        # history: 매 epoch마다 loss, acc, val_loss, val_acc 값을 history에 저장

# 저장된 최적 weight를 불러오기
model.load_weights('best_weights.hdf5')

# 모델 데이터 전체(모델 구조, weights, 모델의 최적화 상태) 저장
save_model(model, 'classify_voice_model.tf')  # tensorflow 버젼 2.x 용 형식으로 저장

#================================================================================
# [5] Model Evaluation
#       학습 상태 평가
#================================================================================
print('Wn********** Evaluation **********')

scores = model.evaluate_generator(train_generator, steps=step_size_train)

for i in range(len(model.metrics_names)) :
    print('%s: %.2f %%' %(model.metrics_names[i], scores[i]*100))

#================================================================================
# [6] Model Prediction
#       테스트 데이터에 대한 예측
#================================================================================
print('Wn********** Prediction **********')
np.set_printoptions(formatter={'float': lambda x: '{0:0.3f}'.format(x)})
    # 소수점 이하 3자리 까지만 출력

print(test_generator.class_indices)
    # 각 폴더가 무슨(몇번째) 카테고리 데이터(class)를 의미하는지 출력
    # {'0': 0, '1': 1, '2': 2, '3': 3, '4': 4, '5': 5, '6': 6, '7': 7, '8': 8, '9': 9} 출력

cls_md = "Class Mode: "+test_generator.class_mode
print(cls_md)
    # 출력되는 클래스 모드 출력 : Class Mode: categorical
    # Categirocal : 미리 정한 카테고리에 대해 0~1 사이의 값 출력,
    #               출력값이 제일 큰 클래스를 선택한 답으로 간주하는 방식

print('Wn[[[ 학습한 데이터에 대한 predition 및 출력 - 임의순서로 출력 ]]]')
output = model.predict_generator(train_generator, steps=step_size_train)
```

```
print(output)

print('Wn[[[ 테스트 데이터에 대한 predition 및 출력 - 폴더명 알파벳순으로 출력 ]]]')
output = model.predict_generator(test_generator, steps=step_size_test, verbose=1)

#-----------------------------------------
# 테스트 데이터에 대한 분류 결과 모두 출력하기
#-----------------------------------------
test_filenames = []
for file in test_generator.filenames :      # 출력을 위해 테스트용 파일명 리스트 먼저 생성
    test_filenames.append(file)
    #print(file)

for no in range(len(output)) :
    print("Wn[",no, "]번째 이미지 ", test_filenames[no], " 에 대한 분류 결과")
    print('Wt',output[no])                  # 인식과 출력층 값 출력
    maxValIndices = [i for i, x in enumerate(output[no]) if x == max(output[no])]
    print("Wt계산한 답은 ",end = '')
    print(maxValIndices)


#-----------------------------------------
# (matplotlib 라이브러리를 이용하여) 학습중 정확도와 손실 관련 값 그리프로 출력
#-----------------------------------------
    # 기록된 history 값 출력 ==> dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy'])
print(history.history.keys())

    # history 패러미터 값을 그래프로 출력
plt.plot(history.history['accuracy'])
plt.plot(history.history['loss'])

    # 그래프 제목 출력
plt.title('model accuracy & loss')
    # 왼쪽/아랫쪽에 y/x 축의 의미를 나타내는 라벨값 출력
plt.ylabel('accuracy/loss')
plt.xlabel('epoch')
    # 왼쪽 상단에 범례 출력
plt.legend(['accuracy', 'loss'], loc='upper left')
print("Wn종료하려면 그래프 출력 창을 닫으시오.Wn")
plt.show()


#=================================================
# [7] (테스트용 입력 데이터에 대한) Hidden Layer Activation 시각화
#=================================================
# 입력 영상에 대한 모든 convolution layer와 pooling layer의 activation을
# 출력하는 Keras model 생성

    # (1) 테스트용 이미지 불러와서 tensor로 변환
img_path = 'nice to know you-test.jpg'                  # 테스트용 숫자 영상 한장

img = image.load_img(img_path, target_size=TARGET_SIZE)    # PIL format으로 불러오기
img_tensor = image.img_to_array(img)                    # Numpy Array로 변경
img_tensor = np.expand_dims(img_tensor, axis=0)         # 차원 추가 (1, 32, 28, 3)
print(img_tensor.shape)
plt.imshow(img_tensor[0])
img_tensor /= 255.

#plt.imshow(img_tensor[0])
print(img_tensor)
plt.show()

    # (2) 앞서 만든 모델(model)의 레이어 중에서,
    #       앞쪽의 몇개 레이어를 출력노드로 하는 새로운 모델(이름을 activation_model로) 생성
    #       이 모델의 입력은 앞에서 만든 모델(model)의 입력과 동일
layerNoToSee = 6
layer_outputs = [layer.output for layer in model.layers[:layerNoToSee]]
    # Creates a model that will return these outputs, given the model input
activation_model = models.Model(inputs=model.input, outputs=layer_outputs)

    # 테스트용 입력 영상을 대상으로 predict 실행하여 결과(activations)를 얻음
    # Returns a list of five Numpy arrays: one array per layer activation
activations = activation_model.predict(img_tensor)

    # 시각화
```

```
layer_names = []
for layer in model.layers[:layerNoToSee]:
    layer_names.append(layer.name) # Names of the layers

images_per_row = 16                # 화면 한 줄에 나타낼 그림의 개수 (16장/줄)

#size_row = 32
#size_col = 28

# feature maps 출력
for layer_name, layer_activation in zip(layer_names, activations):
        # 한 layer(= featue map)에 있는 feature(필터) 수
    n_features = layer_activation.shape[-1]
    print("필터[Feature] 수 :",n_features)
        #The feature map has shape (1, row_size, col_size, n_features).
    size_row = layer_activation.shape[1]
    size_col = layer_activation.shape[2]

    print("출력 image크기 [row, col] :",size_row, size_col)
    n_rows = n_features // images_per_row           # 줄 수#
    print("n_rows: ",n_rows)

    # 전체 필터를 담을 수 있는 배열 생성
    # 예: 30x26 크기의 feature map을 2줄 출력하려면 30x26(픽셀/한장)x16(장/줄)x2(줄) => 60x416
    filter_grid = []
    filter_grid = np.zeros((size_row*n_rows, size_col*images_per_row))

    #display_grid = np.zeros((size_row * n_rows, images_per_row * size_col))
    print("filter_grid shape: ",filter_grid.shape)

        # Tiles each filter into a big horizontal grid
    for row in range(n_rows):
        for col in range(images_per_row):
            channel_image = layer_activation[0,
                                             :, :,
                                             row * images_per_row + col]
            #print(channel_image)

            #-------------------------------------------------------------------
            # 특징값은 임의의 음수를 포함한 실수 값이므로, 출력가능한 0~255 사이값으로 변환
            #-------------------------------------------------------------------
            mean = channel_image.mean()
            print("Wnchannel 평균: ",mean)  # feature 평균값
            print("nchannel 표준편차: ",channel_image.std())
            channel_image -= channel_image.std()
            channel_image /= channel_image.std()
            channel_image *= 64
            channel_image += 128
            channel_image = np.clip(channel_image, 0, 255).astype('uint8')

            #print("Feature map")
            #print(channel_image)
                # 큰 그림상에 출력할 위치 선정
            row_pos = row*size_row
            col_pos = col*size_col
                # 해당 feature map을 전체 그림에 복사
            #print("row:{}, col:{}, row_pos:{}, col_pos:{}".format(row,col,row_pos,col_pos))
            filter_grid[row_pos:row_pos+size_row, col_pos:col_pos+size_col] = channel_image

            #----------------------
            # 각각의 필터 윈도우 출력
            #----------------------
            window_name = str(layer_name)+" "+str(row*images_per_row+col)
            cv2.namedWindow(window_name,cv2.WINDOW_NORMAL)
            cv2.moveWindow(window_name,col*(size_col+100), row*(size_row+200))
            cv2.imshow(window_name,channel_image)

    print(filter_grid)
    cv2.namedWindow("Filters",cv2.WINDOW_NORMAL)
    cv2.moveWindow("Filters",500,200)
    cv2.imshow('Filters',filter_grid)
    key = cv2.waitKey(0)
    if key == 27 :
        break
    cv2.destroyAllWindows()
```

```
    '''
    scale_row = 1. / size_row
    scale_col = 1. / size_col

    scale_row = 1.
    scale_col = 1.

    plt.figure(figsize=(scale_col * display_grid.shape[1],
                        scale_row * display_grid.shape[0]))
    plt.title(layer_name)
    plt.grid(False)
    #plt.show()
    print("display_grid: ",display_grid.shape[0],display_grid.shape[1])
    print(display_grid)
    #plt.imshow(display_grid, aspect='auto', cmap='viridis')
    #cv2.imshow('filters',display_grid)
    plt.imshow(display_grid)
    #cv2.waitKey(0)
    '''

cv2.destroyAllWindows()
```