Food-101 데이터 세트에서 Keras 및 TensorFlow를 사용한 다중 클래스 분류

Food-101 데이터셋 - https://www.vision.ee.ethz.ch/datasets_extra/food-101/

Notebook 원본 제공: Avinash Kappa

 Run in Google Colab  View source on GitHub

TensorFlow 2.0 미리보기 설치

- TensorFlow 2.0 미리보기를 테스트할 수 있습니다.
- 아래 코드 셀을 사용하여 TensorFlow 2.0 Preview를 설치.

```
#!pip install tf-nightly-gpu-2.0-preview
```

Food 101 Dataset 다운로드

- 필요한 라이브러리 임포트

```
from __future__ import absolute_import, division, print_function

import tensorflow as tf

import tensorflow.keras.backend as K
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
from tensorflow.keras import regularizers
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.layers import GlobalAveragePooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ModelCheckpoint, CSVLogger
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.regularizers import l2

from tensorflow import keras
from tensorflow.keras import models
from tensorflow.keras.applications.inception_v3 import preprocess_input

import cv2
import os
import random
import collections
from collections import defaultdict

from shutil import copy
from shutil import copytree, rmtree

import numpy as np

import matplotlib.pyplot as plt
import matplotlib.image as img
%matplotlib inline


# TF 버전 및 GPU 활성화 여부 확인
print(tf.__version__)
print(tf.test.gpu_device_name())
```

```
2.12.0
/device:GPU:0
```

```
# "tensorflow/examples" 저장소를 복제하여 훈련된 모델을 평가하기 위한 이미지가 있는 저장소를 만듦
!git clone https://github.com/tensorflow/examples.git
```

```
Cloning into 'examples'...
remote: Enumerating objects: 23470, done.
remote: Counting objects: 100% (430/430), done.
remote: Compressing objects: 100% (276/276), done.
remote: Total 23470 (delta 124), reused 371 (delta 86), pack-reused 23040
```

```
Receiving objects: 100% (23470/23470), 43.96 MiB | 23.24 MiB/s, done.
Resolving deltas: 100% (12773/12773), done.
```

```
# 데이터를 다운로드하고 추출하기 위한 함수

def get_data_extract():
  if "food-101" in os.listdir():
    print("Dataset already exists")
  else:
    tf.keras.utils.get_file(
    'food-101.tar.gz',
    'http://data.vision.ee.ethz.ch/cvl/food-101.tar.gz',
    cache_subdir='/content',
    extract=True,
    archive_format='tar',
    cache_dir=None
    )
    print("Dataset downloaded and extracted!")
```

- Food-101 데이터 세트의 크기는 5GB입니다. 완료하는 데 시간이 좀 걸릴 수 있습니다

```
# 데이터를 다운로드하고 폴더에 압축을 풉니다.
get_data_extract()
```

```
Downloading data from http://data.vision.ee.ethz.ch/cvl/food-101.tar.gz
4996278331/4996278331 [==============================] - 204s 0us/step
Dataset downloaded and extracted!
```

데이터 세트 구조 및 파일 이해

1. 사용되는 데이터셋은 Food 101입니다.
2. 이 데이터셋은 총 101,000개의 이미지로 구성되어 있습니다.
3. 101개의 다양한 음식 카테고리로 이루어진 다중 클래스 데이터셋입니다.
4. 각 음식 종류별로 750개의 훈련 샘플과 250개의 테스트 샘플이 있습니다.
5. 데이터셋 웹페이지에는 다음과 같은 주의사항이 있습니다: 훈련 이미지는 일부 노이즈가 포함된 상태로 처리되었습니다. 이는 주로 강렬한 색상이나 때로는 잘못된 레이블로 나타날 수 있습니다.
6. 모든 이미지는 한쪽의 최대 길이가 512 픽셀이 되도록 크기가 조정되었습니다.
7. 전체 데이터셋의 크기는 5GB입니다.

```
# 추출된 데이터 세트 폴더 확인
os.listdir('food-101/')
```

```
['images', 'license_agreement.txt', 'README.txt', 'meta']
```

이미지 폴더에는 각각 1000개의 이미지가 있는 101개의 폴더가 있습니다. 각 폴더에는 특정 식품 등급의 이미지가 포함되어 있습니다.

```
os.listdir('/content/food-101/images')
```

```
['deviled_eggs',
 'mussels',
 'grilled_salmon',
 'prime_rib',
 'dumplings',
 'spaghetti_carbonara',
 'bruschetta',
 'creme_brulee',
 'ceviche',
 'red_velvet_cake',
 'risotto',
 'strawberry_shortcake',
 'bibimbap',
 'caprese_salad',
 'churros',
 'club_sandwich',
 'peking_duck',
 'frozen_yogurt',
 'guacamole',
 'caesar_salad',
 'gyoza',
 'sashimi',
 'tacos',
```

```
        'chicken_wings',
        'falafel',
        'french_toast',
        'foie_gras',
        'fish_and_chips',
        'oysters',
        'lobster_bisque',
        'samosa',
        'hamburger',
        'beef_carpaccio',
        'spaghetti_bolognese',
        'beef_tartare',
        'beignets',
        'cup_cakes',
        'donuts',
        'baby_back_ribs',
        'baklava',
        'clam_chowder',
        'lobster_roll_sandwich',
        'breakfast_burrito',
        'paella',
        'ramen',
        'omelette',
        'crab_cakes',
        'croque_madame',
        'beet_salad',
        'hot_dog',
        'french_fries',
        'chicken_quesadilla',
        'fried_rice',
        'lasagna',
        'pancakes',
        'onion_rings',
        'seaweed_salad',
        'sushi'
```

meta 폴더에는 텍스트 파일(train.txt 및 test.txt)이 포함되어 있습니다.

train.txt는 트레이닝 세트에 속하는 이미지 목록을 포함합니다.

test.txt는 테스트 세트에 속하는 이미지 목록을 포함합니다.

classes.txt에는 모든 종류의 식품 목록이 포함되어 있습니다.

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

```
Mounted at /content/drive
```

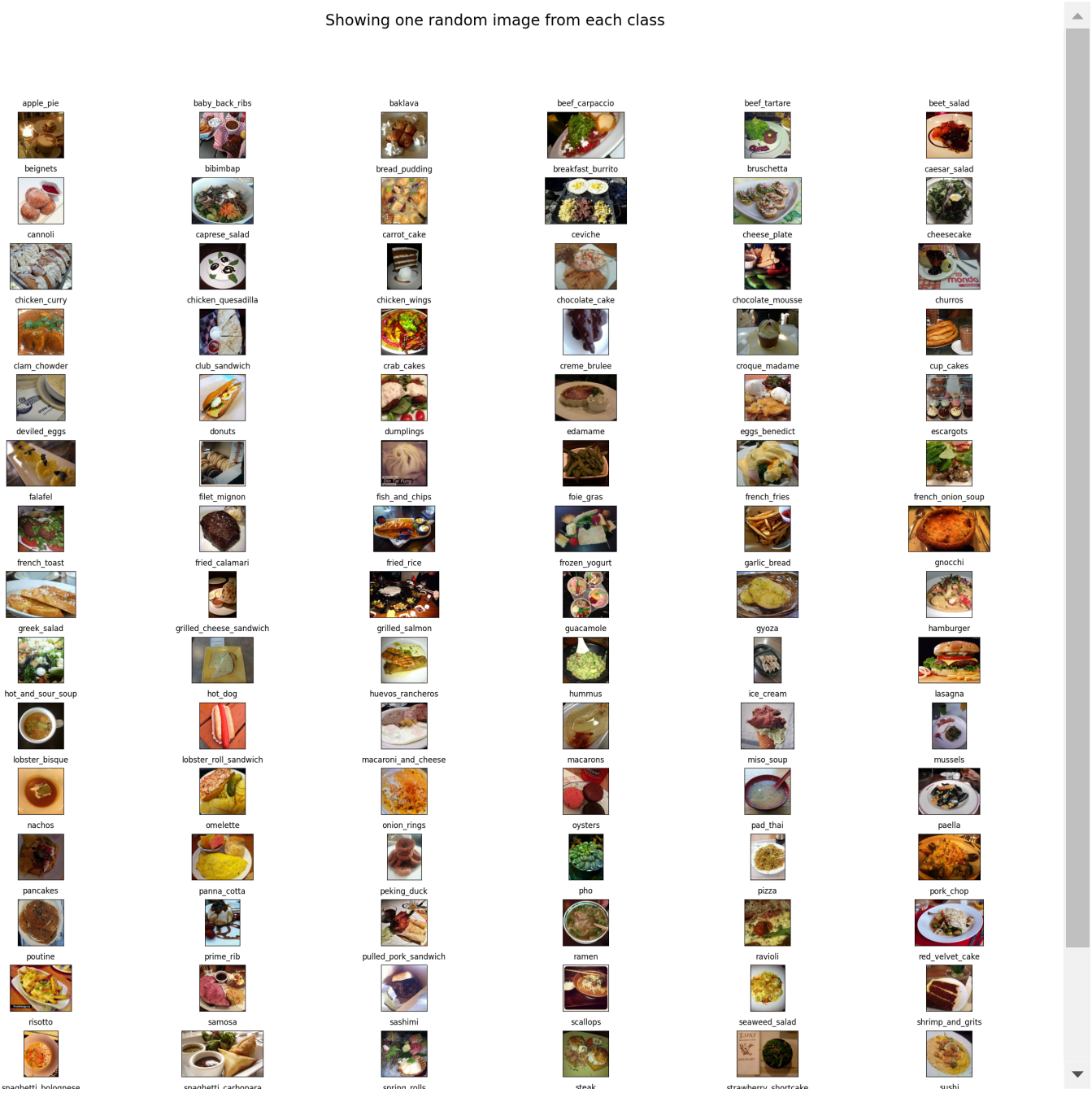
```
os.listdir('/content/food-101/meta')
```

```
['test.json',
 'classes.txt',
 'test.txt',
 'labels.txt',
 'train.txt',
 'train.json']
```

101개 클래스 각각의 임의 이미지 시각화

```
# Visualize the data, showing one image per class from 101 classes
rows = 17
cols = 6
fig, ax = plt.subplots(rows, cols, figsize=(25,25))
fig.suptitle("Showing one random image from each class", y=1.05, fontsize=24) # Adding  y=1.05, fontsize=24 helped me fix the suptitle overlapping witl
data_dir = "/content/food-101/images/"
foods_sorted = sorted(os.listdir(data_dir))
food_id = 0
for i in range(rows):
  for j in range(cols):
    try:
      food_selected = foods_sorted[food_id]
      food_id += 1
    except:
      break
    food_selected_images = os.listdir(os.path.join(data_dir,food_selected)) # returns the list of all files present in each food category
    food_selected_random = np.random.choice(food_selected_images) # picks one food item from the list as choice, takes a list and returns one random i
    img = plt.imread(os.path.join(data_dir,food_selected, food_selected_random))
    ax[i][j].imshow(img)
```

```
        ax[i][j].set_title(food_selected, pad = 10)

    plt.setp(ax, xticks=[],yticks=[])
    plt.tight_layout()
    # https://matplotlib.org/users/tight_layout_guide.html
```



train.txt 및 test.txt를 사용하여 이미지 데이터를 훈련 및 테스트로 분할

```
# Helper method to split dataset into train and test folders
def prepare_data(filepath, src,dest):
  classes_images = defaultdict(list)
  with open(filepath, 'r') as txt:
      paths = [read.strip() for read in txt.readlines()]
      for p in paths:
        food = p.split('/')
        classes_images[food[0]].append(food[1] + '.jpg')

  for food in classes_images.keys():
    print("₩nCopying images into ",food)
    if not os.path.exists(os.path.join(dest,food)):
      os.makedirs(os.path.join(dest,food))
    for i in classes_images[food]:
      copy(os.path.join(src,food,i), os.path.join(dest,food,i))
  print("Copying Done!")


# train.txt 파일을 사용하여 food-101/image에서 food-101/train으로 이미지를 복사하여 train 데이터 세트를 준비합니다.
print("Creating train data...")
prepare_data('/content/food-101/meta/train.txt', '/content/food-101/images', '/content/food-101/train')
```

```
Copying images into  bread_pudding

Copying images into  breakfast_burrito

Copying images into  bruschetta

Copying images into  caesar_salad

Copying images into  cannoli

Copying images into  caprese_salad

Copying images into  carrot_cake

Copying images into  ceviche

Copying images into  cheesecake

Copying images into  cheese_plate

Copying images into  chicken_curry

Copying images into  chicken_quesadilla

Copying images into  chicken_wings

Copying images into  chocolate_cake

Copying images into  chocolate_mousse

Copying images into  churros

Copying images into  clam_chowder

Copying images into  club_sandwich

Copying images into  crab_cakes

Copying images into  creme_brulee

Copying images into  croque_madame

Copying images into  cup_cakes

Copying images into  deviled_eggs

Copying images into  donuts

Copying images into  dumplings

Copying images into  edamame

Copying images into  eggs_benedict

Copying images into  escargots

Copying images into  falafel
```

```
# test.txt 파일을 사용하여 food-101/image에서 food-101/test로 이미지를 복사하여 테스트 데이터를 준비합니다.
print("Creating test data...")
prepare_data('/content/food-101/meta/test.txt', '/content/food-101/images', '/content/food-101/test')
```

```
Copying images into  cup_cakes
Copying images into  deviled_eggs
Copying images into  donuts
Copying images into  dumplings
Copying images into  edamame
Copying images into  eggs_benedict
Copying images into  escargots
Copying images into  falafel
Copying images into  filet_mignon
Copying images into  fish_and_chips
Copying images into  foie_gras
Copying images into  french_fries
Copying images into  french_onion_soup
Copying images into  french_toast
Copying images into  fried_calamari
Copying images into  fried_rice
Copying images into  frozen_yogurt
Copying images into  garlic_bread
Copying images into  gnocchi
Copying images into  greek_salad
Copying images into  grilled_cheese_sandwich
Copying images into  grilled_salmon
Copying images into  guacamole
Copying images into  gyoza
Copying images into  hamburger
Copying images into  hot_and_sour_soup
Copying images into  hot_dog
Copying images into  huevos_rancheros
Copying images into  hummus
Copying images into  ice_cream
```

```
# train 폴더에 몇 개의 파일이 있는지 확인

train_files = sum([len(files) for i, j, files in os.walk("/content/food-101/train")])
print("Total number of samples in train folder")
print(train_files)
```

```
Total number of samples in train folder
75750
```

```
# 테스트 폴더에 몇 개의 파일이 있는지 확인
test_files = sum([len(files) for i, j, files in os.walk("/content/food-101/test")])
print("Total number of samples in test folder")
print(test_files)
```

```
Total number of samples in test folder
25250
```

소수의 클래스로 데이터 하위 집합 만들기

지금은 훈련 데이터와 테스트 데이터가 준비되었지만, 101개의 클래스를 모두 다루기는 많은 시간과 계산량이 필요. 따라서, 3개의 클래스로 데이터셋을 제한하여 train_mini, test_mini를 만들어 진행.

```
# 101가지 음식 목록(a,b,c순서)
foods_sorted
```

```
      'fried_calamari',
      'fried_rice',
      'frozen_yogurt',
      'garlic_bread',
      'gnocchi',
      'greek_salad',
      'grilled_cheese_sandwich',
      'grilled_salmon',
      'guacamole',
      'gyoza',
      'hamburger',
      'hot_and_sour_soup',
      'hot_dog',
      'huevos_rancheros',
      'hummus',
      'ice_cream',
      'lasagna',
      'lobster_bisque',
      'lobster_roll_sandwich',
      'macaroni_and_cheese',
      'macarons',
      'miso_soup',
      'mussels',
      'nachos',
      'omelette',
      'onion_rings',
      'oysters',
      'pad_thai',
      'paella',
      'pancakes',
      'panna_cotta',
      'peking_duck',
      'pho',
      'pizza',
      'pork_chop',
      'poutine',
      'prime_rib',
      'pulled_pork_sandwich',
      'ramen',
      'ravioli',
      'red_velvet_cake',
      'risotto',
      'samosa',
      'sashimi',
      'scallops',
      'seaweed_salad',
      'shrimp_and_grits',
      'spaghetti_bolognese',
      'spaghetti_carbonara',
      'spring_rolls',
      'steak',
      'strawberry_shortcake',
      'sushi',
      'tacos',
      'takoyaki',
      'tiramisu',
      'tuna_tartare',
      'waffles']
```

```
# "train_mini"와 "test_mini" 데이터 샘플을 생성하는 도우미 메서드를 만드는 것이 필요합니다.
def dataset_mini(food_list, src, dest):
  if os.path.exists(dest):
    rmtree(dest) # dataset_mini(이미 존재하는 경우) 폴더를 제거하여 원하는 클래스만 갖도록 합니다.
  os.makedirs(dest)
  for food_item in food_list :
    print("Copying images into",food_item)
    copytree(os.path.join(src,food_item), os.path.join(dest,food_item))


# 3개의 식품 항목을 선택하고 동일한 항목에 대해 별도의 데이터 폴더 생성
food_list = ['samosa','pizza','omelette']
src_train = '/content/food-101/train'
dest_train = '/content/food-101/train_mini'
src_test = '/content/food-101/test'
dest_test = '/content/food-101/test_mini'


print("Creating train data folder with new classes")
dataset_mini(food_list, src_train, dest_train)
```

```
    Creating train data folder with new classes
    Copying images into samosa
    Copying images into pizza
    Copying images into omelette
```

```
print("Total number of samples in train folder")
train_files = sum([len(files) for i, j, files in os.walk("/content/food-101/train_mini")])
print(train_files)
```

```
    Total number of samples in train folder
    2250
```

```
print("Creating test data folder with new classes")
dataset_mini(food_list, src_test, dest_test)
```

```
    Creating test data folder with new classes
    Copying images into samosa
    Copying images into pizza
    Copying images into omelette
```

```
print("Total number of samples in test folder")
test_files = sum([len(files) for i, j, files in os.walk("/content/food-101/test_mini")])
print(test_files)
```

```
    Total number of samples in test folder
    750
```

"Food 101" 데이터셋을 사용하여 사전 훈련된 Inception 모델을 파인튜닝

많은 딥러닝 라이브러리들(Keras 등)은 미리 훈련된(pretrained) 모델들을 제공합니다.

이러한 모델들은 VGG, Inception, ResNet 등과 같은 효율적인 아키텍처의 심층 신경망으로, 이미지넷(ImageNet)과 같은 대규모 데이터셋에서 사전에 훈련되어 있습니다.

사전 훈련된 모델을 사용하면, 이미 학습된 가중치를 활용하고 이를 기반으로 새로운 데이터에 대해 몇 개의 레이어를 추가하여 모델을 파인튜닝할 수 있습니다.

이를 통해 기존의 학습된 지식을 활용하여 새로운 데이터셋에 더 빠르게 수렴하고, 처음부터 모델을 훈련하는 것보다 시간과 계산량을 절약할 수 있습니다. 이러한 방법은 파인튜닝을 통해 원래의 모델에 새로운 데이터를 적용할 수 있게 해줍니다. 이렇게 파인튜닝된 모델은 원래의 모델보다 더 높은 성능을 발휘할 수 있습니다.

- 현재 3개의 클래스(사모사, 피자, 오믈렛)가 있는 데이터 세트의 하위 집합이 있습니다.
- 아래 코드를 사용하여 Inceptionv3 사전 훈련된 모델을 미세 조정합니다.

```
def train_model(n_classes,num_epochs, nb_train_samples,nb_validation_samples):
  K.clear_session()

  img_width, img_height = 299, 299
  train_data_dir = '/content/food-101/train_mini'
  validation_data_dir = '/content/food-101/test_mini'
  batch_size = 16
  bestmodel_path = 'bestmodel_'+str(n_classes)+'class.hdf5'
  trainedmodel_path = 'trainedmodel_'+str(n_classes)+'class.hdf5'
  history_path = 'history_'+str(n_classes)+'.log'

  train_datagen = ImageDataGenerator(
      preprocessing_function=preprocess_input,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True)

  test_datagen = ImageDataGenerator(preprocessing_function=preprocess_input)

  train_generator = train_datagen.flow_from_directory(
      train_data_dir,
      target_size=(img_height, img_width),
      batch_size=batch_size,
      class_mode='categorical')

  validation_generator = test_datagen.flow_from_directory(
      validation_data_dir,
      target_size=(img_height, img_width),
      batch_size=batch_size,
      class_mode='categorical')
```

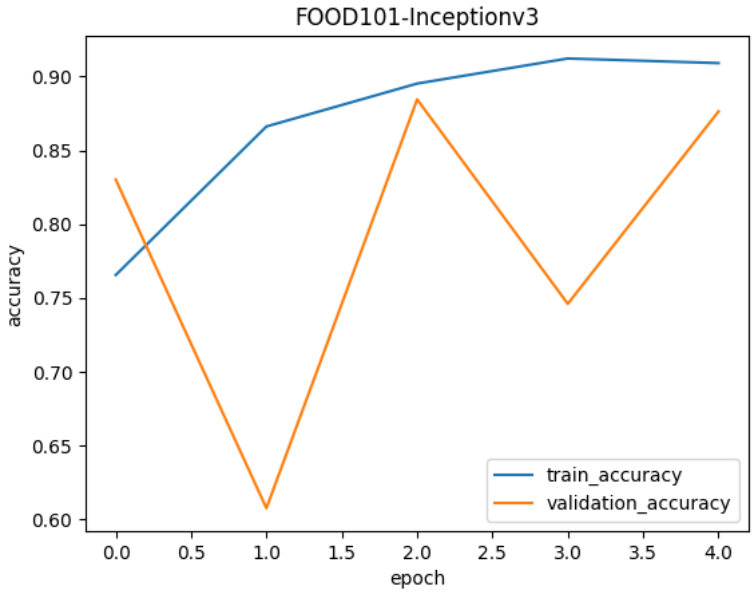
```
  inception = InceptionV3(weights='imagenet', include_top=False)
  x = inception.output
  x = GlobalAveragePooling2D()(x)
  x = Dense(128,activation='relu')(x)
  x = Dropout(0.2)(x)

  predictions = Dense(n_classes,kernel_regularizer=regularizers.l2(0.005), activation='softmax')(x)

  model = Model(inputs=inception.input, outputs=predictions)
  model.compile(optimizer=SGD(lr=0.0001, momentum=0.9), loss='categorical_crossentropy', metrics=['accuracy'])
  checkpoint = ModelCheckpoint(filepath=bestmodel_path, verbose=1, save_best_only=True)
  csv_logger = CSVLogger(history_path)

  history = model.fit_generator(train_generator,
                    steps_per_epoch = nb_train_samples // batch_size,
                    validation_data=validation_generator,
                    validation_steps=nb_validation_samples // batch_size,
                    epochs=num_epochs,
                    verbose=1,
                    callbacks=[csv_logger, checkpoint])

  model.save(trainedmodel_path)
  class_map = train_generator.class_indices
  return history, class_map


# 3개 클래스의 데이터로 모델 훈련
n_classes = 3
epochs = 5
nb_train_samples = train_files
nb_validation_samples = test_files

history, class_map_3 = train_model(n_classes,epochs, nb_train_samples,nb_validation_samples)
print(class_map_3)
```

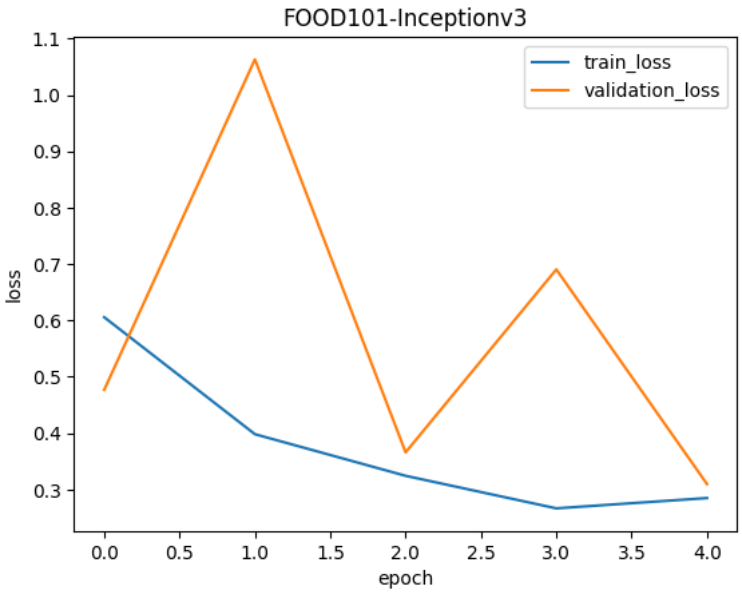
```
Found 2250 images belonging to 3 classes.
Found 750 images belonging to 3 classes.
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels_
87910968/87910968 [==============================] - 0s 0us/step
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.SGD.
<ipython-input-24-b7530807f663>:46: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.
  history = model.fit_generator(train_generator,
Epoch 1/5
140/140 [==============================] - ETA: 0s - loss: 0.6056 - accuracy: 0.7654
Epoch 1: val_loss improved from inf to 0.47675, saving model to bestmodel_3class.hdf5
140/140 [==============================] - 113s 563ms/step - loss: 0.6056 - accuracy: 0.7654 - val_loss: 0.4768 - val_accuracy: 0.8302
Epoch 2/5
140/140 [==============================] - ETA: 0s - loss: 0.3981 - accuracy: 0.8662
Epoch 2: val_loss did not improve from 0.47675
140/140 [==============================] - 74s 530ms/step - loss: 0.3981 - accuracy: 0.8662 - val_loss: 1.0631 - val_accuracy: 0.6073
Epoch 3/5
140/140 [==============================] - ETA: 0s - loss: 0.3243 - accuracy: 0.8953
Epoch 3: val_loss improved from 0.47675 to 0.36589, saving model to bestmodel_3class.hdf5
140/140 [==============================] - 71s 503ms/step - loss: 0.3243 - accuracy: 0.8953 - val_loss: 0.3659 - val_accuracy: 0.8845
Epoch 4/5
140/140 [==============================] - ETA: 0s - loss: 0.2665 - accuracy: 0.9123
Epoch 4: val_loss did not improve from 0.36589
140/140 [==============================] - 71s 507ms/step - loss: 0.2665 - accuracy: 0.9123 - val_loss: 0.6904 - val_accuracy: 0.7459
Epoch 5/5
140/140 [==============================] - ETA: 0s - loss: 0.2848 - accuracy: 0.9091
Epoch 5: val_loss improved from 0.36589 to 0.30958, saving model to bestmodel_3class.hdf5
140/140 [==============================] - 71s 508ms/step - loss: 0.2848 - accuracy: 0.9091 - val_loss: 0.3096 - val_accuracy: 0.8764
{'omelette': 0, 'pizza': 1, 'samosa': 2}
```

정확도 및 손실 플롯 시각화

```
def plot_accuracy(history,title):
    plt.title(title)
    plt.plot(history.history['accuracy']) # change acc to accuracy if testing TF 2.0
    plt.plot(history.history['val_accuracy']) # change val_accuracy if testing TF 2.0
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train_accuracy', 'validation_accuracy'], loc='best')
    plt.show()


def plot_loss(history,title):
    plt.title(title)
    plt.plot(history.history['loss'])
```

```
    plt.plot(history.history['val_loss'])
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train_loss', 'validation_loss'], loc='best')
    plt.show()


plot_accuracy(history,'FOOD101-Inceptionv3')
plot_loss(history,'FOOD101-Inceptionv3')
```





- 플롯은 모델의 정확도가 에포크에 따라 증가하고 손실이 감소했음을 보여줍니다.
- 검증 정확도는 많은 에포크에서 훈련 정확도보다 더 높았습니다.
- 다양한 클래스의 데이터가 포함된 ImageNet에서 훈련된 사전 훈련된 모델을 사용했습니다.
- 드롭아웃을 사용하면 검증 정확도가 높아질 수 있습니다.
- 저장된 최고의 모델은 93%의 Top-1 검증 정확도를 가집니다.

가장 잘 훈련된 모델을 사용하여 인터넷에서 새 이미지에 대한 클래스 예측

```
%%time
# 예측을 위해 가장 잘 저장된 모델 로드

K.clear_session()
model_best = load_model('bestmodel_3class.hdf5',compile = False)
```

```
    CPU times: user 3.03 s, sys: 112 ms, total: 3.14 s
    Wall time: 3.15 s
```

- **compile=False로 설정하고 세션을 지우면 저장된 모델의 로드 속도가 빨라집니다**
- **위의 추가 기능이 없으면 모델 로딩이 1분 이상 걸립니다**

```
def predict_class(model, images, show = True):
  for img in images:
    img = image.load_img(img, target_size=(299, 299))
    img = image.img_to_array(img)
    img = np.expand_dims(img, axis=0)
    img = preprocess_input(img)

    pred = model.predict(img)
    index = np.argmax(pred)
    food_list.sort()
    pred_value = food_list[index]
    #print(pred)
    if show:
        plt.imshow(img[0])
        plt.axis('off')
        plt.title(pred_value)
        plt.show()


# 이미지 목록을 만들고 학습된 모델을 테스트합니다.
images = []
imagepath = '/content/food-101/train/'
images.append(imagepath+'samosa.jpg')
images.append(imagepath+'pizza.jpg')
images.append(imagepath+'omelette.jpg')
predict_class(model_best, images, True)
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-29-687c8b45d3e0> in <cell line: 7>()
      5 images.append(imagepath+'pizza.jpg')
      6 images.append(imagepath+'omelette.jpg')
----> 7 predict_class(model_best, images, True)

                                 1 frames
/usr/local/lib/python3.10/dist-packages/keras/utils/image_utils.py in load_img(path, grayscale, color_mode, target_size, interpolation,
keep_aspect_ratio)
    420         if isinstance(path, pathlib.Path):
    421             path = str(path.resolve())
--> 422         with open(path, "rb") as f:
    423             img = pil_image.open(io.BytesIO(f.read()))
    424     else:

FileNotFoundError: [Errno 2] No such file or directory: '/content/food-101/train/samosa.jpg'

 STACK OVERFLOW 검색
```

해결하였었는데, 실수로 run을 클릭하여 사라진 데이터들때문에 에러가 떴습니다. 밑 과정에서는 정상적으로 진행 됩니다.

11개의 데이터 클래스로 Inceptionv3 모델 미세 조정

3개 클래스로 훈련된 모델이 새로운 데이터를 사용하여 모든 3개의 테스트 이미지의 클래스를 올바르게 예측했습니다.

FOOD-101 데이터셋은 101개의 클래스로 구성되어 있습니다. 101개 클래스의 데이터를 사용하는 경우, 미리 훈련된 모델을 파인튜닝하여 각 에포크가 한 시간 이상 소요됩니다.

하지만 101개 클래스를 포함하여 모델이 어떻게 수행되는지 확인하기 위해, 이와 같은 모델을 사용하여 11개의 무작위 클래스를 선택하여 파인튜닝하고 훈련하고 있습니다.

```
# n개의 무작위 음식 클래스를 선택하는 함수입니다.
def pick_n_random_classes(n):
  random.seed(9000)
  food_list = []
  random_food_indices = random.sample(range(len(foods_sorted)),n) # n개의 무작위 음식 등급을 선택하고 있습니다.
  for i in random_food_indices:
    food_list.append(foods_sorted[i])
  food_list.sort()
  print("These are the randomly picked food classes we will be training the model on...₩n", food_list)
  return food_list
```

```
# 11개 클래스로 시도. 또한 이번에는 음식 클래스를 무작위로 선택.
n = 11
food_list = pick_n_random_classes(11)
```

```
These are the randomly picked food classes we will be training the model on...
 ['beef_tartare', 'chicken_curry', 'chocolate_mousse', 'french_toast', 'fried_rice', 'hot_dog', 'ice_cream', 'lasagna', 'oysters', 'pizza', 'tak
```

```
# 클래스의 새 데이터 하위 집합 만들기
print("Creating training data folder with new classes...")
dataset_mini(food_list, src_train, dest_train)
```

```
Creating training data folder with new classes...
Copying images into beef_tartare
Copying images into chicken_curry
Copying images into chocolate_mousse
Copying images into french_toast
Copying images into fried_rice
Copying images into hot_dog
Copying images into ice_cream
Copying images into lasagna
Copying images into oysters
Copying images into pizza
Copying images into takoyaki
```

```
print("Total number of samples in train folder")
train_files = sum([len(files) for i, j, files in os.walk("/content/food-101/train_mini")])
print(train_files)
```

```
Total number of samples in train folder
8250
```

```
print("Creating test data folder with new classes")
dataset_mini(food_list, src_test, dest_test)
```

```
Creating test data folder with new classes
Copying images into beef_tartare
Copying images into chicken_curry
Copying images into chocolate_mousse
Copying images into french_toast
Copying images into fried_rice
Copying images into hot_dog
Copying images into ice_cream
Copying images into lasagna
Copying images into oysters
Copying images into pizza
Copying images into takoyaki
```

```
print("Total number of samples in test folder")
test_files = sum([len(files) for i, j, files in os.walk("/content/food-101/test_mini")])
print(test_files)
```

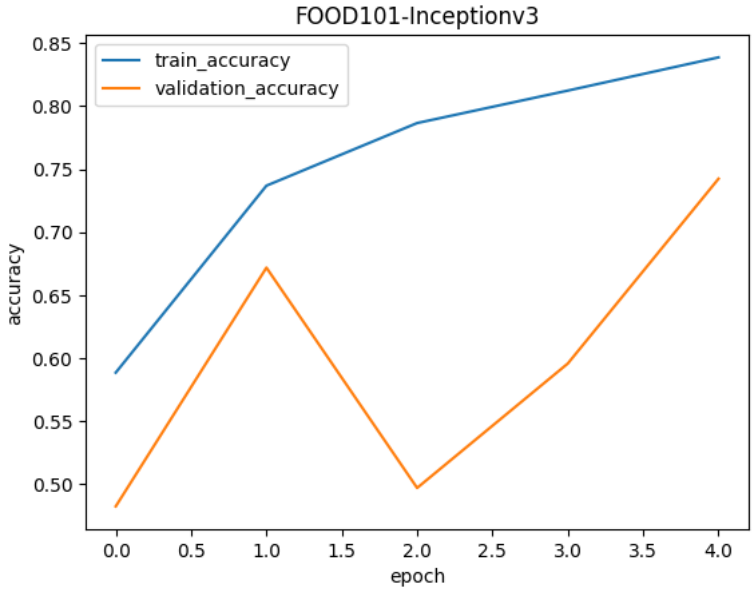
```
Total number of samples in test folder
2750
```

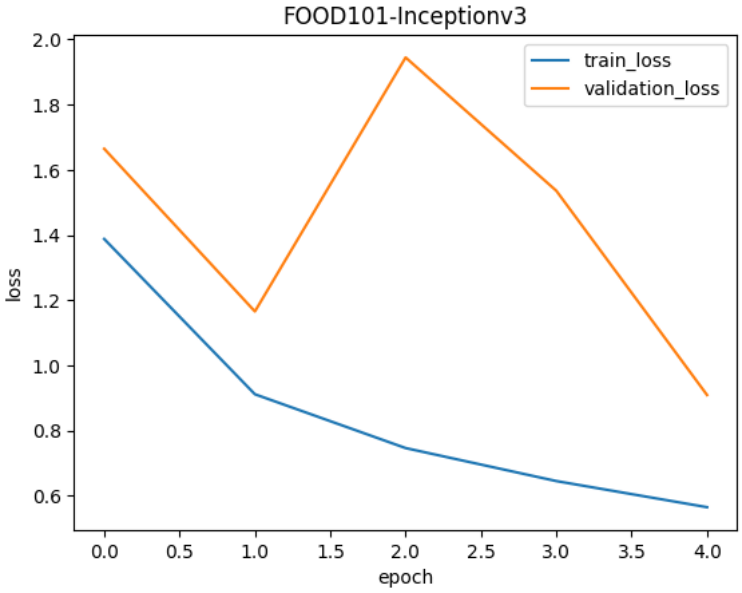
```
# 11개 클래스의 데이터로 모델 훈련
n_classes = 11
epochs = 5
nb_train_samples = train_files
nb_validation_samples = test_files

history, class_map_11 = train_model(n_classes,epochs, nb_train_samples,nb_validation_samples)
print(class_map_11)
```

```
Found 8250 images belonging to 11 classes.
Found 2750 images belonging to 11 classes.
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.SGD.
<ipython-input-24-b7530807f663>:46: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.
  history = model.fit_generator(train_generator,
Epoch 1/5
515/515 [==============================] - ETA: 0s - loss: 1.3879 - accuracy: 0.5885
Epoch 1: val_loss improved from inf to 1.66465, saving model to bestmodel_11class.hdf5
515/515 [==============================] - 274s 498ms/step - loss: 1.3879 - accuracy: 0.5885 - val_loss: 1.6646 - val_accuracy: 0.4825
Epoch 2/5
515/515 [==============================] - ETA: 0s - loss: 0.9117 - accuracy: 0.7368
Epoch 2: val_loss improved from 1.66465 to 1.16570, saving model to bestmodel_11class.hdf5
515/515 [==============================] - 254s 492ms/step - loss: 0.9117 - accuracy: 0.7368 - val_loss: 1.1657 - val_accuracy: 0.6718
Epoch 3/5
515/515 [==============================] - ETA: 0s - loss: 0.7466 - accuracy: 0.7865
Epoch 3: val_loss did not improve from 1.16570
515/515 [==============================] - 253s 490ms/step - loss: 0.7466 - accuracy: 0.7865 - val_loss: 1.9447 - val_accuracy: 0.4971
Epoch 4/5
515/515 [==============================] - ETA: 0s - loss: 0.6455 - accuracy: 0.8122
```

```
Epoch 4: val_loss did not improve from 1.16570
515/515 [==============================] - 255s 494ms/step - loss: 0.6455 - accuracy: 0.8122 - val_loss: 1.5359 - val_accuracy: 0.5958
Epoch 5/5
515/515 [==============================] - ETA: 0s - loss: 0.5653 - accuracy: 0.8386
Epoch 5: val_loss improved from 1.16570 to 0.90948, saving model to bestmodel_11class.hdf5
515/515 [==============================] - 254s 493ms/step - loss: 0.5653 - accuracy: 0.8386 - val_loss: 0.9095 - val_accuracy: 0.7423
{'beef_tartare': 0, 'chicken_curry': 1, 'chocolate_mousse': 2, 'french_toast': 3, 'fried_rice': 4, 'hot_dog': 5, 'ice_cream': 6, 'lasagna': 7, '
```

```
plot_accuracy(history,'FOOD101-Inceptionv3')
plot_loss(history,'FOOD101-Inceptionv3')
```





그래프에서 모델의 정확도가 에포크(epoch)마다 증가하고 손실(loss)이 감소한 것을 확인할 수 있습니다. 많은 에포크에서 검증 정확도(validation accuracy)가 훈련 정확도(training accuracy)보다 높았습니다. ImageNet 데이터셋에서 사전 훈련된 모델을 사용했기 때문에 다양한 클래스의 데이터를 가지고 있을 수 있습니다. 드롭아웃(dropout)을 사용하는 것은 더 높은 검증 정확도를 이끌어낼 수 있습니다. 각 에포크가 약 6분씩 소요되므로, 에포크 수를 10으로 설정했습니다. 손실이 아직 감소하고 있으므로 모델은 더 많은 에포크로 훈련될 수 있습니다. 정확도 향상을 위해 에포크 수를 늘려 볼 수 있었지만, GPU사양 이슈와, 시간관계상 5회로 진행하였습니다.

```
%%time
# 예측을 위해 가장 정확도가 높은 모델 로드

K.clear_session()
model_best = load_model('bestmodel_11class.hdf5',compile = False)
```

```
CPU times: user 3.07 s, sys: 85.6 ms, total: 3.16 s
Wall time: 3.18 s
```

```
# 다운로드한 이미지 목록을 만들고 훈련된 모델을 테스트합니다.
images = []
```

```
images.append(imagepath+'fried_rice/1008935.jpg')
images.append(imagepath+'hot_dog/1000288.jpg')
images.append(imagepath+'ice_cream/1050427.jpg')
images.append(imagepath+'pizza/1008104.jpg')
predict_class(model_best, images, True)
```