In [1]:

```
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
```

```
Bad key "text.kerning_factor" on line 4 in
C:\Users\Dikesh\Anaconda3\envs\tensorflow\lib\site-packages\matplotlib\mpl-
data\stylelib\_classic_test_patch.mplstyle.
You probably need to get an updated matplotlibrc file from
https://github.com/matplotlib/matplotlib/blob/v3.1.3/matplotlibrc.template
or from the matplotlib source distribution
```

In [2]:

```
# Clone tensorflow/examples repo which has images to evaluate trained model
!git clone https://github.com/tensorflow/examples.git
```

```
fatal: destination path 'examples' already exists and is not an empty directory.
```

In [3]:

```
# Helper function to download data and extract

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!")
```

In [4]:

```
# Download data and extract it to folder
get_data_extract()
```

```
Dataset already exists
```

In [5]:

```
# Check the extracted dataset folder
os.listdir('food-101/')
```

Out[5]:

```
['.DS_Store',
 'images',
 'license_agreement.txt',
 'meta',
 'README.txt',
 'test',
 'test_mini',
 'train',
 'train_mini']
```

images folder contains 101 folders with 1000 images each
Each folder contains images of a specific food class

Visualize random image from each of the 101 classes

In [6]:

```
import matplotlib.pyplot as plt
import matplotlib.image as img
%matplotlib inline
import numpy as np
from collections import defaultdict
import collections
import os
```

In [7]:

```
# 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 with axes issue
data_dir = "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
    if food_selected == '.DS_Store':
        continue
    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 li
st as choice, takes a list and returns one random item
    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
```

Showing one random image from each class

Split the image data into train and test using train.txt and test.txt

In [8]:

```
# 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)):
```

```
    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!")
```

In [9]:

```
# Prepare train dataset by copying images from food-101/images to food-101/train using the file tr
ain.txt
print("Creating train data...")
prepare_data('food-101/meta/train.txt', 'food-101/images', 'food-101/train')
```

```
Creating train data...

Copying images into  apple_pie

Copying images into  baby_back_ribs

Copying images into  baklava

Copying images into  beef_carpaccio

Copying images into  beef_tartare

Copying images into  beet_salad

Copying images into  beignets

Copying images into  bibimbap

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
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
Copying images into  lasagna
Copying images into  lobster_bisque
Copying images into  lobster_roll_sandwich
Copying images into  macaroni_and_cheese
Copying images into  macarons
Copying images into  miso_soup
Copying images into  mussels
Copying images into  nachos
Copying images into  omelette
Copying images into  onion_rings
```

```
Copying images into  oysters

Copying images into  pad_thai

Copying images into  paella

Copying images into  pancakes

Copying images into  panna_cotta

Copying images into  peking_duck

Copying images into  pho

Copying images into  pizza

Copying images into  pork_chop

Copying images into  poutine

Copying images into  prime_rib

Copying images into  pulled_pork_sandwich

Copying images into  ramen

Copying images into  ravioli

Copying images into  red_velvet_cake

Copying images into  risotto

Copying images into  samosa

Copying images into  sashimi

Copying images into  scallops

Copying images into  seaweed_salad

Copying images into  shrimp_and_grits

Copying images into  spaghetti_bolognese

Copying images into  spaghetti_carbonara

Copying images into  spring_rolls

Copying images into  steak

Copying images into  strawberry_shortcake

Copying images into  sushi

Copying images into  tacos

Copying images into  takoyaki

Copying images into  tiramisu

Copying images into  tuna_tartare

Copying images into  waffles
Copying Done!
```

In [10]:

```
# Prepare test data by copying images from food-101/images to food-101/test using the file
test.txt
print("Creating test data...")
prepare_data('food-101/meta/test.txt', 'food-101/images', 'food-101/test')
```

```
Creating test data...

Copying images into  apple_pie
```

```
Copying images into  baby_back_ribs
Copying images into  baklava
Copying images into  beef_carpaccio
Copying images into  beef_tartare
Copying images into  beet_salad
Copying images into  beignets
Copying images into  bibimbap
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
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
Copying images into  lasagna
Copying images into  lobster_bisque
Copying images into  lobster_roll_sandwich
Copying images into  macaroni_and_cheese
Copying images into  macarons
Copying images into  miso_soup
Copying images into  mussels
Copying images into  nachos
Copying images into  omelette
Copying images into  onion_rings
Copying images into  oysters
Copying images into  pad_thai
Copying images into  paella
Copying images into  pancakes
Copying images into  panna_cotta
Copying images into  peking_duck
Copying images into  pho
Copying images into  pizza
Copying images into  pork_chop
```

```
Copying images into  poutine

Copying images into  prime_rib

Copying images into  pulled_pork_sandwich

Copying images into  ramen

Copying images into  ravioli

Copying images into  red_velvet_cake

Copying images into  risotto

Copying images into  samosa

Copying images into  sashimi

Copying images into  scallops

Copying images into  seaweed_salad

Copying images into  shrimp_and_grits

Copying images into  spaghetti_bolognese

Copying images into  spaghetti_carbonara

Copying images into  spring_rolls

Copying images into  steak

Copying images into  strawberry_shortcake

Copying images into  sushi

Copying images into  tacos

Copying images into  takoyaki

Copying images into  tiramisu

Copying images into  tuna_tartare

Copying images into  waffles
Copying Done!
```

In [11]:

```
# Check how many files are in the train folder

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

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

In [12]:

```
# Check how many files are in the test folder
test_files = sum([len(files) for i, j, files in os.walk("food-101/test")])
print("Total number of samples in test folder")
print(test_files)
```

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

In [13]:

```
# List of all 101 types of foods(sorted alphabetically)
foods_sorted
```

```
foods_sorted
```

Out[13]:

```
['apple_pie',
 'baby_back_ribs',
 'baklava',
 'beef_carpaccio',
 'beef_tartare',
 'beet_salad',
 'beignets',
 'bibimbap',
 'bread_pudding',
 'breakfast_burrito',
 'bruschetta',
 'caesar_salad',
 'cannoli',
 'caprese_salad',
 'carrot_cake',
 'ceviche',
 'cheese_plate',
 'cheesecake',
 'chicken_curry',
 'chicken_quesadilla',
 'chicken_wings',
 'chocolate_cake',
 'chocolate_mousse',
 'churros',
 'clam_chowder',
 'club_sandwich',
 'crab_cakes',
 'creme_brulee',
 'croque_madame',
 'cup_cakes',
 'deviled_eggs',
 'donuts',
 'dumplings',
 'edamame',
 'eggs_benedict',
 'escargots',
 'falafel',
 'filet_mignon',
 'fish_and_chips',
 'foie_gras',
 'french_fries',
 'french_onion_soup',
 'french_toast',
 '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']
```

In [14]:

```
# Helper method to create train_mini and test_mini data samples
def dataset_mini(food_list, src, dest):
  if os.path.exists(dest):
    rmtree(dest) # removing dataset_mini(if it already exists) folders so that we will have only
the classes that we want
  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))
```

Fine tune Inception Pretrained model using Food 101 dataset

In [15]:

```
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 = 'food-101/train_mini'
  validation_data_dir = '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
```

Visualize the accuracy and loss plots

In [16]:

```
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()
```

In [19]:

```
# Lets try with more classes than just 3. Also, this time lets randomly pick the food classes
n = 101
food_list = sorted(os.listdir("food-101/images/"))
#pick_n_random_classes(11)
src_train = 'food-101/train'
dest_train = 'food-101/train_mini'
src_test = 'food-101/test'
dest_test = 'food-101/test_mini'
```

In [20]:

```
# Create the new data subset of n classes
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 apple pie
```

```
Copying images into baby_back_ribs
Copying images into baklava
Copying images into beef_carpaccio
Copying images into beef_tartare
Copying images into beet_salad
Copying images into beignets
Copying images into bibimbap
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 cheese_plate
Copying images into cheesecake
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
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
Copying images into lasagna
Copying images into lobster_bisque
Copying images into lobster_roll_sandwich
Copying images into macaroni_and_cheese
Copying images into macarons
Copying images into miso_soup
Copying images into mussels
Copying images into nachos
Copying images into omelette
Copying images into onion_rings
Copying images into oysters
Copying images into pad_thai
Copying images into paella
Copying images into pancakes
Copying images into panna_cotta
Copying images into peking_duck
Copying images into pho
Copying images into pizza
Copying images into pork_chop
```

```
Copying images into poutine
Copying images into prime_rib
Copying images into pulled_pork_sandwich
Copying images into ramen
Copying images into ravioli
Copying images into red_velvet_cake
Copying images into risotto
Copying images into samosa
Copying images into sashimi
Copying images into scallops
Copying images into seaweed_salad
Copying images into shrimp_and_grits
Copying images into spaghetti_bolognese
Copying images into spaghetti_carbonara
Copying images into spring_rolls
Copying images into steak
Copying images into strawberry_shortcake
Copying images into sushi
Copying images into tacos
Copying images into takoyaki
Copying images into tiramisu
Copying images into tuna_tartare
Copying images into waffles
```

In [21]:

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

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

In [22]:

```
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 apple_pie
Copying images into baby_back_ribs
Copying images into baklava
Copying images into beef_carpaccio
Copying images into beef_tartare
Copying images into beet_salad
Copying images into beignets
Copying images into bibimbap
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 cheese_plate
Copying images into cheesecake
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 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
Copying images into lasagna
Copying images into lobster_bisque
Copying images into lobster_roll_sandwich
Copying images into macaroni_and_cheese
Copying images into macarons
Copying images into miso_soup
Copying images into mussels
Copying images into nachos
Copying images into omelette
Copying images into onion_rings
Copying images into oysters
Copying images into pad_thai
Copying images into paella
Copying images into pancakes
Copying images into panna_cotta
Copying images into peking_duck
Copying images into pho
Copying images into pizza
Copying images into pork_chop
Copying images into poutine
Copying images into prime_rib
Copying images into pulled_pork_sandwich
Copying images into ramen
Copying images into ravioli
Copying images into red_velvet_cake
Copying images into risotto
Copying images into samosa
Copying images into sashimi
Copying images into scallops
Copying images into seaweed_salad
Copying images into shrimp_and_grits
Copying images into spaghetti_bolognese
Copying images into spaghetti_carbonara
Copying images into spring_rolls
Copying images into steak
Copying images into strawberry_shortcake
Copying images into sushi
Copying images into tacos
Copying images into takoyaki
Copying images into tiramisu
Copying images into tuna_tartare
Copying images into waffles
```

In [23]:

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

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

In [28]:

```
# Train the model with data from 101 classes
n_classes = 101
epochs = 30
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 75750 images belonging to 101 classes.
Found 25250 images belonging to 101 classes.
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
Train for 4734 steps, validate for 1578 steps
Epoch 1/30
4733/4734 [============================>.] - ETA: 0s - loss: 4.8078 - accuracy: 0.0893
Epoch 00001: val_loss improved from inf to 3.81820, saving model to bestmodel_101class.hdf5
4734/4734 [==============================] - 2072s 438ms/step - loss: 4.8078 - accuracy: 0.0893 -
val_loss: 3.8182 - val_accuracy: 0.3259
Epoch 2/30
4733/4734 [============================>.] - ETA: 0s - loss: 3.5474 - accuracy: 0.3261
Epoch 00002: val_loss improved from 3.81820 to 2.43587, saving model to bestmodel_101class.hdf5
4734/4734 [==============================] - 1467s 310ms/step - loss: 3.5473 - accuracy: 0.3261 -
val_loss: 2.4359 - val_accuracy: 0.5527
Epoch 3/30
4733/4734 [============================>.] - ETA: 0s - loss: 2.7557 - accuracy: 0.4637
Epoch 00003: val_loss improved from 2.43587 to 1.89866, saving model to bestmodel_101class.hdf5
4734/4734 [==============================] - 1466s 310ms/step - loss: 2.7557 - accuracy: 0.4637 -
val_loss: 1.8987 - val_accuracy: 0.6509
Epoch 4/30
4733/4734 [============================>.] - ETA: 0s - loss: 2.3334 - accuracy: 0.5475
Epoch 00004: val_loss improved from 1.89866 to 1.61797, saving model to bestmodel_101class.hdf5
4734/4734 [==============================] - 1465s 309ms/step - loss: 2.3335 - accuracy: 0.5474 -
val_loss: 1.6180 - val_accuracy: 0.7028
Epoch 5/30
4733/4734 [============================>.] - ETA: 0s - loss: 2.0618 - accuracy: 0.6026
Epoch 00005: val_loss improved from 1.61797 to 1.47033, saving model to bestmodel_101class.hdf5
4734/4734 [==============================] - 1490s 315ms/step - loss: 2.0618 - accuracy: 0.6026 -
val_loss: 1.4703 - val_accuracy: 0.7309
Epoch 6/30
4733/4734 [============================>.] - ETA: 0s - loss: 1.8649 - accuracy: 0.6434
Epoch 00006: val_loss improved from 1.47033 to 1.35507, saving model to bestmodel_101class.hdf5
4734/4734 [==============================] - 1474s 311ms/step - loss: 1.8648 - accuracy: 0.6434 -
val_loss: 1.3551 - val_accuracy: 0.7554
Epoch 7/30
4733/4734 [============================>.] - ETA: 0s - loss: 1.7133 - accuracy: 0.6748
Epoch 00007: val_loss improved from 1.35507 to 1.26830, saving model to bestmodel_101class.hdf5
4734/4734 [==============================] - 1457s 308ms/step - loss: 1.7133 - accuracy: 0.6747 -
val_loss: 1.2683 - val_accuracy: 0.7691
Epoch 8/30
4733/4734 [============================>.] - ETA: 0s - loss: 1.5762 - accuracy: 0.7026
Epoch 00008: val_loss improved from 1.26830 to 1.21156, saving model to bestmodel_101class.hdf5
4734/4734 [==============================] - 1469s 310ms/step - loss: 1.5761 - accuracy: 0.7026 -
val_loss: 1.2116 - val_accuracy: 0.7774
Epoch 9/30
4733/4734 [============================>.] - ETA: 0s - loss: 1.4680 - accuracy: 0.7236
Epoch 00009: val_loss improved from 1.21156 to 1.16644, saving model to bestmodel_101class.hdf5
4734/4734 [==============================] - 1456s 307ms/step - loss: 1.4679 - accuracy: 0.7236 -
val_loss: 1.1664 - val_accuracy: 0.7848
Epoch 10/30
4733/4734 [============================>.] - ETA: 0s - loss: 1.3698 - accuracy: 0.7442
Epoch 00010: val_loss improved from 1.16644 to 1.11317, saving model to bestmodel_101class.hdf5
4734/4734 [==============================] - 1460s 308ms/step - loss: 1.3697 - accuracy: 0.7443 -
val_loss: 1.1132 - val_accuracy: 0.7956
Epoch 11/30
4733/4734 [============================>.] - ETA: 0s - loss: 1.2788 - accuracy: 0.7636
```

```
Epoch 00011: val_loss improved from 1.11317 to 1.08022, saving model to bestmodel_101class.hdf5
4734/4734 [==============================] - 1463s 309ms/step - loss: 1.2788 - accuracy: 0.7635 -
val_loss: 1.0802 - val_accuracy: 0.7980
Epoch 12/30
4733/4734 [============================>.] - ETA: 0s - loss: 1.1983 - accuracy: 0.7809
Epoch 00012: val_loss improved from 1.08022 to 1.05605, saving model to bestmodel_101class.hdf5
4734/4734 [==============================] - 1463s 309ms/step - loss: 1.1982 - accuracy: 0.7809 -
val_loss: 1.0560 - val_accuracy: 0.8027
Epoch 13/30
4733/4734 [============================>.] - ETA: 0s - loss: 1.1197 - accuracy: 0.7965
Epoch 00013: val_loss improved from 1.05605 to 1.02754, saving model to bestmodel_101class.hdf5
4734/4734 [==============================] - 1457s 308ms/step - loss: 1.1198 - accuracy: 0.7965 -
val_loss: 1.0275 - val_accuracy: 0.8060
Epoch 14/30
4733/4734 [============================>.] - ETA: 0s - loss: 1.0521 - accuracy: 0.8121
Epoch 00014: val_loss improved from 1.02754 to 1.02065, saving model to bestmodel_101class.hdf5
4734/4734 [==============================] - 1471s 311ms/step - loss: 1.0521 - accuracy: 0.8120 -
val_loss: 1.0207 - val_accuracy: 0.8070
Epoch 15/30
4733/4734 [============================>.] - ETA: 0s - loss: 0.9880 - accuracy: 0.8252
Epoch 00015: val_loss improved from 1.02065 to 1.01076, saving model to bestmodel_101class.hdf5
4734/4734 [==============================] - 1455s 307ms/step - loss: 0.9880 - accuracy: 0.8251 -
val_loss: 1.0108 - val_accuracy: 0.8070
Epoch 16/30
4733/4734 [============================>.] - ETA: 0s - loss: 0.9300 - accuracy: 0.8383
Epoch 00016: val_loss improved from 1.01076 to 0.98589, saving model to bestmodel_101class.hdf5
4734/4734 [==============================] - 1456s 308ms/step - loss: 0.9299 - accuracy: 0.8383 -
val_loss: 0.9859 - val_accuracy: 0.8135
Epoch 17/30
4733/4734 [============================>.] - ETA: 0s - loss: 0.8723 - accuracy: 0.8511
Epoch 00017: val_loss improved from 0.98589 to 0.96936, saving model to bestmodel_101class.hdf5
4734/4734 [==============================] - 1493s 315ms/step - loss: 0.8723 - accuracy: 0.8510 -
val_loss: 0.9694 - val_accuracy: 0.8150
Epoch 18/30
4733/4734 [============================>.] - ETA: 0s - loss: 0.8207 - accuracy: 0.8618
Epoch 00018: val_loss did not improve from 0.96936
4734/4734 [==============================] - 1450s 306ms/step - loss: 0.8207 - accuracy: 0.8618 -
val_loss: 0.9706 - val_accuracy: 0.8132
Epoch 19/30
4733/4734 [============================>.] - ETA: 0s - loss: 0.7770 - accuracy: 0.8721
Epoch 00019: val_loss improved from 0.96936 to 0.95823, saving model to bestmodel_101class.hdf5
4734/4734 [==============================] - 1449s 306ms/step - loss: 0.7771 - accuracy: 0.8721 -
val_loss: 0.9582 - val_accuracy: 0.8160
Epoch 20/30
4733/4734 [============================>.] - ETA: 0s - loss: 0.7287 - accuracy: 0.8835
Epoch 00020: val_loss did not improve from 0.95823
4734/4734 [==============================] - 1449s 306ms/step - loss: 0.7287 - accuracy: 0.8835 -
val_loss: 0.9645 - val_accuracy: 0.8142
Epoch 21/30
4733/4734 [============================>.] - ETA: 0s - loss: 0.6874 - accuracy: 0.8921
Epoch 00021: val_loss improved from 0.95823 to 0.95805, saving model to bestmodel_101class.hdf5
4734/4734 [==============================] - 1458s 308ms/step - loss: 0.6874 - accuracy: 0.8921 -
val_loss: 0.9581 - val_accuracy: 0.8157
Epoch 22/30
4733/4734 [============================>.] - ETA: 0s - loss: 0.6477 - accuracy: 0.9012
Epoch 00022: val_loss improved from 0.95805 to 0.95238, saving model to bestmodel_101class.hdf5
4734/4734 [==============================] - 1449s 306ms/step - loss: 0.6476 - accuracy: 0.9012 -
val_loss: 0.9524 - val_accuracy: 0.8161
Epoch 23/30
4733/4734 [============================>.] - ETA: 0s - loss: 0.6132 - accuracy: 0.9091
Epoch 00023: val_loss did not improve from 0.95238
4734/4734 [==============================] - 1451s 306ms/step - loss: 0.6133 - accuracy: 0.9091 -
val_loss: 0.9528 - val_accuracy: 0.8167
Epoch 24/30
4733/4734 [============================>.] - ETA: 0s - loss: 0.5776 - accuracy: 0.9179
Epoch 00024: val_loss improved from 0.95238 to 0.95065, saving model to bestmodel_101class.hdf5
4734/4734 [==============================] - 1451s 307ms/step - loss: 0.5776 - accuracy: 0.9179 -
val_loss: 0.9507 - val_accuracy: 0.8152
Epoch 25/30
4733/4734 [============================>.] - ETA: 0s - loss: 0.5473 - accuracy: 0.9244
Epoch 00025: val_loss improved from 0.95065 to 0.94649, saving model to bestmodel_101class.hdf5
4734/4734 [==============================] - 1456s 308ms/step - loss: 0.5473 - accuracy: 0.9244 -
val_loss: 0.9465 - val_accuracy: 0.8160
Epoch 26/30
4733/4734 [============================>.] - ETA: 0s - loss: 0.5136 - accuracy: 0.9320
Epoch 00026: val_loss did not improve from 0.94649
4734/4734 [==============================] - 1462s 309ms/step - loss: 0.5136 - accuracy: 0.9320 -
```

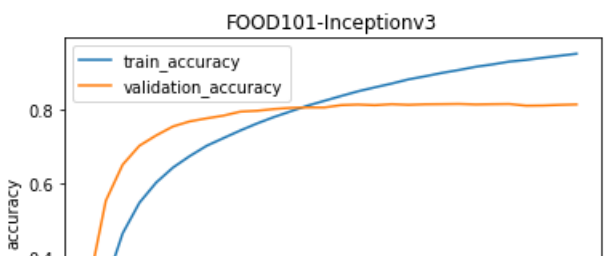
```
val_loss: 0.9468 - val_accuracy: 0.8165
Epoch 27/30
4733/4734 [============================>.] - ETA: 0s - loss: 0.4916 - accuracy: 0.9365
Epoch 00027: val_loss did not improve from 0.94649
4734/4734 [==============================] - 1465s 310ms/step - loss: 0.4916 - accuracy: 0.9365 -
val_loss: 0.9644 - val_accuracy: 0.8119
Epoch 28/30
4733/4734 [============================>.] - ETA: 0s - loss: 0.4656 - accuracy: 0.9425
Epoch 00028: val_loss did not improve from 0.94649
4734/4734 [==============================] - 1553s 328ms/step - loss: 0.4655 - accuracy: 0.9425 -
val_loss: 0.9674 - val_accuracy: 0.8124
Epoch 29/30
4733/4734 [============================>.] - ETA: 0s - loss: 0.4426 - accuracy: 0.9479
Epoch 00029: val_loss did not improve from 0.94649
4734/4734 [==============================] - 1466s 310ms/step - loss: 0.4426 - accuracy: 0.9479 -
val_loss: 0.9602 - val_accuracy: 0.8141
Epoch 30/30
4733/4734 [============================>.] - ETA: 0s - loss: 0.4177 - accuracy: 0.9535
Epoch 00030: val_loss did not improve from 0.94649
4734/4734 [==============================] - 1463s 309ms/step - loss: 0.4177 - accuracy: 0.9535 -
val_loss: 0.9569 - val_accuracy: 0.8153
{'apple_pie': 0, 'baby_back_ribs': 1, 'baklava': 2, 'beef_carpaccio': 3, 'beef_tartare': 4,
'beet_salad': 5, 'beignets': 6, 'bibimbap': 7, 'bread_pudding': 8, 'breakfast_burrito': 9,
'bruschetta': 10, 'caesar_salad': 11, 'cannoli': 12, 'caprese_salad': 13, 'carrot_cake': 14, 'cevi
che': 15, 'cheese_plate': 16, 'cheesecake': 17, 'chicken_curry': 18, 'chicken_quesadilla': 19, 'ch
icken_wings': 20, 'chocolate_cake': 21, 'chocolate_mousse': 22, 'churros': 23, 'clam_chowder': 24,
'club_sandwich': 25, 'crab_cakes': 26, 'creme_brulee': 27, 'croque_madame': 28, 'cup_cakes': 29, '
deviled_eggs': 30, 'donuts': 31, 'dumplings': 32, 'edamame': 33, 'eggs_benedict': 34, 'escargots':
35, 'falafel': 36, 'filet_mignon': 37, 'fish_and_chips': 38, 'foie_gras': 39, 'french_fries': 40,
'french_onion_soup': 41, 'french_toast': 42, 'fried_calamari': 43, 'fried_rice': 44,
'frozen_yogurt': 45, 'garlic_bread': 46, 'gnocchi': 47, 'greek_salad': 48,
'grilled_cheese_sandwich': 49, 'grilled_salmon': 50, 'guacamole': 51, 'gyoza': 52, 'hamburger': 53
, 'hot_and_sour_soup': 54, 'hot_dog': 55, 'huevos_rancheros': 56, 'hummus': 57, 'ice_cream': 58, '
lasagna': 59, 'lobster_bisque': 60, 'lobster_roll_sandwich': 61, 'macaroni_and_cheese': 62, 'macar
ons': 63, 'miso_soup': 64, 'mussels': 65, 'nachos': 66, 'omelette': 67, 'onion_rings': 68, 'oyster
s': 69, 'pad_thai': 70, 'paella': 71, 'pancakes': 72, 'panna_cotta': 73, 'peking_duck': 74, 'pho':
75, 'pizza': 76, 'pork_chop': 77, 'poutine': 78, 'prime_rib': 79, 'pulled_pork_sandwich': 80, 'ram
en': 81, 'ravioli': 82, 'red_velvet_cake': 83, 'risotto': 84, 'samosa': 85, 'sashimi': 86, 'scallo
ps': 87, 'seaweed_salad': 88, 'shrimp_and_grits': 89, 'spaghetti_bolognese': 90,
'spaghetti_carbonara': 91, 'spring_rolls': 92, 'steak': 93, 'strawberry_shortcake': 94, 'sushi': 9
5, 'tacos': 96, 'takoyaki': 97, 'tiramisu': 98, 'tuna_tartare': 99, 'waffles': 100}
```

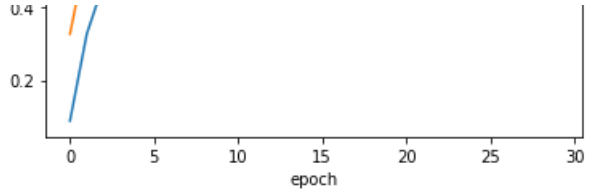
In [29]:

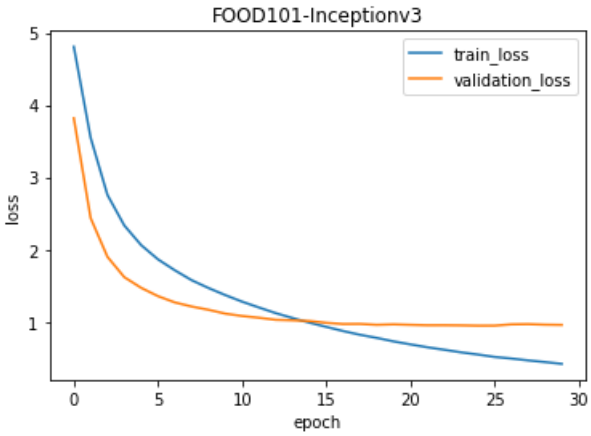
```
def plot_accuracy(history,title):
    plt.title(title)
    plt.plot(history.history['accuracy']) # change acc to accuracy if testing TF 2.0
    plt.plot(history.history['val_accuracy']) # change val_accuracy if testing TF 2.0
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train_accuracy',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







In [30]:

```
%%time
# Loading the best saved model to make predictions

K.clear_session()
model_best = load_model('bestmodel_101class.hdf5',compile = False)
```

```
Wall time: 3.58 s
```

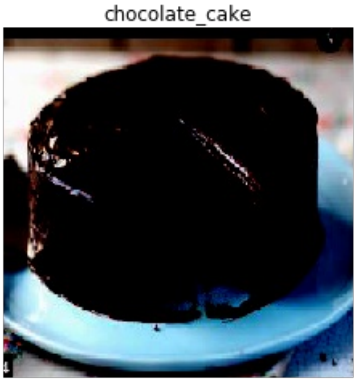
In [31]:

```
# Make a list of downloaded images and test the trained model
images = []
imagepath = 'C:/Users/Dikesh/Desktop/pictures/'
images.append(imagepath+'1.jpg')
images.append(imagepath+'2.jpg')
images.append(imagepath+'3.jpg')
images.append(imagepath+'4.jpg')
images.append(imagepath+'5.jpg')
images.append(imagepath+'6.jpg')
images.append(imagepath+'7.jpg')
images.append(imagepath+'8.jpg')
images.append(imagepath+'9.jpg')
images.append(imagepath+'10.jpg')
images.append(imagepath+'11.jpg')
images.append(imagepath+'12.jpg')
images.append(imagepath+'13.jpg')
images.append(imagepath+'14.jpg')
images.append(imagepath+'15.jpg')
images.append(imagepath+'16.jpg')
images.append(imagepath+'17.jpg')
images.append(imagepath+'18.jpg')
images.append(imagepath+'pizza.jpg')
images.append(imagepath+'dum.jpg')
images.append(imagepath+'samosa.jpg')
images.append(imagepath+'omelette.jpg')
predict_class(model_best, images, True)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```


```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```




```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

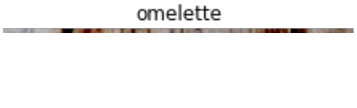

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```


```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```


```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

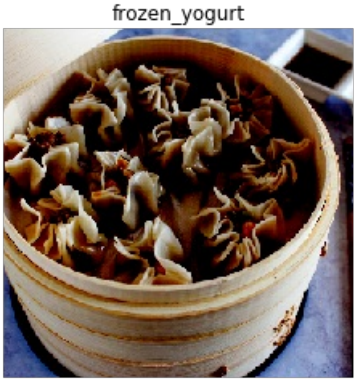

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```



```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for
integers).
```

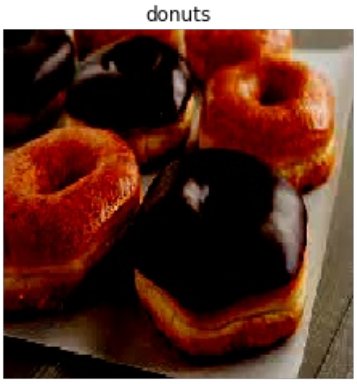

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for
integers).
```


```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for
integers).
```




```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for
integers).
```


```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for
integers).
```




```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for
integers).
```


```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for
integers).
```


```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for
```

```
integers).
```


```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for
integers).
```


```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for
integers).
```


```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for
integers).
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```


```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

In [ ]:

```
```