TASK 5

Develop a model that can accurately recognize food items from images and estimate their calorie content, enabling users to track their dietary intake and make informed food choices.

In [3]:
```
import tensorflow as tf
import matplotlib.image as img
%matplotlib inline
import numpy as np
from collections import defaultdict
import collections
from shutil import copy
from shutil import copytree, rmtree
import tensorflow.keras.backend as K
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import matplotlib.pyplot as plt
import numpy as np
import os
import random
import tensorflow as tf
import tensorflow.keras.backend as K
from tensorflow.keras import regularizers
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatte
from tensorflow.keras.layers import Convolution2D, MaxPooling2D, ZeroP
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
import cv2
```

In [4]:
```
# Helper function to download data and extract
def get_data_extract():
  if "food-101" in os.listdir():
    print("Dataset already exists")
  else:
    print("Downloading the data...")
    !wget http://data.vision.ee.ethz.ch/cvl/food-101.tar.gz
    print("Dataset downloaded!")
    print("Extracting data..")
    !tar xzvf food-101.tar.gz
    print("Extraction done!")
```

In [5]:
```
# Download data and extract it to folder

get_data_extract()
```

```
Dataset already exists
```

In [6]:
```
# Check the extracted dataset folder
!ls food-101/
```

```
README.txt  images  license_agreement.txt  meta
```

In [7]:
```
os.listdir('food-101/images')
```

```
Out[7]: ['macarons',
         'french_toast',
         'lobster_bisque',
         'prime_rib',
         'pork_chop',
         'guacamole',
         'baby_back_ribs',
         'mussels',
         'beef_carpaccio',
         'poutine',
         'hot_and_sour_soup',
         'seaweed_salad',
         'foie_gras',
         'dumplings',
         'peking_duck',
         'takoyaki',
         'bibimbap',
         'falafel',
         'pulled_pork_sandwich',
         'lobster_roll_sandwich',
         'carrot_cake',
         'beet_salad',
         'panna_cotta',
         'donuts',
         'red_velvet_cake',
         'grilled_cheese_sandwich',
         'cannoli',
         'spring_rolls',
         'shrimp_and_grits',
         'clam_chowder',
         'omelette',
         'fried_calamari',
         'caprese_salad',
         'oysters',
         'scallops',
         'ramen',
         'grilled_salmon',
         'croque_madame',
         'filet_mignon',
         'hamburger',
         'spaghetti_carbonara',
         'miso_soup',
         'bread_pudding',
         'lasagna',
         'crab_cakes',
         'cheesecake',
         'spaghetti_bolognese',
         'cup_cakes',
         'creme_brulee',
         'waffles',
         'fish_and_chips',
         'paella',
         'macaroni_and_cheese',
         'chocolate_mousse',
         'ravioli',
         'chicken_curry',
         'caesar_salad',
         'nachos',
         'tiramisu',
         'frozen_yogurt',
         'ice_cream',
         'risotto',
         'club_sandwich',
         'strawberry_shortcake',
         'steak',
         'churros',
         'garlic_bread',
         'baklava',
         'bruschetta',
         'hummus',
         'chicken_wings',
         'greek_salad',
         'tuna_tartare',
         'chocolate_cake',
         'gyoza',
         'eggs_benedict',
         'deviled_eggs',
         'samosa',
         'sushi',
         'breakfast_burrito',
         'ceviche',
         'beef_tartare',
         'apple_pie',
         '.DS_Store',
         'huevos_rancheros',
         'beignets',
         'pizza',
         'edamame',
         'french_onion_soup',
         'hot_dog',
         'tacos',
         'chicken_quesadilla',
         'pho',
         'gnocchi',
         'pancakes',
         'fried_rice',
         'cheese_plate',
         'onion_rings',
         'escargots',
         'sashimi',
         'pad_thai',
         'french_fries']
```

```
In [8]: os.listdir('food-101/meta')
```

```
Out[8]: ['test.txt',
 'train.json',
 'labels.txt',
 'test.json',
 'train.txt',
 'classes.txt']
```

In [9]:
```
!head food-101/meta/train.txt
```

```
apple_pie/1005649
apple_pie/1014775
apple_pie/1026328
apple_pie/1028787
apple_pie/1043283
apple_pie/1050519
apple_pie/1057749
apple_pie/1057810
apple_pie/1072416
apple_pie/1074856
```

In [10]:
```
!head food-101/meta/classes.txt
```

```
apple_pie
baby_back_ribs
baklava
beef_carpaccio
beef_tartare
beet_salad
beignets
bibimbap
bread_pudding
breakfast_burrito
```

In [11]:
```
/isualize the data, showing one image per class from 101 classes
vs = 17
ls = 6
g, ax = plt.subplots(rows, cols, figsize=(25,25))
g.suptitle("Showing one random image from each class", y=1.05, fontsize
ta_dir = "food-101/images/"
ods_sorted = sorted(os.listdir(data_dir))
od_id = 0
r 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_selecte
  food_selected_random = np.random.choice(food_selected_images) # pick
  img = plt.imread(os.path.join(data_dir,food_selected, food_selected_
  ax[i][j].imshow(img)
  ax[i][j].set_title(food_selected, pad = 10)

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

Showing one random image from each class

In [12]:
```python
# 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!")
```

In [13]:
```
# Prepare train dataset by copying images from food-101/images to food
print("Creating train data...")
prepare_data('/food-101/food-101/meta/train.txt', '/food-101/food-101/
```

```
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
```

In [14]:
```
# Prepare test data by copying images from food-101/images to food-101
print("Creating test data...")
prepare_data('food-101/food-101/meta/test.txt', 'food-101/food-101/ima
```

```
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
```

In [15]:
```
# Check how many files are in the train folder
print("Total number of samples in train folder")
!find train -type d -or -type f -printf '.' | wc -c
```

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

In [16]:
```
# Check how many files are in the test folder
print("Total number of samples in test folder")
!find test -type d -or -type f -printf '.' | wc -c
```

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

In [17]:
```
os.chdir('/')
```

In [18]:
```
# List of all 101 types of foods(sorted alphabetically)
del foods_sorted[0] # remove .DS_Store from the list
```

In [19]:
```
foods_sorted
```

```
Out[19]: ['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 [20]:
```
# 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
  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)
```

In [21]:
```
# picking 3 food items and generating separate data folders for the sa
food_list = ['apple_pie','pizza','omelette']
src_train = 'train'
dest_train = 'train_mini/'
src_test = 'test'
dest_test = 'test_mini/'
```

In [22]:
```
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 apple_pie
Copying images into pizza
Copying images into omelette
```

In [23]:
```
print("Total number of samples in train folder")

!find /kaggle/working/train_mini -type d -or -type f -printf '.' | wc
```

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

In [24]:
```
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 pizza
Copying images into omelette
```

In [25]:
```
print("Total number of samples in test folder")
!find /kaggle/working/test_mini -type d -or -type f -printf '.' | wc -
```

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

In [26]:
```
from tensorflow.keras.applications.resnet50 import ResNet50

K.clear_session()
n_classes = 3
img_width, img_height = 224, 224
train_data_dir = 'train_mini'
validation_data_dir = 'test_mini'
nb_train_samples = 2250 #75750
nb_validation_samples = 750 #25250
batch_size = 16

train_datagen = ImageDataGenerator(
    rescale=1. / 255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1. / 255)

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')


resnet50 = ResNet50(weights='imagenet', include_top=False)
x = resnet50.output
x = GlobalAveragePooling2D()(x)
x = Dense(128,activation='relu')(x)
x = Dropout(0.2)(x)

predictions = Dense(3,kernel_regularizer=regularizers.l2(0.005), acti

model = Model(inputs=resnet50.input, outputs=predictions)
model.compile(optimizer=SGD(lr=0.0001, momentum=0.9), loss='categorica
checkpointer = ModelCheckpoint(filepath='best_model_3class.hdf5', verb
csv_logger = CSVLogger('history_3class.log')

history = model.fit_generator(train_generator,
                    steps_per_epoch = nb_train_samples // batch_size,
                    validation_data=validation_generator,
                    validation_steps=nb_validation_samples // batch_s
                    epochs=30,
                    verbose=1,
                    callbacks=[csv_logger, checkpointer])

model.save('model_trained_3class.hdf5')
```

```
acc: 0.8254
Epoch 00005: val_loss improved from 1.80936 to 1.65712, saving model
to /kaggle/working/best_model_3class.hdf5
140/140 [==============================] - 39s 279ms/step - loss: 0.
4758 - acc: 0.8257 - val_loss: 1.6571 - val_acc: 0.4049
Epoch 6/30
139/140 [============================>.] - ETA: 0s - loss: 0.4308 -
acc: 0.8542
Epoch 00006: val_loss improved from 1.65712 to 0.86345, saving model
to /kaggle/working/best_model_3class.hdf5
140/140 [==============================] - 40s 282ms/step - loss: 0.
4302 - acc: 0.8548 - val_loss: 0.8635 - val_acc: 0.6522
Epoch 7/30
139/140 [============================>.] - ETA: 0s - loss: 0.4069
```

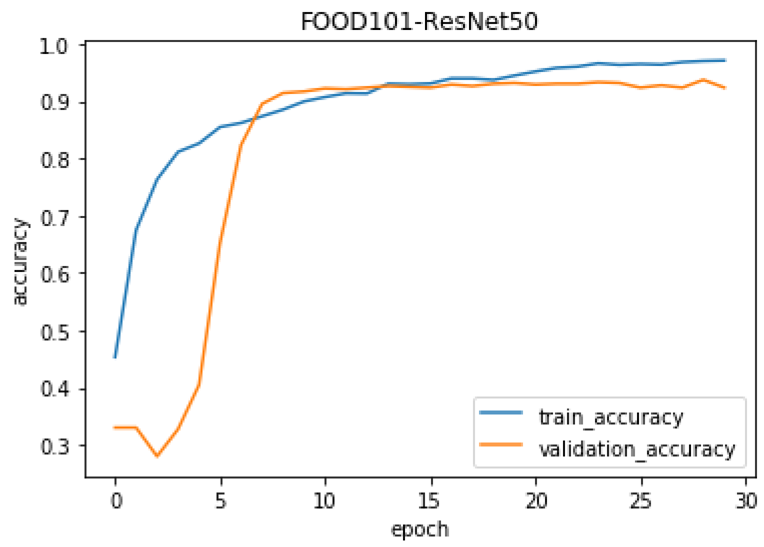
In [27]:
```python
class_map_3 = train_generator.class_indices
class_map_3
```

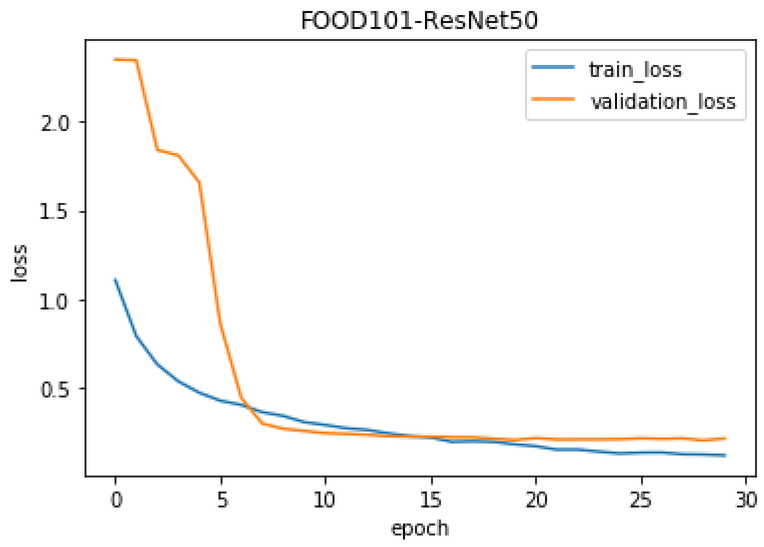
Out[27]:
```
{'apple_pie': 0, 'omelette': 1, 'pizza': 2}
```

Visualize the accuracy and loss plots

In [28]:
```python
def plot_accuracy(history,title):
    plt.title(title)
    plt.plot(history.history['acc'])
    plt.plot(history.history['val_acc'])
    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()
```

In [29]:
```
plot_accuracy(history,'FOOD101-ResNet50')
plot_loss(history,'FOOD101-ResNet50')
```





In [30]:
```
%%time
# Loading the best saved model to make predictions
K.clear_session()
model_best = load_model('/kaggle/working/best_model_3class.hdf5',comp
```

```
CPU times: user 6.64 s, sys: 194 ms, total: 6.84 s
Wall time: 6.79 s
```

In [31]:
```
def predict_class(model, images, show = True):
  for img in images:
    img = image.load_img(img, target_size=(224, 224))
    img = image.img_to_array(img)
    img = np.expand_dims(img, axis=0)
    img /= 255.

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

In [35]:
```
# Make a list of downloaded images and test the trained model
images = []
images.append('applepie.jpg')
images.append('pizza.jpg')
images.append('omelette.jpg')
predict_class(model_best, images, True)
```

apple_pie

pizza

omelette

In [36]:
```
# Helper function to select n random food classes
def pick_n_random_classes(n):
  food_list = []
  random_food_indices = random.sample(range(len(foods_sorted)),n) # We
  for i in random_food_indices:
    food_list.append(foods_sorted[i])
  food_list.sort()
  return food_list
```

In [37]:
```
Lets try with more classes than just 3. Also, this time lets randomly
= 11
od_list = pick_n_random_classes(n)
od_list = ['apple_pie', 'beef_carpaccio', 'bibimbap', 'cup_cakes', 'fo
int("These are the randomly picked food classes we will be training th
```

```
These are the randomly picked food classes we will be training the m
odel on...
 ['apple_pie', 'beef_carpaccio', 'bibimbap', 'cup_cakes', 'foie_gra
s', 'french_fries', 'garlic_bread', 'pizza', 'spring_rolls', 'spaghe
tti_carbonara', 'strawberry_shortcake']
```

In [38]:
```
# 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 beef_carpaccio
Copying images into bibimbap
Copying images into cup_cakes
Copying images into foie_gras
Copying images into french_fries
Copying images into garlic_bread
Copying images into pizza
Copying images into spring_rolls
Copying images into spaghetti_carbonara
Copying images into strawberry_shortcake
```

In [39]:
```
print("Total number of samples in train folder")
!find train_mini/ -type d -or -type f -printf '.' | wc -c
```

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

In [40]:
```
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 beef_carpaccio
Copying images into bibimbap
Copying images into cup_cakes
Copying images into foie_gras
Copying images into french_fries
Copying images into garlic_bread
Copying images into pizza
Copying images into spring_rolls
Copying images into spaghetti_carbonara
Copying images into strawberry_shortcake
```

```
In [41]: print("Total number of samples in test folder")
         !find test_mini/ -type d -or -type f -printf '.' | wc -c
```

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

In [42]:
```
# Let's use a pretrained Inceptionv3 model on subset of data with 11 
K.clear_session()

n_classes = n
img_width, img_height = 224, 224
train_data_dir = 'train_mini'
validation_data_dir = 'test_mini'
nb_train_samples = 8250 #75750
nb_validation_samples = 2750 #25250
batch_size = 16

train_datagen = ImageDataGenerator(
    rescale=1. / 255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1. / 255)

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')


resnet50 = ResNet50(weights='imagenet', include_top=False)
x = resnet50.output
x = GlobalAveragePooling2D()(x)
x = Dense(128,activation='relu')(x)
x = Dropout(0.2)(x)

predictions = Dense(n,kernel_regularizer=regularizers.l2(0.005), acti

model = Model(inputs=resnet50.input, outputs=predictions)
model.compile(optimizer=SGD(lr=0.0001, momentum=0.9), loss='categorica
checkpointer = ModelCheckpoint(filepath='best_model_11class.hdf5', ve
csv_logger = CSVLogger('history_11class.log')

history_11class = model.fit_generator(train_generator,
                    steps_per_epoch = nb_train_samples // batch_size,
                    validation_data=validation_generator,
                    validation_steps=nb_validation_samples // batch_s
                    epochs=30,
                    verbose=1,
                    callbacks=[csv_logger, checkpointer])

model.save('model_trained_11class.hdf5')
```

```
Found 8250 images belonging to 11 classes.
Found 2750 images belonging to 11 classes.
```

```
/opt/conda/lib/python3.6/site-packages/keras_applications/resnet50.p
y:265: UserWarning: The output shape of `ResNet50(include_top=False)
` has been changed since Keras 2.2.0.
  warnings.warn('The output shape of `ResNet50(include_top=False)` '

Epoch 1/30
514/515 [============================>.] - ETA: 0s - loss: 2.0506 -
acc: 0.3578
Epoch 00001: val_loss improved from inf to 2.63834, saving model to
/kaggle/working/best_model_11class.hdf5
515/515 [==============================] - 163s 316ms/step - loss:
2.0494 - acc: 0.3581 - val_loss: 2.6383 - val_acc: 0.1758
Epoch 2/30
```

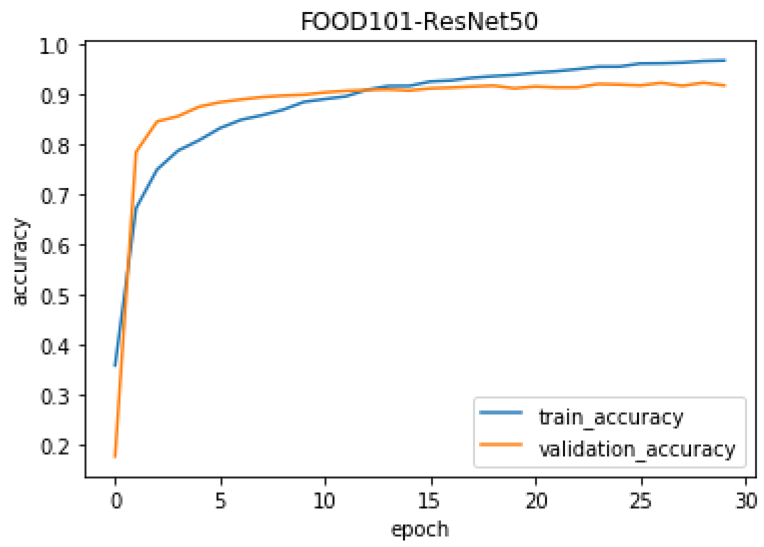
In [43]:

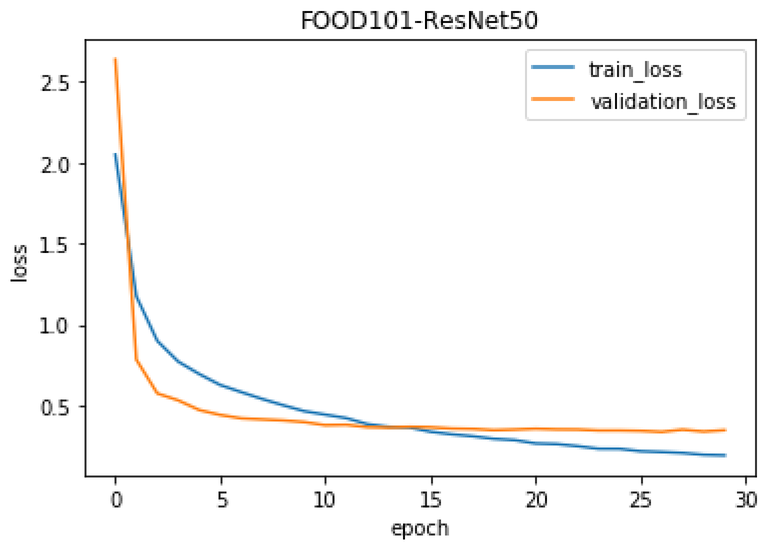
```
class_map_11 = train_generator.class_indices
class_map_11
```

Out[43]:

```
{'apple_pie': 0,
 'beef_carpaccio': 1,
 'bibimbap': 2,
 'cup_cakes': 3,
 'foie_gras': 4,
 'french_fries': 5,
 'garlic_bread': 6,
 'pizza': 7,
 'spaghetti_carbonara': 8,
 'spring_rolls': 9,
 'strawberry_shortcake': 10}
```

In [44]:
```
plot_accuracy(history_11class,'FOOD101-ResNet50')
plot_loss(history_11class,'FOOD101-ResNet50')
```





In [45]:
```
%%time
# Loading the best saved model to make predictions
K.clear_session()
model_best = load_model('/kaggle/working/best_model_11class.hdf5',comp
```

```
CPU times: user 6.88 s, sys: 165 ms, total: 7.05 s
Wall time: 7.05 s
```

In [47]:
```
# Make a list of downloaded images and test the trained model
images = []
images.append('cupcakes.jpg')
# images.append('pizza.jpg')
images.append('springrolls.jpg')
images.append('garlicbread.jpg')
predict_class(model_best, images, True)
```

cup_cakes

spring_rolls

garlic_bread