In [1]:
```
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
from shutil import copy
from collections import defaultdict
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import layers, models
```

```
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for th
is version of SciPy (detected version 1.24.3
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

In [2]:
```
# Function to download and extract the dataset if not already present
def download_dataset():
    kaggle_input_directory = '/kaggle/input/'
    print(os.listdir(kaggle_input_directory))
    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 > /dev/null 2>&1
        print("Extraction done!")
```

In [3]:
```
# Function to resize all images in the dataset
def resize_images(data_path):
    dirs = os.listdir(data_path)
    try:
        for c in dirs:
            ch_path = os.path.join(data_path, c)
            if os.path.isdir(ch_path) and c != '.DS_Store':
                print(f'Resizing {c} images')
                for f in os.listdir(os.path.join(ch_path, "")):
                    img_path = os.path.join(ch_path, f)
                    if f.endswith('.jpg') or f.endswith('.jpeg') or f.endswith('.png'):
                        img = cv2.imread(img_path)
                        imgRescale = cv2.resize(img, (255, 255))
                        output_path_rescaled = os.path.join(ch_path,f'{os.path.splitext(f)[0]}.jpg')
```

```
                        cv2.imwrite(output_path_rescaled, imgRescale)
    except Exception as error:
        print(f"An error has occurred: {error}")
```

In [4]:

```
# Function to prepare train and test data
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 [12]:

```
# Function to build and train the CNN model
def build_and_train_model(train_generator, test_generator, num_classes):
    model = models.Sequential()
    model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(255, 255, 3)))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(128, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Flatten())
    model.add(layers.Dense(256, activation='relu'))
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(num_classes, activation='softmax'))

    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
                  loss=tf.keras.losses.SparseCategoricalCrossentropy(),
                  metrics=['accuracy'])

    history = model.fit(
        train_generator,
        epochs=50,
        verbose=1,
```

```
        validation_data=test_generator,
    )

    return history  # Return the history object for potential further analysis
```

In [14]:
```
# Main function
def main():
    download_dataset()
```

In [15]:
```
    data_path = '/kaggle/input/food-101/food-101/food-101/images'
    train_file_path = '/kaggle/input/food-101/food-101/food-101/meta/train.txt'
    test_file_path = '/kaggle/input/food-101/food-101/food-101/meta/test.txt'
    train_data_dir = 'train'
    test_data_dir = 'test'
```

In [16]:
```
    # Resize images
    resize_images(data_path)
```

```
Resizing macarons images
Resizing french_toast images
Resizing lobster_bisque images
Resizing prime_rib images
Resizing pork_chop images
Resizing guacamole images
Resizing baby_back_ribs images
Resizing mussels images
Resizing beef_carpaccio images
Resizing poutine images
Resizing hot_and_sour_soup images
Resizing seaweed_salad images
Resizing foie_gras images
Resizing dumplings images
Resizing peking_duck images
Resizing takoyaki images
Resizing bibimbap images
Resizing falafel images
Resizing pulled_pork_sandwich images
Resizing lobster_roll_sandwich images
Resizing carrot_cake images
Resizing beet_salad images
Resizing panna_cotta images
Resizing donuts images
Resizing red_velvet_cake images
Resizing grilled_cheese_sandwich images
Resizing cannoli images
```

```
Resizing spring_rolls images
Resizing shrimp_and_grits images
Resizing clam_chowder images
Resizing omelette images
Resizing fried_calamari images
Resizing caprese_salad images
Resizing oysters images
Resizing scallops images
Resizing ramen images
Resizing grilled_salmon images
Resizing croque_madame images
Resizing filet_mignon images
Resizing hamburger images
Resizing spaghetti_carbonara images
Resizing miso_soup images
Resizing bread_pudding images
Resizing lasagna images
Resizing crab_cakes images
Resizing cheesecake images
Resizing spaghetti_bolognese images
Resizing cup_cakes images
Resizing creme_brulee images
Resizing waffles images
Resizing fish_and_chips images
Resizing paella images
Resizing macaroni_and_cheese images
Resizing chocolate_mousse images
Resizing ravioli images
Resizing chicken_curry images
Resizing caesar_salad images
Resizing nachos images
Resizing tiramisu images
Resizing frozen_yogurt images
Resizing ice_cream images
Resizing risotto images
Resizing club_sandwich images
Resizing strawberry_shortcake images
Resizing steak images
Resizing churros images
Resizing garlic_bread images
Resizing baklava images
Resizing bruschetta images
Resizing hummus images
Resizing chicken_wings images
Resizing greek_salad images
Resizing tuna_tartare images
Resizing chocolate_cake images
```

```
Resizing gyoza images
Resizing eggs_benedict images
Resizing deviled_eggs images
Resizing samosa images
Resizing sushi images
Resizing breakfast_burrito images
Resizing ceviche images
Resizing beef_tartare images
Resizing apple_pie images
Resizing huevos_rancheros images
Resizing beignets images
Resizing pizza images
Resizing edamame images
Resizing french_onion_soup images
Resizing hot_dog images
Resizing tacos images
Resizing chicken_quesadilla images
Resizing pho images
Resizing gnocchi images
Resizing pancakes images
Resizing fried_rice images
Resizing cheese_plate images
Resizing onion_rings images
Resizing escargots images
Resizing sashimi images
Resizing pad_thai images
Resizing french_fries images
```

In [22]:

```
# Prepare train and test data
print("Creating train data...")
prepare_data(train_file_path, data_path, train_data_dir)
print("Creating test data...")
prepare_data(test_file_path, data_path, test_data_dir)
```

```
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!
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 [24]:
```
    # Create data generators
train_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(255, 255),
    batch_size=32,
    class_mode='sparse',
    shuffle=True
    )
```

```
Found 75750 images belonging to 101 classes.
```

In [25]:
```
test_datagen = ImageDataGenerator(rescale=1./255)
test_generator = test_datagen.flow_from_directory(
    test_data_dir,
    target_size=(255, 255),
    batch_size=32,
    class_mode='sparse',
    shuffle=False
    )
```

```
Found 25250 images belonging to 101 classes.
```

In [26]:
```
# Build and train the CNN model
history = build_and_train_model(train_generator, test_generator, num_classes=101)
```

```
Epoch 1/50
2368/2368 [==============================] - 287s 119ms/step - loss: 4.6221 - accuracy: 0.0087 - val_loss: 4.6152 - val_accuracy:
0.0099
Epoch 2/50
2368/2368 [==============================] - 277s 117ms/step - loss: 4.6159 - accuracy: 0.0086 - val_loss: 4.6152 - val_accuracy:
0.0101
Epoch 3/50
2368/2368 [==============================] - 279s 118ms/step - loss: 4.6161 - accuracy: 0.0099 - val_loss: 4.6148 - val_accuracy:
0.0110
Epoch 4/50
2368/2368 [==============================] - 290s 122ms/step - loss: 4.5228 - accuracy: 0.0240 - val_loss: 4.4075 - val_accuracy:
0.0374
Epoch 5/50
2368/2368 [==============================] - 280s 118ms/step - loss: 4.3486 - accuracy: 0.0477 - val_loss: 4.1543 - val_accuracy:
0.0761
Epoch 6/50
2368/2368 [==============================] - 277s 117ms/step - loss: 4.0807 - accuracy: 0.0839 - val_loss: 3.9545 - val_accuracy:
0.1034
Epoch 7/50
2368/2368 [==============================] - 272s 115ms/step - loss: 3.8070 - accuracy: 0.1241 - val_loss: 3.8933 - val_accuracy:
0.1116
Epoch 8/50
2368/2368 [==============================] - 276s 117ms/step - loss: 3.4417 - accuracy: 0.1896 - val_loss: 3.9217 - val_accuracy:
0.1088
Epoch 9/50
2368/2368 [==============================] - 273s 115ms/step - loss: 3.0250 - accuracy: 0.2623 - val_loss: 4.0684 - val_accuracy:
0.1052
Epoch 10/50
2368/2368 [==============================] - 274s 116ms/step - loss: 2.6020 - accuracy: 0.3425 - val_loss: 4.2456 - val_accuracy:
0.1071
Epoch 11/50
2368/2368 [==============================] - 276s 116ms/step - loss: 2.1861 - accuracy: 0.4300 - val_loss: 4.5735 - val_accuracy:
0.0953
Epoch 12/50
2368/2368 [==============================] - 271s 114ms/step - loss: 1.8562 - accuracy: 0.5040 - val_loss: 4.9273 - val_accuracy:
0.0977
Epoch 13/50
2368/2368 [==============================] - 267s 113ms/step - loss: 1.5755 - accuracy: 0.5684 - val_loss: 5.4819 - val_accuracy:
0.0957
Epoch 14/50
2368/2368 [==============================] - 275s 116ms/step - loss: 1.3843 - accuracy: 0.6135 - val_loss: 5.6795 - val_accuracy:
0.0911
Epoch 15/50
2368/2368 [==============================] - 267s 113ms/step - loss: 1.2120 - accuracy: 0.6547 - val_loss: 6.0718 - val_accuracy:
0.0936
Epoch 16/50
2368/2368 [==============================] - 273s 115ms/step - loss: 1.0866 - accuracy: 0.6869 - val_loss: 6.3066 - val_accuracy:
```

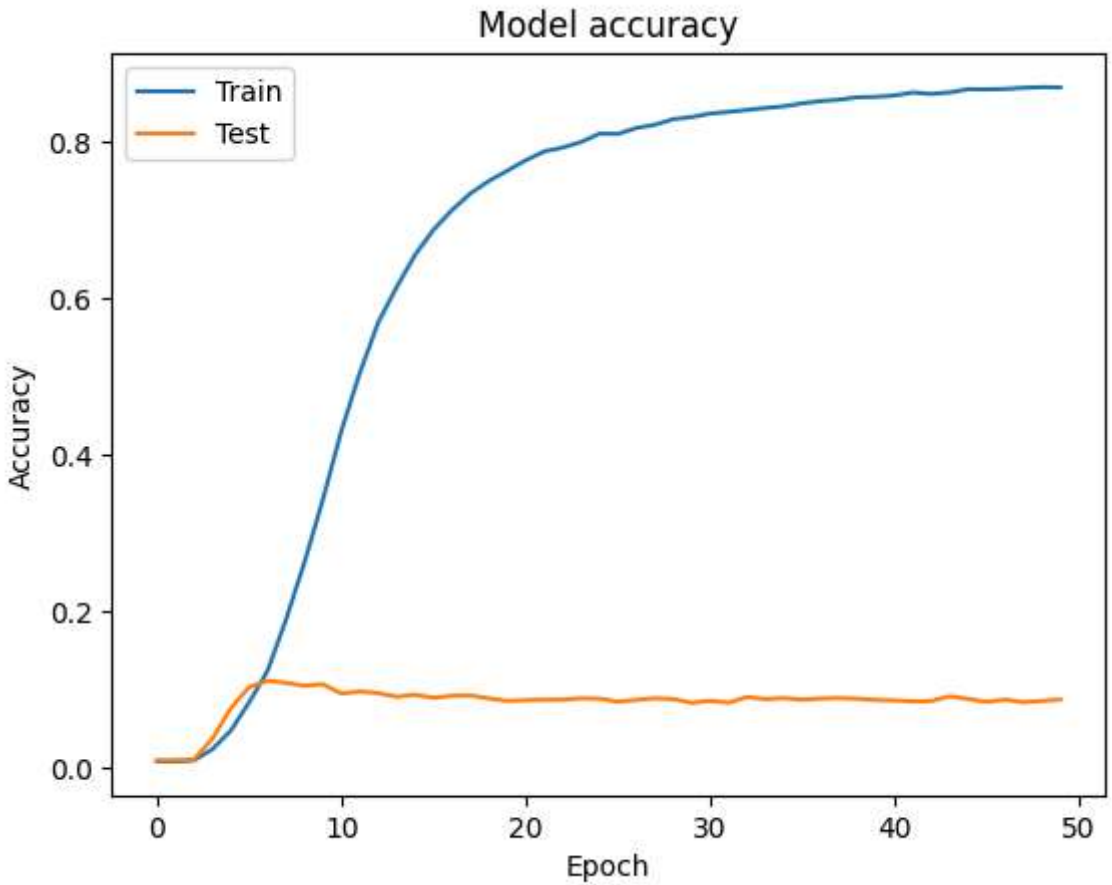
```
0.0898
Epoch 17/50
2368/2368 [==============================] - 267s 113ms/step - loss: 0.9980 - accuracy: 0.7118 - val_loss: 6.9483 - val_accuracy:
0.0923
Epoch 18/50
2368/2368 [==============================] - 273s 115ms/step - loss: 0.9226 - accuracy: 0.7332 - val_loss: 6.9088 - val_accuracy:
0.0926
Epoch 19/50
2368/2368 [==============================] - 267s 113ms/step - loss: 0.8680 - accuracy: 0.7490 - val_loss: 7.0800 - val_accuracy:
0.0891
Epoch 20/50
2368/2368 [==============================] - 270s 114ms/step - loss: 0.8141 - accuracy: 0.7622 - val_loss: 7.2209 - val_accuracy:
0.0857
Epoch 21/50
2368/2368 [==============================] - 260s 110ms/step - loss: 0.7711 - accuracy: 0.7754 - val_loss: 7.3998 - val_accuracy:
0.0867
Epoch 22/50
2368/2368 [==============================] - 271s 114ms/step - loss: 0.7323 - accuracy: 0.7867 - val_loss: 8.1932 - val_accuracy:
0.0872
Epoch 23/50
2368/2368 [==============================] - 281s 119ms/step - loss: 0.7155 - accuracy: 0.7917 - val_loss: 8.1394 - val_accuracy:
0.0874
Epoch 24/50
2368/2368 [==============================] - 277s 117ms/step - loss: 0.6848 - accuracy: 0.7987 - val_loss: 8.7574 - val_accuracy:
0.0891
Epoch 25/50
2368/2368 [==============================] - 268s 113ms/step - loss: 0.6492 - accuracy: 0.8095 - val_loss: 8.4497 - val_accuracy:
0.0887
Epoch 26/50
2368/2368 [==============================] - 271s 115ms/step - loss: 0.6426 - accuracy: 0.8091 - val_loss: 8.4226 - val_accuracy:
0.0847
Epoch 27/50
2368/2368 [==============================] - 269s 113ms/step - loss: 0.6214 - accuracy: 0.8167 - val_loss: 9.1426 - val_accuracy:
0.0870
Epoch 28/50
2368/2368 [==============================] - 270s 114ms/step - loss: 0.6147 - accuracy: 0.8208 - val_loss: 8.7818 - val_accuracy:
0.0891
Epoch 29/50
2368/2368 [==============================] - 282s 119ms/step - loss: 0.5941 - accuracy: 0.8279 - val_loss: 8.8216 - val_accuracy:
0.0881
Epoch 30/50
2368/2368 [==============================] - 269s 114ms/step - loss: 0.5778 - accuracy: 0.8306 - val_loss: 7.9335 - val_accuracy:
0.0834
Epoch 31/50
2368/2368 [==============================] - 266s 112ms/step - loss: 0.5620 - accuracy: 0.8351 - val_loss: 9.1232 - val_accuracy:
0.0859
Epoch 32/50
```

```
2368/2368 [==============================] - 275s 116ms/step - loss: 0.5543 - accuracy: 0.8373 - val_loss: 8.0605 - val_accuracy:
0.0836
Epoch 33/50
2368/2368 [==============================] - 265s 112ms/step - loss: 0.5493 - accuracy: 0.8398 - val_loss: 9.4933 - val_accuracy:
0.0906
Epoch 34/50
2368/2368 [==============================] - 269s 114ms/step - loss: 0.5389 - accuracy: 0.8422 - val_loss: 9.4202 - val_accuracy:
0.0882
Epoch 35/50
2368/2368 [==============================] - 266s 112ms/step - loss: 0.5326 - accuracy: 0.8443 - val_loss: 9.7841 - val_accuracy:
0.0893
Epoch 36/50
2368/2368 [==============================] - 269s 114ms/step - loss: 0.5155 - accuracy: 0.8482 - val_loss: 10.3736 - val_accuracy:
0.0873
Epoch 37/50
2368/2368 [==============================] - 266s 112ms/step - loss: 0.5136 - accuracy: 0.8510 - val_loss: 10.8258 - val_accuracy:
0.0889
Epoch 38/50
2368/2368 [==============================] - 269s 114ms/step - loss: 0.5027 - accuracy: 0.8528 - val_loss: 10.5552 - val_accuracy:
0.0894
Epoch 39/50
2368/2368 [==============================] - 268s 113ms/step - loss: 0.4973 - accuracy: 0.8559 - val_loss: 10.1964 - val_accuracy:
0.0886
Epoch 40/50
2368/2368 [==============================] - 268s 113ms/step - loss: 0.4992 - accuracy: 0.8564 - val_loss: 10.6013 - val_accuracy:
0.0869
Epoch 41/50
2368/2368 [==============================] - 273s 115ms/step - loss: 0.4888 - accuracy: 0.8582 - val_loss: 9.6040 - val_accuracy:
0.0864
Epoch 42/50
2368/2368 [==============================] - 268s 113ms/step - loss: 0.4746 - accuracy: 0.8620 - val_loss: 10.4727 - val_accuracy:
0.0852
Epoch 43/50
2368/2368 [==============================] - 267s 113ms/step - loss: 0.4868 - accuracy: 0.8601 - val_loss: 10.2352 - val_accuracy:
0.0855
Epoch 44/50
2368/2368 [==============================] - 265s 112ms/step - loss: 0.4790 - accuracy: 0.8621 - val_loss: 11.2470 - val_accuracy:
0.0916
Epoch 45/50
2368/2368 [==============================] - 260s 110ms/step - loss: 0.4648 - accuracy: 0.8662 - val_loss: 11.2667 - val_accuracy:
0.0884
Epoch 46/50
2368/2368 [==============================] - 274s 116ms/step - loss: 0.4680 - accuracy: 0.8659 - val_loss: 10.6363 - val_accuracy:
0.0846
Epoch 47/50
2368/2368 [==============================] - 265s 112ms/step - loss: 0.4658 - accuracy: 0.8665 - val_loss: 11.0492 - val_accuracy:
0.0873
```

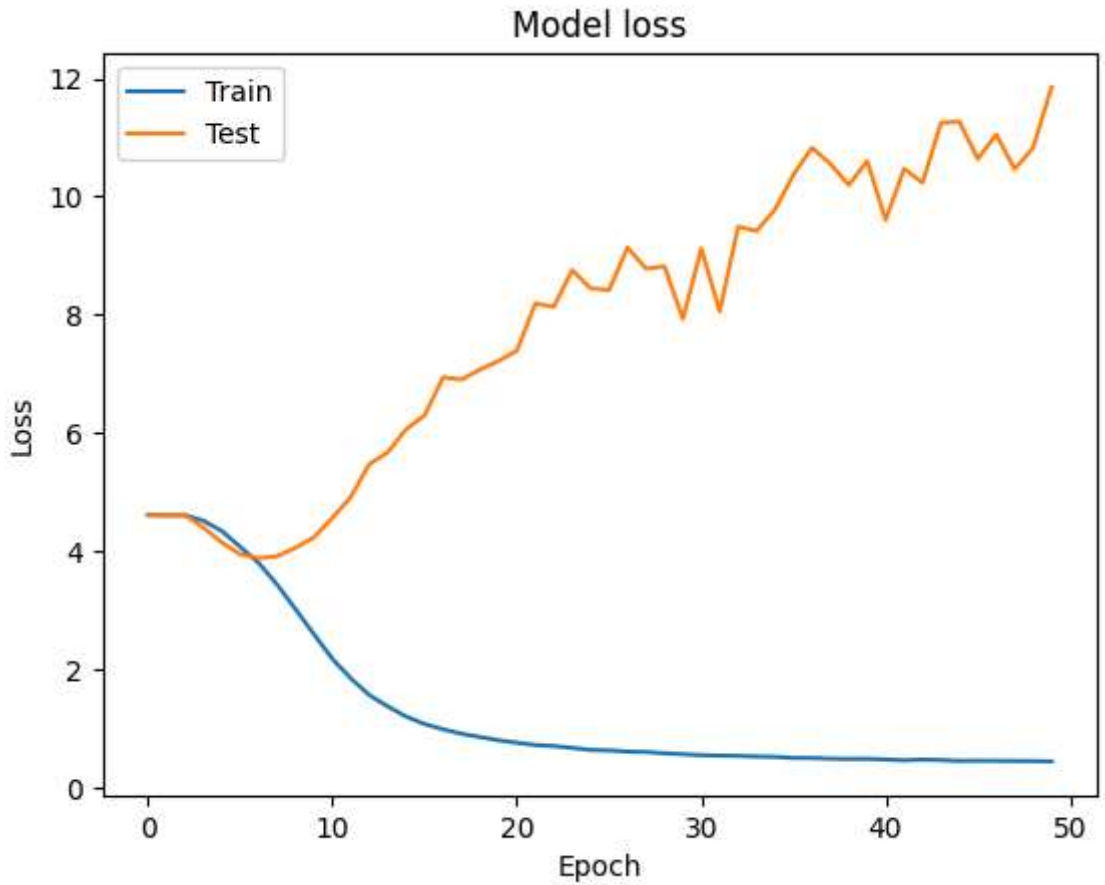
```
Epoch 48/50
2368/2368 [==============================] - 270s 114ms/step - loss: 0.4635 - accuracy: 0.8680 - val_loss: 10.4650 - val_accuracy:
0.0842
Epoch 49/50
2368/2368 [==============================] - 261s 110ms/step - loss: 0.4615 - accuracy: 0.8688 - val_loss: 10.8283 - val_accuracy:
0.0857
Epoch 50/50
2368/2368 [==============================] - 268s 113ms/step - loss: 0.4566 - accuracy: 0.8684 - val_loss: 11.8503 - val_accuracy:
0.0875
```

In [28]:

```
# Plotting training & test accuracy values
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



In [30]:
```
# Plotting training & test loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



In [31]:
```
if __name__ == "__main__":
    main()
```

```
['test-image', 'food-101']
Downloading the data...
--2024-03-16 14:05:39--  http://data.vision.ee.ethz.ch/cvl/food-101.tar.gz
Resolving data.vision.ee.ethz.ch (data.vision.ee.ethz.ch)... 129.132.52.178, 2001:67c:10ec:36c2::178
Connecting to data.vision.ee.ethz.ch (data.vision.ee.ethz.ch)|129.132.52.178|:80... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://data.vision.ee.ethz.ch/cvl/food-101.tar.gz [following]
--2024-03-16 14:05:39--  https://data.vision.ee.ethz.ch/cvl/food-101.tar.gz
Connecting to data.vision.ee.ethz.ch (data.vision.ee.ethz.ch)|129.132.52.178|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 4996278331 (4.7G) [application/x-gzip]
Saving to: 'food-101.tar.gz'
```

```
food-101.tar.gz     100%[===================>]   4.65G  31.9MB/s    in 2m 42s

2024-03-16 14:08:22 (29.5 MB/s) - 'food-101.tar.gz' saved [4996278331/4996278331]

Dataset downloaded!
Extracting data..
Extraction done!
```