In [2]:

```
# This Python 3 environment comes with many helpful analytics libraries inst
# It is defined by the kaggle/python Docker image: https://github.com/kaggle
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will l

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        (os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that
# You can also write temporary files to /kaggle/temp/, but they won't be sav
```

1. Importing Libraries

In [3]:

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import random
import os
import pathlib
from keras.preprocessing import image
from keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D, Flatt
import tensorflow_hub as hub
import pandas as pd
import numpy as np

import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.image as img

import cv2
import itertools
import pathlib
import warnings
import os
import random
import time
import gc
from IPython.display import Markdown, display
from PIL import Image
from random import randint
import warnings
warnings.filterwarnings('ignore')

from imblearn.over_sampling import SMOTE

from sklearn.model_selection import train_test_split
from sklearn.metrics import matthews_corrcoef as MCC
from sklearn.metrics import balanced_accuracy_score as BAS
from sklearn.metrics import classification_report, confusion_matrix, accurac

import keras
from tensorflow import keras
from keras import Sequential
from keras import layers
import tensorflow as tf
import tensorflow_addons as tfa
from tensorflow.keras.preprocessing import image_dataset_from_directory
from keras.utils.vis_utils import plot_model
from tensorflow.keras import Sequential, Input
#from keras.utils import to_categorical
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.layers import Dense, Dropout,SeparableConv2D, Activati
from tensorflow.keras.layers import Conv2D, Flatten
from tensorflow.keras.callbacks import ReduceLROnPlateau,EarlyStopping, Mode
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.preprocessing.image import ImageDataGenerator as IDG
```

```
/opt/conda/lib/python3.10/site-packages/tensorflow_io/python/ops/__init__.p
y:98: UserWarning: unable to load libtensorflow_io_plugins.so: unable to ope
n file: libtensorflow_io_plugins.so, from paths: ['/opt/conda/lib/python3.1
0/site-packages/tensorflow_io/python/ops/libtensorflow_io_plugins.so']
caused by: ['/opt/conda/lib/python3.10/site-packages/tensorflow_io/python/op
s/libtensorflow_io_plugins.so: undefined symbol: _ZN3tsl6StatusC1EN10tensorf
low5error4CodeESt17basic_string_viewIcSt11char_traitsIcEENS_14SourceLocation
E']
  warnings.warn(f"unable to load libtensorflow_io_plugins.so: {e}")
/opt/conda/lib/python3.10/site-packages/tensorflow_io/python/ops/__init__.p
y:104: UserWarning: file system plugins are not loaded: unable to open file:
libtensorflow_io.so, from paths: ['/opt/conda/lib/python3.10/site-packages/t
ensorflow_io/python/ops/libtensorflow_io.so']
caused by: ['/opt/conda/lib/python3.10/site-packages/tensorflow_io/python/op
s/libtensorflow_io.so: undefined symbol: _ZTVN10tensorflow13GcsFileSystemE']
  warnings.warn(f"file system plugins are not loaded: {e}")
```

2. Load Dataset

In [4]:

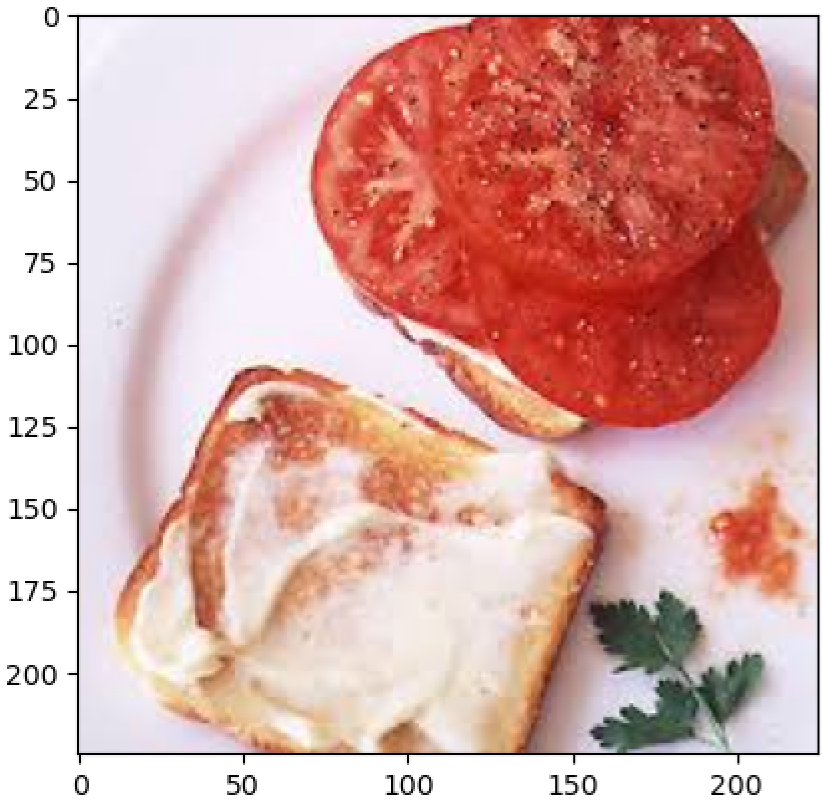
```
data = "/kaggle/input/food-image-classification-dataset/Food Classification
```

In [5]:

```
img = mpimg.imread("/kaggle/input/food-image-classification-dataset/Food Cla
plt.imshow(img)
```

Out[5]:

```
<matplotlib.image.AxesImage at 0x79ae609f2a10>
```



In [6]:

```
# Define parameters values
IMG_SIZE = (256, 256)
VALID_SPLIT = 0.3
BATCH_SIZE = 32
SEED = 42
PATH = "/kaggle/input/food-image-classification-dataset/Food Classification

# Get train image with generator
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    PATH,
    validation_split=VALID_SPLIT,
    subset="training",
    seed=SEED,
    image_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    label_mode="categorical"
)
```

```
Found 23873 files belonging to 34 classes.
Using 16712 files for training.
```

In [7]:

```
# Get validation image with generator
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    PATH,
    validation_split=VALID_SPLIT,
    subset="validation",
    seed=SEED,
    image_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    label_mode="categorical"
)
```

```
Found 23873 files belonging to 34 classes.
Using 7161 files for validation.
```

3. Data Visualisation EDA

In [8]:

```
# Collecte the class names.
path = "/kaggle/input/food-image-classification-dataset/Food Classification
class_names = sorted(os.listdir(path))
n_classes = len(class_names)

# Print
print("No. Classes : {}".format(n_classes))
print("Classes     : {}".format(class_names))
```

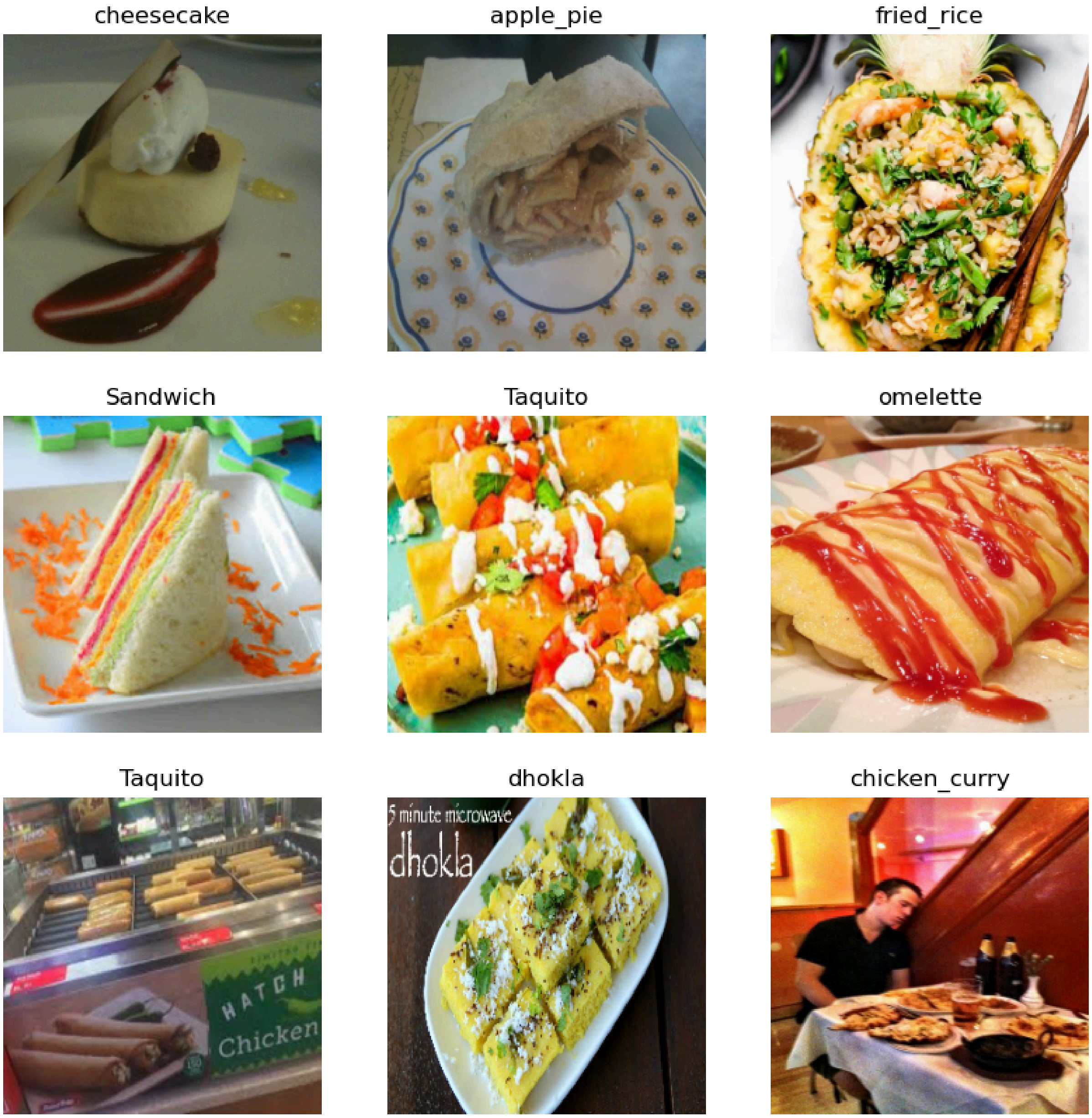
```
No. Classes : 34
Classes     : ['Baked Potato', 'Crispy Chicken', 'Donut', 'Fries', 'Hot Do
g', 'Sandwich', 'Taco', 'Taquito', 'apple_pie', 'burger', 'butter_naan', 'ch
ai', 'chapati', 'cheesecake', 'chicken_curry', 'chole_bhature', 'dal_makhan
i', 'dhokla', 'fried_rice', 'ice_cream', 'idli', 'jalebi', 'kaathi_rolls',
'kadai_paneer', 'kulfi', 'masala_dosa', 'momos', 'omelette', 'paani_puri',
'pakode', 'pav_bhaji', 'pizza', 'samosa', 'sushi']
```

In [9]:

```
# Get class names
classes = train_ds.class_names
```

In [10]:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(classes[np.argmax(labels[i])])
        plt.axis("off")
```



cheesecake apple_pie fried_rice

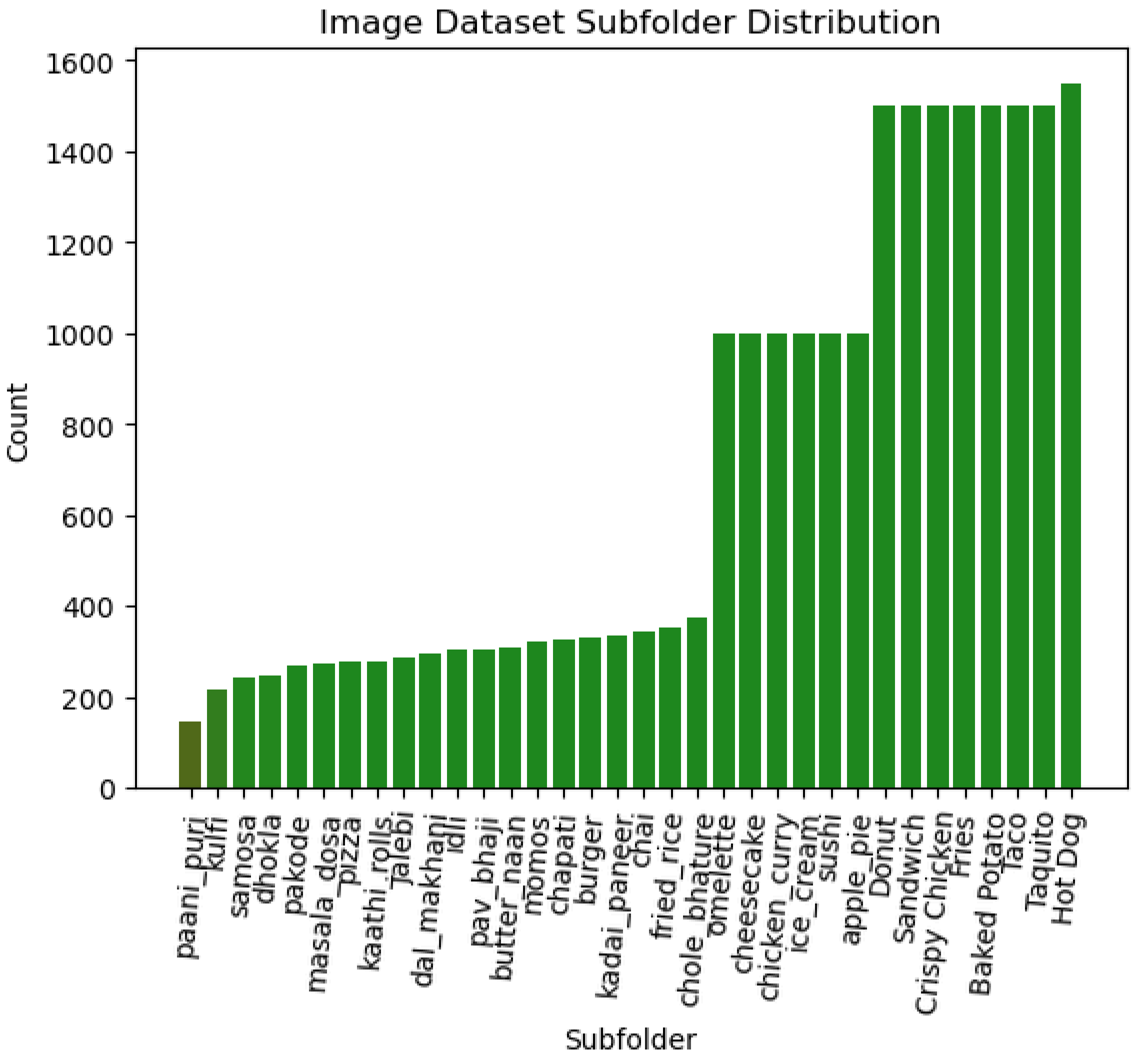
Sandwich Taquito omelette

Taquito dhokla chicken_curry

In [11]:

```python
import os
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors

# Set the path for the dataset directory
dataset_dir = path

# Initialize a dictionary to store the counts of images in each subfolder
subfolder_counts = {}

# Iterate over the subfolders in the dataset directory
for subfolder in os.listdir(dataset_dir):
    subfolder_path = os.path.join(dataset_dir, subfolder)
    if os.path.isdir(subfolder_path):
        # Count the number of image files in each subfolder
        subfolder_counts[subfolder] = len(os.listdir(subfolder_path))

# Sort the subfolder counts in ascending order
```

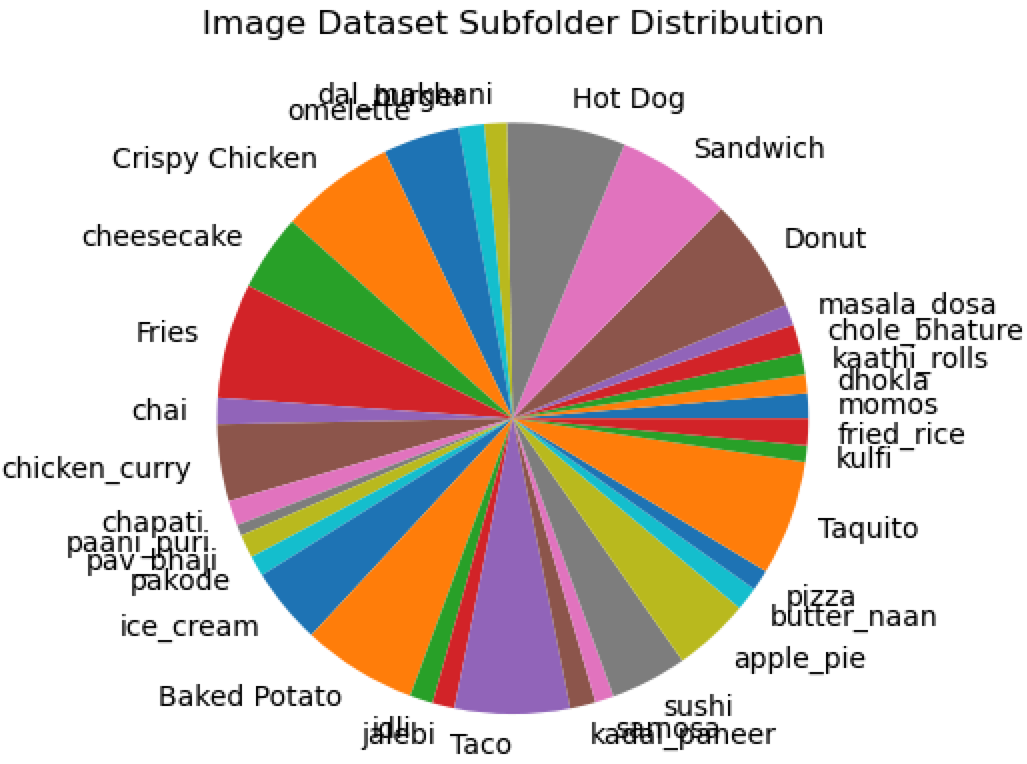
```
sorted_counts = sorted(subfolder_counts.items(), key=lambda x: x[1])
labels, counts = zip(*sorted_counts)

# Define brown-to-green color gradient
cmap = mcolors.LinearSegmentedColormap.from_list('BrownToGreen', ['#8B4513',

# Create the bar graph with color gradient
plt.bar(labels, counts, color=cmap(counts))
plt.xlabel('Subfolder')
plt.ylabel('Count')
plt.title('Image Dataset Subfolder Distribution')
plt.xticks(rotation=85)
plt.show()
```



In [12]:

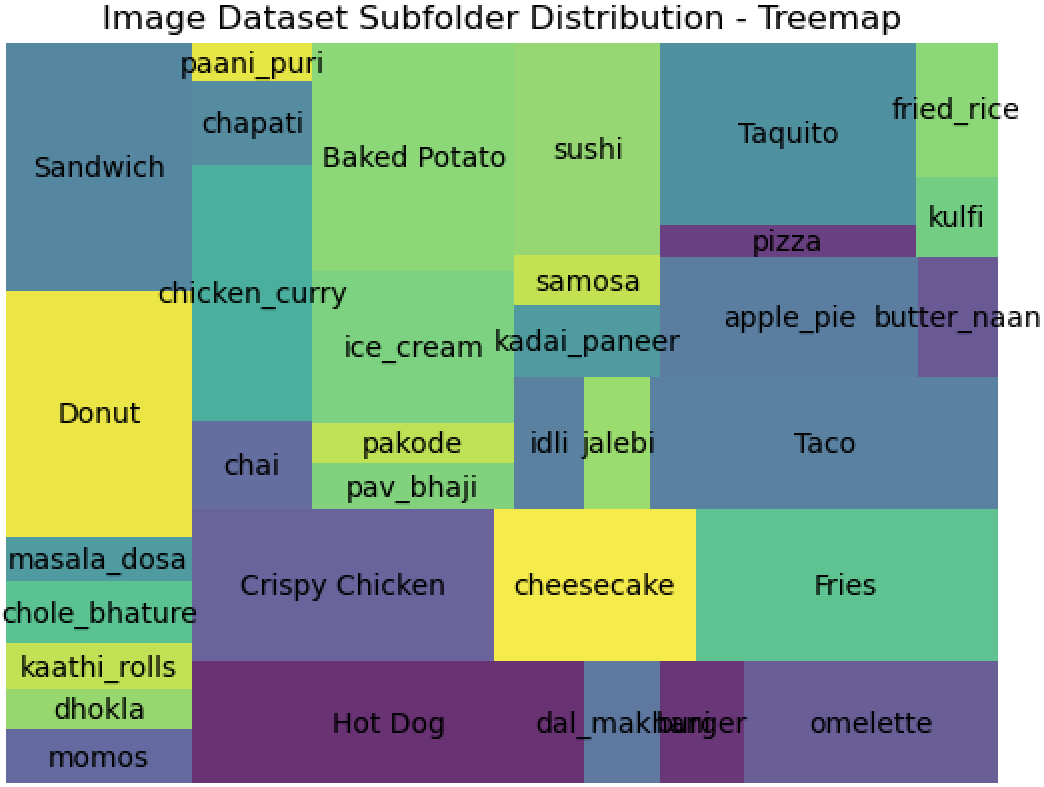
```
import os
import matplotlib.pyplot as plt

# Set the path for the dataset directory
dataset_dir = path

# Initialize a dictionary to store the counts of images in each subfolder
subfolder_counts = {}
```

```
# Iterate over the subfolders in the dataset directory
for subfolder in os.listdir(dataset_dir):
    subfolder_path = os.path.join(dataset_dir, subfolder)
    if os.path.isdir(subfolder_path):
        # Count the number of image files in each subfolder
        subfolder_counts[subfolder] = len(os.listdir(subfolder_path))

# Extract the subfolder names and their respective counts
labels = list(subfolder_counts.keys())
counts = list(subfolder_counts.values())

# Create the pie chart
plt.pie(counts, labels=labels)
plt.title('Image Dataset Subfolder Distribution')

plt.show()
```



In [13]:

```
import os
import matplotlib.pyplot as plt
import squarify

# Set the path for the dataset directory
dataset_dir = path

# Initialize a dictionary to store the counts of images in each subfolder
subfolder_counts = {}

# Iterate over the subfolders in the dataset directory
for subfolder in os.listdir(dataset_dir):
    subfolder_path = os.path.join(dataset_dir, subfolder)
    if os.path.isdir(subfolder_path):
        # Count the number of image files in each subfolder
        subfolder_counts[subfolder] = len(os.listdir(subfolder_path))

# Extract the subfolder names and their respective counts
labels = list(subfolder_counts.keys())
counts = list(subfolder_counts.values())

# Calculate the relative sizes for the treemap
sizes = [count/sum(counts) for count in counts]

# Create the treemap
squarify.plot(sizes=sizes, label=labels, alpha=0.8)

# Configure plot settings
plt.axis('off')
plt.title('Image Dataset Subfolder Distribution - Treemap')
plt.show()
```



Preprocessing

In [14]:
```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
import os
```

In [15]:
```
IMG_SIZE = (256, 256)
VALID_SPLIT = 0.3
BATCH_SIZE = 32
SEED = 42
PATH = "/kaggle/input/food-image-classification-dataset/Food Classification
```

In [16]:

```
def decode_img(image):
    """ Decode the image to ensure it is readable. """
    try:
        img = tf.image.decode_jpeg(image, channels=3)
        img = tf.image.resize(img, IMG_SIZE)
        return img
    except tf.errors.InvalidArgumentError:
        return None  # Return None for invalid image

def load_image(image_path):
    """ Read the image from path and decode. """
    img = tf.io.read_file(image_path)
    img = decode_img(img)
    if img is None:
        return tf.zeros(IMG_SIZE)  # Return a black image as a placeholder
    return img
```

In [17]:

```
# Load and preprocess training and validation datasets
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    PATH,
    validation_split=VALID_SPLIT,
    subset="training",
    seed=SEED,
    image_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    label_mode="categorical"
)

val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    PATH,
    validation_split=VALID_SPLIT,
    subset="validation",
    seed=SEED,
    image_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    label_mode="categorical"
)

# Apply normalization
normalization_layer = tf.keras.layers.Rescaling(1./255)

# Apply normalization and data augmentation
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.RandomFlip('horizontal'),
    tf.keras.layers.RandomRotation(0.2),
    tf.keras.layers.RandomZoom(0.2),
])

# Applying augmentation and normalization
train_ds = train_ds.map(lambda x, y: (data_augmentation(x), y))
train_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
val_ds = val_ds.map(lambda x, y: (normalization_layer(x), y))

# Prefetch to improve performance
```

```
train_ds = train_ds.cache().prefetch(buffer_size=tf.data.experimental.AUTOTU
val_ds = val_ds.cache().prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
```

```
Found 23873 files belonging to 34 classes.
Using 16712 files for training.
Found 23873 files belonging to 34 classes.
Using 7161 files for validation.
```

****Build the ANN Model****

In [18]:

```
# Build the ANN model
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(IMG_SIZE[0], IMG_SIZE[1], 3)),  # F
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(34, activation='softmax')  # 34 output classes
])

# Model summary
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 flatten (Flatten)           (None, 196608)            0         
                                                                 
 dense (Dense)               (None, 512)               100663808 
                                                                 
 dropout (Dropout)           (None, 512)               0         
                                                                 
 dense_1 (Dense)             (None, 256)               131328    
                                                                 
 dropout_1 (Dropout)         (None, 256)               0         
                                                                 
 dense_2 (Dense)             (None, 128)               32896     
                                                                 
 dropout_2 (Dropout)         (None, 128)               0         
                                                                 
 dense_3 (Dense)             (None, 34)                4386      
                                                                 
=================================================================
Total params: 100,832,418
Trainable params: 100,832,418
Non-trainable params: 0
_________________________________________________________________
```

In [22]:

```
# Compile the model
model.compile(
    optimizer=tf.keras.optimizers.Adam(),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

In [25]:

```
import tensorflow as tf
import os
import glob
import shutil
import tempfile
from PIL import Image
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, GlobalAveragePooling2D,
from tensorflow.keras.applications import VGG16
from tensorflow.keras.optimizers import Adam

# Paths
original_data_dir = "/kaggle/input/food-image-classification-dataset/Food Cl
selected_classes = ['Baked Potato', 'Crispy Chicken', 'Donut', 'Fries', 'Hot

# Step 1: Filter valid, readable, TensorFlow-compatible images
def prepare_clean_data(original_dir, selected_classes):
    temp_dir = tempfile.mkdtemp()
    valid_extensions = ('.jpg', '.jpeg', '.png', '.bmp')

    for class_name in selected_classes:
        src_folder = os.path.join(original_dir, class_name)
        dst_folder = os.path.join(temp_dir, class_name)
        os.makedirs(dst_folder, exist_ok=True)

        for file in os.listdir(src_folder):
            src_file = os.path.join(src_folder, file)
            if file.lower().endswith(valid_extensions):
                try:
                    # Open and re-save image to guarantee proper encoding
                    with Image.open(src_file) as img:
                        img = img.convert('RGB')  # force RGB mode
                        dst_file = os.path.join(dst_folder, file)
                        img.save(dst_file, format='JPEG')  # re-save as good
                except Exception as e:
                    print(f"Skipping corrupted or incompatible image: {src_f
                    continue
    return temp_dir

filtered_data_dir = prepare_clean_data(original_data_dir, selected_classes)

# Step 2: Load datasets
batch_size = 32
img_size = (256, 256)

train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    filtered_data_dir,
    labels='inferred',
    label_mode='categorical',
    batch_size=batch_size,
    image_size=img_size,
    validation_split=0.3,
    subset='training',
    seed=123,
    shuffle=True
)

val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    filtered_data_dir,
    labels='inferred',
    label_mode='categorical',
    batch_size=batch_size,
    image_size=img_size,
    validation_split=0.3,
    subset='validation',
    seed=123,
    shuffle=True
)

# Step 3: Prefetch for performance
AUTOTUNE = tf.data.AUTOTUNE
train_ds = train_ds.prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.prefetch(buffer_size=AUTOTUNE)

# Step 4: Build model
base_model = VGG16(include_top=False, weights='imagenet', input_shape=(256,
base_model.trainable = False  # freeze VGG16

model = Sequential([
    Rescaling(1./255, input_shape=(256, 256, 3)),
    base_model,
    GlobalAveragePooling2D(),
    Dense(512, activation='relu'),
    Dropout(0.3),
    Dense(256, activation='relu'),
    Dropout(0.3),
    Dense(128, activation='relu'),
    Dropout(0.2),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(len(selected_classes), activation='softmax')
])

model.compile(
    optimizer=Adam(learning_rate=1e-4),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

model.summary()

# Step 5: Train
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=20,
    verbose=1
)
```

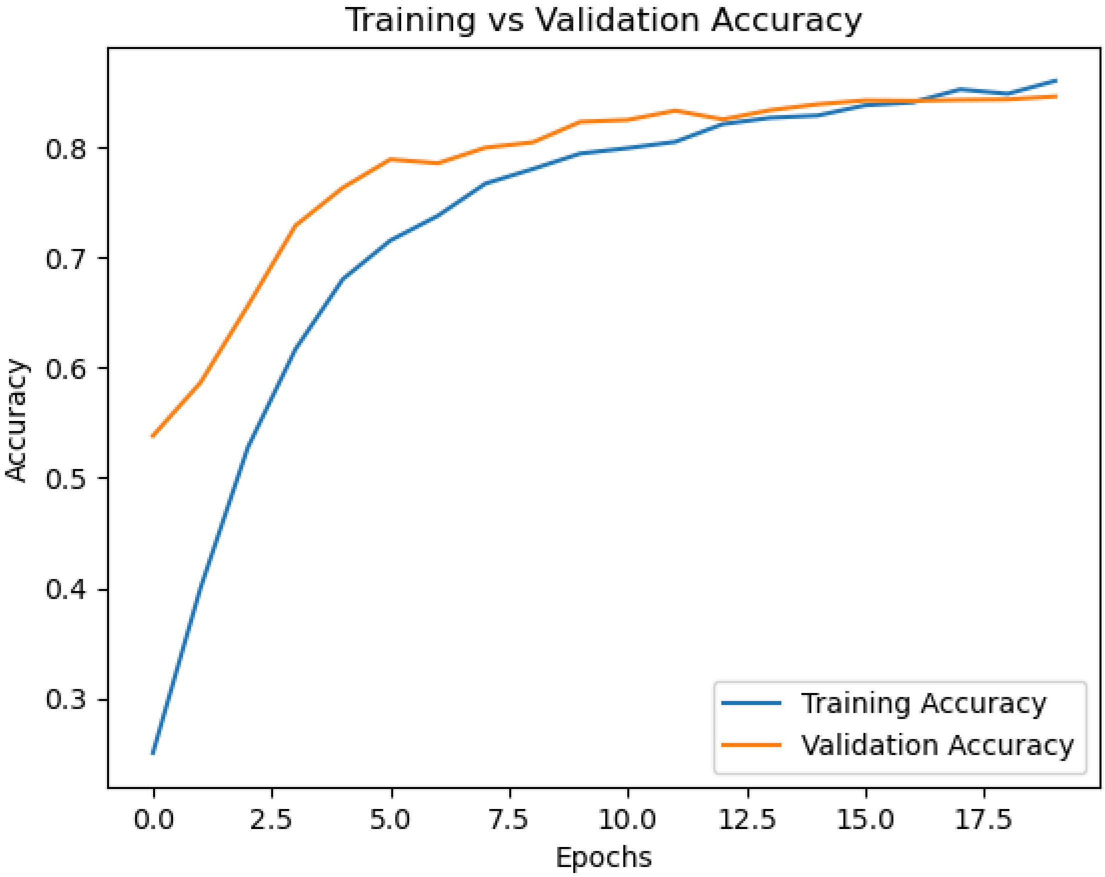
```
Found 7548 files belonging to 5 classes.
Using 5284 files for training.
Found 7548 files belonging to 5 classes.
Using 2264 files for validation.
Downloading data from https://storage.googleapis.com/tensorflow/keras-applic
ations/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 [==============================] - 0s 0us/step
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_1 (Rescaling)     (None, 256, 256, 3)       0

 vgg16 (Functional)          (None, 8, 8, 512)         14714688

 global_average_pooling2d (G  (None, 512)              0
 lobalAveragePooling2D)

 dense_12 (Dense)            (None, 512)               262656

 dropout_9 (Dropout)         (None, 512)               0

 dense_13 (Dense)            (None, 256)               131328

 dropout_10 (Dropout)        (None, 256)               0

 dense_14 (Dense)            (None, 128)               32896

 dropout_11 (Dropout)        (None, 128)               0

 dense_15 (Dense)            (None, 64)                8256

 dropout_12 (Dropout)        (None, 64)                0

 dense_16 (Dense)            (None, 5)                 325

=================================================================
Total params: 15,150,149
Trainable params: 435,461
Non-trainable params: 14,714,688
_________________________________________________________________
Epoch 1/20
166/166 [==============================] - 61s 310ms/step - loss: 1.5956 - a
ccuracy: 0.2508 - val_loss: 1.4683 - val_accuracy: 0.5380
Epoch 2/20
166/166 [==============================] - 45s 268ms/step - loss: 1.4067 - a
ccuracy: 0.3999 - val_loss: 1.1155 - val_accuracy: 0.5861
Epoch 3/20
166/166 [==============================] - 44s 266ms/step - loss: 1.1514 - a
ccuracy: 0.5276 - val_loss: 0.9329 - val_accuracy: 0.6559
Epoch 4/20
166/166 [==============================] - 44s 266ms/step - loss: 0.9931 - a
ccuracy: 0.6164 - val_loss: 0.7782 - val_accuracy: 0.7284
Epoch 5/20
166/166 [==============================] - 44s 265ms/step - loss: 0.8682 - a
ccuracy: 0.6800 - val_loss: 0.6760 - val_accuracy: 0.7628
```

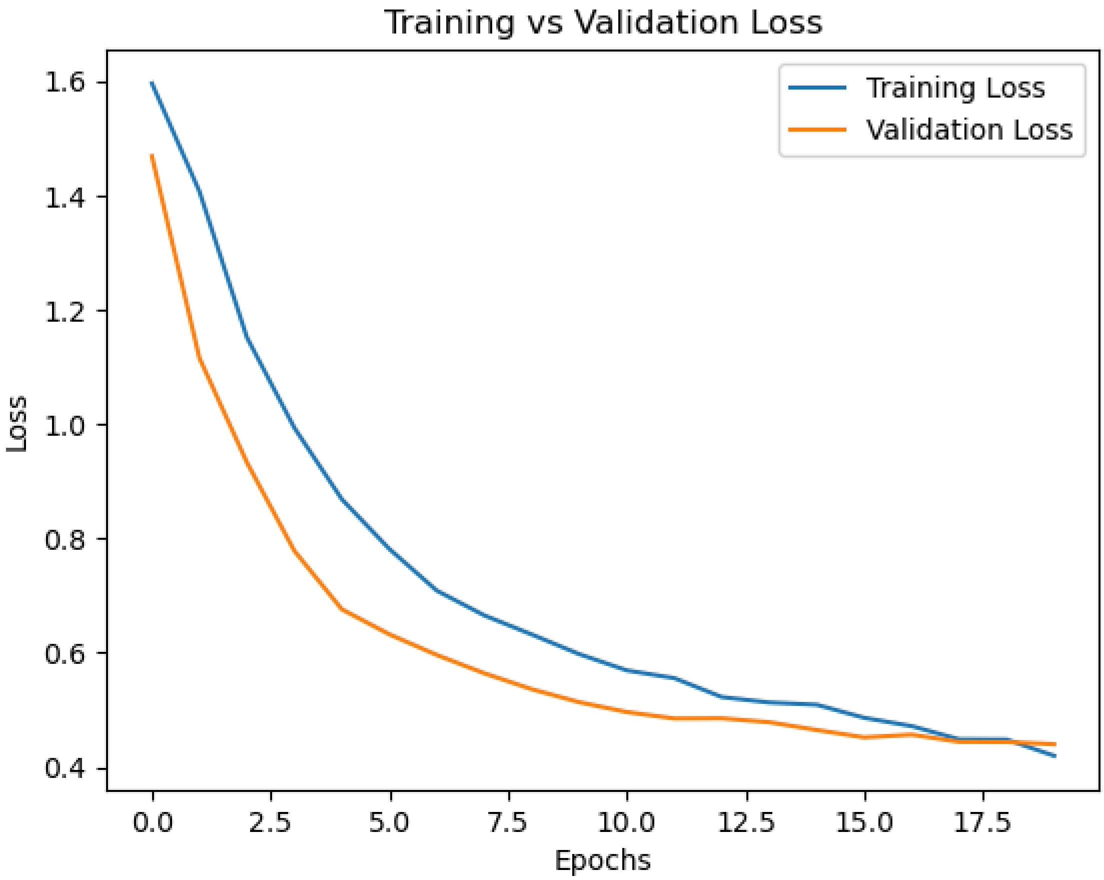
```
Epoch 6/20
166/166 [==============================] - 44s 264ms/step - loss: 0.7814 - a
ccuracy: 0.7150 - val_loss: 0.6323 - val_accuracy: 0.7884
Epoch 7/20
166/166 [==============================] - 44s 267ms/step - loss: 0.7084 - a
ccuracy: 0.7373 - val_loss: 0.5961 - val_accuracy: 0.7849
Epoch 8/20
166/166 [==============================] - 44s 266ms/step - loss: 0.6652 - a
ccuracy: 0.7665 - val_loss: 0.5641 - val_accuracy: 0.7990
Epoch 9/20
166/166 [==============================] - 44s 265ms/step - loss: 0.6318 - a
ccuracy: 0.7797 - val_loss: 0.5362 - val_accuracy: 0.8039
Epoch 10/20
166/166 [==============================] - 44s 265ms/step - loss: 0.5974 - a
ccuracy: 0.7937 - val_loss: 0.5135 - val_accuracy: 0.8224
Epoch 11/20
166/166 [==============================] - 44s 264ms/step - loss: 0.5691 - a
ccuracy: 0.7986 - val_loss: 0.4963 - val_accuracy: 0.8242
Epoch 12/20
166/166 [==============================] - 44s 266ms/step - loss: 0.5555 - a
ccuracy: 0.8043 - val_loss: 0.4850 - val_accuracy: 0.8326
Epoch 13/20
166/166 [==============================] - 44s 266ms/step - loss: 0.5224 - a
ccuracy: 0.8204 - val_loss: 0.4853 - val_accuracy: 0.8246
Epoch 14/20
166/166 [==============================] - 44s 266ms/step - loss: 0.5129 - a
ccuracy: 0.8261 - val_loss: 0.4785 - val_accuracy: 0.8330
Epoch 15/20
166/166 [==============================] - 44s 266ms/step - loss: 0.5090 - a
ccuracy: 0.8282 - val_loss: 0.4649 - val_accuracy: 0.8383
Epoch 16/20
166/166 [==============================] - 44s 264ms/step - loss: 0.4861 - a
ccuracy: 0.8374 - val_loss: 0.4520 - val_accuracy: 0.8419
Epoch 17/20
166/166 [==============================] - 44s 264ms/step - loss: 0.4716 - a
ccuracy: 0.8401 - val_loss: 0.4567 - val_accuracy: 0.8414
Epoch 18/20
166/166 [==============================] - 44s 267ms/step - loss: 0.4485 - a
ccuracy: 0.8518 - val_loss: 0.4443 - val_accuracy: 0.8423
Epoch 19/20
166/166 [==============================] - 44s 266ms/step - loss: 0.4481 - a
ccuracy: 0.8478 - val_loss: 0.4439 - val_accuracy: 0.8428
Epoch 20/20
166/166 [==============================] - 44s 265ms/step - loss: 0.4197 - a
ccuracy: 0.8596 - val_loss: 0.4401 - val_accuracy: 0.8454
```

In [26]:

```
# Evaluate the model
loss, accuracy = model.evaluate(val_ds)
print(f"Validation Loss: {loss:.4f}")
print(f"Validation Accuracy: {accuracy:.4f}")

# Plot training and validation accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training vs Validation Accuracy')
plt.show()

# Plot training and validation loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Training vs Validation Loss')
plt.show()
```

```
71/71 [==============================] - 13s 184ms/step - loss: 0.4401 - acc
uracy: 0.8454
Validation Loss: 0.4401
Validation Accuracy: 0.8454
```




Training vs Validation Loss
Training Loss
Validation Loss
Loss
1.6
1.4
1.2
1.0
0.8
0.6
0.4
Epochs
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

```
In [ ]:
```