Food Image Classification Project

Food Vision project using Transfer Learning with Tensorflow and Keras

Loading the Data

The code snippet is using the wget command to download a file from a specified URL. The file being downloaded appears to be an archive named archive_indian.zip, which is likely to contain data or resources related to my project.Once executed, the file will be saved in the current working directory for further use, such as data extraction or analysis.

In [3]:
```
!wget https://drive.usercontent.google.com/download?id=1wjLZ0juO3zsCkXmaGCuwz13Z6DdbxwVm&export=download&authus
```

```
--2024-12-30 13:10:07--  https://drive.usercontent.google.com/download?id=1wjLZ0juO3zsCkXmaGCuwz13Z6DdbxwVm
Resolving drive.usercontent.google.com (drive.usercontent.google.com)... 142.250.101.132, 2607:f8b0:4023:c06::84
Connecting to drive.usercontent.google.com (drive.usercontent.google.com)|142.250.101.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2378 (2.3K) [text/html]
Saving to: ‘download?id=1wjLZ0juO3zsCkXmaGCuwz13Z6DdbxwVm’

download?id=1wjLZ0j 100%[===================>]   2.32K  --.-KB/s    in 0s

2024-12-30 13:10:07 (43.0 MB/s) - ‘download?id=1wjLZ0juO3zsCkXmaGCuwz13Z6DdbxwVm’ saved [2378/2378]
```

Using Python in a Google Colab environment, the workflow includes mounting Google Drive, copying files to the working directory, and extracting compressed files into a specified folder. This streamlined approach ensures efficient handling of data stored in ZIP formats, making it ready for use in various machine learning, data analysis, or software development tasks.

In [4]:
```
from google.colab import drive
drive.mount('/content/drive')

# Copy the file from Google Drive
!cp /content/drive/MyDrive/archive_indian.zip .

# Extract the ZIP file
import zipfile

with zipfile.ZipFile("archive_indian.zip", 'r') as zip_ref:
    zip_ref.extractall("extracted_folder")

print("Extraction complete. Files:")
!ls extracted_folder
```

```
Mounted at /content/drive
Extraction complete. Files:
dataset  Model  test
```

The code snippet below defines the file paths for the training and testing datasets, pointing to their respective directories where the data is stored:

train_path is set to the directory containing training data, and test_path is set to the directory containing validation or testing data.

In [5]:
```
train_path ="/content/extracted_folder/dataset/Dataset/train"
test_path = "/content/extracted_folder/dataset/Dataset/val"
```

In [6]:
```
!wget https://raw.githubusercontent.com/mrdbourke/tensorflow-deep-learning/refs/heads/main/extras/helper_functi
```

```
--2024-12-30 13:44:16--  https://raw.githubusercontent.com/mrdbourke/tensorflow-deep-learning/refs/heads/main/ex
tras/helper_functions.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.109.133, 185.199.110
.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 10246 (10K) [text/plain]
Saving to: ‘helper_functions.py’

helper_functions.py 100%[===================>]  10.01K  --.-KB/s    in 0.001s

2024-12-30 13:44:16 (15.7 MB/s) - ‘helper_functions.py’ saved [10246/10246]
```

In [7]:
```
# Import helper functions we're going to use
```

```
from helper_functions import create_tensorboard_callback, plot_loss_curves, unzip_data, walk_through_dir
```

In [8]:
```
# Walk through 10 percent data directory and list number of files
walk_through_dir("dataset")
```

In [9]:
```
train_path[1:]
```

Out[9]:
```
'content/extracted_folder/dataset/Dataset/train'
```

Data Processing and Understanding

This code below focuses on analyzing the contents of a dataset directory to determine the distribution of images across its folders. Using Python, the code traverses the directory structure, counts the number of subfolders and image files within each directory, and outputs these details. This provides an overview of the dataset's organization, helping to assess data availability and structure for tasks such as image classification or dataset preprocessing in machine learning projects.

In [10]:
```
# How many images in each folder?
import os

# Walk through 10 percent data directory and list number of files
for dirpath, dirnames, filenames in os.walk("extracted_folder"):
  print(f"There are {len(dirnames)} directories and {len(filenames)} images in '{dirpath}'.")
```

```
There are 3 directories and 0 images in 'extracted_folder'.
There are 1 directories and 4 images in 'extracted_folder/Model'.
There are 1 directories and 1 images in 'extracted_folder/Model/v1_inceptionV3'.
There are 0 directories and 2 images in 'extracted_folder/Model/v1_inceptionV3/variables'.
There are 1 directories and 0 images in 'extracted_folder/dataset'.
There are 2 directories and 0 images in 'extracted_folder/dataset/Dataset'.
There are 20 directories and 0 images in 'extracted_folder/dataset/Dataset/train'.
There are 0 directories and 134 images in 'extracted_folder/dataset/Dataset/train/kulfi'.
There are 0 directories and 191 images in 'extracted_folder/dataset/Dataset/train/kaathi_rolls'.
There are 0 directories and 250 images in 'extracted_folder/dataset/Dataset/train/fried_rice'.
There are 0 directories and 187 images in 'extracted_folder/dataset/Dataset/train/masala_dosa'.
There are 0 directories and 85 images in 'extracted_folder/dataset/Dataset/train/paani_puri'.
There are 0 directories and 179 images in 'extracted_folder/dataset/Dataset/train/pakode'.
There are 0 directories and 184 images in 'extracted_folder/dataset/Dataset/train/pizza'.
There are 0 directories and 203 images in 'extracted_folder/dataset/Dataset/train/dal_makhani'.
There are 0 directories and 228 images in 'extracted_folder/dataset/Dataset/train/kadai_paneer'.
There are 0 directories and 210 images in 'extracted_folder/dataset/Dataset/train/pav_bhaji'.
There are 0 directories and 194 images in 'extracted_folder/dataset/Dataset/train/jalebi'.
There are 0 directories and 207 images in 'extracted_folder/dataset/Dataset/train/idli'.
There are 0 directories and 230 images in 'extracted_folder/dataset/Dataset/train/chapati'.
There are 0 directories and 247 images in 'extracted_folder/dataset/Dataset/train/chai'.
There are 0 directories and 261 images in 'extracted_folder/dataset/Dataset/train/chole_bhature'.
There are 0 directories and 225 images in 'extracted_folder/dataset/Dataset/train/momos'.
There are 0 directories and 217 images in 'extracted_folder/dataset/Dataset/train/butter_naan'.
There are 0 directories and 233 images in 'extracted_folder/dataset/Dataset/train/burger'.
There are 0 directories and 167 images in 'extracted_folder/dataset/Dataset/train/dhokla'.
There are 0 directories and 164 images in 'extracted_folder/dataset/Dataset/train/samosa'.
There are 20 directories and 0 images in 'extracted_folder/dataset/Dataset/val'.
There are 0 directories and 49 images in 'extracted_folder/dataset/Dataset/val/kulfi'.
There are 0 directories and 61 images in 'extracted_folder/dataset/Dataset/val/kaathi_rolls'.
There are 0 directories and 72 images in 'extracted_folder/dataset/Dataset/val/fried_rice'.
There are 0 directories and 60 images in 'extracted_folder/dataset/Dataset/val/masala_dosa'.
There are 0 directories and 29 images in 'extracted_folder/dataset/Dataset/val/paani_puri'.
There are 0 directories and 59 images in 'extracted_folder/dataset/Dataset/val/pakode'.
There are 0 directories and 60 images in 'extracted_folder/dataset/Dataset/val/pizza'.
There are 0 directories and 67 images in 'extracted_folder/dataset/Dataset/val/dal_makhani'.
There are 0 directories and 78 images in 'extracted_folder/dataset/Dataset/val/kadai_paneer'.
There are 0 directories and 61 images in 'extracted_folder/dataset/Dataset/val/pav_bhaji'.
There are 0 directories and 66 images in 'extracted_folder/dataset/Dataset/val/jalebi'.
There are 0 directories and 67 images in 'extracted_folder/dataset/Dataset/val/idli'.
There are 0 directories and 69 images in 'extracted_folder/dataset/Dataset/val/chapati'.
There are 0 directories and 67 images in 'extracted_folder/dataset/Dataset/val/chai'.
There are 0 directories and 83 images in 'extracted_folder/dataset/Dataset/val/chole_bhature'.
There are 0 directories and 68 images in 'extracted_folder/dataset/Dataset/val/momos'.
There are 0 directories and 61 images in 'extracted_folder/dataset/Dataset/val/butter_naan'.
There are 0 directories and 67 images in 'extracted_folder/dataset/Dataset/val/burger'.
There are 0 directories and 55 images in 'extracted_folder/dataset/Dataset/val/dhokla'.
There are 0 directories and 51 images in 'extracted_folder/dataset/Dataset/val/samosa'.
There are 20 directories and 0 images in 'extracted_folder/test'.
There are 0 directories and 31 images in 'extracted_folder/test/kulfi'.
There are 0 directories and 27 images in 'extracted_folder/test/kaathi_rolls'.
There are 0 directories and 28 images in 'extracted_folder/test/fried_rice'.
There are 0 directories and 29 images in 'extracted_folder/test/masala_dosa'.
There are 0 directories and 30 images in 'extracted_folder/test/paani_puri'.
There are 0 directories and 32 images in 'extracted_folder/test/pakode'.
There are 0 directories and 31 images in 'extracted_folder/test/pizza'.
There are 0 directories and 25 images in 'extracted_folder/test/dal_makhani'.
There are 0 directories and 28 images in 'extracted_folder/test/kadai_paneer'.
There are 0 directories and 34 images in 'extracted_folder/test/pav_bhaji'.
There are 0 directories and 29 images in 'extracted_folder/test/jalebi'.
There are 0 directories and 28 images in 'extracted_folder/test/idli'.
There are 0 directories and 28 images in 'extracted_folder/test/chapati'.
There are 0 directories and 30 images in 'extracted_folder/test/chai'.
There are 0 directories and 32 images in 'extracted_folder/test/chole_bhature'.
There are 0 directories and 30 images in 'extracted_folder/test/momos'.
There are 0 directories and 29 images in 'extracted_folder/test/butter_naan'.
There are 0 directories and 31 images in 'extracted_folder/test/burger'.
There are 0 directories and 23 images in 'extracted_folder/test/dhokla'.
There are 0 directories and 30 images in 'extracted_folder/test/samosa'.
```

The provided code imports key libraries and modules from TensorFlow and Keras for building and training deep learning models, specifically for tasks involving image data. It includes tools for constructing sequential neural network models, leveraging pre-trained architectures like ResNet50, and defining layers, optimizers, and loss functions. The code also incorporates data augmentation techniques via ImageDataGenerator to enhance model performance and callbacks like EarlyStopping and ModelCheckpoint for efficient and robust training. This setup is foundational for implementing and fine-tuning advanced image classification or recognition systems.

Loading the dependencies

In [11]:
```
import tensorflow as tf
import tensorflow_hub as hub
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential  # For creating Sequential models
from tensorflow.keras.applications import ResNet50  # If using pre-trained models from Keras Applications
from tensorflow.keras import layers, models, optimizers, losses, metrics
from tensorflow.keras.models import Sequential
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np
```

In [12]:
```
import random
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

The code sets up data inputs for a deep learning image classification task using TensorFlow and Keras. It defines paths for training and testing datasets and configures an ImageDataGenerator to preprocess image data by rescaling pixel values to a range of 0 to 1. Training and testing datasets are loaded using the .flow_from_directory method, which organizes images into batches of a specified size (32) and resizes them to a uniform shape (224x224). The class_mode="categorical" parameter indicates that the data is prepared for multi-class classification tasks. This setup ensures efficient and standardized input data for training and evaluation of the model.

Cross-Validation

In [13]:
```
# Setup data inputs
from tensorflow.keras.preprocessing.image import ImageDataGenerator

IMAGE_SHAPE = (224, 224)
BATCH_SIZE = 32

train_path = "/content/extracted_folder/dataset/Dataset/train"
test_path = "/content/extracted_folder/dataset/Dataset/val"

train_datagen = ImageDataGenerator(rescale=1/255.)
test_datagen = ImageDataGenerator(rescale=1/255.)

print("Training images:")
train_data = train_datagen.flow_from_directory(train_path,
                                               target_size=IMAGE_SHAPE,
                                               batch_size=BATCH_SIZE,
                                               class_mode="categorical")

print("Testing images:")
test_data = train_datagen.flow_from_directory(test_path,
                                              target_size=IMAGE_SHAPE,
                                              batch_size=BATCH_SIZE,
                                              class_mode="categorical")
```

```
Training images:
Found 3996 images belonging to 20 classes.
Testing images:
Found 1250 images belonging to 20 classes.
```

The code defines a function to create a TensorBoard callback for monitoring and visualizing training progress in TensorFlow models. The function, create_tensorboard_callback, dynamically generates a log directory path based on the provided directory name, experiment name, and the current date and time. It initializes a TensorBoard callback that will save log files to this path, enabling detailed tracking of metrics such as loss, accuracy, and other custom evaluations during training. This modular approach allows the creation of unique TensorBoard logs for different experiments, facilitating organized and efficient experimentation.

In [14]:
```
# Create tensorboard callback (functionized because need to create a new one for each model)
import datetime
def create_tensorboard_callback(dir_name, experiment_name):
  log_dir = dir_name + "/" + experiment_name + "/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
  tensorboard_callback = tf.keras.callbacks.TensorBoard(
      log_dir=log_dir
  )
  print(f"Saving TensorBoard log files to: {log_dir}")
  return tensorboard_callback
```

The code snippet specifies URLs for accessing pre-trained feature vector models, including ResNet 50 V2 and EfficientNetB0 (versions 1 and 2). These models are widely used in deep learning for feature extraction, particularly in transfer learning tasks. The URLs point to repositories where the models are hosted, enabling seamless integration into TensorFlow projects. By leveraging these pre-trained models, developers can utilize robust feature representations trained on large datasets like ImageNet, which helps improve performance and reduce training time for custom machine learning tasks.

In [15]:
```
# Resnet 50 V2 feature vector
resnet_url = "https://www.kaggle.com/models/google/resnet-v2/TensorFlow2/50-feature-vector/2"

# Original: EfficientNetB0 feature vector (version 1)
efficientnet_url = "https://www.kaggle.com/models/google/efficientnet-v2/TensorFlow2/imagenet1k-b0-feature-vect
```

```
# # New: EfficientNetB0 feature vector (version 2)
# efficientnet_url = "https://tfhub.dev/google/imagenet/efficientnet_v2_imagenet1k_b0/feature_vector/2"
```

The above code defines a function to create a customizable image classification model using a pre-trained feature extraction layer from TensorFlow Hub. The model leverages transfer learning by freezing the pre-trained layer's weights to retain its learned patterns while adding new layers for task-specific classification. It incorporates an additional dense layer for intermediate feature learning and a final output layer tailored to classify images into a specified number of categories. This approach allows efficient training and robust performance, even with limited data, by building on the strengths of a pre-trained model.

Model Building

In [16]:
```
def create_model(model_url, num_classes=20):
  """Takes a TensorFlow Hub URL and creates a Keras Sequential model with it.

  Args:
    model_url (str): A TensorFlow Hub feature extraction URL.
    num_classes (int): Number of output neurons in output layer,
      should be equal to number of target classes, default 10.

  Returns:
    An uncompiled Keras Sequential model with model_url as feature
    extractor layer and Dense output layer with num_classes outputs.
  """
  # Download the pretrained model and save it as a Keras layer
  feature_extractor_layer = hub.KerasLayer(model_url,
                                           trainable=False, # freeze the underlying patterns
                                           name='feature_extraction_layer',
                                           input_shape=(32, 224, 224, 3)) # define the input image shape

  # Create our own model
  model = tf.keras.Sequential([
    feature_extractor_layer, # use the feature extraction layer as the base
    tf.keras.layers.Dense(256, activation='relu', name='extra_dense_layer'),  # Added extra layer
    layers.Dense(num_classes, activation='softmax', name='output_layer') # create our own output layer
  ])

  return model
```

The code retrieves a single batch of data from the train_data dataset, which contains both images and their corresponding labels. It then prints the shapes of the images and labels to confirm the structure of the dataset. This helps verify that the data is being loaded and batched correctly, ensuring compatibility with the model during training or evaluation.

In [20]:
```
# Retrieve one batch of data
images, labels = next(train_data)

# Check the shape of images and labels
print("Images shape:", images.shape)
print("Labels shape:", labels.shape)
```

```
Images shape: (32, 224, 224, 3)
Labels shape: (32, 20)
```

In [17]:
```
import tensorflow as tf
import tensorflow_hub as hub

print("TensorFlow version:", tf.__version__)
print("TensorFlow Hub version:", hub.__version__)
```

```
TensorFlow version: 2.17.1
TensorFlow Hub version: 0.16.1
```

This code snippet demonstrates how to use TensorFlow Hub to integrate a pre-trained ResNet model as a feature extractor. It creates a KerasLayer using a ResNet-50 model URL, with the pre-trained weights frozen to retain its learned features. The feature extraction layer is tested with a randomly generated input tensor representing an image batch of size 1 and dimensions (224, 224, 3). The output shape of the feature extractor is printed to verify that the layer processes the input correctly and returns the expected feature vector.

In [18]:
```
import tensorflow as tf
import tensorflow_hub as hub

# Define the ResNet URL
resnet_url = "https://tfhub.dev/google/imagenet/resnet_v2_50/feature_vector/5"

# Create the feature extraction layer
feature_extraction_layer = hub.KerasLayer(
    resnet_url,
    trainable=False,
    input_shape=(224, 224, 3)
)
```

```
# Test with a sample input
sample_input = tf.random.normal([1, 224, 224, 3])  # Batch size = 1
sample_output = feature_extraction_layer(sample_input)
print("Output shape:", sample_output.shape)
```

```
Output shape: (1, 2048)
```

The code then checks if the created layer is a valid Keras layer by verifying its type, ensuring that it integrates smoothly into Keras models for further use in tasks like image classification or other computer vision applications.

In [19]:
```
import tensorflow_hub as hub

resnet_url = "https://www.kaggle.com/models/google/resnet-v2/TensorFlow2/50-feature-vector/2"
feature_extraction_layer = hub.KerasLayer(resnet_url, trainable=False, input_shape=(224, 224, 3))

print("Is KerasLayer a valid Keras layer?", isinstance(feature_extraction_layer, tf.keras.layers.Layer))
```

```
Is KerasLayer a valid Keras layer? False
```

The model is used as a non-trainable feature extraction layer, meaning the pre-trained weights remain fixed during further model training. The input shape for the images is set to 224x224 pixels with 3 color channels. The code then verifies if the loaded layer is a valid Keras layer by checking its type, ensuring it is compatible with Keras models for use in tasks like image recognition or other computer vision applications.

In [20]:
```
mobilenet_url = "https://www.kaggle.com/models/google/mobilenet-v2/TensorFlow2/035-128-feature-vector/2"
feature_extraction_layer = hub.KerasLayer(mobilenet_url, trainable=False, input_shape=(224, 224, 3))
print("Is KerasLayer a valid Keras layer?", isinstance(feature_extraction_layer, tf.keras.layers.Layer))
```

```
Is KerasLayer a valid Keras layer? False
```

Note: Both the ResNet and MobileNetV2 layers were not directly compatible with the model, so I was required to replicate a Keras model from scratch with the necessary parameters.

The code below generates a custom implementation of the ResNet-50 architecture using Keras. It constructs the model by building residual blocks with bottleneck layers, each containing multiple convolutional layers followed by batch normalization and ReLU activation functions. The residual_block function creates individual residual blocks, while the resnet_block function groups multiple blocks together. The main ResNet50 function assembles the model with an initial convolutional layer, followed by several stages of residual blocks, and concludes with global average pooling and a dense softmax output layer. This architecture is suitable for image classification tasks, with the flexibility to adjust the number of output classes. The model is instantiated with an input shape of 224x224x3 and 10 output classes.

Deep Learning Architecture

In [21]:
```
import tensorflow as tf
from tensorflow.keras.layers import Conv2D, BatchNormalization, ReLU, Add, MaxPooling2D, GlobalAveragePooling2D
from tensorflow.keras.models import Model

def residual_block(inputs, filters, strides=(1, 1), use_projection=False):
    """
    Defines a ResNet bottleneck residual block.

    Args:
        inputs: Input tensor.
        filters: Number of filters for the bottleneck block.
        strides: Strides for the convolution.
        use_projection: Whether to use a projection shortcut to match dimensions.

    Returns:
        Output tensor after applying the residual block.
    """
    shortcut = inputs

    # First convolution (1x1)
    x = Conv2D(filters, (1, 1), strides=strides, padding='same')(inputs)
    x = BatchNormalization()(x)
    x = ReLU()(x)

    # Second convolution (3x3)
    x = Conv2D(filters, (3, 3), strides=(1, 1), padding='same')(x)
    x = BatchNormalization()(x)
    x = ReLU()(x)

    # Third convolution (1x1)
    x = Conv2D(filters * 4, (1, 1), strides=(1, 1), padding='same')(x)
    x = BatchNormalization()(x)

    # Use projection shortcut if dimensions differ
    if use_projection:
        shortcut = Conv2D(filters * 4, (1, 1), strides=strides, padding='same')(inputs)
        shortcut = BatchNormalization()(shortcut)
```

```python
    # Add skip connection
    x = Add()([x, shortcut])
    x = ReLU()(x)

    return x

def resnet_block(inputs, filters, blocks, strides=(1, 1)):
    """
    Creates a ResNet block consisting of multiple residual blocks.

    Args:
        inputs: Input tensor.
        filters: Number of filters for the bottleneck block.
        blocks: Number of residual blocks in the block.
        strides: Strides for the first convolution in the first block.

    Returns:
        Output tensor after applying the ResNet block.
    """
    # First block uses projection
    x = residual_block(inputs, filters, strides=strides, use_projection=True)

    # Remaining blocks use identity shortcut
    for _ in range(1, blocks):
        x = residual_block(x, filters, strides=(1, 1))

    return x

def ResNet50(input_shape=(224, 224, 3), num_classes=1000):
    """
    Builds the ResNet-50 model.

    Args:
        input_shape: Shape of the input tensor.
        num_classes: Number of output classes.

    Returns:
        A Keras Model representing the ResNet-50 architecture.
    """
    inputs = Input(shape=input_shape)

    # Initial convolution and max-pooling
    x = Conv2D(64, (7, 7), strides=(2, 2), padding='same')(inputs)
    x = BatchNormalization()(x)
    x = ReLU()(x)
    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)

    # ResNet blocks
    x = resnet_block(x, 64, blocks=3, strides=(1, 1))  # Conv2_x
    x = resnet_block(x, 128, blocks=4, strides=(2, 2))  # Conv3_x
    x = resnet_block(x, 256, blocks=6, strides=(2, 2))  # Conv4_x
    x = resnet_block(x, 512, blocks=3, strides=(2, 2))  # Conv5_x

    # Global Average Pooling and Dense output layer
    x = GlobalAveragePooling2D()(x)
    outputs = Dense(num_classes, activation='softmax')(x)

    # Create model
    model = Model(inputs, outputs, name="ResNet50")

    return model

# Instantiate and summarize the model
resnet_model = ResNet50(input_shape=(224, 224, 3), num_classes=10)
resnet_model.summary()
```

```
Model: "ResNet50"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━┓
┃ Layer (type)               ┃ Output Shape            ┃        Param # ┃ Connected to            ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━┩
│ input_layer (InputLayer)   │ (None, 224, 224, 3)     │              0 │ -                       │
├────────────────────────────┼─────────────────────────┼────────────────┼─────────────────────────┤
│ conv2d (Conv2D)            │ (None, 112, 112, 64)    │          9,472 │ input_layer[0][0]       │
├────────────────────────────┼─────────────────────────┼────────────────┼─────────────────────────┤
│ batch_normalization        │ (None, 112, 112, 64)    │            256 │ conv2d[0][0]            │
│ (BatchNormalization)       │                         │                │                         │
├────────────────────────────┼─────────────────────────┼────────────────┼─────────────────────────┤
│ re_lu (ReLU)               │ (None, 112, 112, 64)    │              0 │ batch_normalization[0… │
├────────────────────────────┼─────────────────────────┼────────────────┼─────────────────────────┤
│ max_pooling2d              │ (None, 56, 56, 64)      │              0 │ re_lu[0][0]             │
│ (MaxPooling2D)             │                         │                │                         │
├────────────────────────────┼─────────────────────────┼────────────────┼─────────────────────────┤
│ conv2d_1 (Conv2D)          │ (None, 56, 56, 64)      │          4,160 │ max_pooling2d[0][0]     │
```

batch_normalization_1 (BatchNormalization)	(None, 56, 56, 64)	256	conv2d_1[0][0]
re_lu_1 (ReLU)	(None, 56, 56, 64)	0	batch_normalization_1…
conv2d_2 (Conv2D)	(None, 56, 56, 64)	36,928	re_lu_1[0][0]
batch_normalization_2 (BatchNormalization)	(None, 56, 56, 64)	256	conv2d_2[0][0]
re_lu_2 (ReLU)	(None, 56, 56, 64)	0	batch_normalization_2…
conv2d_3 (Conv2D)	(None, 56, 56, 256)	16,640	re_lu_2[0][0]
conv2d_4 (Conv2D)	(None, 56, 56, 256)	16,640	max_pooling2d[0][0]
batch_normalization_3 (BatchNormalization)	(None, 56, 56, 256)	1,024	conv2d_3[0][0]
batch_normalization_4 (BatchNormalization)	(None, 56, 56, 256)	1,024	conv2d_4[0][0]
add (Add)	(None, 56, 56, 256)	0	batch_normalization_3… batch_normalization_4…
re_lu_3 (ReLU)	(None, 56, 56, 256)	0	add[0][0]
conv2d_5 (Conv2D)	(None, 56, 56, 64)	16,448	re_lu_3[0][0]
batch_normalization_5 (BatchNormalization)	(None, 56, 56, 64)	256	conv2d_5[0][0]
re_lu_4 (ReLU)	(None, 56, 56, 64)	0	batch_normalization_5…
conv2d_6 (Conv2D)	(None, 56, 56, 64)	36,928	re_lu_4[0][0]
batch_normalization_6 (BatchNormalization)	(None, 56, 56, 64)	256	conv2d_6[0][0]
re_lu_5 (ReLU)	(None, 56, 56, 64)	0	batch_normalization_6…
conv2d_7 (Conv2D)	(None, 56, 56, 256)	16,640	re_lu_5[0][0]
batch_normalization_7 (BatchNormalization)	(None, 56, 56, 256)	1,024	conv2d_7[0][0]
add_1 (Add)	(None, 56, 56, 256)	0	batch_normalization_7… re_lu_3[0][0]
re_lu_6 (ReLU)	(None, 56, 56, 256)	0	add_1[0][0]
conv2d_8 (Conv2D)	(None, 56, 56, 64)	16,448	re_lu_6[0][0]
batch_normalization_8 (BatchNormalization)	(None, 56, 56, 64)	256	conv2d_8[0][0]
re_lu_7 (ReLU)	(None, 56, 56, 64)	0	batch_normalization_8…
conv2d_9 (Conv2D)	(None, 56, 56, 64)	36,928	re_lu_7[0][0]
batch_normalization_9 (BatchNormalization)	(None, 56, 56, 64)	256	conv2d_9[0][0]
re_lu_8 (ReLU)	(None, 56, 56, 64)	0	batch_normalization_9…
conv2d_10 (Conv2D)	(None, 56, 56, 256)	16,640	re_lu_8[0][0]
batch_normalization_10 (BatchNormalization)	(None, 56, 56, 256)	1,024	conv2d_10[0][0]
add_2 (Add)	(None, 56, 56, 256)	0	batch_normalization_1… re_lu_6[0][0]
re_lu_9 (ReLU)	(None, 56, 56, 256)	0	add_2[0][0]
conv2d_11 (Conv2D)	(None, 28, 28, 128)	32,896	re_lu_9[0][0]
batch_normalization_11 (BatchNormalization)	(None, 28, 28, 128)	512	conv2d_11[0][0]
re_lu_10 (ReLU)	(None, 28, 28, 128)	0	batch_normalization_1…

```
│ conv2d_12 (Conv2D)          │ (None, 28, 28, 128)    │     147,584 │ re_lu_10[0][0]        │
├─────────────────────────────┼────────────────────────┼─────────────┼───────────────────────┤
│ batch_normalization_12      │ (None, 28, 28, 128)    │         512 │ conv2d_12[0][0]       │
│ (BatchNormalization)        │                        │             │                       │
├─────────────────────────────┼────────────────────────┼─────────────┼───────────────────────┤
│ re_lu_11 (ReLU)             │ (None, 28, 28, 128)    │           0 │ batch_normalization_1… │
├─────────────────────────────┼────────────────────────┼─────────────┼───────────────────────┤
│ conv2d_13 (Conv2D)          │ (None, 28, 28, 512)    │      66,048 │ re_lu_11[0][0]        │
├─────────────────────────────┼────────────────────────┼─────────────┼───────────────────────┤
│ conv2d_14 (Conv2D)          │ (None, 28, 28, 512)    │     131,584 │ re_lu_9[0][0]         │
├─────────────────────────────┼────────────────────────┼─────────────┼───────────────────────┤
│ batch_normalization_13      │ (None, 28, 28, 512)    │       2,048 │ conv2d_13[0][0]       │
│ (BatchNormalization)        │                        │             │                       │
├─────────────────────────────┼────────────────────────┼─────────────┼───────────────────────┤
│ batch_normalization_14      │ (None, 28, 28, 512)    │       2,048 │ conv2d_14[0][0]       │
│ (BatchNormalization)        │                        │             │                       │
├─────────────────────────────┼────────────────────────┼─────────────┼───────────────────────┤
│ add_3 (Add)                 │ (None, 28, 28, 512)    │           0 │ batch_normalization_1… │
│                             │                        │             │ batch_normalization_1… │
├─────────────────────────────┼────────────────────────┼─────────────┼───────────────────────┤
│ re_lu_12 (ReLU)             │ (None, 28, 28, 512)    │           0 │ add_3[0][0]           │
├─────────────────────────────┼────────────────────────┼─────────────┼───────────────────────┤
│ conv2d_15 (Conv2D)          │ (None, 28, 28, 128)    │      65,664 │ re_lu_12[0][0]        │
├─────────────────────────────┼────────────────────────┼─────────────┼───────────────────────┤
│ batch_normalization_15      │ (None, 28, 28, 128)    │         512 │ conv2d_15[0][0]       │
│ (BatchNormalization)        │                        │             │                       │
├─────────────────────────────┼────────────────────────┼─────────────┼───────────────────────┤
│ re_lu_13 (ReLU)             │ (None, 28, 28, 128)    │           0 │ batch_normalization_1… │
├─────────────────────────────┼────────────────────────┼─────────────┼───────────────────────┤
│ conv2d_16 (Conv2D)          │ (None, 28, 28, 128)    │     147,584 │ re_lu_13[0][0]        │
├─────────────────────────────┼────────────────────────┼─────────────┼───────────────────────┤
│ batch_normalization_16      │ (None, 28, 28, 128)    │         512 │ conv2d_16[0][0]       │
│ (BatchNormalization)        │                        │             │                       │
├─────────────────────────────┼────────────────────────┼─────────────┼───────────────────────┤
│ re_lu_14 (ReLU)             │ (None, 28, 28, 128)    │           0 │ batch_normalization_1… │
├─────────────────────────────┼────────────────────────┼─────────────┼───────────────────────┤
│ conv2d_17 (Conv2D)          │ (None, 28, 28, 512)    │      66,048 │ re_lu_14[0][0]        │
├─────────────────────────────┼────────────────────────┼─────────────┼───────────────────────┤
│ batch_normalization_17      │ (None, 28, 28, 512)    │       2,048 │ conv2d_17[0][0]       │
│ (BatchNormalization)        │                        │             │                       │
├─────────────────────────────┼────────────────────────┼─────────────┼───────────────────────┤
│ add_4 (Add)                 │ (None, 28, 28, 512)    │           0 │ batch_normalization_1… │
│                             │                        │             │ re_lu_12[0][0]        │
├─────────────────────────────┼────────────────────────┼─────────────┼───────────────────────┤
│ re_lu_15 (ReLU)             │ (None, 28, 28, 512)    │           0 │ add_4[0][0]           │
├─────────────────────────────┼────────────────────────┼─────────────┼───────────────────────┤
│ conv2d_18 (Conv2D)          │ (None, 28, 28, 128)    │      65,664 │ re_lu_15[0][0]        │
├─────────────────────────────┼────────────────────────┼─────────────┼───────────────────────┤
│ batch_normalization_18      │ (None, 28, 28, 128)    │         512 │ conv2d_18[0][0]       │
│ (BatchNormalization)        │                        │             │                       │
├─────────────────────────────┼────────────────────────┼─────────────┼───────────────────────┤
│ re_lu_16 (ReLU)             │ (None, 28, 28, 128)    │           0 │ batch_normalization_1… │
├─────────────────────────────┼────────────────────────┼─────────────┼───────────────────────┤
│ conv2d_19 (Conv2D)          │ (None, 28, 28, 128)    │     147,584 │ re_lu_16[0][0]        │
├─────────────────────────────┼────────────────────────┼─────────────┼───────────────────────┤
│ batch_normalization_19      │ (None, 28, 28, 128)    │         512 │ conv2d_19[0][0]       │
│ (BatchNormalization)        │                        │             │                       │
├─────────────────────────────┼────────────────────────┼─────────────┼───────────────────────┤
│ re_lu_17 (ReLU)             │ (None, 28, 28, 128)    │           0 │ batch_normalization_1… │
├─────────────────────────────┼────────────────────────┼─────────────┼───────────────────────┤
│ conv2d_20 (Conv2D)          │ (None, 28, 28, 512)    │      66,048 │ re_lu_17[0][0]        │
├─────────────────────────────┼────────────────────────┼─────────────┼───────────────────────┤
│ batch_normalization_20      │ (None, 28, 28, 512)    │       2,048 │ conv2d_20[0][0]       │
│ (BatchNormalization)        │                        │             │                       │
├─────────────────────────────┼────────────────────────┼─────────────┼───────────────────────┤
│ add_5 (Add)                 │ (None, 28, 28, 512)    │           0 │ batch_normalization_2… │
│                             │                        │             │ re_lu_15[0][0]        │
├─────────────────────────────┼────────────────────────┼─────────────┼───────────────────────┤
│ re_lu_18 (ReLU)             │ (None, 28, 28, 512)    │           0 │ add_5[0][0]           │
├─────────────────────────────┼────────────────────────┼─────────────┼───────────────────────┤
│ conv2d_21 (Conv2D)          │ (None, 28, 28, 128)    │      65,664 │ re_lu_18[0][0]        │
├─────────────────────────────┼────────────────────────┼─────────────┼───────────────────────┤
│ batch_normalization_21      │ (None, 28, 28, 128)    │         512 │ conv2d_21[0][0]       │
│ (BatchNormalization)        │                        │             │                       │
├─────────────────────────────┼────────────────────────┼─────────────┼───────────────────────┤
│ re_lu_19 (ReLU)             │ (None, 28, 28, 128)    │           0 │ batch_normalization_2… │
├─────────────────────────────┼────────────────────────┼─────────────┼───────────────────────┤
│ conv2d_22 (Conv2D)          │ (None, 28, 28, 128)    │     147,584 │ re_lu_19[0][0]        │
├─────────────────────────────┼────────────────────────┼─────────────┼───────────────────────┤
│ batch_normalization_22      │ (None, 28, 28, 128)    │         512 │ conv2d_22[0][0]       │
│ (BatchNormalization)        │                        │             │                       │
├─────────────────────────────┼────────────────────────┼─────────────┼───────────────────────┤
│ re_lu_20 (ReLU)             │ (None, 28, 28, 128)    │           0 │ batch_normalization_2… │
```

conv2d_23 (Conv2D)	(None, 28, 28, 512)	66,048	re_lu_20[0][0]
batch_normalization_23 (BatchNormalization)	(None, 28, 28, 512)	2,048	conv2d_23[0][0]
add_6 (Add)	(None, 28, 28, 512)	0	batch_normalization_2… re_lu_18[0][0]
re_lu_21 (ReLU)	(None, 28, 28, 512)	0	add_6[0][0]
conv2d_24 (Conv2D)	(None, 14, 14, 256)	131,328	re_lu_21[0][0]
batch_normalization_24 (BatchNormalization)	(None, 14, 14, 256)	1,024	conv2d_24[0][0]
re_lu_22 (ReLU)	(None, 14, 14, 256)	0	batch_normalization_2…
conv2d_25 (Conv2D)	(None, 14, 14, 256)	590,080	re_lu_22[0][0]
batch_normalization_25 (BatchNormalization)	(None, 14, 14, 256)	1,024	conv2d_25[0][0]
re_lu_23 (ReLU)	(None, 14, 14, 256)	0	batch_normalization_2…
conv2d_26 (Conv2D)	(None, 14, 14, 1024)	263,168	re_lu_23[0][0]
conv2d_27 (Conv2D)	(None, 14, 14, 1024)	525,312	re_lu_21[0][0]
batch_normalization_26 (BatchNormalization)	(None, 14, 14, 1024)	4,096	conv2d_26[0][0]
batch_normalization_27 (BatchNormalization)	(None, 14, 14, 1024)	4,096	conv2d_27[0][0]
add_7 (Add)	(None, 14, 14, 1024)	0	batch_normalization_2… batch_normalization_2…
re_lu_24 (ReLU)	(None, 14, 14, 1024)	0	add_7[0][0]
conv2d_28 (Conv2D)	(None, 14, 14, 256)	262,400	re_lu_24[0][0]
batch_normalization_28 (BatchNormalization)	(None, 14, 14, 256)	1,024	conv2d_28[0][0]
re_lu_25 (ReLU)	(None, 14, 14, 256)	0	batch_normalization_2…
conv2d_29 (Conv2D)	(None, 14, 14, 256)	590,080	re_lu_25[0][0]
batch_normalization_29 (BatchNormalization)	(None, 14, 14, 256)	1,024	conv2d_29[0][0]
re_lu_26 (ReLU)	(None, 14, 14, 256)	0	batch_normalization_2…
conv2d_30 (Conv2D)	(None, 14, 14, 1024)	263,168	re_lu_26[0][0]
batch_normalization_30 (BatchNormalization)	(None, 14, 14, 1024)	4,096	conv2d_30[0][0]
add_8 (Add)	(None, 14, 14, 1024)	0	batch_normalization_3… re_lu_24[0][0]
re_lu_27 (ReLU)	(None, 14, 14, 1024)	0	add_8[0][0]
conv2d_31 (Conv2D)	(None, 14, 14, 256)	262,400	re_lu_27[0][0]
batch_normalization_31 (BatchNormalization)	(None, 14, 14, 256)	1,024	conv2d_31[0][0]
re_lu_28 (ReLU)	(None, 14, 14, 256)	0	batch_normalization_3…
conv2d_32 (Conv2D)	(None, 14, 14, 256)	590,080	re_lu_28[0][0]
batch_normalization_32 (BatchNormalization)	(None, 14, 14, 256)	1,024	conv2d_32[0][0]
re_lu_29 (ReLU)	(None, 14, 14, 256)	0	batch_normalization_3…
conv2d_33 (Conv2D)	(None, 14, 14, 1024)	263,168	re_lu_29[0][0]
batch_normalization_33 (BatchNormalization)	(None, 14, 14, 1024)	4,096	conv2d_33[0][0]

add_9 (Add)	(None, 14, 14, 1024)	0	batch_normalization_3… re_lu_27[0][0]
re_lu_30 (ReLU)	(None, 14, 14, 1024)	0	add_9[0][0]
conv2d_34 (Conv2D)	(None, 14, 14, 256)	262,400	re_lu_30[0][0]
batch_normalization_34 (BatchNormalization)	(None, 14, 14, 256)	1,024	conv2d_34[0][0]
re_lu_31 (ReLU)	(None, 14, 14, 256)	0	batch_normalization_3…
conv2d_35 (Conv2D)	(None, 14, 14, 256)	590,080	re_lu_31[0][0]
batch_normalization_35 (BatchNormalization)	(None, 14, 14, 256)	1,024	conv2d_35[0][0]
re_lu_32 (ReLU)	(None, 14, 14, 256)	0	batch_normalization_3…
conv2d_36 (Conv2D)	(None, 14, 14, 1024)	263,168	re_lu_32[0][0]
batch_normalization_36 (BatchNormalization)	(None, 14, 14, 1024)	4,096	conv2d_36[0][0]
add_10 (Add)	(None, 14, 14, 1024)	0	batch_normalization_3… re_lu_30[0][0]
re_lu_33 (ReLU)	(None, 14, 14, 1024)	0	add_10[0][0]
conv2d_37 (Conv2D)	(None, 14, 14, 256)	262,400	re_lu_33[0][0]
batch_normalization_37 (BatchNormalization)	(None, 14, 14, 256)	1,024	conv2d_37[0][0]
re_lu_34 (ReLU)	(None, 14, 14, 256)	0	batch_normalization_3…
conv2d_38 (Conv2D)	(None, 14, 14, 256)	590,080	re_lu_34[0][0]
batch_normalization_38 (BatchNormalization)	(None, 14, 14, 256)	1,024	conv2d_38[0][0]
re_lu_35 (ReLU)	(None, 14, 14, 256)	0	batch_normalization_3…
conv2d_39 (Conv2D)	(None, 14, 14, 1024)	263,168	re_lu_35[0][0]
batch_normalization_39 (BatchNormalization)	(None, 14, 14, 1024)	4,096	conv2d_39[0][0]
add_11 (Add)	(None, 14, 14, 1024)	0	batch_normalization_3… re_lu_33[0][0]
re_lu_36 (ReLU)	(None, 14, 14, 1024)	0	add_11[0][0]
conv2d_40 (Conv2D)	(None, 14, 14, 256)	262,400	re_lu_36[0][0]
batch_normalization_40 (BatchNormalization)	(None, 14, 14, 256)	1,024	conv2d_40[0][0]
re_lu_37 (ReLU)	(None, 14, 14, 256)	0	batch_normalization_4…
conv2d_41 (Conv2D)	(None, 14, 14, 256)	590,080	re_lu_37[0][0]
batch_normalization_41 (BatchNormalization)	(None, 14, 14, 256)	1,024	conv2d_41[0][0]
re_lu_38 (ReLU)	(None, 14, 14, 256)	0	batch_normalization_4…
conv2d_42 (Conv2D)	(None, 14, 14, 1024)	263,168	re_lu_38[0][0]
batch_normalization_42 (BatchNormalization)	(None, 14, 14, 1024)	4,096	conv2d_42[0][0]
add_12 (Add)	(None, 14, 14, 1024)	0	batch_normalization_4… re_lu_36[0][0]
re_lu_39 (ReLU)	(None, 14, 14, 1024)	0	add_12[0][0]
conv2d_43 (Conv2D)	(None, 7, 7, 512)	524,800	re_lu_39[0][0]
batch_normalization_43 (BatchNormalization)	(None, 7, 7, 512)	2,048	conv2d_43[0][0]
re_lu_40 (ReLU)	(None, 7, 7, 512)	0	batch_normalization_4…

```
│ conv2d_44 (Conv2D)            │ (None, 7, 7, 512)  │ 2,359,808 │ re_lu_40[0][0]           │
├───────────────────────────────┼────────────────────┼───────────┼──────────────────────────┤
│ batch_normalization_44        │ (None, 7, 7, 512)  │     2,048 │ conv2d_44[0][0]          │
│ (BatchNormalization)          │                    │           │                          │
├───────────────────────────────┼────────────────────┼───────────┼──────────────────────────┤
│ re_lu_41 (ReLU)               │ (None, 7, 7, 512)  │         0 │ batch_normalization_4…   │
├───────────────────────────────┼────────────────────┼───────────┼──────────────────────────┤
│ conv2d_45 (Conv2D)            │ (None, 7, 7, 2048) │ 1,050,624 │ re_lu_41[0][0]           │
├───────────────────────────────┼────────────────────┼───────────┼──────────────────────────┤
│ conv2d_46 (Conv2D)            │ (None, 7, 7, 2048) │ 2,099,200 │ re_lu_39[0][0]           │
├───────────────────────────────┼────────────────────┼───────────┼──────────────────────────┤
│ batch_normalization_45        │ (None, 7, 7, 2048) │     8,192 │ conv2d_45[0][0]          │
│ (BatchNormalization)          │                    │           │                          │
├───────────────────────────────┼────────────────────┼───────────┼──────────────────────────┤
│ batch_normalization_46        │ (None, 7, 7, 2048) │     8,192 │ conv2d_46[0][0]          │
│ (BatchNormalization)          │                    │           │                          │
├───────────────────────────────┼────────────────────┼───────────┼──────────────────────────┤
│ add_13 (Add)                  │ (None, 7, 7, 2048) │         0 │ batch_normalization_4…   │
│                               │                    │           │ batch_normalization_4…   │
├───────────────────────────────┼────────────────────┼───────────┼──────────────────────────┤
│ re_lu_42 (ReLU)               │ (None, 7, 7, 2048) │         0 │ add_13[0][0]             │
├───────────────────────────────┼────────────────────┼───────────┼──────────────────────────┤
│ conv2d_47 (Conv2D)            │ (None, 7, 7, 512)  │ 1,049,088 │ re_lu_42[0][0]           │
├───────────────────────────────┼────────────────────┼───────────┼──────────────────────────┤
│ batch_normalization_47        │ (None, 7, 7, 512)  │     2,048 │ conv2d_47[0][0]          │
│ (BatchNormalization)          │                    │           │                          │
├───────────────────────────────┼────────────────────┼───────────┼──────────────────────────┤
│ re_lu_43 (ReLU)               │ (None, 7, 7, 512)  │         0 │ batch_normalization_4…   │
├───────────────────────────────┼────────────────────┼───────────┼──────────────────────────┤
│ conv2d_48 (Conv2D)            │ (None, 7, 7, 512)  │ 2,359,808 │ re_lu_43[0][0]           │
├───────────────────────────────┼────────────────────┼───────────┼──────────────────────────┤
│ batch_normalization_48        │ (None, 7, 7, 512)  │     2,048 │ conv2d_48[0][0]          │
│ (BatchNormalization)          │                    │           │                          │
├───────────────────────────────┼────────────────────┼───────────┼──────────────────────────┤
│ re_lu_44 (ReLU)               │ (None, 7, 7, 512)  │         0 │ batch_normalization_4…   │
├───────────────────────────────┼────────────────────┼───────────┼──────────────────────────┤
│ conv2d_49 (Conv2D)            │ (None, 7, 7, 2048) │ 1,050,624 │ re_lu_44[0][0]           │
├───────────────────────────────┼────────────────────┼───────────┼──────────────────────────┤
│ batch_normalization_49        │ (None, 7, 7, 2048) │     8,192 │ conv2d_49[0][0]          │
│ (BatchNormalization)          │                    │           │                          │
├───────────────────────────────┼────────────────────┼───────────┼──────────────────────────┤
│ add_14 (Add)                  │ (None, 7, 7, 2048) │         0 │ batch_normalization_4…   │
│                               │                    │           │ re_lu_42[0][0]           │
├───────────────────────────────┼────────────────────┼───────────┼──────────────────────────┤
│ re_lu_45 (ReLU)               │ (None, 7, 7, 2048) │         0 │ add_14[0][0]             │
├───────────────────────────────┼────────────────────┼───────────┼──────────────────────────┤
│ conv2d_50 (Conv2D)            │ (None, 7, 7, 512)  │ 1,049,088 │ re_lu_45[0][0]           │
├───────────────────────────────┼────────────────────┼───────────┼──────────────────────────┤
│ batch_normalization_50        │ (None, 7, 7, 512)  │     2,048 │ conv2d_50[0][0]          │
│ (BatchNormalization)          │                    │           │                          │
├───────────────────────────────┼────────────────────┼───────────┼──────────────────────────┤
│ re_lu_46 (ReLU)               │ (None, 7, 7, 512)  │         0 │ batch_normalization_5…   │
├───────────────────────────────┼────────────────────┼───────────┼──────────────────────────┤
│ conv2d_51 (Conv2D)            │ (None, 7, 7, 512)  │ 2,359,808 │ re_lu_46[0][0]           │
├───────────────────────────────┼────────────────────┼───────────┼──────────────────────────┤
│ batch_normalization_51        │ (None, 7, 7, 512)  │     2,048 │ conv2d_51[0][0]          │
│ (BatchNormalization)          │                    │           │                          │
├───────────────────────────────┼────────────────────┼───────────┼──────────────────────────┤
│ re_lu_47 (ReLU)               │ (None, 7, 7, 512)  │         0 │ batch_normalization_5…   │
├───────────────────────────────┼────────────────────┼───────────┼──────────────────────────┤
│ conv2d_52 (Conv2D)            │ (None, 7, 7, 2048) │ 1,050,624 │ re_lu_47[0][0]           │
├───────────────────────────────┼────────────────────┼───────────┼──────────────────────────┤
│ batch_normalization_52        │ (None, 7, 7, 2048) │     8,192 │ conv2d_52[0][0]          │
│ (BatchNormalization)          │                    │           │                          │
├───────────────────────────────┼────────────────────┼───────────┼──────────────────────────┤
│ add_15 (Add)                  │ (None, 7, 7, 2048) │         0 │ batch_normalization_5…   │
│                               │                    │           │ re_lu_45[0][0]           │
├───────────────────────────────┼────────────────────┼───────────┼──────────────────────────┤
│ re_lu_48 (ReLU)               │ (None, 7, 7, 2048) │         0 │ add_15[0][0]             │
├───────────────────────────────┼────────────────────┼───────────┼──────────────────────────┤
│ global_average_pooling2d      │ (None, 2048)       │         0 │ re_lu_48[0][0]           │
│ (GlobalAveragePooling2D)      │                    │           │                          │
├───────────────────────────────┼────────────────────┼───────────┼──────────────────────────┤
│ dense (Dense)                 │ (None, 10)         │    20,490 │ global_average_poolin…   │
└───────────────────────────────┴────────────────────┴───────────┴──────────────────────────┘
 Total params: 23,608,202 (90.06 MB)
 Trainable params: 23,555,082 (89.86 MB)
 Non-trainable params: 53,120 (207.50 KB)
```

This code outlines the process of training a custom ResNet-50 model using TensorFlow and Keras for image classification. The dataset is loaded using image_dataset_from_directory for both training and testing, with images resized to 224x224 pixels and labels encoded in a categorical format. Data augmentation is applied to the training data through random transformations like flipping, rotation, and zooming to improve generalization. The custom ResNet-50 model, defined earlier, is compiled with the Adam optimizer and categorical cross-entropy loss for multi-class classification. The model is then trained for 10 epochs, with training and validation steps specified, followed by evaluation on the test data. Finally, the trained model is saved to a file for later use.

In [23]:

```python
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing import image_dataset_from_directory
from tensorflow.keras.layers import Conv2D, BatchNormalization, ReLU, Add, MaxPooling2D, GlobalAveragePooling2D
from tensorflow.keras.models import Model

# Assuming the custom ResNet50 function is already defined
# Import or define it here if needed

# Paths to your data directories
train_path = "/content/extracted_folder/dataset/Dataset/train"
test_path = "/content/extracted_folder/dataset/Dataset/val"

# Parameters
BATCH_SIZE = 32
IMG_SIZE = (224,224)
IMG_HEIGHT = 224
IMG_WIDTH = 224
NUM_CLASSES = 20  # Change this based on your dataset

# Data Augmentation and Preprocessing
train_datagen = tf.keras.Sequential([
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(0.2),
    layers.RandomZoom(0.2),
    layers.RandomHeight(0.2),
    layers.RandomWidth(0.2),
], name="data_augmentation")


# Load and preprocess the datasets
train_data = image_dataset_from_directory(
    directory=train_path,
    image_size=IMG_SIZE,
    label_mode="categorical",  # Ensure labels are one-hot encoded
    batch_size=BATCH_SIZE
)

test_data = image_dataset_from_directory(
    directory=test_path,
    image_size=IMG_SIZE,
    label_mode="categorical",
    batch_size=BATCH_SIZE
)

# Build the Custom ResNet Model
model = ResNet50(input_shape=(IMG_HEIGHT, IMG_WIDTH, 3), num_classes=NUM_CLASSES)

# Compile the Model
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# Train the Model
history = model.fit(
    train_data,
    validation_data=test_data,
    epochs=10,  # Adjust the number of epochs
    steps_per_epoch=len(train_data),
    validation_steps=len(test_data)
)

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

# Save the Model
model.save("custom_resnet_model.h5")
```

```
Found 3996 files belonging to 20 classes.
Found 1250 files belonging to 20 classes.
Epoch 1/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 158s 790ms/step - accuracy: 0.1292 - loss: 3.7093 - val_accuracy: 0.0512 - val_loss
: 308.3112
Epoch 2/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 578us/step - accuracy: 0.0000e+00 - loss: 0.0000e+00
Epoch 3/10
/usr/lib/python3.10/contextlib.py:153: UserWarning: Your input ran out of data; interrupting training. Make sure
that your dataset or generator can generate at least `steps_per_epoch * epochs` batches. You may need to use the
`.repeat()` function when building your dataset.
  self.gen.throw(typ, value, traceback)
125/125 ━━━━━━━━━━━━━━━━━━━━ 68s 541ms/step - accuracy: 0.1771 - loss: 2.7523 - val_accuracy: 0.1096 - val_loss:
5.9696
Epoch 4/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 169us/step - accuracy: 0.0000e+00 - loss: 0.0000e+00
Epoch 5/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 80s 525ms/step - accuracy: 0.2815 - loss: 2.3676 - val_accuracy: 0.1456 - val_loss:
3.3234
Epoch 6/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 6s 47ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00
Epoch 7/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 65s 522ms/step - accuracy: 0.3419 - loss: 2.1511 - val_accuracy: 0.1184 - val_loss:
3.7253
Epoch 8/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 6s 47ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00
Epoch 9/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 77s 526ms/step - accuracy: 0.3554 - loss: 2.1524 - val_accuracy: 0.0912 - val_loss:
5.1093
Epoch 10/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 6s 47ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00
40/40 ━━━━━━━━━━━━━━━━━━━━ 15s 369ms/step - accuracy: 0.0851 - loss: 5.0821
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. T
his file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_m
odel.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
Validation Loss: 5.109333515167236, Validation Accuracy: 0.09120000153779984
```

In [24]:
```
train_data.class_names
```

Out[24]:
```
['burger',
 'butter_naan',
 'chai',
 'chapati',
 'chole_bhature',
 'dal_makhani',
 'dhokla',
 'fried_rice',
 'idli',
 'jalebi',
 'kaathi_rolls',
 'kadai_paneer',
 'kulfi',
 'masala_dosa',
 'momos',
 'paani_puri',
 'pakode',
 'pav_bhaji',
 'pizza',
 'samosa']
```

This code demonstrates how to load, process, and augment images from a training dataset for a machine learning model. The process_and_augment_image function reads an image from the file system, resizes it to the target size of 224x224 pixels, and applies a series of data augmentation techniques, such as horizontal flipping, rotation, zoom, and changes in height and width. The code randomly selects three images from different classes in the training dataset, processes them, applies augmentation, and then displays the original and augmented images side by side for visual inspection. The augmentation helps to increase the variability of the dataset, which can improve the model's ability to generalize by exposing it to different transformations of the data.

Data augmentation plays a crucial role in improving the performance of deep learning models by artificially expanding the training dataset. This process helps the model become more robust by simulating different conditions that the model might encounter in real-world scenarios, such as variations in orientation, scale, or position. The augmented images provide additional diverse examples without the need for more raw data, which is particularly valuable when the dataset is limited. By incorporating data augmentation, the model is less likely to overfit and can generalize better to unseen data, leading to improved accuracy and robustness during training and evaluation.

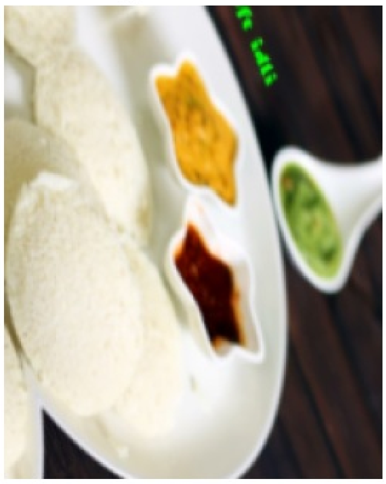
Data Augmentation

In [25]:
```
import os
import random
import matplotlib.pyplot as plt
```

```python
import matplotlib.image as mpimg
import tensorflow as tf
from tensorflow.keras import layers

# Define the train directory
train_dir = "/content/extracted_folder/dataset/Dataset/train"

# Define the data augmentation pipeline
data_augmentation = tf.keras.Sequential([
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(0.2),
    layers.RandomZoom(0.2),
    layers.RandomHeight(0.2),
    layers.RandomWidth(0.2),
], name="data_augmentation")

# Get the class names from the training directory
class_names = os.listdir(train_dir)

# Function to process and augment a single image
def process_and_augment_image(image_path):
    img = tf.io.read_file(image_path)
    img = tf.image.decode_image(img, channels=3)
    img = tf.image.resize(img, (224, 224))  # Resize to the expected size for augmentation
    img = tf.expand_dims(img, axis=0)  # Add batch dimension
    augmented_img = data_augmentation(img, training=True)  # Apply data augmentation
    return img[0].numpy().astype("uint8"), augmented_img[0].numpy().astype("uint8")

# Select three random images
for _ in range(3):
    target_class = random.choice(class_names)  # Choose a random class
    target_dir = os.path.join(train_dir, target_class)  # Create the target directory path

    # Ensure the target directory exists
    if not os.path.exists(target_dir):
        raise FileNotFoundError(f"Target directory {target_dir} does not exist.")

    # Choose a random image from the target directory
    random_image = random.choice(os.listdir(target_dir))  # Choose a random image
    random_image_path = os.path.join(target_dir, random_image)  # Get the full path

    # Process and augment the image
    original_img, augmented_img = process_and_augment_image(random_image_path)

    # Plot the original and augmented images
    plt.figure(figsize=(8, 4))

    # Original image
    plt.subplot(1, 2, 1)
    plt.imshow(original_img)
    plt.title(f"Original: {target_class}")
    plt.axis("off")

    # Augmented image
    plt.subplot(1, 2, 2)
    plt.imshow(augmented_img)
    plt.title(f"Augmented: {target_class}")
    plt.axis("off")

    plt.show()
```

Original: idli

Augmented: idli



Original: butter_naan

Augmented: butter_naan

Original: momos

Augmented: momos

In [27]:
```
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.applications import EfficientNetV2B0

# Define the input shape
IMG_SIZE = (224, 224, 3)
NUM_CLASSES = 20  # Example number of classes

# Create the base model from EfficientNetV2B0
base_model = EfficientNetV2B0(include_top=False, input_shape=IMG_SIZE, weights="imagenet")
base_model.trainable = False  # Freeze the base model

# Create the input layer
inputs = tf.keras.Input(shape=IMG_SIZE)

# Pass inputs through the base model
x = base_model(inputs)

# Ensure x has the correct 4D shape
print(f"Shape after base model: {x.shape}")  # Expect (None, H, W, C)

# Apply Global Average Pooling
x = layers.GlobalAveragePooling2D(name="global_average_pooling_layer")(x)

# Add a dense output layer
outputs = layers.Dense(NUM_CLASSES, activation="softmax", name="output_layer")(x)

# Create the model
model = tf.keras.Model(inputs, outputs, name="EfficientNetV2B0_Model")

# Compile the model
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
    loss="categorical_crossentropy",
    metrics=["accuracy"]
)

# Display the model summary
model.summary()
```

```
Shape after base model: (None, 7, 7, 1280)
Model: "EfficientNetV2B0_Model"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃         Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ input_layer_6 (InputLayer)           │ (None, 224, 224, 3)         │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ efficientnetv2-b0 (Functional)       │ (None, 7, 7, 1280)          │       5,919,312 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ global_average_pooling_layer         │ (None, 1280)                │               0 │
│ (GlobalAveragePooling2D)             │                             │                 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ output_layer (Dense)                 │ (None, 20)                  │          25,620 │
└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘

 Total params: 5,944,932 (22.68 MB)
 Trainable params: 25,620 (100.08 KB)
 Non-trainable params: 5,919,312 (22.58 MB)
```

In [27]:

In [2]:

In [1]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: