```
!unzip '/content/drive/MyDrive/SVM/dogs-vs-cats.zip'
```

```
Archive:  /content/drive/MyDrive/SVM/dogs-vs-cats.zip
  inflating: sampleSubmission.csv
  inflating: test1.zip
  inflating: train.zip
```

```
!pip install albumentations
```

```
Requirement already satisfied: albumentations in /usr/local/lib/python3.10/dist-packages (1.3.1)
Requirement already satisfied: numpy>=1.11.1 in /usr/local/lib/python3.10/dist-packages (from albumentations) (1.25.2)
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from albumentations) (1.11.4)
Requirement already satisfied: scikit-image>=0.16.1 in /usr/local/lib/python3.10/dist-packages (from albumentations) (0.19.3)
Requirement already satisfied: PyYAML in /usr/local/lib/python3.10/dist-packages (from albumentations) (6.0.1)
Requirement already satisfied: qudida>=0.0.4 in /usr/local/lib/python3.10/dist-packages (from albumentations) (0.0.4)
Requirement already satisfied: opencv-python-headless>=4.1.1 in /usr/local/lib/python3.10/dist-packages (from albumentations) (4.9.
Requirement already satisfied: scikit-learn>=0.19.1 in /usr/local/lib/python3.10/dist-packages (from qudida>=0.0.4->albumentations)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from qudida>=0.0.4->albumentations) (4
Requirement already satisfied: networkx>=2.2 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.16.1->albumentations)
Requirement already satisfied: pillow!=7.1.0,!=7.1.1,!=8.3.0,>=6.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-image>
Requirement already satisfied: imageio>=2.4.1 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.16.1->albumentations
Requirement already satisfied: tifffile>=2019.7.26 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.16.1->albumenta
Requirement already satisfied: PyWavelets>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.16.1->albumentati
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.16.1->albumentation
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.19.1->qudida>=0.0.4->
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.19.1->qudida>=
```

```
# Disable warnings in the notebook to maintain clean output cells
import warnings  # Import the warnings module to handle warnings
warnings.filterwarnings('ignore')  # Filter and ignore warnings to maintain clean output

# Import necessary libraries
import numpy as np  # Import numpy library for numerical operations
import pandas as pd  # Import pandas library for data manipulation and analysis
import matplotlib.pyplot as plt  # Import matplotlib library for data visualization
import seaborn as sns  # Import seaborn library for statistical data visualization
import os  # Import os module for interacting with the operating system
import cv2  # Import OpenCV library for computer vision tasks
import zipfile  # Import zipfile module for working with zip files
import random  # Import random module for generating random numbers and sequences
import albumentations  # Import albumentations library for image augmentation
import subprocess  # Import subprocess module for running shell commands
from sklearn.model_selection import train_test_split  # Import train_test_split function from sklearn for splitting dataset
from keras.preprocessing.image import ImageDataGenerator  # Import ImageDataGenerator from keras for image data augmentation
from tensorflow.keras.applications import ResNet50V2  # Import ResNet50V2 model from keras for transfer learning
from tensorflow.keras.applications.inception_v3 import InceptionV3  # Import InceptionV3 model from keras for transfer learning
from tensorflow.keras.applications import MobileNetV2  # Import MobileNetV2 model from keras for transfer learning
from tensorflow.keras.models import Model  # Import Model class from keras for defining custom neural network architectures
from tensorflow.keras.layers import Dense, Dropout, GlobalAveragePooling2D, Input  # Import layers from keras for building neural netwo
from tensorflow.keras.optimizers import Adam  # Import Adam optimizer from keras for model optimization
from keras.utils import plot_model  # Import plot_model function from keras for visualizing model architectures
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping  # Import callbacks from keras for adjusting learning rate and
from tensorflow.keras.applications.resnet_v2 import preprocess_input as resnet_preprocess_input  # Import preprocess_input function fro
from tensorflow.keras.applications.inception_v3 import preprocess_input as inceptionv3_preprocess_input  # Import preprocess_input func
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input as mobilenetv2_preprocess_input  # Import preprocess_input func
from tensorflow.keras.models import load_model  # Import load_model function from keras for loading pre-trained models
from sklearn.svm import SVC  # Import SVC (Support Vector Classifier) from sklearn for machine learning classification
from sklearn.decomposition import PCA  # Import PCA (Principal Component Analysis) from sklearn for dimensionality reduction
from sklearn.metrics import accuracy_score  # Import accuracy_score function from sklearn for evaluating classification accuracy
from joblib import dump  # Import dump function from joblib for saving trained models
from IPython.display import FileLink, display  # Import FileLink and display functions from IPython.display for displaying files
```

```
# Configure the visual appearance of Seaborn plots
sns.set(rc={'axes.facecolor': 'gainsboro'}, style='darkgrid')
```

```
# Specify the path to the zip file
zip_path = '/content/train.zip'

# Specify the directory where we want to extract the contents of the zip file
extract_path = '/kaggle/working/'

# Extract the zip file
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)

# After extracting, specify the directory of the actual images
actual_extracted_path = os.path.join(extract_path, 'train')

# List all filenames in the actual extracted directory
filenames = os.listdir(actual_extracted_path)

# Separate filenames into cats and dogs
cat_images = [filename for filename in filenames if "cat" in filename]
dog_images = [filename for filename in filenames if "dog" in filename]

# Get the count of each class
cat_count = len(cat_images)
dog_count = len(dog_images)

# Calculate the percentages
total_images = cat_count + dog_count
cat_percentage = (cat_count / total_images) * 100
dog_percentage = (dog_count / total_images) * 100

# Plotting
labels = ['Cats', 'Dogs']
counts = [cat_count, dog_count]
percentages = [cat_percentage, dog_percentage]

# Set the figure size
plt.figure(figsize=(15, 4))

# Create a horizontal bar plot
ax = sns.barplot(y=labels, x=counts, orient='h', color='#33312b')

# Set x-axis interval
ax.set_xticks([0, 15000])

# Annotate each bar with the count and percentage
for i, p in enumerate(ax.patches):
    width = p.get_width()
    ax.text(width + 100, p.get_y() + p.get_height()/2.,
            '{:1.2f}% ({})'.format(percentages[i], counts[i]),
            va="center", fontsize=15)

# Set the x-label for the plot
plt.xlabel('Number of Images', fontsize=14)

# Set the title and show the plot
plt.title("Number of images per class", fontsize=18)
plt.show()
```


```
# Lists to store heights and widths of all images
heights = []
widths = []

# Initialize a set to store unique dimensions and channels
```

```
unique_dims = set()
unique_channels = set()

# Loop over filenames and check dimensions
for filename in filenames:
    img = cv2.imread(os.path.join(actual_extracted_path, filename))
    if img is not None:
        # Add the dimensions (height, width, channels) to the set
        unique_dims.add((img.shape[0], img.shape[1]))

        # Add the channels to the set
        unique_channels.add(img.shape[2])

        # Append heights and widths for statistical calculations
        heights.append(img.shape[0])
        widths.append(img.shape[1])

# Check if all images have the same dimension
if len(unique_dims) == 1:
    print(f"All images have the same dimensions: {list(unique_dims)[0]}")
else:
    print(f"There are {len(unique_dims)} different image dimensions in the dataset.")
    print(f"Min height: {min(heights)}, Max height: {max(heights)}, Mean height: {np.mean(heights):.2f}")
    print(f"Min width: {min(widths)}, Max width: {max(widths)}, Mean width: {np.mean(widths):.2f}")

# Check if all images have the same number of channels
if len(unique_channels) == 1:
    channel = list(unique_channels)[0]
    if channel == 3:
        print("All images are color images.")
    else:
        print("All images have the same number of channels, but they are not color images.")
else:
    print("Images have different numbers of channels.")
```

```
There are 8513 different image dimensions in the dataset.
Min height: 32, Max height: 768, Mean height: 360.48
Min width: 42, Max width: 1050, Mean width: 404.10
All images are color images.
```

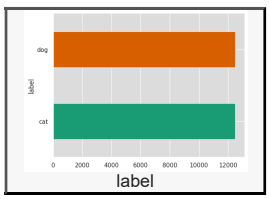
```
# Setting the random seed for reproducibility
np.random.seed(42)  # Set the random seed to ensure reproducibility of results

# Randomly select 6 images from each category
random_cat_images = np.random.choice(cat_images, 6)  # Randomly select 6 cat images
random_dog_images = np.random.choice(dog_images, 6)  # Randomly select 6 dog images

# Function to plot images
def plot_images(images, title):
    plt.figure(figsize=(14, 3))  # Create a new figure with a specified size
    for i, img_name in enumerate(images):  # Iterate over the selected images
        plt.subplot(1, 6, i+1)  # Create subplots in a row
        img = cv2.imread(os.path.join(actual_extracted_path, img_name))  # Read the image
        # Convert the BGR image (default in OpenCV) to RGB
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        plt.imshow(img)  # Display the image
        plt.axis('off')  # Turn off axis labels
    plt.suptitle(title, fontsize=20)  # Add a centered title to the plot
    plt.show()  # Show the plot

# Plot the images
plot_images(random_cat_images, "Randomly Selected Cat Images")  # Plot randomly selected cat images
plot_images(random_dog_images, "Randomly Selected Dog Images")  # Plot randomly selected dog images
```

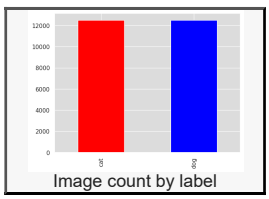

```
# Deleting unnecessary variables to free up memory
del filenames, heights, widths, unique_dims, unique_channels
```

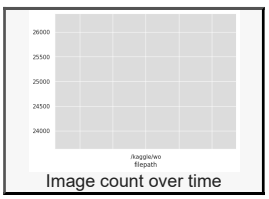
```
# Initialize an empty list to store image file paths and their respective labels
data = []

# Append the cat image file paths with label "cat" to the data list
data.extend([(os.path.join(actual_extracted_path, filename), "cat") for filename in cat_images])

# Append the dog image file paths with label "dog" to the data list
data.extend([(os.path.join(actual_extracted_path, filename), "dog") for filename in dog_images])

# Convert the collected data into a DataFrame
df = pd.DataFrame(data, columns=['filepath', 'label'])

# Display the first few entries of the DataFrame
df.head()
```

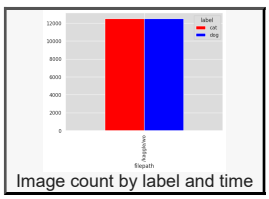
| | filepath | label |
|---|---|---|
| 0 | /kaggle/working/train/cat.9888.jpg | cat |
| 1 | /kaggle/working/train/cat.3282.jpg | cat |
| 2 | /kaggle/working/train/cat.3912.jpg | cat |
| 3 | /kaggle/working/train/cat.5854.jpg | cat |
| 4 | /kaggle/working/train/cat.7369.jpg | cat |

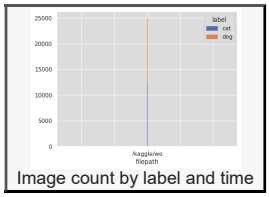
Next steps: Generate code with df View recommended plots




```
# Deleting unnecessary variables to free up memory
del data, cat_images, dog_images
```

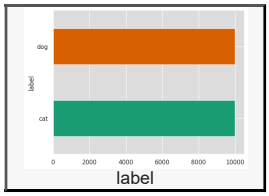
```
# Split the data into training and validation sets
train_df, val_df = train_test_split(df, test_size=0.2, stratify=df['label'], random_state=42)

# Display the shape of the training and validation sets
print("Training data shape:", train_df.shape)
print("Validation data shape:", val_df.shape)

# Deleting the original DataFrame to free up memory
del df
```

```
Training data shape: (20000, 2)
Validation data shape: (5000, 2)
```

```
# Display the first few rows of the train DataFrame
train_df.head()
```

| | filepath | label |
|---|---|---|
| **20022** | /kaggle/working/train/dog.5324.jpg | dog |
| **4993** | /kaggle/working/train/cat.2569.jpg | cat |
| **24760** | /kaggle/working/train/dog.7079.jpg | dog |
| **13775** | /kaggle/working/train/dog.10092.jpg | dog |
| **20504** | /kaggle/working/train/dog.11030.jpg | dog |

Next steps: Generate code with `train_df` View recommended plots

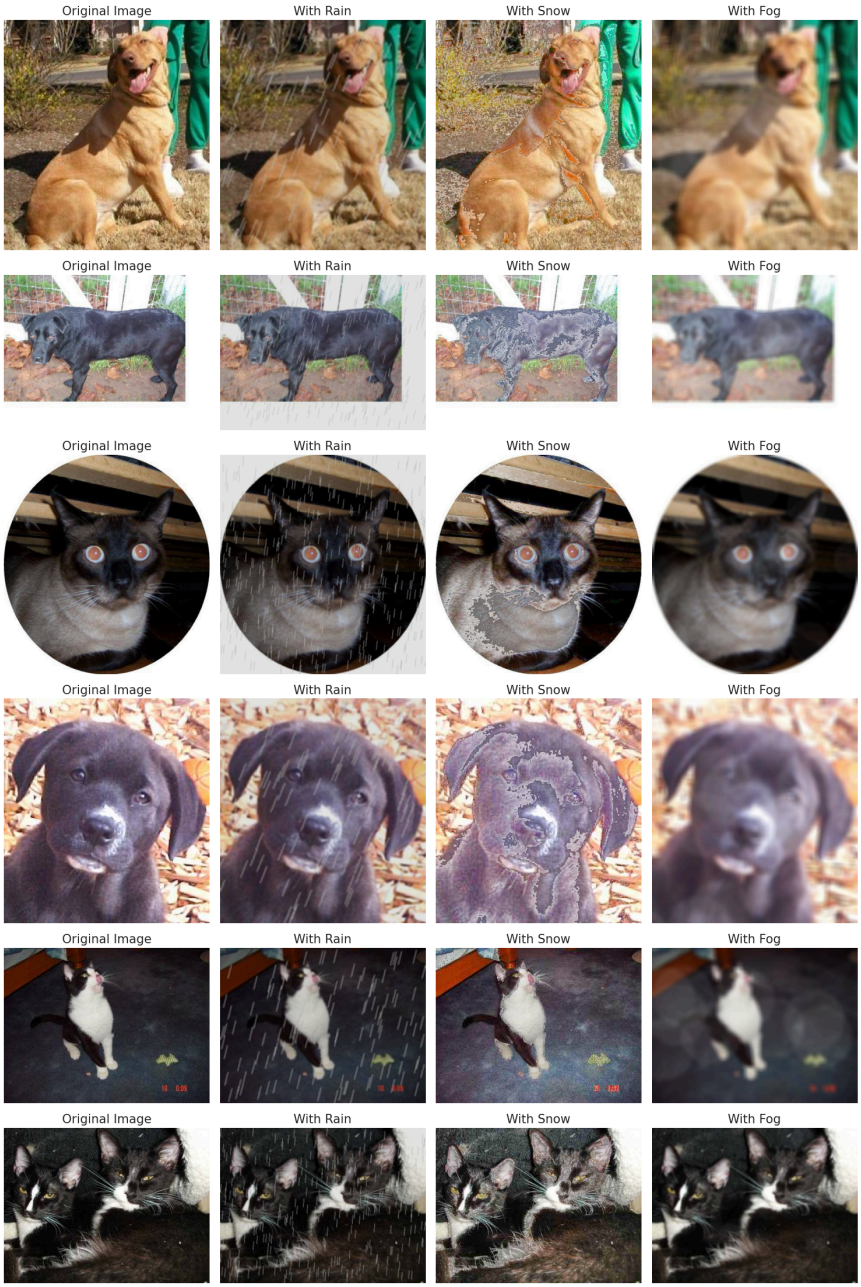



```
def add_rain(image,
             slant_lower=-10,            # Lower bound for rain slant
             slant_upper=10,             # Upper bound for rain slant
             drop_length=20,             # Length of the raindrops
             drop_width=1,               # Width of the raindrops
             drop_color=(200, 200, 200), # Color of the raindrops
             blur_value=3,               # Amount of blur added due to rain
             brightness_coefficient=0.9, # Adjust brightness of image
             rain_type=None):            # Type of rain (drizzle or None for regular)

    """Add rain effect to the input image using specified parameters."""

    # Create the rain augmentation using albumentations
    rain_aug = albumentations.RandomRain(
                                       slant_lower=slant_lower,
                                       slant_upper=slant_upper,
                                       drop_length=drop_length,
                                       drop_width=drop_width,
                                       drop_color=drop_color,
                                       blur_value=blur_value,
                                       brightness_coefficient=brightness_coefficient,
                                       rain_type=rain_type,
                                       p=1  # Probability of augmentation. Set to 1 for always applying
    )

    # Apply the rain augmentation to the image
    augmented = rain_aug(image=image)

    return augmented['image']
```

```
def add_snow(image,
             snow_point_lower=0.1,          # Minimum intensity of snow
             snow_point_upper=0.2,          # Maximum intensity of snow
             brightness_coeff=2.0):         # Brightness coefficient for the image post snow effect

    """Add snow effect to the input image using specified parameters."""

    # Define the snow augmentation from albumentations with provided parameters
    snow_aug = albumentations.RandomSnow(snow_point_lower=snow_point_lower,
                                         snow_point_upper=snow_point_upper,
                                         brightness_coeff=brightness_coeff,
                                         p=1  # Probability of augmentation. Set to 1 for always applying
                                         )

    # Apply the snow augmentation to the image
    augmented = snow_aug(image=image)

    return augmented['image']
```

```
def add_fog(image,
            fog_coef_lower=0.3,           # Minimum intensity of fog
            fog_coef_upper=1,             # Maximum intensity of fog
            alpha_coef=0.08):             # Transparency of the fog

    """Add fog effect to the input image using specified parameters."""

    # Create the fog augmentation using albumentations
    fog_aug = albumentations.RandomFog(fog_coef_lower=fog_coef_lower,
                                       fog_coef_upper=fog_coef_upper,
                                       alpha_coef=alpha_coef,
                                       p=1  # Probability of augmentation. Set to 1 for always applying
                                       )

    # Apply the fog augmentation to the image
    augmented = fog_aug(image=image)

    return augmented['image']
```

```
# Setting the random seed for reproducibility
np.random.seed(42)  # Set the random seed to ensure reproducibility of results

# Select 6 random images from the train_df
random_image_paths = random.sample(train_df['filepath'].tolist(), 6)  # Randomly select 6 image paths from the DataFrame

# For each image, display original and the ones with effects
for image_path in random_image_paths:  # Iterate over the selected image paths
    # Read the image
    image = cv2.imread(image_path)  # Read the image using OpenCV
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  # Convert the image from BGR to RGB color space

    # Apply augmentations
    image_rain = add_rain(image)  # Add rain effect to the image
    image_snow = add_snow(image)  # Add snow effect to the image
    image_fog = add_fog(image)    # Add fog effect to the image

    # Display images
    fig, axes = plt.subplots(1, 4, figsize=(15, 5))  # Create a figure with subplots

    axes[0].imshow(image)  # Display the original image
    axes[0].set_title("Original Image", fontsize=15)  # Set the title for the subplot
    axes[0].axis("off")  # Turn off axis labels

    axes[1].imshow(image_rain)  # Display the image with rain effect
    axes[1].set_title("With Rain", fontsize=15)  # Set the title for the subplot
    axes[1].axis("off")  # Turn off axis labels

    axes[2].imshow(image_snow)  # Display the image with snow effect
    axes[2].set_title("With Snow", fontsize=15)  # Set the title for the subplot
    axes[2].axis("off")  # Turn off axis labels

    axes[3].imshow(image_fog)  # Display the image with fog effect
    axes[3].set_title("With Fog", fontsize=15)  # Set the title for the subplot
    axes[3].axis("off")  # Turn off axis labels

    plt.tight_layout()  # Adjust the layout of the subplots
    plt.show()  # Show the plot
```


Original Image
With Rain
With Snow
With Fog
Original Image
With Rain
With Snow
With Fog
Original Image
With Rain
With Snow
With Fog
Original Image
With Rain
With Snow
With Fog
Original Image
With Rain
With Snow
With Fog
Original Image
With Rain
With Snow
With Fog

```
def random_weather_effect(img):
    """
    Apply a random weather effect (rain, snow, fog) or no effect to an image.

    Parameters:
    - img: Input image to be augmented.

    Returns:
    - Augmented image with a random weather effect or the original image.
    """

    # List of possible weather effects including no effect (lambda x: x)
    effects = [add_snow, add_rain, add_fog, lambda x: x]

    # Randomly choose one effect from the list
    effect = random.choice(effects)

    # Apply the chosen effect and return the result
    return effect(img)
```

```
def apply_effects_and_save(image_path):
    """
    Apply a random weather effect to an image at a given path and overwrite the original image.

    This function reads an image from a specified file path using OpenCV, applies one of several
    weather effects (snow, rain, fog, or none), and saves the altered image back to the same path,
    effectively overwriting the original image.

    Parameters:
    - image_path (str): The file path to the image that will be augmented with a weather effect.

    Raises:
    - ValueError: If the image at the given path cannot be opened or processed.
    """

    # Read the image file using OpenCV
    img = cv2.imread(image_path)

    # Check if the image was correctly opened
    if img is None:
        raise ValueError(f"Image at path {image_path} could not be opened.")

    # Convert from BGR to RGB since OpenCV uses BGR by default
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    # Apply random weather effect
    augmented_img = random_weather_effect(img)

    # Convert back from RGB to BGR before saving with OpenCV
    augmented_img = cv2.cvtColor(augmented_img, cv2.COLOR_RGB2BGR)

    # Save the augmented image back to the original path
    cv2.imwrite(image_path, augmented_img)
```

```
# Iterate over the images in the train set DataFrame
for index, row in train_df.iterrows():
    image_path = row['filepath']
    # Check if the image file exists
    if os.path.exists(image_path):
        try:
            apply_effects_and_save(image_path)  # Apply random weather effect and save the image
        except ValueError as e:
            print(e)  # Print any errors encountered during the process
    else:
        print(f"Image not found at path: {image_path}")  # Print a message if the image file is not found
```

```
def display_random_images(image_paths, labels, num_images=16):
    """
    Display a random set of images in a grid.

    Parameters:
    - image_paths (list): List of filepaths to images.
    - labels (list): Corresponding labels for the images.
    - num_images (int): Total number of images to display.
    """
    # Select a random subset of image paths and their labels
    random_indices = np.random.choice(len(image_paths), num_images, replace=False)
    selected_image_paths = [image_paths[i] for i in random_indices]
    selected_labels = [labels[i] for i in random_indices]

    # Calculate number of rows and columns for the subplot grid
    num_rows = num_images // 4
    num_cols = 4

    # Create the subplots
    fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 15))
    axes = axes.ravel()

    # Read and display each image
    for i, (path, label, ax) in enumerate(zip(selected_image_paths, selected_labels, axes)):
        # Read the image using OpenCV
        img = cv2.imread(path)
        # Convert BGR image to RGB
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        ax.imshow(img)
        ax.set_title(label, fontsize=15)
        ax.axis('off')

    plt.tight_layout()
    plt.show()
```

```
# Display the images
display_random_images(train_df['filepath'].tolist(), train_df['label'].tolist())
```


```
def create_data_generators(preprocessing_function=None, batch_size=32, image_dimensions=(224, 224)):
    """
    Creates and returns training and validation data generators with optional weather effects and preprocessing.

    Parameters:
    - preprocessing_function (function, optional): Preprocessing function specific to a model. Defaults to None.
    - batch_size (int, optional): Number of images per batch for the generators. Defaults to 32.
    - image_dimensions (tuple, optional): Dimensions to which the images will be resized (height, width). Defaults to (224, 224).

    Returns:
    - train_generator (ImageDataGenerator): Generator for training data with augmentations.
    - val_generator (ImageDataGenerator): Generator for validation data without augmentations.

    Notes:
    - The training generator uses augmentations.
    - The validation generator does not use any augmentations.
    - If provided, the preprocessing function is applied to both generators.
    """

    # Define our training data generator with specific augmentations
    train_datagen = ImageDataGenerator(
        rotation_range=15,                          # Randomly rotate the images by up to 15 degrees
```

```
        width_shift_range=0.15,                       # Randomly shift images horizontally by up to 15% of the width
        height_shift_range=0.15,                      # Randomly shift images vertically by up to 15% of the height
        zoom_range=0.15,                              # Randomly zoom in or out by up to 15%
        horizontal_flip=True,                         # Randomly flip images horizontally
        vertical_flip=False,                          # Do not flip images vertically as it doesn't make sense in our context
        shear_range=0.02,                             # Apply slight shear transformations
        preprocessing_function=preprocessing_function  # Apply preprocessing function if provided
    )

    # Define our validation data generator without any augmentations but with the preprocessing function if provided
    val_datagen = ImageDataGenerator(
        preprocessing_function=preprocessing_function  # Apply preprocessing function if provided
    )

    # Create an iterable generator for training data
    train_generator = train_datagen.flow_from_dataframe(
        dataframe=train_df,                 # DataFrame containing training data
        x_col="filepath",                   # Column with paths to image files
        y_col="label",                      # Column with image labels
        target_size=image_dimensions,       # Resize all images to size of 224x224
        batch_size=batch_size,              # Number of images per batch
        class_mode='binary',                # Specify binary classification task
        seed=42,                            # Seed for random number generator to ensure reproducibility
        shuffle=True                        # Shuffle the data to ensure the model gets a randomized batch during training
    )

    # Create an iterable generator for validation data
    val_generator = val_datagen.flow_from_dataframe(
        dataframe=val_df,                   # DataFrame containing validation data
        x_col="filepath",
        y_col="label",
        target_size=image_dimensions,
        batch_size=batch_size,
        class_mode='binary',
        seed=42,
        shuffle=False                       # Shuffling not necessary for validation data
    )

    # Return the training and validation generators
    return train_generator, val_generator
```

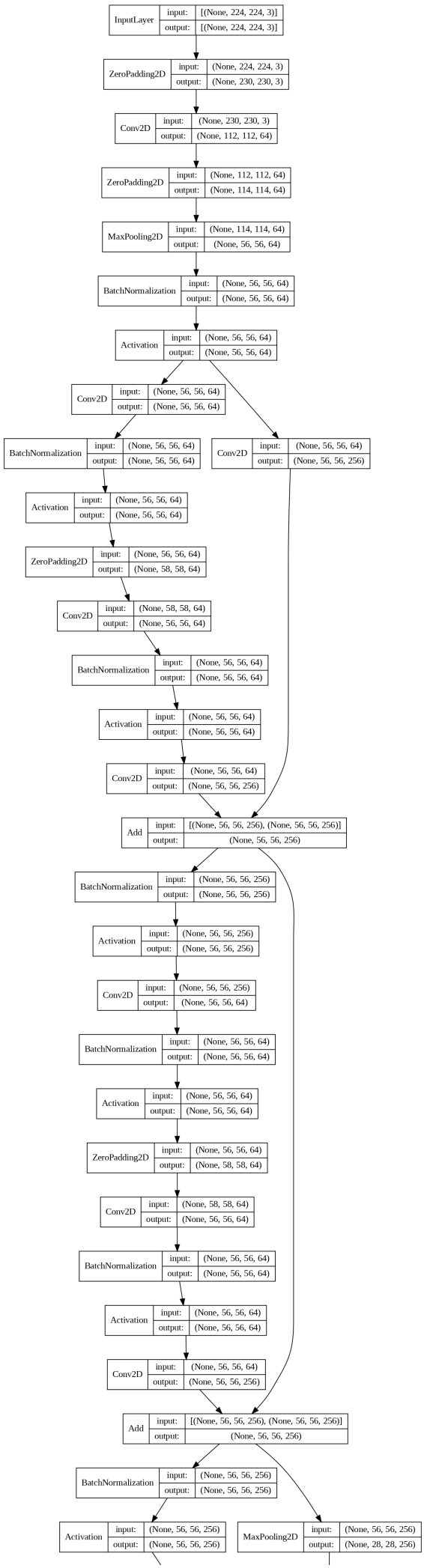
```
# Load the ResNet50V2 model pre-trained on ImageNet data, excluding the top classifier
resnet50v2_base = ResNet50V2(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
resnet50v2_base.summary()
```

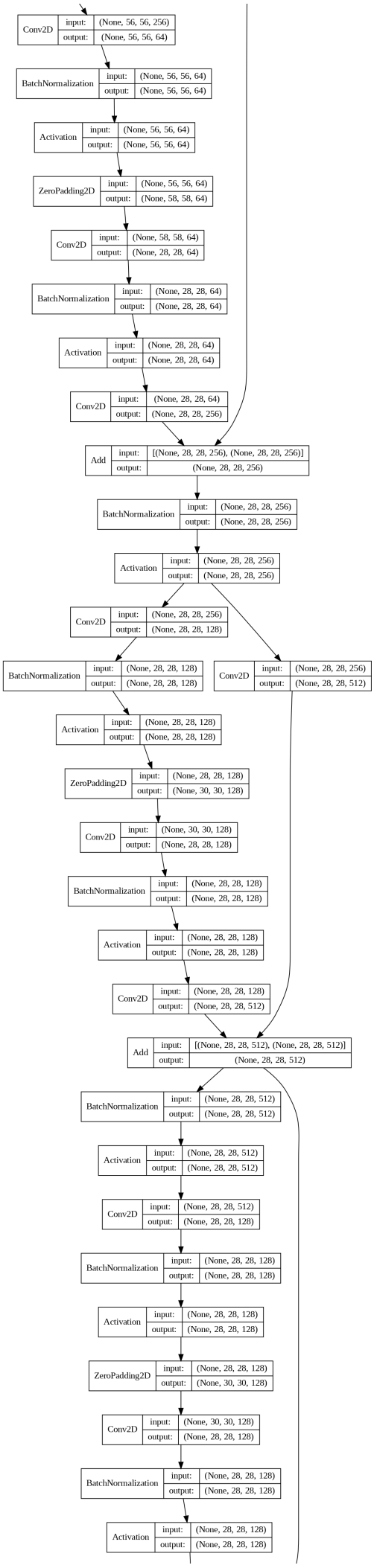
```
 conv5_block3_3_conv (Conv2  (None, 7, 7, 2048)           1050624   ['conv5_block3_2_relu[0][0]']
 D)

 conv5_block3_out (Add)      (None, 7, 7, 2048)           0         ['conv5_block2_out[0][0]',
                                                                     'conv5_block3_3_conv[0][0]']

 post_bn (BatchNormalizatio  (None, 7, 7, 2048)           8192      ['conv5_block3_out[0][0]']
 n)

 post_relu (Activation)      (None, 7, 7, 2048)           0         ['post_bn[0][0]']

==================================================================================================
Total params: 23564800 (89.89 MB)
Trainable params: 23519360 (89.72 MB)
Non-trainable params: 45440 (177.50 KB)
__________________________________________________________________________________________________
```

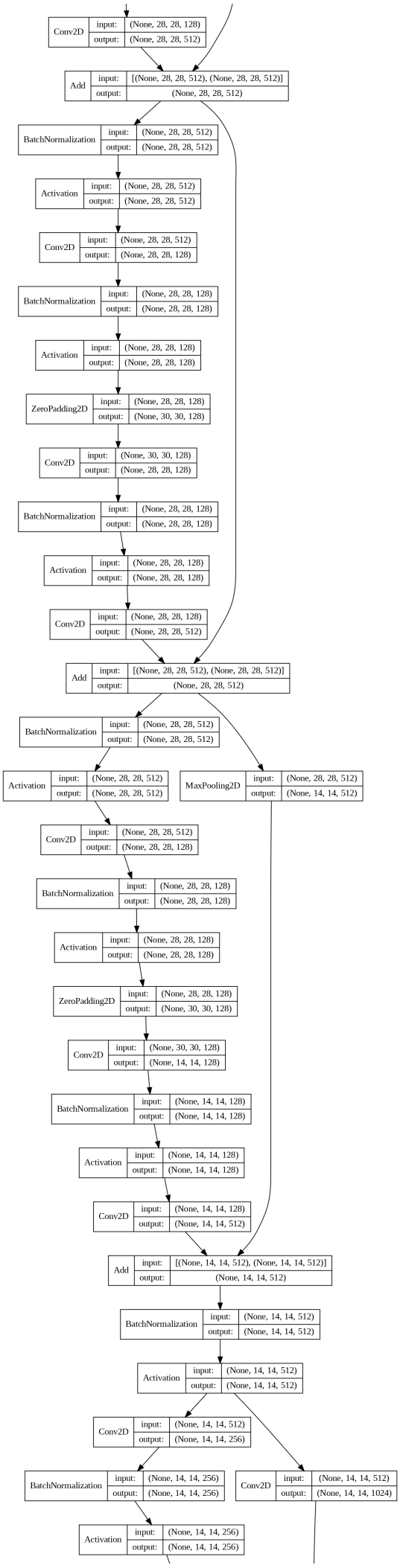
```
# Freezes all layers in the ResNet50V2 base model
resnet50v2_base.trainable = False
```

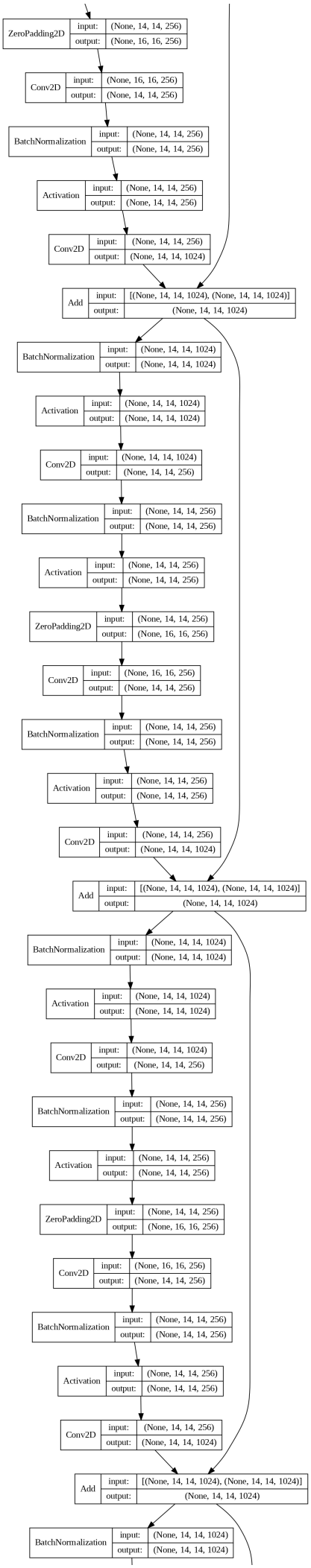
```
x = resnet50v2_base.output
# Add a global spatial average pooling layer to reduce the dimensions of the feature maps
x = GlobalAveragePooling2D()(x)
# Add a fully-connected layer to learn more complex representations specific to our task
x = Dense(1024, activation='relu')(x)
# Add a dropout layer for regularization
x = Dropout(0.5)(x)
# Add a logistic layer for binary classification
predictions = Dense(1, activation='sigmoid')(x)

# This is the model we will train
resnet50v2_model = Model(inputs=resnet50v2_base.input, outputs=predictions)

# Compile the model after setting layers to non-trainable
resnet50v2_model.compile(optimizer=Adam(learning_rate=0.0001), loss='binary_crossentropy', metrics=['accuracy'])
```

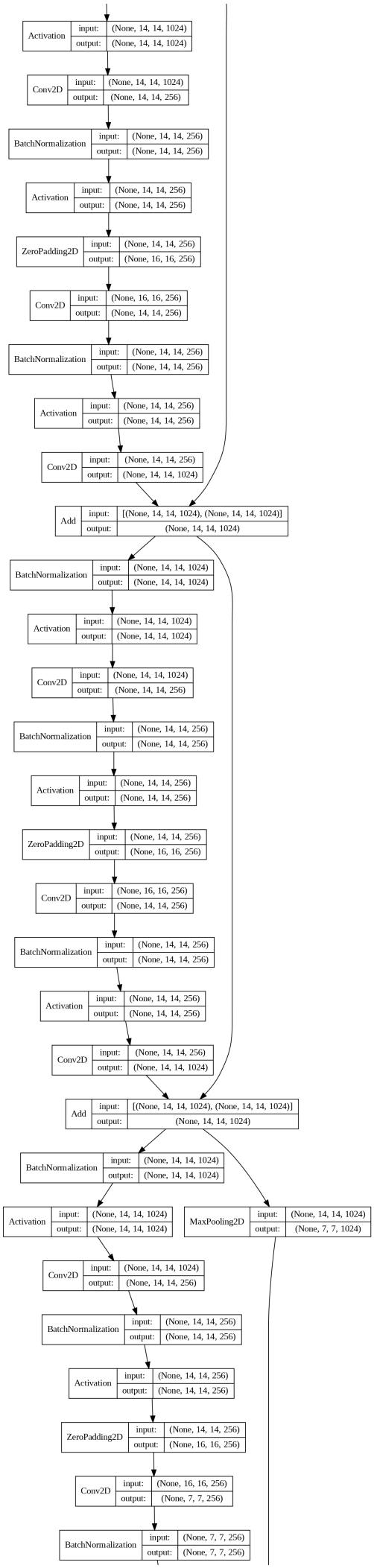
```
plot_model(
    resnet50v2_model,           # The ResNet50V2 model object to visualize
    show_shapes=True,           # Display the shapes of the layers
    show_layer_names=False,     # Hide the names of the layers
    dpi=150                      # Set the resolution of the generated plot
)
```

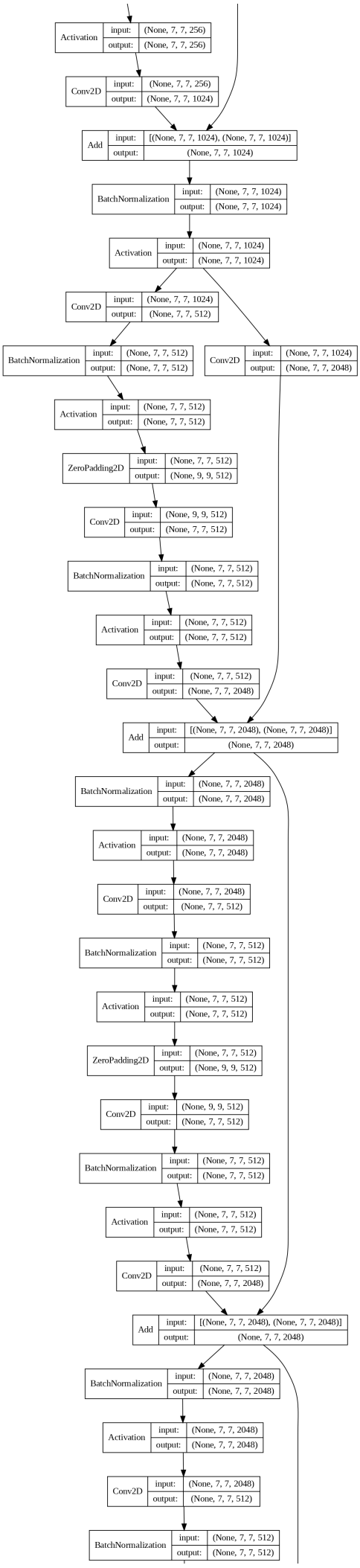

InputLayer | input: [(None, 224, 224, 3)] | output: [(None, 224, 224, 3)]
ZeroPadding2D | input: (None, 224, 224, 3) | output: (None, 230, 230, 3)
Conv2D | input: (None, 230, 230, 3) | output: (None, 112, 112, 64)
ZeroPadding2D | input: (None, 112, 112, 64) | output: (None, 114, 114, 64)
MaxPooling2D | input: (None, 114, 114, 64) | output: (None, 56, 56, 64)
BatchNormalization | input: (None, 56, 56, 64) | output: (None, 56, 56, 64)
Activation | input: (None, 56, 56, 64) | output: (None, 56, 56, 64)
Conv2D | input: (None, 56, 56, 64) | output: (None, 56, 56, 64)
BatchNormalization | input: (None, 56, 56, 64) | output: (None, 56, 56, 64)
Conv2D | input: (None, 56, 56, 64) | output: (None, 56, 56, 256)
Activation | input: (None, 56, 56, 64) | output: (None, 56, 56, 64)
ZeroPadding2D | input: (None, 56, 56, 64) | output: (None, 58, 58, 64)
Conv2D | input: (None, 58, 58, 64) | output: (None, 56, 56, 64)
BatchNormalization | input: (None, 56, 56, 64) | output: (None, 56, 56, 64)
Activation | input: (None, 56, 56, 64) | output: (None, 56, 56, 64)
Conv2D | input: (None, 56, 56, 64) | output: (None, 56, 56, 256)
Add | input: [(None, 56, 56, 256), (None, 56, 56, 256)] | output: (None, 56, 56, 256)
BatchNormalization | input: (None, 56, 56, 256) | output: (None, 56, 56, 256)
Activation | input: (None, 56, 56, 256) | output: (None, 56, 56, 256)
Conv2D | input: (None, 56, 56, 256) | output: (None, 56, 56, 64)
BatchNormalization | input: (None, 56, 56, 64) | output: (None, 56, 56, 64)
Activation | input: (None, 56, 56, 64) | output: (None, 56, 56, 64)
ZeroPadding2D | input: (None, 56, 56, 64) | output: (None, 58, 58, 64)
Conv2D | input: (None, 58, 58, 64) | output: (None, 56, 56, 64)
BatchNormalization | input: (None, 56, 56, 64) | output: (None, 56, 56, 64)
Activation | input: (None, 56, 56, 64) | output: (None, 56, 56, 64)
Conv2D | input: (None, 56, 56, 64) | output: (None, 56, 56, 256)
Add | input: [(None, 56, 56, 256), (None, 56, 56, 256)] | output: (None, 56, 56, 256)
BatchNormalization | input: (None, 56, 56, 256) | output: (None, 56, 56, 256)
Activation | input: (None, 56, 56, 256) | output: (None, 56, 56, 256)
MaxPooling2D | input: (None, 56, 56, 256) | output: (None, 28, 28, 256)

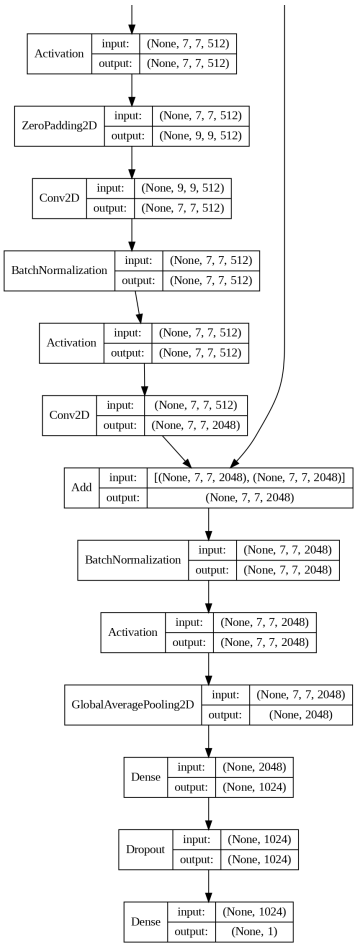

Conv2D input: (None, 56, 56, 256) output: (None, 56, 56, 64)
BatchNormalization input: (None, 56, 56, 64) output: (None, 56, 56, 64)
Activation input: (None, 56, 56, 64) output: (None, 56, 56, 64)
ZeroPadding2D input: (None, 56, 56, 64) output: (None, 58, 58, 64)
Conv2D input: (None, 58, 58, 64) output: (None, 28, 28, 64)
BatchNormalization input: (None, 28, 28, 64) output: (None, 28, 28, 64)
Activation input: (None, 28, 28, 64) output: (None, 28, 28, 64)
Conv2D input: (None, 28, 28, 64) output: (None, 28, 28, 256)
Add input: [(None, 28, 28, 256), (None, 28, 28, 256)] output: (None, 28, 28, 256)
BatchNormalization input: (None, 28, 28, 256) output: (None, 28, 28, 256)
Activation input: (None, 28, 28, 256) output: (None, 28, 28, 256)
Conv2D input: (None, 28, 28, 256) output: (None, 28, 28, 128)
BatchNormalization input: (None, 28, 28, 128) output: (None, 28, 28, 128)
Conv2D input: (None, 28, 28, 256) output: (None, 28, 28, 512)
Activation input: (None, 28, 28, 128) output: (None, 28, 28, 128)
ZeroPadding2D input: (None, 28, 28, 128) output: (None, 30, 30, 128)
Conv2D input: (None, 30, 30, 128) output: (None, 28, 28, 128)
BatchNormalization input: (None, 28, 28, 128) output: (None, 28, 28, 128)
Activation input: (None, 28, 28, 128) output: (None, 28, 28, 128)
Conv2D input: (None, 28, 28, 128) output: (None, 28, 28, 512)
Add input: [(None, 28, 28, 512), (None, 28, 28, 512)] output: (None, 28, 28, 512)
BatchNormalization input: (None, 28, 28, 512) output: (None, 28, 28, 512)
Activation input: (None, 28, 28, 512) output: (None, 28, 28, 512)
Conv2D input: (None, 28, 28, 512) output: (None, 28, 28, 128)
BatchNormalization input: (None, 28, 28, 128) output: (None, 28, 28, 128)
Activation input: (None, 28, 28, 128) output: (None, 28, 28, 128)
ZeroPadding2D input: (None, 28, 28, 128) output: (None, 30, 30, 128)
Conv2D input: (None, 30, 30, 128) output: (None, 28, 28, 128)
BatchNormalization input: (None, 28, 28, 128) output: (None, 28, 28, 128)
Activation input: (None, 28, 28, 128) output: (None, 28, 28, 128)


Conv2D input: (None, 28, 28, 128) output: (None, 28, 28, 512)
Add input: [(None, 28, 28, 512), (None, 28, 28, 512)] output: (None, 28, 28, 512)
BatchNormalization input: (None, 28, 28, 512) output: (None, 28, 28, 512)
Activation input: (None, 28, 28, 512) output: (None, 28, 28, 512)
Conv2D input: (None, 28, 28, 512) output: (None, 28, 28, 128)
BatchNormalization input: (None, 28, 28, 128) output: (None, 28, 28, 128)
Activation input: (None, 28, 28, 128) output: (None, 28, 28, 128)
ZeroPadding2D input: (None, 28, 28, 128) output: (None, 30, 30, 128)
Conv2D input: (None, 30, 30, 128) output: (None, 28, 28, 128)
BatchNormalization input: (None, 28, 28, 128) output: (None, 28, 28, 128)
Activation input: (None, 28, 28, 128) output: (None, 28, 28, 128)
Conv2D input: (None, 28, 28, 128) output: (None, 28, 28, 512)
Add input: [(None, 28, 28, 512), (None, 28, 28, 512)] output: (None, 28, 28, 512)
BatchNormalization input: (None, 28, 28, 512) output: (None, 28, 28, 512)
Activation input: (None, 28, 28, 512) output: (None, 28, 28, 512)
MaxPooling2D input: (None, 28, 28, 512) output: (None, 14, 14, 512)
Conv2D input: (None, 28, 28, 512) output: (None, 28, 28, 128)
BatchNormalization input: (None, 28, 28, 128) output: (None, 28, 28, 128)
Activation input: (None, 28, 28, 128) output: (None, 28, 28, 128)
ZeroPadding2D input: (None, 28, 28, 128) output: (None, 30, 30, 128)
Conv2D input: (None, 30, 30, 128) output: (None, 14, 14, 128)
BatchNormalization input: (None, 14, 14, 128) output: (None, 14, 14, 128)
Activation input: (None, 14, 14, 128) output: (None, 14, 14, 128)
Conv2D input: (None, 14, 14, 128) output: (None, 14, 14, 512)
Add input: [(None, 14, 14, 512), (None, 14, 14, 512)] output: (None, 14, 14, 512)
BatchNormalization input: (None, 14, 14, 512) output: (None, 14, 14, 512)
Activation input: (None, 14, 14, 512) output: (None, 14, 14, 512)
Conv2D input: (None, 14, 14, 512) output: (None, 14, 14, 256)
BatchNormalization input: (None, 14, 14, 256) output: (None, 14, 14, 256)
Conv2D input: (None, 14, 14, 512) output: (None, 14, 14, 1024)
Activation input: (None, 14, 14, 256) output: (None, 14, 14, 256)


ZeroPadding2D input: (None, 14, 14, 256) output: (None, 16, 16, 256)
Conv2D input: (None, 16, 16, 256) output: (None, 14, 14, 256)
BatchNormalization input: (None, 14, 14, 256) output: (None, 14, 14, 256)
Activation input: (None, 14, 14, 256) output: (None, 14, 14, 256)
Conv2D input: (None, 14, 14, 256) output: (None, 14, 14, 1024)
Add input: [(None, 14, 14, 1024), (None, 14, 14, 1024)] output: (None, 14, 14, 1024)
BatchNormalization input: (None, 14, 14, 1024) output: (None, 14, 14, 1024)
Activation input: (None, 14, 14, 1024) output: (None, 14, 14, 1024)
Conv2D input: (None, 14, 14, 1024) output: (None, 14, 14, 256)
BatchNormalization input: (None, 14, 14, 256) output: (None, 14, 14, 256)
Activation input: (None, 14, 14, 256) output: (None, 14, 14, 256)
ZeroPadding2D input: (None, 14, 14, 256) output: (None, 16, 16, 256)
Conv2D input: (None, 16, 16, 256) output: (None, 14, 14, 256)
BatchNormalization input: (None, 14, 14, 256) output: (None, 14, 14, 256)
Activation input: (None, 14, 14, 256) output: (None, 14, 14, 256)
Conv2D input: (None, 14, 14, 256) output: (None, 14, 14, 1024)
Add input: [(None, 14, 14, 1024), (None, 14, 14, 1024)] output: (None, 14, 14, 1024)
BatchNormalization input: (None, 14, 14, 1024) output: (None, 14, 14, 1024)
Activation input: (None, 14, 14, 1024) output: (None, 14, 14, 1024)
Conv2D input: (None, 14, 14, 1024) output: (None, 14, 14, 256)
BatchNormalization input: (None, 14, 14, 256) output: (None, 14, 14, 256)
Activation input: (None, 14, 14, 256) output: (None, 14, 14, 256)
ZeroPadding2D input: (None, 14, 14, 256) output: (None, 16, 16, 256)
Conv2D input: (None, 16, 16, 256) output: (None, 14, 14, 256)
BatchNormalization input: (None, 14, 14, 256) output: (None, 14, 14, 256)
Activation input: (None, 14, 14, 256) output: (None, 14, 14, 256)
Conv2D input: (None, 14, 14, 256) output: (None, 14, 14, 1024)
Add input: [(None, 14, 14, 1024), (None, 14, 14, 1024)] output: (None, 14, 14, 1024)
BatchNormalization input: (None, 14, 14, 1024) output: (None, 14, 14, 1024)


Activation | input: (None, 14, 14, 1024) | output: (None, 14, 14, 1024)
Conv2D | input: (None, 14, 14, 1024) | output: (None, 14, 14, 256)
BatchNormalization | input: (None, 14, 14, 256) | output: (None, 14, 14, 256)
Activation | input: (None, 14, 14, 256) | output: (None, 14, 14, 256)
ZeroPadding2D | input: (None, 14, 14, 256) | output: (None, 16, 16, 256)
Conv2D | input: (None, 16, 16, 256) | output: (None, 14, 14, 256)
BatchNormalization | input: (None, 14, 14, 256) | output: (None, 14, 14, 256)
Activation | input: (None, 14, 14, 256) | output: (None, 14, 14, 256)
Conv2D | input: (None, 14, 14, 256) | output: (None, 14, 14, 1024)
Add | input: [(None, 14, 14, 1024), (None, 14, 14, 1024)] | output: (None, 14, 14, 1024)
BatchNormalization | input: (None, 14, 14, 1024) | output: (None, 14, 14, 1024)
Activation | input: (None, 14, 14, 1024) | output: (None, 14, 14, 1024)
Conv2D | input: (None, 14, 14, 1024) | output: (None, 14, 14, 256)
BatchNormalization | input: (None, 14, 14, 256) | output: (None, 14, 14, 256)
Activation | input: (None, 14, 14, 256) | output: (None, 14, 14, 256)
ZeroPadding2D | input: (None, 14, 14, 256) | output: (None, 16, 16, 256)
Conv2D | input: (None, 16, 16, 256) | output: (None, 14, 14, 256)
BatchNormalization | input: (None, 14, 14, 256) | output: (None, 14, 14, 256)
Activation | input: (None, 14, 14, 256) | output: (None, 14, 14, 256)
Conv2D | input: (None, 14, 14, 256) | output: (None, 14, 14, 1024)
Add | input: [(None, 14, 14, 1024), (None, 14, 14, 1024)] | output: (None, 14, 14, 1024)
BatchNormalization | input: (None, 14, 14, 1024) | output: (None, 14, 14, 1024)
Activation | input: (None, 14, 14, 1024) | output: (None, 14, 14, 1024)
MaxPooling2D | input: (None, 14, 14, 1024) | output: (None, 7, 7, 1024)
Conv2D | input: (None, 14, 14, 1024) | output: (None, 14, 14, 256)
BatchNormalization | input: (None, 14, 14, 256) | output: (None, 14, 14, 256)
Activation | input: (None, 14, 14, 256) | output: (None, 14, 14, 256)
ZeroPadding2D | input: (None, 14, 14, 256) | output: (None, 16, 16, 256)
Conv2D | input: (None, 16, 16, 256) | output: (None, 7, 7, 256)
BatchNormalization | input: (None, 7, 7, 256) | output: (None, 7, 7, 256)


Activation | input: (None, 7, 7, 256) | output: (None, 7, 7, 256)
Conv2D | input: (None, 7, 7, 256) | output: (None, 7, 7, 1024)
Add | input: [(None, 7, 7, 1024), (None, 7, 7, 1024)] | output: (None, 7, 7, 1024)
BatchNormalization | input: (None, 7, 7, 1024) | output: (None, 7, 7, 1024)
Activation | input: (None, 7, 7, 1024) | output: (None, 7, 7, 1024)
Conv2D | input: (None, 7, 7, 1024) | output: (None, 7, 7, 512)
BatchNormalization | input: (None, 7, 7, 512) | output: (None, 7, 7, 512)
Conv2D | input: (None, 7, 7, 1024) | output: (None, 7, 7, 2048)
Activation | input: (None, 7, 7, 512) | output: (None, 7, 7, 512)
ZeroPadding2D | input: (None, 7, 7, 512) | output: (None, 9, 9, 512)
Conv2D | input: (None, 9, 9, 512) | output: (None, 7, 7, 512)
BatchNormalization | input: (None, 7, 7, 512) | output: (None, 7, 7, 512)
Activation | input: (None, 7, 7, 512) | output: (None, 7, 7, 512)
Conv2D | input: (None, 7, 7, 512) | output: (None, 7, 7, 2048)
Add | input: [(None, 7, 7, 2048), (None, 7, 7, 2048)] | output: (None, 7, 7, 2048)
BatchNormalization | input: (None, 7, 7, 2048) | output: (None, 7, 7, 2048)
Activation | input: (None, 7, 7, 2048) | output: (None, 7, 7, 2048)
Conv2D | input: (None, 7, 7, 2048) | output: (None, 7, 7, 512)
BatchNormalization | input: (None, 7, 7, 512) | output: (None, 7, 7, 512)
Activation | input: (None, 7, 7, 512) | output: (None, 7, 7, 512)
ZeroPadding2D | input: (None, 7, 7, 512) | output: (None, 9, 9, 512)
Conv2D | input: (None, 9, 9, 512) | output: (None, 7, 7, 512)
BatchNormalization | input: (None, 7, 7, 512) | output: (None, 7, 7, 512)
Activation | input: (None, 7, 7, 512) | output: (None, 7, 7, 512)
Conv2D | input: (None, 7, 7, 512) | output: (None, 7, 7, 2048)
Add | input: [(None, 7, 7, 2048), (None, 7, 7, 2048)] | output: (None, 7, 7, 2048)
BatchNormalization | input: (None, 7, 7, 2048) | output: (None, 7, 7, 2048)
Activation | input: (None, 7, 7, 2048) | output: (None, 7, 7, 2048)
Conv2D | input: (None, 7, 7, 2048) | output: (None, 7, 7, 512)
BatchNormalization | input: (None, 7, 7, 512) | output: (None, 7, 7, 512)


Activation
input: (None, 7, 7, 512)
output: (None, 7, 7, 512)
ZeroPadding2D
input: (None, 7, 7, 512)
output: (None, 9, 9, 512)
Conv2D
input: (None, 9, 9, 512)
output: (None, 7, 7, 512)
BatchNormalization
input: (None, 7, 7, 512)
output: (None, 7, 7, 512)
Activation
input: (None, 7, 7, 512)
output: (None, 7, 7, 512)
Conv2D
input: (None, 7, 7, 512)
output: (None, 7, 7, 2048)
Add
input: [(None, 7, 7, 2048), (None, 7, 7, 2048)]
output: (None, 7, 7, 2048)
BatchNormalization
input: (None, 7, 7, 2048)
output: (None, 7, 7, 2048)
Activation
input: (None, 7, 7, 2048)
output: (None, 7, 7, 2048)
GlobalAveragePooling2D
input: (None, 7, 7, 2048)
output: (None, 2048)
Dense
input: (None, 2048)
output: (None, 1024)
Dropout
input: (None, 1024)
output: (None, 1024)
Dense
input: (None, 1024)
output: (None, 1)

```
# Model summary
resnet50v2_model.summary()
```

```
conv5_block3_preact_bn (Ba  (None, 7, 7, 2048)          8192      ['conv5_block2_out[0][0]']
tchNormalization)

conv5_block3_preact_relu (  (None, 7, 7, 2048)          0         ['conv5_block3_preact_bn[0][0]
Activation)                                                       ']

conv5_block3_1_conv (Conv2  (None, 7, 7, 512)           1048576   ['conv5_block3_preact_relu[0][
D)                                                                0]']

conv5_block3_1_bn (BatchNo  (None, 7, 7, 512)           2048      ['conv5_block3_1_conv[0][0]']
rmalization)

conv5_block3_1_relu (Activ  (None, 7, 7, 512)           0         ['conv5_block3_1_bn[0][0]']
ation)

conv5_block3_2_pad (ZeroPa  (None, 9, 9, 512)           0         ['conv5_block3_1_relu[0][0]']
dding2D)

conv5_block3_2_conv (Conv2  (None, 7, 7, 512)           2359296   ['conv5_block3_2_pad[0][0]']
D)

conv5_block3_2_bn (BatchNo  (None, 7, 7, 512)           2048      ['conv5_block3_2_conv[0][0]']
rmalization)

conv5_block3_2_relu (Activ  (None, 7, 7, 512)           0         ['conv5_block3_2_bn[0][0]']
ation)

conv5_block3_3_conv (Conv2  (None, 7, 7, 2048)          1050624   ['conv5_block3_2_relu[0][0]']
D)

conv5_block3_out (Add)      (None, 7, 7, 2048)          0         ['conv5_block2_out[0][0]',
                                                                   'conv5_block3_3_conv[0][0]']

post_bn (BatchNormalizatio  (None, 7, 7, 2048)          8192      ['conv5_block3_out[0][0]']
n)

post_relu (Activation)      (None, 7, 7, 2048)          0         ['post_bn[0][0]']

global_average_pooling2d (  (None, 2048)                0         ['post_relu[0][0]']
GlobalAveragePooling2D)

dense (Dense)               (None, 1024)                2098176   ['global_average_pooling2d[0][
                                                                  0]']

dropout (Dropout)           (None, 1024)                0         ['dense[0][0]']

dense_1 (Dense)             (None, 1)                   1025      ['dropout[0][0]']

==================================================================================================
Total params: 25664001 (97.90 MB)
Trainable params: 2099201 (8.01 MB)
Non-trainable params: 23564800 (89.89 MB)
__________________________________________________________________________________________________
```

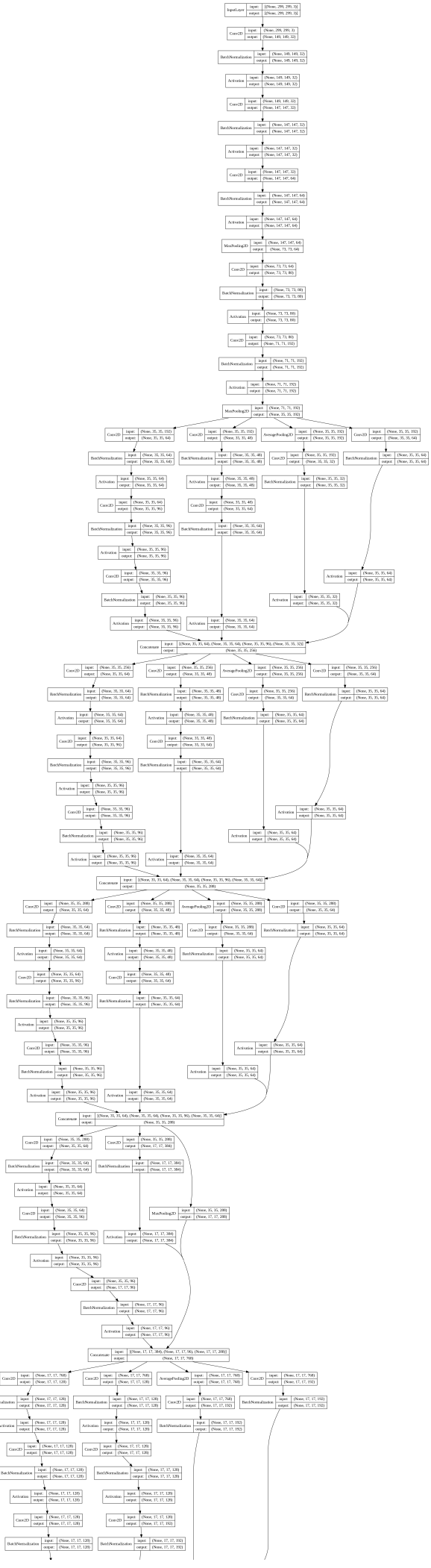
```
# Load the InceptionV3 model pre-trained on ImageNet data, excluding the top classifier
inceptionv3_base = InceptionV3(weights='imagenet', include_top=False, input_shape=(299, 299, 3))
inceptionv3_base.summary()

# Freezes all layers in the InceptionV3 base model
inceptionv3_base.trainable = False

# Following the same pattern to add new layers
x = inceptionv3_base.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(1, activation='sigmoid')(x)

# This is the model we will train
inceptionv3_model = Model(inputs=inceptionv3_base.input, outputs=predictions)

# Compile the model after setting layers to non-trainable
inceptionv3_model.compile(optimizer=Adam(learning_rate=0.0001), loss='binary_crossentropy', metrics=['accuracy'])

# InceptionV3 model summary
inceptionv3_model.summary()
```

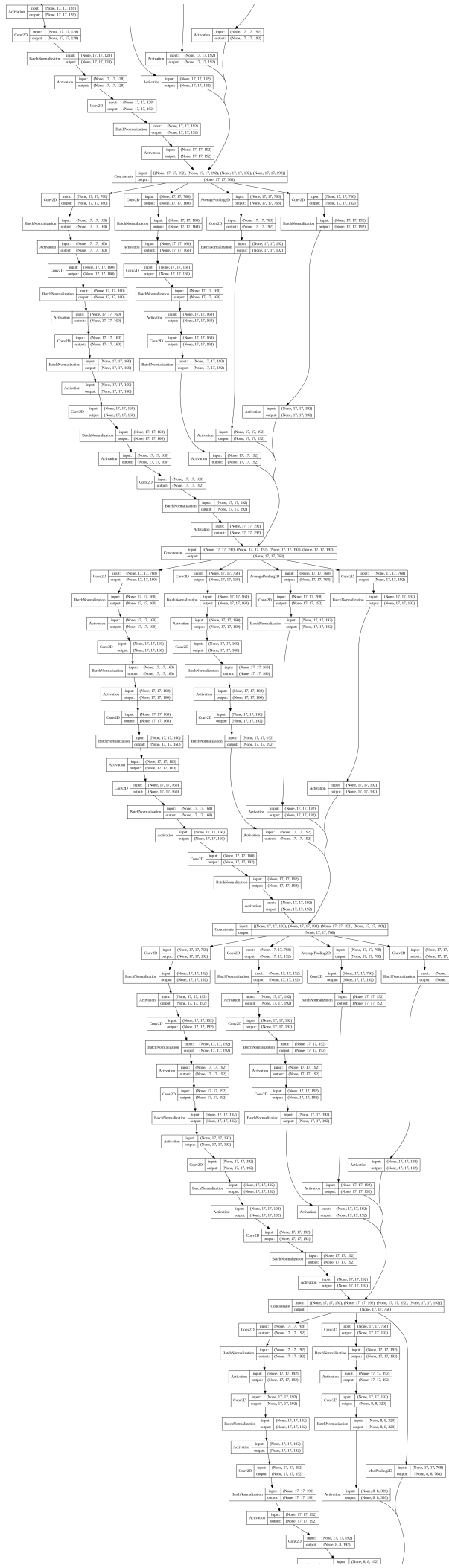
```
activation_87 (Activation)  (None, 8, 8, 384)           0         ['batch_normalization_87[0][0]
                                                                   ']

activation_88 (Activation)  (None, 8, 8, 384)           0         ['batch_normalization_88[0][0]
                                                                   ']

activation_91 (Activation)  (None, 8, 8, 384)           0         ['batch_normalization_91[0][0]
                                                                   ']

activation_92 (Activation)  (None, 8, 8, 384)           0         ['batch_normalization_92[0][0]
                                                                   ']

batch_normalization_93 (Ba  (None, 8, 8, 192)           576       ['conv2d_93[0][0]']
tchNormalization)

activation_85 (Activation)  (None, 8, 8, 320)           0         ['batch_normalization_85[0][0]
                                                                   ']

mixed9_1 (Concatenate)      (None, 8, 8, 768)           0         ['activation_87[0][0]',
                                                                   'activation_88[0][0]']

concatenate_1 (Concatenate  (None, 8, 8, 768)           0         ['activation_91[0][0]',
)                                                                  'activation_92[0][0]']

activation_93 (Activation)  (None, 8, 8, 192)           0         ['batch_normalization_93[0][0]
                                                                   ']

mixed10 (Concatenate)       (None, 8, 8, 2048)          0         ['activation_85[0][0]',
                                                                   'mixed9_1[0][0]',
                                                                   'concatenate_1[0][0]',
                                                                   'activation_93[0][0]']

global_average_pooling2d_1  (None, 2048)                0         ['mixed10[0][0]']
 (GlobalAveragePooling2D)

dense_2 (Dense)             (None, 1024)                2098176   ['global_average_pooling2d_1[0
                                                                   ][0]']

dropout_1 (Dropout)         (None, 1024)                0         ['dense_2[0][0]']

dense_3 (Dense)             (None, 1)                   1025      ['dropout_1[0][0]']

==================================================================================================
Total params: 23901985 (91.18 MB)
Trainable params: 2099201 (8.01 MB)
Non-trainable params: 21802784 (83.17 MB)
__________________________________________________________________________________________________
```

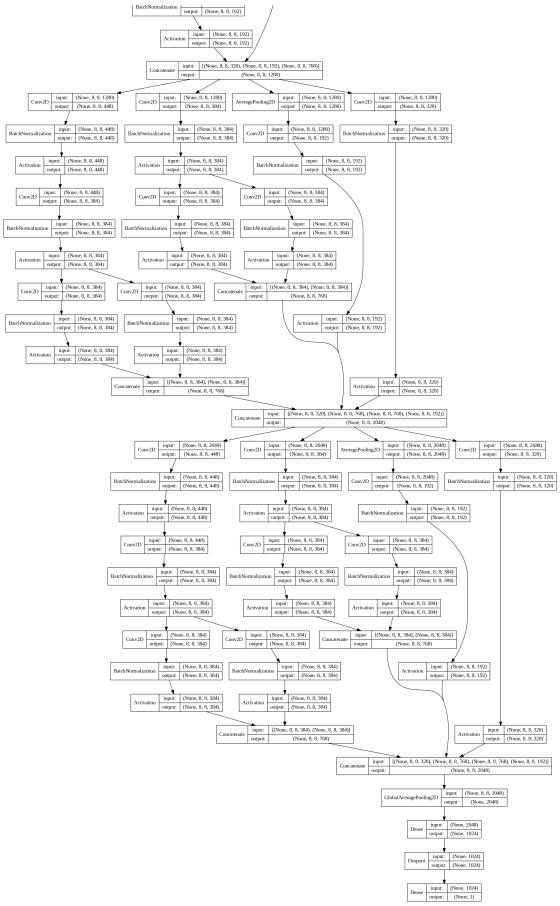
```
# Load the InceptionV3 model pre-trained on ImageNet data, excluding the top classifier
inceptionv3_base = InceptionV3(weights='imagenet', include_top=False, input_shape=(299, 299, 3))
inceptionv3_base.summary()

# Freezes all layers in the InceptionV3 base model
inceptionv3_base.trainable = False

# Following the same pattern to add new layers
x = inceptionv3_base.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(1, activation='sigmoid')(x)

# This is the model we will train
inceptionv3_model = Model(inputs=inceptionv3_base.input, outputs=predictions)

# Compile the model after setting layers to non-trainable
inceptionv3_model.compile(optimizer=Adam(learning_rate=0.0001), loss='binary_crossentropy', metrics=['accuracy'])

# InceptionV3 model summary
inceptionv3_model.summary()
```

```
 activation_185 (Activation  (None, 8, 8, 384)           0         ['batch_normalization_185[0][0
 )                                                                  ]']

 activation_186 (Activation  (None, 8, 8, 384)           0         ['batch_normalization_186[0][0
 )                                                                  ]']

 batch_normalization_187 (B  (None, 8, 8, 192)           576       ['conv2d_187[0][0]']
 atchNormalization)

 activation_179 (Activation  (None, 8, 8, 320)           0         ['batch_normalization_179[0][0
 )                                                                  ]']

 mixed9_1 (Concatenate)      (None, 8, 8, 768)           0         ['activation_181[0][0]',
                                                                    'activation_182[0][0]']

 concatenate_3 (Concatenate  (None, 8, 8, 768)           0         ['activation_185[0][0]',
 )                                                                  'activation_186[0][0]']

 activation_187 (Activation  (None, 8, 8, 192)           0         ['batch_normalization_187[0][0
 )                                                                  ]']

 mixed10 (Concatenate)       (None, 8, 8, 2048)          0         ['activation_179[0][0]',
                                                                    'mixed9_1[0][0]',
                                                                    'concatenate_3[0][0]',
                                                                    'activation_187[0][0]']

 global_average_pooling2d_2  (None, 2048)                0         ['mixed10[0][0]']
  (GlobalAveragePooling2D)

 dense_4 (Dense)             (None, 1024)                2098176   ['global_average_pooling2d_2[0
                                                                   ][0]']

 dropout_2 (Dropout)         (None, 1024)                0         ['dense_4[0][0]']

 dense_5 (Dense)             (None, 1)                   1025      ['dropout_2[0][0]']

==================================================================================================
Total params: 23901985 (91.18 MB)
Trainable params: 2099201 (8.01 MB)
Non-trainable params: 21802784 (83.17 MB)
__________________________________________________________________________________________________
```

```
plot_model(
    inceptionv3_model,         # The InceptionV3 model object to visualize
    show_shapes=True,          # Display the shapes of the layers
    show_layer_names=False,    # Hide the names of the layers
    dpi=150                    # Set the resolution of the generated plot
)
```







```
# Load the MobileNetV2 model pre-trained on ImageNet data, excluding the top classifier
mobilenetv2_base = MobileNetV2(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
mobilenetv2_base.summary()
```

```
 block_15_expand_relu (ReLU  (None, 7, 7, 960)           0         ['block_15_expand_BN[0][0]']
 )

 block_15_depthwise (Depthw  (None, 7, 7, 960)           8640      ['block_15_expand_relu[0][0]']
 iseConv2D)

 block_15_depthwise_BN (Bat  (None, 7, 7, 960)           3840      ['block_15_depthwise[0][0]']
 chNormalization)

 block_15_depthwise_relu (R  (None, 7, 7, 960)           0         ['block_15_depthwise_BN[0][0]'
 eLU)                                                              ]

 block_15_project (Conv2D)   (None, 7, 7, 160)           153600    ['block_15_depthwise_relu[0][0
                                                                   ]']

 block_15_project_BN (Batch  (None, 7, 7, 160)           640       ['block_15_project[0][0]']
 Normalization)

 block_15_add (Add)          (None, 7, 7, 160)           0         ['block_14_add[0][0]',
                                                                    'block_15_project_BN[0][0]']

 block_16_expand (Conv2D)    (None, 7, 7, 960)           153600    ['block_15_add[0][0]']

 block_16_expand_BN (BatchN  (None, 7, 7, 960)           3840      ['block_16_expand[0][0]']
 ormalization)

 block_16_expand_relu (ReLU  (None, 7, 7, 960)           0         ['block_16_expand_BN[0][0]']
 )

 block_16_depthwise (Depthw  (None, 7, 7, 960)           8640      ['block_16_expand_relu[0][0]']
 iseConv2D)

 block_16_depthwise_BN (Bat  (None, 7, 7, 960)           3840      ['block_16_depthwise[0][0]']
 chNormalization)

 block_16_depthwise_relu (R  (None, 7, 7, 960)           0         ['block_16_depthwise_BN[0][0]'
 eLU)                                                              ]

 block_16_project (Conv2D)   (None, 7, 7, 320)           307200    ['block_16_depthwise_relu[0][0
                                                                   ]']

 block_16_project_BN (Batch  (None, 7, 7, 320)           1280      ['block_16_project[0][0]']
 Normalization)

 Conv_1 (Conv2D)             (None, 7, 7, 1280)          409600    ['block_16_project_BN[0][0]']

 Conv_1_bn (BatchNormalizat  (None, 7, 7, 1280)          5120      ['Conv_1[0][0]']
 ion)

 out_relu (ReLU)             (None, 7, 7, 1280)          0         ['Conv_1_bn[0][0]']

==================================================================================================
Total params: 2257984 (8.61 MB)
Trainable params: 2223872 (8.48 MB)
Non-trainable params: 34112 (133.25 KB)
__________________________________________________________________________________________________
```

```
# Freeze all layers in the MobileNetV2 base model
mobilenetv2_base.trainable = False
```

```
# Following the same pattern to add new layers to MobileNetV2 base
x = mobilenetv2_base.output
x = GlobalAveragePooling2D()(x)
x = Dense(128, activation='relu')(x)  # Reduced number of neurons!
x = Dropout(0.5)(x)
predictions = Dense(1, activation='sigmoid')(x)

# This is the model we will train
mobilenetv2_nn_model = Model(inputs=mobilenetv2_base.input, outputs=predictions)

# Compile the model after setting layers to non-trainable
mobilenetv2_nn_model.compile(optimizer=Adam(learning_rate=0.0001), loss='binary_crossentropy', metrics=['accuracy'])

# MobileNetV2 model summary
mobilenetv2_nn_model.summary()
```

```
block_15_depthwise_relu (R  (None, 7, 7, 960)           0         ['block_15_depthwise_BN[0][0]'
eLU)                                                              ]

block_15_project (Conv2D)   (None, 7, 7, 160)           153600    ['block_15_depthwise_relu[0][0
                                                                  ]']

block_15_project_BN (Batch  (None, 7, 7, 160)           640       ['block_15_project[0][0]']
Normalization)

block_15_add (Add)          (None, 7, 7, 160)           0         ['block_14_add[0][0]',
                                                                   'block_15_project_BN[0][0]']

block_16_expand (Conv2D)    (None, 7, 7, 960)           153600    ['block_15_add[0][0]']

block_16_expand_BN (BatchN  (None, 7, 7, 960)           3840      ['block_16_expand[0][0]']
ormalization)

block_16_expand_relu (ReLU  (None, 7, 7, 960)           0         ['block_16_expand_BN[0][0]']
)

block_16_depthwise (Depthw  (None, 7, 7, 960)           8640      ['block_16_expand_relu[0][0]']
iseConv2D)

block_16_depthwise_BN (Bat  (None, 7, 7, 960)           3840      ['block_16_depthwise[0][0]']
chNormalization)

block_16_depthwise_relu (R  (None, 7, 7, 960)           0         ['block_16_depthwise_BN[0][0]'
eLU)                                                              ]

block_16_project (Conv2D)   (None, 7, 7, 320)           307200    ['block_16_depthwise_relu[0][0
                                                                  ]']

block_16_project_BN (Batch  (None, 7, 7, 320)           1280      ['block_16_project[0][0]']
Normalization)

Conv_1 (Conv2D)             (None, 7, 7, 1280)          409600    ['block_16_project_BN[0][0]']

Conv_1_bn (BatchNormalizat  (None, 7, 7, 1280)          5120      ['Conv_1[0][0]']
ion)

out_relu (ReLU)             (None, 7, 7, 1280)          0         ['Conv_1_bn[0][0]']

global_average_pooling2d_3  (None, 1280)                0         ['out_relu[0][0]']
 (GlobalAveragePooling2D)

dense_6 (Dense)             (None, 128)                 163968    ['global_average_pooling2d_3[0
                                                                  ][0]']

dropout_3 (Dropout)         (None, 128)                 0         ['dense_6[0][0]']

dense_7 (Dense)             (None, 1)                   129       ['dropout_3[0][0]']

==================================================================================================
Total params: 2422081 (9.24 MB)
Trainable params: 164097 (641.00 KB)
Non-trainable params: 2257984 (8.61 MB)
__________________________________________________________________________________________________
```

```
# Add a GlobalAveragePooling layer to reduce the feature maps to a single vector per map
gap = GlobalAveragePooling2D()(mobilenetv2_base.output)

# Construct the feature extractor model
mobilenetv2_feature_extractor = Model(inputs=mobilenetv2_base.input, outputs=gap)
```

```
# Investigate the output shape of feature extractor
output_shape = mobilenetv2_feature_extractor.output_shape
print("Output shape of feature extractor model:", output_shape)
```

```
Output shape of feature extractor model: (None, 1280)
```

```
def train_model(model, preprocessing_function, image_dimensions=(224, 224), batch_size=32, num_epochs=20):
    """
    Trains the given model and returns the trained model, its history, and the validation generator.

    Parameters:
    - model: Model, a compiled instance of a Keras model to be trained.
    - preprocessing_function: function, preprocessing function to be applied to input data.
    - image_dimensions: tuple, dimensions of the images (width, height).
    - batch_size: int, number of samples per batch of computation.
    - num_epochs: int, number of epochs to train the model.

    Returns:
    - model: The trained model instance.
    - history: A History object containing the training history.
    - val_generator: The validation data generator.
    """

    # Create data generators
    train_generator, val_generator = create_data_generators(preprocessing_function, batch_size, image_dimensions)

    # Define the callbacks
    reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, min_lr=0.00001)
    early_stopping = EarlyStopping(monitor='val_loss', mode='min', patience=10, restore_best_weights=True, verbose=1)

    # Train the model
    history = model.fit(
        train_generator,
        steps_per_epoch=len(train_generator),
        epochs=num_epochs,
        validation_data=val_generator,
        validation_steps=len(val_generator),
        callbacks=[reduce_lr, early_stopping]
    )

    return model, history, val_generator
```

```
# Train the ResNet50V2 model classifier
resnet50v2_model, resnet50v2_history, resnet50v2_val_generator = train_model(
    resnet50v2_model,
    preprocessing_function=resnet_preprocess_input,  # The preprocessing function for ResNet50V2
    image_dimensions=(224, 224),                     # Adjusting to the expected input size for ResNet50V2
    batch_size=32,
    num_epochs=0
)
```

```
Found 20000 validated image filenames belonging to 2 classes.
Found 5000 validated image filenames belonging to 2 classes.
```

```
# Train the InceptionV3 model classifier
inceptionv3_model, inceptionv3_history, inceptionv3_val_generator = train_model(
    inceptionv3_model,
    preprocessing_function=inceptionv3_preprocess_input,  # The preprocessing function for InceptionV3
    image_dimensions=(299, 299),                          # Adjusting to the expected input size for InceptionV3
    batch_size=32,
    num_epochs=0
)
```

```
Found 20000 validated image filenames belonging to 2 classes.
Found 5000 validated image filenames belonging to 2 classes.
```

```
# Train the MobileNetV2 neural network classifier
mobilenetv2_nn_model, mobilenetv2_nn_history, mobilenetv2_nn_val_generator = train_model(
    mobilenetv2_nn_model,
    preprocessing_function=mobilenetv2_preprocess_input,  # The preprocessing function for MobileNetV2
    image_dimensions=(224, 224),                          # Input size for MobileNetV2
    batch_size=32,
    num_epochs=0
)
```

```
Found 20000 validated image filenames belonging to 2 classes.
Found 5000 validated image filenames belonging to 2 classes.
```

```
def create_feature_extraction_generator(preprocessing_function, dataframe, x_col, y_col, batch_size=32, image_dimensions=(224, 224)):
    """
    Creates a data generator without augmentation for feature extraction purposes.

    Parameters:
    - preprocessing_function: function, preprocessing function to be applied to input data.
    - dataframe: pandas.DataFrame, the dataframe containing the file paths and labels.
    - x_col: str, the name of the dataframe column containing the file paths.
    - y_col: str, the name of the dataframe column containing the labels.
    - batch_size: int, number of samples per batch of computation.
    - image_dimensions: tuple, dimensions of the images (width, height).

    Returns:
    - generator: An iterable Keras generator that outputs batches of preprocessed images.
    """

    # Initialize the ImageDataGenerator with the preprocessing function
    datagen = ImageDataGenerator(preprocessing_function=preprocessing_function)

    # Create the generator to read images from the dataframe
    generator = datagen.flow_from_dataframe(
        dataframe=dataframe,          # The dataframe containing paths and labels
        x_col=x_col,                  # Column in dataframe that contains the paths to the images
        y_col=y_col,                  # Column in dataframe that contains the labels
        target_size=image_dimensions, # The dimensions to which all images found will be resized
        batch_size=batch_size,        # Number of images to be yielded from the generator per batch
        class_mode='binary',          # Class mode for binary classification
        shuffle=False                 # No shuffling. Important to keep data in order!
    )

    return generator
```

```
def extract_features(model, generator):
    """
    Extract features using the given model and data from the generator.

    Parameters:
    - model: The pre-trained model used for feature extraction.
    - generator: The generator that yields batches of input data.

    Returns:
    - features: A numpy array of extracted features.
    - labels: A numpy array of corresponding labels.
    """
    # Use the model's predict method to extract features
    features = model.predict(generator, steps=len(generator), verbose=1)

    # Retrieve the labels from the generator's classes attribute
    labels = generator.classes

    return features, labels
```

```
def reduce_dimensionality(features, n_components=0.90, fit_pca=None):
    """
    Reduce the dimensionality of the given feature set using PCA.

    Parameters:
    - features: array-like, feature set to reduce.
    - n_components: int, float, or None, the number of principal components to keep.
                    If 0 < n_components < 1, select the number of components such that
                    the amount of variance that needs to be explained is greater than the
                    percentage specified by n_components.
    - fit_pca: PCA object from sklearn, pre-fitted PCA to use for transforming features.
               If None, fit a new PCA on the provided features.

    Returns:
    - reduced_features: array-like, feature set transformed into reduced dimensionality space.
    - pca: PCA object, the PCA used to transform the features. Only returned if fit_pca is None.
    """
    # If no pre-fitted PCA is provided, fit PCA on the features
    if fit_pca is None:
        pca = PCA(n_components=n_components)
        reduced_features = pca.fit_transform(features)
        # Print the cumulative explained variance ratio
        print(f'Cumulative explained variance by {pca.n_components_} principal components: {np.sum(pca.explained_variance_ratio_):.2f}'
        return reduced_features, pca
    else:
        # If a pre-fitted PCA is provided, transform the features using it
        reduced_features = fit_pca.transform(features)
        return reduced_features
```

```
def train_svm(features, labels):
    """
    Trains an SVM classifier using the provided features and labels.

    Parameters:
    - features: array-like, shape (n_samples, n_features)
      The feature vectors extracted from the images.
    - labels: array-like, shape (n_samples,)
      The target labels corresponding to each feature vector.

    Returns:
    - svm: object
      The trained SVM classifier.
    """

    # Initialize the SVM classifier with a linear kernel.
    # Linear is chosen for its efficiency and effectiveness for high dimensional datasets especially image data.
    svm = SVC(kernel='linear')

    # Train the SVM classifier on the provided features and labels.
    svm.fit(features, labels)

    # The trained model is returned for further use, such as prediction on test data.
    return svm
```

```
# Create data generators for training and validation sets
train_generator = create_feature_extraction_generator(mobilenetv2_preprocess_input, train_df, "filepath", "label", 32, (224, 224))
val_generator = create_feature_extraction_generator(mobilenetv2_preprocess_input, val_df, "filepath", "label", 32, (224, 224))

# Extract features from the generators using the feature extractor model
train_features, train_labels = extract_features(mobilenetv2_feature_extractor, train_generator)
val_features, val_labels = extract_features(mobilenetv2_feature_extractor, val_generator)

# Reduce the dimensionality of the training features and get the fitted PCA
train_features_reduced, pca = reduce_dimensionality(train_features, n_components=0.90)
# Use the fitted PCA to transform the validation features
val_features_reduced = reduce_dimensionality(val_features, fit_pca=pca)

# Train the SVM model on the reduced training features
svm_model = train_svm(train_features_reduced, train_labels)

# Predict on the reduced validation features
val_predictions = svm_model.predict(val_features_reduced)

# Calculate and print the SVM classifier accuracy on the validation set
svm_validation_accuracy = accuracy_score(val_labels, val_predictions)
print(f"SVM Classifier Validation Accuracy: {svm_validation_accuracy * 100:.2f}%")
```

```
Found 20000 validated image filenames belonging to 2 classes.
Found 5000 validated image filenames belonging to 2 classes.
625/625 [==============================] - 943s 1s/step
157/157 [==============================] - 210s 1s/step
Cumulative explained variance by 544 principal components: 0.90
SVM Classifier Validation Accuracy: 98.44%
```