Final Project Submission:Image Classification Model with Deep Learning

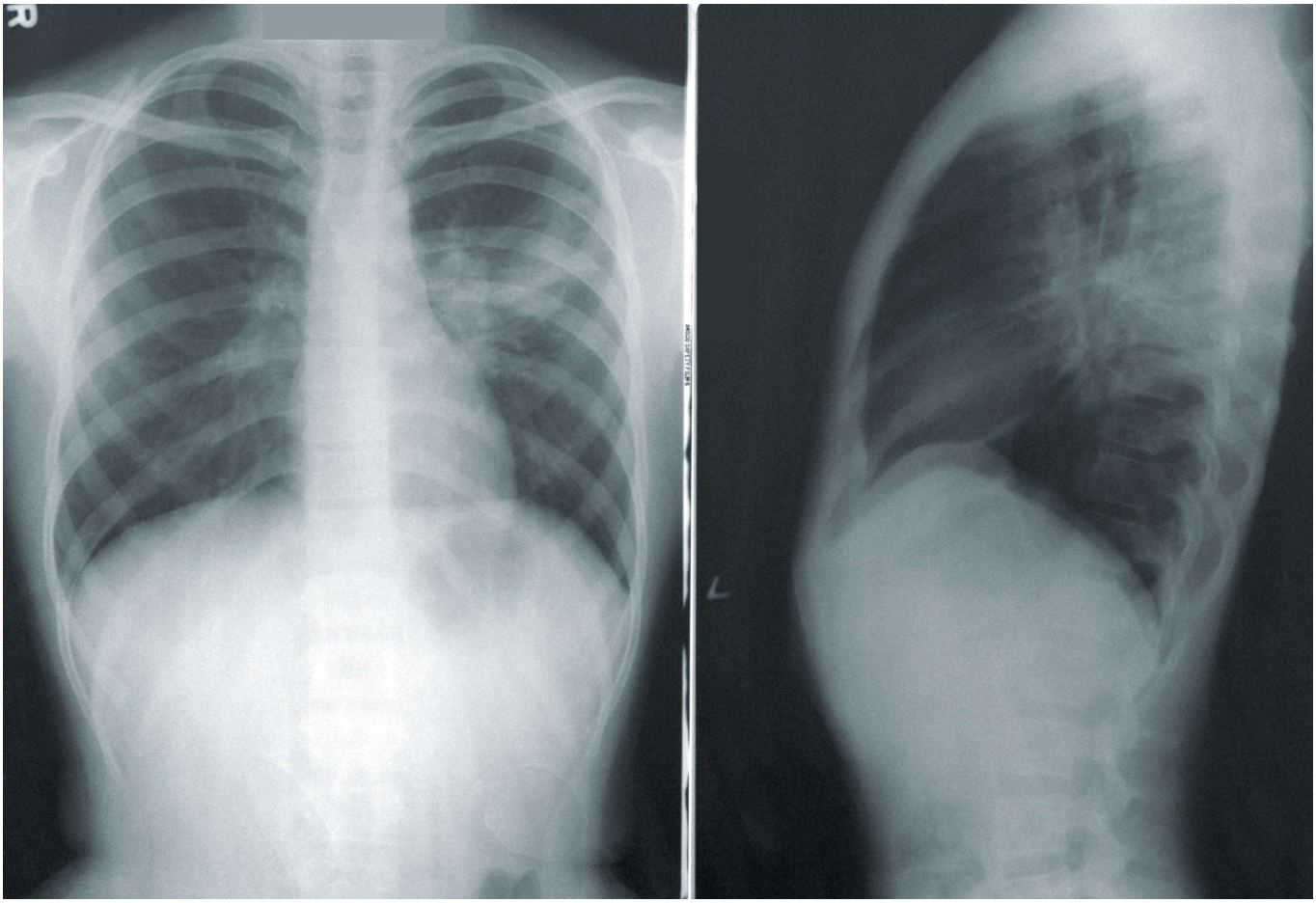
(Phase 4)

- Student Name: Tenicka Norwood
- Program Pace: self-paced
- Scheduled Project Review Time: Friday, September 15, 2023, 12 pm
- Instructor name: Morgan Jones
- Blog post Url: https://medium.com/mlearning-ai/fueling-student-success-1723abd2991b

Disclaimer:

This Jupyter notebook and its contents are **intended solely for educational purposes**. The included business case and the results of the deep learning models should not be interpreted as medical advice, and have not received endorsement or approval from any professional or medical organization.

The models and outcomes presented here are for illustrative purposes **only**. Users should **not** use these models or their outcomes for making real-world decisions without consulting appropriate domain experts and medical professionals. Any actions taken based on the information in this notebook are at the user's own risk. The author and contributors of this notebook disclaim any liability for the accuracy, completeness, or efficacy of the information provided.



Business Case:

- **Stakeholder**: Zephyr Health Services
- **Business Case**: I am a new data scientist at Zephyr Healthcare Solutions and have been tasked with leveraging deep learning to bolster the company's medical diagnostic systems. Zephyr's objective is to create an optimized Convolutional Neural Network (CNN) for precise pneumonia detection from chest X-ray images, aligning with our commitment to advancing diagnostic accuracy and patient care.

According to a 2022 report (click here for full report) by Johns Hopkins over **700 thousand children under 5** die from pneumonia each year.

Objectives: The main objectives of this project are:

- Develop a robust and efficient system for early childhood pneumonia detection using a Convolutional Neural Network(CNN) which detects the presence of pneumonia with a **high degree** of precision.
- Generate a system that can be validated and deployed across a variety of healthcare settings, to reach underserved populations.

Metrics

We will prioritize precision in this project over accuracy. We will also aim for balance between recall (sensitivity) while maintaining a high level of precision (specificity). With these objectives in mind, we aim to reduce the number of false positives and increase the model's ability to correctly identify patients with pneumonia. In this context, false positives could lead to unnecessary treatment or interventions.

- **True positives** are the cases when a model predicts positive instances correctly. Ex. patients predicted to have PNEUMONIA and do have PNEUMONIA

***False positives** are the cases when a model predicts positive instances incorrectly. Ex. patients predicted to have PNEUMONIA but the chest scan label is NORMAL.

***False negatives** are the cases when a model mispredicts negative instances. Ex. patients predicted to have NORMAL chest scans and do have PNEUMONIA.

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

A high precision indicates that when our model predicts the presence of PNEUMONIA, the patient will likely have PNEUMONIA.

$$Recall = \frac{True\ Positive}{False\ Negative + True\ Positive}$$

A high precision indicates that when our model predicts the presence of PNEUMONIA, the patient will likely have PNEUMONIA.

For our project we will use F1-score or the harmonic mean of recall and precision.

$$F1 - Score = \frac{2 \cdot True\ Positive}{2 \cdot True\ Positive + False\ Positive + False\ Negative}$$

Our model will be successful where:

$$F1 - Score >= .90$$

Data Understanding

Chest X-ray images (anterior-posterior) were choseN from cohorts of pediatric patients aged one to five years at the Guangzhou Women and Children's Medical Center in Guangzhou, China.

The images were obtained during routine clinical care. To ensure quality, all chest radiographs went through an initial screening to remove low-quality or unreadable scans.

Subsequently, two expert physicians graded the diagnoses of the images before they were used in training sets.

A third expert reviewed the validation set to mitigate any potential grading errors.

The dataset is orginally accredited to Kermany, Daniel; Zhang, Kang; Goldbaum, Michael (2018), “Labeled Optical Coherence Tomography (OCT) and Chest X-Ray Images for Classification”, Mendeley Data, V2, doi: 10.17632/rscbjbr9sj.2.

For ease of use we will use Kaggle to access a zip archive of this dataset later in this section.

Method:

In this project, I will use the **OSEMiN** pipeline to:

- Obtain → Import the data.
- Scrub → Manage the datatypes, and resolve missing data or duplicates.
- Explore → Identify patterns within the relationships between variables in the data.
- Model → Create a set of predictive models.
- iNterpret → Identify insights and create visualizations of findings.

For our model we could prioritize:

- accuracy - the proportion of correctly predicted labels out of all of the samples in the testing dataset.
- precision - measures the model's accuracy in predicting true positives as a proportion of all positives.
- recall- measures the model's ability to measure all positive instances correctly.
- f1-score - the harmonic mean of precision and recall, used when we want to consider both identifying true positives and minimizing false negatives.

Obtain

ObtainData Class Description

The `ObtainData` class is a utility class designed to retrieve datasets stored in Google Drive. This class provides an approach to mount Google Drive, verify the existence of a specified dataset folder, and extract its contents if necessary.

Features:

- Mounts Google Drive to the Colab environment, providing a user allows access to the files stored in thier Google Drive.
- Prompts the user to input the location of the dataset folder within their Google Drive
- Checks to see if the specified dataset folder exists and provides feedback to the user.
- If the dataset folder exists, this class provides a convenient way to access the dataset files.

Usage:

- Create an instance of the `ObtainData` class.
- Invoke the `get_dataset_location()` method to prompt the user to input the location of the dataset folder in their Google Drive.
- The class automatically verifies the existence of the specified folder and provides feedback accordingly.
- The dataset files can then be accessed and processed as needed.

```
!pip install lime
```

```
Requirement already satisfied: lime in /usr/local/lib/python3.10/dist-packages (0.2.0.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from lime) (3.7.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from lime) (1.23.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from lime) (1.11.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from lime) (4.66.1)
Requirement already satisfied: scikit-learn>=0.18 in /usr/local/lib/python3.10/dist-packages (from lime) (1.2.2)
Requirement already satisfied: scikit-image>=0.12 in /usr/local/lib/python3.10/dist-packages (from lime) (0.19.3)
Requirement already satisfied: networkx>=2.2 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.12->lime) (3.1)
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>=0.12->lime) (9.4.0
Requirement already satisfied: imageio>=2.4.1 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.12->lime) (2.31.3)
Requirement already satisfied: tifffile>=2019.7.26 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.12->lime) (2023.8.30)
Requirement already satisfied: PyWavelets>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.12->lime) (1.4.1)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.12->lime) (23.1)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.18->lime) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.18->lime) (3.2.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->lime) (1.1.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib->lime) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->lime) (4.42.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->lime) (1.4.5)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->lime) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib->lime) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib->lime) (1.16.0)
```

Import libraries and Visualization Packages

Importing libraries at the beginning allows access to modules and other tools throughout this project that help to make the tasks within this project manageable to implement. The main libraries used within this project include:

- `pandas` : a data analysis and manipulation library which allows for flexible reading, writing, and reshaping of data
- `numpy` : a key library that brings the computational power of languages like C to Python
- `matplotlib` : a comprehensive visualization library
- `seaborn` : a data visualization library based on matplotlib
- `tensorflow` : an open-source machine learning library developed by Google that enables the creation and training of machine learning models through computational graphs
- `keras` a key component of TensorFlow's ecosystem that provides a modular interface to design, build and train deep learning models
- `sklearn` : a machine learning library for data processing and modeling.
- `lime` Local Interpretable Model-agnostic Explanations is a library designed to provide clear explanations for machine learning model predictions.

```
# Import libraries and visualization packages
import lime
from lime import lime_base, lime_image
from skimage.segmentation import mark_boundaries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
%matplotlib inline
import cv2, time
import datetime
```

```
import datetime
import os
import shutil
import math
import tensorflow as tf
from keras.models import Sequential, Model
from keras.layers import Input, Flatten, Dropout, Dense
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG19, InceptionV3, resnet50
from sklearn.metrics import classification_report, confusion_matrix
from mlxtend.plotting import plot_confusion_matrix
```

Check library versions

Ensuring the reproducibility of results stands as a cornerstone of effective modeling. In the ever-evolving landscape of information, the refinement of models remains integral to advancing science. An inherent facet of collaborative success lies in transparently conveying the tool versions employed in model generation. This practice not only facilitates seamless collaboration but also equips fellow researchers with a foundation for their own endeavors to replicate this work. As such, we document the versions of key libraries so that others may replicate and explore this data.

```
# Check Pandas version
print("Pandas version")
pd.__version__
```

```
Pandas version
'1.5.3'
```

```
# Check Numpy version
print("Numpy version")
np.__version__
```

```
Numpy version
'1.23.5'
```

```
# Check Seaborn version
print("Seaborn version")
sns.__version__
```

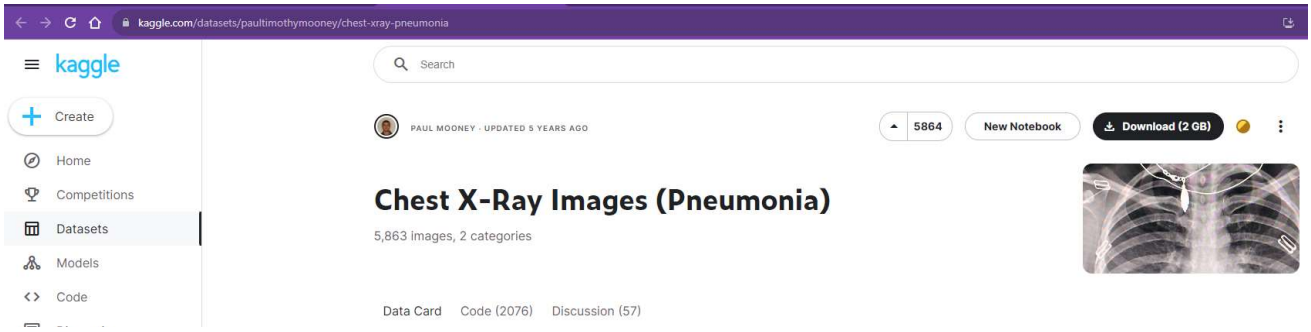
```
Seaborn version
'0.12.2'
```

```
# Check Tensorflow version
print("Tensorflow version")
tf.__version__
```

```
Tensorflow version
'2.13.0'
```

Downloading and Preparing Pneumonia Dataset for Deep Learning Analysis

1. **Create or Log in to Your Kaggle Account:** If you do not already have a Kaggle account, create one. If you have an account log in.
2. **Access the Pneumonia Dataset:** Go to the following direct link to access Paul Timothy Mooney's Pneumonia dataset on Kaggle: [Pneumonia Dataset]
3. **Download the Dataset:** On the dataset page, you will see a "Download" button. Click on it to download the dataset.



The dataset is approximately **2GB**.

4. **Unzip the file Add the unzipped archive to your Google Drive:** After downloading and unzipping the dataset you'll have a folder named 'archive'. This folder contains the dataset. To use this notebook you will need to provide the location of the .zip file in your Google Drive.

5. **Run the next two cells without making any chages** Mount your google drive to allow colab access to your google drive. Unzip the file so you can use the contents of this notebook. The file should include train, test and val folders that contain subfolders with both NORMAL and PNEUMONIA images.

```
# Run cell without making any changes
from google.colab import drive

class ObtainData:
  """
  A class to obtain dataset location from Google Drive.

  Usage:
  data_obtainer = ObtainData()
  dataset_location = data_obtainer.get_dataset_location()
  """

  def __init__(self):
      self.drive_mounted = False

  def mount_drive(self):
    """
    Mounts Google Drive to '/content/drive'.
    """
    drive.mount('/content/drive')
    self.drive_mounted = True

  def get_dataset_location(self):
    """
    Prompts the user to enter the location of the datset folder in their Google Drive.
    Returns the full file path of the dataset location.
    """

    while True:
      # Munt Google Drive if not already mounted
      if not self.drive_mounted:
        self.mount_drive()

      # Provide a template for the user input
      example_input ="/MyDrive/Your_Folder_name/"
      dataset_location = input(f"Enter the location of the dataset folder in your Google Drive (e.g., {example_input}):")
      file_path = f'/content/drive{dataset_location}'

      # Check if the file exists
      if os.path.exists(file_path):
        print(f" The file '{dataset_location}' exists in your Google Drive.")
        return file_path
      else:
        print(f" The file '{dataset_location}' does not exist in your Google Drive. Please try again.")
```

In this section, we will set up a convenient way to access the dataset using the `ObtainData` class. This will help us to smoothly acquire the dataset's location from the connected Google Drive by interacting with the user using prompts. This step ensures that the required data is available for analysis.

```
# Run cell without making any changes
# Create an instance of the ObtainData class
data_obtainer = ObtainData()

# Get the dataset location
dataset_location = data_obtainer.get_dataset_location()
print(f"Datatset location")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
Enter the location of the dataset folder in your Google Drive (e.g., /MyDrive/Your_Folder_name/):/MyDrive/classification/chest_xray
 The file '/MyDrive/classification/chest_xray' exists in your Google Drive.
Datatset location
```

Next we check the current working directory and ensure that we are working in the dataset's location.

```
print(os.getcwd())
```

```
/content
```

```
os.chdir(dataset_location)
print(os.getcwd())
```

```
/content/drive/MyDrive/classification/chest_xray
```

DatasetPaths Class Description

The `DatasetPaths` class is a utility class for managing paths within a dataset. This class simplifies file path retrieval and ensures efficient data handling. It allows users to access specific file paths based on set names and class names.

Features:

- Defines attributes for different subsets(e.g. train, test, val) and class names(e.g. NORMAL and PNEUMONIA) in a dataset.
- Enables streamlined access to file paths for specific subsets and classes, reducing the complexity of file handling
- Offers a structured approach to organize and retrieve file paths, enhancing data manipulation and analysis.

Usage:

- Create an instance of the `DatasetPaths` class
- Utilize the attributes defined in the class to easily access file paths for different subsets and classes within the dataset.

```
import os
import shutil
import random
class DatasetPaths:
  """
  Helper class to manage paths for different sets and classes of the dataset
  """

  def __init__(self, base_location):
    self.base_location = base_location
    self.class_names = ['NORMAL', 'PNEUMONIA']
    self.set_names = ['train', 'test', 'val']

  def get_single_path(self, set_name):
      return os.path.join(self.base_location, set_name)
  def get_path(self,set_name, class_name):
    """
    Get the path for a specific set and class.

    Parameters:
        set_name(str): Name of the dataset set('train', 'test', 'val')
        class_name(str): Name of the class ('NORMAL' or 'PNEUMONIA')

    Returns:
        path(str): Path to the specified set and class.
    """
    return f"{self.base_location}/{set_name}/{class_name}"

  def get_all_paths(self):
    """
    Get a dictionary containing all paths for the dataset.

    Returns:
      paths(dict): Dictionary containing paths for each set and class.
    """

    paths = {}
    for set_name in self.set_names:
        paths[set_name] = {}
        for class_name in self.class_names:
          paths[set_name][class_name] = self.get_path(set_name, class_name)
    return paths

  def move_images_to_validation(self):
      """
      Move 20% of the images from the training set to the validation set, preserving class subfolders.
      """
      # Define the source (training) and destination (validation) paths
      src_path = self.get_single_path('train')
      dst_path = self.get_single_path('val')

      for class_name in self.class_names:
          class_src_path = self.get_path('train', class_name)
          class_dst_path = self.get_path('val', class_name)

          # Create the class subfolders in the validation set if they don't exist
          os.makedirs(class_dst_path, exist_ok=True)

          class_images = os.listdir(class_src_path)
          num_images_to_move = int(0.2 * len(class_images))

          random_images_to_move = random.sample(class_images, num_images_to_move)

          for image in random_images_to_move:
```

```
            src = os.path.join(class_src_path, image)
            dst = os.path.join(class_dst_path, image)
            shutil.move(src, dst)
```

Here we create an instance of `DatasetPaths` class. This helps us establish the base location of the dataset in Google Drive. We use this to identify the location.

```
# Provide the base location of the dataset in Google Drive
base_dataset_location = dataset_location

# Create an instance of the DatasetPaths class
dataset_paths = DatasetPaths(base_dataset_location)

# Get all paths using the DatasetPaths instance
all_paths = dataset_paths.get_all_paths()

# Print the paths
for set_name in dataset_paths.set_names:
    for class_name in dataset_paths.class_names:
        print(f"{set_name.capitalize()} {class_name}: {all_paths[set_name][class_name]}")
```

```
Train NORMAL: /content/drive/MyDrive/classification/chest_xray/train/NORMAL
Train PNEUMONIA: /content/drive/MyDrive/classification/chest_xray/train/PNEUMONIA
Test NORMAL: /content/drive/MyDrive/classification/chest_xray/test/NORMAL
Test PNEUMONIA: /content/drive/MyDrive/classification/chest_xray/test/PNEUMONIA
Val NORMAL: /content/drive/MyDrive/classification/chest_xray/val/NORMAL
Val PNEUMONIA: /content/drive/MyDrive/classification/chest_xray/val/PNEUMONIA
```

Next let's leverage the methods and attributes of the `DatasetPaths` class to obtain paths and print the number of images for each set and each class.

```
# Create a dictionary to store the number of images in each class for each set
num_images_dict = {}

# Loop through sets and classes to count images
for set_name in dataset_paths.set_names:
    num_images_dict[set_name] = {}
    for class_name in dataset_paths.class_names:
        # Get the path using the DatasetPaths class
        path = dataset_paths.get_path(set_name, class_name)

        # Count the number of images in the directory
        num_images = len(os.listdir(path))

        # Store the count in the num_images_dict
        num_images_dict[set_name][class_name] = num_images

        # Print the count
        print(f" {set_name.capitalize()} {class_name}: {num_images} images")
```

```
Train NORMAL: 859 images
Train PNEUMONIA: 2480 images
Test NORMAL: 234 images
Test PNEUMONIA: 390 images
Val NORMAL: 490 images
Val PNEUMONIA: 1403 images
```

```
# Provide the base location of the dataset in Google Drive
base_dataset_location = dataset_location

# Create an instance of the DatasetPaths class
dataset_paths = DatasetPaths(base_dataset_location)

# Move images from training to validation set
dataset_paths.move_images_to_validation()
```

We made our validation set a bit more comparable to to our test set by taking 20% of our training set and appending it to our original validation set since the original validation set only had 16 images.

```
# Create a dictionary to store the number of images in each class for each set
num_images_dict = {}

# Loop through sets and classes to count images
for set_name in dataset_paths.set_names:
```

```
    num_images_dict[set_name] = {}
    for class_name in dataset_paths.class_names:
        # Get the path using the DatasetPaths class
        path = dataset_paths.get_path(set_name, class_name)

        # Count the number of images in the directory
        num_images = len(os.listdir(path))

        # Store the count in the num_images_dict
        num_images_dict[set_name][class_name] = num_images

        # Print the count
        print(f" {set_name.capitalize()} {class_name}: {num_images} images")
```

```
Train NORMAL: 859 images
Train PNEUMONIA: 2480 images
Test NORMAL: 234 images
Test PNEUMONIA: 390 images
Val NORMAL: 490 images
Val PNEUMONIA: 1403 images
```

In this section, we used the `DatasetPaths` class to calculate and display the distributeion of images across different sets and classess within the dataset. The output displays the number of images for each combination of training, testing, and validation sets along with the two class categories: 'NORMAL' and 'PNEUMONIA'. Next, we will create a visualization to get a quick view of the dataset's composition.

ClassDistributionPlot Class Description

The `ClassDistributionPlot` class is a helper class designed to create a bar plot to visualize the distribution of classes across different dataset sets.

Features:

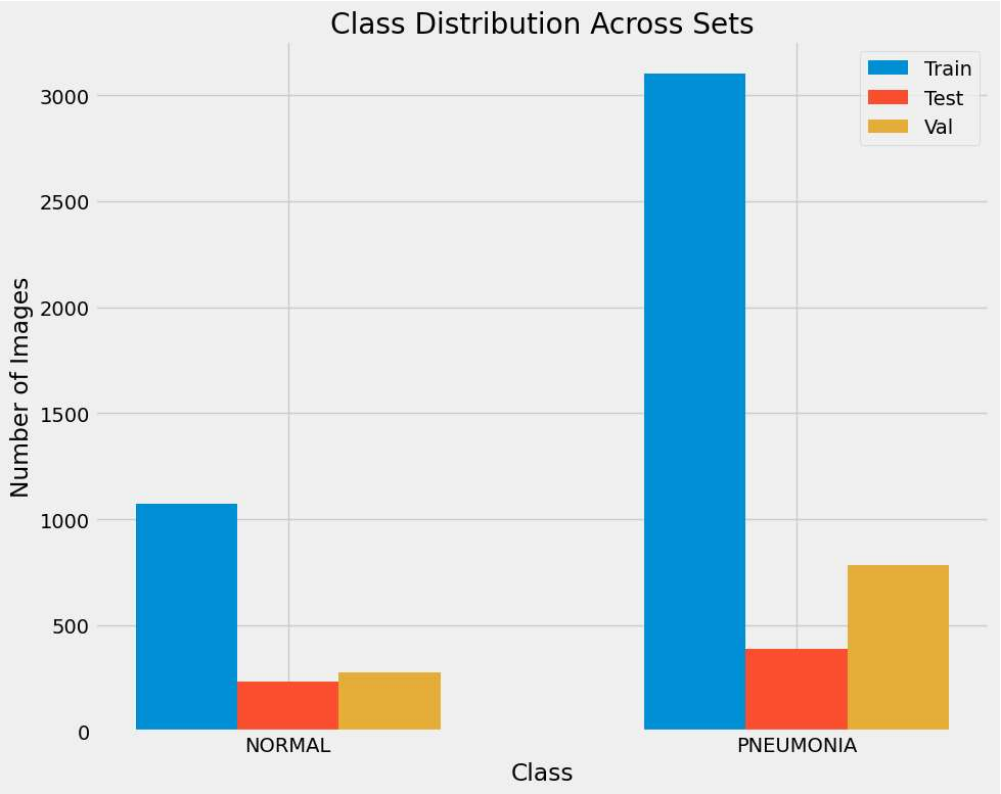
- Uses the `DatasetPaths` class to manage and access dataset paths.
- Accepts a dictionary containing the count of images per class for each dataset set.
- Aligns bars for each class for comparison
- Adopts the `fivethirthyeight` style for consistency.

Usage:

1. Create an instance of the `DatasetPaths` class.
2. Create an instance of the `ClassDistributionPlot` class, providing a dictionary with the class distribution data.
3. Use the `plot()` method to generate bar plots that illustrate class distribution across different sets.

```
class ClassDistributionPlot:
    """
    Helper class to create class distribution plots for different dataset sets.
    """

    def __init__(self, dataset_paths, set_names, class_names):
        """
        Initialize the ClassDistributionPlot instance.

        Parameters:
            dataset_paths (DatasetPaths): An instance of the DatasetPaths class.
            set_names (list): List of dataset set names ('train', 'test', 'val').
            class_names (list): List of class names.
        """
        self.dataset_paths = dataset_paths
        self.set_names = set_names
        self.class_names = class_names

    def plot_distribution(self):
        """
        Create and display class distribution plots for different dataset sets.
        """
        plt.style.use('fivethirtyeight')
        plt.figure(figsize=(10, 8))
        x = range(len(self.class_names))

        for i, set_name in enumerate(self.set_names):
            num_images_per_class = [
                len(os.listdir(self.dataset_paths.get_path(set_name, class_name)))
                for class_name in self.class_names
            ]
            plt.bar([j + i * 0.2 for j in x], num_images_per_class, width=0.2,
                    align='center', label=set_name.capitalize())

        plt.xlabel('Class')
        plt.ylabel('Number of Images')
        plt.title('Class Distribution Across Sets')
```

```
        plt.xticks([i + 0.2 for i in x], self.class_names)
        plt.legend()

        plt.tight_layout()
        plt.show()


# Extract class names and corresponding number of images for each class
set_names = dataset_paths.set_names
class_names = dataset_paths.class_names
num_images_per_class = {
    'train': [num_images_dict['train'][class_name] for class_name in class_names],
    'test': [num_images_dict['test'][class_name] for class_name in class_names],
    'val': [num_images_dict['val'][class_name] for class_name in class_names]
}

# Create an instance of the ClassDistributionPlot class
distribution_plot = ClassDistributionPlot(dataset_paths, set_names, class_names)

# Plot and display class distribution
distribution_plot.plot_distribution()
```



There is a noticeable class imbalance in the training image set, with the PNEUMONIA class containing almost 3 times as many images as the NORMAL class. There are now 20% of the training images appended to the original validation set which had equal number(8) of each class in the validation set since the orginal validation set was so much smaller than either the testing or training set.

```
from IPython.lib.display import exists
import random

class ImageBalancer:
    """
    Helper class to balance the number of images between different classes in a dataset.
    """

    def __init__(self, dataset_paths, class_names, set_names, target_set_name):
        """
        Initialize the ImageBalancer instance.

        Parameters:
```

```
        dataset_paths (DatasetPaths): An instance of the DatasetPaths class.
        class_names (list): List of class names.
        set_names (list): List of dataset set names ('train', 'test', 'val').
        target_set_name (str): The name of the dataset set to balance.
    """
    self.dataset_paths = dataset_paths
    self.class_names = class_names
    self.set_names = set_names
    self.target_set_name = target_set_name

def balance_images(self):
    """
    Balance the number of images between classes in the target dataset set.
    """
    target_set_path = self.dataset_paths.get_path(self.target_set_name, '')

    # Find the desired number of images as the minimum number of images in subfolders of each class
    desired_num_images = min(
        [len(os.listdir(self.dataset_paths.get_path(self.target_set_name, class_name))) for class_name in self.class_names]
    )

    for class_name in self.class_names:
        class_images = os.listdir(self.dataset_paths.get_path(self.target_set_name, class_name))

        # Check if the number of images in this class exceeds desired_num_images
        if len(class_images) > desired_num_images:
            random_class_images = random.sample(class_images, desired_num_images)

            new_class_path = os.path.join(target_set_path, class_name + '_resampled')
            os.makedirs(new_class_path, exist_ok=True)

            for image in random_class_images:
                src = os.path.join(self.dataset_paths.get_path(self.target_set_name, class_name), image)
                dst = os.path.join(new_class_path, image)
                shutil.copy(src, dst)
    # Create a new directory to store the balanced training images
    new_train_path = os.path.join(self.dataset_paths.base_location, 'train_resampled')
    os.makedirs(new_train_path, exist_ok=True)

    # Copy the balanced PNEUMONIA images to the new balanced training directory
    shutil.copytree(self.dataset_paths.get_path(self.target_set_name, 'PNEUMONIA_resampled'),
                    os.path.join(new_train_path, 'PNEUMONIA'))

    # Copy the original NORMAL images to the new balanced training directory
    shutil.copytree(self.dataset_paths.get_path(self.target_set_name, 'NORMAL'),
                    os.path.join(new_train_path, 'NORMAL'))
```

Next we created an instance of the `ImageBalancer` class, which is a custom helper class designed to balance the number of images between different classes in a dataset. This instance is named `image_balancer`.

Overview

The `ImageBalancer` class is a useful tool for addressing class imbalance issues in datasets commonly encountered in machine learning and computer vision tasks. By evenly distributing the number of images across classes within a specified dataset set (e.g., 'train', 'test', 'val'), it helps improve the training and evaluation of machine learning models.

Initialization

To create the `image_balancer` instance, we provided the following parameters during initialization:

- `dataset_paths`: An instance of the `DatasetPaths` class, which likely contains paths to the dataset.
- `class_names`: A list of class names, representing the different categories or classes in the dataset.
- `set_names`: A list of dataset set names, typically including 'train', 'test', and 'val'.
- `target_set_name`: The name of the dataset set that we want to balance, in this case, 'train'.

Balancing Images

The primary purpose of this instance is to balance the number of images in the 'train' dataset set. It accomplishes this by ensuring that each class within the 'train' set has an equal number of images, which is determined by the class with the fewest images.

The `balance_images` method of the `image_balancer` instance performs the following steps:

1. It calculates the desired number of images as the minimum number of images in subfolders of each class.
2. For each class, if the number of images exceeds the desired number, it randomly selects a subset of images to keep.
3. The selected images are then copied to a new directory, named after the class with '_resampled' appended, within the 'train' set.
4. Finally, it creates a new directory called 'resampled_train' and copies the balanced 'PNEUMONIA' and original 'NORMAL' images into this directory.

By using this instance, we ensure that our 'train' dataset set is balanced, providing a more equitable representation of different classes and potentially improving the performance of our machine learning models.

Let's proceed with the image balancing process using the `image_balancer` instance.

```
# Create an instance of the ImageBalancer class
image_balancer = ImageBalancer(dataset_paths, class_names, set_names, target_set_name='train')

# Balance the images in the target dataset
image_balancer.balance_images()
```

Next, we will visualize the distribution of classes within a specific dataset set, specifically the 'resampled_train' set. To achieve this, we'll be using an instance of the `ClassDistributionPlot` class, which is a custom class designed for visualizing class distributions in a dataset. The code snippet below demonstrates how we set up and utilize this class instance.

Set and Class Names

Before proceeding with the visualization, we define the following essential components:

- **Set Names**: We specify the dataset set we want to visualize, which, in this case, is 'resampled_train.' This set typically represents our balanced training data.
- **Class Names**: We provide a list of class names that correspond to the categories or classes within the 'resampled_train' dataset set. In this example, we have two classes: 'NORMAL' and 'PNEUMONIA.'

Class Distribution Plot Instance

To create a visual representation of the class distribution, we instantiate the `ClassDistributionPlot` class. This class is designed to take the dataset paths, set names, and class names as input and generate a meaningful plot illustrating the distribution of samples across classes.

```
#Define the set and class names
set_names = ['train_resampled']
class_names = ['NORMAL', 'PNEUMONIA']

# Create an instance of the ClassDistributionPlot class
distribution_plot = ClassDistributionPlot(dataset_paths, set_names, class_names)

# Plot the distribution of the resampled training data
distribution_plot.plot_distribution()
```

Class Distribution Across Sets

Next we will implement The `ScrubData` class for preprocessing our images.

Key Features

The `ScrubData` class provides the following key features:

1. **Count Images**: It counts the number of images in each class for different sets (e.g., training, testing, validation) within your dataset.
2. **Create Data Generator**: It creates data generators using the Keras `ImageDataGenerator` class, simplifying the configuration of data augmentation and normalization.
3. **Plot Sample Images**: It enables you to visualize sample images from your dataset for a specified class, aiding in data exploration and understanding.

Usage

Here's how you can use the `ScrubData` class:

- Initialize the ScrubData instance with your dataset location and optional parameters
  - `base_dataset_location = '/path/to/your/dataset'`
  - `scrubber = ScrubData(base_dataset_location)`
- Count images in different sets
- Create data generators
- Plot sample images for data exploration

Scrub

The `ScrubData` class provides methods for handling image data, including counting images, creating data generators, and visualizing sample images. It seems to be designed for preprocessing and exploring image datasets, particularly for a binary classification problem involving 'NORMAL' and 'PNEUMONIA' classes.

```
import random
from keras.preprocessing.image import ImageDataGenerator

class ScrubData:
    """
    The ScrubData class is responsible for managing and processing the dataset for training and testing.

    Parameters:
        base_dataset_location (str): The base directory path of the dataset.
        image_size (tuple): The dimensions to which the images will be resized.
        class_names (list): List of class names in the dataset.
        batch_size_train (dict): Number of images in each class for the training set.
        batch_size_test (dict): Number of images in each class for the test set.
        batch_size_val (dict): Number of images in each class for the validation set.

    Methods:
        count_images(set_name): Counts the number of images in each class within a specified set.
        create_data_generator(directory, batch_size): Creates a data generator for a specified directory.
        plot_sample_images(data_generator, class_name, num_images, num_rows, num_cols, figsize):
            Plots a set of sample images from the data generator.
    """
    # Constructor method to initialize class attributes
    def __init__(self, base_dataset_location, image_size=(224, 224), class_names=['NORMAL', 'PNEUMONIA']):
        self.base_dataset_location = base_dataset_location
        self.image_size = image_size
        self.class_names = class_names

        # Calculate batch sizes for different sets
        self.batch_size_train = self.count_images('train_resampled')
        self.batch_size_test = self.count_images('test')
        self.batch_size_val = self.count_images('val')

    # Method to count the number of images in each class within a set
    def count_images(self, set_name):
        counts = {}
        set_path = os.path.join(self.base_dataset_location, set_name)

        for class_name in self.class_names:
            class_path = os.path.join(set_path, class_name)
```

```
        if os.path.exists(class_path):
            num_images = len(os.listdir(class_path))
            counts[class_name] = num_images

    return counts
# Method to create a data generator for a specified directory
def create_data_generator(self, directory, batch_size):
    normalization_params = {
        'NORMAL': {'rescale': 1.0 / 255.0},
        'PNEUMONIA': {'rescale': 1.0 / 255.0}
    }
    # Specify class mode for the data generator
    class_mode = 'binary'

    return ImageDataGenerator(**normalization_params[class_name]).flow_from_directory(
        directory,
        target_size=self.image_size,
        batch_size=batch_size,
        class_mode=class_mode,
        seed=42
    )

# Method to plot sample images from a data generator
def plot_sample_images(self, data_generator, class_name, num_images=16, num_rows=4, num_cols=4, figsize=(10, 10)):
    class_index = self.class_names.index(class_name)
    class_indices = [index for index, label in enumerate(data_generator.labels) if label == class_index]
    random_class_indices = random.sample(class_indices, num_images)

    plt.figure(figsize=figsize)
    plt.suptitle(class_name, fontsize=16)

    for i, index in enumerate(random_class_indices):
        plt.subplot(num_rows, num_cols, i + 1)
        image = data_generator._get_batches_of_transformed_samples([index])[0][0]
        plt.imshow(image)
        plt.axis('off')

    # Add an empty subplot for the white space
    plt.subplot(num_rows, num_cols, num_images + 1)
    plt.axis('off')

    plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    plt.show()
```

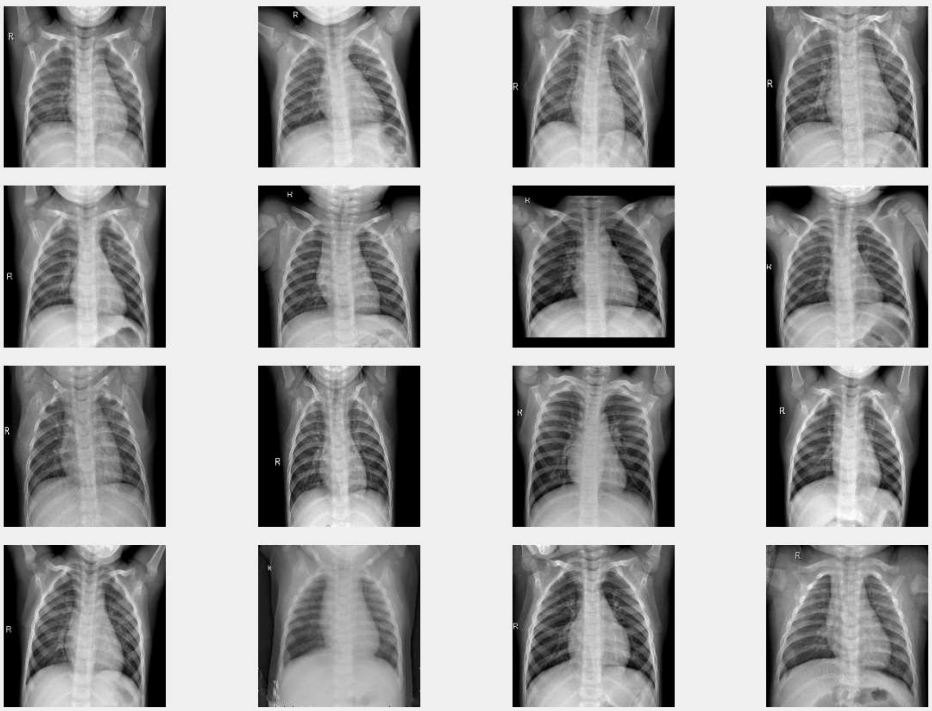
Explore

In this section we will perform visualizations on data that we preprocessed using the `ScrubData` class. To create an array of NORMAL and PNEUMONIA images and to review the pixel intensities of sample images from each class.

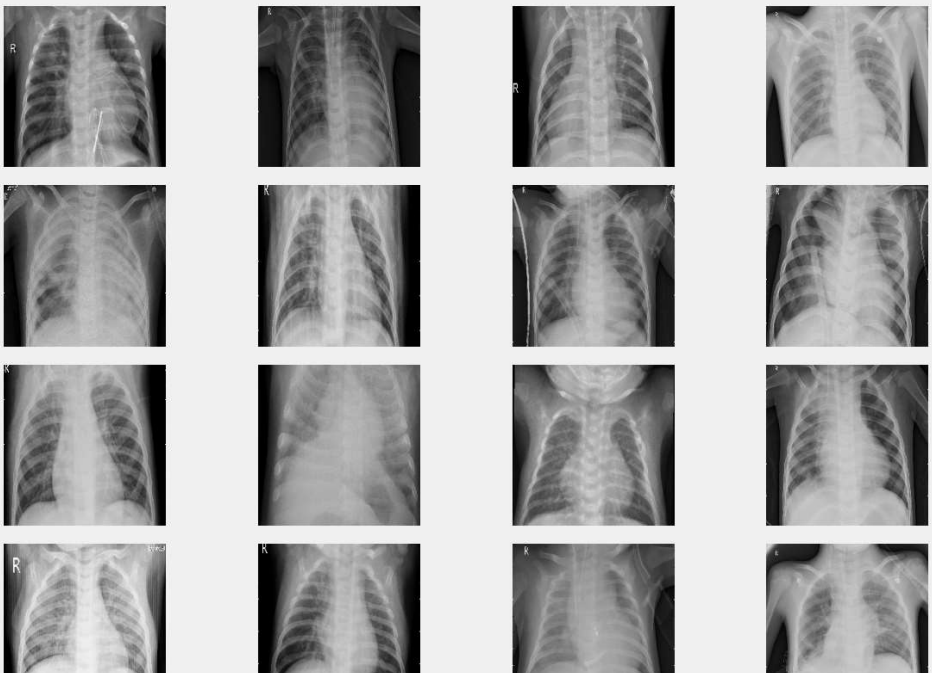
```
if __name__ == "__main__":
    scrubber = ScrubData(base_dataset_location)

    train_generator = scrubber.create_data_generator('train_resampled', scrubber.batch_size_train)
    test_generator = scrubber.create_data_generator('test', scrubber.batch_size_test)
    val_generator = scrubber.create_data_generator('val', scrubber.batch_size_val)

    # Plot sample images
    scrubber.plot_sample_images(train_generator, class_name='NORMAL', num_images=16, num_rows=5, num_cols=4, figsize=(12, 12))
    scrubber.plot_sample_images(train_generator, class_name='PNEUMONIA', num_images=16, num_rows=5, num_cols=4, figsize=(12, 12))
```

```
Found 2146 images belonging to 2 classes.
Found 624 images belonging to 2 classes.
Found 1059 images belonging to 2 classes.
```

NORMAL

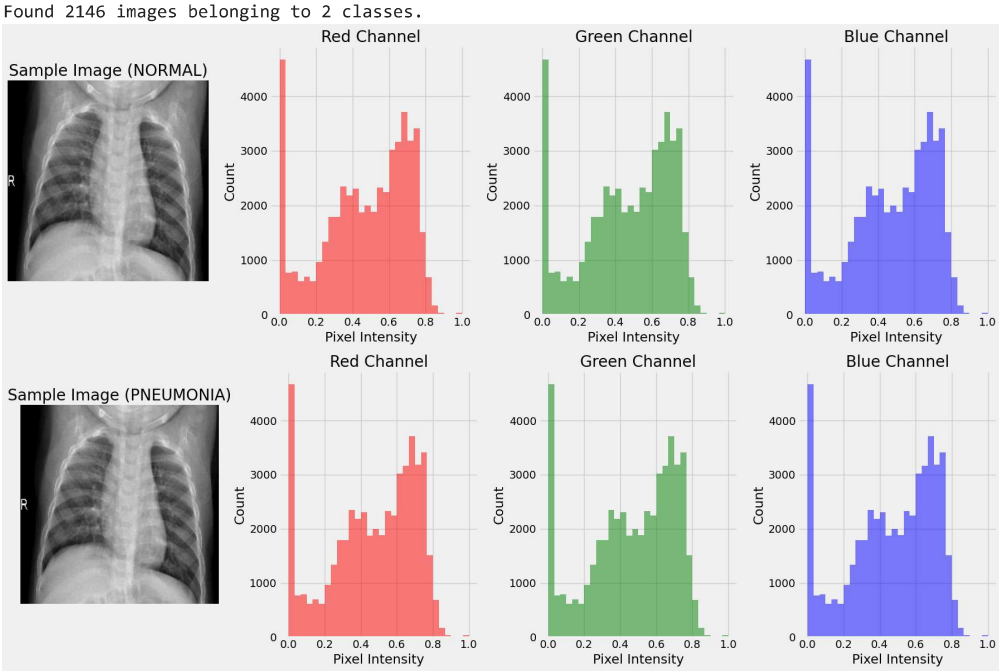


PNEUMONIA



```
# Create data generators for each class
normal_generator = scrubber.create_data_generator('train_resampled', scrubber.batch_size_train['NORMAL'])
pneumonia_generator = scrubber.create_data_generator('train_resampled', scrubber.batch_size_train['PNEUMONIA'])

# Get a batch of sample data from each generator
normal_batch_images, _ = next(normal_generator)
pneumonia_batch_images, _ = next(pneumonia_generator)

# Function to plot sample image and histograms
def plot_sample_with_histograms(sample_image, class_name):
    plt.figure(figsize=(18, 6))

    # Plot sample image
    plt.subplot(1, 4, 1)
    plt.imshow(sample_image)
    plt.title(f'Sample Image ({class_name})')
    plt.axis('off')

    # Plot histograms
    for i, channel in enumerate(['Red', 'Green', 'Blue']):
        plt.subplot(1, 4, i + 2)
        plt.hist(sample_image[:, :, i].flatten(), bins=30, color=channel.lower(), alpha=0.5)
        plt.xlabel('Pixel Intensity')
        plt.ylabel('Count')
        plt.title(f'{channel} Channel')

    plt.tight_layout()
    plt.show()

# Plot sample image with histograms for NORMAL class
plot_sample_with_histograms(normal_batch_images[0], 'NORMAL')

# Plot sample image with histograms for PNEUMONIA class
plot_sample_with_histograms(pneumonia_batch_images[0], 'PNEUMONIA')
```

```
Found 2146 images belonging to 2 classes.
Found 2146 images belonging to 2 classes.
```



The sample images both appear to be grayscale, so let's move on to create a model.

Model

Brief introduction to CNNs

The way animals understand images is a fascinating process. According to Sarvamangala and Kulkarni (click here for full text), a Convolutional Neural Networks is a deep learning algorithm that is inspired by the visual cortex of animal brains and aims to imitate the visual machinery of animals. CNNS have convolution layers, activation functions, pooling and fully connected laryers as their building blocks and are made of convolution have learnable weights and biases similar to neurons.

Baseline Model Introduction

We are using a baseline Convolutional Neural Network (CNN) model to classify images as either being in one of two classes: PNEUMONIA or NORMAL. This model serves as a starting point for image classification with deep learning project and helps to establish a performance baseline against which we can compare more advanced models or techniques.

Model Description

- **Name**: Baseline CNN Model
- **Purpose**: The baseline model is designed to correctly classify images into one of two classes *'NORMAL'* or *'PNEUMONIA'*
- **Architecture**: The model architecture consists of several layers, including convolutional layers for feature extraction, pooling layers for downsampling, and fully connected layers for classification.
- **Input Shape**: The model expects input data in the shape of [(224, 224, 3)]
- **Output**: The model produces 2 output classes, and it uses the softmax activation function for multi-class classification.

Key Components

- **Convolutional Layers**: The model includes 3 convolutional layers with ReLU activation functions, which learn spatial features from the input data.
    1. conv2d_6: This is the first convolutional layer with 32 filters.
    2. conv2d_7: This is the second convolutional layer with 64 filters.

3. conv2d_8: This is the third convolutional layer with 128 filters.

- **Pooling Layers**: Max-pooling layers are used for spatial down-sampling, reducing the spatial dimensions of the feature maps. There are 3 max-pooling layers:

  1. max_pooling2d_6:
     - Input Shape: (None, 222, 222, 32)
     - Output Shape: (None, 111, 111, 32)
     - Description: This max-pooling layer follows the first convolutional layer (conv2d_6). It takes the feature maps produced by conv2d_6 with a size of (222, 222, 32) and reduces their spatial dimensions by a factor of 2. As a result, the output shape becomes (111, 111, 32).
  2. max_pooling2d_7:
     - Input Shape: (None, 109, 109, 64)
     - Output Shape: (None, 54, 54, 64)
     - Description: This max-pooling layer follows the second convolutional layer (conv2d_7). It takes the feature maps produced by conv2d_7 with a size of (109, 109, 64) and reduces their spatial dimensions by a factor of 2. The output shape becomes (54, 54, 64).
  3. max_pooling2d_8:
     - Input Shape: (None, 52, 52, 128)
     - Output Shape: (None, 26, 26, 128)
     - Description: This max-pooling layer follows the third convolutional layer (conv2d_8). It takes the feature maps produced by conv2d_8 with a size of (52, 52, 128) and reduces their spatial dimensions by a factor of 2. The output shape becomes (26, 26, 128)

- **Flatten Layer**: After feature extraction, the flatten layer transforms the 2D feature maps into a 1D vector.
- **Fully Connected Layers**: Dense layers with ReLU activation functions are used for classification. ReLU is an activation function commonly used in neural networks. It returns zero for negative input values and passes through positive input values directly. Dense layers with ReLU activation functions are typically used towards the end of the neural network architecture. They take the output from the preceding layers (which represent abstract features learned from the data) and apply a nonlinear transformation to it. This helps in learning complex decision boundaries necessary for classification tasks.
- **Dropout**: Dropout layers are added to mitigate overfitting during training.

Data Preprocessing

- **Data Generators**: We use data generators to load and preprocess training and testing data.
- **Data Augmentation**: Data augmentation techniques, such as rotation, shifting, shearing, and flipping, are applied to increase the diversity of training examples.

Training

- **Optimizer**: The model is trained using the RMS(Root Mean Square Propagation) optimizer which is supposed to adapt the learning rate for each parameter during training.
- **Loss Function**: We use sparse categorical cross-entropy as the loss function for this multi-class classification task.
- **Metrics**: During training and evaluation, we monitor accuracy and other relevant metrics.

Training History

- We train the baseline model for 25 epochs and monitor its performance on both the training and validation datasets.
- The training history includes metrics like accuracy, loss, precision, recall, and F1-score.

The baseline model serves as a foundation for our project. We will explore various strategies to improve its performance, such as hyperparameter tuning, model architecture adjustments, and data augmentation techniques. The baseline model provides a starting point for our project, and its performance metrics will be used as a reference to evaluate the effectiveness of future improvements. As we proceed, we aim to develop more advanced models and techniques to achieve higher accuracy and robustness in solving the task at hand.

```
# Create an instance of the DatasetPaths class
dataset_paths = DatasetPaths(base_dataset_location)

# Define the set name and class name
train_set_name = 'train_resampled'
test_set_name = 'test'
val_set_name = 'val'

# Get the complete train_data_path using the DatasetPaths class
train_data_path = dataset_paths.get_single_path(train_set_name)

# Get the complete test_data_path using the DatasetPaths
test_data_path = dataset_paths.get_single_path(test_set_name)
```

```
# Get the complete val_data_path using the DatasetPaths class
val_data_path = dataset_paths.get_single_path(val_set_name)


import tensorflow as tf
from keras.models import Sequential
from keras.layers import Conv2D, SeparableConv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization
from keras.metrics import Precision, Accuracy, Recall
# Set image dimensions and classes
input_shape = (224, 224, 3)
num_classes = 2

# Create data generators for train and test sets
train_generator = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.1,
    brightness_range=[0.2,1.0],
    horizontal_flip=True
)

test_generator = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255)
val_generator = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255)

train_data = train_generator.flow_from_directory(
    train_data_path ,
    target_size=input_shape[:2],
    batch_size=32,
    class_mode='binary'
)

test_data = test_generator.flow_from_directory(
    test_data_path,
    target_size=input_shape[:2],
    batch_size=32,
    class_mode='binary'
)

valdata = val_generator.flow_from_directory(
    val_data_path ,
    target_size=input_shape[:2],
    batch_size=32,
    class_mode='binary'
)
# Create the baseline CNN model
baseline_model = Sequential([
    Conv2D(16, (3,3), activation='relu', input_shape=input_shape),
    MaxPooling2D(2,2),
    Conv2D(32, (3,3), activation='relu'),
    MaxPooling2D(2,2),
    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D(2,2),
    Conv2D(128, (3,3), activation='relu'),
    MaxPooling2D(2,2),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(1, activation='sigmoid')
])
metrics =[
    Precision(name='precision'),
    'accuracy',
    Recall(name='recall')

]
# Compile the model
baseline_model.compile(optimizer='RMSprop',
                       loss='binary_crossentropy',
                       metrics= metrics)

# Train the model
history = baseline_model.fit(
    train_data,
    steps_per_epoch=train_data.samples // train_data.batch_size,
    epochs=25,
    validation_data=val_data,
    validation_steps=val_data.samples // val_data.batch_size,
)
```

```
# Evaluate the model on the test data
val_loss, val_precision, val_accuracy, val_recall = baseline_model.evaluate(val_data)
print(f"Val Loss: {val_loss:.4f}, Val Precision: {val_precision:.4f},Val Accuracy: {val_accuracy:.4f}, Val Recall: {val_recall:.4f}")

# Predict labels for the test data
y_pred = baseline_model.predict(val_data)
y_pred_binary = np.argmax(y_pred, axis=1)  # Convert probabilities to binary predictions

# True labels for the test data
y_true = val_data.classes
```

```
Found 2146 images belonging to 2 classes.
Found 624 images belonging to 2 classes.
Found 1059 images belonging to 2 classes.
Epoch 1/25
67/67 [==============================] - 60s 868ms/step - loss: 0.6746 - precision: 0.6120 - accuracy: 0.6154 - recall: 0.6346 - val_loss: 1.2748 - val
Epoch 2/25
67/67 [==============================] - 59s 879ms/step - loss: 0.4348 - precision: 0.8389 - accuracy: 0.8174 - recall: 0.7848 - val_loss: 0.5360 - val
Epoch 3/25
67/67 [==============================] - 60s 890ms/step - loss: 0.3515 - precision: 0.8834 - accuracy: 0.8661 - recall: 0.8442 - val_loss: 0.4327 - val
Epoch 4/25
67/67 [==============================] - 59s 888ms/step - loss: 0.2874 - precision: 0.9141 - accuracy: 0.8893 - recall: 0.8608 - val_loss: 0.2304 - val
Epoch 5/25
67/67 [==============================] - 61s 907ms/step - loss: 0.2901 - precision: 0.9065 - accuracy: 0.8798 - recall: 0.8463 - val_loss: 0.2235 - val
Epoch 6/25
67/67 [==============================] - 61s 915ms/step - loss: 0.2707 - precision: 0.9108 - accuracy: 0.8955 - recall: 0.8772 - val_loss: 0.2437 - val
Epoch 7/25
67/67 [==============================] - 61s 918ms/step - loss: 0.2655 - precision: 0.9142 - accuracy: 0.9040 - recall: 0.8908 - val_loss: 0.2058 - val
Epoch 8/25
67/67 [==============================] - 60s 894ms/step - loss: 0.2514 - precision: 0.9077 - accuracy: 0.9049 - recall: 0.9017 - val_loss: 0.3359 - val
Epoch 9/25
67/67 [==============================] - 62s 922ms/step - loss: 0.2570 - precision: 0.9154 - accuracy: 0.9040 - recall: 0.8903 - val_loss: 0.3121 - val
Epoch 10/25
67/67 [==============================] - 60s 903ms/step - loss: 0.2337 - precision: 0.9247 - accuracy: 0.9125 - recall: 0.8975 - val_loss: 0.3279 - val
Epoch 11/25
67/67 [==============================] - 60s 891ms/step - loss: 0.2330 - precision: 0.9183 - accuracy: 0.9125 - recall: 0.9053 - val_loss: 0.4087 - val
Epoch 12/25
67/67 [==============================] - 59s 890ms/step - loss: 0.2222 - precision: 0.9156 - accuracy: 0.9111 - recall: 0.9052 - val_loss: 0.3528 - val
Epoch 13/25
67/67 [==============================] - 62s 929ms/step - loss: 0.2316 - precision: 0.9255 - accuracy: 0.9186 - recall: 0.9096 - val_loss: 0.1956 - val
Epoch 14/25
67/67 [==============================] - 61s 905ms/step - loss: 0.1904 - precision: 0.9368 - accuracy: 0.9276 - recall: 0.9163 - val_loss: 0.6355 - val
Epoch 15/25
67/67 [==============================] - 62s 923ms/step - loss: 0.1936 - precision: 0.9327 - accuracy: 0.9257 - recall: 0.9177 - val_loss: 0.2778 - val
Epoch 16/25
67/67 [==============================] - 61s 908ms/step - loss: 0.1939 - precision: 0.9307 - accuracy: 0.9253 - recall: 0.9183 - val_loss: 0.3380 - val
Epoch 17/25
67/67 [==============================] - 61s 919ms/step - loss: 0.2000 - precision: 0.9303 - accuracy: 0.9276 - recall: 0.9242 - val_loss: 0.2133 - val
Epoch 18/25
67/67 [==============================] - 60s 903ms/step - loss: 0.1616 - precision: 0.9533 - accuracy: 0.9390 - recall: 0.9236 - val_loss: 0.1299 - val
Epoch 19/25
67/67 [==============================] - 59s 889ms/step - loss: 0.1949 - precision: 0.9304 - accuracy: 0.9229 - recall: 0.9136 - val_loss: 0.1656 - val
Epoch 20/25
67/67 [==============================] - 61s 906ms/step - loss: 0.1791 - precision: 0.9399 - accuracy: 0.9347 - recall: 0.9293 - val_loss: 0.1649 - val
Epoch 21/25
67/67 [==============================] - 60s 890ms/step - loss: 0.1715 - precision: 0.9370 - accuracy: 0.9357 - recall: 0.9335 - val_loss: 0.1257 - val
Epoch 22/25
67/67 [==============================] - 60s 890ms/step - loss: 0.1537 - precision: 0.9453 - accuracy: 0.9399 - recall: 0.9336 - val_loss: 0.1539 - val
Epoch 23/25
67/67 [==============================] - 60s 891ms/step - loss: 0.1693 - precision: 0.9443 - accuracy: 0.9399 - recall: 0.9345 - val_loss: 0.2852 - val
Epoch 24/25
67/67 [==============================] - 59s 887ms/step - loss: 0.1567 - precision: 0.9477 - accuracy: 0.9437 - recall: 0.9397 - val_loss: 0.2071 - val
Epoch 25/25
67/67 [==============================] - 59s 888ms/step - loss: 0.1581 - precision: 0.9479 - accuracy: 0.9395 - recall: 0.9300 - val_loss: 0.1243 - val
34/34 [==============================] - 10s 284ms/step - loss: 0.1240 - precision: 0.9681 - accuracy: 0.9528 - recall: 0.9681
Val Loss: 0.1240, Val Precision: 0.9681,Val Accuracy: 0.9528, Val Recall: 0.9681
34/34 [==============================] - 10s 287ms/step
```

Here we will print a summary of the baseline model metrics and architecture.

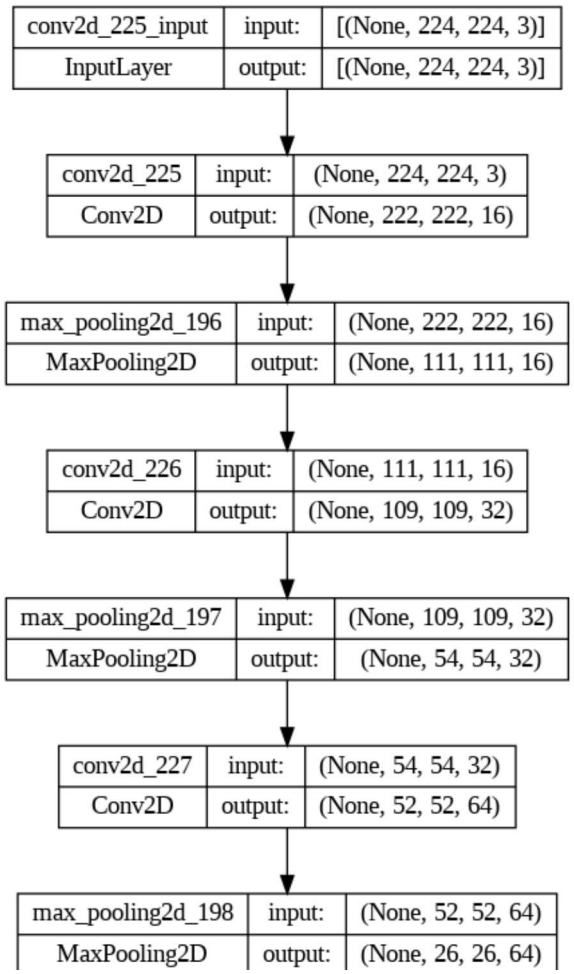
```
# Print the summary of the model

baseline_model.summary()
```

```
Model: "sequential_59"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_225 (Conv2D)         (None, 222, 222, 16)      448

 max_pooling2d_196 (MaxPool  (None, 111, 111, 16)      0
 ing2D)
```

```
 conv2d_226 (Conv2D)         (None, 109, 109, 32)      4640

 max_pooling2d_197 (MaxPool  (None, 54, 54, 32)        0
 ing2D)

 conv2d_227 (Conv2D)         (None, 52, 52, 64)        18496

 max_pooling2d_198 (MaxPool  (None, 26, 26, 64)        0
 ing2D)

 conv2d_228 (Conv2D)         (None, 24, 24, 128)       73856

 max_pooling2d_199 (MaxPool  (None, 12, 12, 128)       0
 ing2D)

 flatten_63 (Flatten)        (None, 18432)             0

 dense_137 (Dense)           (None, 128)               2359424

 dense_138 (Dense)           (None, 64)                8256

 dropout_60 (Dropout)        (None, 64)                0

 dense_139 (Dense)           (None, 1)                 65

=================================================================
Total params: 2465185 (9.40 MB)
Trainable params: 2465185 (9.40 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

This section provides an overview of the model's architecture, layer types, and parameter counts. It serves as a quick reference for understanding the model's structure.

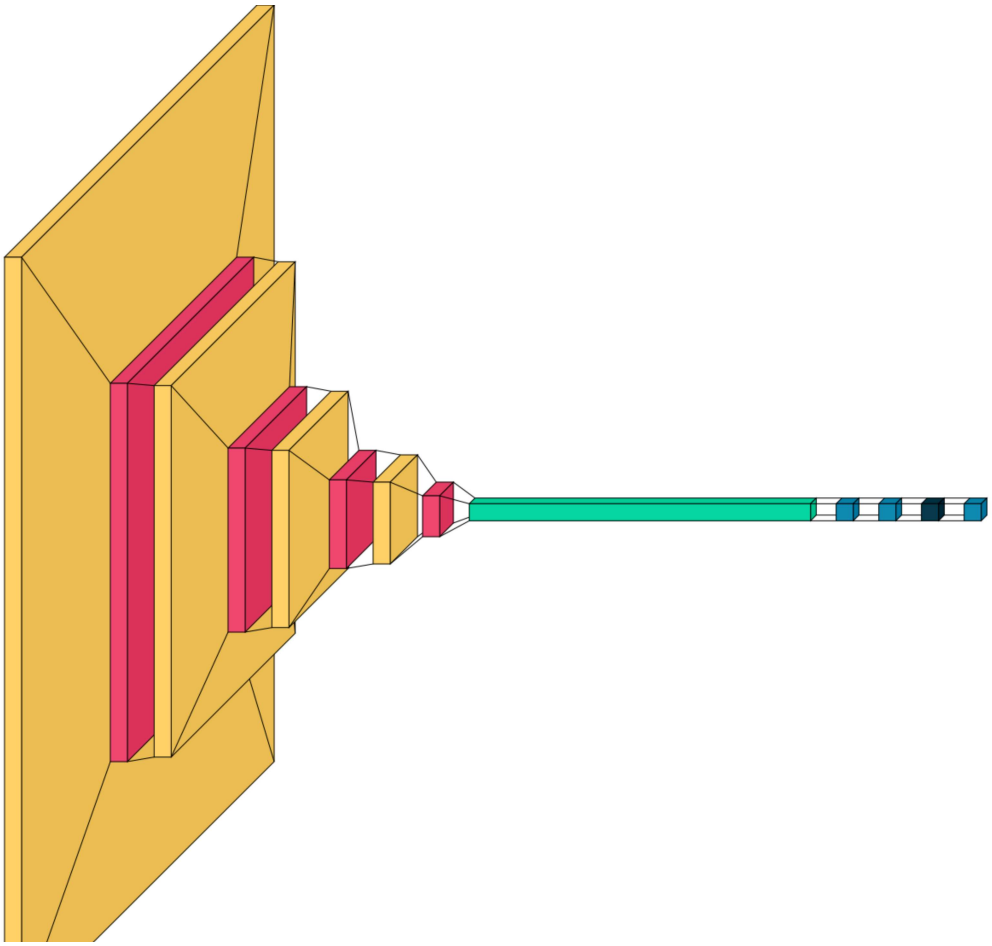
```
from tensorflow.keras.utils import plot_model
# Save a visualization of the model's architecture to a file
plot_model(baseline_model,
           to_file='baseline_model_architecture.png',
           show_shapes=True,
           show_layer_names=True
           )
```




```
!pip install visualkeras
```

```
Requirement already satisfied: visualkeras in /usr/local/lib/python3.10/dist-packages (0.0.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from visualkeras) (9.4.0)
Requirement already satisfied: numpy>=1.18.1 in /usr/local/lib/python3.10/dist-packages (from visualkeras) (1.23.5)
Requirement already satisfied: aggdraw>=1.3.11 in /usr/local/lib/python3.10/dist-packages (from visualkeras) (1.3.16)
```

```
import visualkeras
from PIL import ImageFont
font = ImageFont.truetype("Montserrat-Black.ttf", 18)
visualkeras.layered_view(baseline_model,
                         to_file='baseline_model_architecture_view.png',
                         legend=True,
                         font=font, draw_volume=True, spacing=30)
```



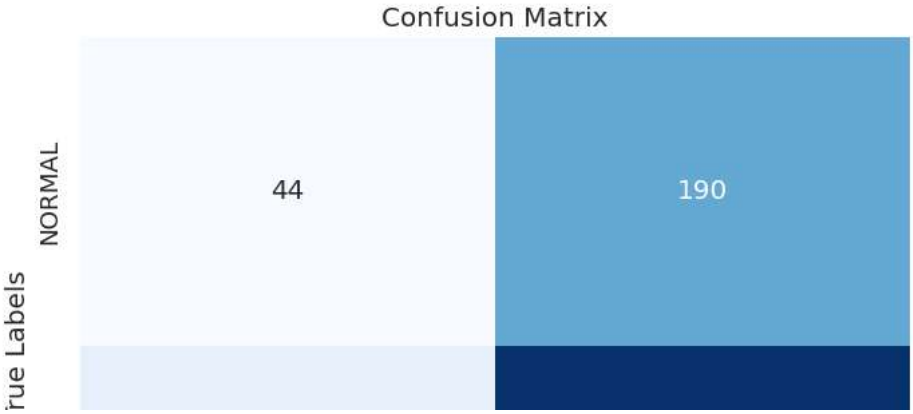
In this section, we will evaluate the performance of our trained model by predicting labels for the test dataset and visualizing the results using a confusion matrix. A confusion matrix is a valuable tool for assessing the classification performance, allowing us to understand how well the model is classifying instances into different categories.

We will start by making predictions on the test dataset using our trained model and then generate a confusion matrix. The confusion matrix provides insights into the true positive, true negative, false positive, and false negative predictions made by the model.

Our baseline model has unsatisfactory values for the f1-score and labels the majority of our images as beinng in the PNEUMONIA class.

```
y_true = test_data.classes
y_pred = baseline_model.predict(test_data)
threshold = 0.5
y_pred_binary = (y_pred > threshold).astype(int)
y_true = np.concatenate([y_true for _ in range(y_pred_binary.shape[0] // len(y_true))])  # Match shapes

# Now compute the confusion matrix
confusion = confusion_matrix(y_true, y_pred_binary)

# Plot the confusion matrix as a heatmap
plt.figure(figsize=(8, 6))
sns.set(font_scale=1.2)
sns.heatmap(confusion, annot=True, fmt="d", cmap="Blues", cbar=False,
            xticklabels=test_data.class_indices.keys(),
            yticklabels=test_data.class_indices.keys())
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```

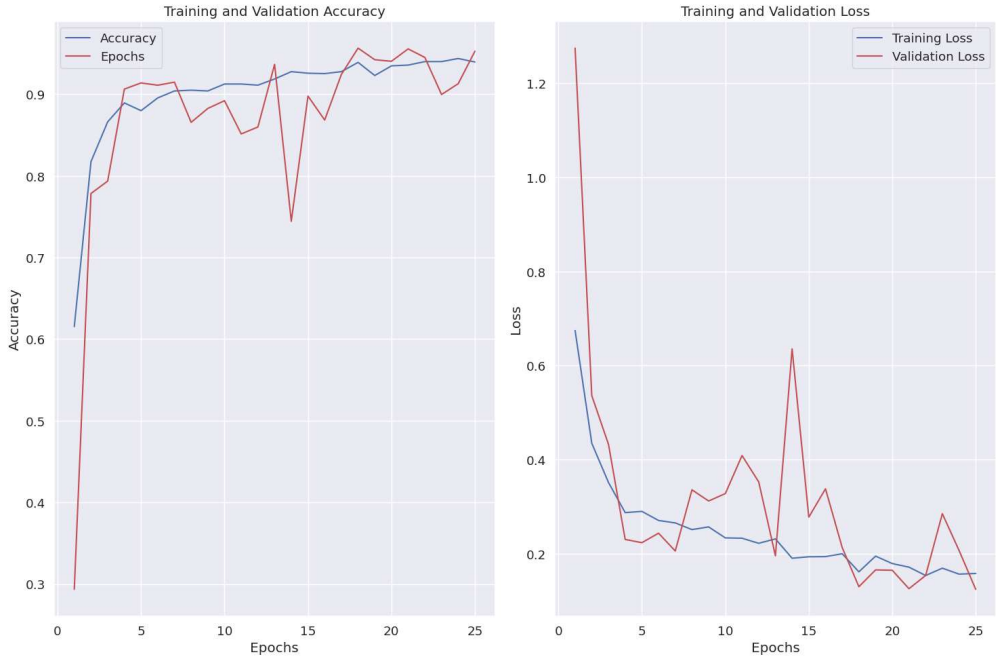
```
20/20 [==============================] - 5s 268ms/step
```




```
# Print classification report (F1-score, precision, accuracy, recall)
report = classification_report(y_true, y_pred_binary, target_names=test_data.class_indices)
print("Classification Report: \n", report)
```

```
Classification Report:
               precision    recall  f1-score   support

      NORMAL       0.40      0.19      0.26       234
   PNEUMONIA       0.63      0.83      0.72       390

    accuracy                           0.59       624
   macro avg       0.51      0.51      0.49       624
weighted avg       0.54      0.59      0.54       624
```

```
train_accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
train_loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(train_accuracy) + 1)

plt.figure(figsize=(15, 10))

# Plot precision vs. epoch
plt.subplot(1, 2, 1)
plt.plot(epochs, train_accuracy, 'b', label='Accuracy')
plt.plot(epochs, val_accuracy, 'r', label='Epochs')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Plot loss vs. epoch
plt.subplot(1, 2, 2)
plt.plot(epochs, train_loss, 'b', label='Training Loss')
plt.plot(epochs, val_loss, 'r', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

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



As we modify the base model, We can also try to adjust the weights of each class in the original training set to see if that affects the output of the model.

Revised CNN Model

In this section, we will visualize the training progress of our deep learning model by plotting key performance metrics over epochs. The two metrics we'll focus on are accuracy and loss, which provide insights into how well the model is learning and whether it's overfitting or underfitting.

- **Training Accuracy vs. Validation Accuracy**: By plotting these curves, we can assess how well the model generalizes to unseen data. An increasing training accuracy with a decreasing validation accuracy could indicate overfitting.
- **Training Loss vs. Validation Loss**: Monitoring the loss curves helps us understand the convergence and stability of the model during training. Ideally, both training and validation loss should decrease over epochs.

Let's visualize the training and validation accuracy, as well as the training and validation loss, to gain a better understanding of our model's performance.

In this section, we make improvements to the Convolutional Neural Network (CNN) model for image classification and evaluate its performance. We employ the following techniques to enhance the model's performance:

- **Regualrization**: We introduce regularization techniques like dropout and L2 regularization to prevent overfitting.
- **Early Stopping**: We introduce an early stopping mechanism to monitor the validation loss during training. If the validation loss does not improve for a certain number of epochs, training is stopped early, preventing overfitting and saving time.
- **Data Augmentation**: We introduce random rotations, translations and scaling of images to increase the diversity of our training data.
- **Revised CNN Model**: We redefine the CNN architecture by adding more convolutional layers and fully connected layers to potentially capture more intricate patterns and features in the data.
- **Increased Epochs**: We increase the number of training epochs to allow the model to learn from the data for a longer duration, potentially improving its performance.
- **Model Evaluation**: After training the revised model, we evaluate its performance using various metrics, including accuracy, loss, F1-score, precision, and recall. We also provide a classification report to gain insights into the model's performance across different classes.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout,BatchNormalization, GlobalAveragePooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import classification_report
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.regularizers import l2

# Set image dimensions and classes
```

```
input_shape = (150, 150, 3)
num_classes = 2

# Define a learning rate scheduler callback using ReduceLROnPlateau
reduce_lr = ReduceLROnPlateau(
    monitor='val_accuracy', # The metric to monitor for improvement (validation accuracy in this case).
    factor=0.3,             # The factor by which the learning rate will be reduced.
    patience=2,            # Number of epochs with no improvement after which learning rate will be reduced.
    verbose=1,              # Display messages about learning rate adjustments.
    mode='max',            # Automatically determines the direction of improvement based on the monitored metric.
    min_delta=0.0001,       # Minimum change in the monitored metric to qualify as an improvement
    cooldown=4,             # Number of epochs to wait before resuming normal learning after a learning rate reduction.
    min_lr=0.000000001            # The lower bound for the learning rate.
)

# Define the EarlyStopping callback
early_stopping = EarlyStopping(
    monitor='val_loss',  # Monitor validation loss
    patience=5,           # Number of epochs with no improvement before stopping
    restore_best_weights=True  # Restore the best model weights
)

# Create data generators for train and test sets
train_generator = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
)

test_generator = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255)
val_generator = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255)
train_data = train_generator.flow_from_directory(
    train_data_path ,
    target_size=input_shape[:2],
    batch_size=32,
    class_mode='binary',
    shuffle=True
)

test_data = test_generator.flow_from_directory(
    test_data_path,
    target_size=input_shape[:2],
    batch_size=32,
    class_mode='binary',
    shuffle=False
)
val_data = val_generator.flow_from_directory(
    val_data_path,
    target_size=input_shape[:2],
    batch_size=32,
    class_mode='binary',
    shuffle=False
)

# Create the baseline CNN model
cnn_model_with_callbacks = tf.keras.models.Sequential([

    Conv2D(16, (3, 3), activation='relu', padding="same", input_shape=input_shape),
    Conv2D(16, (3, 3), padding="same", activation='relu'),
    MaxPooling2D((2, 2)),

    Conv2D(32, (3, 3), activation='relu', padding="same"),
    Conv2D(32, (3, 3), padding="same", activation='relu'),
    MaxPooling2D((2, 2)),

    Conv2D(64, (3, 3), activation='relu', padding="same"),
    Conv2D(64, (3, 3), padding="same", activation='relu'),
    MaxPooling2D((2, 2)),

    Conv2D(96, (3, 3), dilation_rate=(2, 2), activation='relu', padding="same"),
    Conv2D(96, (3, 3), padding="valid", activation='relu'),
    MaxPooling2D((2, 2)),

    Conv2D(128, (3, 3), dilation_rate=(2, 2), activation='relu', padding="same"),
    Conv2D(128, (3, 3), padding="valid", activation='relu'),
    MaxPooling2D((2, 2)),

    Flatten(),
    Dense(units=256, activation='relu'),
    Dropout(0.5),
    Dense(units=1, activation='sigmoid'),
```

```
])


# Compile the model
cnn_model_with_callbacks.compile(optimizer='Adam',
                        loss='binary_crossentropy',
                        metrics=metrics)
class_weight = {0:0.93, 1: 0.68}
# Train the model
history = cnn_model_with_callbacks.fit(
    train_data,
    steps_per_epoch=train_data.samples // train_data.batch_size,
    epochs=50, #increased epochs
    validation_data=val_data,
    validation_steps=val_data.samples // val_data.batch_size,
    #class_weight= class_weight,
    callbacks=[reduce_lr, early_stopping]
)

# Evaluate the model on the testdata
val_loss, val_precision, val_accuracy, val_recall = cnn_model_with_callbacks.evaluate(val_data)
print(f"Val Loss: {val_loss:.4f}, Val Precision: {val_precision:.4f},Val Accuracy: {val_accuracy:.4f}, Val Recall: {val_recall:.4f}")

# Predict labels for the val data
y_pred = cnn_model_with_callbacks.predict(val_data)
y_pred_binary = np.argmax(y_pred, axis=1)  # Convert probabilities to binary predictions

# True labels for the val data
y_true = test_data.classes
```

```
Found 2146 images belonging to 2 classes.
Found 624 images belonging to 2 classes.
Found 1893 images belonging to 2 classes.
Epoch 1/50
67/67 [==============================] - 58s 783ms/step - loss: 0.6940 - precision: 0.4967 - accuracy: 0.5071 - recall: 0.7513 - val_loss: 0.6992 - val
Epoch 2/50
67/67 [==============================] - 51s 770ms/step - loss: 0.6519 - precision: 0.6451 - accuracy: 0.6017 - recall: 0.4522 - val_loss: 0.4718 - val
Epoch 3/50
67/67 [==============================] - 52s 772ms/step - loss: 0.4133 - precision: 0.8507 - accuracy: 0.8254 - recall: 0.7886 - val_loss: 0.2554 - val
Epoch 4/50
67/67 [==============================] - 52s 782ms/step - loss: 0.3261 - precision: 0.8981 - accuracy: 0.8780 - recall: 0.8521 - val_loss: 0.2953 - val
Epoch 5/50
67/67 [==============================] - 52s 776ms/step - loss: 0.2732 - precision: 0.9095 - accuracy: 0.8945 - recall: 0.8759 - val_loss: 0.2508 - val
Epoch 6/50
67/67 [==============================] - 51s 762ms/step - loss: 0.2261 - precision: 0.9237 - accuracy: 0.9153 - recall: 0.9054 - val_loss: 0.2620 - val
Epoch 7/50
67/67 [==============================] - 51s 764ms/step - loss: 0.1827 - precision: 0.9415 - accuracy: 0.9352 - recall: 0.9281 - val_loss: 0.1351 - val
Epoch 8/50
67/67 [==============================] - 52s 783ms/step - loss: 0.1834 - precision: 0.9371 - accuracy: 0.9352 - recall: 0.9327 - val_loss: 0.2541 - val
Epoch 9/50
67/67 [==============================] - ETA: 0s - loss: 0.2429 - precision: 0.9127 - accuracy: 0.8988 - recall: 0.8816
Epoch 9: ReduceLROnPlateau reducing learning rate to 0.0003000000142492354.
67/67 [==============================] - 53s 795ms/step - loss: 0.2429 - precision: 0.9127 - accuracy: 0.8988 - recall: 0.8816 - val_loss: 0.2361 - val
Epoch 10/50
67/67 [==============================] - 52s 785ms/step - loss: 0.2107 - precision: 0.9407 - accuracy: 0.9229 - recall: 0.9024 - val_loss: 0.2636 - val
Epoch 11/50
67/67 [==============================] - 50s 754ms/step - loss: 0.1825 - precision: 0.9397 - accuracy: 0.9281 - recall: 0.9148 - val_loss: 0.1808 - val
Epoch 12/50
67/67 [==============================] - 50s 764ms/step - loss: 0.1517 - precision: 0.9565 - accuracy: 0.9456 - recall: 0.9339 - val_loss: 0.1775 - val
60/60 [==============================] - 17s 272ms/step - loss: 0.1348 - precision: 0.9802 - accuracy: 0.9509 - recall: 0.9530
Val Loss: 0.1348, Val Precision: 0.9802,Val Accuracy: 0.9509, Val Recall: 0.9530
60/60 [==============================] - 17s 271ms/step
```

Model Summary: Sequential CNN with Callbacks Architecture

- **Layer Details**:
  - **conv2d_138 (Conv2D)**: This initial convolutional layer extracts 16 features from the input images with a kernel size of 3x3.
  - **conv2d_139 (Conv2D)**: Another convolutional layer follows, enriching the feature representation.
  - **max_pooling2d_100 (MaxPooling2D)**: This layer reduces the spatial dimensions, enhancing computational efficiency.
  - **conv2d_140 (Conv2D)**: Further convolutional operations are performed, increasing the depth of feature extraction.
  - **conv2d_141 (Conv2D)**: This convolutional layer continues to refine the features.
  - **max_pooling2d_101 (MaxPooling2D)**: Another max-pooling layer aids in dimensionality reduction.
  - **conv2d_142 (Conv2D)**: This layer increases the complexity of the feature representation.
  - **conv2d_143 (Conv2D)**: A similar convolutional operation is applied, deepening the model's understanding.
  - **max_pooling2d_102 (MaxPooling2D)**: Max-pooling helps maintain spatial relationships while reducing computational load.
  - **conv2d_144 (Conv2D)**: This convolutional layer extracts even more complex features from the data.
  - **conv2d_145 (Conv2D)**: The model continues to refine its feature extraction capabilities.
  - **max_pooling2d_103 (MaxPooling2D)**: Further dimensionality reduction is performed to enhance efficiency.
  - **conv2d_146 (Conv2D)**: This layer increases the complexity of feature learning.

- **conv2d_147 (Conv2D)**: Another convolutional operation aids in capturing intricate patterns.
- **max_pooling2d_104 (MaxPooling2D)**: Final pooling operation before transitioning to fully connected layers.
- **flatten_22 (Flatten)**: Converts the output of convolutional layers into a flat vector for dense layers.
- **dense_58 (Dense)**: A densely connected layer with 256 units, applying non-linearity to the learned features.
- **dropout_45 (Dropout)**: Dropout layer with a 50% drop rate to reduce overfitting.
- **dense_59 (Dense)**: Final dense layer with a single unit and sigmoid activation for binary classification.

- **Total Parameters**: 764,241 (2.92 MB)
- **Trainable Parameters**: 764,241 (2.92 MB)
- **Non-trainable Parameters**: 0 (0.00 Byte)

Model Purpose and Usage:

This architecture is designed for image classification tasks. It excels in extracting intricate patterns and features from input images, making it suitable for tasks where detailed analysis is crucial.

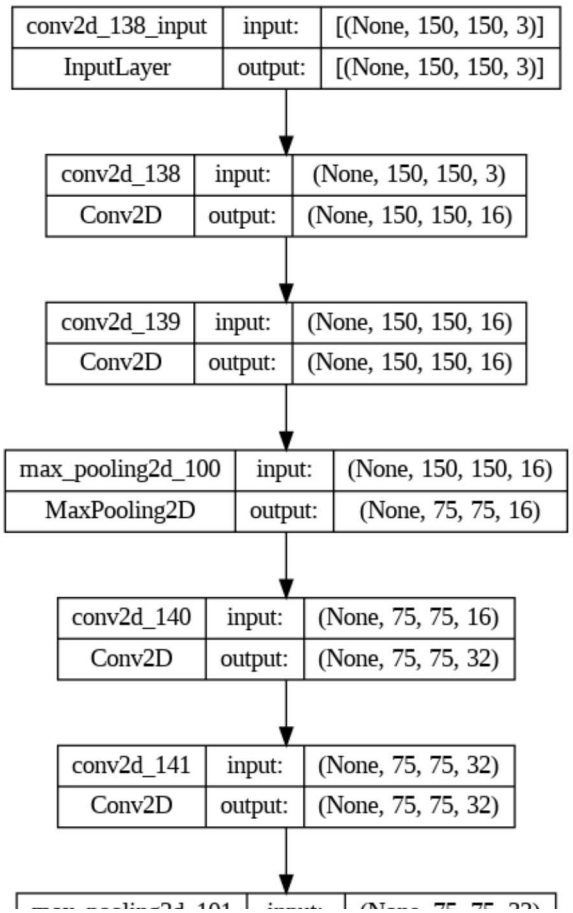
Important Notes:

- This model should be used primarily for educational purposes and not as a replacement for professional medical advice or diagnosis.
- Before applying this model to real-world decisions, it is strongly advised to consult with domain experts and medical professionals.
- The author and contributors disclaim any liability for the accuracy, completeness, or efficacy of the information provided by this model.

```
# Print the summary of the model

cnn_model_with_callbacks.summary()
```

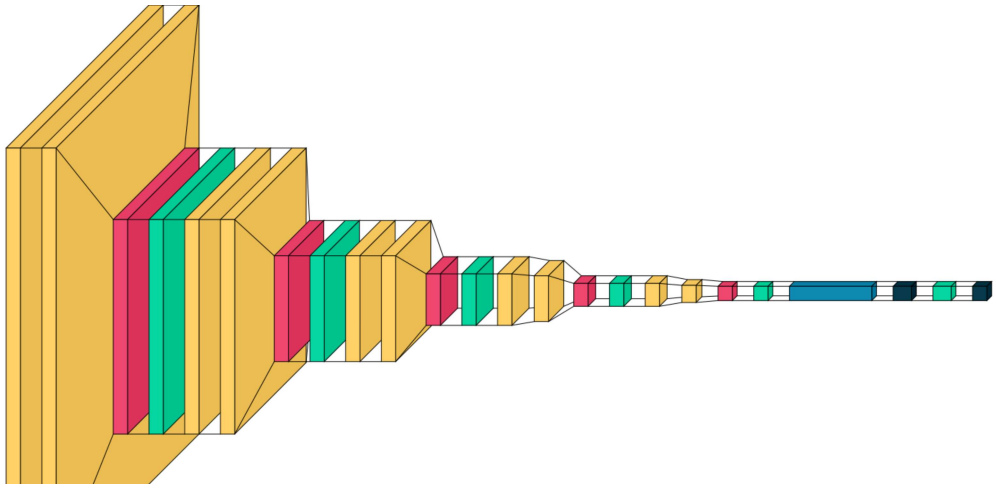
```
Model: "sequential_22"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_138 (Conv2D)         (None, 150, 150, 16)      448

 conv2d_139 (Conv2D)         (None, 150, 150, 16)      2320

 max_pooling2d_100 (MaxPool  (None, 75, 75, 16)        0
 ing2D)

 conv2d_140 (Conv2D)         (None, 75, 75, 32)        4640

 conv2d_141 (Conv2D)         (None, 75, 75, 32)        9248

 max_pooling2d_101 (MaxPool  (None, 37, 37, 32)        0
 ing2D)

 conv2d_142 (Conv2D)         (None, 37, 37, 64)        18496

 conv2d_143 (Conv2D)         (None, 37, 37, 64)        36928

 max_pooling2d_102 (MaxPool  (None, 18, 18, 64)        0
 ing2D)

 conv2d_144 (Conv2D)         (None, 18, 18, 96)        55392

 conv2d_145 (Conv2D)         (None, 16, 16, 96)        83040

 max_pooling2d_103 (MaxPool  (None, 8, 8, 96)          0
 ing2D)

 conv2d_146 (Conv2D)         (None, 8, 8, 128)         110720

 conv2d_147 (Conv2D)         (None, 6, 6, 128)         147584

 max_pooling2d_104 (MaxPool  (None, 3, 3, 128)         0
 ing2D)

 flatten_22 (Flatten)        (None, 1152)              0

 dense_58 (Dense)            (None, 256)               295168

 dropout_45 (Dropout)        (None, 256)               0

 dense_59 (Dense)            (None, 1)                 257

=================================================================
Total params: 764241 (2.92 MB)
Trainable params: 764241 (2.92 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
from tensorflow.keras.utils import plot_model
# Save a visualization of the model's architecture to a file
plot_model(cnn_model_with_callbacks, to_file='cnn_model_with_callbacks_architecture.png', show_shapes=True,show_layer_names=True)
```



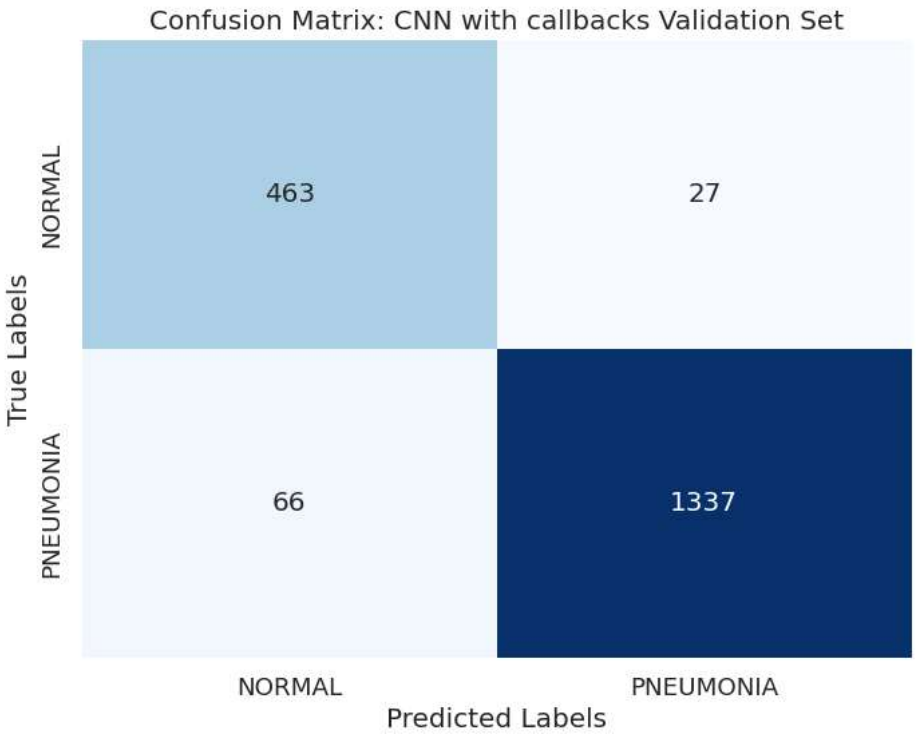

```
!pip install visualkeras

    Collecting visualkeras
      Downloading visualkeras-0.0.2-py3-none-any.whl (12 kB)
    Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from visualkeras) (9.4.0)
    Requirement already satisfied: numpy>=1.18.1 in /usr/local/lib/python3.10/dist-packages (from visualkeras) (1.23.5)
    Collecting aggdraw>=1.3.11 (from visualkeras)
      Downloading aggdraw-1.3.16-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (993 kB)
         ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 993.0/993.0 kB 11.4 MB/s eta 0:00:00
    Installing collected packages: aggdraw, visualkeras
    Successfully installed aggdraw-1.3.16 visualkeras-0.0.2
```

```
import visualkeras
from PIL import ImageFont
font = ImageFont.truetype("Montserrat-Black.ttf", 18)
visualkeras.layered_view(cnn_model_with_callbacks,
                         to_file='cnn_model_with_callbacks_architecture_view.png',
                         legend=True,
                         font=font, draw_volume=True, spacing=30)
```



```
y_true = val_data.classes
y_pred = cnn_model_with_callbacks.predict(val_data)
threshold = 0.5
y_pred_binary = (y_pred > threshold).astype(int)
y_true = np.concatenate([y_true for _ in range(y_pred_binary.shape[0] // len(y_true))])  # Match shapes

# Now compute the confusion matrix
confusion = confusion_matrix(y_true, y_pred_binary)

# Plot the confusion matrix as a heatmap
plt.figure(figsize=(8, 6))
sns.set(font_scale=1.2)
sns.heatmap(confusion, annot=True, fmt="d", cmap="Blues", cbar=False,
            xticklabels=val_data.class_indices.keys(),
            yticklabels=val_data.class_indices.keys())
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix: CNN with callbacks Validation Set')
plt.show()
```

```
60/60 [==============================] - 17s 273ms/step
```



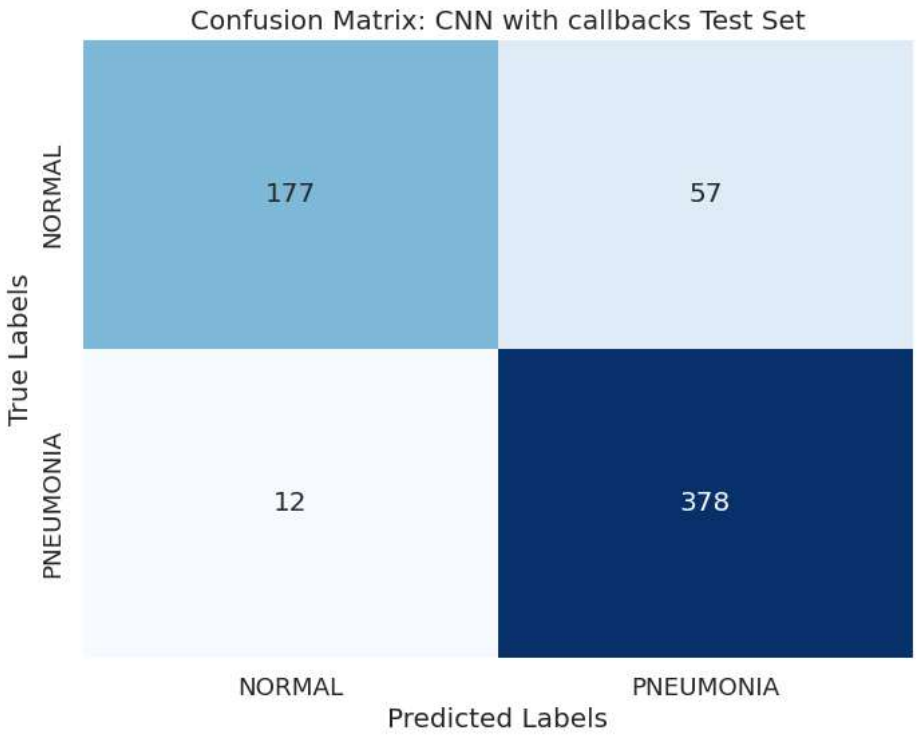

```
# Print classification report (F1-score, precision, accuracy, recall)
report = classification_report(y_true, y_pred_binary, target_names=test_data.class_indices)
print("Revised CNN Classification Report: \n", report)
```

```
Revised CNN Classification Report:
               precision    recall  f1-score   support

      NORMAL       0.88      0.94      0.91       490
   PNEUMONIA       0.98      0.95      0.97      1403

    accuracy                           0.95      1893
   macro avg       0.93      0.95      0.94      1893
```

```
     weighted avg       0.95      0.95      0.95      1893


train_accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
train_loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(train_accuracy) + 1)

plt.figure(figsize=(15, 10))

# Plot precision vs. epoch
plt.subplot(1, 2, 1)
plt.plot(epochs, train_accuracy, 'b', label='Accuracy')
plt.plot(epochs, val_accuracy, 'r', label='Epochs')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Plot loss vs. epoch
plt.subplot(1, 2, 2)
plt.plot(epochs, train_loss, 'b', label='Training Loss')
plt.plot(epochs, val_loss, 'r', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

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

```
y_true = test_data.classes
y_pred = cnn_model_with_callbacks.predict(test_data)
threshold = 0.5
y_pred_binary = (y_pred > threshold).astype(int)
y_true = np.concatenate([y_true for _ in range(y_pred_binary.shape[0] // len(y_true))])  # Match shapes

# Now compute the confusion matrix
confusion = confusion_matrix(y_true, y_pred_binary)

# Plot the confusion matrix as a heatmap
plt.figure(figsize=(8, 6))
sns.set(font_scale=1.2)
sns.heatmap(confusion, annot=True, fmt="d", cmap="Blues", cbar=False,
            xticklabels=test_data.class_indices.keys(),
            yticklabels=test_data.class_indices.keys())
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix: CNN with callbacks Test Set')
plt.show()
```

```
20/20 [==============================] - 5s 255ms/step
```

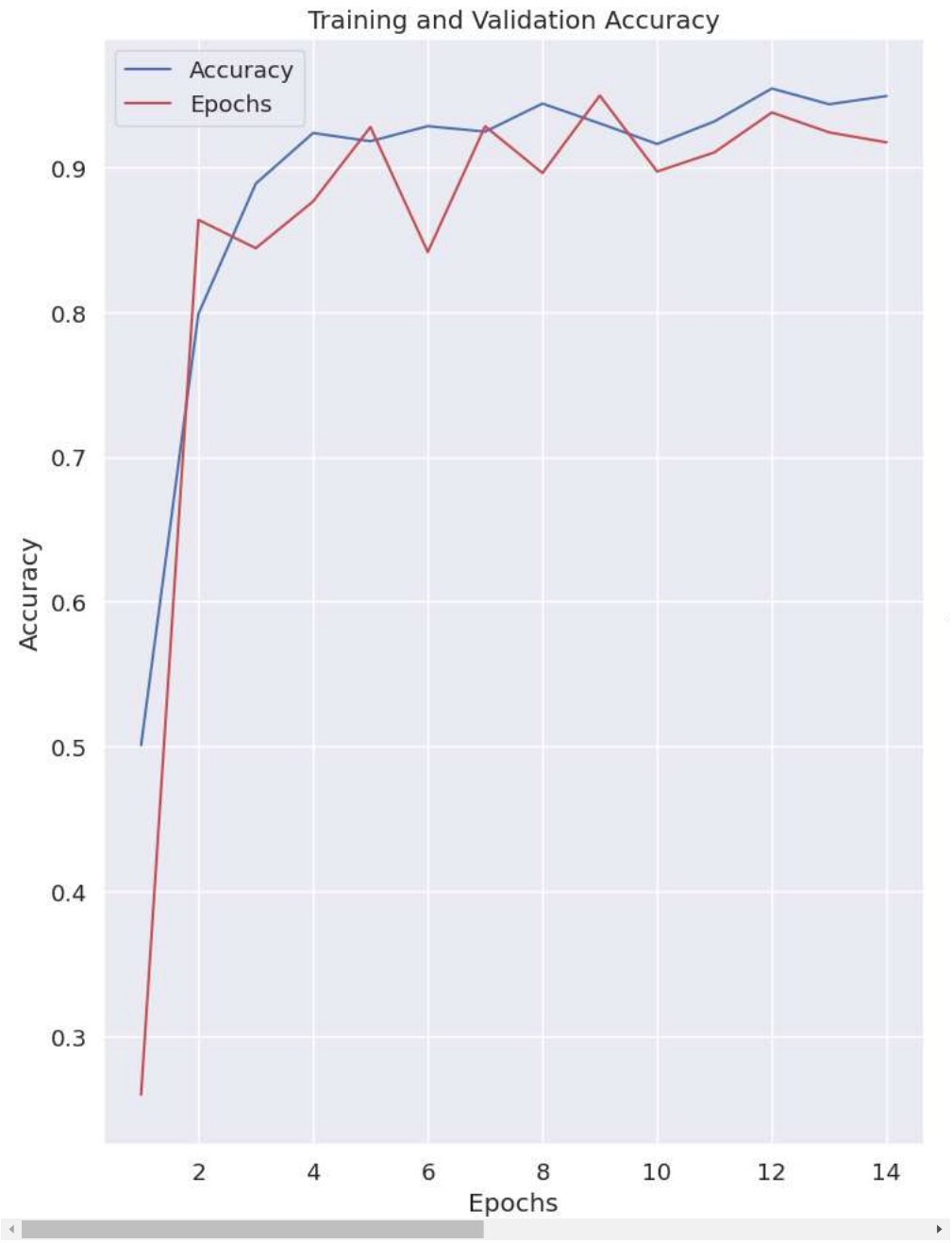



```
# Print classification report (F1-score, precision, accuracy, recall)
report = classification_report(y_true, y_pred_binary, target_names=test_data.class_indices)
print("Revised CNN Classification Report for Test Data: \n", report)
```

```
Revised CNN Classification Report for Test Data:
               precision    recall  f1-score   support

      NORMAL       0.94      0.76      0.84       234
   PNEUMONIA       0.87      0.97      0.92       390

    accuracy                           0.89       624
   macro avg       0.90      0.86      0.88       624
weighted avg       0.89      0.89      0.89       624
```

```
train_accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
train_loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(train_accuracy) + 1)

plt.figure(figsize=(15, 10))

# Plot precision vs. epoch
plt.subplot(1, 2, 1)
plt.plot(epochs, train_accuracy, 'b', label='Accuracy')
plt.plot(epochs, val_accuracy, 'r', label='Epochs')
```

```
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Plot loss vs. epoch
plt.subplot(1, 2, 2)
plt.plot(epochs, train_loss, 'b', label='Training Loss')
plt.plot(epochs, val_loss, 'r', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

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



This code implements a Convolutional Neural Network (CNN) for binary image classification. The model is built using TensorFlow's Keras API. Here's an overview of the key components:

- **Model Architecture**: The CNN architecture consists of several convolutional layers with ReLU activation, max-pooling layers, batch normalization, and fully connected layers.
- **ImageDataGenerator**: Data augmentation is performed using `tf.keras.preprocessing.image.ImageDataGenerator` to improve model generalization.
- **Callbacks**: Two callbacks are employed - `ReduceLROnPlateau` for dynamically adjusting learning rates, and `EarlyStopping` to prevent overfitting by stopping training early if the validation loss plateaus.
- **Regularization**: L2 regularization is applied to the convolutional and dense layers to prevent overfitting.

- **Model Compilation**: The model is compiled with the RMSprop optimizer and sparse categorical cross-entropy loss function.
- **Training**: The model is trained on the training data with validation on a separate test set. Training stops early if no improvement is observed in the validation loss.
- **Evaluation**: The model is evaluated on the test set, and metrics like loss and accuracy are printed.
- **Predictions and Classification Report**: Predictions are generated for the test data and compared against true labels. The classification report provides metrics such as F1-score, precision, accuracy, and recall.

Here is a summary and flow chart of the revised model.

Revised CNN Model with L2 regularization

In this section, we will evaluate the performance of our revised trained model with L2 Regualarization by predicting labels for the test dataset and visualizing the results using a confusion matrix. A confusion matrix is a valuable tool for assessing the classification performance, allowing us to understand how well the model is classifying instances into different categories.

We will start by making predictions on the test dataset using our trained model and then generate a confusion matrix. The confusion matrix provides insights into the true positive, true negative, false positive, and false negative predictions made by the model.

Although the Test Accuracy is ~ 84%, the model needs to be adjusted to account for the class imbalance.

While the test accuracy was

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout,BatchNormalization, GlobalAveragePooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import classification_report
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.regularizers import l2

# Set image dimensions and classes
input_shape = (150, 150, 3)
num_classes = 2

# Define a learning rate scheduler callback using ReduceLROnPlateau
reduce_lr = ReduceLROnPlateau(
    monitor='val_accuracy', # The metric to monitor for improvement (validation accuracy in this case).
    factor=0.3,             # The factor by which the learning rate will be reduced.
    patience=2,            # Number of epochs with no improvement after which learning rate will be reduced.
    verbose=1,              # Display messages about learning rate adjustments.
    mode='max',            # Automatically determines the direction of improvement based on the monitored metric.
    min_delta=0.0001,       # Minimum change in the monitored metric to qualify as an improvement
    cooldown=4,             # Number of epochs to wait before resuming normal learning after a learning rate reduction.
    min_lr=0.000000001            # The lower bound for the learning rate.
)

# Define the EarlyStopping callback
early_stopping = EarlyStopping(
    monitor='val_loss',  # Monitor validation loss
    patience=5,           # Number of epochs with no improvement before stopping
    restore_best_weights=True  # Restore the best model weights
)

# Create data generators for train and test sets
train_generator = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
)

test_generator = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255)
val_generator = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255)
train_data = train_generator.flow_from_directory(
    train_data_path ,
    target_size=input_shape[:2],
    batch_size=32,
    class_mode='binary',
    shuffle=True
)

test_data = test_generator.flow_from_directory(
    test_data_path,
    target_size=input_shape[:2],
    batch_size=32,
    class_mode='binary',
    shuffle=False
)
```

```
val_data = val_generator.flow_from_directory(
    val_data_path,
    target_size=input_shape[:2],
    batch_size=32,
    class_mode='binary',
    shuffle=False
)

# Create the baseline CNN model with L2 regularization
cnn_model_with_l2_regularization = tf.keras.models.Sequential([

    Conv2D(16, (3, 3), activation='relu', padding="same", input_shape=input_shape,
           kernel_regularizer=l2(0.01)),
    Conv2D(16, (3, 3), padding="same", activation='relu'),
    MaxPooling2D((2, 2)),

    Conv2D(32, (3, 3), activation='relu', padding="same"),
    Conv2D(32, (3, 3), padding="same", activation='relu'),
    MaxPooling2D((2, 2)),

    Conv2D(64, (3, 3), activation='relu', padding="same"),
    Conv2D(64, (3, 3), padding="same", activation='relu'),
    MaxPooling2D((2, 2)),

    Conv2D(96, (3, 3), dilation_rate=(2, 2), activation='relu', padding="same"),
    Conv2D(96, (3, 3), padding="valid", activation='relu'),
    MaxPooling2D((2, 2)),

    Conv2D(128, (3, 3), dilation_rate=(2, 2), activation='relu', padding="same"),
    Conv2D(128, (3, 3), padding="valid", activation='relu'),
    MaxPooling2D((2, 2)),

    Flatten(),
    Dense(units=256, activation='relu'),
    #Dropout(0.5),
    Dense(units=1, activation='sigmoid'),

])


# Compile the model
cnn_model_with_l2_regularization.compile(optimizer='Adam',
                        loss='binary_crossentropy',
                        metrics=metrics)
class_weight = {0:0.93, 1: 0.68}
# Train the model
history = cnn_model_with_l2_regularization.fit(
    train_data,
    steps_per_epoch=train_data.samples // train_data.batch_size,
    epochs=25, #increased epochs
    validation_data=val_data,
    validation_steps=val_data.samples // val_data.batch_size,
    class_weight= class_weight,
    callbacks=[reduce_lr, early_stopping]
)

# Evaluate the model on the testdata
val_loss, val_precision, val_accuracy, val_recall = cnn_model_with_l2_regularization.evaluate(val_data)
print(f"Val Loss: {val_loss:.4f}, Val Precision: {val_precision:.4f},Val Accuracy: {val_accuracy:.4f}, Val Recall: {val_recall:.4f}")

# Predict labels for the val data
y_pred = cnn_model_with_l2_regularization.predict(val_data)
y_pred_binary = np.argmax(y_pred, axis=1)  # Convert probabilities to binary predictions

# True labels for the val data
y_true = test_data.classes
```

```
Found 2146 images belonging to 2 classes.
Found 624 images belonging to 2 classes.
Found 1893 images belonging to 2 classes.
Epoch 1/25
67/67 [==============================] - 53s 721ms/step - loss: 0.5904 - precision: 0.4688 - accuracy: 0.5009 - recall: 0.0075 - val_loss: 0.7324 - val
Epoch 2/25
67/67 [==============================] - 49s 732ms/step - loss: 0.3576 - precision: 0.8547 - accuracy: 0.7990 - recall: 0.7217 - val_loss: 0.3732 - val
Epoch 3/25
67/67 [==============================] - 48s 728ms/step - loss: 0.2390 - precision: 0.9289 - accuracy: 0.8888 - recall: 0.8419 - val_loss: 0.4131 - val
Epoch 4/25
67/67 [==============================] - 49s 730ms/step - loss: 0.1852 - precision: 0.9436 - accuracy: 0.9238 - recall: 0.9017 - val_loss: 0.3565 - val
Epoch 5/25
67/67 [==============================] - 49s 729ms/step - loss: 0.1835 - precision: 0.9476 - accuracy: 0.9182 - recall: 0.8860 - val_loss: 0.2303 - val
Epoch 6/25
67/67 [==============================] - 50s 746ms/step - loss: 0.1716 - precision: 0.9451 - accuracy: 0.9286 - recall: 0.9103 - val_loss: 0.4586 - val
Epoch 7/25
67/67 [==============================] - 50s 754ms/step - loss: 0.1781 - precision: 0.9463 - accuracy: 0.9248 - recall: 0.9007 - val_loss: 0.2021 - val
```

```
Epoch 8/25
67/67 [==============================] - 50s 744ms/step - loss: 0.1460 - precision: 0.9598 - accuracy: 0.9442 - recall: 0.9272 - val_loss: 0.2855 - val
Epoch 9/25
67/67 [==============================] - 50s 744ms/step - loss: 0.1703 - precision: 0.9525 - accuracy: 0.9305 - recall: 0.9067 - val_loss: 0.1458 - val
Epoch 10/25
67/67 [==============================] - 50s 744ms/step - loss: 0.1778 - precision: 0.9462 - accuracy: 0.9163 - recall: 0.8826 - val_loss: 0.2847 - val
Epoch 11/25
67/67 [==============================] - ETA: 0s - loss: 0.1438 - precision: 0.9427 - accuracy: 0.9319 - recall: 0.9194
Epoch 11: ReduceLROnPlateau reducing learning rate to 0.0003000000142492354.
67/67 [==============================] - 50s 752ms/step - loss: 0.1438 - precision: 0.9427 - accuracy: 0.9319 - recall: 0.9194 - val_loss: 0.2720 - val
Epoch 12/25
67/67 [==============================] - 50s 751ms/step - loss: 0.1176 - precision: 0.9643 - accuracy: 0.9546 - recall: 0.9442 - val_loss: 0.1950 - val
Epoch 13/25
67/67 [==============================] - 50s 746ms/step - loss: 0.1266 - precision: 0.9599 - accuracy: 0.9437 - recall: 0.9263 - val_loss: 0.2167 - val
Epoch 14/25
67/67 [==============================] - 50s 753ms/step - loss: 0.1248 - precision: 0.9628 - accuracy: 0.9494 - recall: 0.9345 - val_loss: 0.2283 - val
60/60 [==============================] - 16s 269ms/step - loss: 0.1455 - precision: 0.9665 - accuracy: 0.9498 - recall: 0.9658
Val Loss: 0.1455, Val Precision: 0.9665,Val Accuracy: 0.9498, Val Recall: 0.9658
60/60 [==============================] - 16s 265ms/step
```

In this section, we aim to assess the performance of our model by generating a confusion matrix based on its predictions for the test data. A confusion matrix is a valuable tool in classification tasks as it provides insights into the model's ability to correctly classify instances into different classes.

Predicting Labels for Test Data

We start by utilizing our trained model, referred to as `revised_model_1`, to make predictions on the test data. The predictions are initially in the form of probabilities for each class. To make them interpretable, we convert these probabilities into binary predictions.

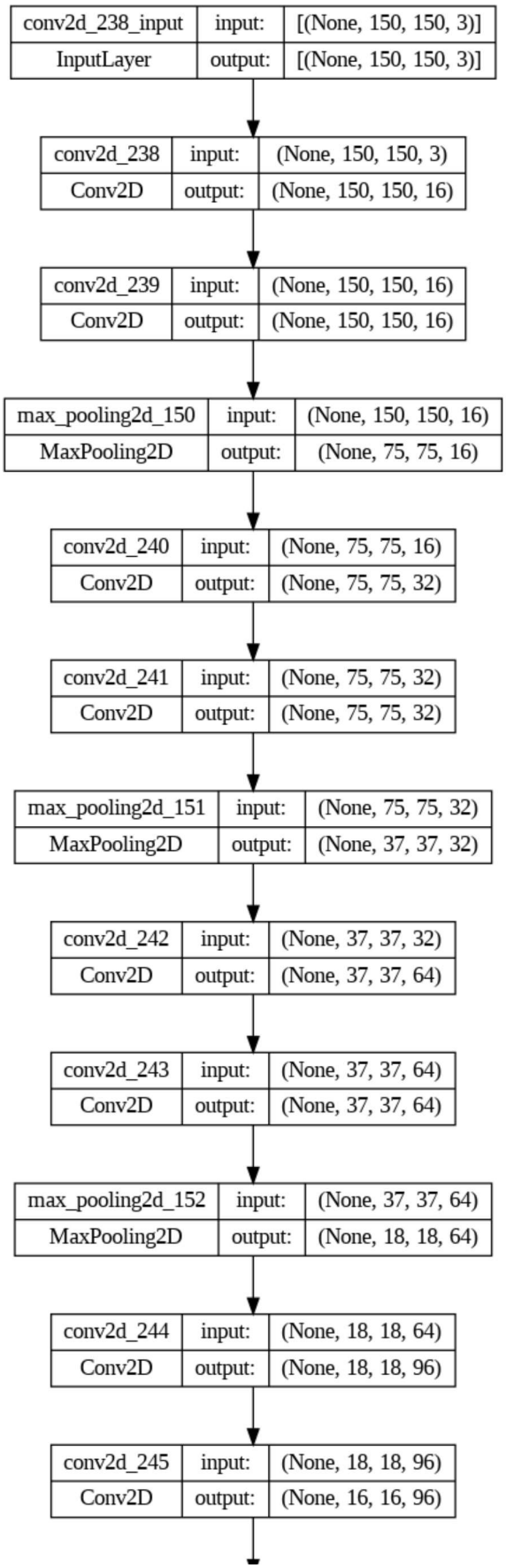
True Labels for Test Data

We also retrieve the true labels for the test data. These labels represent the actual classes to which each instance in the test set belongs.

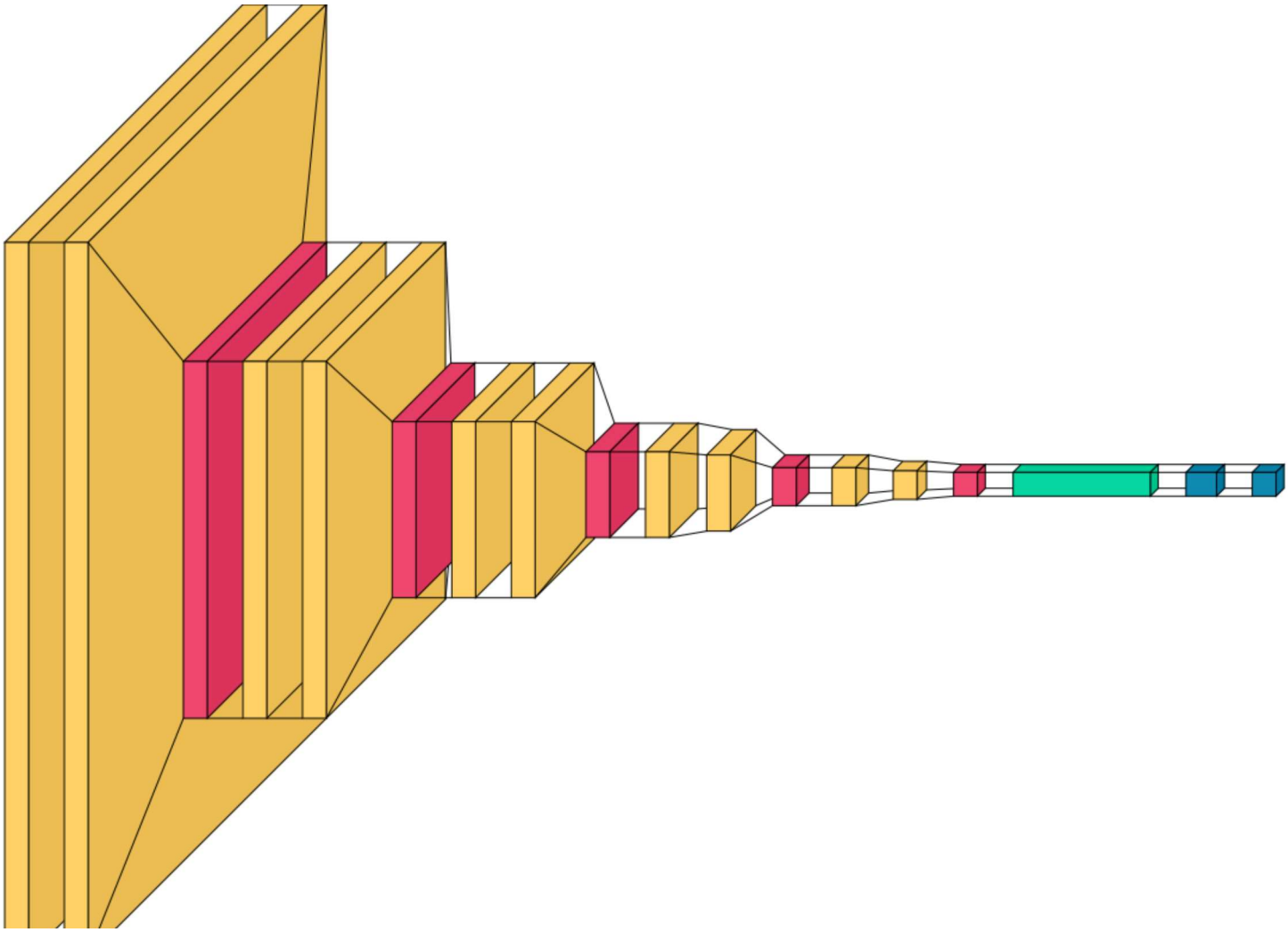
Generating the Confusion Matrix

The confusion matrix is created using the true labels `y_true` and the predicted labels `y_pred_binary`. This matrix provides a detailed breakdown of how many instances from each class were correctly or incorrectly classified.

```
from tensorflow.keras.utils import plot_model
# Save a visualization of the model's architecture to a file
plot_model(cnn_model_with_l2_regularization, to_file='cnn_model_with_l2_regularization.png', show_shapes=True,show_layer_names=True)
```

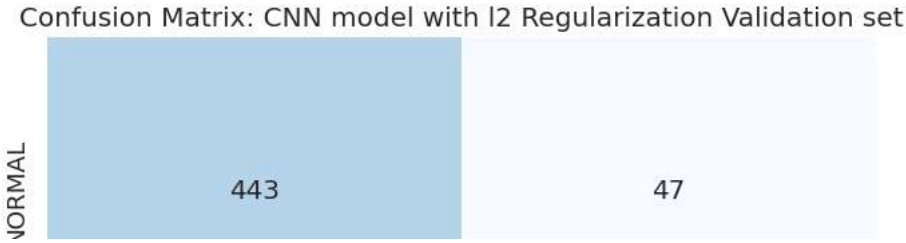



```
import visualkeras
from PIL import ImageFont
font = ImageFont.truetype("Montserrat-Black.ttf", 18)
visualkeras.layered_view(cnn_model_with_l2_regularization,
                         to_file='cnn_model_with_l2_regularization_architecture_view.png',
                         legend=True,
                         font=font, draw_volume=True, spacing=30)
```



```
# Predict labels for the val data
y_true = val_data.classes
y_pred = cnn_model_with_l2_regularization.predict(val_data)
threshold = 0.5
y_pred_binary = (y_pred > threshold).astype(int)
y_true = np.concatenate([y_true for _ in range(y_pred_binary.shape[0] // len(y_true))])  # Match shapes

# Now compute the confusion matrix
confusion = confusion_matrix(y_true, y_pred_binary)

# Plot the confusion matrix as a heatmap
plt.figure(figsize=(8, 6))
sns.set(font_scale=1.2)
sns.heatmap(confusion, annot=True, fmt="d", cmap="Blues", cbar=False,
            xticklabels=val_data.class_indices.keys(),
            yticklabels=val_data.class_indices.keys())
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix: CNN model with l2 Regularization Validation set')
plt.show()
```

```
60/60 [==============================] - 17s 270ms/step
```

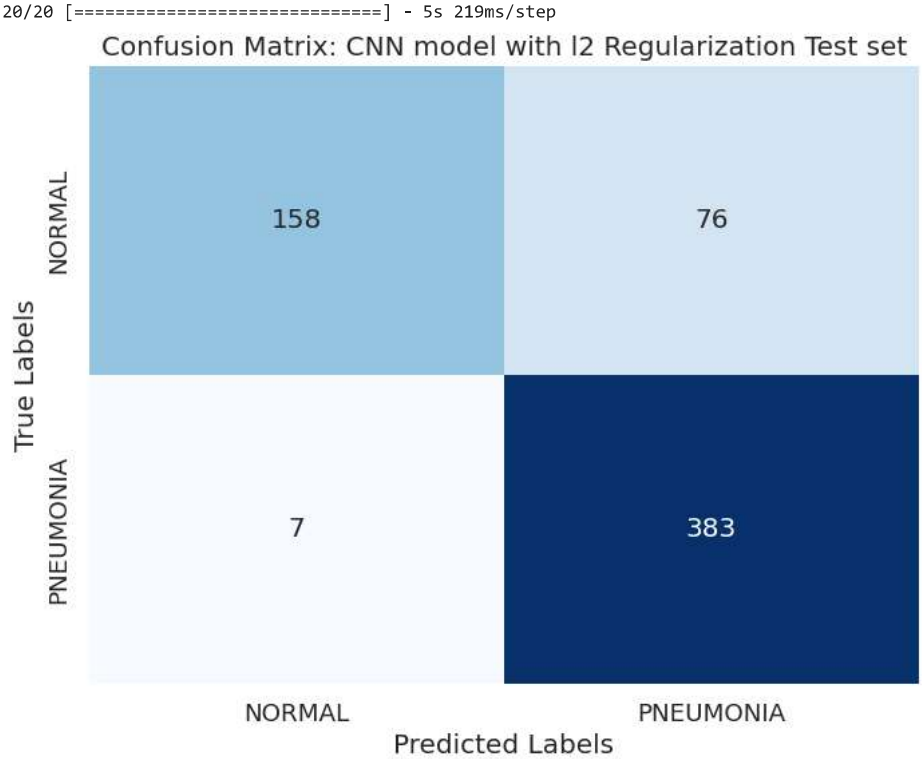



```
# Print classification report (F1-score, precision, accuracy, recall)
report = classification_report(y_true, y_pred_binary, target_names=test_data.class_indices)
print("Revised CNN with l2 Regularization Classification Report on validation data: \n", report)
```

```
Revised CNN with l2 Regularization Classification Report on validation data:
               precision    recall  f1-score   support

      NORMAL       0.90      0.90      0.90       490
   PNEUMONIA       0.97      0.97      0.97      1403

    accuracy                           0.95      1893
   macro avg       0.93      0.93      0.93      1893
weighted avg       0.95      0.95      0.95      1893
```

```
# Predict labels for the test data
y_true = test_data.classes
y_pred = cnn_model_with_l2_regularization.predict(test_data)
threshold = 0.5
y_pred_binary = (y_pred > threshold).astype(int)
y_true = np.concatenate([y_true for _ in range(y_pred_binary.shape[0] // len(y_true))])  # Match shapes

# Now compute the confusion matrix
confusion = confusion_matrix(y_true, y_pred_binary)

# Plot the confusion matrix as a heatmap
plt.figure(figsize=(8, 6))
sns.set(font_scale=1.2)
sns.heatmap(confusion, annot=True, fmt="d", cmap="Blues", cbar=False,
            xticklabels=val_data.class_indices.keys(),
            yticklabels=val_data.class_indices.keys())
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix: CNN model with l2 Regularization Test set')
plt.show()
```

```
20/20 [==============================] - 5s 219ms/step
```




```
# Print classification report (F1-score, precision, accuracy, recall)
report = classification_report(y_true, y_pred_binary, target_names=test_data.class_indices)
print("Revised CNN with l2 Regularization Classification Report on Test data: \n", report)
```

```
Revised CNN with l2 Regularization Classification Report on Test data:
              precision    recall  f1-score   support

      NORMAL       0.96      0.68      0.79       234
   PNEUMONIA       0.83      0.98      0.90       390

    accuracy                           0.87       624
   macro avg       0.90      0.83      0.85       624
weighted avg       0.88      0.87      0.86       624
```

```
train_accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
train_loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(train_accuracy) + 1)

plt.figure(figsize=(15, 10))

# Plot precision vs. epoch
plt.subplot(1, 2, 1)
plt.plot(epochs, train_accuracy, 'b', label='Accuracy')
plt.plot(epochs, val_accuracy, 'r', label='Epochs')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Plot loss vs. epoch
plt.subplot(1, 2, 2)
plt.plot(epochs, train_loss, 'b', label='Training Loss')
plt.plot(epochs, val_loss, 'r', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

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

The summary provides information about the total number of parameters in the model, including trainable and non-trainable parameters. In this case, there are approximately 2.8 million parameters in the network. Here is some more information about the layers and parameters in this revised model.

- **max_pooling2d_3 (MaxPooling2D):**
  - Output Shape: (None, 111, 111, 32)
  - Description: This is a MaxPooling2D layer that reduces the spatial dimensions of the input tensor. It takes an input of shape (None, height, width, channels) and outputs a tensor with dimensions (None, new_height, new_width, channels). In this case, it reduces the height and width dimensions from the previous layer, resulting in an output shape of (None, 111, 111, 32).
- **conv2d_5 (Conv2D):**
  - Output Shape: (None, 109, 109, 64)
  - Description: This is a Conv2D layer, which applies a 2D convolution operation to the input. It convolves the input with 64 different filters, which can help identify various features in the data. The output has a shape of (None, 109, 109, 64).
- **max_pooling2d_4 (MaxPooling2D):**
  - Output Shape: (None, 54, 54, 64)
  - Description: Another MaxPooling2D layer that further reduces the spatial dimensions of the input. It halves both the height and width dimensions, resulting in an output shape of (None, 54, 54, 64).
- **conv2d_6 (Conv2D):**
  - Output Shape: (None, 52, 52, 128)
  - Description: Another Conv2D layer with 128 filters. It performs another round of convolution, potentially detecting higher-level features compared to the previous layer. The output shape is (None, 52, 52, 128).
- **conv2d_7 (Conv2D):**
  - Output Shape: (None, 50, 50, 128)
  - Description: This Conv2D layer continues to detect features using 128 filters. The output shape is (None, 50, 50, 128).
- **max_pooling2d_5 (MaxPooling2D):**
  - Output Shape: (None, 25, 25, 128)
  - Description: Another MaxPooling2D layer, further reducing spatial dimensions by half. It results in an output shape of (None, 25, 25, 128).
- **flatten_1 (Flatten):**
  - Output Shape: (None, 80000)
  - Description: This layer flattens the 3D output to a 1D tensor. It's preparing the data for the subsequent fully connected layers.
- **dense_2 (Dense):**
  - Output Shape: (None, 512)
  - Description: This is a fully connected layer with 512 units, applying a linear operation to the input data. It introduces non-linearity into the model.
- **dropout_1 (Dropout):**
  - Output Shape: (None, 512)
  - Description: Dropout is a regularization technique that randomly sets a fraction of input units to zero during training. It helps prevent overfitting.
- **dense_3 (Dense):**
  - Output Shape: (None, 2)
  - Description: This is the final dense layer with 2 units, which is suitable for binary classification. It applies a softmax activation function to output probabilities of the two classes.

Visualizing Accuracy and Loss

We plot the accuracy and loss against the number of epochs to observe their trends over the course of training.

Accuracy Plot (Left):

The blue line represents the training accuracy, showing how well the model performs on the training data as training progresses. The red line represents the validation accuracy, indicating how well the model generalizes to unseen validation data.

Loss Plot (Right):

The blue line represents the training loss, which measures the error between predicted and actual values during training. The red line represents the validation loss, providing insights into the model's performance on unseen data.

Next, We will visualize the architecture of our revised model using the `plot_model` function provided by Keras. This function creates a visual representation of the neural network's layers, including their shapes and names. We'll save this visualization as an image file for reference.

```
train_accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
train_loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(train_accuracy) + 1)

plt.figure(figsize=(15, 10))

# Plot precision vs. epoch
plt.subplot(1, 2, 1)
plt.plot(epochs, train_accuracy, 'b', label='Accuracy')
plt.plot(epochs, val_accuracy, 'r', label='Epochs')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Plot loss vs. epoch
plt.subplot(1, 2, 2)
plt.plot(epochs, train_loss, 'b', label='Training Loss')
plt.plot(epochs, val_loss, 'r', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

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

Visualizing Accuracy and Loss We plot the accuracy and loss against the number of epochs to observe their trends over the course of training for the revised model.

In this section, we define and train a revised Convolutional Neural Network (CNN) model for our classification task. This model architecture adheres to PEP8 compliance, ensuring readability and maintainability.

The CNN architecture comprises several convolutional and max-pooling layers, followed by fully connected layers. We use ReLU activation functions for the convolutional layers and a sigmoid activation function for the output layer in order to perform binary classification.

Additionally, we incorporate L2 regularization to mitigate overfitting. To monitor the training process, we implement early stopping, which halts training if the validation loss doesn't improve for a specified number of epochs.

The model is trained on the provided training data, and its performance is evaluated on the test data. We report the final test loss and accuracy. Furthermore, we generate a classification report that includes metrics such as F1-score, precision, accuracy, and recall.

Let's proceed with training and evaluating the model.

Let's take a look at its architecture and parameters. The summary provides a detailed overview of the layers, their output shapes, and the number of parameters.

To gain a visual understanding of the revised model's architecture, we've generated a diagram showcasing the various layers and their connections. This visualization is a helpful tool for comprehending the flow of information through the network.

In the diagram, you can observe the input layer, followed by a series of convolutional and pooling layers. The information flows through the network, undergoing transformations in each layer, until it reaches the output layer, which provides the final prediction.

This visualization aids in conceptualizing the neural network's structure and is valuable for debugging and optimizing the model.

The following code segment introduces a revised Convolutional Neural Network (CNN) model designed to optimize for the F1-score metric. This metric is crucial in scenarios where a balance between precision and recall is essential.

Model Architecture

The architecture consists of several convolutional layers followed by max-pooling layers, which effectively extract features from the input data. The model is designed to progressively reduce spatial dimensions while increasing depth. Batch Normalization layers have been added to stabilize and accelerate the training process.

Compilation and Training

The model is compiled with the Adam optimizer, binary cross-entropy loss function, and accuracy metric. It is then trained for an extended number of epochs to further refine its performance. Early stopping is employed to prevent overfitting.

Evaluation

The performance of the model is evaluated on the test data, providing insights into its generalization capabilities. The evaluation results include the test loss and accuracy.

Results and Classification Report

Furthermore, a detailed classification report is generated, encompassing various metrics such as F1-score, precision, accuracy, and recall. This comprehensive report offers a deeper understanding of the model's performance.

This revised model with a focus on the F1-score metric can be a powerful tool for tasks where achieving a balance between precision and recall is of utmost importance.

The revised Convolutional Neural Network (CNN) model, designed to optimize for the F1-score metric, has the following architecture:

- Input Shape: (224, 224, 3)

- Number of Classes: 1 (For binary classification)
- Regularization: L2 (0.01) applied to one convolutional layer

Model Architecture

1. Conv2D Layer: 32 filters, 3x3 kernel, ReLU activation
2. MaxPooling2D Layer: 2x2 pool size
3. Conv2D Layer: 32 filters, 3x3 kernel, ReLU activation
4. MaxPooling2D Layer: 2x2 pool size
5. Conv2D Layer: 64 filters, 3x3 kernel, ReLU activation
6. MaxPooling2D Layer: 2x2 pool size
7. Conv2D Layer: 64 filters, 3x3 kernel, ReLU activation
8. MaxPooling2D Layer: 2x2 pool size
9. Flatten Layer
10. Dense Layer: 512 units, ReLU activation
11. Dense Layer: 1 unit, Sigmoid activation (For binary classification)

Compilation

- Optimizer: Nadam
- Loss Function: Binary Cross-Entropy
- Metrics: Accuracy

Training

The model is trained with an extended number of epochs (100) using the training dataset. Early stopping is employed to prevent overfitting.

Evaluation

The model's performance is evaluated on the test data. The evaluation results include the test loss and accuracy.

Results and Classification Report

A detailed classification report is generated, encompassing various metrics such as F1-score, precision, accuracy, and recall. This comprehensive report offers a deeper understanding of the model's performance.

This visualization provides a graphical overview of the different layers, their connections, and the flow of information within the model. The architecture of the revised Convolutional Neural Network (CNN) model has been visually represented below:

In this section, we define a revised Convolutional Neural Network (CNN) model called `revised_model_5` and train it on the provided dataset. The model architecture consists of multiple convolutional layers, batch normalization, max-pooling, dropout, and fully connected layers. It is designed for binary classification.

Data Preprocessing

We use the `ImageDataGenerator` to augment the training data. The generator applies various transformations like rotation, shifting, shearing, zooming, and horizontal flipping to create a diverse set of training samples.

Model Architecture

The model (`revised_model_5`) is composed of the following layers:

1. Conv2D Layer (32 filters, 3x3 kernel, 'relu' activation) with Batch Normalization and Max Pooling.
2. Conv2D Layer (64 filters, 3x3 kernel, 'relu' activation) with Dropout, Batch Normalization, and Max Pooling.
3. Conv2D Layer (64 filters, 3x3 kernel, 'relu' activation) with Batch Normalization and Max Pooling.
4. Conv2D Layer (128 filters, 3x3 kernel, 'relu' activation) with Dropout, Batch Normalization, and Max Pooling.
5. Conv2D Layer (256 filters, 3x3 kernel, 'relu' activation) with Dropout, Batch Normalization, and Max Pooling.
6. Flatten Layer followed by Fully Connected Layers (128 units, 'relu' activation) with Dropout.
7. Output Dense Layer (1 unit, 'sigmoid' activation) for binary classification.

Model Compilation and Training

The model is compiled with RMSprop optimizer and binary cross-entropy loss. It is then trained for 50 epochs on the training data with early stopping.

Model Evaluation

The trained model is evaluated on the test data, and metrics such as test loss and accuracy are displayed. Additionally, a classification report is generated, including F1-score, precision, accuracy, and recall.

Summary

The summary of the model architecture is provided below, detailing the layers, output shapes, and trainable parameters.

To provide a visual representation of the model's architecture, we use the `plot_model` utility from TensorFlow. This tool generates an image illustrating the different layers and their connections.

```
import tensorflow as tf
from tensorflow.keras.applications import VGG19
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout, GlobalAveragePooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.callbacks import LearningRateScheduler
from tensorflow.keras.regularizers import l2
from keras.metrics import Precision, Accuracy, Recall

# Set image dimensions and classes
input_shape = (224, 224, 3)
num_classes = 2

# Define a learning rate scheduler callback using ReduceLROnPlateau
reduce_lr = ReduceLROnPlateau(
    monitor='val_accuracy', # The metric to monitor for improvement (validation accuracy in this case).
    factor=0.3,             # The factor by which the learning rate will be reduced.
    patience=2,            # Number of epochs with no improvement after which learning rate will be reduced.
    verbose=1,              # Display messages about learning rate adjustments.
    mode='max',            # Automatically determines the direction of improvement based on the monitored metric.
    min_delta=0.0001,       # Minimum change in the monitored metric to qualify as an improvement
    cooldown=4,             # Number of epochs to wait before resuming normal learning after a learning rate reduction.
    min_lr=0.000000001            # The lower bound for the learning rate.
)

# Define the EarlyStopping callback
early_stopping = EarlyStopping(
    monitor='val_loss',  # Monitor validation loss
    patience=5,           # Number of epochs with no improvement before stopping
    restore_best_weights=True  # Restore the best model weights
)

# Create data generators for train and test sets
train_generator = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
)

test_generator = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255)
val_generator = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255)
train_data = train_generator.flow_from_directory(
    train_data_path ,
    target_size=input_shape[:2],
    batch_size=32,
    class_mode='binary',
    shuffle=True
)

test_data = test_generator.flow_from_directory(
    test_data_path,
    target_size=input_shape[:2],
    batch_size=32,
    class_mode='binary',
    shuffle=False
)
val_data = val_generator.flow_from_directory(
    val_data_path,
    target_size=input_shape[:2],
    batch_size=32,
    class_mode='binary',
    shuffle=False
)

# Load the pre-trained VGG19 model without the top layer
base_model = VGG19(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Create a new model by adding custom layers on top of the base model
vgg19_model = Sequential([
    base_model,  # Add the VGG19 base model


    # Add Global Average Pooling layer to reduce spatial dimensions
```

```
    # Add Global Average Pooling layer to reduce spatial dimensions
    GlobalAveragePooling2D(),

    Dense(units=256, activation='relu'),
    Dense(units=1, activation='sigmoid'),
])

# Freeze the weights of the pre-trained layers
for layer in base_model.layers:
    layer.trainable = False
# Define your desired initial learning rate
# initial_learning_rate = 0.0001

# Define a function to schedule the learning rate
#def lr_schedule(epoch):
#    return initial_learning_rate * 0.01 ** epoch

# Create a learning rate scheduler
#lr_scheduler = LearningRateScheduler(lr_schedule)
metrics =[
    Precision(name='precision'),
    'accuracy',
    Recall(name='recall')
]
# Compile the model
vgg19_model.compile(optimizer='Adam',
                        loss='binary_crossentropy',
                        metrics=metrics)

# Print model summary
vgg19_model.summary()
```

```
Found 2146 images belonging to 2 classes.
Found 624 images belonging to 2 classes.
Found 1893 images belonging to 2 classes.
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg19 (Functional)          (None, 7, 7, 512)         20024384

 global_average_pooling2d_3  (None, 512)               0
  (GlobalAveragePooling2D)

 dense_12 (Dense)            (None, 256)               131328

 dense_13 (Dense)            (None, 1)                 257

=================================================================
Total params: 20155969 (76.89 MB)
Trainable params: 131585 (514.00 KB)
Non-trainable params: 20024384 (76.39 MB)
_________________________________________________________________
```

Pneumonia Classification with VGG19-based CNN

In this section, we'll be tackling the task of pneumonia classification using a Convolutional Neural Network (CNN) architecture. Specifically, we'll leverage the power of the VGG19 model, a popular pre-trained deep learning architecture, to improve the performance of our classification task.

Dataset Overview

The dataset comprises chest X-ray images, annotated with labels indicating whether a patient has pneumonia or not. Our objective is to build a robust model that can accurately distinguish between these two classes, thus aiding in automated pneumonia diagnosis.

Model Architecture

We'll employ a VGG19-based CNN, utilizing a transfer learning approach. The VGG19 model has been pre-trained on a large-scale image classification task and has learned rich feature representations. By leveraging this pre-trained architecture, we can potentially enhance our model's ability to extract relevant features from the chest X-ray images.

Training and Evaluation

We'll train the model on a carefully curated dataset and evaluate its performance on a separate test set. We'll monitor key metrics such as accuracy, loss, and potentially other evaluation metrics like precision, recall, and F1-score.

Let's dive into the code and build this powerful pneumonia classification model!

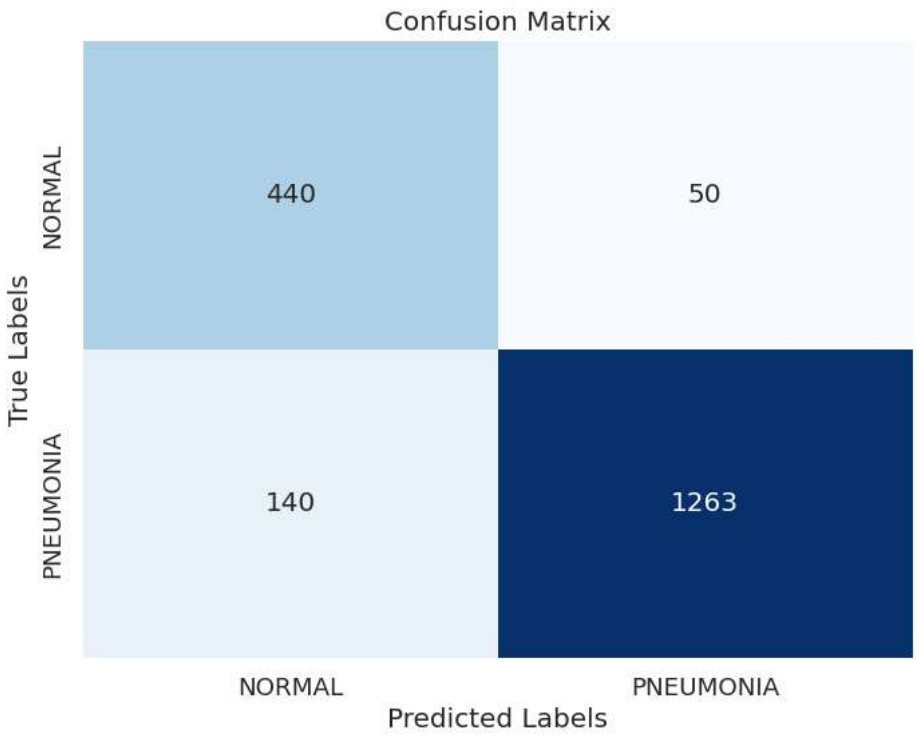
```
# Import callbacks to adjust and save models while training
from keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
from tensorflow.keras.regularizers import l2
```

```python
# Create ModelCheckpoint for callbacks
# autosave best Model
best_model_file = "vgg19.h5"
best_model = ModelCheckpoint(
    best_model_file, monitor='val_accuracy', mode='max', verbose=1, save_best_only=True)


# Train the fifth revised model with a higher number of epochs
history = vgg19_model.fit(
    train_data,
    steps_per_epoch=train_data.samples // train_data.batch_size,
    epochs=50,  # increased number of epochs
    validation_data=val_data,
    validation_steps=val_data.samples // val_data.batch_size,
    #class_weight=class_weight,
    callbacks=[best_model, early_stopping, reduce_lr],

)


# Evaluate the model on the val data
val_loss, val_precision, val_accuracy, val_recall = vgg19_model.evaluate(val_data)
print(f"Val Loss: {val_loss:.4f}, Val Precision: {val_precision:.4f},Val Accuracy: {val_accuracy:.4f}, Val Recall: {val_recall:.4f}")

# Predict labels for the val data
y_pred = vgg19_model.predict(val_data)
y_pred_binary = np.argmax(y_pred, axis=1)  # Convert probabilities to binary predictions

# True labels for the val data
y_true = val_data.classes
```

```
Epoch 1/50
67/67 [==============================] - ETA: 0s - loss: 0.4610 - precision: 0.8118 - accuracy: 0.8046 - recall: 0.7926
Epoch 1: val_accuracy improved from -inf to 0.86388, saving model to vgg19.h5
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3000: UserWarning: You are saving your model as an HDF5 file via `model.save()`. T
  saving_api.save_model(
67/67 [==============================] - 72s 959ms/step - loss: 0.4610 - precision: 0.8118 - accuracy: 0.8046 - recall: 0.7926 - val_loss: 0.3492 - val
Epoch 2/50
67/67 [==============================] - ETA: 0s - loss: 0.3238 - precision: 0.8833 - accuracy: 0.8704 - recall: 0.8532
Epoch 2: val_accuracy did not improve from 0.86388
67/67 [==============================] - 63s 936ms/step - loss: 0.3238 - precision: 0.8833 - accuracy: 0.8704 - recall: 0.8532 - val_loss: 0.3671 - val
Epoch 3/50
67/67 [==============================] - ETA: 0s - loss: 0.2913 - precision: 0.8963 - accuracy: 0.8860 - recall: 0.8735
Epoch 3: val_accuracy improved from 0.86388 to 0.89195, saving model to vgg19.h5
67/67 [==============================] - 65s 964ms/step - loss: 0.2913 - precision: 0.8963 - accuracy: 0.8860 - recall: 0.8735 - val_loss: 0.2590 - val
Epoch 4/50
67/67 [==============================] - ETA: 0s - loss: 0.2689 - precision: 0.9048 - accuracy: 0.8964 - recall: 0.8869
Epoch 4: val_accuracy did not improve from 0.89195
67/67 [==============================] - 66s 989ms/step - loss: 0.2689 - precision: 0.9048 - accuracy: 0.8964 - recall: 0.8869 - val_loss: 0.3010 - val
Epoch 5/50
67/67 [==============================] - ETA: 0s - loss: 0.2478 - precision: 0.9109 - accuracy: 0.9016 - recall: 0.8902
Epoch 5: val_accuracy did not improve from 0.89195

Epoch 5: ReduceLROnPlateau reducing learning rate to 0.0003000000142492354.
67/67 [==============================] - 64s 963ms/step - loss: 0.2478 - precision: 0.9109 - accuracy: 0.9016 - recall: 0.8902 - val_loss: 0.2661 - val
Epoch 6/50
67/67 [==============================] - ETA: 0s - loss: 0.2301 - precision: 0.9305 - accuracy: 0.9172 - recall: 0.9013
Epoch 6: val_accuracy did not improve from 0.89195
67/67 [==============================] - 64s 957ms/step - loss: 0.2301 - precision: 0.9305 - accuracy: 0.9172 - recall: 0.9013 - val_loss: 0.2786 - val
Epoch 7/50
67/67 [==============================] - ETA: 0s - loss: 0.2191 - precision: 0.9373 - accuracy: 0.9172 - recall: 0.8937
Epoch 7: val_accuracy improved from 0.89195 to 0.89936, saving model to vgg19.h5
67/67 [==============================] - 68s 1s/step - loss: 0.2191 - precision: 0.9373 - accuracy: 0.9172 - recall: 0.8937 - val_loss: 0.2408 - val_pr
Epoch 8/50
67/67 [==============================] - ETA: 0s - loss: 0.2186 - precision: 0.9287 - accuracy: 0.9196 - recall: 0.9094
Epoch 8: val_accuracy did not improve from 0.89936
67/67 [==============================] - 65s 976ms/step - loss: 0.2186 - precision: 0.9287 - accuracy: 0.9196 - recall: 0.9094 - val_loss: 0.2570 - val
Epoch 9/50
67/67 [==============================] - ETA: 0s - loss: 0.2241 - precision: 0.9145 - accuracy: 0.9101 - recall: 0.9041
Epoch 9: val_accuracy improved from 0.89936 to 0.90042, saving model to vgg19.h5
67/67 [==============================] - 65s 973ms/step - loss: 0.2241 - precision: 0.9145 - accuracy: 0.9101 - recall: 0.9041 - val_loss: 0.2503 - val
Epoch 10/50
67/67 [==============================] - ETA: 0s - loss: 0.2117 - precision: 0.9318 - accuracy: 0.9215 - recall: 0.9088
Epoch 10: val_accuracy improved from 0.90042 to 0.90201, saving model to vgg19.h5
67/67 [==============================] - 65s 978ms/step - loss: 0.2117 - precision: 0.9318 - accuracy: 0.9215 - recall: 0.9088 - val_loss: 0.2495 - val
Epoch 11/50
67/67 [==============================] - ETA: 0s - loss: 0.2170 - precision: 0.9279 - accuracy: 0.9182 - recall: 0.9058
Epoch 11: val_accuracy improved from 0.90201 to 0.90254, saving model to vgg19.h5
67/67 [==============================] - 65s 964ms/step - loss: 0.2170 - precision: 0.9279 - accuracy: 0.9182 - recall: 0.9058 - val_loss: 0.2512 - val
Epoch 12/50
67/67 [==============================] - ETA: 0s - loss: 0.1940 - precision: 0.9448 - accuracy: 0.9338 - recall: 0.9216
Epoch 12: val_accuracy did not improve from 0.90254
67/67 [==============================] - 63s 936ms/step - loss: 0.1940 - precision: 0.9448 - accuracy: 0.9338 - recall: 0.9216 - val_loss: 0.2861 - val
60/60 [==============================] - 19s 306ms/step - loss: 0.2403 - precision: 0.9619 - accuracy: 0.8996 - recall: 0.9002
Val Loss: 0.2403, Val Precision: 0.9619,Val Accuracy: 0.8996, Val Recall: 0.9002
60/60 [==============================] - 18s 288ms/step
```

```
# Predict labels for the test data
y_pred = vgg19_model.predict(val_data)
threshold = 0.5
y_pred_binary = (y_pred > threshold).astype(int)
y_true = np.concatenate([y_true for _ in range(y_pred_binary.shape[0] // len(y_true))])  # Match shapes

# Now compute the confusion matrix
confusion = confusion_matrix(y_true, y_pred_binary)

# Plot the confusion matrix as a heatmap
plt.figure(figsize=(8, 6))
sns.set(font_scale=1.2)
sns.heatmap(confusion, annot=True, fmt="d", cmap="Blues", cbar=False,
            xticklabels=val_data.class_indices.keys(),
            yticklabels=val_data.class_indices.keys())
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```

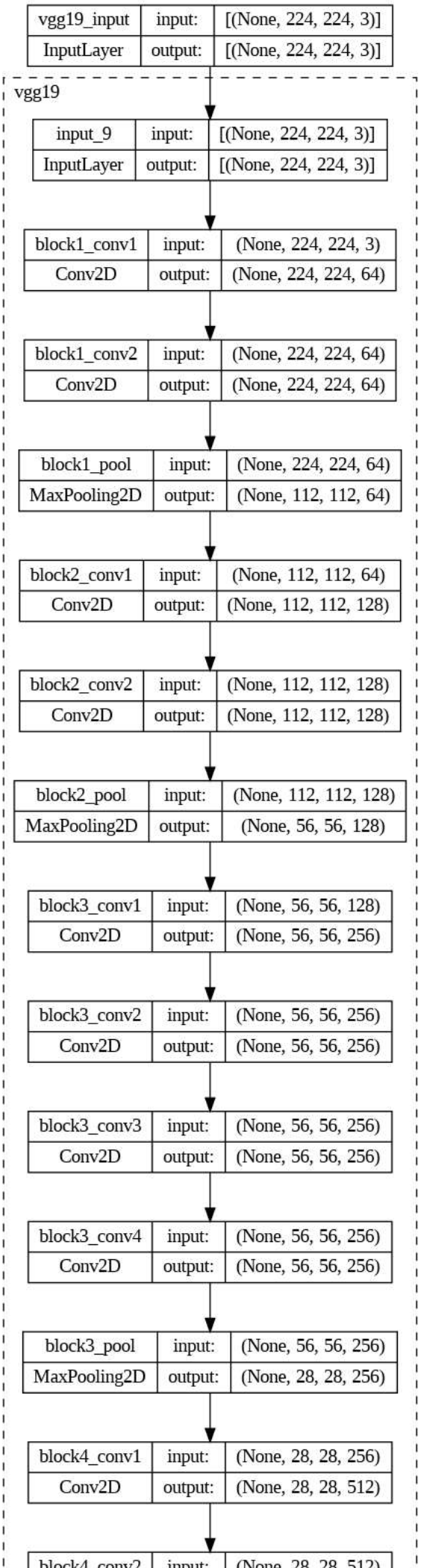
```
60/60 [==============================] - 17s 285ms/step
```

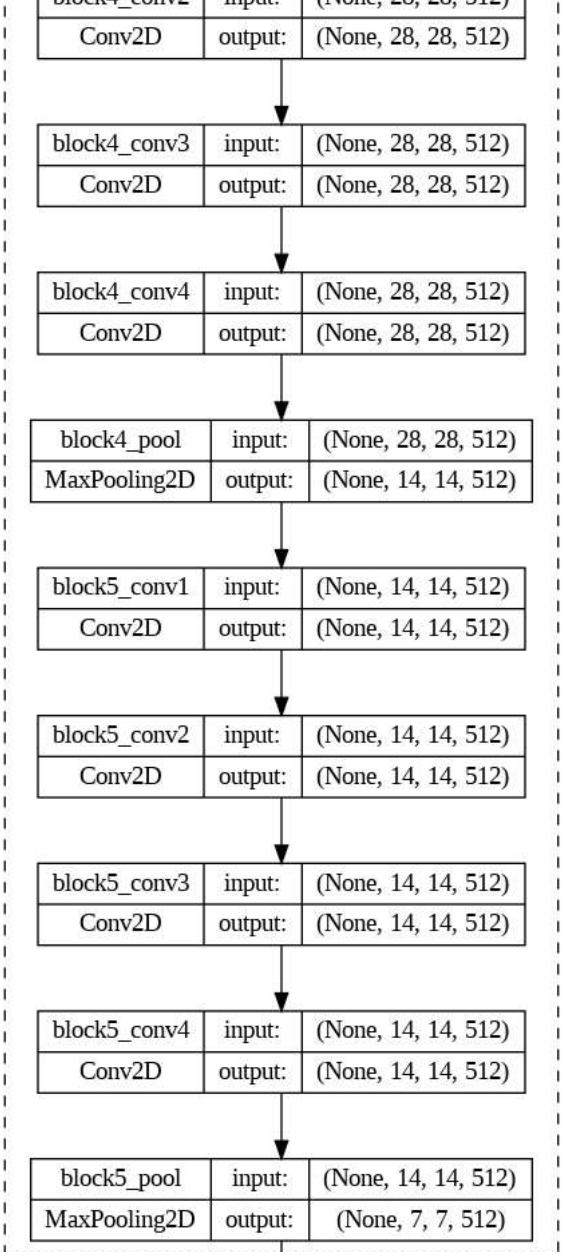


In this section, we'll generate a visual representation of the architecture of our pneumonia classification model (model_5). This visualization will provide a clear overview of the different layers and connections within the network. Each node in the diagram represents a layer, and the connections indicate how information flows through the network.

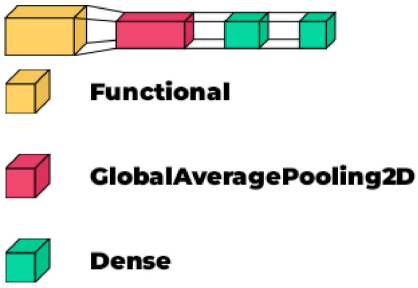
We'll save this visualization as an image file ('model_5_architecture.png') for future reference and documentation. This visual representation will be helpful for understanding the structure of our neural network and for sharing insights with others.

```
from tensorflow.keras.utils import plot_model
# Save a visualization of the model's architecture to a file
plot_model(vgg19_model, to_file='vgg19_model.png', show_shapes=True,show_layer_names=True, expand_nested=True)
```


vgg19_input
input:
[(None, 224, 224, 3)]
InputLayer
output:
[(None, 224, 224, 3)]
vgg19
input_9
input:
[(None, 224, 224, 3)]
InputLayer
output:
[(None, 224, 224, 3)]
block1_conv1
input:
(None, 224, 224, 3)
Conv2D
output:
(None, 224, 224, 64)
block1_conv2
input:
(None, 224, 224, 64)
Conv2D
output:
(None, 224, 224, 64)
block1_pool
input:
(None, 224, 224, 64)
MaxPooling2D
output:
(None, 112, 112, 64)
block2_conv1
input:
(None, 112, 112, 64)
Conv2D
output:
(None, 112, 112, 128)
block2_conv2
input:
(None, 112, 112, 128)
Conv2D
output:
(None, 112, 112, 128)
block2_pool
input:
(None, 112, 112, 128)
MaxPooling2D
output:
(None, 56, 56, 128)
block3_conv1
input:
(None, 56, 56, 128)
Conv2D
output:
(None, 56, 56, 256)
block3_conv2
input:
(None, 56, 56, 256)
Conv2D
output:
(None, 56, 56, 256)
block3_conv3
input:
(None, 56, 56, 256)
Conv2D
output:
(None, 56, 56, 256)
block3_conv4
input:
(None, 56, 56, 256)
Conv2D
output:
(None, 56, 56, 256)
block3_pool
input:
(None, 56, 56, 256)
MaxPooling2D
output:
(None, 28, 28, 256)
block4_conv1
input:
(None, 28, 28, 256)
Conv2D
output:
(None, 28, 28, 512)



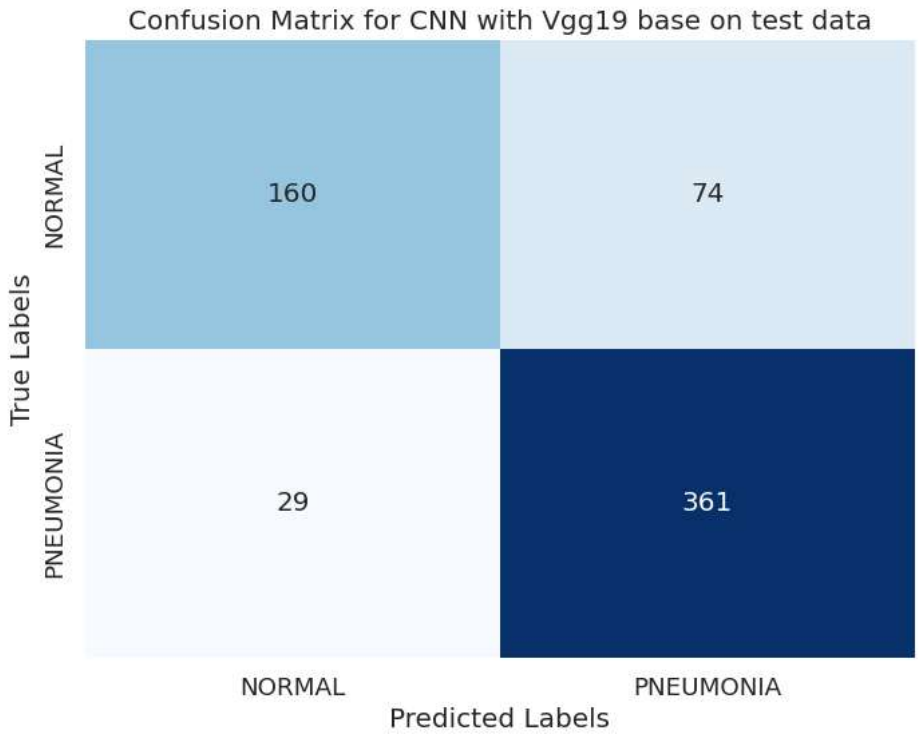

```
import visualkeras
from PIL import ImageFont
font = ImageFont.truetype("Montserrat-Black.ttf", 18)
visualkeras.layered_view(vgg19_model,
                         to_file='vgg19_model_architecture_view.png',
                         legend=True,
                         font=font, draw_volume=True, spacing=30)
```




```
y_true = test_data.classes
# Predict labels for the test data
y_pred = vgg19_model.predict(test_data)
threshold = 0.5
y_pred_binary = (y_pred > threshold).astype(int)
y_true = np.concatenate([y_true for _ in range(y_pred_binary.shape[0] // len(y_true))])  # Match shapes
# True labels for the val data

# Now compute the confusion matrix
confusion = confusion_matrix(y_true, y_pred_binary)

# Plot the confusion matrix as a heatmap
```

```
plt.figure(figsize=(8, 6))
sns.set(font_scale=1.2)
sns.heatmap(confusion, annot=True, fmt="d", cmap="Blues", cbar=False,
            xticklabels=test_data.class_indices.keys(),
            yticklabels=test_data.class_indices.keys())
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix for CNN with Vgg19 base on test data')
plt.show()
```

```
20/20 [==============================] - 5s 238ms/step
```

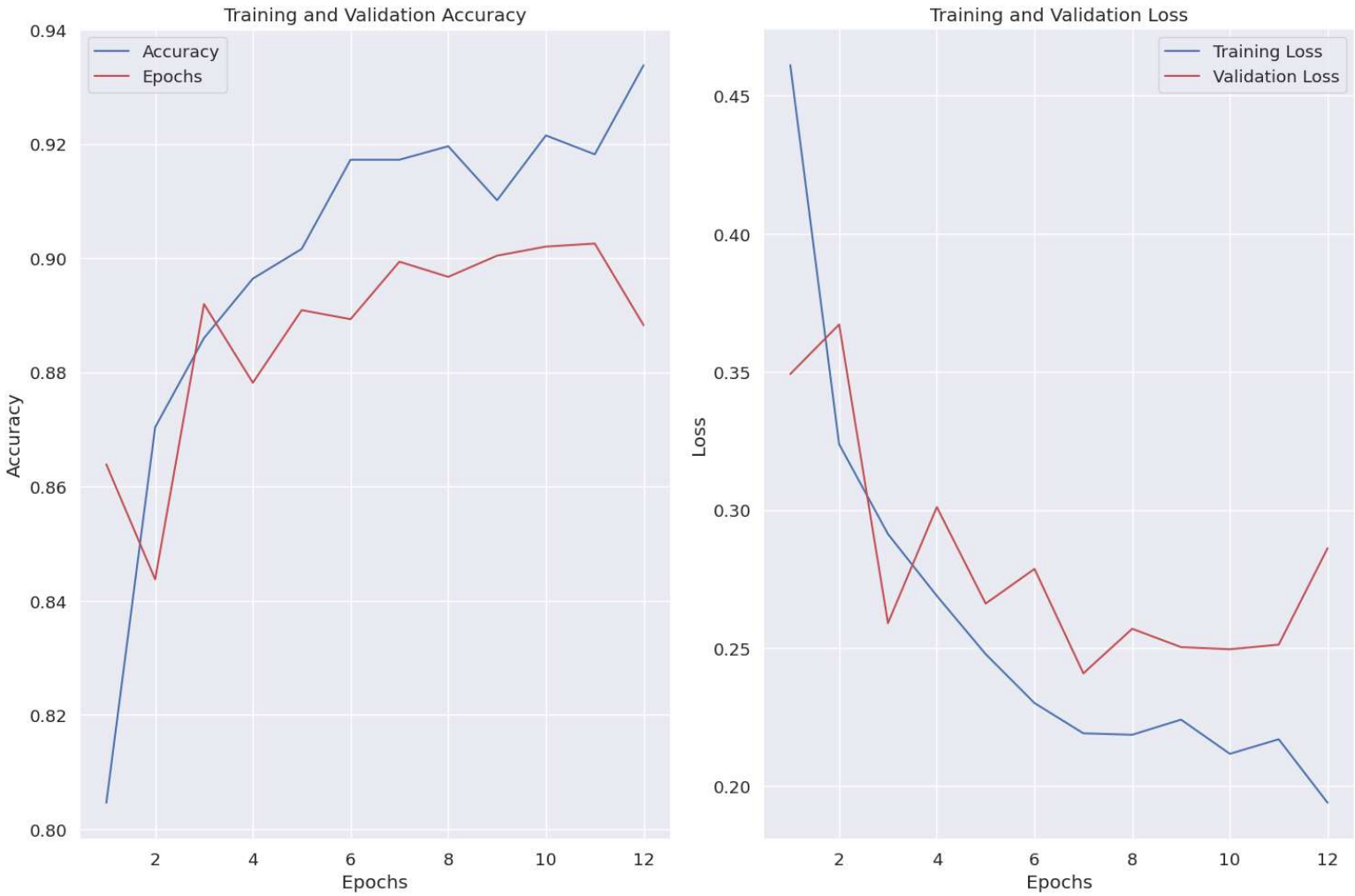



```
# Print classification report (F1-score, precision, accuracy, recall)
report = classification_report(y_true, y_pred_binary, target_names=test_data.class_indices)
print("CNN with vgg19 base layer Classification Report on Test data: \n", report)
```

```
CNN with vgg19 base layer Classification Report on Test data:
                precision    recall  f1-score   support

           NORMAL       0.85      0.68      0.76       234
        PNEUMONIA       0.83      0.93      0.88       390

         accuracy                           0.83       624
        macro avg       0.84      0.80      0.82       624
     weighted avg       0.84      0.83      0.83       624
```

```
train_accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
train_loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(train_accuracy) + 1)

plt.figure(figsize=(15, 10))

# Plot precision vs. epoch
plt.subplot(1, 2, 1)
plt.plot(epochs, train_accuracy, 'b', label='Accuracy')
plt.plot(epochs, val_accuracy, 'r', label='Epochs')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Plot loss vs. epoch
plt.subplot(1, 2, 2)
plt.plot(epochs, train_loss, 'b', label='Training Loss')
plt.plot(epochs, val_loss, 'r', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
```

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



In this code snippet, we're setting up a transfer learning pipeline using the MobileNetV2 architecture pretrained on ImageNet:

1. We define the input shape and number of classes for our specific binary classification task.
2. EarlyStopping callback is set up to monitor validation loss and prevent overfitting.
3. Data generators are created for both the training and testing sets with augmentation for the training set.
4. The InceptionV3 base model (pre-trained on ImageNet) is loaded and its layers are frozen to retain pre-trained weights.
5. A custom classification head is added, consisting of Global Average Pooling, Dense, Dropout, and Dense layers.
6. The base model and custom head are combined to form the final model.
7. The model is compiled using the Adam optimizer and sparse categorical crossentropy loss.
8. A summary of the model architecture is printed for review.

```
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.models import Model
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout
from tensorflow.keras.optimizers import Adam

# Load InceptionV3 base model (pre-trained on ImageNet)
base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=input_shape)

best_model_file = "MobileNetV2.h5"
best_model = ModelCheckpoint(
    best_model_file, monitor='val_accuracy', mode='max', verbose=1, save_best_only=True)
```

```
# Freeze the layers of the pre-trained model
for layer in base_model.layers:
    layer.trainable = False

# Add custom classification head
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='softmax')(x)
x = Dropout(0.2)(x)
predictions = Dense(1, activation='sigmoid')(x)

# Combine base model and custom head
model_mobilenetv2 = Model(inputs=base_model.input, outputs=predictions)

# Compile the model
model_mobilenetv2.compile(optimizer=Adam(),
                          loss='binary_crossentropy',
                          metrics=metrics)

# Print a summary of the model architecture
model_mobilenetv2.summary()
```

```
      chNormalization)

      block_12_depthwise_relu (R  (None, 14, 14, 576)          0         ['block_12_depthwise_BN[0][0]'
      eLU)                                                               ]

      block_12_project (Conv2D)   (None, 14, 14, 96)           55296     ['block_12_depthwise_relu[0][0
                                                                         ]']

      block_12_project_BN (Batch  (None, 14, 14, 96)           384       ['block_12_project[0][0]']
      Normalization)

      block_12_add (Add)          (None, 14, 14, 96)           0         ['block_11_add[0][0]',
                                                                          'block_12_project_BN[0][0]']

      block_13_expand (Conv2D)    (None, 14, 14, 576)          55296     ['block_12_add[0][0]']

      block_13_expand_BN (BatchN  (None, 14, 14, 576)          2304      ['block_13_expand[0][0]']
      ormalization)

      block_13_expand_relu (ReLU  (None, 14, 14, 576)          0         ['block_13_expand_BN[0][0]']
      )

      block_13_pad (ZeroPadding2  (None, 15, 15, 576)          0         ['block_13_expand_relu[0][0]']
      D)

      block_13_depthwise (Depthw  (None, 7, 7, 576)            5184      ['block_13_pad[0][0]']
      iseConv2D)

      block_13_depthwise_BN (Bat  (None, 7, 7, 576)            2304      ['block_13_depthwise[0][0]']
      chNormalization)

      block_13_depthwise_relu (R  (None, 7, 7, 576)            0         ['block_13_depthwise_BN[0][0]'
      eLU)                                                               ]

      block_13_project (Conv2D)   (None, 7, 7, 160)            92160     ['block_13_depthwise_relu[0][0
                                                                         ]']

      block_13_project_BN (Batch  (None, 7, 7, 160)            640       ['block_13_project[0][0]']
      Normalization)

      block_14_expand (Conv2D)    (None, 7, 7, 960)            153600    ['block_13_project_BN[0][0]']

      block_14_expand_BN (BatchN  (None, 7, 7, 960)            3840      ['block_14_expand[0][0]']
      ormalization)

      block_14_expand_relu (ReLU  (None, 7, 7, 960)            0         ['block_14_expand_BN[0][0]']
      )

      block_14_depthwise (Depthw  (None, 7, 7, 960)            8640      ['block_14_expand_relu[0][0]']
      iseConv2D)

      block_14_depthwise_BN (Bat  (None, 7, 7, 960)            3840      ['block_14_depthwise[0][0]']
      chNormalization)

      block_14_depthwise_relu (R  (None, 7, 7, 960)            0         ['block_14_depthwise_BN[0][0]'
      eLU)                                                               ]

      block_14_project (Conv2D)   (None, 7, 7, 160)            153600    ['block_14_depthwise_relu[0][0
                                                                         ]']
```

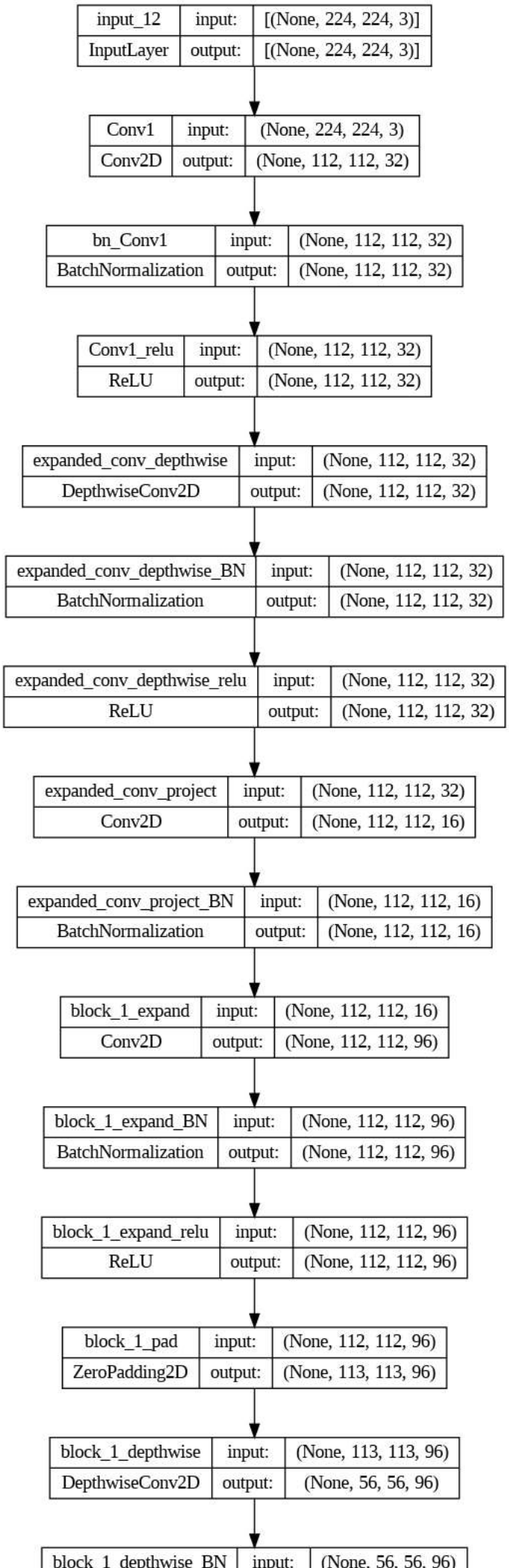
```
history = model_mobilenetv2.fit(
    train_data,
    steps_per_epoch=train_data.samples // train_data.batch_size,
    epochs=50,  # increased number of epochs
    validation_data=val_data,
    validation_steps=val_data.samples // val_data.batch_size,
```

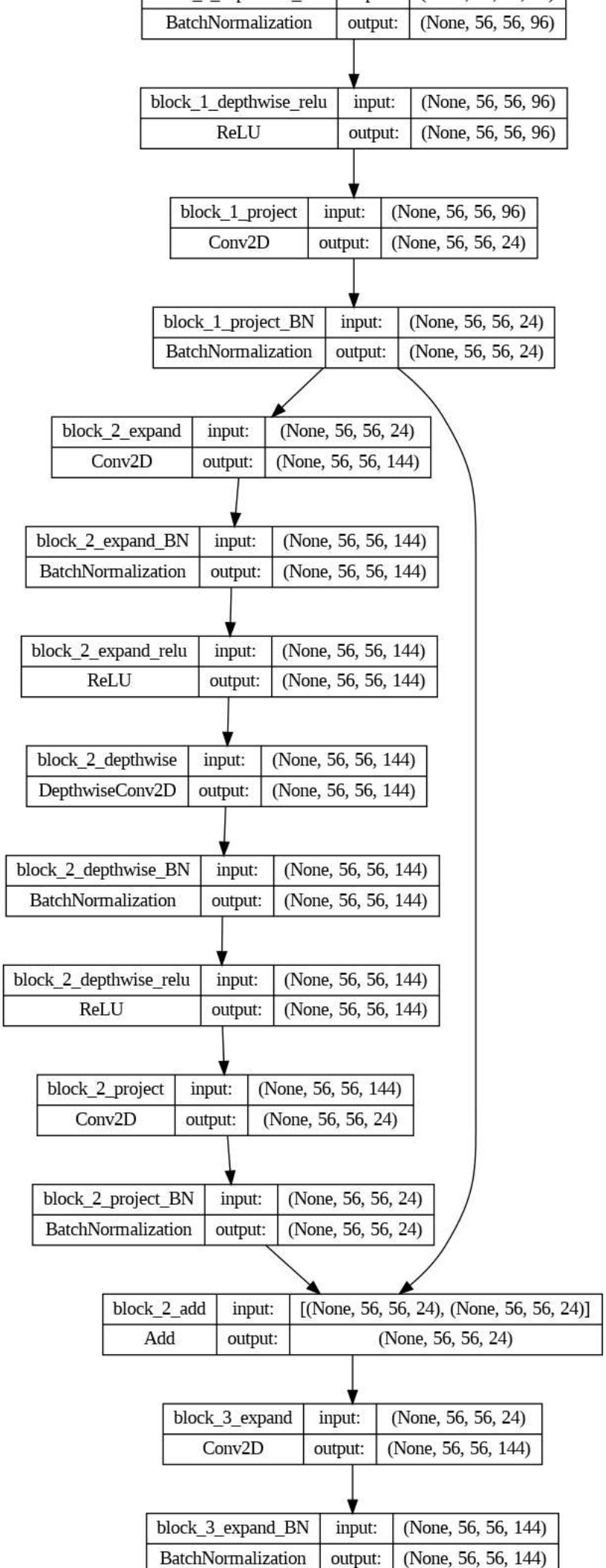
```
    #class_weight=class_weight,
    callbacks=[best_model, early_stopping, reduce_lr],

)


# Evaluate the model on the val data
val_loss, val_precision, val_accuracy, val_recall = model_mobilenetv2.evaluate(val_data)
print(f"Val Loss: {val_loss:.4f}, Val Precision: {val_precision:.4f},Val Accuracy: {val_accuracy:.4f}, Val Recall: {val_recall:.4f}")

# Predict labels for the val data
y_pred = model_mobilenetv2.predict(val_data)
y_pred_binary = np.argmax(y_pred, axis=1)  # Convert probabilities to binary predictions

# True labels for the val data
y_true = val_data.classes

# Print classification report (F1-score, precision, accuracy, recall)
report = classification_report(y_true, y_pred_binary, target_names=test_data.class_indices)
print("MobileNet v2 Model Classification Report: \n", report)
```

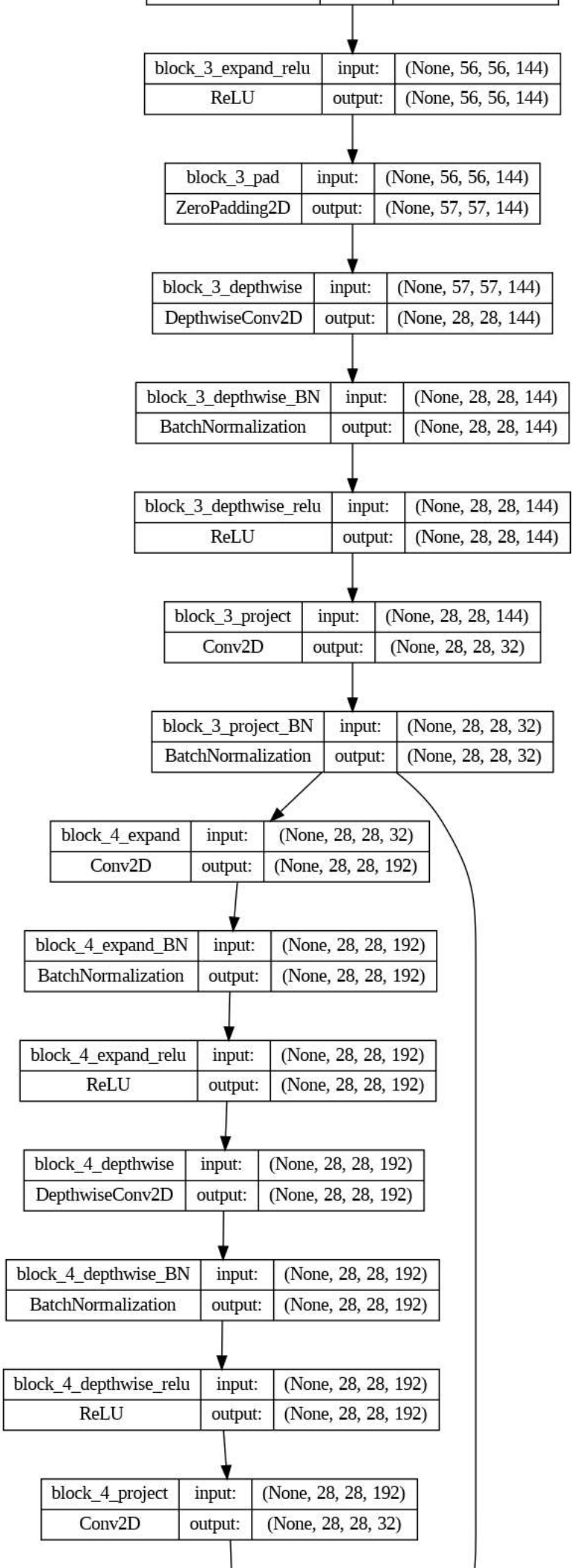
```
Epoch 7/50
67/67 [==============================] - ETA: 0s - loss: 0.5404 - precision: 0.8135 - accuracy: 0.8567 - recall: 0.9252
Epoch 7: val_accuracy improved from 0.92479 to 0.92638, saving model to MobileNetV2.h5
67/67 [==============================] - 63s 949ms/step - loss: 0.5404 - precision: 0.8135 - accuracy: 0.8567 - recall: 0.9252 - val_loss: 0.5426 -
Epoch 8/50
67/67 [==============================] - ETA: 0s - loss: 0.5347 - precision: 0.8059 - accuracy: 0.8529 - recall: 0.9312
Epoch 8: val_accuracy did not improve from 0.92638
67/67 [==============================] - 63s 946ms/step - loss: 0.5347 - precision: 0.8059 - accuracy: 0.8529 - recall: 0.9312 - val_loss: 0.5392 -
Epoch 9/50
67/67 [==============================] - ETA: 0s - loss: 0.5307 - precision: 0.7976 - accuracy: 0.8467 - recall: 0.9290
Epoch 9: val_accuracy did not improve from 0.92638

Epoch 9: ReduceLROnPlateau reducing learning rate to 9.000000427477062e-05.
67/67 [==============================] - 63s 943ms/step - loss: 0.5307 - precision: 0.7976 - accuracy: 0.8467 - recall: 0.9290 - val_loss: 0.5335 -
Epoch 10/50
67/67 [==============================] - ETA: 0s - loss: 0.5226 - precision: 0.8140 - accuracy: 0.8614 - recall: 0.9366
Epoch 10: val_accuracy did not improve from 0.92638
67/67 [==============================] - 64s 956ms/step - loss: 0.5226 - precision: 0.8140 - accuracy: 0.8614 - recall: 0.9366 - val_loss: 0.5334 -
Epoch 11/50
67/67 [==============================] - ETA: 0s - loss: 0.5227 - precision: 0.7935 - accuracy: 0.8458 - recall: 0.9336
Epoch 11: val_accuracy did not improve from 0.92638
67/67 [==============================] - 62s 923ms/step - loss: 0.5227 - precision: 0.7935 - accuracy: 0.8458 - recall: 0.9336 - val_loss: 0.5336 -
Epoch 12/50
67/67 [==============================] - ETA: 0s - loss: 0.5192 - precision: 0.8089 - accuracy: 0.8571 - recall: 0.9346
Epoch 12: val_accuracy did not improve from 0.92638
67/67 [==============================] - 62s 929ms/step - loss: 0.5192 - precision: 0.8089 - accuracy: 0.8571 - recall: 0.9346 - val_loss: 0.5294 -
Epoch 13/50
67/67 [==============================] - ETA: 0s - loss: 0.5203 - precision: 0.8071 - accuracy: 0.8571 - recall: 0.9362
Epoch 13: val_accuracy did not improve from 0.92638
67/67 [==============================] - 62s 932ms/step - loss: 0.5203 - precision: 0.8071 - accuracy: 0.8571 - recall: 0.9362 - val_loss: 0.5271 -
Epoch 14/50
67/67 [==============================] - ETA: 0s - loss: 0.5137 - precision: 0.8140 - accuracy: 0.8633 - recall: 0.9424
Epoch 14: val_accuracy did not improve from 0.92638

Epoch 14: ReduceLROnPlateau reducing learning rate to 2.700000040931627e-05.
67/67 [==============================] - 62s 924ms/step - loss: 0.5137 - precision: 0.8140 - accuracy: 0.8633 - recall: 0.9424 - val_loss: 0.5266 -
Epoch 15/50
67/67 [==============================] - ETA: 0s - loss: 0.5125 - precision: 0.7989 - accuracy: 0.8500 - recall: 0.9357
Epoch 15: val_accuracy did not improve from 0.92638
67/67 [==============================] - 62s 931ms/step - loss: 0.5125 - precision: 0.7989 - accuracy: 0.8500 - recall: 0.9357 - val_loss: 0.5259 -
Epoch 16/50
67/67 [==============================] - ETA: 0s - loss: 0.5120 - precision: 0.8147 - accuracy: 0.8619 - recall: 0.9366
Epoch 16: val_accuracy did not improve from 0.92638
67/67 [==============================] - 64s 958ms/step - loss: 0.5120 - precision: 0.8147 - accuracy: 0.8619 - recall: 0.9366 - val_loss: 0.5240 -
Epoch 17/50
67/67 [==============================] - ETA: 0s - loss: 0.5138 - precision: 0.8033 - accuracy: 0.8519 - recall: 0.9330
Epoch 17: val_accuracy did not improve from 0.92638
67/67 [==============================] - 61s 907ms/step - loss: 0.5138 - precision: 0.8033 - accuracy: 0.8519 - recall: 0.9330 - val_loss: 0.5232 -
Epoch 18/50
67/67 [==============================] - ETA: 0s - loss: 0.5109 - precision: 0.8079 - accuracy: 0.8581 - recall: 0.9406
Epoch 18: val_accuracy did not improve from 0.92638
67/67 [==============================] - 61s 915ms/step - loss: 0.5109 - precision: 0.8079 - accuracy: 0.8581 - recall: 0.9406 - val_loss: 0.5231 -
Epoch 19/50
67/67 [==============================] - ETA: 0s - loss: 0.5121 - precision: 0.8031 - accuracy: 0.8553 - recall: 0.9425
Epoch 19: val_accuracy did not improve from 0.92638

Epoch 19: ReduceLROnPlateau reducing learning rate to 8.100000013655517e-06.
67/67 [==============================] - 60s 902ms/step - loss: 0.5121 - precision: 0.8031 - accuracy: 0.8553 - recall: 0.9425 - val_loss: 0.5242 -
```

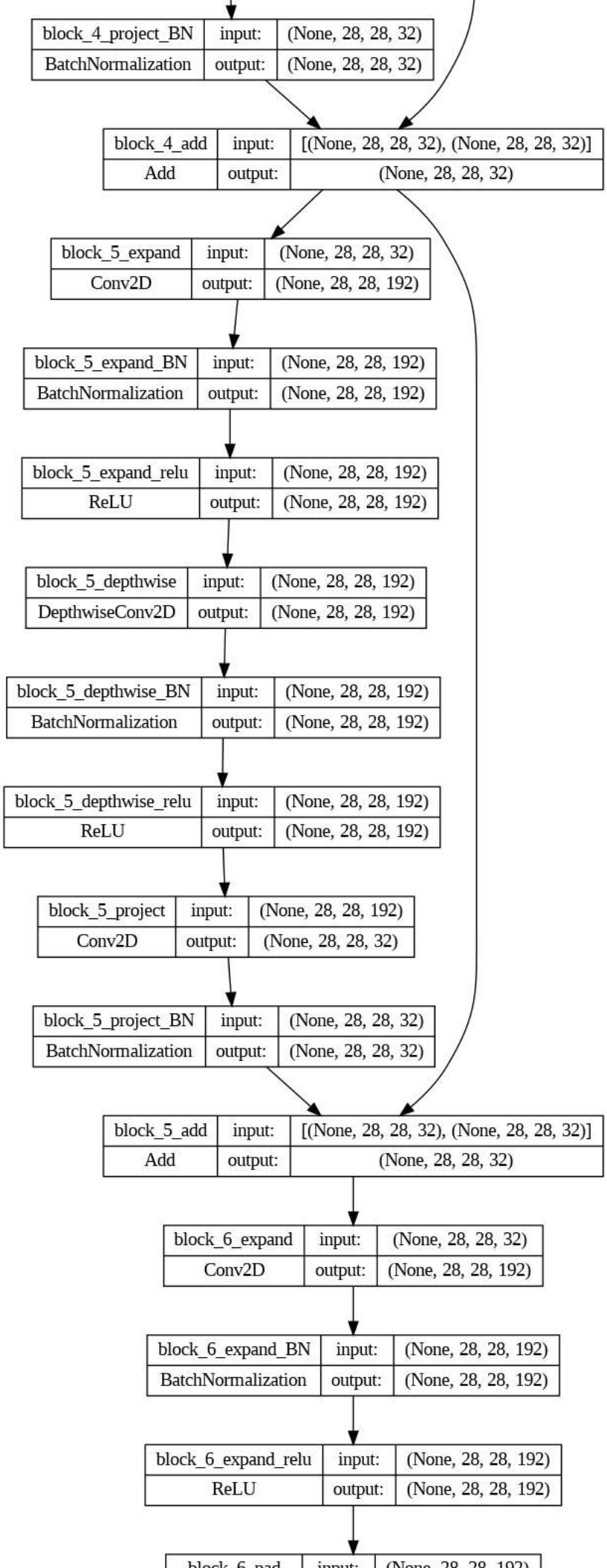
This model may be too complicated to use with our simple dataset, more research is needed to test the viability of using this model for our purposes.

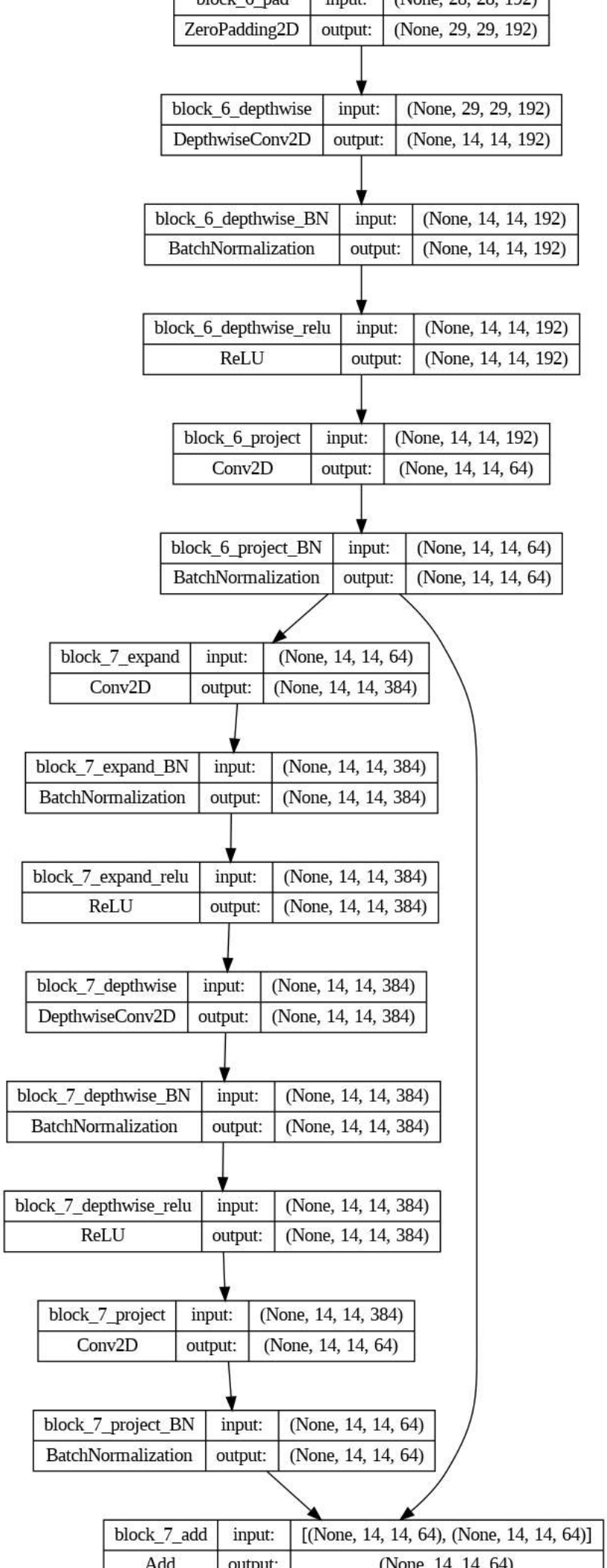
```
from tensorflow.keras.utils import plot_model
# Save a visualization of the model's architecture to a file
plot_model(model_mobilenetv2, to_file='model_mobilenetv2_architecture.png', show_shapes=True,show_layer_names=True)
```

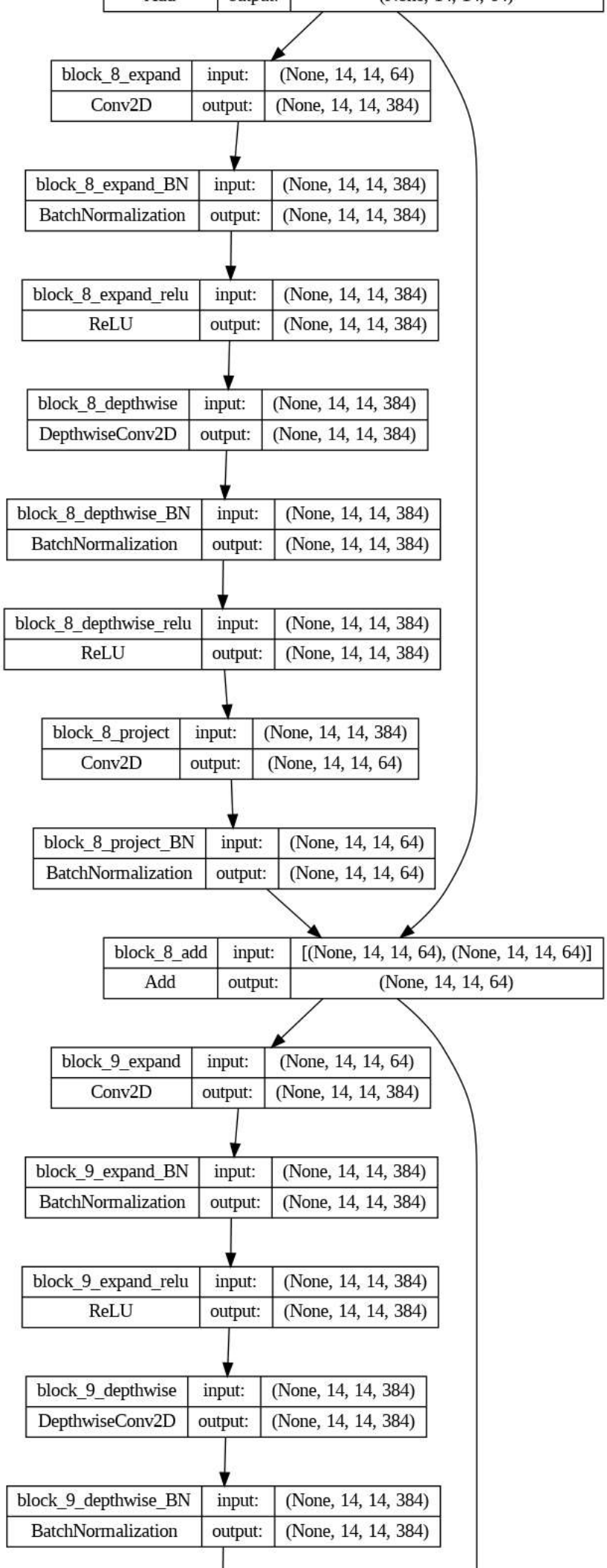

input_12 | input: | [(None, 224, 224, 3)]
InputLayer | output: | [(None, 224, 224, 3)]
Conv1 | input: | (None, 224, 224, 3)
Conv2D | output: | (None, 112, 112, 32)
bn_Conv1 | input: | (None, 112, 112, 32)
BatchNormalization | output: | (None, 112, 112, 32)
Conv1_relu | input: | (None, 112, 112, 32)
ReLU | output: | (None, 112, 112, 32)
expanded_conv_depthwise | input: | (None, 112, 112, 32)
DepthwiseConv2D | output: | (None, 112, 112, 32)
expanded_conv_depthwise_BN | input: | (None, 112, 112, 32)
BatchNormalization | output: | (None, 112, 112, 32)
expanded_conv_depthwise_relu | input: | (None, 112, 112, 32)
ReLU | output: | (None, 112, 112, 32)
expanded_conv_project | input: | (None, 112, 112, 32)
Conv2D | output: | (None, 112, 112, 16)
expanded_conv_project_BN | input: | (None, 112, 112, 16)
BatchNormalization | output: | (None, 112, 112, 16)
block_1_expand | input: | (None, 112, 112, 16)
Conv2D | output: | (None, 112, 112, 96)
block_1_expand_BN | input: | (None, 112, 112, 96)
BatchNormalization | output: | (None, 112, 112, 96)
block_1_expand_relu | input: | (None, 112, 112, 96)
ReLU | output: | (None, 112, 112, 96)
block_1_pad | input: | (None, 112, 112, 96)
ZeroPadding2D | output: | (None, 113, 113, 96)
block_1_depthwise | input: | (None, 113, 113, 96)
DepthwiseConv2D | output: | (None, 56, 56, 96)
block_1_depthwise_BN | input: | (None, 56, 56, 96)

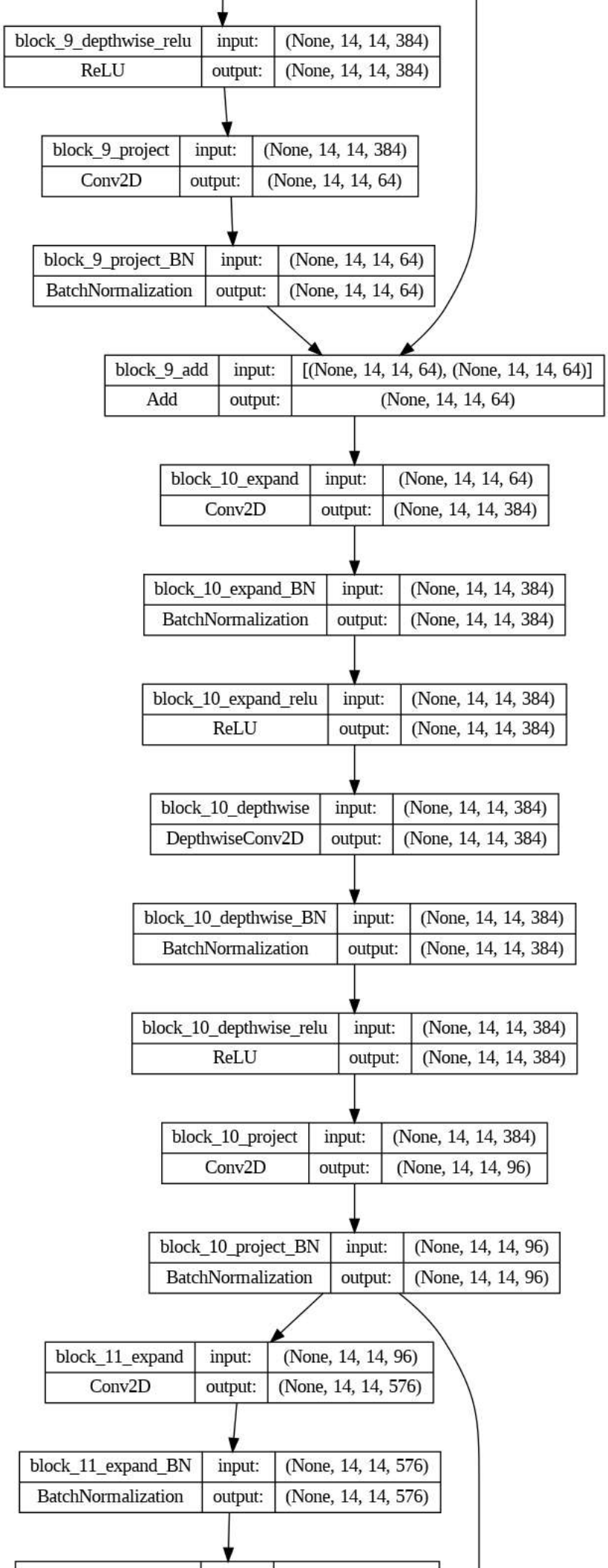

BatchNormalization output: (None, 56, 56, 96)
block_1_depthwise_relu input: (None, 56, 56, 96)
ReLU output: (None, 56, 56, 96)
block_1_project input: (None, 56, 56, 96)
Conv2D output: (None, 56, 56, 24)
block_1_project_BN input: (None, 56, 56, 24)
BatchNormalization output: (None, 56, 56, 24)
block_2_expand input: (None, 56, 56, 24)
Conv2D output: (None, 56, 56, 144)
block_2_expand_BN input: (None, 56, 56, 144)
BatchNormalization output: (None, 56, 56, 144)
block_2_expand_relu input: (None, 56, 56, 144)
ReLU output: (None, 56, 56, 144)
block_2_depthwise input: (None, 56, 56, 144)
DepthwiseConv2D output: (None, 56, 56, 144)
block_2_depthwise_BN input: (None, 56, 56, 144)
BatchNormalization output: (None, 56, 56, 144)
block_2_depthwise_relu input: (None, 56, 56, 144)
ReLU output: (None, 56, 56, 144)
block_2_project input: (None, 56, 56, 144)
Conv2D output: (None, 56, 56, 24)
block_2_project_BN input: (None, 56, 56, 24)
BatchNormalization output: (None, 56, 56, 24)
block_2_add input: [(None, 56, 56, 24), (None, 56, 56, 24)]
Add output: (None, 56, 56, 24)
block_3_expand input: (None, 56, 56, 24)
Conv2D output: (None, 56, 56, 144)
block_3_expand_BN input: (None, 56, 56, 144)
BatchNormalization output: (None, 56, 56, 144)

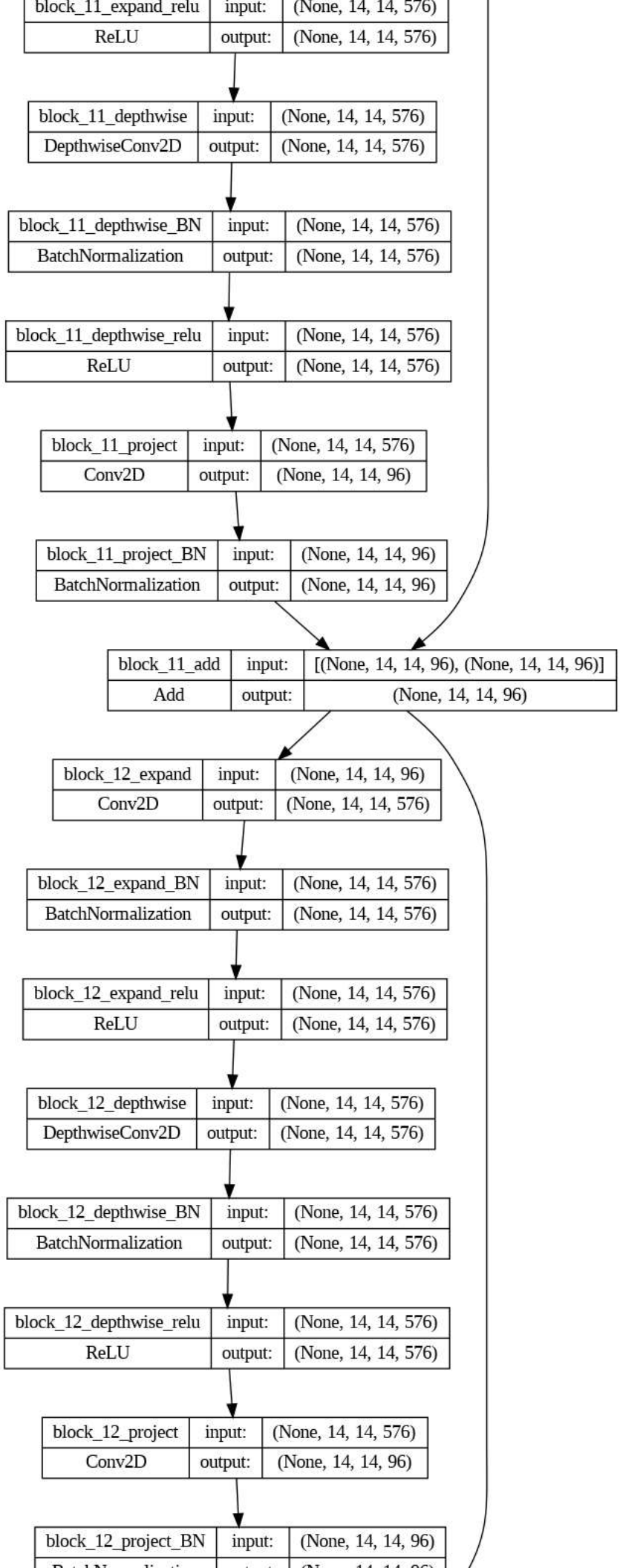

block_3_expand_relu
ReLU
input:
output:
(None, 56, 56, 144)
(None, 56, 56, 144)
block_3_pad
ZeroPadding2D
input:
output:
(None, 56, 56, 144)
(None, 57, 57, 144)
block_3_depthwise
DepthwiseConv2D
input:
output:
(None, 57, 57, 144)
(None, 28, 28, 144)
block_3_depthwise_BN
BatchNormalization
input:
output:
(None, 28, 28, 144)
(None, 28, 28, 144)
block_3_depthwise_relu
ReLU
input:
output:
(None, 28, 28, 144)
(None, 28, 28, 144)
block_3_project
Conv2D
input:
output:
(None, 28, 28, 144)
(None, 28, 28, 32)
block_3_project_BN
BatchNormalization
input:
output:
(None, 28, 28, 32)
(None, 28, 28, 32)
block_4_expand
Conv2D
input:
output:
(None, 28, 28, 32)
(None, 28, 28, 192)
block_4_expand_BN
BatchNormalization
input:
output:
(None, 28, 28, 192)
(None, 28, 28, 192)
block_4_expand_relu
ReLU
input:
output:
(None, 28, 28, 192)
(None, 28, 28, 192)
block_4_depthwise
DepthwiseConv2D
input:
output:
(None, 28, 28, 192)
(None, 28, 28, 192)
block_4_depthwise_BN
BatchNormalization
input:
output:
(None, 28, 28, 192)
(None, 28, 28, 192)
block_4_depthwise_relu
ReLU
input:
output:
(None, 28, 28, 192)
(None, 28, 28, 192)
block_4_project
Conv2D
input:
output:
(None, 28, 28, 192)
(None, 28, 28, 32)

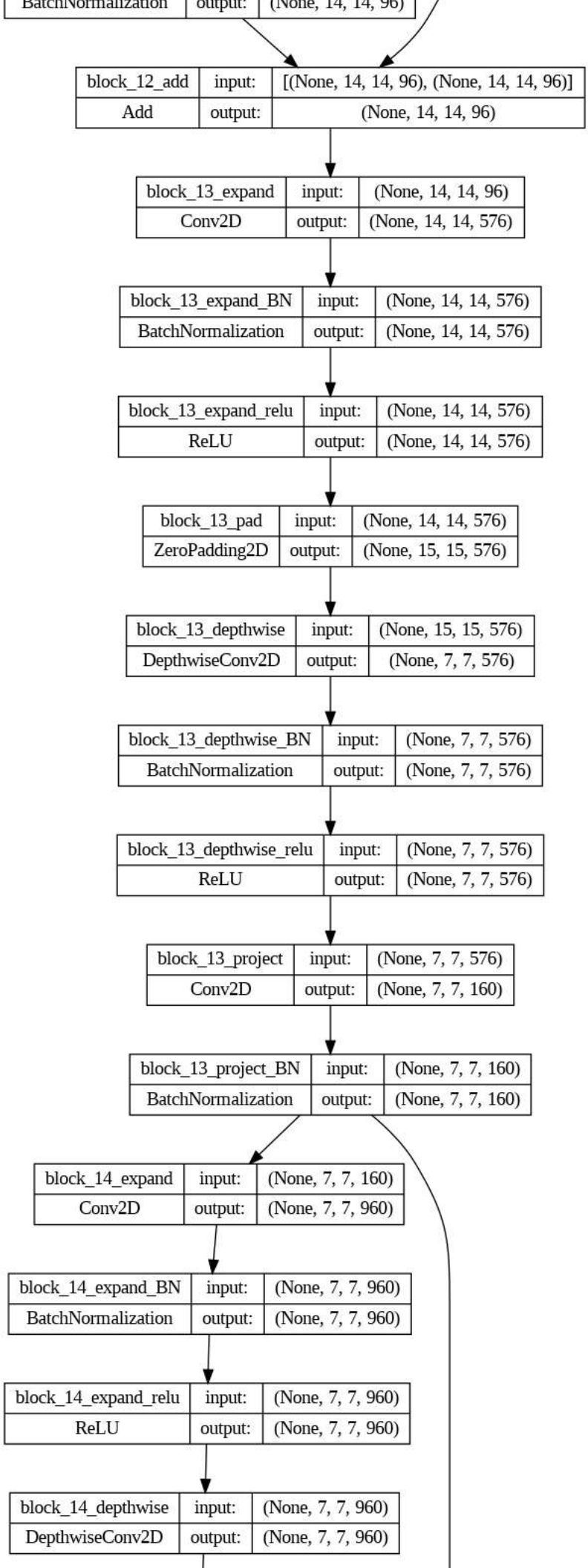

block_4_project_BN
input:
(None, 28, 28, 32)
BatchNormalization
output:
(None, 28, 28, 32)
block_4_add
input:
[(None, 28, 28, 32), (None, 28, 28, 32)]
Add
output:
(None, 28, 28, 32)
block_5_expand
input:
(None, 28, 28, 32)
Conv2D
output:
(None, 28, 28, 192)
block_5_expand_BN
input:
(None, 28, 28, 192)
BatchNormalization
output:
(None, 28, 28, 192)
block_5_expand_relu
input:
(None, 28, 28, 192)
ReLU
output:
(None, 28, 28, 192)
block_5_depthwise
input:
(None, 28, 28, 192)
DepthwiseConv2D
output:
(None, 28, 28, 192)
block_5_depthwise_BN
input:
(None, 28, 28, 192)
BatchNormalization
output:
(None, 28, 28, 192)
block_5_depthwise_relu
input:
(None, 28, 28, 192)
ReLU
output:
(None, 28, 28, 192)
block_5_project
input:
(None, 28, 28, 192)
Conv2D
output:
(None, 28, 28, 32)
block_5_project_BN
input:
(None, 28, 28, 32)
BatchNormalization
output:
(None, 28, 28, 32)
block_5_add
input:
[(None, 28, 28, 32), (None, 28, 28, 32)]
Add
output:
(None, 28, 28, 32)
block_6_expand
input:
(None, 28, 28, 32)
Conv2D
output:
(None, 28, 28, 192)
block_6_expand_BN
input:
(None, 28, 28, 192)
BatchNormalization
output:
(None, 28, 28, 192)
block_6_expand_relu
input:
(None, 28, 28, 192)
ReLU
output:
(None, 28, 28, 192)

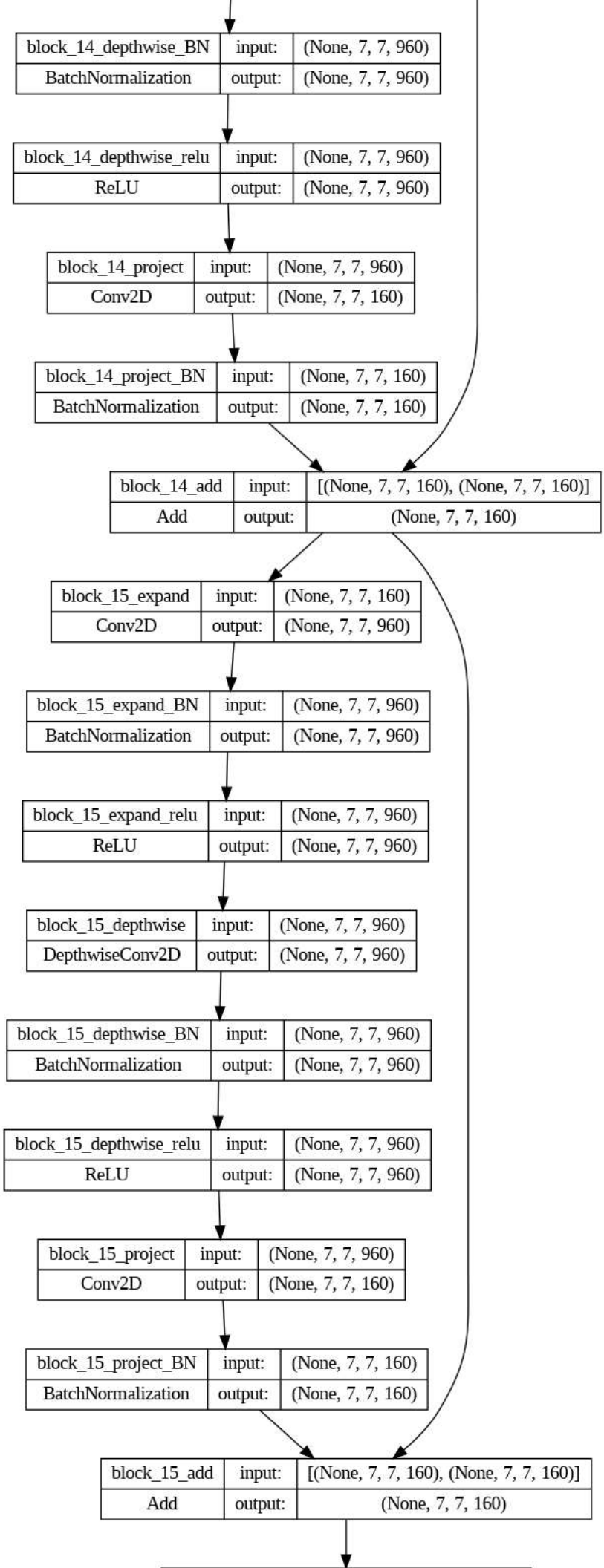

block_6_pad
input:
(None, 28, 28, 192)
ZeroPadding2D
output:
(None, 29, 29, 192)
block_6_depthwise
input:
(None, 29, 29, 192)
DepthwiseConv2D
output:
(None, 14, 14, 192)
block_6_depthwise_BN
input:
(None, 14, 14, 192)
BatchNormalization
output:
(None, 14, 14, 192)
block_6_depthwise_relu
input:
(None, 14, 14, 192)
ReLU
output:
(None, 14, 14, 192)
block_6_project
input:
(None, 14, 14, 192)
Conv2D
output:
(None, 14, 14, 64)
block_6_project_BN
input:
(None, 14, 14, 64)
BatchNormalization
output:
(None, 14, 14, 64)
block_7_expand
input:
(None, 14, 14, 64)
Conv2D
output:
(None, 14, 14, 384)
block_7_expand_BN
input:
(None, 14, 14, 384)
BatchNormalization
output:
(None, 14, 14, 384)
block_7_expand_relu
input:
(None, 14, 14, 384)
ReLU
output:
(None, 14, 14, 384)
block_7_depthwise
input:
(None, 14, 14, 384)
DepthwiseConv2D
output:
(None, 14, 14, 384)
block_7_depthwise_BN
input:
(None, 14, 14, 384)
BatchNormalization
output:
(None, 14, 14, 384)
block_7_depthwise_relu
input:
(None, 14, 14, 384)
ReLU
output:
(None, 14, 14, 384)
block_7_project
input:
(None, 14, 14, 384)
Conv2D
output:
(None, 14, 14, 64)
block_7_project_BN
input:
(None, 14, 14, 64)
BatchNormalization
output:
(None, 14, 14, 64)
block_7_add
input:
[(None, 14, 14, 64), (None, 14, 14, 64)]
Add
output:
(None, 14, 14, 64)

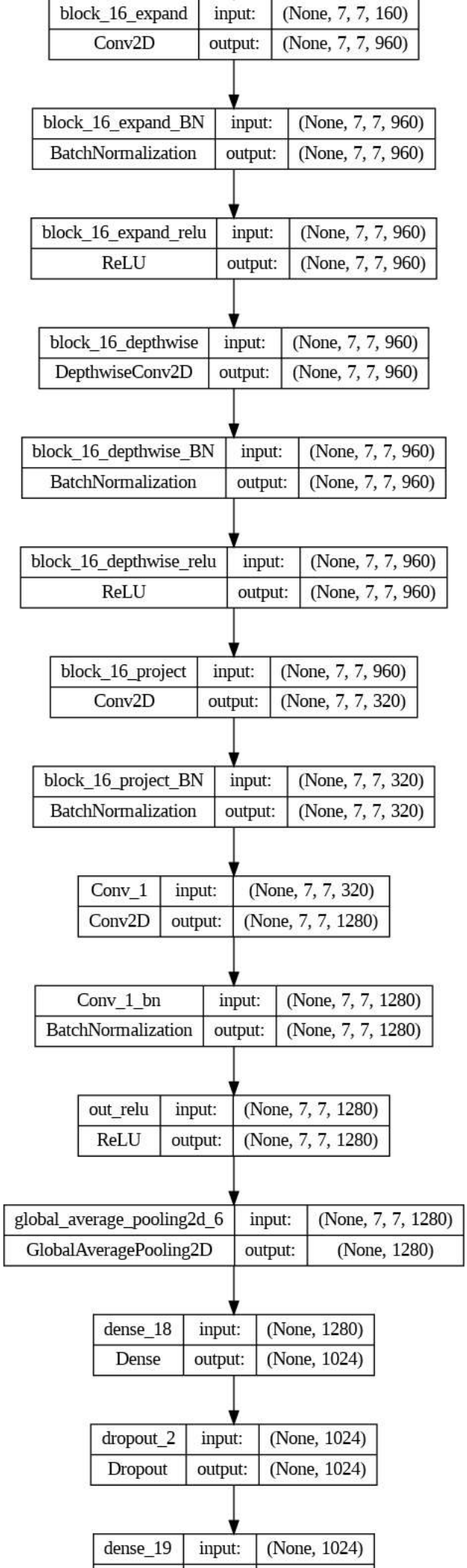

block_8_expand
Conv2D
input:
output:
(None, 14, 14, 64)
(None, 14, 14, 384)
block_8_expand_BN
BatchNormalization
(None, 14, 14, 384)
block_8_expand_relu
ReLU
block_8_depthwise
DepthwiseConv2D
block_8_depthwise_BN
block_8_depthwise_relu
block_8_project
(None, 14, 14, 64)
block_8_project_BN
block_8_add
Add
[(None, 14, 14, 64), (None, 14, 14, 64)]
block_9_expand
block_9_expand_BN
block_9_expand_relu
block_9_depthwise
block_9_depthwise_BN


block_9_depthwise_relu
ReLU
input:
output:
(None, 14, 14, 384)
(None, 14, 14, 384)
block_9_project
Conv2D
input:
output:
(None, 14, 14, 384)
(None, 14, 14, 64)
block_9_project_BN
BatchNormalization
input:
output:
(None, 14, 14, 64)
(None, 14, 14, 64)
block_9_add
Add
input:
output:
[(None, 14, 14, 64), (None, 14, 14, 64)]
(None, 14, 14, 64)
block_10_expand
Conv2D
input:
output:
(None, 14, 14, 64)
(None, 14, 14, 384)
block_10_expand_BN
BatchNormalization
input:
output:
(None, 14, 14, 384)
(None, 14, 14, 384)
block_10_expand_relu
ReLU
input:
output:
(None, 14, 14, 384)
(None, 14, 14, 384)
block_10_depthwise
DepthwiseConv2D
input:
output:
(None, 14, 14, 384)
(None, 14, 14, 384)
block_10_depthwise_BN
BatchNormalization
input:
output:
(None, 14, 14, 384)
(None, 14, 14, 384)
block_10_depthwise_relu
ReLU
input:
output:
(None, 14, 14, 384)
(None, 14, 14, 384)
block_10_project
Conv2D
input:
output:
(None, 14, 14, 384)
(None, 14, 14, 96)
block_10_project_BN
BatchNormalization
input:
output:
(None, 14, 14, 96)
(None, 14, 14, 96)
block_11_expand
Conv2D
input:
output:
(None, 14, 14, 96)
(None, 14, 14, 576)
block_11_expand_BN
BatchNormalization
input:
output:
(None, 14, 14, 576)
(None, 14, 14, 576)


block_11_expand_relu
ReLU
input:
output:
(None, 14, 14, 576)
(None, 14, 14, 576)
block_11_depthwise
DepthwiseConv2D
input:
output:
(None, 14, 14, 576)
(None, 14, 14, 576)
block_11_depthwise_BN
BatchNormalization
input:
output:
(None, 14, 14, 576)
(None, 14, 14, 576)
block_11_depthwise_relu
ReLU
input:
output:
(None, 14, 14, 576)
(None, 14, 14, 576)
block_11_project
Conv2D
input:
output:
(None, 14, 14, 576)
(None, 14, 14, 96)
block_11_project_BN
BatchNormalization
input:
output:
(None, 14, 14, 96)
(None, 14, 14, 96)
block_11_add
Add
input:
output:
[(None, 14, 14, 96), (None, 14, 14, 96)]
(None, 14, 14, 96)
block_12_expand
Conv2D
input:
output:
(None, 14, 14, 96)
(None, 14, 14, 576)
block_12_expand_BN
BatchNormalization
input:
output:
(None, 14, 14, 576)
(None, 14, 14, 576)
block_12_expand_relu
ReLU
input:
output:
(None, 14, 14, 576)
(None, 14, 14, 576)
block_12_depthwise
DepthwiseConv2D
input:
output:
(None, 14, 14, 576)
(None, 14, 14, 576)
block_12_depthwise_BN
BatchNormalization
input:
output:
(None, 14, 14, 576)
(None, 14, 14, 576)
block_12_depthwise_relu
ReLU
input:
output:
(None, 14, 14, 576)
(None, 14, 14, 576)
block_12_project
Conv2D
input:
output:
(None, 14, 14, 576)
(None, 14, 14, 96)
block_12_project_BN
input:
(None, 14, 14, 96)


BatchNormalization
output:
(None, 14, 14, 96)
block_12_add
input:
[(None, 14, 14, 96), (None, 14, 14, 96)]
Add
output:
(None, 14, 14, 96)
block_13_expand
input:
(None, 14, 14, 96)
Conv2D
output:
(None, 14, 14, 576)
block_13_expand_BN
input:
(None, 14, 14, 576)
BatchNormalization
output:
(None, 14, 14, 576)
block_13_expand_relu
input:
(None, 14, 14, 576)
ReLU
output:
(None, 14, 14, 576)
block_13_pad
input:
(None, 14, 14, 576)
ZeroPadding2D
output:
(None, 15, 15, 576)
block_13_depthwise
input:
(None, 15, 15, 576)
DepthwiseConv2D
output:
(None, 7, 7, 576)
block_13_depthwise_BN
input:
(None, 7, 7, 576)
BatchNormalization
output:
(None, 7, 7, 576)
block_13_depthwise_relu
input:
(None, 7, 7, 576)
ReLU
output:
(None, 7, 7, 576)
block_13_project
input:
(None, 7, 7, 576)
Conv2D
output:
(None, 7, 7, 160)
block_13_project_BN
input:
(None, 7, 7, 160)
BatchNormalization
output:
(None, 7, 7, 160)
block_14_expand
input:
(None, 7, 7, 160)
Conv2D
output:
(None, 7, 7, 960)
block_14_expand_BN
input:
(None, 7, 7, 960)
BatchNormalization
output:
(None, 7, 7, 960)
block_14_expand_relu
input:
(None, 7, 7, 960)
ReLU
output:
(None, 7, 7, 960)
block_14_depthwise
input:
(None, 7, 7, 960)
DepthwiseConv2D
output:
(None, 7, 7, 960)


block_14_depthwise_BN
BatchNormalization
input:
output:
(None, 7, 7, 960)
(None, 7, 7, 960)
block_14_depthwise_relu
ReLU
input:
output:
(None, 7, 7, 960)
(None, 7, 7, 960)
block_14_project
Conv2D
input:
output:
(None, 7, 7, 960)
(None, 7, 7, 160)
block_14_project_BN
BatchNormalization
input:
output:
(None, 7, 7, 160)
(None, 7, 7, 160)
block_14_add
Add
input:
output:
[(None, 7, 7, 160), (None, 7, 7, 160)]
(None, 7, 7, 160)
block_15_expand
Conv2D
input:
output:
(None, 7, 7, 160)
(None, 7, 7, 960)
block_15_expand_BN
BatchNormalization
input:
output:
(None, 7, 7, 960)
(None, 7, 7, 960)
block_15_expand_relu
ReLU
input:
output:
(None, 7, 7, 960)
(None, 7, 7, 960)
block_15_depthwise
DepthwiseConv2D
input:
output:
(None, 7, 7, 960)
(None, 7, 7, 960)
block_15_depthwise_BN
BatchNormalization
input:
output:
(None, 7, 7, 960)
(None, 7, 7, 960)
block_15_depthwise_relu
ReLU
input:
output:
(None, 7, 7, 960)
(None, 7, 7, 960)
block_15_project
Conv2D
input:
output:
(None, 7, 7, 960)
(None, 7, 7, 160)
block_15_project_BN
BatchNormalization
input:
output:
(None, 7, 7, 160)
(None, 7, 7, 160)
block_15_add
Add
input:
output:
[(None, 7, 7, 160), (None, 7, 7, 160)]
(None, 7, 7, 160)


block_16_expand
input:
(None, 7, 7, 160)
Conv2D
output:
(None, 7, 7, 960)
block_16_expand_BN
input:
(None, 7, 7, 960)
BatchNormalization
output:
(None, 7, 7, 960)
block_16_expand_relu
input:
(None, 7, 7, 960)
ReLU
output:
(None, 7, 7, 960)
block_16_depthwise
input:
(None, 7, 7, 960)
DepthwiseConv2D
output:
(None, 7, 7, 960)
block_16_depthwise_BN
input:
(None, 7, 7, 960)
BatchNormalization
output:
(None, 7, 7, 960)
block_16_depthwise_relu
input:
(None, 7, 7, 960)
ReLU
output:
(None, 7, 7, 960)
block_16_project
input:
(None, 7, 7, 960)
Conv2D
output:
(None, 7, 7, 320)
block_16_project_BN
input:
(None, 7, 7, 320)
BatchNormalization
output:
(None, 7, 7, 320)
Conv_1
input:
(None, 7, 7, 320)
Conv2D
output:
(None, 7, 7, 1280)
Conv_1_bn
input:
(None, 7, 7, 1280)
BatchNormalization
output:
(None, 7, 7, 1280)
out_relu
input:
(None, 7, 7, 1280)
ReLU
output:
(None, 7, 7, 1280)
global_average_pooling2d_6
input:
(None, 7, 7, 1280)
GlobalAveragePooling2D
output:
(None, 1280)
dense_18
input:
(None, 1280)
Dense
output:
(None, 1024)
dropout_2
input:
(None, 1024)
Dropout
output:
(None, 1024)
dense_19
input:
(None, 1024)

Dense	output:	(None, 1)

To gain a better understanding of the model's architecture, we use the `plot_model` function from TensorFlow to generate a visual representation. This plot includes the shapes of the layers and their names.

Interpret

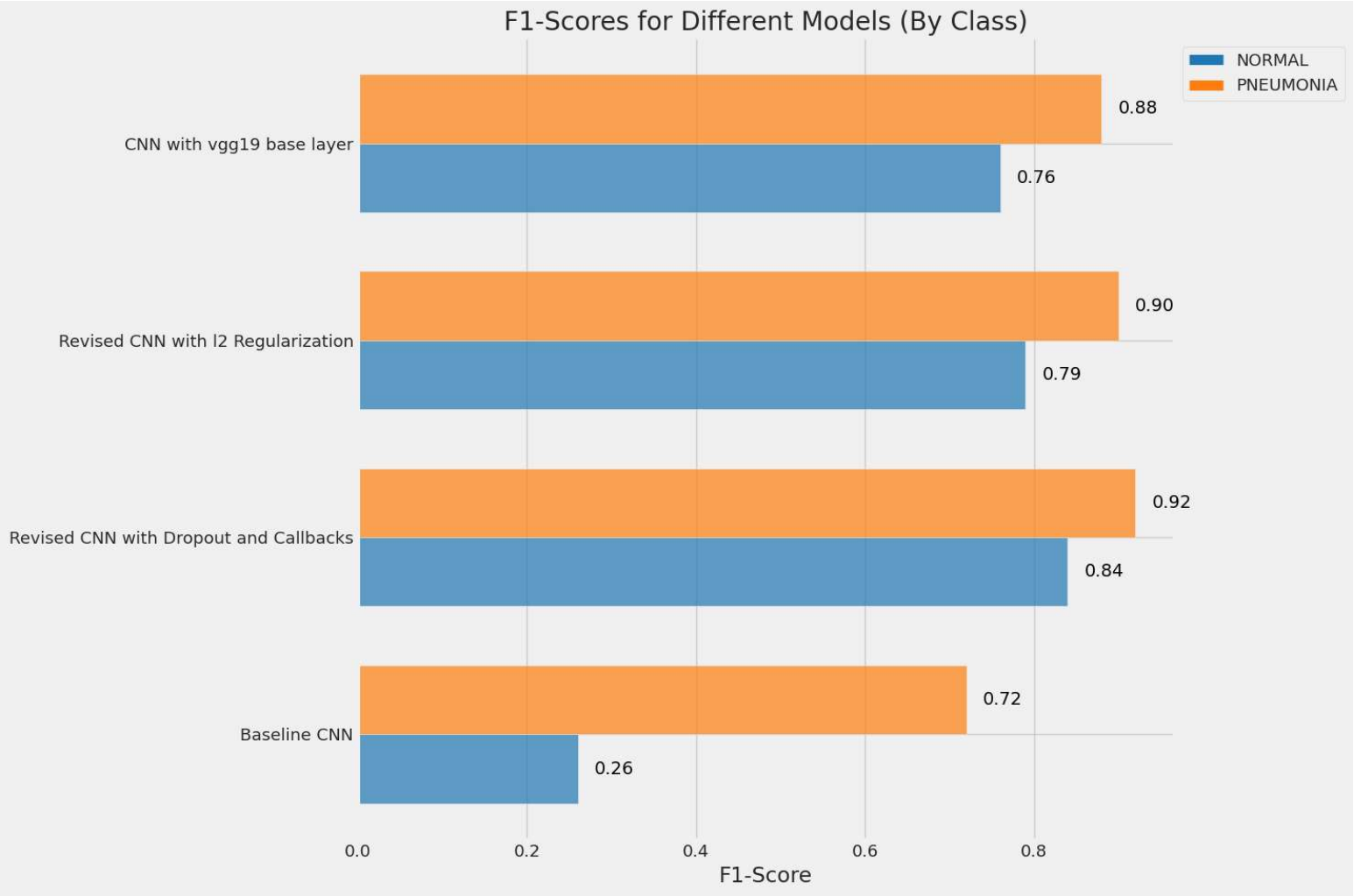
In this section, we visualize the performance of various machine learning models by plotting their F1 scores. The F1 score is a measure of a model's accuracy that considers both precision and recall. A higher F1 score indicates better overall performance.

The models include a Baseline CNN, a Revised CNN with Dropout and Callbacks, a Revised CNN with l2 Regularization, and a CNN with a vgg19 base layer. Each model's performance is assessed based on their F1-scores, which balance precision and recall.

The grouped bar chart below represents the F1-scores for both classes (NORMAL and PNEUMONIA) across the different models.Among these models, the **Revised CNN with Dropout and Callbacks** demonstrates the highest F1-scores for both classes *(0.80 for NORMAL images and 0.92 for PNEUMONIA images)*, indicating superior performance in this context.

```
# Use Style fivethirtyeight for consistency
plt.style.use('fivethirtyeight')


# Define the models and their corresponding F1-scores
models = [
    "Baseline CNN",
    "Revised CNN with Dropout and Callbacks",
    "Revised CNN with l2 Regularization",
    "CNN with vgg19 base layer"
]

# Define F1-Scores for NORMAL and PNEUMONIA classes for each model
f1_scores = [
    {"NORMAL": 0.26, "PNEUMONIA": 0.72},  # F1-Scores for Baseline CNN
    {"NORMAL": 0.84, "PNEUMONIA": 0.92},  # F1-Scores for Revised CNN
    {"NORMAL": 0.79, "PNEUMONIA": 0.90},  # F1-Scores for Revised CNN with l2 Regularization
    {"NORMAL": 0.76, "PNEUMONIA": 0.88}   # F1-Scores for CNN with vgg19 base layer
]

# Set the fivethirtyeight style
plt.style.use('fivethirtyeight')

# Create a figure and axis object
fig, ax = plt.subplots(figsize=(15, 10))

# Define the positions for each group
x = np.arange(len(models))
bar_width = 0.35  # Width of the bars

# Define colors for NORMAL and PNEUMONIA classes
colors = {'NORMAL': '#1f77b4', 'PNEUMONIA': '#ff7f0e'}

# Transpose the data for plotting
f1_scores_transposed = {'NORMAL': [], 'PNEUMONIA': []}
for model_scores in f1_scores:
    f1_scores_transposed['NORMAL'].append(model_scores['NORMAL'])
    f1_scores_transposed['PNEUMONIA'].append(model_scores['PNEUMONIA'])

# Generate the grouped bar chart
for i, (model, scores) in enumerate(zip(models, f1_scores_transposed['NORMAL'])):
    ax.barh(x[i] - bar_width/2, scores, bar_width, color=colors['NORMAL'], alpha=0.7)
    ax.text(scores + 0.02, x[i] - bar_width/2, f'{scores:.2f}', va='center', ha='left', color='black')

for i, (model, scores) in enumerate(zip(models, f1_scores_transposed['PNEUMONIA'])):
    ax.barh(x[i] + bar_width/2, scores, bar_width, color=colors['PNEUMONIA'], alpha=0.7)
    ax.text(scores + 0.02, x[i] + bar_width/2, f'{scores:.2f}', va='center', ha='left', color='black')

# Add labels and title
ax.set_xlabel('F1-Score')
ax.set_title('F1-Scores for Different Models (By Class)')
ax.set_yticks(x)
ax.set_yticklabels(models)

# Add a custom legend with correct labels and colors
ax.legend(handles=[
    plt.Line2D([0], [0], color=colors['NORMAL'], lw=8),
    plt.Line2D([0], [0], color=colors['PNEUMONIA'], lw=8)
], labels=['NORMAL', 'PNEUMONIA'], loc='upper left', bbox_to_anchor=(1, 1))
```

```
# Show the plot
plt.tight_layout()
plt.show()
```



This plot provides a quick comparison of model performance, helping us identify which models perform best in terms of F1 scores.

```
# Import necessary libraries
from matplotlib import pyplot  # Importing pyplot from matplotlib
import seaborn as sns  # Importing seaborn for enhanced visualization

# Set the figure size using seaborn
sns.set(rc={'figure.figsize':(15,12)})

# Create a horizontal bar plot
sns.barplot(x=model_f1_score.values, y=model_f1_score.index, orient='h')

# Set labels and title
plt.xlabel('F1-Score PNEUMONIA')  # X-axis label
plt.xticks(rotation='vertical', fontsize=14)  # Rotate x-axis labels for better readability
plt.title('F1-Scores of All Models tested')  # Set the title of the plot
```

This code performs image classification using a revised CNN model (`revised_model_4`) trained on a dataset. It then employs LIME (Local Interpretable Model-agnostic Explanations) to provide insights into the model's predictions.

Functions for Explanation and Visualization

The code defines two functions:

1. `explain_image`: This function explains an image's classification using LIME. It displays the true class, predicted class, and features contributing to the classification.
2. `visualize_features`: Similar to `explain_image`, this function visualizes features contributing to the classification, but it allows for specifying the true class separately.

Model Architecture and Training

The model (`revised_model_4`) is a CNN composed of convolutional, max pooling, dropout, and dense layers. It is compiled with the RMSprop optimizer and binary cross-entropy loss. Early stopping is used to prevent overfitting.

The model is trained for 500 epochs and evaluated on the test data.

LIME Explanation

An instance of the LIME explainer is initialized. Two random images are selected from the training data, and their classifications and contributing features are visualized using LIME.

Please ensure the necessary libraries, dataset, and model are available to run this code.

```
from lime.lime_image import LimeImageExplainer
import matplotlib.pyplot as plt
import numpy as np

# Create an explainer
explainer = LimeImageExplainer()

# Define a function to predict using your model
predict_fn = lambda x: cnn_model_with_callbacks.predict(x)

# Choose an image from your validation set to explain (you can change the index as needed)
image_index = 0
sample_image = train_data[0][0][image_index]  # Assuming 0th batch, 0th element, and image_index

# Explain the prediction
explanation = explainer.explain_instance(sample_image, predict_fn, top_labels=1, hide_color=0, num_samples=1000)

# Get the model's prediction for the image
predicted_class = np.argmax(from lime.lime_image import LimeImageExplainer
import matplotlib.pyplot as plt
import numpy as np
import random

# Create an explainer
explainer = LimeImageExplainer()

# Define a function to predict using your model
predict_fn = lambda x: vgg19_model.predict(x)

# Get the total number of images in your test set
total_images = len(test_data.classes)

# Define the batch size
batch_size = 32

# Choose two random image indices from your test set
random_indices = random.sample(range(total_images), 2)

# Get the batch and index within the batch for each random index
for image_index in random_indices:
    batch = image_index // batch_size
    index_within_batch = image_index % batch_size

    # Get the image using the batch and index
    sample_image = test_data[batch][0][index_within_batch]

    # Explain the prediction
    explanation = explainer.explain_instance(sample_image, predict_fn, top_labels=1, hide_color=0, num_samples=1000)

    # Get the model's prediction for the image
    predicted_class = np.argmax(cnn_model_with_callbacks.predict(sample_image.reshape(1, *input_shape)), axis=1)[0]

    # Map numerical class to labels
    class_labels = {0: "Normal", 1: "PNEUMONIA"}
    predicted_class_label = class_labels[predicted_class]
    true_label = class_labels[test_data.classes[image_index]]

    # Show the explanation with labels
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4))
```

```
    # Show the original image with labels
    ax1.imshow(sample_image)
    ax1.set_title(f"True Label: {true_label}\nPredicted Label: {predicted_class_label}")

    # Show the explanation
    temp, mask = explanation.get_image_and_mask(predicted_class, positive_only=False, num_features=5, hide_rest=False)
    ax2.imshow(mark_boundaries(temp / 2 + 0.5, mask))
    ax2.set_title("LIME Explanation for Best Model")

    plt.show().predict(sample_image.reshape(1, *input_shape)), axis=1)[0]

# Map numerical class to labels
class_labels = {0: "Normal", 1: "PNEUMONIA"}
predicted_class_label = class_labels[predicted_class]
true_label = class_labels[train_data.classes[image_index]]

# Show the explanation with labels
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4))

# Show the original image with labels
ax1.imshow(sample_image)
ax1.set_title(f"True Label: {true_label}\nPredicted Label: {predicted_class_label}")

# Show the explanation
temp, mask = explanation.get_image_and_mask(predicted_class, positive_only=False, num_features=5, hide_rest=False)
ax2.imshow(mark_boundaries(temp / 2 + 0.5, mask))
ax2.set_title("LIME Explanation for Best Model")

plt.show()
```

```
100%                                                1000/1000 [00:13<00:00, 74.16it/s]
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 26ms/step
```

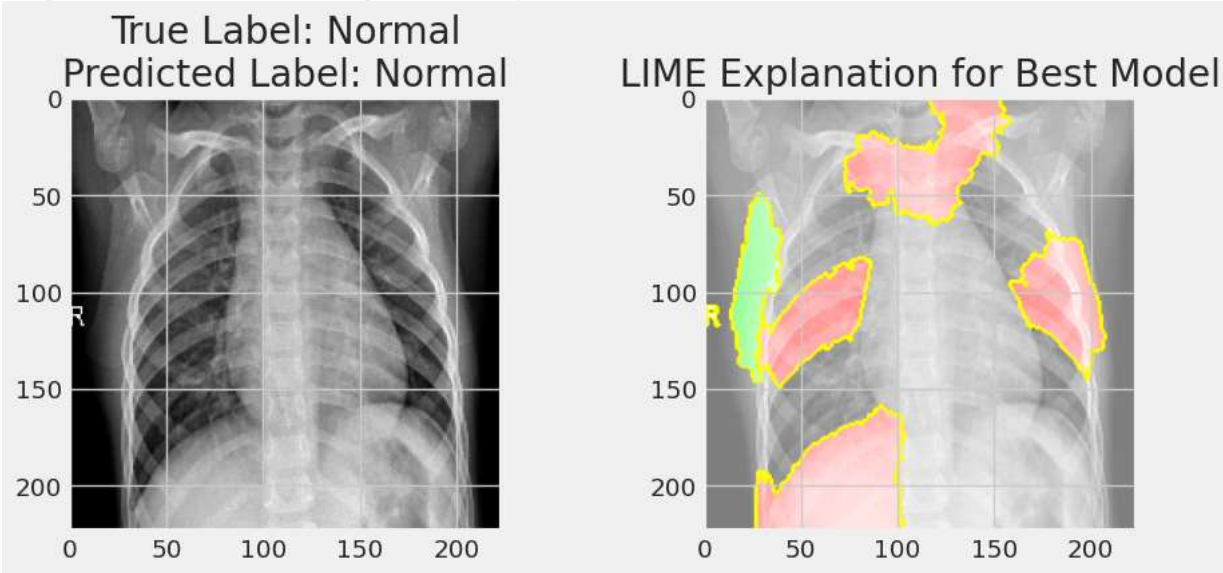
Lime identified a random image from the test dataset that we correctly predicted the true label and highlighted the features that the model determined were important. We could use the output of LIME to create image criteria for end users of our product. For example, a radiologist coul be asked to crop images that add ambiguity to the model.

```
from lime.lime_image import LimeImageExplainer
import matplotlib.pyplot as plt
import numpy as np
import random

# Create an explainer
explainer = LimeImageExplainer(verbose=False)

# Define a function to predict using your model
predict_fn = lambda x: vgg19_model.predict(x)

# Get the total number of images in your test set
```

```
total_images = len(test_data.classes)

# Define the batch size
batch_size = 32

# Choose two random image indices from your test set
random_indices = random.sample(range(total_images), 2)

# Get the batch and index within the batch for each random index
for image_index in random_indices:
    batch = image_index // batch_size
    index_within_batch = image_index % batch_size

    # Get the image using the batch and index
    sample_image = test_data[batch][0][index_within_batch]

    # Explain the prediction
    explanation = explainer.explain_instance(sample_image, predict_fn, top_labels=1, hide_color=0, num_samples=1000)

    # Get the model's prediction for the image
    predicted_class = np.argmax(vgg19_model.predict(sample_image.reshape(1, *input_shape)), axis=1)[0]

    # Map numerical class to labels
    class_labels = {0: "Normal", 1: "PNEUMONIA"}
    predicted_class_label = class_labels[predicted_class]
    true_label = class_labels[test_data.classes[image_index]]

    # Show the explanation with labels
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4))

    # Show the original image with labels
    ax1.imshow(sample_image)
    ax1.set_title(f"True Label: {true_label}\nPredicted Label: {predicted_class_label}")

    # Show the explanation
    temp, mask = explanation.get_image_and_mask(predicted_class, positive_only=False, num_features=5, hide_rest=False);
    ax2.imshow(mark_boundaries(temp / 2 + 0.5, mask))
    ax2.set_title("LIME Explanation for Best Model")

    plt.show()
```

```
100%                                                    1000/1000 [00:13<00:00, 74.81it/s]
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 24ms/step
```

```
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 37ms/step
```



100% 1000/1000 [00:13<00:00, 74.15it/s]

```
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 24ms/step
```

```
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
```

Recommendations

Optimal Scanner Placements

To maximize the effectiveness of our Pneumonia detection system, it is recommended to strategically deploy X-ray scanners and computer systems with our model in the following locations to run a pilot to collect more expert verifiable data:

- **Pediatric Wards and Clinics**: Ensure accessibility to children in healthcare facilities where pneumonia cases are most frequently diagnosed and treated.
- **High-Risk Areas and Communities**: Identify regions with elevated incidences of childhood pneumonia and establish scanning facilities in proximity to these communities.

Expert-Evaluated Data for Model Enhancement

Utilizing images that have been expert-evaluated is crucial for improving the accuracy and performance of our model. This can be achieved through the following steps:

- **Collaboration with Radiology Experts**: Establish partnerships with radiologists and other medical professionals to evaluate X-ray images. Their expertise will contribute to a high-quality annotated dataset.
- **Continuous Feedback Loop**: Implement a feedback mechanism to incorporate expert evaluations into the training pipeline, ensuring that the model learns from expert insights.

Expedited Testing Process

Efficient testing is essential for timely diagnosis and treatment. To speed up the testing process, consider the following strategies:

- **Automation of Analysis**: Implement automated tools for initial image analysis, allowing experts to focus on more complex cases and reducing the overall processing time.
- **Parallel Processing**: Leverage the power of parallel computing to process multiple scans simultaneously, further accelerating the testing pipeline.

SWOT Analysis for Rapid Deployment

Before deploying our system in countries with high childhood pneumonia rates, conduct a SWOT (Strengths, Weaknesses, Opportunities, Threats) analysis to make informed decisions:

- **Strengths**: Evaluate the advantages our system brings, such as accurate and rapid diagnosis, which can significantly impact patient outcomes.
- **Weaknesses**: Identify potential shortcomings, like the need for specialized equipment or training, and develop mitigation strategies.
- **Opportunities**: Assess the potential for collaboration with local healthcare providers and organizations, as well as opportunities for community outreach and education.
- **Threats**: Consider external factors that may hinder successful deployment, such as regulatory challenges or resource constraints.

This comprehensive SWOT analysis will provide a clear roadmap for the rapid and effective deployment of our Pneumonia detection system in regions with high childhood pneumonia rates.

Future Work