Image Classification using Convolutional Neural Networks (CNN) - X-ray Images

Author: Group 6 - Angela Nyaga, Sophy Owuor and Rowlandson Kariuki

Overview

The aim of this project is to use deep learning models to to classify XRAY images as either belonging to a healthy patient or to a patient suffering from Pneumonia.The approach used involves combining advanced image processing and machine learning to develop a robust classification system.

Business Understanding

In the realm of medical diagnostics, the detection of pneumonia through chest X-ray images poses a significant challenge, especially considering the limited availability of radiologists. The manual analysis of X-rays is time-consuming and relies heavily on the expertise of specialized professionals. This bottleneck can lead to delays in diagnosis and treatment, impacting patient outcomes. Consequently, there is a critical need for an automated and efficient solution that can accurately determine the presence or absence of pneumonia in X-ray images, mitigating the dependence on scarce human resources.

Objectives:

1. Develop a Robust CNN Model:

Design and train a convolutional neural network (CNN) tailored for the nuances of chest X-ray images that can optimize the model's ability to discern pneumonia-related patterns.

2. Accurate Pneumonia Detection:

Train the model to accurately identify indicators of pneumonia in X-ray images.

3. Limited Radiologist Dependency:

Alleviate the strain on radiologist resources by creating a model capable of interpreting and analyzing X-rays without the need for specialized human expertise.

4. Timely Diagnosis and Treatment:

Accelerating the pneumonia diagnosis process through automated X-ray analysis can lead to quicker initiation of treatment, ultimately improving patient outcomes. Resource Optimization:

Setup

```
# run on terminal or run this cell if using colab
!pip install split-folders
!pip install scikit-learn
```

```
Collecting split-folders
  Downloading split_folders-0.5.1-py3-none-any.whl (8.4 kB)
Installing collected packages: split-folders
Successfully installed split-folders-0.5.1
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-pa
```

```
# importing necessary libraries
# Standard Libraries
import os

# Data Manipulation Libraries
import numpy as np
from PIL import Image

# Data Splitting and Preprocessing
import splitfolders
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator, img_to_array

# Data Visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Machine Learning Libraries
import tensorflow as tf
from tensorflow.keras import optimizers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from sklearn import metrics
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import RandomOverSampler
from tensorflow.keras.applications import VGG16
from tensorflow.keras import models
from tensorflow.keras import layers
from keras.applications import VGG19
from keras.layers import concatenate
```

Examining our directory structure and establishing the necessary file paths for upcoming tasks

The original datset (CellData) upon download is divided into two folders: "train" and "test". Both the train and test folders contain two folders: one for "normal" xray images and the other for xray images showing patients with pneumonia.Because of the size of the data,computation is very heavy. Therefore, the following code to access the data has two alternatives. The first is if you have the data locally and your machine is capable of running this notebook without memory issues. The other is through google drive and google colab, which offers a GPU to aid in computation.

```
# run this on jupyter
# Define the base directory path for chest X-ray images
# images_home = '/CellData/chest_xray/'

# Define directory paths for training, testing, and validation sets
# train_files = images_home + "train/"
# test_files = images_home + "test/"



# run this if on colab
from google.colab import drive

# # Mount Google Drive
drive.mount('/content/drive')

# Specify the path to the directory of the data
data_directory = '/content/drive/My Drive/CellData'

# unzip files
!unzip drive/My\ Drive/CellData/CellData.zip

# Define the base directory path for chest X-ray images
images_home = "/content/CellData/chest_xray/"

# Define directory paths for training, testing, and validation sets
train_files = images_home + "train/"
test_files = images_home + "test/"
```

```
Streaming output truncated to the last 5000 lines.
  inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3411116-0002.jpeg
  inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3411116-0003.jpeg
  inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3444356-0001.jpeg
  inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3448549-0001.jpeg
  inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3476904-0001.jpeg
  inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3482198-0001.jpeg
  inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3482198-0002.jpeg
  inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3482198-0003.jpeg
  inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3482198-0004.jpeg
  inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3486729-0001.jpeg
  inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3487615-0001.jpeg
  inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3502771-0002.jpeg
  inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3514363-0001.jpeg
  inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3514363-0002.jpeg
  inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3514363-0003.jpeg
  inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3514363-0004.jpeg
  inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3518933-0001.jpeg
  inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3518986-0001.jpeg
  inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3526928-0002.jpeg
  inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3532468-0001.jpeg
  inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3573766-0001.jpeg
  inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3573766-0003.jpeg
  inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3573766-0004.jpeg
  inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3580922-0001.jpeg
```

```
inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3624836-0001.jpeg
inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3644944-0001.jpeg
inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3644944-0002.jpeg
inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3644944-0003.jpeg
inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3684474-0001.jpeg
inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3685250-0001.jpeg
inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3706164-0001.jpeg
inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3744309-0001.jpeg
inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3744578-0001.jpeg
inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3747940-0001.jpeg
inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3751100-0001.jpeg
inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3758513-0001.jpeg
inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3758513-0002.jpeg
inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3758513-0003.jpeg
inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3758513-0004.jpeg
inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3780378-0001.jpeg
inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3780378-0002.jpeg
inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3780378-0003.jpeg
inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3781678-0001.jpeg
inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3789615-0001.jpeg
inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3789615-0002.jpeg
inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3802540-0001.jpeg
inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3802697-0001.jpeg
inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3803096-0001.jpeg
inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3821719-0001.jpeg
inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3823741-0001.jpeg
inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3848106-0002.jpeg
inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3852346-0002.jpeg
inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3852346-0003.jpeg
inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3853078-0001.jpeg
inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3853078-0002.jpeg
inflating: train_files/val_files/train_resized/NORMAL/NORMAL-3853265-0001.jpeg
inflating: train_files/val_files/train_resized/NORMAL/NORMAL-385855-0001.jpeg
```

Exploratory Analysis (EDA) and Data Preparation

```
# Print the list of filenames in the training directory
print(os.listdir(train_files))
```

```
['PNEUMONIA', 'NORMAL']
```

```
# Define directory paths for normal and pneumonia training images
train_norm = train_files+"NORMAL/"
train_sick = train_files+"PNEUMONIA/"

# Define directory paths for normal and pneumonia training images
test_norm = test_files+"NORMAL/"
test_sick = test_files+"PNEUMONIA/"
```

```
# Print the number of images in the normal and pneumonia training directories
print("Number of normal training images is : ", len(os.listdir(train_norm)))
print("Number of sick training images is : ", len(os.listdir(train_sick)))
print("Number of normal testing images is : ", len(os.listdir(test_norm)))
print("Number of sick testing images is : ", len(os.listdir(test_sick)))
```

```
Number of normal training images is :  1349
Number of sick training images is :  3883
Number of normal testing images is :  234
Number of sick testing images is :  390
```

The current directory structure lacks a set of images that we can use to validate during the modelling process. A validation set helps in tuning hyperparameters and preventing overfitting. It is different from the test set, which is reserved for final model analysis. Because the training set of images contains more images, we will use a subset of it to create a validation set. We will use split folders library because of the simplicity of use.

```
# Run this cell only once
# Taking images from the train folder

# input_folder = train_files
# output_folder = "train_files/val_files"

# # Specify the split ratios
# split_ratios = (0.7, 0.3)

# # Perform the split
# splitfolders.ratio(input_folder, output_folder, seed=42, ratio=split_ratios, group_prefix=
```

The path to our directory has changed with the split. We therefore need to redefine the path to our train dataset and our newly created validation dataset.

```
# specifying path to newly created train and validation set
train_files = "train_files/val_files/train/"
train_norm = train_files +"NORMAL/"
train_sick = train_files +"PNEUMONIA/"

val_files = "train_files/val_files/val/"
val_norm = val_files +"NORMAL/"
val_sick = val_files +"PNEUMONIA/"

# Inspecting how many images are left in the training set and how many are in the val set
print("No of normal train images after split is:", len(os.listdir(train_norm)))
print("No of sick train images after split is:",len(os.listdir(train_sick)))

print("No of normal val images is:", len(os.listdir(val_norm)))
print("No of sick val images is:", len(os.listdir(val_sick)))
```

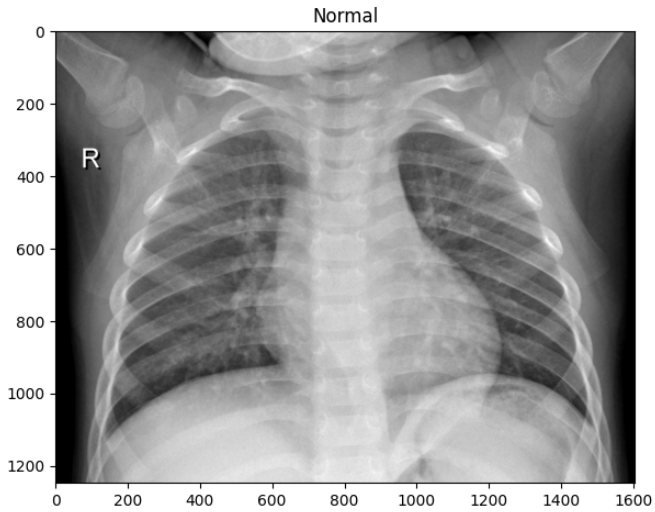
```
No of normal train images after split is: 944
No of sick train images after split is: 2718
No of normal val images is: 405
No of sick val images is: 1165
```

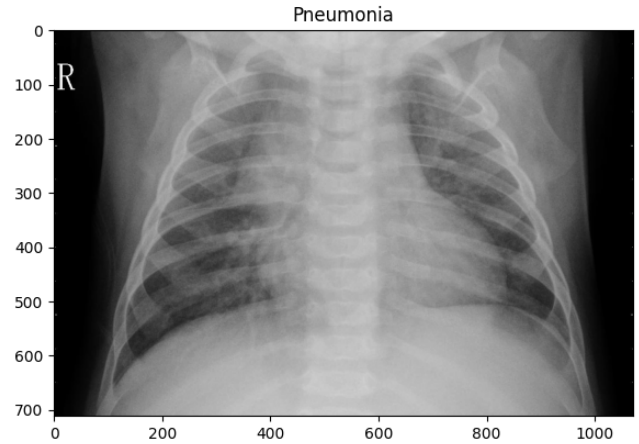
The split was successful. However, with 944 normal images and 2718 "sick"images in the training dataset.

Investigating the characteristics of our images

```
# Select the 41st image from the normal and pneumonia training directories
norm_pic_file = os.listdir(train_norm)[79]
sick_pic_file = os.listdir(train_sick)[88]

# Build full file paths for the selected images
norm_pic_full_filename = train_norm + norm_pic_file
sick_pic_full_filename = train_sick + sick_pic_file

# Load and convert the selected images to binary (black and white)
pic_norm = Image.open(norm_pic_full_filename).convert('L')
pic_sick = Image.open(sick_pic_full_filename).convert('L')

# Create a figure for side-by-side visualization of the normal and pneumonia images
f = plt.figure(figsize=(15,15))
a_norm = f.add_subplot(1,2,1)
img_plot = plt.imshow(pic_norm,cmap="gray")
a_norm.set_title("Normal")

a_sick = f.add_subplot(1,2,2)
img_plot = plt.imshow(pic_sick,cmap="gray")
a_sick.set_title("Pneumonia")
```

```
Text(0.5, 1.0, 'Pneumonia')
```





There exists a disparity in the sizes of the images, indicating a potential imbalance.

```
# confirming that the images are of different sizes
print(pic_norm.size)
print(pic_sick.size)
```

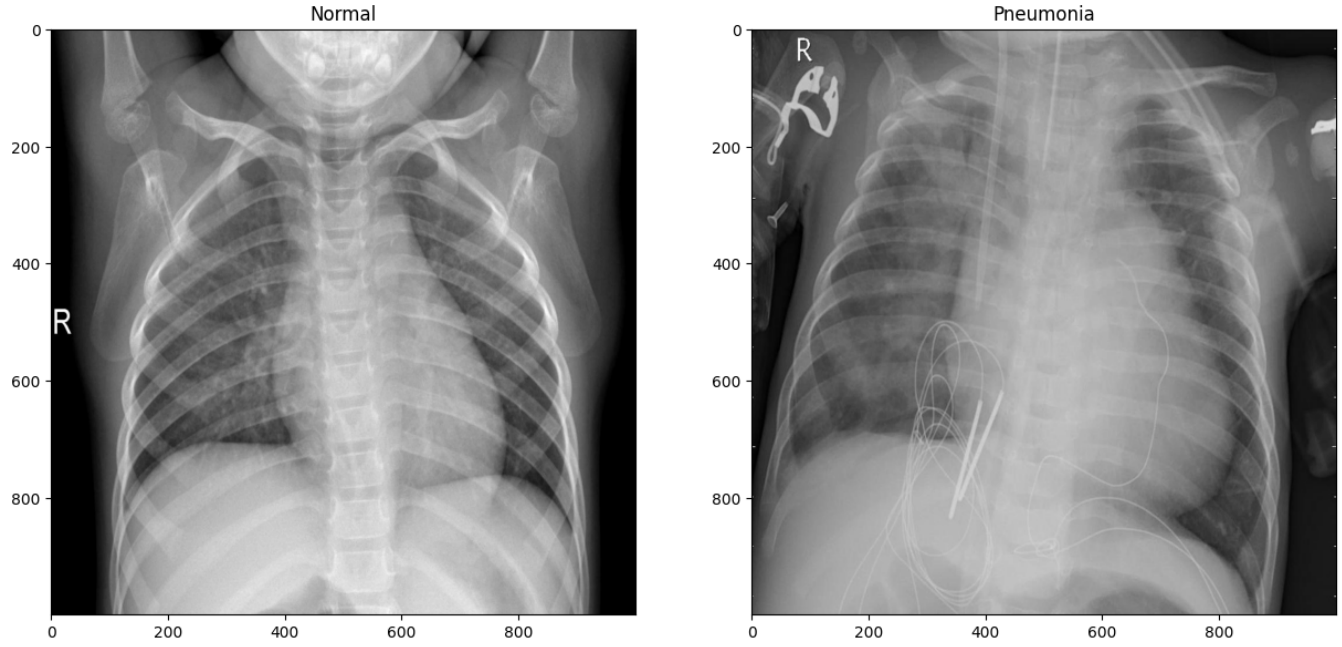
```
(1604, 1248)
(1072, 712)
```

We need to standardize the images by resizing them. CNN will only accept inputs that have fixed sizes. Furthermore, resizing ensures that when modelling, our input has consistent features and patterns, therefore reducing the chances of overfitting. We will create a new directory for the resized images as opposed to overwriting the original images. Though the latter is more efficient, it will lead to loss of original data, which may be needed for future comparison.

```
#run only once
#  Target size based on the images above
# target_size = (1000,1000)

# we want to resize both the normal and pneumonia images in the train files
# variable "train norm" contains directory path to where all the normal train images are fou
# Get a list of all normal train image files in the directory
# normal_train_list = os.listdir(train_norm)

# Create a new directory to store resized images
# resized_images_directory = "train_files/val_files/train_resized/NORMAL/"
# os.makedirs(resized_images_directory, exist_ok=True)

# Resize each normal image and save it to the new directory
# for image_file in normal_train_list:
    # image_path = os.path.join(train_norm, image_file)
    # resized_image = Image.open(image_path).resize(target_size)

    # Save the resized image to the new directory
    # resized_image_path = os.path.join(resized_images_directory, image_file)
    # resized_image.save(resized_image_path)


# variable "train sick" contains directory path to where all the sick train images are found
# Get a list of all "sick" train image files in the directory
# sick_train_list = os.listdir(train_sick)

# Create a new directory to store resized images
# resized_images_directory = "train_files/val_files/train_resized/PNEUMONIA/"
# os.makedirs(resized_images_directory, exist_ok=True)

# Resize each normal image and save it to the new directory
# for image_file in sick_train_list:
    # image_path = os.path.join(train_sick, image_file)
    # resized_image = Image.open(image_path).resize(target_size)

    # Save the resized image to the new directory
    # resized_image_path = os.path.join(resized_images_directory, image_file)
    # resized_image.save(resized_image_path)


# specifying path to the new set of resized train images
train_files_resized = "train_files/val_files/train_resized/"
train_norm_resized = train_files_resized +"NORMAL/"
train_sick_resized = train_files_resized +"PNEUMONIA/"
```

```
# Using the same code above to visually inspect whether our images have been resized
# Select random images from the normal and pneumonia training directories
norm_pic_file = os.listdir(train_norm_resized)[40]
sick_pic_file = os.listdir(train_sick_resized)[105]

# Build full file paths for the selected images
norm_pic_full_filename = train_norm_resized + norm_pic_file
sick_pic_full_filename = train_sick_resized + sick_pic_file

# Load and convert the selected images to binary (black and white)
pic_norm = Image.open(norm_pic_full_filename).convert("L")
pic_sick = Image.open(sick_pic_full_filename).convert("L")

# Create a figure for side-by-side visualization of the normal and pneumonia images
f = plt.figure(figsize=(15,15))
a_norm = f.add_subplot(1,2,1)
img_plot = plt.imshow(pic_norm,cmap="gray")
a_norm.set_title("Normal")

a_sick = f.add_subplot(1,2,2)
img_plot = plt.imshow(pic_sick,cmap="gray")
a_sick.set_title("Pneumonia")
```

```
Text(0.5, 1.0, 'Pneumonia')
```



```
# confirming resizing
print(pic_norm.size)
print(pic_sick.size)
```

```
(1000, 1000)
(1000, 1000)
```

Data Augmentation

Examing the pixel values of our images

```
# Convertimage to a NumPy array to access pixel values
pixel_values_norm = list(pic_norm.getdata())
pixel_values_sick = list(pic_sick.getdata())

# Display the pixel values
print("Pixel values before rescaling:", pixel_values_norm[500:550])
print("Pixel values before rescaling:", pixel_values_sick[1000:1050])
```

```
Pixel values before rescaling: [203, 197, 195, 197, 201, 200, 200, 200, 201, 200, 197, 1
Pixel values before rescaling: [23, 21, 18, 16, 17, 19, 22, 24, 27, 32, 39, 43, 43, 42,
```

We need to normalize our pixel values. Normalizing them will lead to better performance of our neural network and will also increase the computational efficiency. This will be done by passing our data through Image Data Generator. Furthermore, we will apply various transformations, including rotation, to our train set. Creating variations in our train set will present our model with slightly modified verisons of training data and consequently help it to become more robust

```
# create geenrator object for train set
train_generator = ImageDataGenerator(rescale=1./255,
                                     rotation_range=20,
                                     width_shift_range=0.2,
                                     height_shift_range=0.2,
                                     shear_range=0.2,
                                     zoom_range=0.2,
                                     horizontal_flip=True
                                      ).flow_from_directory(train_files_resized,
                                                    target_size=(700,700),
                                                    batch_size=32,shuffle=False)

# create geenrator object for val set
val_generator = ImageDataGenerator(rescale=1./255).flow_from_directory(val_files, target_siz
                                                    batch_size = 32,shuffle=False)

# # create geenrator object for test set
test_generator = ImageDataGenerator(rescale=1./255).flow_from_directory(test_files, target_s
                                                    batch_size = 32,shuffle=False)
```

```
Found 3662 images belonging to 2 classes.
Found 1570 images belonging to 2 classes.
Found 624 images belonging to 2 classes.
```

```
# extract a batch of our data from generator object
train_images, train_labels = next(train_generator)
val_images, val_labels = next(val_generator)
test_images, test_labels = next(test_generator)
```

```
# explore dataset after extraction
print("Train images shape:", train_images.shape)
print("Val images shape:", val_images.shape)
print("Test images shape:", test_images.shape)

print("Train labels shape:", train_labels.shape)
print("Val labels shape:", val_labels.shape)
print("Test labels shape:", test_labels.shape)
```

```
Train images shape: (32, 700, 700, 3)
Val images shape: (32, 700, 700, 3)
Test images shape: (32, 700, 700, 3)
Train labels shape: (32, 2)
Val labels shape: (32, 2)
Test labels shape: (32, 2)
```

The labels to our dataset are as we expect, with each set of data (train,val,set) containing 2 labels: Normal and Pneumonia. Because of the size of the dataset, we could not process and extract the

images in batches larger than 500. However, the shape of all our sets of images are still as we expect, with the given size. Although the shape shows that our images are in RBG, we will maintain the images in 3 color channels because our models only accepts 3 channels for input shape.

Modelling

Baseline Model VGG16

In this project, we've chosen the VGG16 model as our baseline for image classification. The pre-trained VGG16 model is a convolutional neural network (CNN) architecture that has been trained on a large dataset. Renowned for its simplicity and effectiveness, VGG16 comes pre-trained on the ImageNet dataset, offering a solid foundation for our specific task. Leveraging transfer learning, we aim to capitalize on its learned features and subsequently fine-tune the model on our dataset to achieve optimal performance in classifying images

```
# Load the pre-trained VGG16 model
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(700, 700, 3))

# Freeze the convolutional layers
for layer in base_model.layers:
    layer.trainable = False

# Create custom model on top of the pre-trained model
model = models.Sequential()
model.add(base_model)
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(2, activation='softmax'))

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16
58889256/58889256 [==============================] - 0s 0us/step
```

Training the Model. This involves fitting the model to the training data.

```
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // train_generator.batch_size,
    epochs=10,
    validation_data=val_generator,
    validation_steps=val_generator.samples // val_generator.batch_size
)
```

```
Epoch 1/10
114/114 [==============================] - 652s 5s/step - loss: 22.5408 - accuracy: 0.72
Epoch 2/10
114/114 [==============================] - 625s 5s/step - loss: 0.7587 - accuracy: 0.803
Epoch 3/10
114/114 [==============================] - 610s 5s/step - loss: 0.3962 - accuracy: 0.816
Epoch 4/10
114/114 [==============================] - 597s 5s/step - loss: 0.5540 - accuracy: 0.783
Epoch 5/10
114/114 [==============================] - 596s 5s/step - loss: 0.3884 - accuracy: 0.807
Epoch 6/10
114/114 [==============================] - 613s 5s/step - loss: 0.5120 - accuracy: 0.772
Epoch 7/10
114/114 [==============================] - 624s 5s/step - loss: 0.3951 - accuracy: 0.779
Epoch 8/10
114/114 [==============================] - 599s 5s/step - loss: 0.3840 - accuracy: 0.815
Epoch 9/10
114/114 [==============================] - 616s 5s/step - loss: 0.3790 - accuracy: 0.817
Epoch 10/10
114/114 [==============================] - 586s 5s/step - loss: 0.3591 - accuracy: 0.813
```

```
model.save('xray_classification_model.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3079: UserWarning:
  saving_api.save_model(
```

```
# Evaluate model on the test set
evaluation = model.evaluate(test_generator, steps=test_generator.samples // test_generator.b

# Print the evaluation results
print("Test Loss:", evaluation[0])
print("Test Accuracy:", evaluation[1])
```

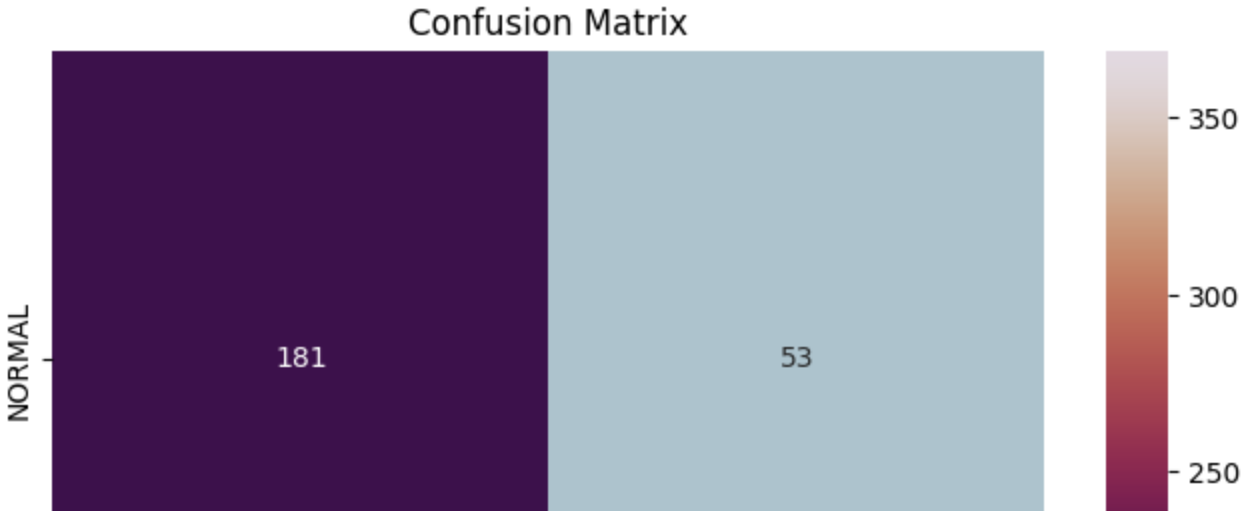
```
19/19 [==============================] - 25s 1s/step - loss: 0.3157 - accuracy: 0.8816
Test Loss: 0.3157232105731964
Test Accuracy: 0.8815789222717285
```

The evaluation results demonstrate strong model performance on the test set. With a test loss of 0.3157, indicating effective predictive capabilities, and a test accuracy of 88.16%, the model

exhibits robust generalization to previously unseen data. The low loss value suggests precise predictions, while the high accuracy underscores the model's efficacy in correctly classifying test images. These results affirm the model's proficiency in the specific binary classification task, showcasing its ability to provide accurate and reliable predictions.

```
# Make Predictions
# use np.ceil to ensure no data is being cut off due to batch size
predictions = model.predict(test_generator, steps= int (np.ceil(test_generator.samples / tes


# Convert Predictions and True Labels to Classes
predicted_classes = np.argmax(predictions, axis=1)
true_classes = test_generator.classes

# Instantiate Confusion Matrix
conf_matrix = confusion_matrix(true_classes, predicted_classes)

# Plot Confusion Matrix
class_labels = list(test_generator.class_indices.keys())
plt.figure(figsize=(8, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='twilight', xticklabels=class_labels, yti
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

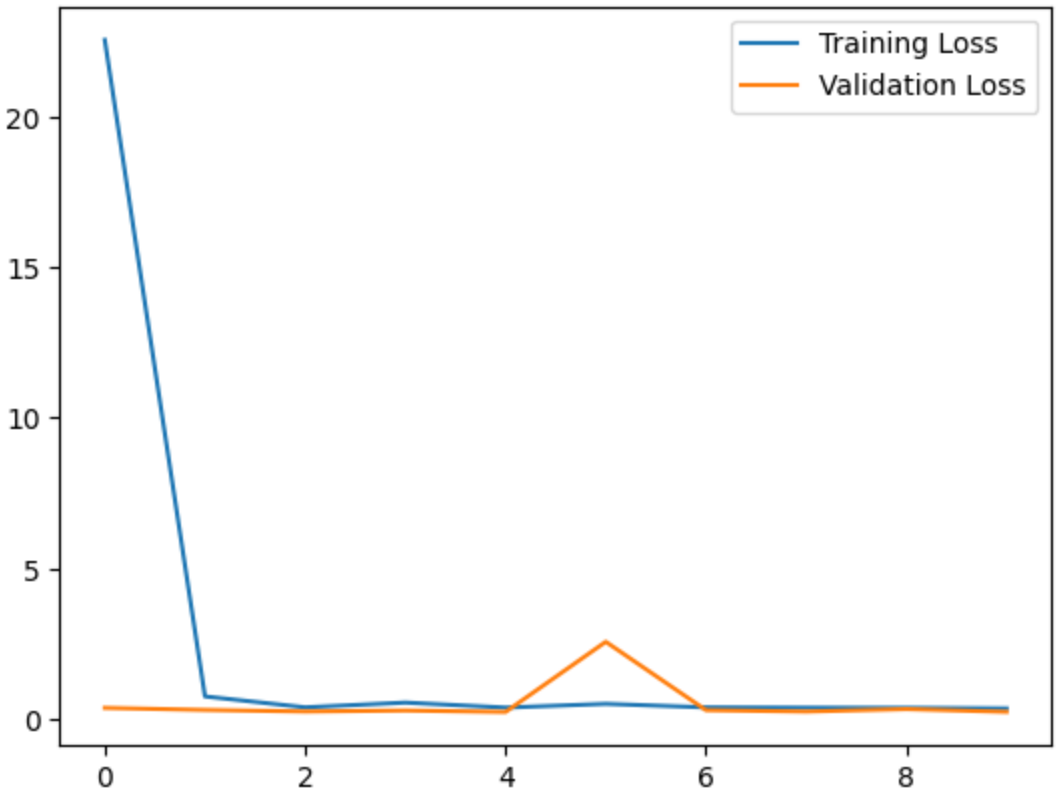
```
20/20 [==============================] - 31s 2s/step
```

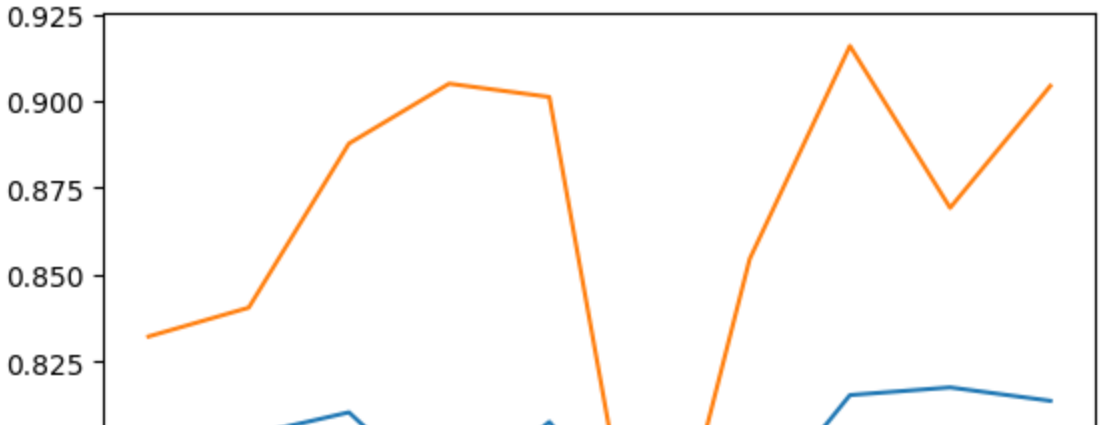


The confusion matrix indicates that our model performed well in correctly identifying cases of pneumonia (369 true positives) but exhibited some misclassifications. Specifically, there were 53 false positives, where the model incorrectly predicted pneumonia when the actual condition was normal. Additionally, there were 21 false negatives, where the model failed to identify pneumonia when it was present. The 181 true negatives indicate accurate predictions of normal cases. Overall, the model demonstrates effectiveness in detecting pneumonia, but there is room for improvement, particularly in reducing false positives and false negatives.

Visualizing Training History:

```
# Plot training history
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.show()

plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.legend()
plt.show()
```





The performance of our model, as assessed through accuracy and loss metrics during training, exhibits a strong alignment with validation scores at each step. This indicates a consistent and effective learning process, where the model's behavior on the training dataset closely mirrors its performance on an independent validation dataset. Such congruence suggests that our model generalizes well to unseen data, reflecting a robust and reliable predictive capability. The close correspondence between training and validation metrics underscores the model's overall proficiency and reliability in making accurate predictions.

Model 2- VGG19

In our project, the second model in our arsenal is the VGG19 architecture, which serves as an upgraded version of the VGG16 model. While both models share the same fundamental principles,

VGG19 features an additional set of convolutional layers, resulting in a deeper neural network with 19 layers. This augmentation introduces a potential increase in representational capacity but also demands more computational resources during training and inference compared to its predecessor. By opting for VGG19, we aim to explore the benefits of a deeper architecture and assess its impact on the model's performance in our image classification task, considering the trade-off between increased complexity and computational efficiency.since our images are grayscale the model was adjusted to fit to that.

```
# Load the pre-trained VGG19 model
cnn_base = VGG19(weights='imagenet', include_top=False, input_shape=(700, 700, 3))

# Freeze the convolutional layers
for layer in cnn_base.layers:
    layer.trainable = False

# Create custom model on top of the pre-trained model
model = models.Sequential()
model.add(cnn_base)

# Continue building the rest of your model
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(2, activation='softmax'))



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

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19
80134624/80134624 [==============================] - 0s 0us/step
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg19 (Functional)          (None, 21, 21, 512)       20024384

 flatten (Flatten)           (None, 225792)            0

 dense (Dense)               (None, 64)                14450752

 dense_1 (Dense)             (None, 128)               8320

 dense_2 (Dense)             (None, 256)               33024

 dense_3 (Dense)             (None, 128)               32896
```

```
 dense_4 (Dense)             (None, 2)                 258

=================================================================
Total params: 34549634 (131.80 MB)
Trainable params: 14525250 (55.41 MB)
Non-trainable params: 20024384 (76.39 MB)
_________________________________________________________________
```

```
# Compile the model
model.compile(loss='binary_crossentropy',  # Binary classification loss
              optimizer=optimizers.RMSprop(learning_rate=2e-5),
              metrics=['acc'])

# Fitting the Model
history = model.fit(train_generator,
                    steps_per_epoch= train_generator.samples // train_generator.batch_size,
                    epochs=10,
                    validation_data=val_generator,
                    validation_steps=val_generator.samples // val_generator.batch_size)
```

```
Epoch 1/10
114/114 [==============================] - 664s 6s/step - loss: 0.6150 - acc: 0.7399 - v
Epoch 2/10
114/114 [==============================] - 640s 6s/step - loss: 0.5042 - acc: 0.7438 - v
Epoch 3/10
114/114 [==============================] - 645s 6s/step - loss: 0.4140 - acc: 0.8118 - v
Epoch 4/10
114/114 [==============================] - 620s 5s/step - loss: 0.3659 - acc: 0.8317 - v
Epoch 5/10
114/114 [==============================] - 614s 5s/step - loss: 0.3314 - acc: 0.8493 - v
Epoch 6/10
114/114 [==============================] - 626s 5s/step - loss: 0.3036 - acc: 0.8645 - v
Epoch 7/10
114/114 [==============================] - 681s 6s/step - loss: 0.2976 - acc: 0.8719 - v
Epoch 8/10
114/114 [==============================] - 636s 6s/step - loss: 0.2952 - acc: 0.8656 - v
Epoch 9/10
114/114 [==============================] - 698s 6s/step - loss: 0.2714 - acc: 0.8813 - v
Epoch 10/10
114/114 [==============================] - 678s 6s/step - loss: 0.2818 - acc: 0.8749 - v
```

```
#save model
model.save('results_second_model.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3079: UserWarning:
  saving_api.save_model(
```

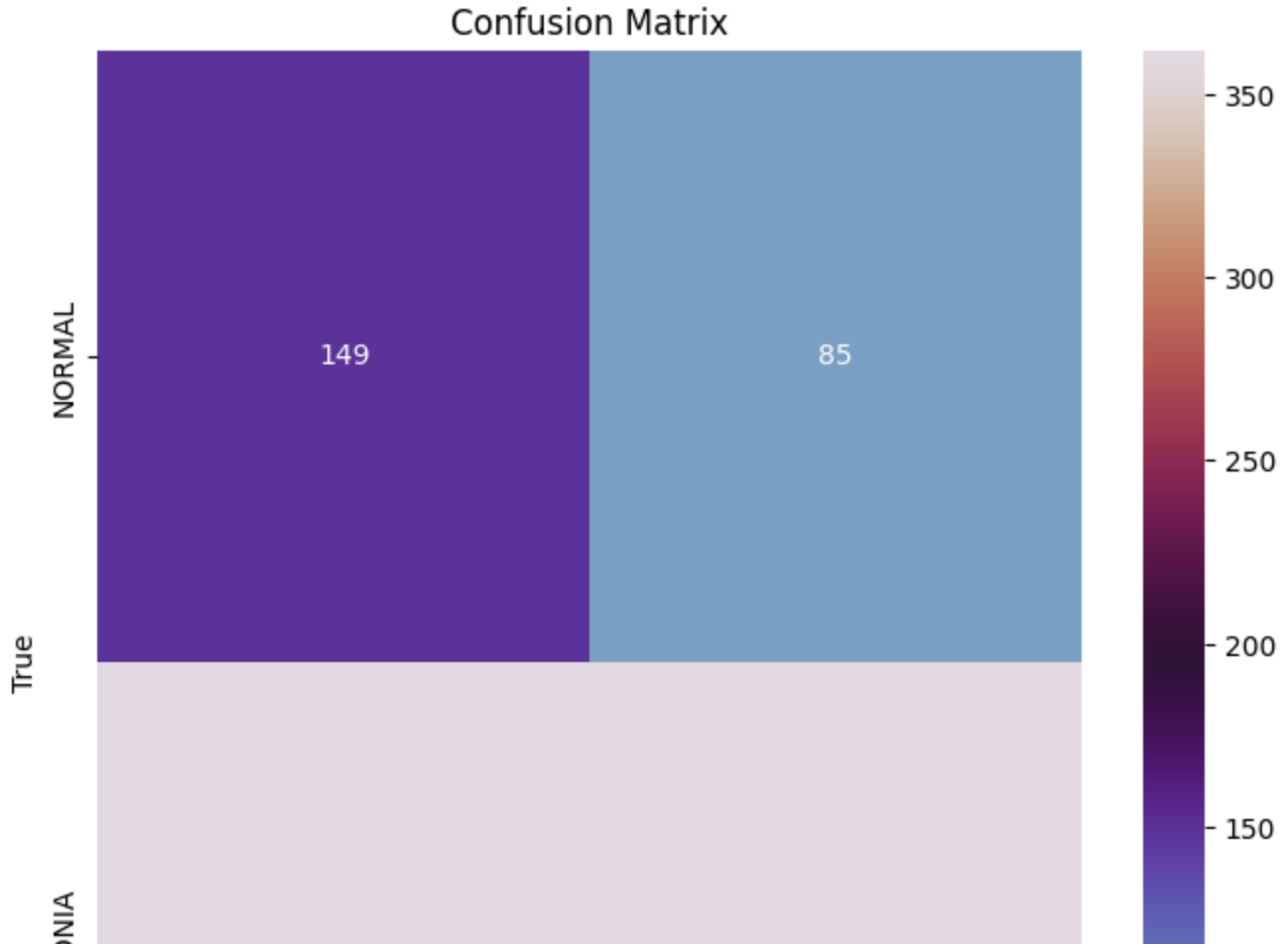
```
# Evaluate model on the test set
evaluation = model.evaluate(test_generator, steps=test_generator.samples // test_generator.b

# Print the evaluation results
print("Test Loss:", evaluation[0])
print("Test Accuracy:", evaluation[1])
```

```
19/19 [==============================] - 33s 2s/step - loss: 0.3852 - acc: 0.8158
Test Loss: 0.3851935863494873
Test Accuracy: 0.8157894611358643
```

Our model demonstrates commendable performance on the test set, achieving a test loss of 0.3852 and an accuracy of 81.58%. This signifies that the model effectively minimized errors during prediction, with the majority of test images correctly classified. While there's room for improvement, these results indicate a robust and reliable performance in the targeted classification task.

```
# Make Predictions
# use np.ceil to ensure no data is being cut off due to batch size
predictions = model.predict(test_generator, steps= int (np.ceil(test_generator.samples / tes


# Convert Predictions and True Labels to Classes
predicted_classes = np.argmax(predictions, axis=1)
true_classes = test_generator.classes

# Instantiate Confusion Matrix
conf_matrix = confusion_matrix(true_classes, predicted_classes)

# Plot Confusion Matrix
class_labels = list(test_generator.class_indices.keys())
plt.figure(figsize=(8, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='twilight', xticklabels=class_labels, yti
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

```
20/20 [==============================] - 46s 2s/step
```



The confusion matrix reveals that our model performs well in identifying pneumonia cases with 362 true positives, indicating accurate predictions. However, there are 85 false positives, where the model wrongly predicts pneumonia in normal cases, and 28 false negatives, representing instances where the model fails to identify pneumonia when it's present. The 149 true negatives correctly identify normal cases. Overall, while the model excels in pneumonia detection, there is room for improvement in reducing false positives and false negatives.

NORMAL PNEUMONIA

```
# Plot training & validation accuracy values
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'], loc='upper left')

# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
```