University for the Creative Arts

BERLIN SCHOOL OF BUSINESS & INNOVATION

Essay / Assignment Title: AI-Driven Pneumonia Diagnosis: Chest X-ray Classification

Programme title: MSc Data Analytics

Name: Birce SARI

Year: 2024

TABLE OF CONTENTS

Statement of compliance with academic ethics and the avoidance of plagiarism

I honestly declare that this dissertation is entirely my own work and none of its part has been copied from printed or electronic sources, translated from foreign sources and reproduced from essays of other researchers or students. Wherever I have been based on ideas or other people texts I clearly declare it through the good use of references following academic ethics.

(In the case that is proved that part of the essay does not constitute an original work, but a copy of an already published essay or from another source, the student will be expelled permanently from the postgraduate program).

Name and Surname (Capital letters):

........................BIRCE SARI...

Date: ...31...../...10.../..24....

INTRODUCTION

In the technological era of humanity, computer vision and artificial intelligence take place in improving healthcare processes by analyzing medical images. This project is to detect pneumonia in chest X-ray images with deep learning techniques. It is expected to implement the optimized learning parameters, scalable model, and high accuracies to predict the patients' illness if applicable. The primary objective of the project is to identify pneumonia accurately and as early as possible to improve patient outcomes and reduce the healthcare processes (Kermany et al., 2018).

Pneumonia is a serious lung infection caused by bacteria, viruses, or fungi, leading to inflammation and fluid in the air sacs, which makes breathing difficult. It is a major global health risk, especially affecting infants, children, the elderly, and those with weakened immune systems, resulting in millions of cases and deaths annually (Association, 2024).

The study by Kermany et al. (2018) focuses on utilizing chest X-rays to identify characteristic patterns of opacity in the lungs. In chest X-rays, pneumonia generally appears as an area of increased whiteness in the lung fields. These opacities represent areas of inflammation and fluid accumulation in the lungs. Pneumonia can be bacterial or viral, but we will consider them as one group.

The Kaggle link supplied (Mooney, 2018) is the main dataset that will serve as the foundation for training, testing, and validation processes for image classification problems. This practical example will help us to understand the dynamic of the computer vision techniques and challenges in deep learning models for medical image classification (Litjens et al., 2017; Rajpurkar et al., 2017). We will implement the Python code for image classification and feature detection in the context of respiratory diseases (Wang et al., 2017).

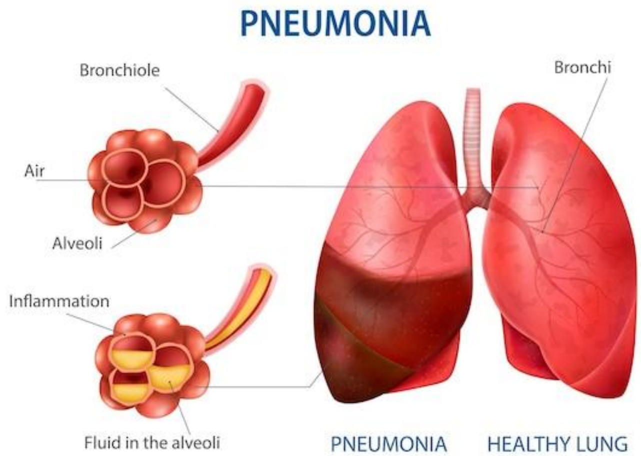

Figure 1: Pneumonia introduction (Harshit, 2024)

The provided dataset is constructed with three folders for each process with corresponding subfolders of ‘Normal’ and ‘Pneumonia’ to classify them for supervised learning models. In total, 5,863 chest X-ray images, provide a substantial resource for developing and evaluating an AI model. The dataset was controlled against low quality or unreadability by expert physicians. This gave us a minimum defect for inputs of modeling and testing accordingly which help to identify the illness accurately (Mooney, 2018).

```python
# DATA VISUALIZATION SECTION

# Parameters working as constants for our visualization to control
figure_size = (18,10)
num_images = 12
# bone, viridis
colormap = 'bone'
title_size = 20

def plot_sample_images(data,LABELS,num_images,figure_size,colormap,title_size):
    """..."""
    random_indices = np.random.choice(len(data),num_images,replace=False)
    plt.figure(figsize=figure_size)
    for i,idx in enumerate(random_indices):
        plt.subplot(3,4,i + 1)
        plt.imshow(data[idx],cmap=colormap)
        plt.title('Pneumonia' if LABELS[idx] == 0 else 'Normal')
        plt.axis('off')
    plt.suptitle("Sample Set Image Examples",size=title_size)
    plt.tight_layout()
    plt.show()

# Checking number of Sample
plot_sample_images(train_data,train_labels,num_images,figure_size,colormap,title_size)

# Training Data Distribution
train_df = pd.DataFrame({
    "Labels": train_labels,
    "Set": "Train"
})
```

Figure 2: EDA python code for data visualization part 1 under 'Data Exploration Analysis.py'

```python
val_df = pd.DataFrame({
    "Labels": val_labels,
    "Set": "Validation"
})

test_df = pd.DataFrame({
    "Labels": test_labels,
    "Set": "Test"
})

# Combine all DataFrames
combined_df = pd.concat([train_df,test_df,val_df])

# Data Distribution Graph
plt.figure(figsize=(9,5))
##0073e6,ff758f
colors = sns.light_palette("#0073e6",n_colors=7)
ax = sns.countplot(data=combined_df,x='Labels',hue='Set',palette=[colors[1],colors[3],colors[6]])
ax.set_xticklabels(['Pneumonia','Normal'])

# Annotate bars with the counts
for p in ax.patches:
    count = int(p.get_height())  # Get height of each bar (the count)
    ax.annotate(f'{count}',  # Text to annotate with
                (p.get_x() + p.get_width() / 2.,count),  # Position of the text
                ha='center',  # Horizontal alignment
                va='baseline')  # Vertical alignment
# displaying the title
plt.title("Image Distribution Graph")
plt.show()
```

Figure 3: EDA python code for data visualization part 2 under 'Data Exploration Analysis.py'

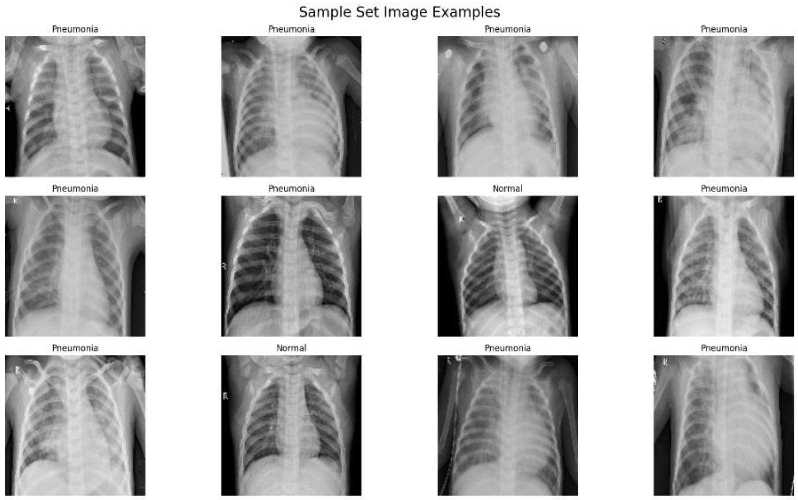

Figure 4: Sample data set for controlling the images uploaded

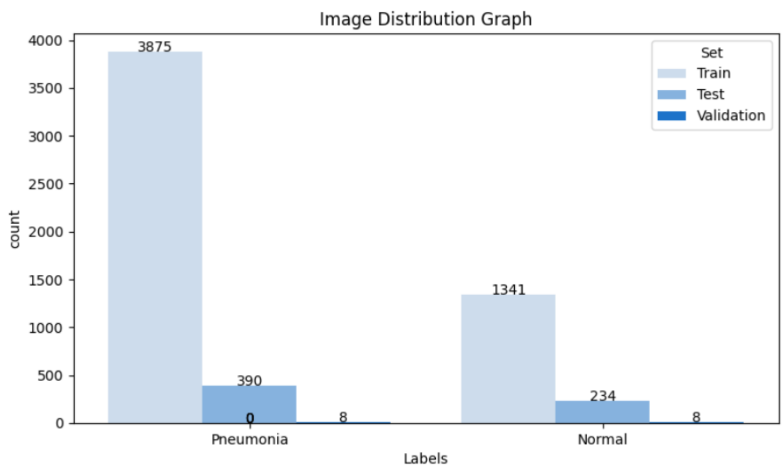

Figure 5: Number of images in each folder distributed into 'pneumonia' and 'normal' subfolders

CHAPTER ONE: Neural Network Architecture Design

Image analysis to classify them can be troubling due to the complexity of the files. Convolutional Neural Networks (CNNs) serve us a special, well-suited class of deep learning models for image analysis. The architecture is designed for adaptive learning of the spatial hierarchical features from inputs. This makes it useful for the detection of pneumonia from chest X-rays (Alzubaidi et al., 2021).

CNN architecture generally consists of specialized layers that are processed in a straightforward; convolutional-layers, pooling-layers, fully-connected-layers, activation functions, and output-layer. Convolutional layers extract features from images by slicing the image into kernels (square dimensions) for elementwise multiplication of summation in each of them to detect edges, textures, or any other structure. It allows local connectivity for the receptive fields for deeper analysis whereas the parameter-sharing algorithm helps the model to have universal parameter weights among all (Mishra, 2020). Pooling-layers reduce the spatial dimensions of the feature maps created in convolutional-layers to control efficiency by decreasing the computations to make the model more robust and reducing the overfitting probability. Max-pooling, average-pooling can be used for each window as the problem is needed to downsample the feature maps. Fully-connected-layers interpret the features of convolutional-layer to determine the classification-layers with flattening of a 3D image into a 1D vector. Activation-functions (most used is Rectified Linear Unit (ReLU)) add non-linearity for more complex patterns to diminish the effect of gradients on learning curves and enhance the processes of reaching optimal parameters. Like ReLU, batch normalization stabilizes the learning process and increases the parameter-choosing process. Batch normalization is not used in all models and they come before activation-layer. The output-layer gives a sigmoid activation layer to determine the images either in the selected category (pneumonia) or out (normal).

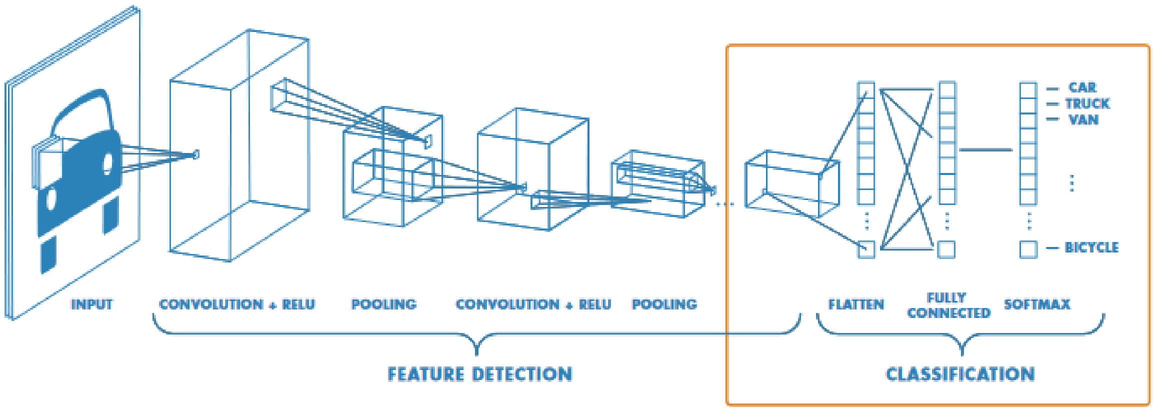

Figure 6: CNN architecture (MATLAB, 2021)

For this project, we choose ResNet50 as a CNN architecture for powerful image classification for its scalability, performance, and higher accuracy. The model uses residual blocks that allow gradient vectors to improve the training process by adjusting the weights to minimize the loss function with smoother learning. Since the high volume of layers can fail the model, ResNet50 helps us bypass the least effective ones to distinguish between normal and pneumonia-affected lungs accurately. This is a pre-trained model with fine tunes that help us elevate the training process with the concept of transfer learning. This model allows us to use it according to our computational resources by scaling its depth as well as modifying the model due to our limitations. The model is especially well-suited for tasks like pneumonia identification from chest X-rays because of its strong feature extraction skills and ability to generalize across various image types. By utilizing the ResNet architecture's advantages, medical practitioners can improve diagnostic precision, which will benefit patients and make better use of available resources.

CHAPTER TWO: Code Implementation

The dataset is studied separately as stated before to understand the images and their size. Since we know these, we will be inspecting the rest of the problem under data loading, data processing, learning, model performance, and as a separate part the prediction samples.

Data Loading Section

```
# Importing Dependencies
import warnings
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import classification_report,confusion_matrix
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout, BatchNormalization
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping, ModelCheckpoint
import os
os.environ["TF_ENABLE_ONEDNN_OPTS"] = "0"  # Disable oneDNN optimizations
import cv2
import numpy as np

# Ignore warnings
warnings.filterwarnings('ignore')
```

Figure 7: Importing dependencies code snippet

The dependencies added due to their usages; pandas and numpy for data manipulation, matplotlib and seaborn added for visualization purposes, cv2 for image processing, tensorflow for machine learning, and sklearn for evaluating the outputs.

```
# DATA LOADING SECTION

# Define directories for dataset
TRAIN_DIR = 'chest_xray/train'
TEST_DIR = 'chest_xray/test'
VAL_DIR = 'chest_xray/val'

# Loading Dataset
# List of label names
LABELS = ['PNEUMONIA','NORMAL']
# Image size to format them into same size during the loading stage
img_size = 256
```

Figure 8: Data directories definition code snippet

The directories are defined for train, test, and validation sets for organized paths for easier usages throughout the code.

```python
def load_image_data(directory,label):

    path = os.path.join(directory,label)
    class_num = LABELS.index(label)
    data,labels = [],[]

    for img in os.listdir(path):
        #img_arr = cv2.imread(os.path.join(path,img))
        img_arr = cv2.imread(os.path.join(path,img),cv2.IMREAD_COLOR)
        resized_arr = cv2.resize(img_arr, (img_size,img_size))
        data.append(resized_arr)
        labels.append(class_num)

    return data,labels
```

Figure 9: Loading image data function code snippet

Data is loaded with corresponding labels, resized, and colored for more layers. We resized for a consistent input for the CNN model of ours.

```python
def load_training_data(data_dir):

    all_data,all_labels = [],[]

    for label in LABELS:
        data,labels = load_image_data(data_dir,label)
        all_data.extend(data)
        all_labels.extend(labels)

    return np.array(all_data),np.array(all_labels)
```

Figure 10: Loading training data function code snippet

In this snippet, each label and image are distributed to the corresponding array for easy usage in the model.

```python
# Load and preprocess training, testing, and validation data
train_data,train_labels = load_training_data(TRAIN_DIR)
test_data,test_labels = load_training_data(TEST_DIR)
val_data,val_labels = load_training_data(VAL_DIR)
```

Figure 11: Loading and pre-processing each set of data code snippet

The main folders of train, test, and val are now imported to the code to use as needed in further steps.

Data Processing Section

```
# DATA VISUALIZATION SECTION (EDA) in Data Exploration Analysis.py file

# DATA PROCESSING SECTION

# Normalize and reshape the data for the model
X_train, X_test, X_val = [x / 255.0 for x in [train_data, test_data, val_data]]
X_train = X_train.reshape(-1, img_size, img_size, 3)
X_test = X_test.reshape(-1, img_size, img_size, 3)
X_val = X_val.reshape(-1, img_size, img_size, 3)
y_train, y_test, y_val = map(np.array, [train_labels, test_labels, val_labels])
```

Figure 12: Normalizing and reshaping data code snippet

Normalizing the datasets to scale them in a range of 0 to 1, reshaping arrays for CNN usage of heights, widths, and color channels for a higher training efficiency and convergence speed.

```
# Enhanced Data Augmentation with ImageDataGenerator
data_generator = ImageDataGenerator(
    rotation_range=40,              # Rotate images up to 40 degrees from 30
    width_shift_range=0.2,          # Shift width up to 20% from 0.1
    height_shift_range=0.2,         # Shift height up to 20% from 0.1
    shear_range=0.2,                # Apply shearing
    zoom_range=0.3,                 # Zoom in/out within 30% from 0.2
    #horizontal_flip=True,           # Flip images horizontally (removed since the X-Rays will not be flipped)
    brightness_range=[0.7, 1.3],    # Adjust brightness (darker to brighter)
    fill_mode='nearest',            # Fill mode for empty pixels after shifts
    channel_shift_range=20.0        # Adjust color intensity by shifting channels
)
```

Figure 13: Enhanced data augmentation code snippet

ImageDataGenerator helps us to increase the intensity of the dataset with various transformation types like rotating, shifting, zooming, and adjusting brightness. This step helps us to expand the dataset by adding reconstructed images for exposing the model to a wider variety of objects.

```
# Checking and balancing the test data
if len(X_test) > len(y_test):
    # Down sampling X_test to match the number of y_test
    selected_indices = np.random.choice(len(X_test), len(y_test), replace=False)
    X_test = X_test[selected_indices]
elif len(X_test) < len(y_test):
    # Down sampling y_test to match the number of X_test
    selected_indices = np.random.choice(len(y_test), len(X_test), replace=False)
    y_test = y_test[selected_indices]

# Ensuring X_test and y_test have the same number of samples
assert len(X_test) == len(y_test), "X_test and y_test must have the same number of samples"

# Printing the new shapes for verification
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
```

Figure 14: Balancing test datasets for the same number of samples code snippet

```python
# Printing the new shapes for verification
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)

# Balancing X_train and y_train
if len(X_train) > len(y_train):
    selected_indices = np.random.choice(len(X_train), len(y_train), replace=False)
    X_train = X_train[selected_indices]
elif len(X_train) < len(y_train):
    selected_indices = np.random.choice(len(y_train), len(X_train), replace=False)
    y_train = y_train[selected_indices]

# Ensuring X_train and y_train have the same number of samples
assert len(X_train) == len(y_train), "X_train and y_train must have the same number of samples"

# Printing the new shapes for verification
print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
```

Figure 15: Balancing train datasets for the same number of samples code snippet

```python
# Balancing X_val and y_val
if len(X_val) > len(y_val):
    selected_indices = np.random.choice(len(X_val), len(y_val), replace=False)
    X_val = X_val[selected_indices]
elif len(X_val) < len(y_val):
    selected_indices = np.random.choice(len(y_val), len(X_val), replace=False)
    y_val = y_val[selected_indices]

# Ensuring X_val and y_val have the same number of samples
assert len(X_val) == len(y_val), "X_val and y_val must have the same number of samples"

# Printing the new shapes to verify them
print("X_val shape:", X_val.shape)
print("y_val shape:", y_val.shape)
```

Figure 16: Balancing validation datasets for the same number of samples code snippet

Since many of the running trials ended up with errors for not being the same size, this code for downsampling was needed for each of the sets during the debugging process.

```
X_test shape: (624, 256, 256, 3)
y_test shape: (624,)
X_train shape: (5216, 256, 256, 3)
y_train shape: (5216,)
X_val shape: (16, 256, 256, 3)
y_val shape: (16,)
```

Figure 17: Outputs of each print statement given in the previous three images

Learning Section

```
# LEARNING SECTION (using ResNet50)

# Load the model with pre-trained weights
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(img_size, img_size, 3))

# Freeze the layers to avoid re-training them
for layer in base_model.layers:
    layer.trainable = False

# Custom model
x = base_model.output
x = GlobalAveragePooling2D()(x)  # Add global pooling layer to reduce parameters -flattens
x = Dense(128, activation='relu')(x)# Fully connected layer
x = Dropout(0.3)(x) # Dropout for regularization
output = Dense(1, activation='sigmoid')(x)  # Sigmoid activation  for binary classification

# Define the model
model = Model(inputs=base_model.input, outputs=output)
```

Figure 18: CNN model with ResNet50 transfer learning code snippet

Resnet50 model is loaded and the preset weight of the ImageNet dataset is used for higher accuracy expectance. To obtain this expectance, we freezed the retraining step, added additional layers to the original model, defined and binary classification with sigmoid function.

```
# Compile the model
optimizer = Adam(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
# Print model summary
model.summary()
```

Figure 19: Compiling model code snippet

Adam optimizer used for binary cross-entropy loss function to introduce the learning rate base for the initial system with a summary of what the general system is structured with. Model output as model.summary() is supplied in appendix part. Our model has many layers, they are input, convolutional, residual, global average pooling, fully-connected (dense), dropout.

```
Total params: 23,850,113 (90.98 MB)
Trainable params: 262,401 (1.00 MB)
Non-trainable params: 23,587,712 (89.98 MB)
```

Figure 20: Model.summary() output screenshot

```
# Unfreeze the last few layers of the base model for fine-tuning
for layer in base_model.layers[-4:]:  # Adjust the number of layers to unfreeze as needed
    layer.trainable = True

# Define callbacks for training
callbacks = [
    ReduceLROnPlateau(monitor='val_accuracy', factor=0.2, patience=3, min_lr=0.0001, verbose=1),
    EarlyStopping(monitor='val_accuracy', patience=5, restore_best_weights=True, verbose=1),
    # ModelCheckpoint('best_model_vgg16.keras', monitor='val_accuracy', save_best_only=True, verbose=1)
    ModelCheckpoint('best_model.keras', monitor='val_accuracy', save_best_only=True, verbose=1)
]
```

Figure 21: Unfreezing and callbacks defined code snippet

We unfreezed the layers of the model for fine-tuning and then added callbacks. Callbacks are for optimizing the performance and accuracy with a better training time, preventing overfitting, and using the best model. To be able to do that, we added ReduceLROnPlateau to tune the learning rate considering the learning curve, EarlyStopping for reducing overfitting while overwriting the weights for the best model, and ModelCheckpoint for reanimating the model with the most recent best model.

```
Epoch 1: val_accuracy improved from -inf to 0.81731, saving model to best_model.keras
163/163 - 741s - 5s/step - accuracy: 0.8930 - loss: 0.2533 - val_accuracy: 0.8173 - val_loss: 0.5543 - learning_rate: 0.0010
Epoch 2/25

Epoch 2: val_accuracy improved from 0.81731 to 0.83654, saving model to best_model.keras
163/163 - 695s - 4s/step - accuracy: 0.9369 - loss: 0.1642 - val_accuracy: 0.8365 - val_loss: 0.5036 - learning_rate: 0.0010
Epoch 3/25

Epoch 3: val_accuracy improved from 0.83654 to 0.90224, saving model to best_model.keras
163/163 - 691s - 4s/step - accuracy: 0.9427 - loss: 0.1455 - val_accuracy: 0.9022 - val_loss: 0.3050 - learning_rate: 0.0010
Epoch 4/25

Epoch 4: val_accuracy did not improve from 0.90224
163/163 - 687s - 4s/step - accuracy: 0.9444 - loss: 0.1362 - val_accuracy: 0.8205 - val_loss: 0.5959 - learning_rate: 0.0010
Epoch 5/25

Epoch 5: val_accuracy did not improve from 0.90224
163/163 - 685s - 4s/step - accuracy: 0.9525 - loss: 0.1217 - val_accuracy: 0.8077 - val_loss: 0.5990 - learning_rate: 0.0010
Epoch 6/25

Epoch 6: ReduceLROnPlateau reducing learning rate to 0.00020000000949949026.

Epoch 6: val_accuracy did not improve from 0.90224
163/163 - 683s - 4s/step - accuracy: 0.9492 - loss: 0.1264 - val_accuracy: 0.8173 - val_loss: 0.6007 - learning_rate: 0.0010
Epoch 7/25

Epoch 7: val_accuracy did not improve from 0.90224
163/163 - 684s - 4s/step - accuracy: 0.9599 - loss: 0.1053 - val_accuracy: 0.8173 - val_loss: 0.6151 - learning_rate: 2.0000e-04
Epoch 8/25

Epoch 8: val_accuracy did not improve from 0.90224
163/163 - 786s - 5s/step - accuracy: 0.9659 - loss: 0.0875 - val_accuracy: 0.8237 - val_loss: 0.5516 - learning_rate: 2.0000e-04
Epoch 8: early stopping
```

Figure 22: Output of the checkpoints

```python
# Data generators for training, validation, and testing data
train_generator = data_generator.flow_from_directory(
    TRAIN_DIR,
    target_size=(img_size, img_size),
    batch_size=32,
    class_mode='binary'
)

val_generator = data_generator.flow_from_directory(
    VAL_DIR,
    target_size=(img_size, img_size),
    batch_size=16,
    class_mode='binary',
    shuffle=False  # Disable shuffle
)

test_generator = data_generator.flow_from_directory(
    TEST_DIR,
    target_size=(img_size, img_size),
    batch_size=32,
    class_mode='binary',
    shuffle=False  # Disable shuffle
)
```

Figure 23: Data generators code snippet

Data generator usage in the code is for augmenting and feeding the batches for optimizing memory usage during the data handling processes.

```
Found 5216 images belonging to 2 classes.
Found 16 images belonging to 2 classes.
Found 624 images belonging to 2 classes.
```

Figure 24: flow_from_directory output screenshot

```python
# Train the model
history = model.fit(
    train_generator,
    epochs=25,
    validation_data=test_generator,
    callbacks=callbacks,
    verbose=2
    #,steps_per_epoch=(len(X_train) // 256)
)
```

Figure 25: Model training code snippet

We need to define the concepts to fit them into our model with test data.

Model Performance

```python
# MODEL PERFORMANCE TESTING

# Visualize Training and Validation Metrics
history_data = history.history
epochs = range(1, len(history_data['accuracy']) + 1)

# Retrieve metrics from the training history
train_acc, train_loss = history_data['accuracy'], history_data['loss']
val_acc, val_loss = history_data['val_accuracy'], history_data['val_loss']
```

Figure 26: Introducing the metrics to monitor code snippet

We are initializing the epoch range and extracting the loss function metrics with accuracy from the trained model.

```python
# Create a figure and axes for the plots
fig, ax = plt.subplots( nrows: 1,  ncols: 2, figsize=(18, 6))

# Main figure title
fig.suptitle( t: 'Model Training and Validation Performance', fontsize=20, fontweight='bold')
# Plot training and validation accuracy
ax[0].plot(epochs, train_acc, 'o-', color='darkgreen', label='Training Accuracy', markersize=8)
ax[0].plot(epochs, val_acc, 's--', color='darkred', label='Validation Accuracy', markersize=8)
ax[0].set_title('Training vs. Validation Accuracy', fontsize=16)
ax[0].set_xlabel('Epochs', fontsize=14)
ax[0].set_ylabel('Accuracy', fontsize=14)
ax[0].legend()
ax[0].grid(True)
# Plot training and validation loss
ax[1].plot(epochs, train_loss, 'o-', color='darkblue', label='Training Loss', markersize=8)
ax[1].plot(epochs, val_loss, 's--', color='orange', label='Validation Loss', markersize=8)
ax[1].set_title('Training vs. Validation Loss', fontsize=16)
ax[1].set_xlabel('Epochs', fontsize=14)
ax[1].set_ylabel('Loss', fontsize=14)
ax[1].legend()
ax[1].grid(True)
# Display the plots
plt.tight_layout(rect=[0, 0.03, 1, 0.95])  # Adjust layout to fit the subtitle
plt.show()
```

Figure 27: Setting up the plotting area and declaring ingredients code snippet

We are creating a figure of line chart for loss function and accuracy metric across epochs for easy comparison. We are aiming to reach a decreased loss while the accuracy increases. Due to the overlapping titles, we needed to tighten it for a lean visual.

```python
# Test Set Evaluation
test_eval = model.evaluate(test_generator, verbose=1)
print("==" * 20)
print(f"Test Set Accuracy - {test_eval[1] * 100:.2f}%")
print(f"Test Set Loss - {test_eval[0]:.4f}")
print("==" * 20)

# Predictions and Confusion Matrix for Test Data
test_predictions = (model.predict(test_generator) >= 0.5).astype(int).reshape(-1)
test_cm = confusion_matrix(test_labels, test_predictions)

plt.figure(figsize=(5, 4))
sns.heatmap(pd.DataFrame(test_cm, index=LABELS, columns=LABELS), cmap="Blues", annot=True, fmt="d")
plt.title("Test Data Confusion Matrix")
plt.xlabel("Predicted Labels")
plt.ylabel("Actual Labels")
plt.show()

print(classification_report(test_labels, test_predictions, target_names=LABELS))
```

Figure 28: Test set evaluation metrics extraction and confusion matrix visual code snippet

We are looking for insights into our model for the test set that the model has never been exposed to. Predictions of our test set with binary thresholds and calculations of the confusion matrix are introduced to obtain the true positives and other possible outcomes. The metrics of f1-score, precision, and recall.

```
Restoring model weights from the end of the best epoch: 3.
20/20 ━━━━━━━━━━━━━━━━━━━━ 84s 4s/step - accuracy: 0.8733 - loss: 0.3820
========================================
Test Set Accuracy - 90.54%
Test Set Loss - 0.2689
========================================
20/20 ━━━━━━━━━━━━━━━━━━━━ 95s 5s/step
              precision    recall  f1-score   support

   PNEUMONIA       0.96      0.47      0.63       390
      NORMAL       0.52      0.97      0.68       234

    accuracy                           0.66       624
   macro avg       0.74      0.72      0.66       624
weighted avg       0.80      0.66      0.65       624
```

Figure 29: Output of the prediction metrics on the test set

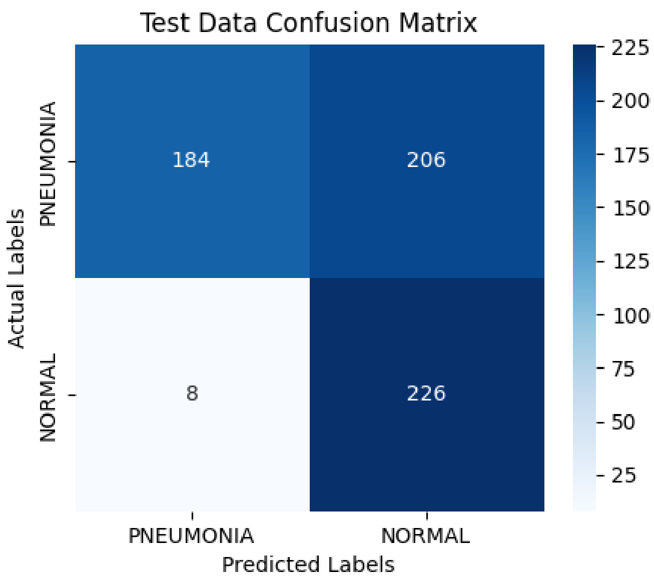

Figure 30: Output, confusion matrix for test data

```python
# Validation Performance
val_evaluation = model.evaluate(val_generator, verbose=1)
print("==" * 20)
print(f"Validation Set Accuracy - {val_evaluation[1] * 100:.2f}%")
print(f"Validation Set Loss - {val_evaluation[0]:.4f}")
print("==" * 20)

# Predictions and Confusion Matrix for Validation Data
val_predictions = (model.predict(val_generator) >= 0.5).astype(int).reshape(-1)
val_cm = confusion_matrix(y_val, val_predictions)

plt.figure(figsize=(6, 5))
sns.heatmap(val_cm, annot=True, fmt="d", cmap="Blues", xticklabels=LABELS, yticklabels=LABELS)
plt.xlabel("Predicted Labels")
plt.ylabel("Actual Labels")
plt.title("Validation Data Confusion Matrix")
plt.show()

print(classification_report(y_val, val_predictions, target_names=LABELS))
```

Figure 31: Output of the prediction metrics on the validation set

Insights for the validation set are also obtained.

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step - accuracy: 0.8750 - loss: 0.2526
========================================
Validation Set Accuracy - 87.50%
Validation Set Loss - 0.2526
========================================
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
              precision    recall  f1-score   support

   PNEUMONIA       0.89      1.00      0.94         8
      NORMAL       1.00      0.88      0.93         8

    accuracy                           0.94        16
   macro avg       0.94      0.94      0.94        16
weighted avg       0.94      0.94      0.94        16
```

Figure 32: Output of the prediction metrics on the validation set and overall information

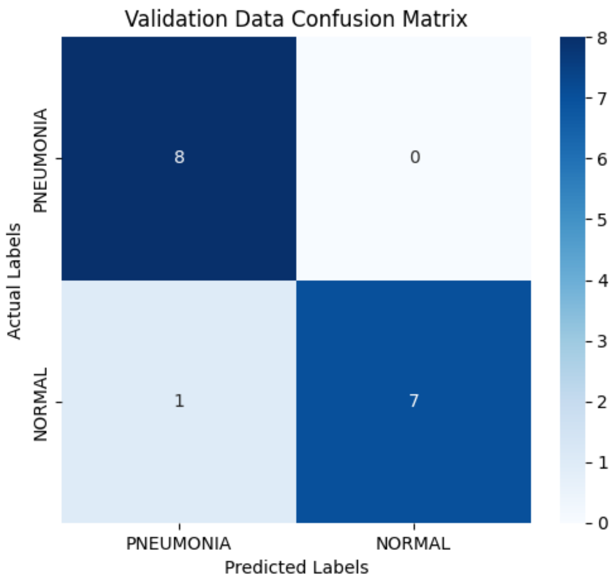

Figure 33: Output, confusion matrix validation data

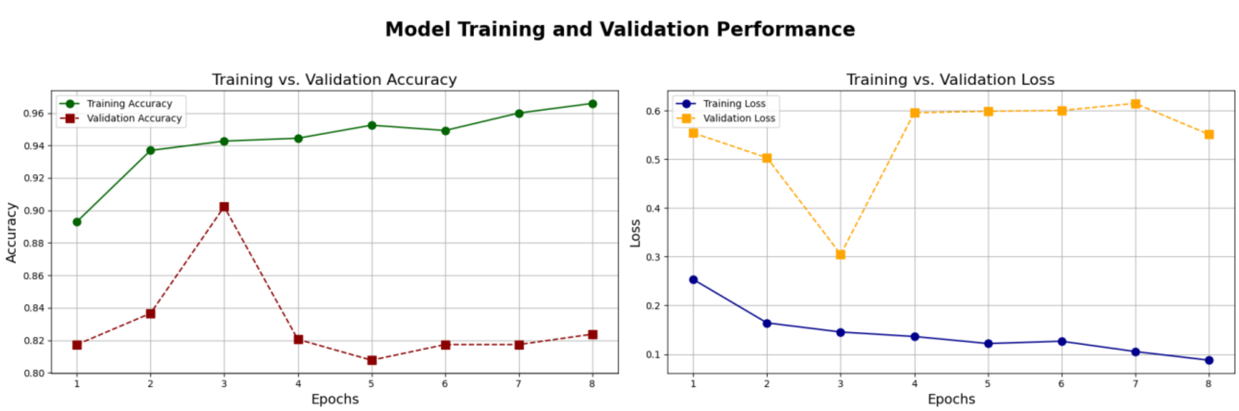

Figure 34: Training vs. test and validation sets accuracy and loss function value graph

__Predictions__

```
def display_sample_images(X,y,predictions,title,cmap):  2 usages
    num_samples = 12  # Display 12 samples
    random_indices = np.random.choice(len(X),num_samples,replace=False)

    plt.figure(figsize=(12,6))
    for i,idx in enumerate(random_indices):
        plt.subplot( *args: 3,4,i + 1)

        # Convert the image to grayscale
        gray_image = np.dot(X[idx][...,:3], b: [0.2989,0.5870,0.1140])  # Convert RGB to grayscale

        # Apply the colormap
        colored_image = plt.cm.get_cmap(cmap)(gray_image)  # Get the colormap and apply it
        colored_image = (colored_image[...,:3] * 255).astype(np.uint8)  # Convert back to uint8 format

        # Display the image with the specified colormap
        plt.imshow(colored_image,interpolation='none')  # Display the colored image

        # Set the title with predicted and actual classes
        plt.title( label: f"Predicted: {predictions[idx]}   Actual: {y[idx]}",fontsize=10)

        # Remove x and y ticks
        plt.axis('off')

    # Set the main title for the figure
    plt.suptitle(title,size=18)

    # Adjust layout to prevent overlapping
    plt.tight_layout()

    # Show the plot
    plt.show()


# Display sample images for validation set with viridis colormap
display_sample_images(X_val,y_val,val_predictions, title: "Validation Set Predictions",cmap='viridis')

# Display sample images for test set with magma colormap
display_sample_images(X_test,y_test,test_predictions, title: "Test Set Predictions",cmap='magma')
```

Figure 35: Visualizing the sample images with their predictions code snippet

In a separate code block, the sample visuals of the saved model are shown to ease the main code. This block collects the information about prediction and the label to make a comparison for the output. For control reasons, pictures turned into grayscale to be colored later. The test set is colored with a perceptually continuous color scale of purple-pink, magma whereas the validation images are with a blue-yellow scale, viridis.

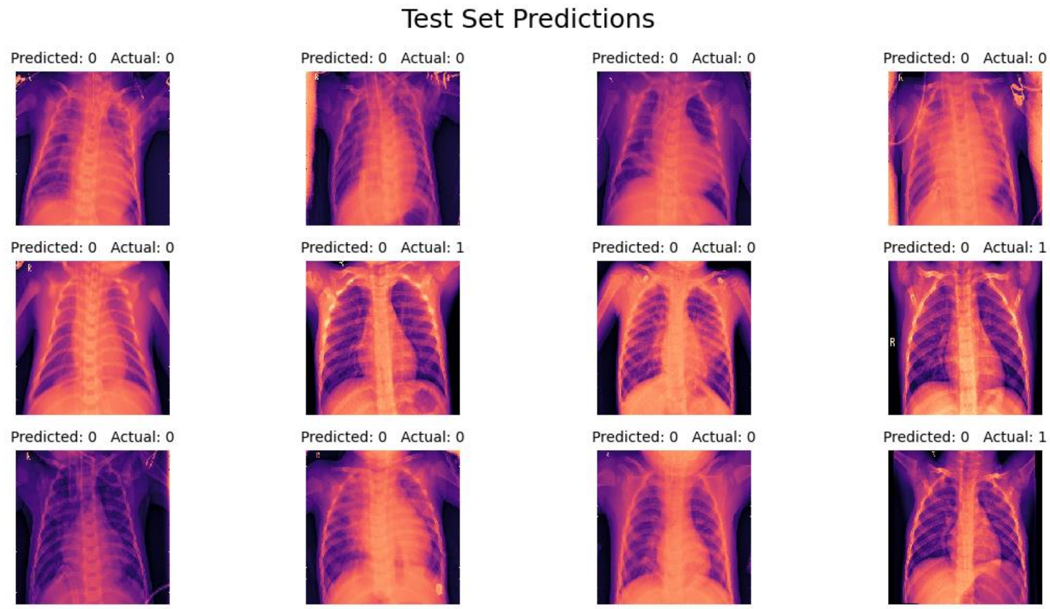

Figure 36: Test samples with their predictions

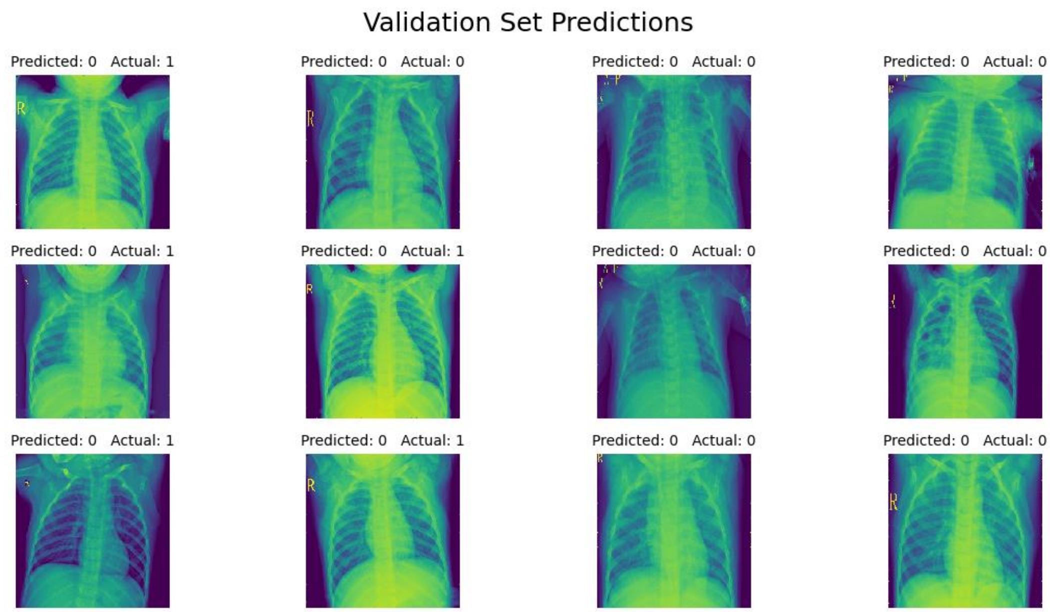

Figure 37: Validation samples with their predictions

CHAPTER THREE: Model Evaluation

	precision	recall	f1-score	support
PNEUMONIA	0.89	1.00	0.94	8
NORMAL	1.00	0.88	0.93	8
accuracy			0.94	16
macro avg	0.94	0.94	0.94	16
weighted avg	0.94	0.94	0.94	16

Figure 38: Model evaluation metrics output

The 94% accuracy gives us the strong ability of the ResNet50 model. Accuracy stands for the correctly classified cases over the total number of cases that exist. However, the test accuracy's decrease to 66% shows us the need for improval on new data. Accuracy provides us with the performance of the application, in our case the image classification of the x-ray images, but we also need to check the balance of the categories. Therefore, we also need to check for precision, the ratio of true positives over predicted positives to find out how many of our pneumonia images are actually the ill patients' x-rays. We expected to have high values for precision because false positives can lead to time, money, and most importantly health consumption for the patient. Illness precision is high (0.89), but this is not enough for our model to serve for health industry. Since medical fields need more accuracy in image classification on new data, we need to investigate the other factors before any further changes.

On the other hand, we also need to check for the true predictions of whether it is pneumonia or a normal image over actual pneumonia patients. This metric is called recall (also known as sensitivity) and shows us the power of a well-designed model. This gives us the cost of a misdiagnosed case if the patient is actually ill. Under this context, our recall value for pneumonia is 1, but also the cost of our prediction on the healthy individuals with 88%. If there is a class imbalance in our dataset, it would be more precise to look at the harmonic mean of these two metrics. To decrease overdiagnosis and undertreatment, F1-score gives us a robust evaluation of our model with over 93% for each class. It is safe to say that our model is suitable for the market, but we need more data to enhance our scores.

CONCLUDING REMARKS

To sum up, our project intended to show the potential of the deep learning algorithms' demonstration for CNN architectures, especially the ResNet50 model for diagnosing the possibility of lung pneumonia detection on X-ray images. This disease is a globally threatening one that can be treated faster if an early diagnosis is made without losing the immune system support of the patients (Kallander et al., 2016). This helps the patients to spend less time, and money, and have healthier life standards. Our project demonstrates the feasibility and validation of practical applications with improved training processes.

Feature extraction, residual connections, and increased layering help the ResNet50 model to recognize underlying patterns of pneumonia that affected lungs. The nature of the model is accurate and yet versatile enough for different qualities of images with transformations. We processed the data, augmented the images, and the model optimization for differentiations in human physiology, to be consistent and reduce the noise with an artificially enlarged dataset.

The model's accuracy is 94%, which is a good enough level to invest more resources on, whereas the validation accuracy (66%) shows us the need for improvement. Our generalization was not enough for the actual market release of the prject. The distribution of the demographics of the patients can be extended to remove any biases for predictions accuracy increase. In our context, both false positives and true negatives are costly predictions that need to be eliminated so as not to cause fatalities. The high precision gives us the result of high correctness for predicting an actual pneumonia patient. That reduces the unnecessary treatments on false positives. Whereas a high recall rate indicates the identification of the model with true cases for illness, which is crucial not to miss any diagnosis. High recall rates play a great role for emergency cases to reduce the complications for the patients. Another indicator for our performance is F1-score which is balanced in our evaluation and ends up with a guarantee for not overdiagnosing healthy people or overlooking instances excessively. The model shows promise for real-world use by attaining a high F1 score, which guarantees that true positive and true negative rates are kept within a reasonable range.

Clinical decision-making requires transparent and understandable models. While ResNet50's layer-by-layer learning process offers some interpretability, further research is needed to enhance with visualizing the specific X-ray regions that influence the model's diagnosis. Clinicians can gain deeper insights into the decision-making process. Similar architecture can be adopted in other fields of radiographical imagery diagnosis. For instance, tumors, fractures, or other lung diseases

Another context that needs to be underlined is the data privacy and ethical regulations system needs for AI usage. This shows us the great regulatory standardizing effort, the clinical usages with auditing, and feedbacking can help both data science fields as well as the healthcare ecosystem. The need for broad data sources and additional pathological examples for a more balanced dataset with medical-specified augmentation generalizations and well-tuned layers can enhance our model as well as it can be used for transfer learning for other models.

BIBLIOGRAPHY

Alzubaidi, L. *et al.* (2021) 'Review of Deep Learning: Concepts, CNN Architectures, challenges, applications, Future Directions', *Journal of Big Data*, 8(1). doi:10.1186/s40537-021-00444-8.

Association, A.L. (2024) *Learn about pneumonia*, *American Lung Association*. Available at: https://www.lung.org/lung-health-diseases/lung-disease-lookup/pneumonia/learn-about-pneumonia (Accessed: 31 October 2024).

Esteva, A., Kuprel, B., Novoa, R. A., Ko, J., Swetter, S. M., Blau, H. M., & Thrun, S. (2017). Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639), 115-118.

Gulshan, V., Peng, L., Coram, M., Stumpe, M. C., Wu, D., Narayanaswamy, A., ... & Kim, R. (2016). Development and validation of a deep learning algorithm for the detection of diabetic retinopathy in retinal fundus photographs. *Jama*, 316(22), 2402-2410.

Harshit, P. (2024) *chest X-ray (pneumonia) diagnosis with CNN(90%)*, *Kaggle*. Available at: https://www.kaggle.com/code/harshitpathak18/chest-x-ray-pneumonia-diagnosis-with-cnn-90?scriptVersionId=197144181 (Accessed: 31 October 2024).

Kallander, K., Burgess, D.H. and Qazi, S.A. (2016) 'Early identification and treatment of pneumonia: A call to action', *The Lancet Global Health*, 4(1). doi:10.1016/s2214-109x(15)00272-7.

Kermany, D. (2018) *Labeled optical coherence tomography (OCT) and chest X-ray images for classification*, *Mendeley Data*. Available at: https://data.mendeley.com/datasets/rscbjbr9sj/2 (Accessed: 30 October 2024).

Kermany, D.S. *et al.* (2018) 'Identifying medical diagnoses and treatable diseases by image-based Deep Learning', *Cell*, 172(5). doi:10.1016/j.cell.2018.02.010.

Litjens, G., Kooi, T., Bejnordi, B. E., Setio, A. A. A., Ciompi, F., Ghafoorian, M., ... & Sánchez, C. I. (2017). A survey on deep learning in medical image analysis. Medical Image Analysis, 42, 60-88.

Liu, L., Amor, A. and Xiao, Y. (2024) *User guide*, *scikit*. Available at: https://scikit-learn.org/stable/user_guide.html (Accessed: 30 October 2024).

MATLAB (2021) *Introducing deep learning with MATLAB, MATLAB & Simulink*. Available at: https://www.mathworks.com/campaigns/offers/deep-learning-with-matlab.html (Accessed: 01 November 2024).

Mishra, M. (2020) *Convolutional Neural Networks, explained*, *Medium*. Available at: https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939 (Accessed: 01 November 2024).

Mooney, P. (2018) *Chest X-ray images (pneumonia)*, *Kaggle*. Available at: https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia (Accessed: 30 October 2024).

Rajpurkar, P., Irvin, J., Zhu, K., Yang, B., Mehta, H., Duan, T., ... & Ng, A. Y. (2017). Chexnet: Radiologist-level pneumonia detection on chest X-rays with deep learning. *arXiv preprint arXiv:1711.05225*.

Wang, X., Peng, Y., Lu, L., Lu, Z., Bagheri, M., & Summers, R. M. (2017). Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 2097-2106).

Table of Figures

APPENDIX

Codes and information helped to produce my work:

https://matplotlib.org/stable/users/explain/colors/colormaps.html

https://www.kaggle.com/code/somendrew/chest-x-ray-pneumonia-91-3-accuracy

https://www.kaggle.com/code/harshitpathak18/chest-x-ray-pneumonia-diagnosis-with-cnn-90?scriptVersionId=197144181

https://seaborn.pydata.org/tutorial/color_palettes.html

My github link:

https://github.com/3irsari/Computer-Vision_Image-Segmentation

model.summary() output figures in order:

Model: "functional"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 256, 256, 3)	0	-
conv1_pad (ZeroPadding2D)	(None, 262, 262, 3)	0	input_layer[0][0]
conv1_conv (Conv2D)	(None, 128, 128, 64)	9,472	conv1_pad[0][0]
conv1_bn (BatchNormalizatio…	(None, 128, 128, 64)	256	conv1_conv[0][0]
conv1_relu (Activation)	(None, 128, 128, 64)	0	conv1_bn[0][0]
pool1_pad (ZeroPadding2D)	(None, 130, 130, 64)	0	conv1_relu[0][0]
pool1_pool (MaxPooling2D)	(None, 64, 64, 64)	0	pool1_pad[0][0]
conv2_block1_1_conv (Conv2D)	(None, 64, 64, 64)	4,160	pool1_pool[0][0]
conv2_block1_1_bn (BatchNormalizatio…	(None, 64, 64, 64)	256	conv2_block1_1_c…
conv2_block1_1_relu (Activation)	(None, 64, 64, 64)	0	conv2_block1_1_b…
conv2_block1_2_conv (Conv2D)	(None, 64, 64, 64)	36,928	conv2_block1_1_r…
conv2_block1_2_bn (BatchNormalizatio…	(None, 64, 64, 64)	256	conv2_block1_2_c…
conv2_block1_2_relu (Activation)	(None, 64, 64, 64)	0	conv2_block1_2_b…

conv2_block1_0_conv (Conv2D)	(None, 64, 64, 256)	16,640	pool1_pool[0][0]
conv2_block1_3_conv (Conv2D)	(None, 64, 64, 256)	16,640	conv2_block1_2_r…
conv2_block1_0_bn (BatchNormalizatio…	(None, 64, 64, 256)	1,024	conv2_block1_0_c…
conv2_block1_3_bn (BatchNormalizatio…	(None, 64, 64, 256)	1,024	conv2_block1_3_c…
conv2_block1_add (Add)	(None, 64, 64, 256)	0	conv2_block1_0_b… conv2_block1_3_b…
conv2_block1_out (Activation)	(None, 64, 64, 256)	0	conv2_block1_add…
conv2_block2_1_conv (Conv2D)	(None, 64, 64, 64)	16,448	conv2_block1_out…
conv2_block2_1_bn (BatchNormalizatio…	(None, 64, 64, 64)	256	conv2_block2_1_c…
conv2_block2_1_relu (Activation)	(None, 64, 64, 64)	0	conv2_block2_1_b…
conv2_block2_2_conv (Conv2D)	(None, 64, 64, 64)	36,928	conv2_block2_1_r…
conv2_block2_2_bn (BatchNormalizatio…	(None, 64, 64, 64)	256	conv2_block2_2_c…
conv2_block2_2_relu (Activation)	(None, 64, 64, 64)	0	conv2_block2_2_b…
conv2_block2_3_conv (Conv2D)	(None, 64, 64, 256)	16,640	conv2_block2_2_r…
conv2_block2_3_bn (BatchNormalizatio…	(None, 64, 64, 256)	1,024	conv2_block2_3_c…
conv2_block2_add (Add)	(None, 64, 64, 256)	0	conv2_block1_out… conv2_block2_3_b…
conv2_block2_out (Activation)	(None, 64, 64, 256)	0	conv2_block2_add…
conv2_block3_1_conv (Conv2D)	(None, 64, 64, 64)	16,448	conv2_block2_out…
conv2_block3_1_bn (BatchNormalizatio…	(None, 64, 64, 64)	256	conv2_block3_1_c…
conv2_block3_1_relu (Activation)	(None, 64, 64, 64)	0	conv2_block3_1_b…
conv2_block3_2_conv (Conv2D)	(None, 64, 64, 64)	36,928	conv2_block3_1_r…
conv2_block3_2_bn (BatchNormalizatio…	(None, 64, 64, 64)	256	conv2_block3_2_c…
conv2_block3_2_relu (Activation)	(None, 64, 64, 64)	0	conv2_block3_2_b…
conv2_block3_3_conv (Conv2D)	(None, 64, 64, 256)	16,640	conv2_block3_2_r…
conv2_block3_3_bn (BatchNormalizatio…	(None, 64, 64, 256)	1,024	conv2_block3_3_c…
conv2_block3_add (Add)	(None, 64, 64, 256)	0	conv2_block2_out… conv2_block3_3_b…
conv2_block3_out (Activation)	(None, 64, 64, 256)	0	conv2_block3_add…
conv3_block1_1_conv (Conv2D)	(None, 32, 32, 128)	32,896	conv2_block3_out…
conv3_block1_1_bn (BatchNormalizatio…	(None, 32, 32, 128)	512	conv3_block1_1_c…

conv3_block1_1_bn (BatchNormalizatio…	(None, 32, 32, 128)	512	conv3_block1_1_c…
conv3_block1_1_relu (Activation)	(None, 32, 32, 128)	0	conv3_block1_1_b…
conv3_block1_2_conv (Conv2D)	(None, 32, 32, 128)	147,584	conv3_block1_1_r…
conv3_block1_2_bn (BatchNormalizatio…	(None, 32, 32, 128)	512	conv3_block1_2_c…
conv3_block1_2_relu (Activation)	(None, 32, 32, 128)	0	conv3_block1_2_b…
conv3_block1_0_conv (Conv2D)	(None, 32, 32, 512)	131,584	conv2_block3_out…
conv3_block1_3_conv (Conv2D)	(None, 32, 32, 512)	66,048	conv3_block1_2_r…
conv3_block1_0_bn (BatchNormalizatio…	(None, 32, 32, 512)	2,048	conv3_block1_0_c…
conv3_block1_3_bn (BatchNormalizatio…	(None, 32, 32, 512)	2,048	conv3_block1_3_c…
conv3_block1_add (Add)	(None, 32, 32, 512)	0	conv3_block1_0_b… conv3_block1_3_b…
conv3_block1_out (Activation)	(None, 32, 32, 512)	0	conv3_block1_add…
conv3_block2_1_conv (Conv2D)	(None, 32, 32, 128)	65,664	conv3_block1_out…
conv3_block2_1_bn (BatchNormalizatio…	(None, 32, 32, 128)	512	conv3_block2_1_c…
conv3_block2_1_relu (Activation)	(None, 32, 32, 128)	0	conv3_block2_1_b…

conv3_block2_2_conv (Conv2D)	(None, 32, 32, 128)	147,584	conv3_block2_1_r…
conv3_block2_2_bn (BatchNormalizatio…	(None, 32, 32, 128)	512	conv3_block2_2_c…
conv3_block2_2_relu (Activation)	(None, 32, 32, 128)	0	conv3_block2_2_b…
conv3_block2_3_conv (Conv2D)	(None, 32, 32, 512)	66,048	conv3_block2_2_r…
conv3_block2_3_bn (BatchNormalizatio…	(None, 32, 32, 512)	2,048	conv3_block2_3_c…
conv3_block2_add (Add)	(None, 32, 32, 512)	0	conv3_block1_out… conv3_block2_3_b…
conv3_block2_out (Activation)	(None, 32, 32, 512)	0	conv3_block2_add…
conv3_block3_1_conv (Conv2D)	(None, 32, 32, 128)	65,664	conv3_block2_out…
conv3_block3_1_bn (BatchNormalizatio…	(None, 32, 32, 128)	512	conv3_block3_1_c…
conv3_block3_1_relu (Activation)	(None, 32, 32, 128)	0	conv3_block3_1_b…
conv3_block3_2_conv (Conv2D)	(None, 32, 32, 128)	147,584	conv3_block3_1_r…
conv3_block3_2_bn (BatchNormalizatio…	(None, 32, 32, 128)	512	conv3_block3_2_c…
conv3_block3_2_relu (Activation)	(None, 32, 32, 128)	0	conv3_block3_2_b…
conv3_block3_3_conv (Conv2D)	(None, 32, 32, 512)	66,048	conv3_block3_2_r…

conv3_block3_3_bn (BatchNormalizatio…	(None, 32, 32, 512)	2,048	conv3_block3_3_c…
conv3_block3_add (Add)	(None, 32, 32, 512)	0	conv3_block2_out… conv3_block3_3_b…
conv3_block3_out (Activation)	(None, 32, 32, 512)	0	conv3_block3_add…
conv3_block4_1_conv (Conv2D)	(None, 32, 32, 128)	65,664	conv3_block3_out…
conv3_block4_1_bn (BatchNormalizatio…	(None, 32, 32, 128)	512	conv3_block4_1_c…
conv3_block4_1_relu (Activation)	(None, 32, 32, 128)	0	conv3_block4_1_b…
conv3_block4_2_conv (Conv2D)	(None, 32, 32, 128)	147,584	conv3_block4_1_r…
conv3_block4_2_bn (BatchNormalizatio…	(None, 32, 32, 128)	512	conv3_block4_2_c…
conv3_block4_2_relu (Activation)	(None, 32, 32, 128)	0	conv3_block4_2_b…
conv3_block4_3_conv (Conv2D)	(None, 32, 32, 512)	66,048	conv3_block4_2_r…
conv3_block4_3_bn (BatchNormalizatio…	(None, 32, 32, 512)	2,048	conv3_block4_3_c…
conv3_block4_add (Add)	(None, 32, 32, 512)	0	conv3_block3_out… conv3_block4_3_b…
conv3_block4_out (Activation)	(None, 32, 32, 512)	0	conv3_block4_add…
conv4_block1_1_conv (Conv2D)	(None, 16, 16, 256)	131,328	conv3_block4_out…

conv4_block1_1_bn (BatchNormalizatio…	(None, 16, 16, 256)	1,024	conv4_block1_1_c…
conv4_block1_1_relu (Activation)	(None, 16, 16, 256)	0	conv4_block1_1_b…
conv4_block1_2_conv (Conv2D)	(None, 16, 16, 256)	590,080	conv4_block1_1_r…
conv4_block1_2_bn (BatchNormalizatio…	(None, 16, 16, 256)	1,024	conv4_block1_2_c…
conv4_block1_2_relu (Activation)	(None, 16, 16, 256)	0	conv4_block1_2_b…
conv4_block1_0_conv (Conv2D)	(None, 16, 16, 1024)	525,312	conv3_block4_out…
conv4_block1_3_conv (Conv2D)	(None, 16, 16, 1024)	263,168	conv4_block1_2_r…
conv4_block1_0_bn (BatchNormalizatio…	(None, 16, 16, 1024)	4,096	conv4_block1_0_c…
conv4_block1_3_bn (BatchNormalizatio…	(None, 16, 16, 1024)	4,096	conv4_block1_3_c…
conv4_block1_add (Add)	(None, 16, 16, 1024)	0	conv4_block1_0_b… conv4_block1_3_b…
conv4_block1_out (Activation)	(None, 16, 16, 1024)	0	conv4_block1_add…
conv4_block2_1_conv (Conv2D)	(None, 16, 16, 256)	262,400	conv4_block1_out…
conv4_block2_1_bn (BatchNormalizatio…	(None, 16, 16, 256)	1,024	conv4_block2_1_c…
conv4_block2_1_relu (Activation)	(None, 16, 16, 256)	0	conv4_block2_1_b…

conv4_block2_2_conv (Conv2D)	(None, 16, 16, 256)	590,080	conv4_block2_1_r…
conv4_block2_2_bn (BatchNormalizatio…	(None, 16, 16, 256)	1,024	conv4_block2_2_c…
conv4_block2_2_relu (Activation)	(None, 16, 16, 256)	0	conv4_block2_2_b…
conv4_block2_3_conv (Conv2D)	(None, 16, 16, 1024)	263,168	conv4_block2_2_r…
conv4_block2_3_bn (BatchNormalizatio…	(None, 16, 16, 1024)	4,096	conv4_block2_3_c…
conv4_block2_add (Add)	(None, 16, 16, 1024)	0	conv4_block1_out… conv4_block2_3_b…
conv4_block2_out (Activation)	(None, 16, 16, 1024)	0	conv4_block2_add…
conv4_block3_1_conv (Conv2D)	(None, 16, 16, 256)	262,400	conv4_block2_out…
conv4_block3_1_bn (BatchNormalizatio…	(None, 16, 16, 256)	1,024	conv4_block3_1_c…
conv4_block3_1_relu (Activation)	(None, 16, 16, 256)	0	conv4_block3_1_b…
conv4_block3_2_conv (Conv2D)	(None, 16, 16, 256)	590,080	conv4_block3_1_r…
conv4_block3_2_bn (BatchNormalizatio…	(None, 16, 16, 256)	1,024	conv4_block3_2_c…
conv4_block3_2_relu (Activation)	(None, 16, 16, 256)	0	conv4_block3_2_b…
conv4_block3_3_conv (Conv2D)	(None, 16, 16, 1024)	263,168	conv4_block3_2_r…

conv4_block3_3_bn (BatchNormalizatio…	(None, 16, 16, 1024)	4,096	conv4_block3_3_c…
conv4_block3_add (Add)	(None, 16, 16, 1024)	0	conv4_block2_out… conv4_block3_3_b…
conv4_block3_out (Activation)	(None, 16, 16, 1024)	0	conv4_block3_add…
conv4_block4_1_conv (Conv2D)	(None, 16, 16, 256)	262,400	conv4_block3_out…
conv4_block4_1_bn (BatchNormalizatio…	(None, 16, 16, 256)	1,024	conv4_block4_1_c…
conv4_block4_1_relu (Activation)	(None, 16, 16, 256)	0	conv4_block4_1_b…
conv4_block4_2_conv (Conv2D)	(None, 16, 16, 256)	590,080	conv4_block4_1_r…
conv4_block4_2_bn (BatchNormalizatio…	(None, 16, 16, 256)	1,024	conv4_block4_2_c…
conv4_block4_2_relu (Activation)	(None, 16, 16, 256)	0	conv4_block4_2_b…
conv4_block4_3_conv (Conv2D)	(None, 16, 16, 1024)	263,168	conv4_block4_2_r…
conv4_block4_3_bn (BatchNormalizatio…	(None, 16, 16, 1024)	4,096	conv4_block4_3_c…
conv4_block4_add (Add)	(None, 16, 16, 1024)	0	conv4_block3_out… conv4_block4_3_b…
conv4_block4_out (Activation)	(None, 16, 16, 1024)	0	conv4_block4_add…
conv4_block5_1_conv (Conv2D)	(None, 16, 16, 256)	262,400	conv4_block4_out…

conv4_block5_1_bn (BatchNormalizatio…	(None, 16, 16, 256)	1,024	conv4_block5_1_c…
conv4_block5_1_relu (Activation)	(None, 16, 16, 256)	0	conv4_block5_1_b…
conv4_block5_2_conv (Conv2D)	(None, 16, 16, 256)	590,080	conv4_block5_1_r…
conv4_block5_2_bn (BatchNormalizatio…	(None, 16, 16, 256)	1,024	conv4_block5_2_c…
conv4_block5_2_relu (Activation)	(None, 16, 16, 256)	0	conv4_block5_2_b…
conv4_block5_3_conv (Conv2D)	(None, 16, 16, 1024)	263,168	conv4_block5_2_r…
conv4_block5_3_bn (BatchNormalizatio…	(None, 16, 16, 1024)	4,096	conv4_block5_3_c…
conv4_block5_add (Add)	(None, 16, 16, 1024)	0	conv4_block4_out… conv4_block5_3_b…
conv4_block5_out (Activation)	(None, 16, 16, 1024)	0	conv4_block5_add…
conv4_block6_1_conv (Conv2D)	(None, 16, 16, 256)	262,400	conv4_block5_out…
conv4_block6_1_bn (BatchNormalizatio…	(None, 16, 16, 256)	1,024	conv4_block6_1_c…
conv4_block6_1_relu (Activation)	(None, 16, 16, 256)	0	conv4_block6_1_b…
conv4_block6_2_conv (Conv2D)	(None, 16, 16, 256)	590,080	conv4_block6_1_r…
conv4_block6_2_bn (BatchNormalizatio…	(None, 16, 16, 256)	1,024	conv4_block6_2_c…
conv4_block6_2_relu (Activation)	(None, 16, 16, 256)	0	conv4_block6_2_b…
conv4_block6_3_conv (Conv2D)	(None, 16, 16, 1024)	263,168	conv4_block6_2_r…
conv4_block6_3_bn (BatchNormalizatio…	(None, 16, 16, 1024)	4,096	conv4_block6_3_c…
conv4_block6_add (Add)	(None, 16, 16, 1024)	0	conv4_block5_out… conv4_block6_3_b…
conv4_block6_out (Activation)	(None, 16, 16, 1024)	0	conv4_block6_add…
conv5_block1_1_conv (Conv2D)	(None, 8, 8, 512)	524,800	conv4_block6_out…
conv5_block1_1_bn (BatchNormalizatio…	(None, 8, 8, 512)	2,048	conv5_block1_1_c…
conv5_block1_1_relu (Activation)	(None, 8, 8, 512)	0	conv5_block1_1_b…
conv5_block1_2_conv (Conv2D)	(None, 8, 8, 512)	2,359,808	conv5_block1_1_r…
conv5_block1_2_bn (BatchNormalizatio…	(None, 8, 8, 512)	2,048	conv5_block1_2_c…
conv5_block1_2_relu (Activation)	(None, 8, 8, 512)	0	conv5_block1_2_b…
conv5_block1_0_conv (Conv2D)	(None, 8, 8, 2048)	2,099,200	conv4_block6_out…
conv5_block1_3_conv (Conv2D)	(None, 8, 8, 2048)	1,050,624	conv5_block1_2_r…
conv5_block1_0_bn (BatchNormalizatio…	(None, 8, 8, 2048)	8,192	conv5_block1_0_c…

(BatchNormalizatio…	2048)		
conv5_block1_3_bn (BatchNormalizatio…	(None, 8, 8, 2048)	8,192	conv5_block1_3_c…
conv5_block1_add (Add)	(None, 8, 8, 2048)	0	conv5_block1_0_b… conv5_block1_3_b…
conv5_block1_out (Activation)	(None, 8, 8, 2048)	0	conv5_block1_add…
conv5_block2_1_conv (Conv2D)	(None, 8, 8, 512)	1,049,088	conv5_block1_out…
conv5_block2_1_bn (BatchNormalizatio…	(None, 8, 8, 512)	2,048	conv5_block2_1_c…
conv5_block2_1_relu (Activation)	(None, 8, 8, 512)	0	conv5_block2_1_b…
conv5_block2_2_conv (Conv2D)	(None, 8, 8, 512)	2,359,808	conv5_block2_1_r…
conv5_block2_2_bn (BatchNormalizatio…	(None, 8, 8, 512)	2,048	conv5_block2_2_c…
conv5_block2_2_relu (Activation)	(None, 8, 8, 512)	0	conv5_block2_2_b…
conv5_block2_3_conv (Conv2D)	(None, 8, 8, 2048)	1,050,624	conv5_block2_2_r…
conv5_block2_3_bn (BatchNormalizatio…	(None, 8, 8, 2048)	8,192	conv5_block2_3_c…
conv5_block2_add (Add)	(None, 8, 8, 2048)	0	conv5_block1_out… conv5_block2_3_b…
conv5_block2_out (Activation)	(None, 8, 8, 2048)	0	conv5_block2_add…
conv5_block3_1_conv (Conv2D)	(None, 8, 8, 512)	1,049,088	conv5_block2_out…
conv5_block3_1_bn (BatchNormalizatio…	(None, 8, 8, 512)	2,048	conv5_block3_1_c…
conv5_block3_1_relu (Activation)	(None, 8, 8, 512)	0	conv5_block3_1_b…
conv5_block3_2_conv (Conv2D)	(None, 8, 8, 512)	2,359,808	conv5_block3_1_r…
conv5_block3_2_bn (BatchNormalizatio…	(None, 8, 8, 512)	2,048	conv5_block3_2_c…
conv5_block3_2_relu (Activation)	(None, 8, 8, 512)	0	conv5_block3_2_b…
conv5_block3_3_conv (Conv2D)	(None, 8, 8, 2048)	1,050,624	conv5_block3_2_r…
conv5_block3_3_bn (BatchNormalizatio…	(None, 8, 8, 2048)	8,192	conv5_block3_3_c…
conv5_block3_add (Add)	(None, 8, 8, 2048)	0	conv5_block2_out… conv5_block3_3_b…
conv5_block3_out (Activation)	(None, 8, 8, 2048)	0	conv5_block3_add…
global_average_poo… (GlobalAveragePool…	(None, 2048)	0	conv5_block3_out…
dense (Dense)	(None, 128)	262,272	global_average_p…
dropout (Dropout)	(None, 128)	0	dense[0][0]
dense_1 (Dense)	(None, 1)	129	dropout[0][0]

```
 Total params: 23,850,113 (90.98 MB)
 Trainable params: 262,401 (1.00 MB)
 Non-trainable params: 23,587,712 (89.98 MB)
Found 5216 images belonging to 2 classes.
Found 16 images belonging to 2 classes.
Found 624 images belonging to 2 classes.
```