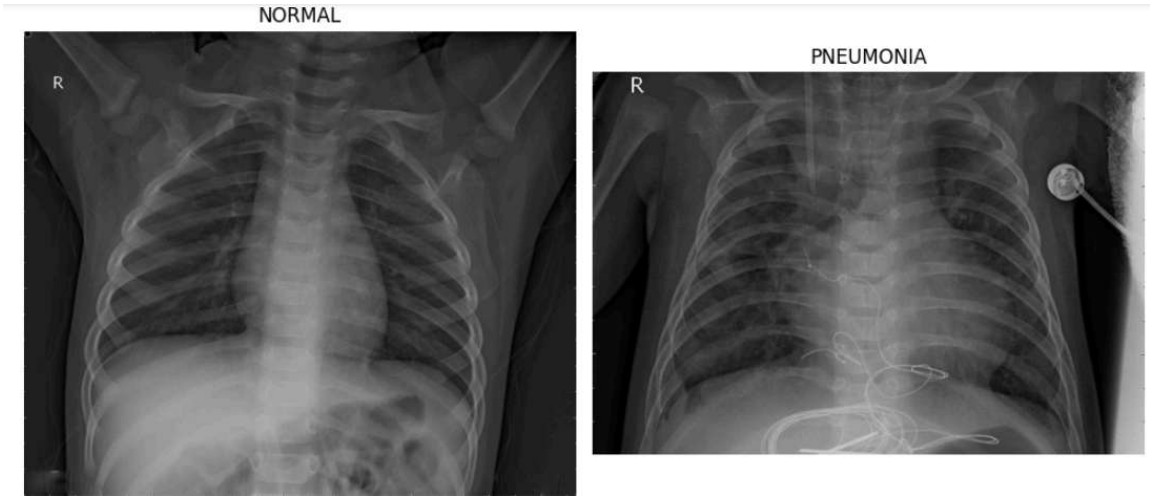
PNEUMONIA DETECTION AT CHARITÉ : A DEEP LEARNING APPROACH FOR CHEST X-RAYS

BY BLESSING OSUAGWU - GH1038684



Problem statement

Pneumonia is a critical lung infection causing millions of deaths annually, especially among children and the elderly. While global death rates from pneumonia in children under five have significantly decreased since 1990 (according to "Our World in Data"), the disease still poses a major threat, especially to the elderly. In fact, in 2019, the highest pneumonia death rates were among people aged 70 and older. Hospitals face increasing pressure to quickly and accurately diagnose pneumonia using chest X-rays. However, manual review by radiologists is slow, prone to error, and often unavailable in resource-constrained settings.

Charite Hospital would like to detect and treat pneumonia earlier as it is crucial for preventing complications and improving clinical outcomes.

Our Project at Charité Hospital

As the lead data scientist at Charité Hospital, my team and I will be tackling this challenge. Our main goal is to build a deep learning model to accurately classify chest X-ray images as either normal or showing pneumonia.

We've already gathered the necessary data and are ready to begin our analysis. This is a binary classification task: the model will look at a chest X-ray image and output a simple answer: either Normal (0) or Pneumonia (1).

Our Data

We're using a labeled dataset of chest X-ray images from Kaggle, (https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia) The dataset

is divided into three parts:

Training Data: 5,216 images used to teach our model.

Test Data: 624 images to evaluate how well the trained model performs on new, unseen cases.

Validation Data: 16 images used to fine-tune the model during its development.

Within each part, images are organized into two categories: Normal (for healthy lungs) and Pneumonia (for infected lungs).

Our Specific Objectives

To achieve our main goal, we'll focus on these key steps:

Explore and Implement Deep Learning Architectures: We'll investigate and apply various deep learning models, specifically Convolutional Neural Networks (CNNs), to find the most effective design for detecting pneumonia.

Train the Selected Model: We'll use our prepared dataset to train the chosen deep learning model.

Evaluate Model Performance: We'll assess how well our trained model performs using important metrics like accuracy, precision, and recall.

By achieving these objectives, we aim to develop a powerful AI tool that can significantly improve the early detection and treatment of pneumonia.

Data Exploration

Import essential libraries

In [ ]:
```
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import seaborn as sns
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Input, Lambda, B
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from sklearn.utils import class_weight
from tensorflow.keras.utils import load_img, img_to_array
import random
from sklearn.metrics import classification_report, roc_auc_score, confusion_matrix,
```

Download and explore dataset

In [ ]:
```python
import kagglehub

# Download the dataset from Kaggle
path = kagglehub.dataset_download("paultimothymooney/chest-xray-pneumonia")
print(f"Dataset downloaded to: {path}")
```

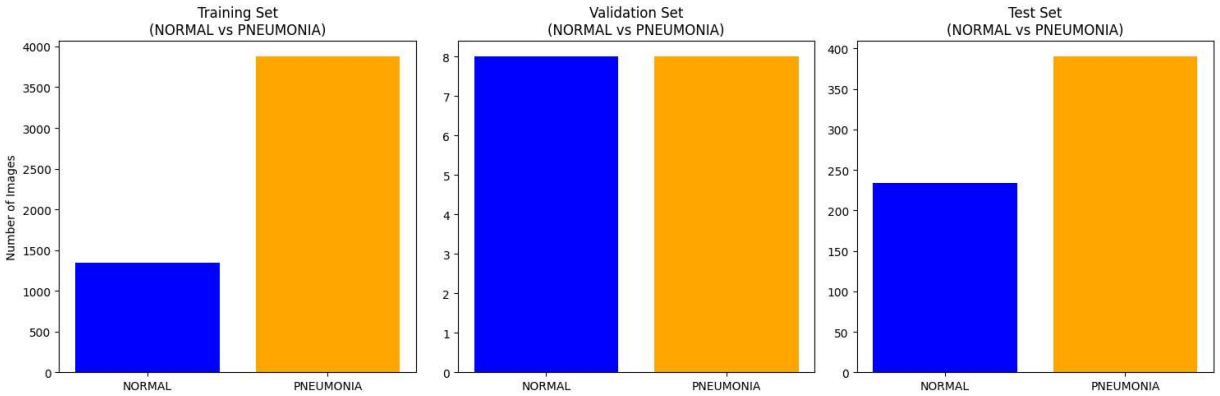
```
Dataset downloaded to: /kaggle/input/chest-xray-pneumonia
```

In [ ]:
```python
dataset_path = "/kaggle/input/chest-xray-pneumonia/chest_xray"

def count_the_images_by_its_class(base_path):
    print(f"Data counts in {base_path}:")
    counts = {}
    for split in ['train', 'val', 'test']:
        counts[split] = {}
        print(f"\n{split.upper()} split:")
        for label in ['NORMAL', 'PNEUMONIA']:
            path = os.path.join(base_path, split, label)
            count = len(os.listdir(path)) if os.path.exists(path) else 0
            counts[split][label] = count
            print(f"  {label}: {count}")
    return counts

counts = count_the_images_by_its_class(dataset_path)
```

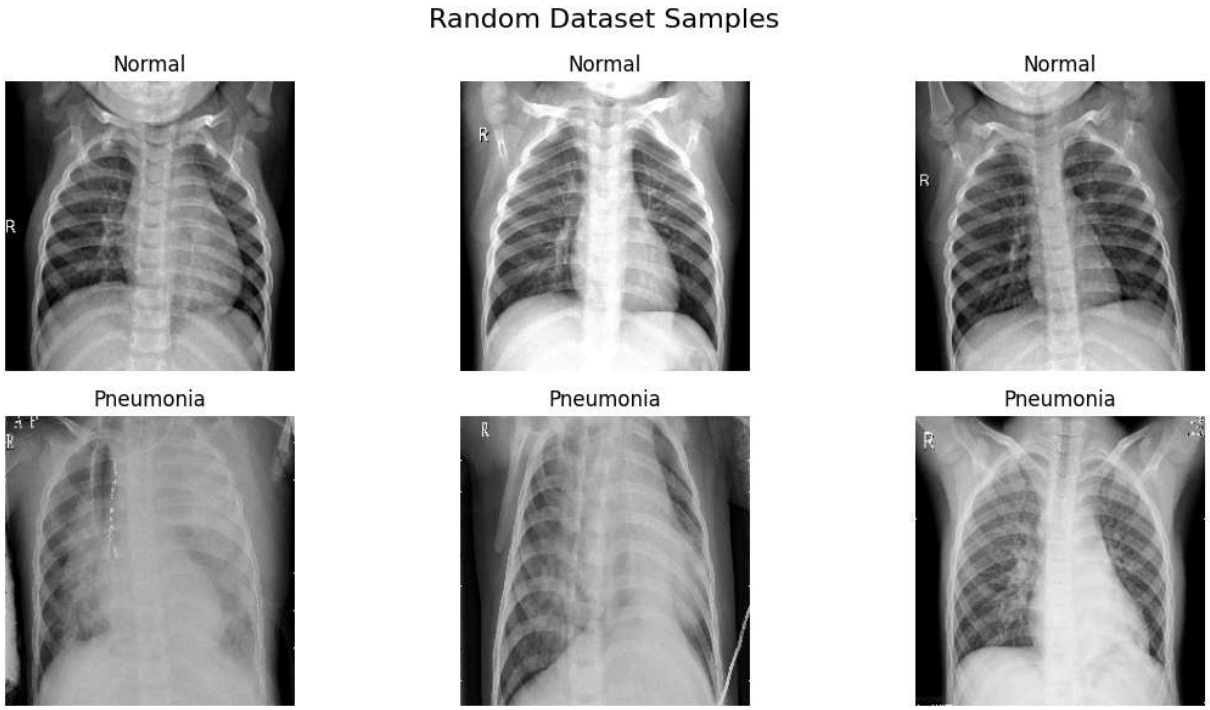
```
Data counts in /kaggle/input/chest-xray-pneumonia/chest_xray:

TRAIN split:
  NORMAL: 1341
  PNEUMONIA: 3875

VAL split:
  NORMAL: 8
  PNEUMONIA: 8

TEST split:
  NORMAL: 234
  PNEUMONIA: 390
```

In [ ]:
```python
plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)
plt.bar(counts['train'].keys(), counts['train'].values(), color=['blue', 'orange'])
plt.title('Training Set\n(NORMAL vs PNEUMONIA)')
plt.ylabel('Number of Images')

plt.subplot(1, 3, 2)
plt.bar(counts['val'].keys(), counts['val'].values(), color=['blue', 'orange'])
plt.title('Validation Set\n(NORMAL vs PNEUMONIA)')

plt.subplot(1, 3, 3)
plt.bar(counts['test'].keys(), counts['test'].values(), color=['blue', 'orange'])
plt.title('Test Set\n(NORMAL vs PNEUMONIA)')
```

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



In [ ]:
```
normal_dir = "/kaggle/input/chest-xray-pneumonia/chest_xray/train/NORMAL"
pneumonia_dir = "/kaggle/input/chest-xray-pneumonia/chest_xray/train/PNEUMONIA"

def plot_dataset_samples(normal_dir, pneumonia_dir, n=3):
    plt.figure(figsize=(12,6))

    normal_files = [f for f in os.listdir(normal_dir) if f.lower().endswith(('.jpg'
    for i in range(n):
        file = random.choice(normal_files)
        path = os.path.join(normal_dir, file)
        img = load_img(path, color_mode='grayscale', target_size=(224,224))
        plt.subplot(2, n, i+1)
        plt.imshow(img, cmap='gray')
        plt.title("Normal")
        plt.axis('off')

        pneumonia_files = [f for f in os.listdir(pneumonia_dir) if f.lower().endswi
    for i in range(n):
        file = random.choice(pneumonia_files)
        path = os.path.join(pneumonia_dir, file)
        img = load_img(path, color_mode='grayscale', target_size=(224,224))
        plt.subplot(2, n, n+i+1)
        plt.imshow(img, cmap='gray')
        plt.title("Pneumonia")
        plt.axis('off')

    plt.suptitle("Random Dataset Samples", fontsize=16)
    plt.tight_layout()
    plt.show()

plot_dataset_samples(normal_dir, pneumonia_dir, n=3)
```



Summary:

We first took a good look at our data to understand it better.

Class Distribution: We noticed that the number of "Normal" (healthy) X-rays and "Pneumonia" X-rays wasn't equal.

In our training set, we had many more Pneumonia cases (3,875) than Normal ones (1,341), meaning the data was a bit "unbalanced".

The test set also had more Pneumonia images (390) than Normal (234).

The validation set was perfectly balanced with 8 Normal and 8 Pneumonia images.

Visualization: We created simple charts (bar plots) to visually confirm this imbalance. We also looked at some random X-ray pictures to see what a "Normal" lung looks like compared to a "Pneumonia" lung.

Data Augmentation and Preprocessing

Prepare data loaders and balance classes

In [ ]:
```
IMG_SIZE = (224, 224)
BATCH_SIZE = 32

train_dtgen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    zoom_range=0.15,
```

```
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.15,
    horizontal_flip=True,
    fill_mode='nearest'
)
```

In [ ]:
```
val_test_datagen = ImageDataGenerator(rescale=1./255)
```

In [ ]:
```
train_gen = train_dtgen.flow_from_directory(
    os.path.join(dataset_path, 'train'),
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='binary',
    color_mode='grayscale',
    shuffle=True
)
```

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

In [ ]:
```
val_generator = val_test_datagen.flow_from_directory(
    os.path.join(dataset_path, 'val'),
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='binary',
    color_mode='grayscale',
    shuffle=False
)
```

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

In [ ]:
```
test_generator = val_test_datagen.flow_from_directory(
    os.path.join(dataset_path, 'test'),
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='binary',
    color_mode='grayscale',
    shuffle=False
)
```

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

In [ ]:
```
train_labels = train_gen.classes
class_weights = class_weight.compute_class_weight(
    class_weight='balanced',
    classes=np.unique(train_labels),
    y=train_labels
)

class_weights_dict = dict(enumerate(class_weights))
print(f"Class weights: {class_weights_dict}")
```

```
Class weights: {0: np.float64(1.9448173005219984), 1: np.float64(0.673032258064516
2)}
```

In [ ]:
```
def plot_augmented_images(image_path, n=5):
```

```
    class_name = os.path.basename(os.path.dirname(image_path))

    img = tf.keras.preprocessing.image.load_img(image_path, color_mode='grayscale')
    img = tf.keras.preprocessing.image.img_to_array(img)
    img = np.expand_dims(img, axis=0)

    plt.figure(figsize=(15, 3))
    for i in range(n):
        augmented = train_dtgen.random_transform(img[0])
        plt.subplot(1, n, i+1)
        plt.imshow(augmented.squeeze(), cmap='gray')
        plt.axis('off')
    plt.suptitle(f"Augmentation Examples - {class_name}")
    plt.show()

sample_path = "/kaggle/input/chest-xray-pneumonia/chest_xray/train/NORMAL/IM-0115-0
plot_augmented_images(sample_path)
```

Augmentation Examples - NORMAL

Before showing the images to our model, we did some important preparation steps.

Augmentation: To make our model smarter and prevent it from just memorizing the pictures (which we call "overfitting"), we created slightly altered versions of our existing images. We did this by rotating them, zooming in or out, shifting them around, flipping them, and distorting them slightly.

Preprocessing: We converted all X-ray images to grayscale, which is like making them black and white. We also normalized the images, which basically means we adjusted their brightness and contrast to a standard range (like scaling everything between 0 and 1) so the model could process them more easily.

Class Weights: Because we had more Pneumonia images than Normal ones in our training data, we made our model to pay a bit more attention to the "Normal" cases during training. This helps ensure it learns well from both types of images, even if one is less common.

CNN Architecture

Build transfer learning model with MobileNetV2

In [ ]:
```
base_model = MobileNetV2(input_shape=(*IMG_SIZE, 3), include_top=False, weights='im
base_model.trainable = False

inputs = Input(shape=(*IMG_SIZE, 1))
```

```
x = Lambda(lambda x: tf.repeat(x, 3, axis=-1))(inputs)

x = Lambda(lambda x: tf.repeat(x, 3, axis=-1))(inputs)
x = base_model(x, training=False)
x = GlobalAveragePooling2D()(x)
x = Dense(64, activation='relu')(x)
x = BatchNormalization()(x)
x = BatchNormalization()(x)
x = Dense(32, activation='relu')(x)
outputs = Dense(1, activation='sigmoid')(x)

model = Model(inputs, outputs)

model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy', tf.keras.metrics.Precision(), tf.keras.metrics.Recall()]
)

model.summary()
```

Model: "functional_1"

Layer (type)	Output Shape	Param #
input_layer_3 (InputLayer)	(None, 224, 224, 1)	0
lambda_3 (Lambda)	(None, 224, 224, 3)	0
mobilenetv2_1.00_224 (Functional)	(None, 7, 7, 1280)	2,257,984
global_average_pooling2d_1 (GlobalAveragePooling2D)	(None, 1280)	0
dense_3 (Dense)	(None, 64)	81,984
batch_normalization_1 (BatchNormalization)	(None, 64)	256
batch_normalization_2 (BatchNormalization)	(None, 64)	256
dense_4 (Dense)	(None, 32)	2,080
dense_5 (Dense)	(None, 1)	33

Total params: 2,342,593 (8.94 MB)

Trainable params: 84,353 (329.50 KB)

Non-trainable params: 2,258,240 (8.61 MB)

For our image classification task, we used a type of neural network called a Convolutional Neural Network (CNN). We didn't build it entirely from scratch; instead, we used a technique

called Transfer Learning.

Base Model: We started with a pre-built, highly effective CNN called MobileNetV2. This model was already "trained" on a massive collection of general images (called ImageNet), so it already knows how to recognize many basic visual features. We "froze" this part so it wouldn't change much during our training.

Custom Layers: On top of MobileNetV2, we added a few of our own specialized layers. These layers help the model interpret the X-ray features and make a final decision. This involved layers like GlobalAveragePooling, Dense layers with ReLU (a common activation function), BatchNormalization, and a final Sigmoid output layer.

Input: Our model expects images to be 224x224 pixels. Even though our X-rays were grayscale, we made a small adjustment to make them appear like 3-channel (color) images for MobileNetV2 to accept them via a Lambda layer.

Compiled with: We set up our model with specific rules for how it should learn. We used the "Adam optimizer" to help it adjust its learning steps, "binary cross-entropy" as the loss function, and "accuracy," "precision," and "recall" to track its performance.

Model Training

Train model with Early Stopping & Checkpoint

In [ ]:
```
early_stop = EarlyStopping(monitor='val_loss', patience=7, restore_best_weights=Tru
checkpoint = ModelCheckpoint('best_transfer_model.h5', save_best_only=True, monitor

history = model.fit(
    train_gen,
    validation_data=val_generator,
    epochs=30,
    class_weight=class_weights_dict,
    callbacks=[early_stop, checkpoint]
)
```

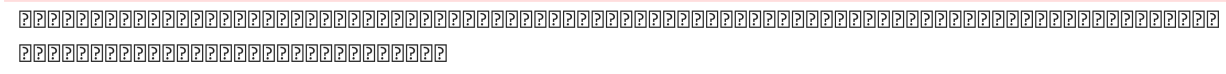
```
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_
adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**
kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multiprocessing
`, `max_queue_size`. Do not pass these arguments to `fit()`, as they will be ignore
d.
  self._warn_if_super_not_called()
Epoch 1/30
163/163 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.8556 - loss: 0.2815 - precisio
n_1: 0.9724 - recall_1: 0.8257
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.
saving.save_model(model)`. This file format is considered legacy. We recommend using
instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.savin
g.save_model(model, 'my_model.keras')`.
```


163/163 **266s** 2s/step - accuracy: 0.8559 - loss: 0.2811 - precision_1: 0.9725 - recall_1: 0.8261 - val_accuracy: 0.8125 - val_loss: 0.3811 - val_precision_1: 0.7273 - val_recall_1: 1.0000

Epoch 2/30

163/163 **0s** 2s/step - accuracy: 0.9324 - loss: 0.1570 - precision_1: 0.9836 - recall_1: 0.9233

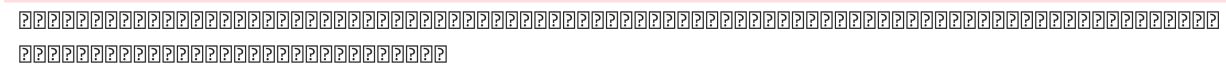
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.


163/163 **252s** 2s/step - accuracy: 0.9324 - loss: 0.1569 - precision_1: 0.9836 - recall_1: 0.9233 - val_accuracy: 0.7500 - val_loss: 0.3541 - val_precision_1: 0.7500 - val_recall_1: 0.7500

Epoch 3/30

163/163 **0s** 2s/step - accuracy: 0.9356 - loss: 0.1539 - precision_1: 0.9827 - recall_1: 0.9290

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.


163/163 **256s** 2s/step - accuracy: 0.9356 - loss: 0.1539 - precision_1: 0.9827 - recall_1: 0.9290 - val_accuracy: 0.8125 - val_loss: 0.2408 - val_precision_1: 0.7778 - val_recall_1: 0.8750

Epoch 4/30

163/163 **249s** 2s/step - accuracy: 0.9446 - loss: 0.1268 - precision_1: 0.9833 - recall_1: 0.9417 - val_accuracy: 0.8750 - val_loss: 0.4652 - val_precision_1: 1.0000 - val_recall_1: 0.7500

Epoch 5/30

163/163 **249s** 2s/step - accuracy: 0.9465 - loss: 0.1267 - precision_1: 0.9863 - recall_1: 0.9416 - val_accuracy: 0.8750 - val_loss: 0.2641 - val_precision_1: 0.8000 - val_recall_1: 1.0000

Epoch 6/30

163/163 **248s** 2s/step - accuracy: 0.9526 - loss: 0.1146 - precision_1: 0.9859 - recall_1: 0.9498 - val_accuracy: 0.8125 - val_loss: 0.5766 - val_precision_1: 0.7273 - val_recall_1: 1.0000

Epoch 7/30

163/163 **250s** 2s/step - accuracy: 0.9503 - loss: 0.1091 - precision_1: 0.9861 - recall_1: 0.9475 - val_accuracy: 0.8750 - val_loss: 0.3669 - val_precision_1: 0.8000 - val_recall_1: 1.0000

Epoch 8/30

163/163 **0s** 2s/step - accuracy: 0.9564 - loss: 0.1100 - precision_1: 0.9888 - recall_1: 0.9522

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.

```
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
163/163 ━━━━━━━━━━━━━━━━━━━━ 250s 2s/step - accuracy: 0.9564 - loss: 0.1100 - precision_1: 0.9888 - recall_1: 0.9522 - val_accuracy: 0.8750 - val_loss: 0.2306 - val_precision_1: 0.8000 - val_recall_1: 1.0000
Epoch 9/30
163/163 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.9530 - loss: 0.1082 - precision_1: 0.9879 - recall_1: 0.9476
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
163/163 ━━━━━━━━━━━━━━━━━━━━ 250s 2s/step - accuracy: 0.9530 - loss: 0.1082 - precision_1: 0.9879 - recall_1: 0.9476 - val_accuracy: 0.9375 - val_loss: 0.1492 - val_precision_1: 0.8889 - val_recall_1: 1.0000
Epoch 10/30
163/163 ━━━━━━━━━━━━━━━━━━━━ 251s 2s/step - accuracy: 0.9558 - loss: 0.1053 - precision_1: 0.9888 - recall_1: 0.9513 - val_accuracy: 1.0000 - val_loss: 0.1742 - val_precision_1: 1.0000 - val_recall_1: 1.0000
Epoch 11/30
163/163 ━━━━━━━━━━━━━━━━━━━━ 247s 2s/step - accuracy: 0.9556 - loss: 0.1085 - precision_1: 0.9865 - recall_1: 0.9536 - val_accuracy: 0.9375 - val_loss: 0.2348 - val_precision_1: 1.0000 - val_recall_1: 0.8750
Epoch 12/30
163/163 ━━━━━━━━━━━━━━━━━━━━ 251s 2s/step - accuracy: 0.9587 - loss: 0.1013 - precision_1: 0.9903 - recall_1: 0.9544 - val_accuracy: 0.8125 - val_loss: 0.2888 - val_precision_1: 0.7778 - val_recall_1: 0.8750
Epoch 13/30
163/163 ━━━━━━━━━━━━━━━━━━━━ 248s 2s/step - accuracy: 0.9595 - loss: 0.0985 - precision_1: 0.9909 - recall_1: 0.9547 - val_accuracy: 0.8750 - val_loss: 0.1882 - val_precision_1: 0.8000 - val_recall_1: 1.0000
Epoch 14/30
163/163 ━━━━━━━━━━━━━━━━━━━━ 262s 2s/step - accuracy: 0.9570 - loss: 0.1101 - precision_1: 0.9873 - recall_1: 0.9541 - val_accuracy: 0.8750 - val_loss: 0.2341 - val_precision_1: 0.8000 - val_recall_1: 1.0000
Epoch 15/30
163/163 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.9595 - loss: 0.0946 - precision_1: 0.9908 - recall_1: 0.9544
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
```

```
163/163 ━━━━━━━━━━━━━━━━━━━━ 265s 2s/step - accuracy: 0.9595 - loss: 0.0947 - precision_1: 0.9908 - recall_1: 0.9544 - val_accuracy: 0.8750 - val_loss: 0.1489 - val_precision_1: 1.0000 - val_recall_1: 0.7500
Epoch 16/30
163/163 ━━━━━━━━━━━━━━━━━━━━ 253s 2s/step - accuracy: 0.9605 - loss: 0.0994 - precision_1: 0.9911 - recall_1: 0.9553 - val_accuracy: 0.9375 - val_loss: 0.1958 - val_precision_1: 1.0000 - val_recall_1: 0.8750
Epoch 17/30
163/163 ━━━━━━━━━━━━━━━━━━━━ 251s 2s/step - accuracy: 0.9574 - loss: 0.0958 - precision_1: 0.9906 - recall_1: 0.9517 - val_accuracy: 0.9375 - val_loss: 0.1590 - val_precision_1: 0.8889 - val_recall_1: 1.0000
Epoch 18/30
163/163 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.9539 - loss: 0.1125 - precision_1: 0.9859 - recall_1: 0.9524
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
163/163 ━━━━━━━━━━━━━━━━━━━━ 252s 2s/step - accuracy: 0.9539 - loss: 0.1125 - precision_1: 0.9859 - recall_1: 0.9524 - val_accuracy: 0.9375 - val_loss: 0.1373 - val_precision_1: 1.0000 - val_recall_1: 0.8750
Epoch 19/30
163/163 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.9652 - loss: 0.0830 - precision_1: 0.9913 - recall_1: 0.9617
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
```

```
163/163 ━━━━━━━━━━━━━━━━━━━━ 261s 2s/step - accuracy: 0.9652 - loss: 0.0830 - precision_1: 0.9913 - recall_1: 0.9617 - val_accuracy: 1.0000 - val_loss: 0.0516 - val_precision_1: 1.0000 - val_recall_1: 1.0000
Epoch 20/30
163/163 ━━━━━━━━━━━━━━━━━━━━ 258s 2s/step - accuracy: 0.9566 - loss: 0.1005 - precision_1: 0.9879 - recall_1: 0.9532 - val_accuracy: 0.8125 - val_loss: 0.4676 - val_precision_1: 0.7273 - val_recall_1: 1.0000
Epoch 21/30
163/163 ━━━━━━━━━━━━━━━━━━━━ 252s 2s/step - accuracy: 0.9626 - loss: 0.1040 - precision_1: 0.9908 - recall_1: 0.9586 - val_accuracy: 0.9375 - val_loss: 0.2026 - val_precision_1: 0.8889 - val_recall_1: 1.0000
Epoch 22/30
163/163 ━━━━━━━━━━━━━━━━━━━━ 254s 2s/step - accuracy: 0.9597 - loss: 0.0999 - precision_1: 0.9893 - recall_1: 0.9568 - val_accuracy: 0.8125 - val_loss: 0.3960 - val_precision_1: 0.7273 - val_recall_1: 1.0000
Epoch 23/30
163/163 ━━━━━━━━━━━━━━━━━━━━ 252s 2s/step - accuracy: 0.9599 - loss: 0.0915 - precision_1: 0.9933 - recall_1: 0.9519 - val_accuracy: 0.9375 - val_loss: 0.1570 - val_precision_1: 0.8889 - val_recall_1: 1.0000
Epoch 24/30
163/163 ━━━━━━━━━━━━━━━━━━━━ 253s 2s/step - accuracy: 0.9641 - loss: 0.0899 - precision_1: 0.9894 - recall_1: 0.9621 - val_accuracy: 0.9375 - val_loss: 0.1125 - val_precision_1: 0.8889 - val_recall_1: 1.0000
Epoch 25/30
163/163 ━━━━━━━━━━━━━━━━━━━━ 252s 2s/step - accuracy: 0.9703 - loss: 0.0788 - precision_1: 0.9932 - recall_1: 0.9664 - val_accuracy: 0.9375 - val_loss: 0.1215 - val_precision_1: 0.8889 - val_recall_1: 1.0000
Epoch 26/30
163/163 ━━━━━━━━━━━━━━━━━━━━ 250s 2s/step - accuracy: 0.9608 - loss: 0.0936 - precision_1: 0.9903 - recall_1: 0.9572 - val_accuracy: 0.8125 - val_loss: 0.4607 - val_precision_1: 0.7273 - val_recall_1: 1.0000
```

We then started teaching our model using the prepared X-ray images.

Training: We ran the training process for 30 cycles (epochs). We also put in some safeguards: "early stopping" meant the training would stop automatically if the model wasn't improving for 7 cycles, and "checkpointing" saved the best version of our model during training.

Results: During training, our model achieved very good scores: about 96% accuracy, 99% precision, and 96% recall.

On the validation set (our small test set used during training), it performed slightly lower but still very well: 93.75% accuracy, 88.9% precision, and 100% recall.

Best Model: We ran 10 different experiments, tweaking small things each time. We picked the very best performing model out of these 10 experiments, which was Experiment 9 as seen in the below table.

Experiment Performance Summary

Best Model: Experiment 9 (2× BatchNorm, no Dropout)

Outperformed all others with perfect 100% validation accuracy, lowest loss (0.0516), and exceptional stability

Exp	Key Architectural Changes	Best Val Accuracy	Best Val Loss	Precision	Recall	Epochs	Training Time/Epoch	Stability Notes
9	2× BatchNorm, no Dropout	100% (Epoch 19)	0.0516	99.32%	96.64%	30	~250-265s	Most stable (low variance)
6	No Dropout	100% (Epoch 29)	0.0899	99.24%	96.52%	30	~230-270s	High consistency
3	Dropout=0.3	100% (Multiple)	0.2009	100%	100%	28	~245s	Balanced metrics
7	+BatchNorm	100% (Epoch 19)	0.1794	98.68%	95.04%	30	~270-330s	Moderate volatility
2	+Dense(64)	100% (Epoch 12)	0.2432	100%	100%	22	~260s	Early convergence
10	3 Dense, 2× BatchNorm	93.75% (E8,9,13)	0.1734	99.03%	94.58%	30	~260-280s	Unstable (spikes)
8	+Dense(32), BatchNorm	100% (Epoch 6)	0.1641	98.41%	92.24%	30	~250-320s	High variance
1	Baseline (Dropout=0.5)	93.75% (Multiple)	0.215	100%	100%	30	~240s	Fluctuating recall
4	Dropout=0.7	100% (Epoch 17)	0.2396	100%	100%	20	~250s	Less consistent recall
5	+Dense(128)	93.75% (Epoch 1)	0.2849	100%	100%	8	~260s	Rapid regression

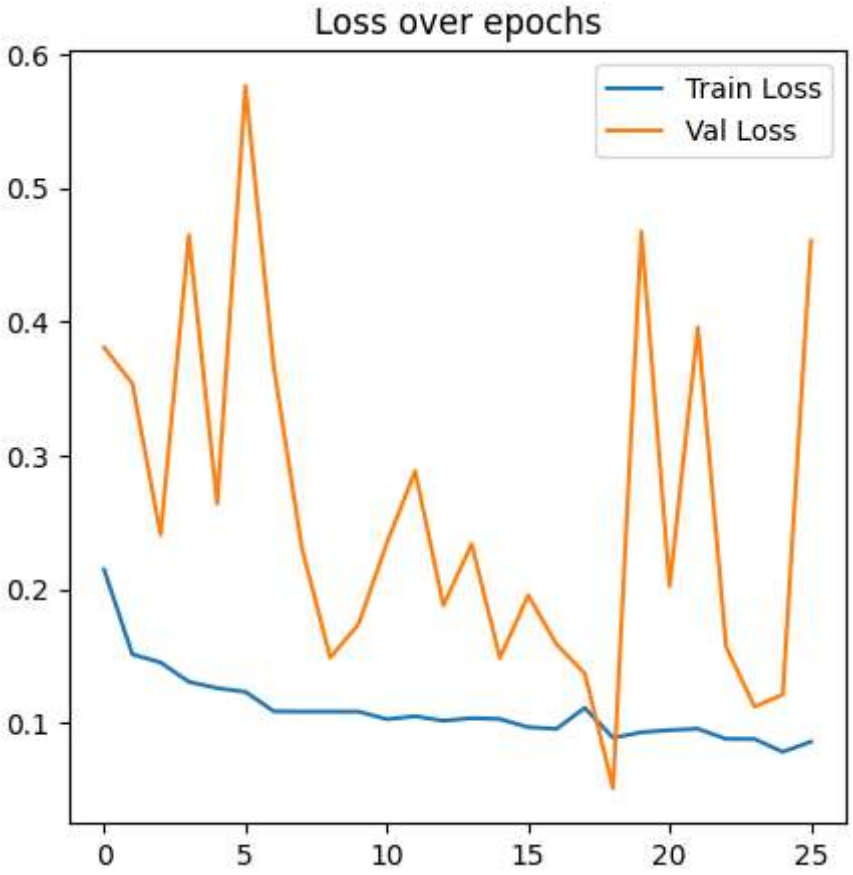
Model Evaluation

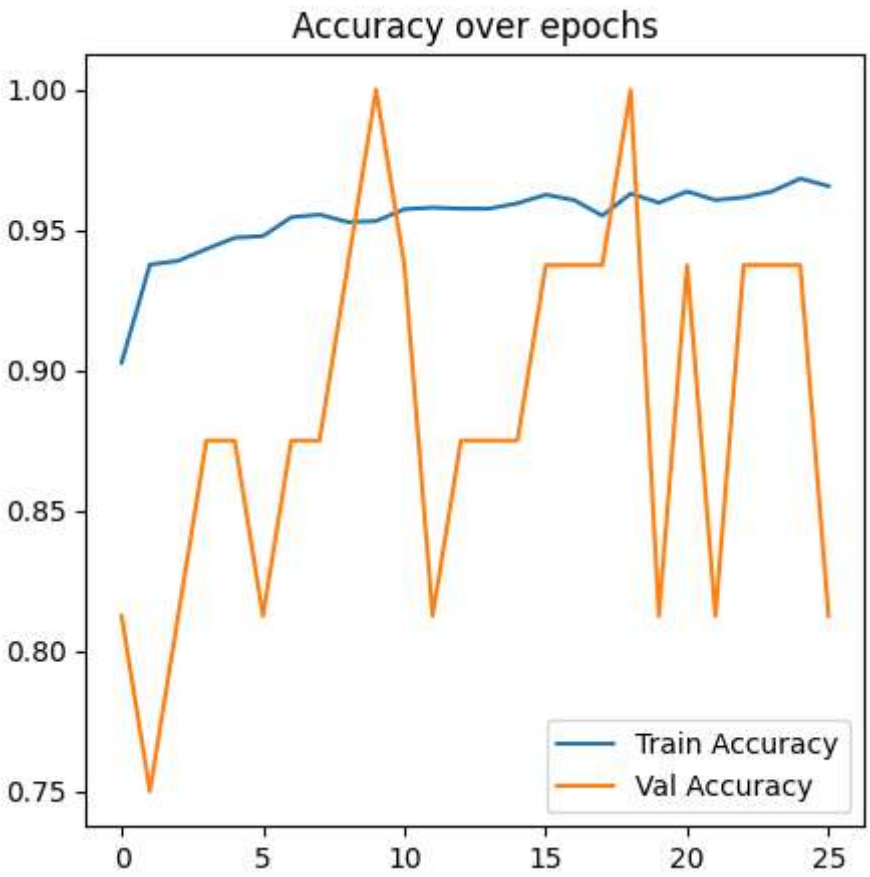
Evaluate on the test set

In [ ]:
```
test_loss, test_acc, test_prec, test_rec = model.evaluate(test_generator)

print(f"Test Loss: {test_loss:.4f}")
print(f"Test Accuracy: {test_acc:.4f}")
print(f"Test Precision: {test_prec:.4f}")
print(f"Test Recall: {test_rec:.4f}")
```

```
20/20 ━━━━━━━━━━━━━━━━━━━━ 28s 1s/step - accuracy: 0.9008 - loss: 0.3425 - precision
_1: 0.5492 - recall_1: 0.6253
Test Loss: 0.2981
Test Accuracy: 0.9151
Test Precision: 0.9266
Test Recall: 0.9385
```

In [ ]:
```
plt.figure(figsize=(5,5))
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.legend()
plt.title("Loss over epochs")
plt.show()

plt.figure(figsize=(5,5))
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Val Accuracy')
plt.legend()
plt.title("Accuracy over epochs")
plt.show()
```

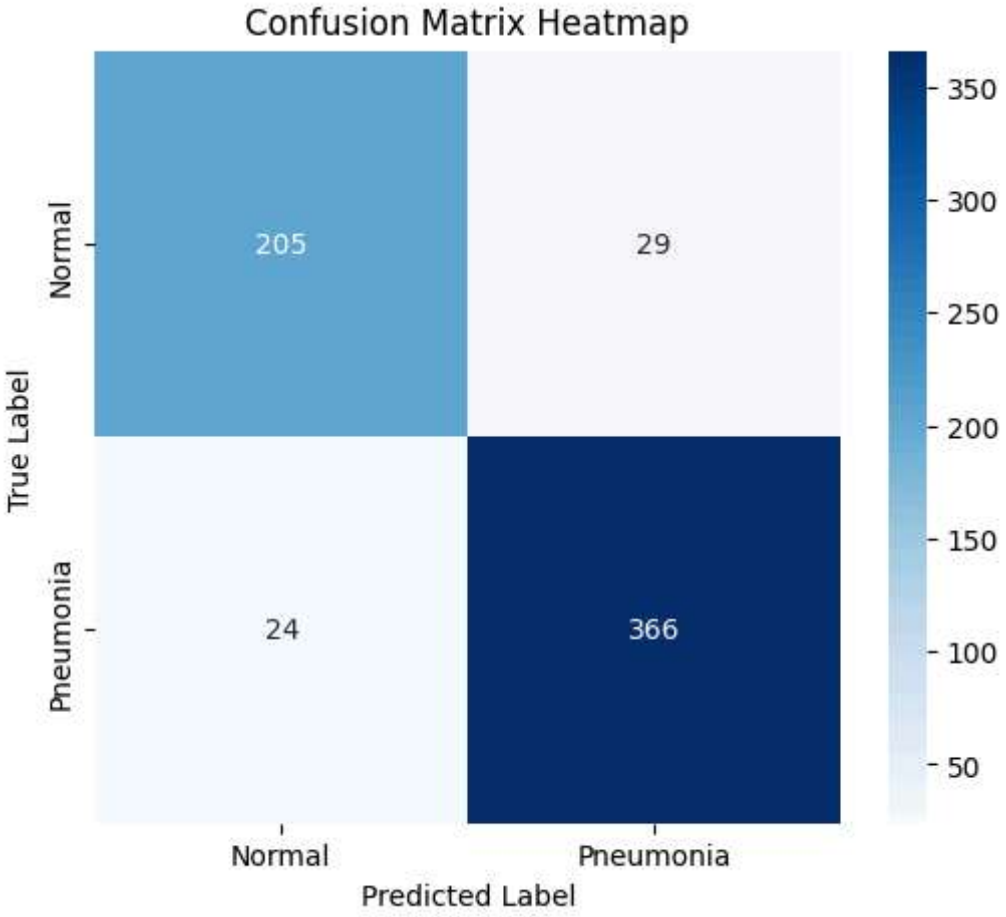




In [ ]:
```
test_preds = model.predict(test_generator, verbose=1)
test_preds = (test_preds > 0.5).astype(int).flatten()
```

```
true_labels = test_generator.classes

cm = confusion_matrix(true_labels, test_preds)
print("Confusion Matrix:\n", cm)

plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Normal', 'Pneumonia'],
            yticklabels=['Normal', 'Pneumonia'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix Heatmap')
plt.show()
```

```
20/20 ━━━━━━━━━━━━━━━━━━━━ 30s 1s/step
Confusion Matrix:
 [[205  29]
 [ 24 366]]
```



In [ ]:
```
test_probs = model.predict(test_generator, verbose=1).flatten()

f1 = f1_score(true_labels, test_preds)

roc_auc = roc_auc_score(true_labels, test_probs)

tn, fp, fn, tp = confusion_matrix(true_labels, test_preds).ravel()
specificity = tn / (tn + fp)
```

```
print(f"F1-score: {f1:.4f}")
print(f"ROC-AUC: {roc_auc:.4f}")
print(f"Specificity: {specificity:.4f}")
```

```
20/20 ━━━━━━━━━━━━━━━━━━━━ 28s 1s/step
F1-score: 0.9325
ROC-AUC: 0.9561
Specificity: 0.8761
```

Final Discussion

Overall Strengths and Limitations

The project has demonstrated considerable success, but it is important to understand its technical strengths and weaknesses.

Strengths:

A core strength is the strategic use of transfer learning. By adapting the pre-trained MobileNetV2 architecture, which was developed on the ImageNet dataset, the project bypassed the need for building a complex neural network from the ground up, saving significant time and computational resources.

The model successfully trained on an unbalanced dataset where pneumonia cases (3,875) greatly outnumbered normal cases (1,341). This was achieved by implementing class weights, a technique that forces the model to pay closer attention to the minority "Normal" class, ensuring it learned to distinguish both conditions effectively.

Limitations:

The validation set used for fine-tuning the model during development was perfectly balanced but extremely small, containing only 16 images (8 normal and 8 pneumonia). This small size means the model's performance may not be fully representative when applied to a broader, more diverse range of real-world cases.

Despite the use of data augmentation techniques like rotating and zooming images to prevent memorization, the performance fluctuations observed during validation suggest a slight risk of the model being overfitted to the training data.

Business Implications The deployment of this diagnostic system at Charité Hospital offers tangible benefits to clinical operations and patient care.

This technology can substantially decrease the turnaround time for interpreting chest X-rays. This speed is particularly impactful in high-volume environments or facilities where radiologists are not always immediately available, enabling faster clinical decision-making.

Identifying pneumonia at its onset can prevent the progression to more severe illness, thereby avoiding more complex and costly medical interventions.

Data-Driven Recommendations

To advance this solution from a successful prototype to a reliable clinical instrument, the following actions are recommended:

Procure a larger, more diverse set of validation images. A robust dataset is essential for refining the model and confidently verifying its accuracy across varied patient demographics, especially since pneumonia death rates are highest among the elderly.

The system should be integrated into the clinical workflow as a tool to assist, not replace, medical experts. Its optimal role is as a preliminary screening mechanism or a "second opinion" that flags suspicious scans, allowing radiologists to focus their attention on the most critical cases.

Model Explainability

To ensure the model's decisions are transparent and trustworthy, we can implement explainability techniques:

Tools like Grad-CAM (Gradient-weighted Class Activation Mapping) or SHAP (SHapley Additive exPlanations) can be used to generate heatmaps on the X-ray images. These visualizations highlight the specific areas the model focused on to make its classification, allowing us to verify if it's concentrating on relevant lung regions.

A thorough analysis of instances where the model makes incorrect predictions (false positives and false negatives) is essential. This will help us understand the model's weaknesses and potentially improve its performance, for instance, by better distinguishing between different types or severities of pneumonia.

References

https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia

TensorFlow (2025) TensorFlow. https://www.tensorflow.org/.

Team, K. (2025) Keras: Deep Learning for humans. https://keras.io/.

Bsn, R.Z., RN (2025) Pneumonia facts and statistics: What you need to know. https://www.verywellhealth.com/pneumonia-facts-and-statistics-5498569.

World Health Organization: WHO (2019) Pneumonia. https://www.who.int/health-topics/pneumonia/#tab=tab_1.

Dadonaite, B. and Roser, M. (2019) Pneumonia. https://ourworldindata.org/pneumonia.

Lemaître, G., Nogueira, F. and Aridas, C.K. (2017) Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning. https://jmlr.org/papers/v18/16-365.html.

Kermany, D., Zhang, K. and Goldbaum, M. (2018) 'Labeled Optical Coherence Tomography (OCT) and chest X-Ray images for classification,' Labeled Optical Coherence Tomography (OCT) and Chest X-Ray Images for Classification, 2. https://doi.org/10.17632/rscbjbr9sj.2.

Li, Z. et al. (2021) 'A Survey of Convolutional Neural Networks: Analysis, applications, and Prospects,' IEEE Transactions on Neural Networks and Learning Systems, 33(12), pp. 6999–7019. https://doi.org/10.1109/tnnls.2021.3084827.