



Tecnológico de Monterrey

Campus Querétaro

Big Data - Image Classification

Gamaliel Marines Olvera
A01708746

Advanced AI

October 26, 2024

1. Introduction	2
2. Big Data	2
3. PySpark as a Tool for Big Data	2
Comparison with Other Tools	3
4. MLlib for Machine Learning	3
5. Implementation	3
6. Results and Analysis	4
7. Potential Improvements	4
8. Alternative Techniques and Tools	4
9. References	5

Abstract

In this study, I explore the integration of Apache Spark's PySpark and MLlib library to create a scalable machine learning model designed to identify pneumonia from chest X-ray images. This paper examines big data's significance, the role of PySpark for data handling, the application of MLlib for machine learning, my implementation process, and potential avenues for improvement. I conclude by discussing alternative methods and the potential for further advancements in this field.

1. Introduction

The proliferation of big data has revolutionized industries, enabling organizations to handle vast datasets that traditional data processing tools cannot manage effectively. In machine learning, big data allows for the creation of robust predictive models, as larger datasets increase model reliability. However, managing and processing such data requires special tools and technologies capable of performing high-speed and efficient data handling. PySpark, the Python API for Apache Spark, is a solution that enables distributed data processing. In this paper I explain the implementation of a scalable image classification model to detect pneumonia in chest X-ray images, using PySpark's data-handling capabilities and MLlib's machine learning algorithms.

2. Big Data

Big data refers to the extensive volume, variety, and velocity of data that traditional systems cannot manage. In machine learning, big data is invaluable, particularly in image classification tasks where each image contributes thousands of features. The ability to harness big data improves model accuracy by allowing for richer datasets.

In medicine, larger datasets can drive significant advancements by improving diagnostic capabilities. However, datasets are often too large to process on a single machine, necessitating distributed computing solutions such as Apache Spark.

3. PySpark as a Tool for Big Data

Apache Spark is an open-source distributed computing system that allows for efficient data processing across a cluster of machines. PySpark, the Python API for Spark, includes:

- **Resilient Distributed Datasets (RDDs):** The foundational Spark data structure that enables parallel data processing across nodes.
- **DataFrame API:** An optimized version of RDDs that supports SQL-like operations, making it easier to manage large datasets.
- **Machine Learning Integration:** PySpark works seamlessly with Spark's MLlib, which provides machine learning algorithms for tasks such as classification, regression, clustering, and collaborative filtering.

Through PySpark, I was able to manage and preprocess a large dataset of chest X-ray images by distributing tasks, enhancing performance and scalability.

Comparison with Other Tools

- **TensorFlow:** Unlike PySpark, which is optimized for distributed data processing, TensorFlow specializes in deep learning tasks, particularly neural networks. While TensorFlow can handle big data, it is most effective for complex neural networks rather than generalized data processing.
- **Pandas:** Pandas is a Python library for data manipulation and analysis, widely used for smaller datasets. While Pandas provides powerful tools for data wrangling, it is limited to a single machine and is not inherently distributed, making it unsuitable for true big data tasks. PySpark, in contrast, can handle datasets that far exceed single-machine memory capacity.
- **Hadoop:** Hadoop is another big data framework. While efficient for batch processing, Hadoop lacks the speed of in-memory processing found in Spark, making Spark generally faster and more suited to iterative machine learning tasks.

4. MLlib for Machine Learning

MLlib is a machine learning library within Spark, designed for scalable ML operations. It includes a variety of algorithms for supervised and unsupervised learning, feature selection, and dimensionality reduction. Unlike traditional machine learning libraries, MLlib is optimized for distributed computing, making it particularly well-suited for big data applications.

For this project, I employed MLlib's **Linear Regression** algorithm to classify images, treating the classification task in a regression framework.

5. Implementation

The dataset used for this study comprised thousands of chest X-ray images, organized into two categories: "NORMAL" and "PNEUMONIA." My implementation consisted of several steps:

1. **Data Loading and Preprocessing:** Using PySpark, I read images from directories, preprocessing them for model compatibility. ([The Dataset consists of 1.24GB](#), what was determined to be considered as Big Data).

```
!pip install pyspark

from pyspark.sql import SparkSession

spark = SparkSession.builder\
    .master("local")\
    .appName("Colab")\
    .config('spark.ui.port', '4050')\
    .getOrCreate()

spark
```

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import input_file_name
import os
from pyspark.sql import SparkSession
from pyspark.ml.image import ImageSchema

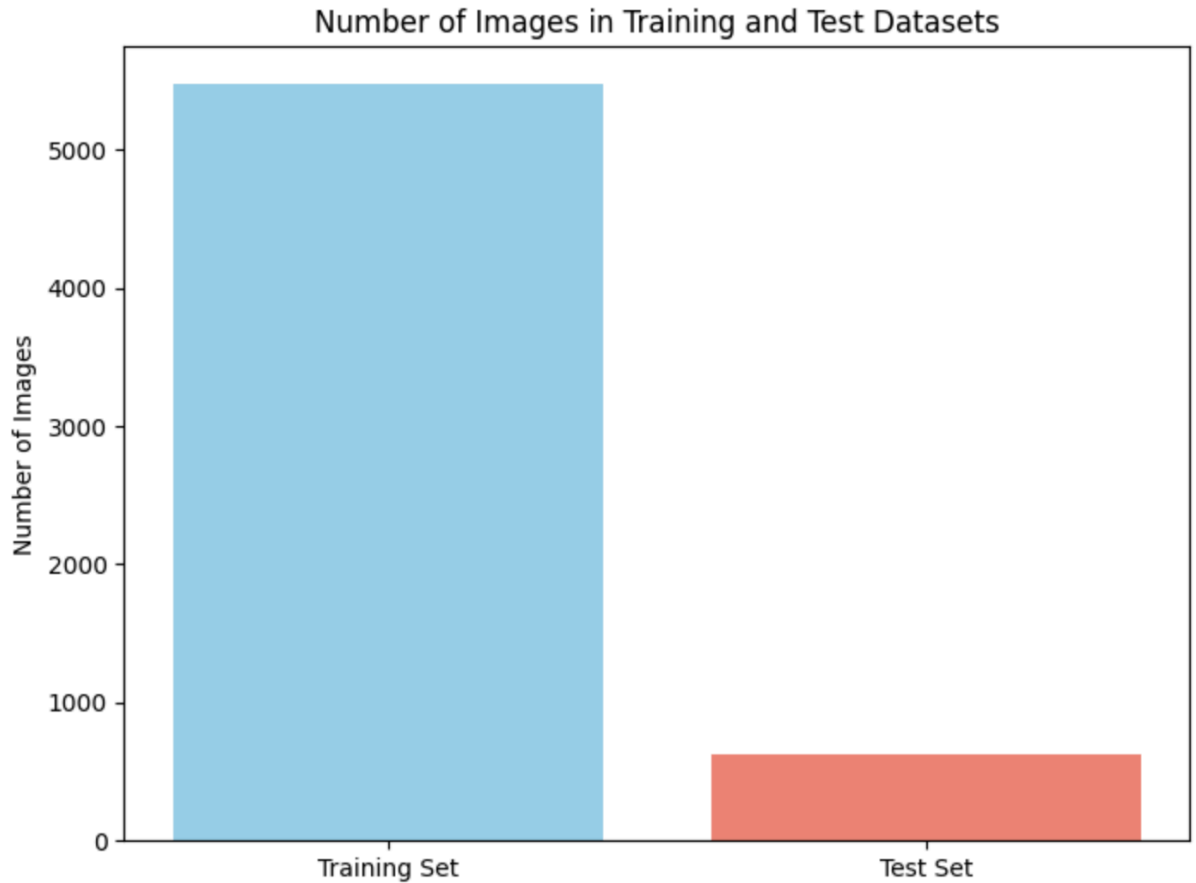
train_dir = os.path.join('/content/drive/MyDrive/Advanced AI/reto big data/chest_xray/train')
test_dir = os.path.join('/content/drive/MyDrive/Advanced AI/reto big data/chest_xray/test')

# Load images recursively from subdirectories (NORMAL, PNEUMONIA)
train_df = spark.read.format("binaryFile") \
    .option("recursiveFileLookup", "true") \
    .option("pathGlobFilter", "*.jpeg") \
    .load(train_dir)

test_df = spark.read.format("binaryFile") \
    .option("recursiveFileLookup", "true") \
    .option("pathGlobFilter", "*.jpeg") \
    .load(test_dir)
```

2. It is important to understand the dataset and its distribution to make adequate decisions. To have a better look at the dataset I plot the number of images in the

dataset by subsets (training set and test set).



With help of this plot and the information gained from it i decided that there too much images for the training set, too few for the Test set. Also a lack of a validation subset.

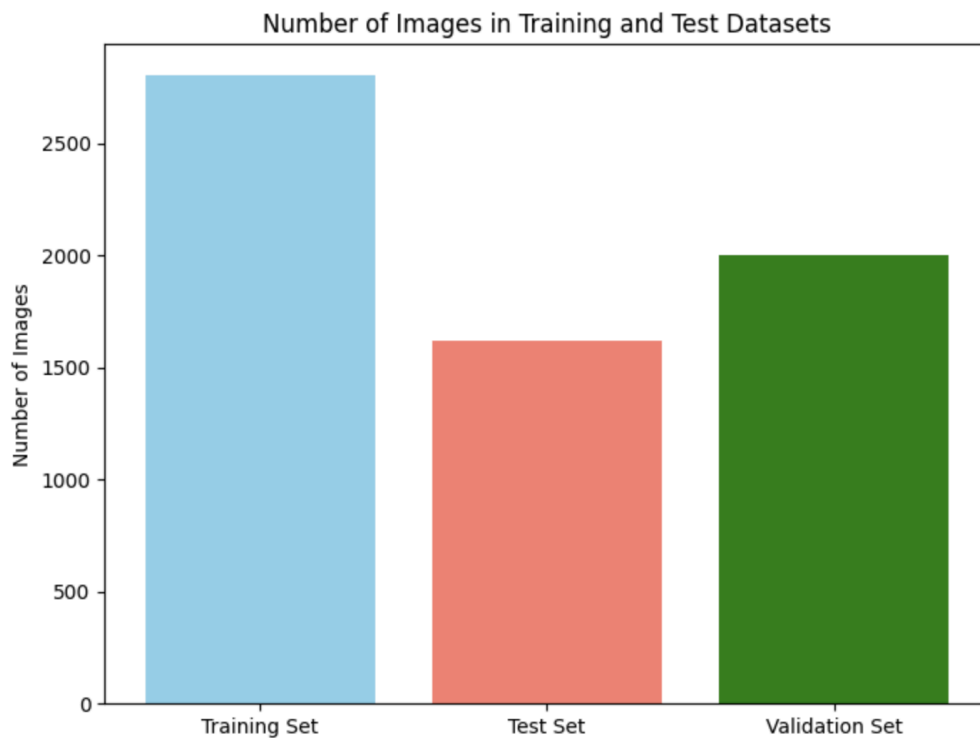
3. Using pyspark, I reorganized the dataset and its distribution.

```
from pyspark.sql import SparkSession
import os
import matplotlib.pyplot as plt

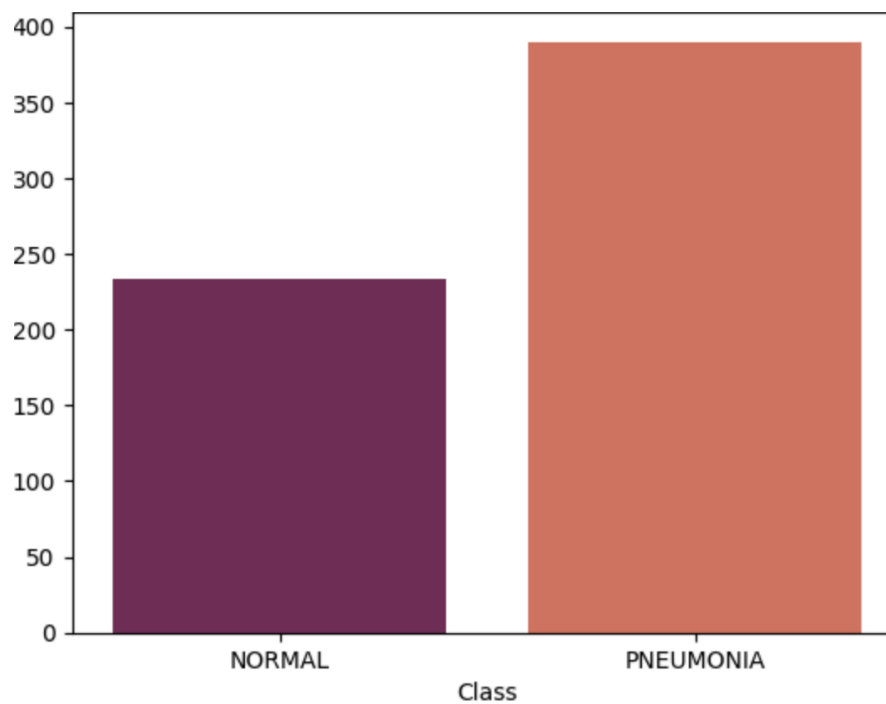
# Sample 1000 images for the new test set and 2000 for the validation set from train_df
test_from_train_df = train_df.sample(fraction=1000/train_df.count(), seed=42)
validation_from_train_df = train_df.sample(fraction=2000/train_df.count(), seed=43)

# Filter out these sampled images from the original training set to avoid duplication
train_remaining_df = train_df.subtract(test_from_train_df).subtract(validation_from_train_df)

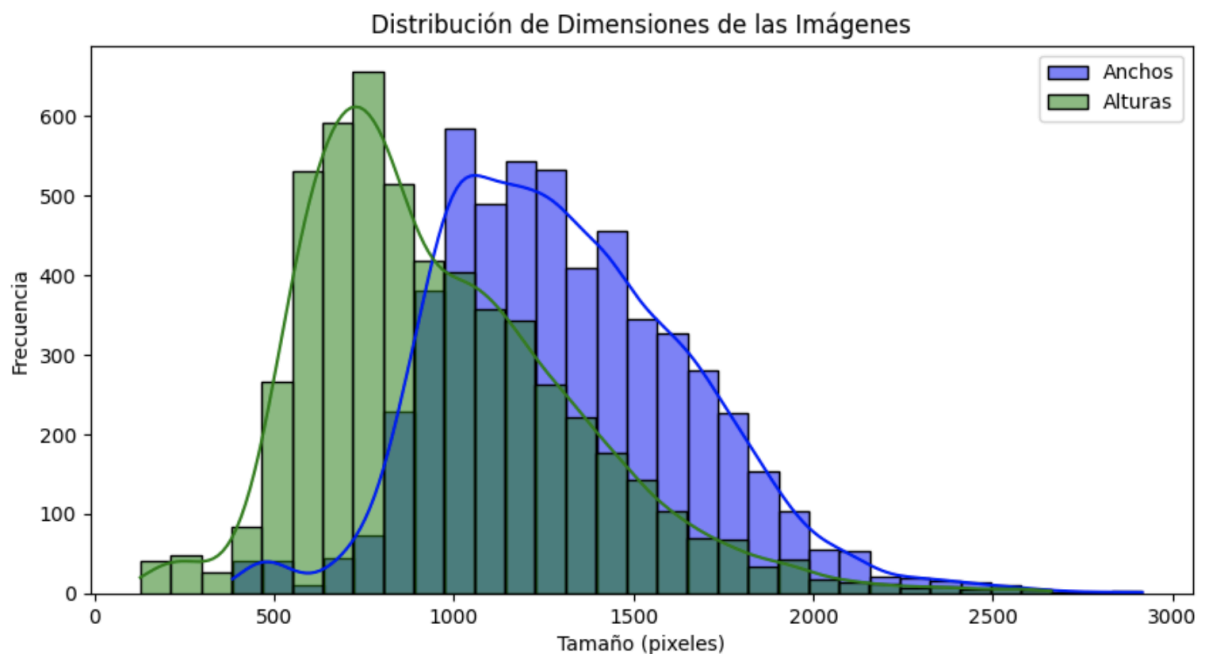
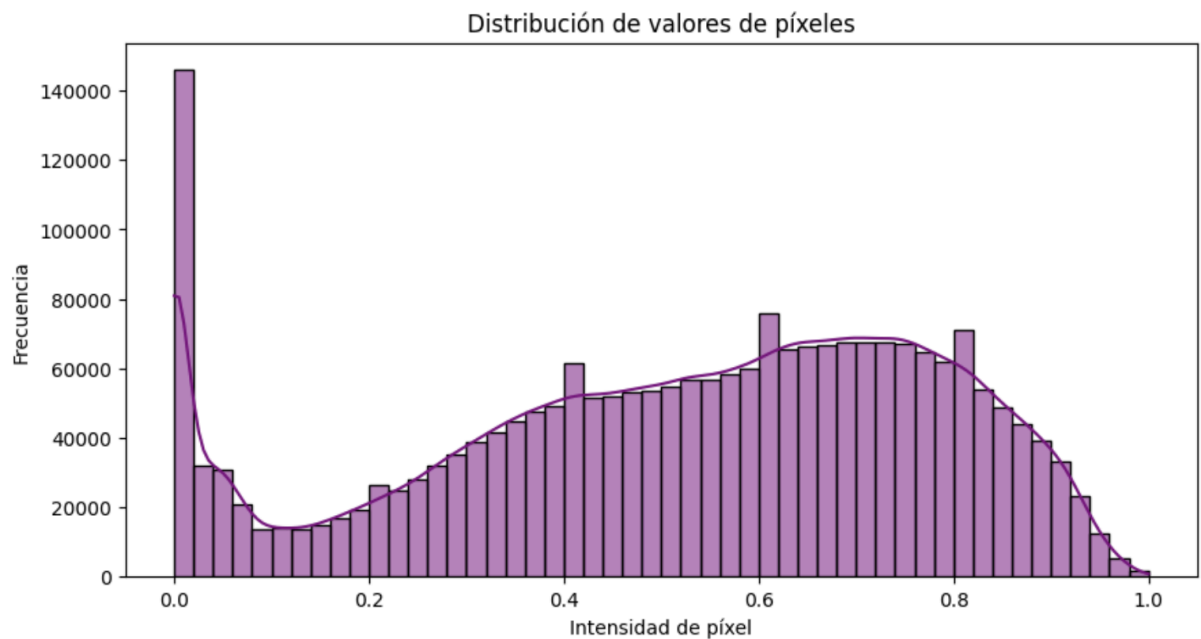
# Combine original test set with new samples from the training set
new_test_df = test_df.union(test_from_train_df)
validation_df = validation_from_train_df
```



4. After I reorganized the dataset, I plot the distribution of classes of the dataset in order to identify possible bias.



5. After organizing the dataset and subsets, I plot the sizes and the dimensions of the images in order to understand more about the nature of the data.



With the information gained I decided to resize the images and convert them to grayscale in order to have less variables that could negatively affect the training of the model by adding noise and prevent a smooth training.


```
from pyspark.sql.functions import udf
from pyspark.sql.types import IntegerType, ArrayType, FloatType
from PIL import Image
import io
import numpy as np

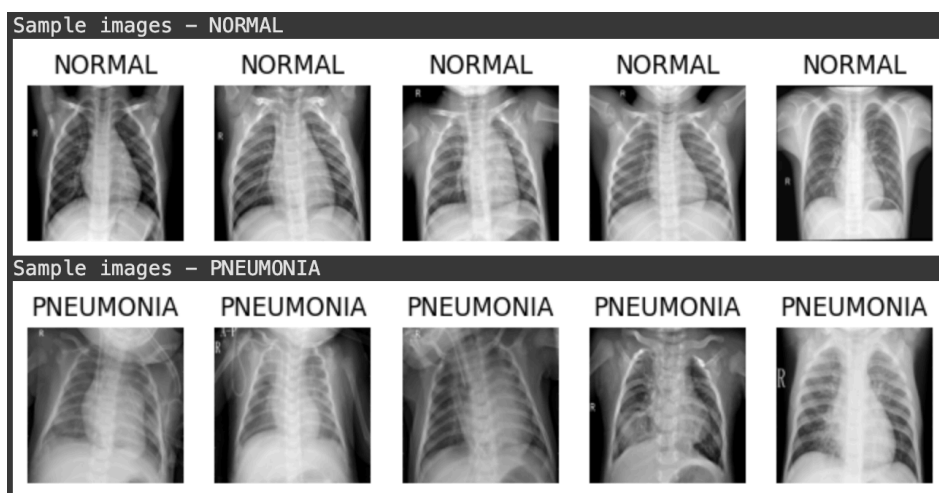
# Function to decode image and convert to feature vector
def decode_image(content):
    img = Image.open(io.BytesIO(content)) # Open image from binary content
    img = img.resize((150, 150)) # Resize image to (150, 150)
    img_array = np.array(img).astype(np.float32).flatten() / 255.0 # Normalize pixel values and flatten
    return img_array.tolist() # Return as a list for Spark compatibility

# Register UDF to decode images
decode_image_udf = udf(decode_image, ArrayType(FloatType()))

# Apply the UDF to the content column to create features
train_df = train_df.withColumn("features", decode_image_udf(train_df['content']))
test_df = test_df.withColumn("features", decode_image_udf(test_df['content']))
```

I resized images to 150×150pixels and converted them to grayscale, flattening each image into a 22,500-element feature vector

6. **Feature Engineering:** I used a user-defined function (UDF) to normalize pixel values and store them as feature vectors. Labels were assigned based on file paths to classify images as normal or pneumonia.
7. **Model Training:** The feature vectors were used as inputs to a linear regression model, where "NORMAL" images were labeled as 0 and "PNEUMONIA" images as 1. The model was trained with specified parameters, including a regularization term to prevent overfitting. (It is important to verify the quality of the dataset, that is why, before training the model, I double checked a fragment of the dataset for each class to corroborate its quality.)

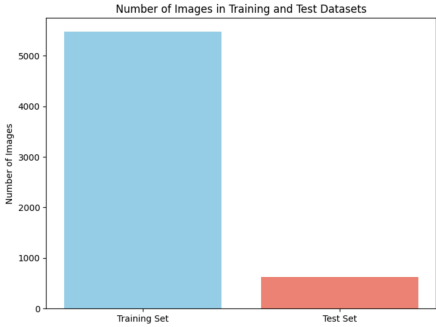
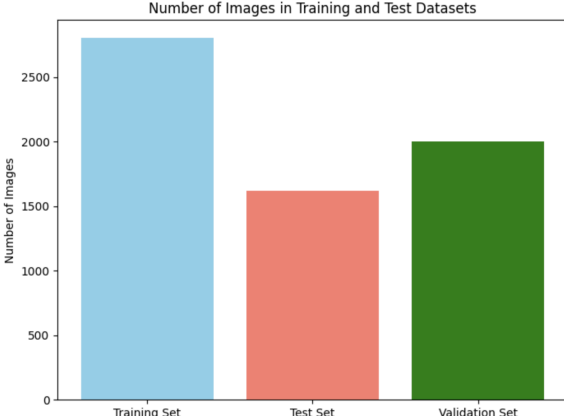


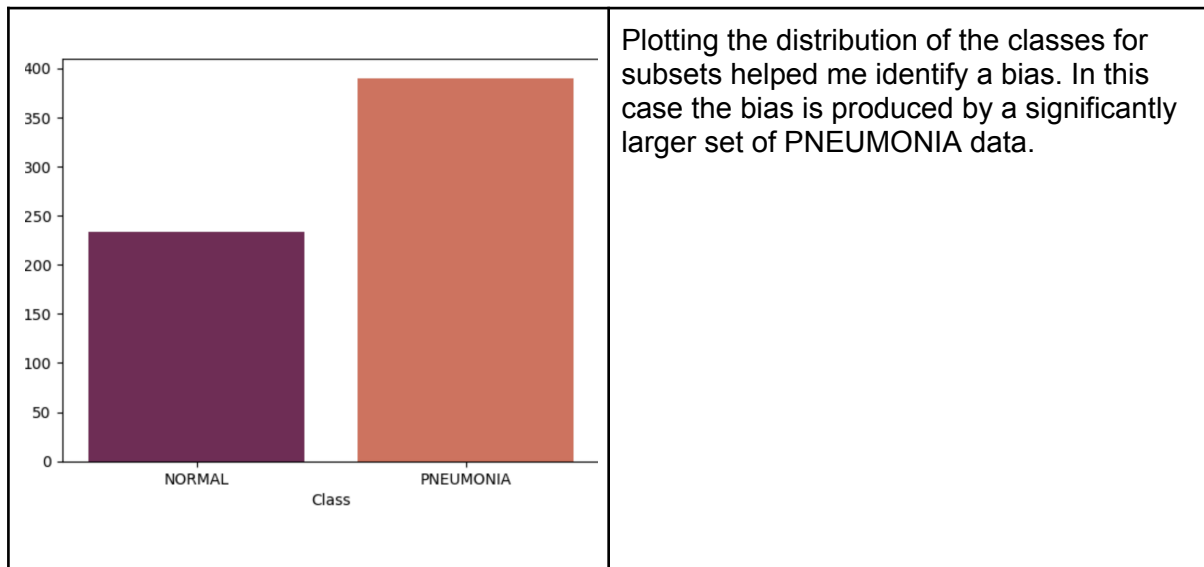
8. **Evaluation:** The model's performance was assessed using Root Mean Squared Error (RMSE) and R-squared (R^2) metrics. While RMSE provides insights into prediction error, R^2 indicates the proportion of label variance explained by the model.

6. Dashboard and Graphs/Plots

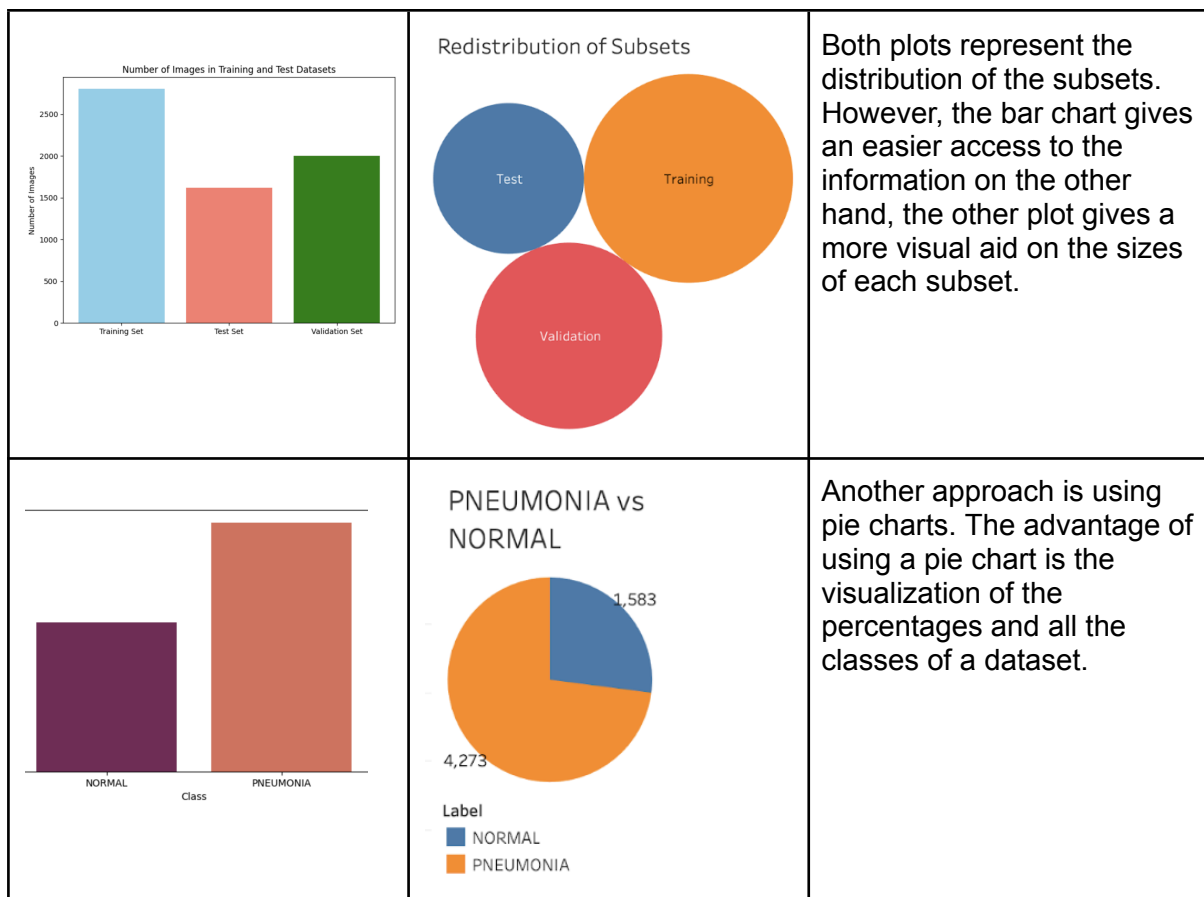
Each one of the graphs/plots used are important and helped me make important decisions. Most of the plots used are bar plots because of the nature of the information represented.

The first and easiest plots to get and interpret are the distribution of the Dataset, Subsets and classes.

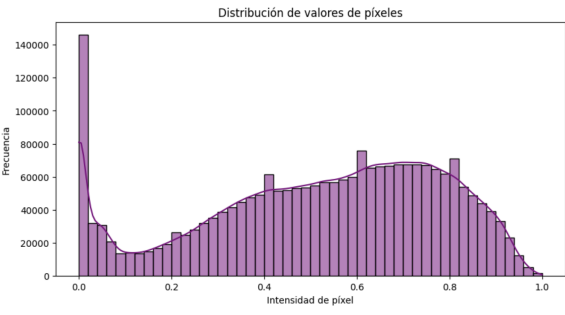
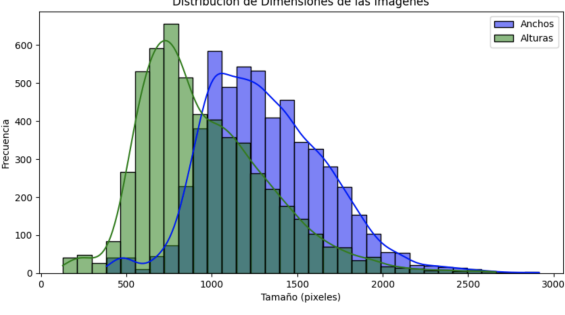
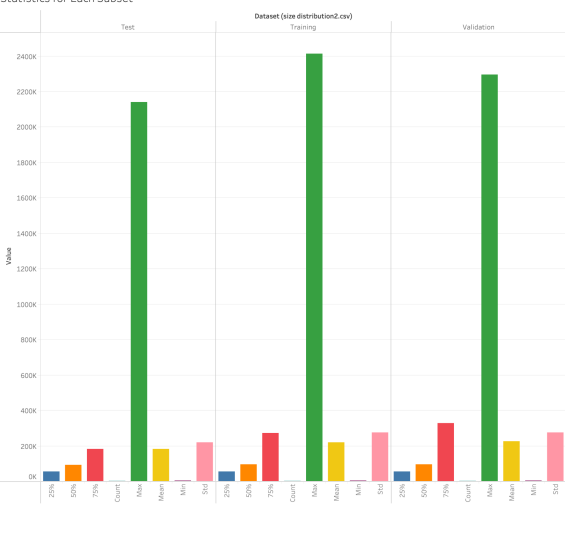
 <table border="1"> <caption>Number of Images in Training and Test Datasets</caption> <thead> <tr> <th>Dataset</th> <th>Number of Images</th> </tr> </thead> <tbody> <tr> <td>Training Set</td> <td>~5500</td> </tr> <tr> <td>Test Set</td> <td>~600</td> </tr> </tbody> </table>	Dataset	Number of Images	Training Set	~5500	Test Set	~600	<p>This plot helped me realize the imbalance for the training and test set, as well as the lack of a validation set.</p>		
Dataset	Number of Images								
Training Set	~5500								
Test Set	~600								
 <table border="1"> <caption>Number of Images in Training and Test Datasets</caption> <thead> <tr> <th>Dataset</th> <th>Number of Images</th> </tr> </thead> <tbody> <tr> <td>Training Set</td> <td>~2800</td> </tr> <tr> <td>Test Set</td> <td>~1600</td> </tr> <tr> <td>Validation Set</td> <td>~2000</td> </tr> </tbody> </table>	Dataset	Number of Images	Training Set	~2800	Test Set	~1600	Validation Set	~2000	<p>Once I added the validation subset, I plotted the distribution of the subsets once more to corroborate that the proportions were adequate.</p> <p>I made sure to have a larger set for training, but having sufficient images for validation and test set to have a better evaluation of the model.</p>
Dataset	Number of Images								
Training Set	~2800								
Test Set	~1600								
Validation Set	~2000								



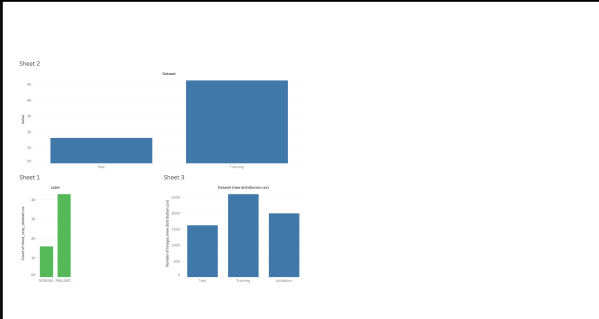

Most of the times there are various ways to plot the same information, but each has its purpose and makes a different emphasis on the display of the information



Other important plots that were essential for my implementation and helped me with making decisions were the following

	<p>Learning the intensity of the pixels and the variety of them through the whole dataset helped me by giving me the arguments to “flatten” and turn them into grayscale. This helps the model because now the model does not have to pay attention to the intensity of the images and get distracted by those details.</p>
	<p>Also, I wanted to know if all the images had the same sizes. So I plotted the heights and widths of the images.</p> <p>Using the information gained, I resized the images to a scale where all of the images had the same height and width while keeping enough information (pixels) and not reducing their quality.</p>
	<p>For a deeper understanding of the subsets and the data in each subset I plot statistical information.</p> <p>The purpose of plotting the min, max, mean, std helped me identify anomalies in each subset e.g. outliers, different sizes of images, how is the deviation of sizes in each subset. By observing the plot and analyzing the information given I can say that each subset behaves in a similar way, thus giving me no reason to alter or modify the subsets.</p>

Using Dashboards I was able to gather the plots and graphs and put them together in one place. Dashboards are really helpful because of all the information gathered in one place which makes it easier to access and gives great context for important decisions. Dashboards can go from basic to more complex and complete plots and graphs

	<p>Starting with a simple dashboard, using bar charts for better understanding and having an overview of the distribution of the original dataset, of the distribution of the modified dataset with each of their subsets, and the classes for each subset.</p>
	<p>Adding a pie chart and a population graph helps me understand the proportions and the percentages of the distribution for the Dataset.</p> <p>I also added the labels and the meaning for each color so it could be easily understood by people who are new to the context of the problem.</p>
	<p>A more complex dashboard includes all the plots and graphs used during the process of completing the project. Adding plots to understand the distribution, the sizes of the images, the classes of the dataset, and statistic information of the subsets.</p> <p>The information gained from each plot helped with decisions made throughout the project as previously explained.</p>

7. Results and Analysis

The trained model's coefficients and intercepts indicate its ability to distinguish between normal and pneumonia X-ray images, albeit within the limitations of a regression model for a binary classification problem. RMSE and R^2 metrics offered a baseline assessment of the model's predictive performance.

8. Potential Improvements

To enhance model performance, future iterations could incorporate the following improvements:

- **Binary Classification Algorithms:** Given the binary nature of pneumonia detection, switching from regression to a classification algorithm, such as logistic regression or decision trees, would likely improve accuracy.
- **Deep Learning Integration:** MLlib's support for deep learning frameworks, such as TensorFlow or PyTorch, would enable convolutional neural networks (CNNs), which are highly effective for image classification.
- **Increased Feature Engineering:** Techniques like feature extraction with pre-trained models (e.g., VGG, ResNet) could improve feature representation, as these models are specifically trained on large image datasets.

9. Alternative Techniques and Tools

Other libraries and platforms that could offer advantages for big data and machine learning tasks:

- **TensorFlow on Spark:** This combination allows for scalable deep learning applications on large datasets.

9. References

Chest X-Ray images (Pneumonia). (2018, 24 marzo). Kaggle.

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

PySpark Overview — PySpark 3.5.3 documentation. (s. f.).

<https://spark.apache.org/docs/latest/api/python/index.html>

Data preprocessing for ML: options and recommendations. (s. f.). TensorFlow.

https://www.tensorflow.org/tfx/guide/tft_bestpractices