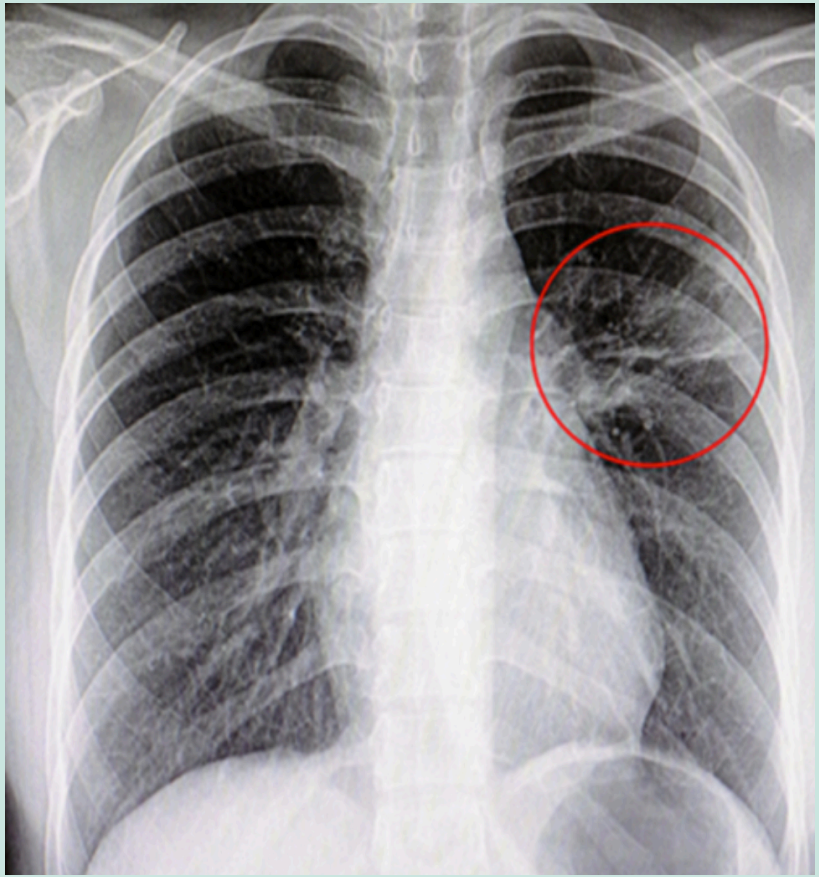


# Pneumonia Detection from X-ray images using Deep Learning Neural Networks

Javier Herbas

# Presentation Outline

Pneumonia background

Data Gathering

Convolutional Neural Networks

Model Evaluation

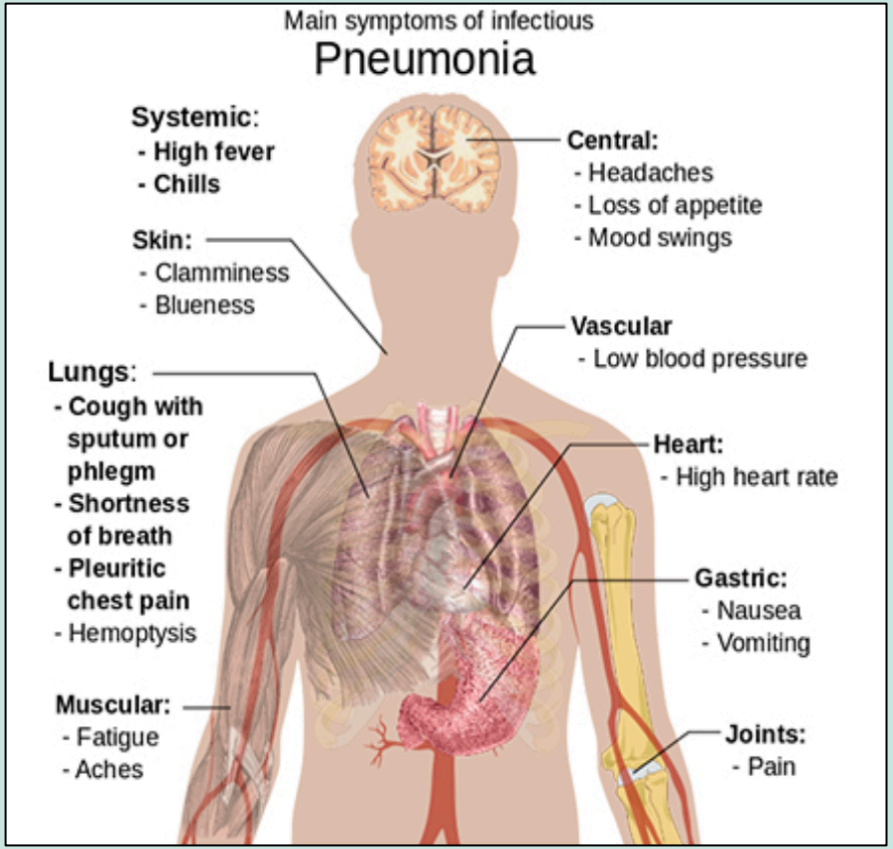
Summary and Recommendations

Way Forward

# Pneumonia Background

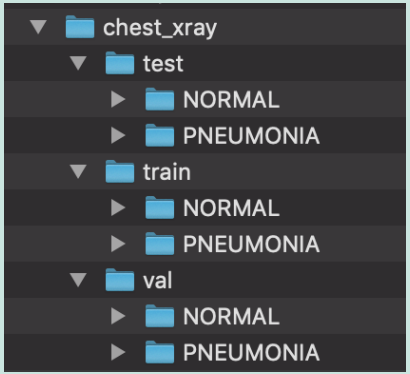
What is Pneumonia?

- Is a lung infection that can be caused by different types of microorganisms including bacteria, viruses, and fungi
- It causes the lung's air sac, or alveoli, to become inflamed and filled up with fluid and/or pus, making breathing more difficult
- Severe pneumonia will inflamed the lungs to a point where they cannot take enough oxygen or expel enough carbon dioxide
- Continuous depravation of oxygen can damage organs such kidney and the heart
- Today the most common diagnosis of severe Covid-19 is severe Pneumonia

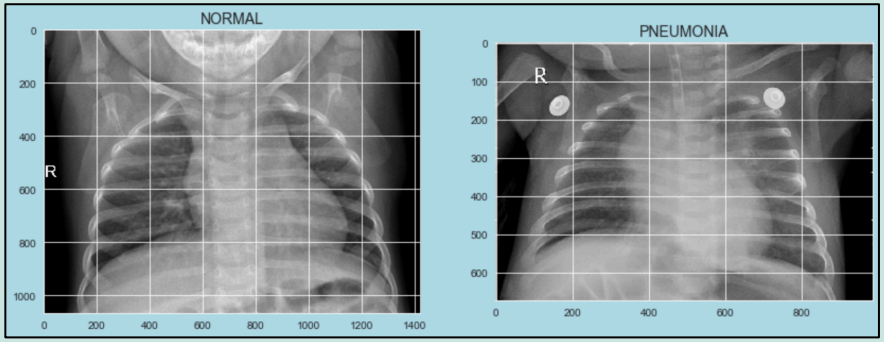


# Data Gathering and QC

- Source: Kaggle
- X-rays: 5856
- Nested Folder Structure:
  - Train subset: 5216 (89.07%)
  - Test subset: 624 ➡ 524 (8.95%)
  - Val subset: 16 ➡ 116 (1.98%)



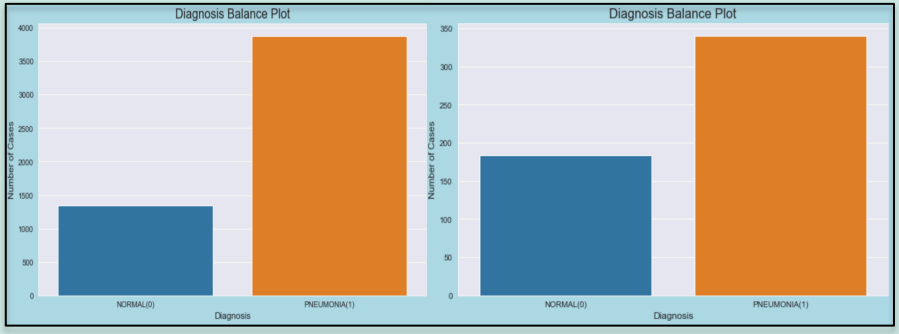

Accessing & plotting X-ray images




Looking at the data distribution

Train Subset

Test Subset




- Images were resize to 256 colors (0-255)
- Train subset was augmented modifying 4 parameters

# Convolutional Neural Network Model

- 15 different model architectures were tested
- Simple 2-layer models ➡ more complex 5-layer models.
- Tests were done with 5, 10 and 15 epochs. If validation accuracy > 70% then model was fitted with 30 and 50 epochs

**Model_5**

```
model_5 = models.Sequential()

# 1st Convolution
model_5.add(layers.Conv2D(32, (3, 3), activation='relu',
                          input_shape=(150, 150, 3)))
model_5.add(layers.MaxPooling2D((2, 2)))

# 2nd Convolution
model_5.add(layers.Conv2D(64, (3, 3), activation='relu'))
model_5.add(layers.MaxPooling2D((2, 2)))

# 3rd Convolution
model_5.add(layers.Conv2D(128, (3, 3), activation='relu'))
model_5.add(layers.MaxPooling2D((2, 2)))

# 4th Convolution
model_5.add(layers.Conv2D(128, (3, 3), activation='relu'))
model_5.add(layers.MaxPooling2D((2, 2)))

# Flattened the layer
model_5.add(layers.Flatten())

# Fully connected layers
model_5.add(layers.Dense(64, activation='relu'))
model_5.add(layers.Dense(2, activation='softmax'))

model_5.summary()
```

**Model_15**

```
model_15 = models.Sequential()

# 1st Convolution block
model_15.add(layers.Conv2D(16, (3, 3), activation='relu', padding='same',
                           input_shape=(150, 150, 3)))
model_15.add(layers.Conv2D(16, (3, 3), activation='relu'))
model_15.add(layers.MaxPooling2D((2, 2)))

# 2nd Convolution block
model_15.add(layers.Conv2D(32, (3, 3), activation='relu', padding='same'))
model_15.add(layers.Conv2D(32, (3, 3), activation='relu'))
model_15.add(layers.MaxPooling2D((2, 2)))

# 3rd Convolution block
model_15.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
model_15.add(layers.Conv2D(64, (3, 3), activation='relu'))
model_15.add(layers.MaxPooling2D((2, 2)))

# 4th Convolution block
model_15.add(layers.Conv2D(96, (3, 3), dilation_rate=(2,2), activation='relu',
                           padding='same'))
model_15.add(layers.Conv2D(96, (3, 3), activation='relu'))
model_15.add(layers.MaxPooling2D((2, 2)))

# 5th Convolution block
model_15.add(layers.Conv2D(128, (3, 3), dilation_rate=(2, 2), activation='relu',
                           padding='same'))
model_15.add(layers.Conv2D(128, (3, 3), activation='relu'))
model_15.add(layers.MaxPooling2D((2, 2)))

# Flattened the layer
model_15.add(layers.Flatten())

# Fully connected layers
model_15.add(layers.Dense(64, activation='relu'))
model_15.add(layers.Dropout(0.4))
model_15.add(layers.Dense(2, activation='softmax'))

model_15.summary()
```

# Model Evaluation


**Model_5:**
Epoch 50/50
Train Accuracy: 0.9722
Train Loss: 0.0803
Val. Accuracy: 0.1954
Train Loss: 0.9310

```
test accuracy: 91.22
test loss: 0.39
```

Almost identical

**Model_15:**
Epoch 50/50
Train Accuracy: 0.9631
Train Loss: 0.1026
Val. Accuracy: 0.9224
Train Loss: 0.2479


```
test accuracy: 90.84
test loss: 0.3
```

# Model Evaluation

| | | Predicted | |
|---|---|---|---|
| | | Negative | Positive |
| Actual | Negative | True Negative | False Positive |
| | Positive | False Negative | True Positive |

- For unbalanced data the accuracy is not the best metric

As a reminder:

Precision = TP/(TP + FP)

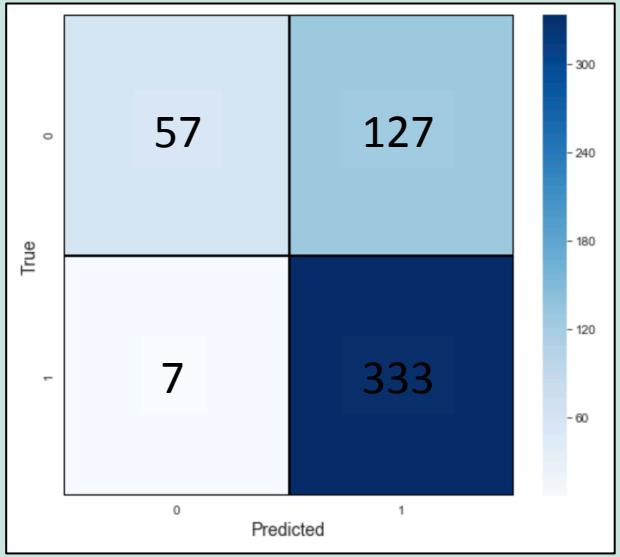
Recall = TP/(TP + FN)

F1 = (TP + TN)/Total

TP = True Positive
FP = False Positive
FN = False Negative
TN = True Negative

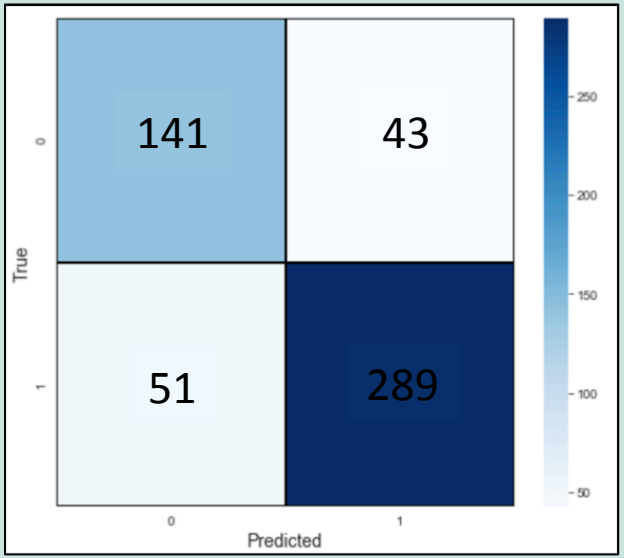
**Model_5**

Precision score: 72.39 %
Recall score: 97.94 %
F1 score: 83.25%




**Model_15**

Precision score: 87.05%
Recall score: 85.00%
F1 score: 86.01%



# Summary & Recommendations

The following recommendations are based on my short experience with Deep Learning Neural Network, and they represent the challenges I faced during this project.

- Check your data's distribution so that you can properly interpret your model after training it. An unbalanced dataset will not be properly evaluated with just accuracy but with other metrics such as precision and recall. Confusion Matrix are very useful

- Don't over complicate the model's architecture as it won't necessarily translate as better. Start simple and add layers as you progress

- Don't limit yourself to just one model. Try different options so that you can compare results and reduce uncertainty

# Way forward

- Train both selected models with a minimum of 100 epochs to try to reach convergence
- Redo the selected models with a split of 80, 20, 20% for train, test and validation subset respectively, and compare to current split used
- Redo this exercise with the original dataset from the Mendeley site (June 2018)

01110100
01101000
01100001
01101110
01101011
00100000
01111001
01101111
01110101