# Pneumothorax Detection in Chest X-rays report

Huda Kouli

September 23, 2024

## Overview

-This report outlines the key steps involved in implementing the Pneumothorax Detection in Chest X-rays project
-Implementation follows the PEP8 style guidelines, and utilises the PyTorch library, as this is a research-oriented project. In contrast, TensorFlow and Keras are commonly employed for projects destined for deployment environments.

## 1. Data processing and augmentation approach

The key steps involved to do the data processing are as follows:

1. **Data Grouping:** The input data consisted of CSV files containing the labels and corresponding image, masks files depicting various patterns. To establish the link between the images and their respective labels, we grouped the images by patient ID, allowing us to associate each chest image with its corresponding mask.

2. **Label Extraction:** From the CSV files, we extracted the labels that corresponded to the masked images, ensuring a seamless pairing of the visual data and their associated annotations.

3. **Dataset Formation:**With the labelled data in hand, we proceeded to split the dataset into training, validation, and test subsets.

-Result for splitting data sizes:

| |
|---|
| Training set size: 8440 |
| Validation set size: 1798 |
| Test set size: 1809 |

**Data Augmentation approach:**

-Here's a set of effective data augmentation techniques that can be applied to prepare the dataset.

1. **Resizing:** The input images have been resized to 512x512 pixels, to ensure consistency and compatibility with the model architecture.

2. **Rotation:** Random rotation within a range of $\pm 15$ degrees to help the model learn to recognize objects and features at different orientations.
   '

3. **Horizontal Flipping:** Applying random horizontal flipping to introduce additional variations in the training data, further enhancing the model's ability to generalise.

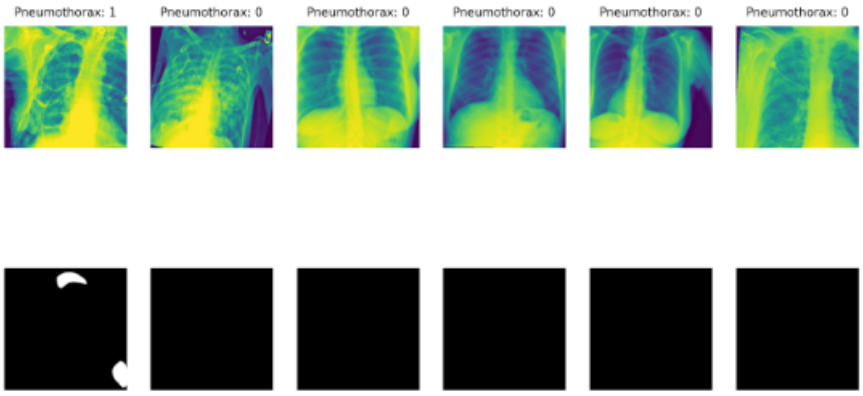


Figure 1: Caption

4. **Brightness and Contrast Adjustments:** adjusting the brightness and contrast of the input images with brightness=0.5, contrast=0.5 to make the model more robust to variations in lighting conditions.
5. **Cropping:** cropping the image to a size of 224x224 pixels, to simulate the model's exposure to different regions of the input image, which can improve its feature extraction capabilities.
6. **Normalisation (First step):** The input data should be normalised to have a mean of 0.5 and a standard deviation of 0.5 for a single channel,to ensure the input data is on a consistent scale and to improve the model's training stability.
7. **Grayscale Conversion:** To prepare the dataset for a ResNet architecture with a single input channel.

**Construct the dataloader pipeline with the following configurations:**

1. **Batch size: 6**, ensuring each batch contains 6 images.
2. **Number of workers: 4**, which sets the number of subprocesses to use for data loading, thereby accelerating the data loading process.

-Here's the results for utilising an initial code to ensure the proper alignment between images, masks, and labels.

### Data Normalisation based on the training set statistics

-To ensure that your model is trained on data with a consistent scale, the final data loader pipeline is normalised based on the training set statistics.

This process involves calculating the mean and standard deviation of the training data and then using these values to normalise the training, validation, and test sets.

In figure 2 you can see Final labelled dataloader pipeline output:

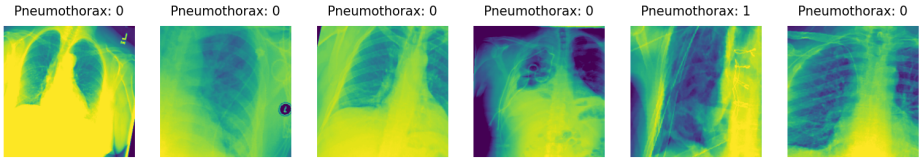


Figure 2: Labelled images

## 2. CNN Architecture.

-ResNet-50 architecture already incorporates Global Average Pooling before the final fully connected layer, so there is no need to manually add that component
-We have modified only the final fully connected layer to output 2 classes, enabling a binary classification task. The two classes are: class 0 for "No Pneumothorax" and class 1 for "Pneumothorax".
-For the fine-tuning process, we have made the following modifications to the architecture:

1. Freezing all the layers except the final fully connected layer
2. Utilising the cross-entropy loss function as the objective
3. Employing the Stochastic Gradient Descent (SGD) optimizer, which is known for its effectiveness with Convolutional Neural Networks (CNNs)
4. Setting the learning rate to 0.001 to prevent catastrophic forgetting
5. Applying a momentum of 0.9 to reduce noise and stabilise the learning process

## 3. Training process

**Fine-Tuning Results for 20 epochs:**

| Accuracy | Precision | Recall | F1-score | ROC-AUC | Process in sec | Process in h/min/sec |
|---|---|---|---|---|---|---|
| 0.7511 | 0.3491 | 0.1598 | 0.2192 | 0.5382 | 13040.91 | 03:37:20 |

**Training for non-normalized data for 50 epochs:**

| Accuracy | Precision | Recall | F1-score | ROC-AUC | Process in sec | Process in h/min/sec |
|---|---|---|---|---|---|---|
| 0.7808 | 0.4892 | 0.04929 | 0.08956 | 0.5174 | 98870.64 | 27:27:50 |

**Training for normalised data based on the training set statistics for 50 epochs:**

| Accuracy | Precision | Recall | F1-score | ROC-AUC | Process in sec | Process in h/min/sec |
|---|---|---|---|---|---|---|
| 0.7818 | 1.0 | 0.0027 | 0.0054 | 0.5013 | 117413.87 | 32:36:53 |

In figure 3 you can see the Training visuals results for 50 epochs on the normalized dataset based on the training set statistics.

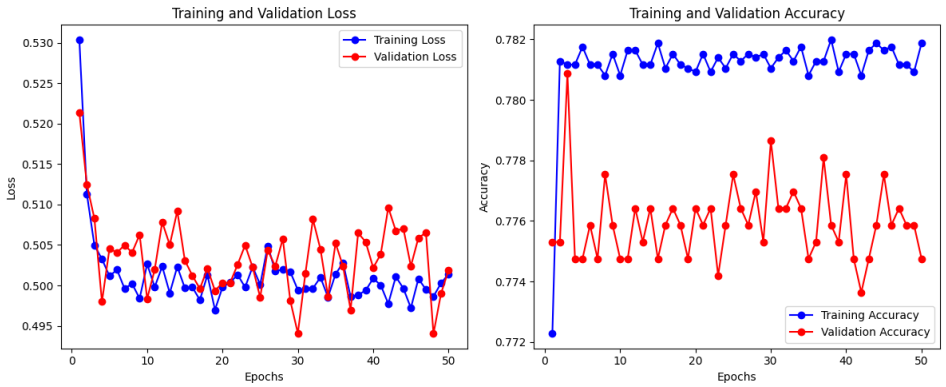


Figure 3: Training visuals

- **Findings:**

  1. **Initial Drop:** in both training and validation loss within the first few epochs. This indicates that the model is quickly learning the basic structure of the data.
  2. **Vibration:** in Both the training and validation losses,This suggests that the model might be having issues with stability due to overfitting or high learning rates.
  3. **Convergence:** Neither the training nor the validation loss seems to converge to a steady value, indicating that the model may not be learning effectively.
  4. **Initial Spike in Training Accuracy:** which stabilises around 78 percent This suggests that the model is fitting the training data well.

5. **Validation Accuracy vibrations:** which indicate that the model is not generalising well to unseen data and might be overfitting to the training set.
6. **Gap Between Training and Validation Accuracy:** which is a typical sign of overfitting.

- **Conclusions:** (To reduce overfitting)

1. **Hyperparameter Tuning Needed:** Consider adjusting the learning rate, adding regularisation (such as dropout or weight decay).
2. **Apply Data Augmentation techniques:** to the training set to help the model generalise better.

## 4. Evaluation Results for applying early stopping and reduce on plateau with patience = 5

-Patient for early stopping is chosen to be 5 after observing the loss changing over the training process.

| Accuracy | Precision | Recall | F1-score | ROC-AUC | Process in sec | Process in h/min/sec |
|---|---|---|---|---|---|---|
| 0.7774 | 0.4502 | 0.0807 | 0.1369 | 0.5266 | 41664.34 | 11:34:24 |

In figure 4 you can see the Training with plateua visuals.

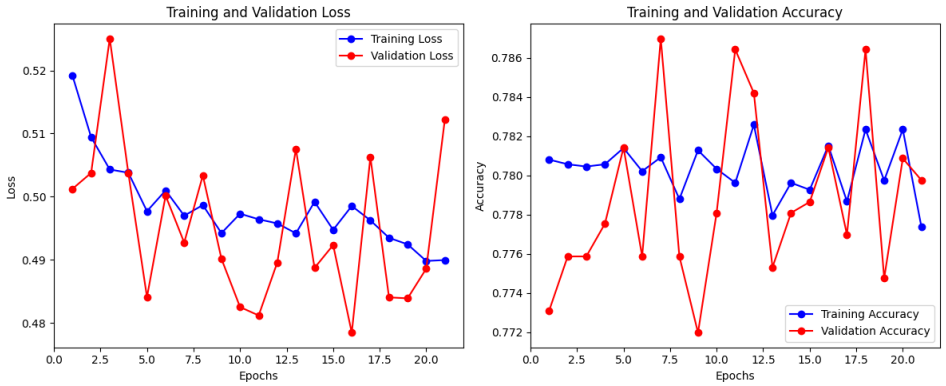


Figure 4: Training with Plateau visuals

- **Findings:**

1. **Vibrations:** In both training and validation losses,This is an indication that the model is struggling to find a stable optimal point, possibly due to a high learning rate or inadequate regularisation.

2. **poor generalisation:** there is a slight downward trend in training loss, suggesting the model is learning, but the validation loss does not show consistent improvement, indicating poor generalisation.
3. **Spikes in Validation Loss:** The large spikes in validation loss, especially around epochs 3, 9, and 21, indicate that the model is not performing well on the validation set and is overfitting on the training data.
4. **Validation Accuracy Spikes:** The validation accuracy has large spikes, sometimes exceeding the training accuracy, which might indicate that the model is occasionally making good predictions but lacks consistency.
5. **High Variability:** There is high variability in validation accuracy, which is not desirable. This suggests that the model is sensitive to changes in the data and is not generalising well.
6. **Training Accuracy Stability:** Training accuracy is relatively stable, hovering around 78-79 percent, but does not show significant improvement. This suggests that the model might have reached a plateau in learning.

- **Conclusions:**

1. **Learning Rate Adjustment:**Reducing the learning rate might help in stabilising the loss and accuracy curves.
2. **Applying Regularisation Techniques:**Adding dropout or L2 regularisation can help reduce overfitting.
3. **Data Augmentation:** Further data augmentation could help the model generalise better by making the training data more diverse.
4. **Early Stopping Indication:** The use of early stopping can be considered, since further training may not lead to improved performance and might even worsen it.

## Testing TTA

| Accuracy | Precision | Recall | F1-score | ROC-AUC | Process in sec | Process in h/min/sec |
|---|---|---|---|---|---|---|
| 0.7683 | 0.0 | 0.0 | 0.0 | 0.5 | 1473.14 | 00:24:33 |

## Enhancements

### Using Binary cross entropy:

-Which combines LogSoftmax and Negative Log-Likelihood loss in one function, which means it calculates the probability distribution and then measures the distance between the predicted and true distributions,the model outputs logits, and CrossEntropyLoss converts these to probabilities using softmax internally and calculates the loss accordingly.
-We need to change the parameter number of classes to 1 since CombinedLoss is designed for binary classification,where the target is expected to be of size (batch size, 1) and consist of binary values (0 or 1).

### Using Gradient Accumulation

-Implementing gradient accumulation is a great way to simulate larger batch sizes when you are constrained by GPU memory. Instead of performing a backpropagation update after every batch, you can accumulate gradients over several batches, then perform an update as if you processed a larger batch It reduces the training time and training stability which leads to smoother convergence

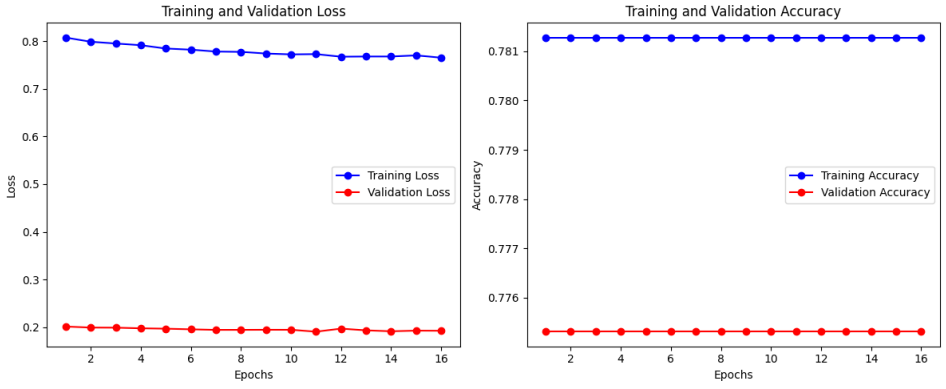


Figure 5: Training results after applying enhancement techniques

- **Findings:**
    1. Both training and validation accuracy are stable, with no visible changes or improvements over epochs.
    2. The training accuracy is slightly higher than the validation accuracy, which suggests the model fits the training data better than the validation data, but there's no overfitting or improvement.
- **Conclusions:**

1. Since the model isn't converging well due to the small batch size we need to increase the batch size to get better performance
2. Since the validation loss isn't improving due to the loss function not being sensitive enough to the actual performance metric, we need to try another loss function.

## 7. Possible Optimizations:

- You can experiment with different types of augmentations.
- Changing the loss function affects how the model learns from the data, so applying this change requires retraining from scratch or fine-tuning the model, depending on how you want to proceed.
- Increasing the gradient accumulation steps.
- Use another CNN architecture.

## References:


1. Pytorch Documentation:
2. Pytorhc transform:
3. PEP8:
4. Training with Pytorch
5. Training with pytorch
6. Understanding RESNET
7. Fine-Tuning github repo