Deep Learning on Chest X-Ray Images for Classification of Thoracic Diseases

Andrew Moy, Samuel Villarreal, & Xiao Xu

DSAN-6600-01-Fall-2025

Professors: Dr. Hickman & Benjamin Houghton

December 8, 2025

Abstract

This Deep Learning project develops a modular Python pipeline to automate the classification of thoracic diseases using frontal chest X-ray images. Our study aims to replicate and enhance the current study analysis made by the National Institute of Health Clinical Center (NIHCC) in 2017, utilizing modern Neural Network architectures. The programming framework integrates weighted binary cross-entropy and evaluation metrics such as F1, precision, recall, and ROC for our multilabel classification setting. Overall, our project demonstrates an accessible, open-source model designed to support further advances in the healthcare industry for thoracic disease detection and embraces the opportunity for adoption of Deep Learning models in the healthcare industry.

Background & Motivation

With the rapid advancement of technology, the healthcare industry has been able to capture and analyze more data than ever before. The most commonly implemented clinical practice is X-rays, which are a form of electromagnetic radiation that penetrates tissue at varying degrees. This type of study enables medical staff to visualize and interpret the internal structure of the human body without the need for invasive intervention, particularly on dense tissues such as bones and teeth. This aims to help diagnose fractures, infections, and lung conditions.

Modern hospitals have been able to capture vast amounts of data images of thorax diseases, and with the help of machine learning and deep learning models, we now have the possibility of streamlining the automation process to evaluate and identify common thorax diseases faster than ever before. In this study, we analyze 112,120 frontal-view X-ray images of 30,805 unique patient records made available by the U.S. NIHCC to identify the 14 most common thorax diseases: Atelectasis, Cardiomegaly, Effusion, Infiltration, Mass, Nodule, Pneumonia, and Pneumothorax. For reference, most of the available information on the web and from health organization sources, collectively estimate that thoracic diseases account for around 3.1 million deaths annually in the United States alone, representing about one-third of all deaths nationwide.

The objective of this project is to develop a well-documented Python framework that applies advanced deep learning models to accurately classify the aforementioned thoracic conditions. We evaluate and compare Convolutional Neural Networks (CNNs) and Transformer-based architectures to determine the most effective approach for this task. The resulting tool is designed to support researchers and healthcare professionals by providing a free, accessible, and adaptable open-source model for identifying common thoracic findings from chest X-ray images.

Methods

A fully modular and extensible framework was implemented in Python and PyTorch to develop a deep learning model for the analysis of X-rays. The system consists of four core components: data preprocessing, model architecture, training and evaluation pipelines, and a centralized configuration module that governs all experimental settings. This section outlines the full implementation of these components.

1. Data Processing Pipeline

All data preprocessing procedures are implemented in the ‘process_data.py’ and ‘preprocessing.py’ Python files. The pipeline is designed to ensure consistency, flexibility, and compatibility with multiple model architectures. The functions were designed to adhere to the Single Responsibility Principle (SRP), which improves the modularity, reusability, and readability of the codebase. The descriptions of the specific functions can be found in the project’s GitHub repository and in Appendix A.

All code is reliant on the consistency of the structure provided by Wang et al., shown in the tree structure below.

```
data/
└── images/
    ├── images_001/
```

```
        └── images/
            ├── 00000001_000.png
            ...
            └── 0000000n_00n.png
    ├── images_002/
        └── images/
            ├── ...
...
    └── images_012/
        └── images/
            └── ...
```

1.1 Processing Data

A module was created to process and structure the data into a format suitable for deep learning, specifically designed for the specific dataset provided. Most functions are necessary for every model configuration and training run. Tasks include dividing the raw data into train, validation, and test sets, converting image labels from strings to multi-hot encodings, and miscellaneous data manipulations specific to the format of this dataset.

1.2 Preprocessing Data

This module was created to manipulate the raw image data prior to training, and is designed to be used a single time after downloading the raw data. Its purpose is to make any necessary image/data manipulations prior to training a model, eliminating the need to make them on the fly while training.

The first preprocessing task involves resizing images. In order to be compatible with the pretrained weights of ResNet-50, each of the x-ray images must be an RGB image and of size 224 by 224 pixels. Each image was converted to this format, which was kept consistent across all trained models.

Data augmentation was also performed during image preprocessing. The number of images was doubled from 112,120 to 224,240 by duplicating each image and reflecting across the y-axis. Many variations of data augmentation, such as rotations and translations, would have been effective for X-ray images, but were excluded from this study due to a lack of computational power, as it would have been unfeasible to train on a dataset expanded by several-fold.

2. Model Architecture

The framework supports three deep learning architectures, enabling direct performance comparison across convolutional and transformer-based approaches. All models are implemented in the 'architectures.py' module, which instantiates the chosen model. The project supports two convolutional neural network (CNN) based models and one transformer-based model. Transfer learning of a pretrained, publicly available model (ResNet-50, ViT-B-16) is used for both the CNN and the transformer, as well as an additional custom CNN.

2.1 Transfer Learning CNN

This model utilizes ResNet-50, a convolutional neural network (CNN) pretrained on ImageNet, as the baseline architecture. The final classification layer is replaced with an output layer corresponding to the 14 thoracic disease classes. Freezing of layers is configurable through the global settings interface, and includes three options: freezing of all ResNet-50 pretrained layers, freezing of all ResNet-50 pretrained layers except for the last layer, and no freezing. These variations have 28,686 trainable parameters, 14,993,422 trainable parameters, and 23,536,718 trainable parameters, respectively. The final dense layer maps to only 14 disease classes rather than 1000 image classes, which subtracts about 2 million parameters from the existing model, and is consistent with the estimated 26 million parameters of ResNet-50.

2.2 Transfer Learning Transformer

This model employs a Vision Transformer (ViT) as the baseline architecture. ViT is also trained on ImageNet, and similarly to ResNet-50, the classifier head is adapted to support multi-label prediction of 14 classes. Also, similarly to ResNet-50, the model has freezable layers, with configurations enabling full freezing, partial freezing, or full fine-tuning of all parameters.

The classification head has 10,766 trainable parameters (embedding dimension * disease classes + disease classes = 768 * 14 + 14 = 10766), while the total parameter count is 85,809,422. The transformer is much slower than ResNet-50, primarily due to the attention mechanism, which runs at a quadratic speed relative to the image dimension. Therefore, while full fine-tuning was available, only full freezing was used in practice.

2.3 Custom Convolutional Neural Network

A custom CNN architecture was constructed using stacked convolutional, batch normalization, pooling, and fully connected layers. This design enables experimentation with lightweight deep learning models and allows us to explore image classification without the need of pretrained representations.

The custom CNN expects 224×224 RGB images and uses three convolutional stages, containing 32, 64, and 128 feature maps, respectively. Each kernel is of size 3 by 3. Between each convolutional layer, a ReLu activation function is used, as well as a 2 by 2 max pooling. The 128×7×7 tensor is flattened into a vector and fed into a dense neural network, with one hidden layer of 512 nodes and ReLU activation. This network maps the tensor to a length 14 vector, representing the 14 disease classes.

3. Configuration Management

Experimental settings are centralized in a single configuration file (‘config.py’) for simplicity, readability, and ease of use. All adjustable variables, hyperparameters, filenames, etc., are specified in this file, and any configurations not listed below cannot be customized. This design allows new models to be trained with minimal modifications to the codebase. Researchers can adjust the following options:

1. Model Architecture (ResNet-50, ViT, Custom CNN)
2. Batch size
3. Learning rate
4. Number of epochs
5. L2 Weight Decay
6. Model Name / Output Filename (for saving and loading of trained models)
7. Number of Folders
8. Number of image folders to use (up to 12)
9. Data Augmentation (Binary – uses images associated with selected image folders)

4. Training and Evaluation Pipeline

The training and evaluation pipeline consists of three main parts: training, evaluation, and loading. Following training, model parameters are saved and can be loaded for later evaluation/use. All functions are modular and generalizable to each of the three different deep learning architectures. The functions include automatic GPU detection and utilization.

4.1 Model Training

The model training function consists of a standard PyTorch training loop. It takes in a DataLoader object for training, validation, and test datasets, and iterates through batches and epochs while computing loss and using backpropagation for gradient-based parameter steps. The loss function used is binary cross-entropy with positive weighting to emphasize rare disease detection, and the optimizer used is Adam with L2 weight regularization. Training and validation loss are tracked across epochs, and validation stopping is triggered upon a single increase in validation loss.

Training time estimates are provided using the ‘tqdm’ Python package, and trained weights are automatically saved to an output folder with the filename and model type given by the user.

4.2 Model Evaluation

The model evaluation accepts a test dataset DataLoader object and the trained model, and outputs the correct image labels and the model's predictions. Due to the multilabel nature of the X-ray data (each image may have multiple diseases present), proper thresholds must be defined for each individual disease class to distinguish between negative and positive predictions. This was accomplished using F1 score maximization to maximize the general effectiveness of the model. It should be noted that F1 score maximization balances precision and recall equally, which may be improper for the field of medicine, where false positives are preferred over false negatives; however, F1 score maximization was used for simplicity.

These predictions are used to evaluate the model's effectiveness, primarily using precision, recall, F1 score, and ROC score. Additionally, precision recall curves and AUC-ROC curves are plotted for all 14 disease classes. All numerical scores and output plots are automatically saved for future reference.

4.3 Model Loading

The ability to save and load models was crucial to the project because model evaluation metrics continually changed throughout its development; the ability to test previously trained models quickly was necessary for the exploration of effective model configurations. The loading feature works by searching through the folder of trained weights created by the training function and automatically detecting a trained model given the model type and filename. If detected, the evaluation pipeline is run on the model, and the outputs are saved.

Results

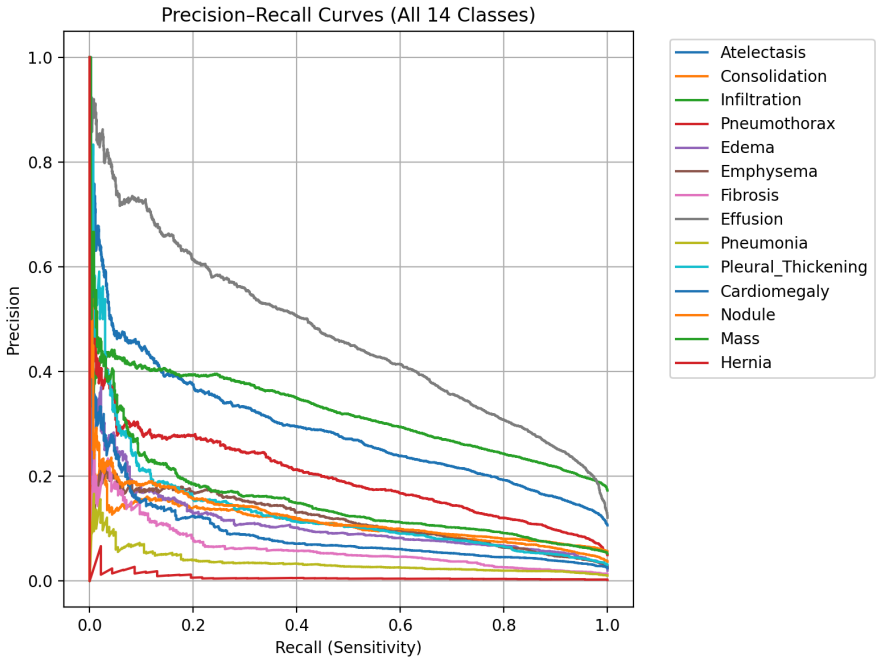

Figure 1

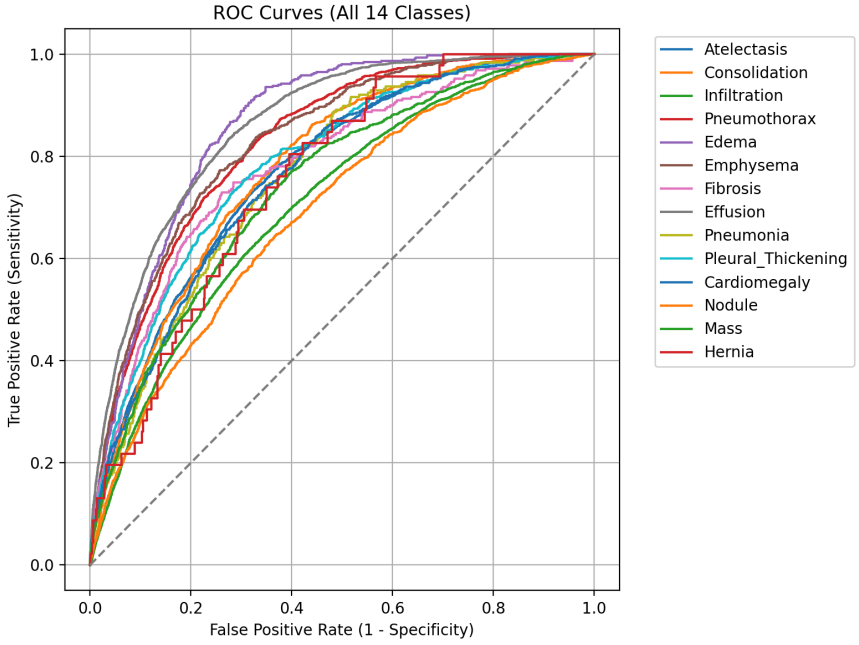

Figure 2

Figure 3

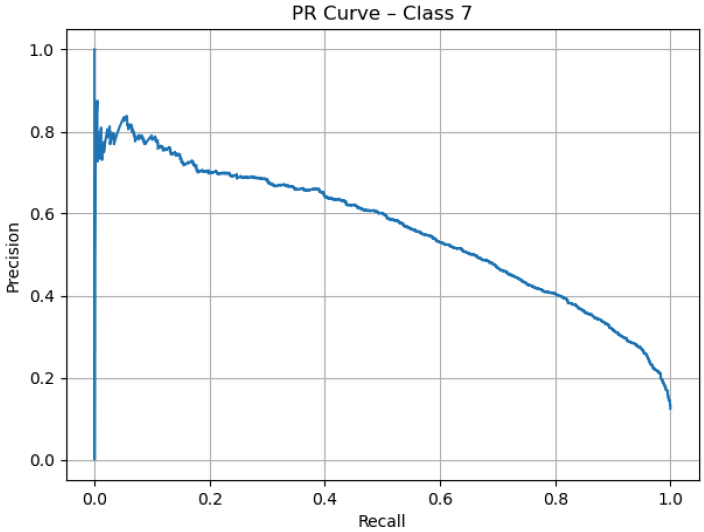

Figure 4

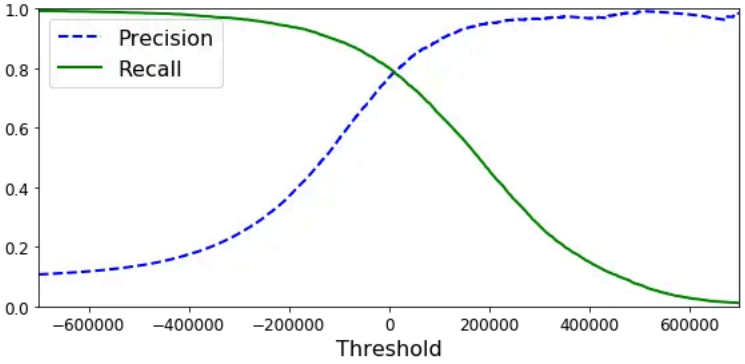

Figure 5

Our Model's (Best) Results so Far

Disease	Precision	Recall	F1
Atelectasis	0.31	0.45	0.37
Consolidation	0.13	0.37	0.19
Infiltration	0.33	0.46	0.38
Pneumothorax	0.31	0.51	0.38
Edema	0.17	0.43	0.24
Emphysema	0.32	0.36	0.34
Fibrosis	0.09	0.24	0.13
Effusion	0.44	0.55	0.49
Pneumonia	0.04	0.17	0.06
Pleural Thickening	0.14	0.30	0.19
Cardiomegaly	0.29	0.31	0.30
Nodule	0.19	0.31	0.24
Mass	0.28	0.35	0.31
Hernia	0.02	0.08	0.03

Figure 6

Discussion

1. Limitations

1.1 Overfitting

All three models were found to overfit to the training data within 2 to 3 epochs of training, but this problem was easily resolved through validation stopping and lowering of the learning rate from 1e-4 to 1e-5. Training rates of 1e-6 and 1e-7 were also tested. While these two alterations were the most impactful changes, L2 regularization was also added to reduce gradient explosion. Furthermore, models with lower parameter counts were trained by freezing layers and adjusting network sizes, although this did not have a significant impact on the overfitting problem.

1.2 Speed Optimization

The speed of model training was a significant issue to overcome. At first, the ResNet-50 model would take upwards of two hours for each epoch of training, and the ViT model even longer. This was unfeasible for the number of models needed, such that optimal hyperparameters could be found.

This training time was reduced through a combination of standard machine learning practices and design choices. Firstly, it was found that the program was loading too many images onto the GPU at once, causing a huge slowdown due to data transfer. This was corrected by lowering the batch size to 16 or 32 (depending on the machine being trained) and manually monitoring GPU performance. Furthermore, images were preprocessed rather than being processed during training. Built-in PyTorch features were utilized as well, such as increasing the number of workers on the PyTorch DataLoader and using the 'Channels Last Memory Format'.

Training time was also reduced by reducing parameter counts, a similar approach as to the overfitting problem. This was a mostly ineffective method, as the number of trainable parameters and layers was an essential aspect of testing different models, and directly impacted their performance. Finally, GPU precision was cut in half from 32 to 16 floating-point numbers to accommodate larger batch sizes and double the training speed. In the end, training time was cut from two hours to about seven minutes.

1.3 Data Imbalance

This X-ray image dataset contained an extreme case of data imbalance, which caused many problems in the development of an effective deep learning model. Due to the "real life" nature of the dataset, some diseases were very rare, while some were very common – for example, only 227 images of a hernia were provided, while 19,894 images of infiltration were provided. This imbalance caused a large bias towards common diseases. This effect was exacerbated by the multilabel nature of the data – x-rays could contain multiple (or zero) diseases, and therefore, the model contains 14 binary classification problems rather than 1 multiclass classification problem. As a result, rare diseases would have an incredible ratio of negatives to positives, such as hernia (111,893 negatives to 227 positives). The model found that always predicting rare diseases to be absent from the image was the most effective way of minimizing loss, while also flagging the presence of common diseases far too often.

Disease	Frequency
Atelectasis	11559
Consolidation	4667
Infiltration	19894
Pneumothorax	5302
Edema	2303
Emphysema	2516
Fibrosis	1686
Effusion	13317
Pneumonia	1431
Pleural Thickening	3385
Cardiomegaly	2776
Nodule	6331
Mass	5782
Hernia	227

Figure 7

Positive weighting was added to the loss function to counteract this effect, putting more emphasis on correctly labeled identification of rare diseases. The effect of this was difficult to balance, as multiplying by the direct ratio of negatives to positives was far too strong, reversing the data imbalance effect (common diseases were not guessed frequently enough), while using the logarithm or square root of the ratio was not strong enough.

2. Findings

2.1 Performance

Performance on all three models was not as effective as that of the original authors. The ResNet-50 model performed the best, with an average F1 score of 0.32, while the custom CNN scored a 0.22, and the ViT model a 0.15. Most of the 14 diseases performed better than random guessing when compared with their image frequency, which shows that the model is capable of detecting them somewhat accurately.

It is clear that the disease with the highest frequency of images had the greatest performance – atelectasis, infiltration, and effusion scored the highest of all the diseases in both

precision and recall, while rare diseases such as hernia scored extremely poorly. Additionally, performance improved with greater training time and the 2x data augmentation. It is for these reasons that we believe computation time was the primary limiting factor in model performance. While some data augmentation was done, a 2x decrease in speed could not be justified for every doubling of images. Given greater computing power and a large increase in data augmentation, we believe the performance would improve drastically.

Another factor contributing to the performance is that most of the transformer architecture parameters had to be frozen, and the number of parameters in the custom CNN was kept extremely minimal to support reasonable training times. These factors may also have contributed to ResNet-50 performing the best of the three models. While the F1 scores were poor, the ROC scores of almost all the models were very good. All three architectures consistently scored between 0.70 and 0.90, and the individual AUC-ROC plots for each disease provide curves representative of high performance. These high ROC scores suggest that the model was able to successfully rank true positives over true negatives, proving that it is effective in detecting disease features. The discrepancy between F1 score performance and ROC score performance may be attributed, once again, to data imbalance. With such an extremely high ratio of negatives to positives, it is difficult to find a proper threshold of probability to distinguish "true" predictions from "false" predictions.

2.2 Comparison to Wang et al.

The original authors vastly outperformed our model on precision, recall, and F1 score, but obtained similar ROC scores. This indicates that their models had far better separation and thresholding between true negatives and true positives, but performed similarly when ranking images in order. This further supports our findings that our model is effective, as well as our hypothesis that our model would perform far better with further training. s

Conclusion

Although our ResNet model did not perform as well as the original authors did in their own study, we demonstrate a workable end-to-end Python pipeline that can process this large amount of radiology images as well as complex thoracic disease information and obtain results that outperform a random guess. Our primary limitation was computational resource capacity

and a heavily concentrated class imbalance; nonetheless, these results and architecture show a promising potential for refinement, scalability, and later deployment. Together, we aim to continue investing time and effort in refining this model, implementing mechanisms to handle data imbalances across diseases, such as data augmentation, weight schemes, and decision thresholds, and leveraging the implementation of model deep learning architectures, such as transformers and hybrid models. Our objective remains the same: to build a useful, accurate, and completely free open source model to streamline thorax disease classification diagnostics available for all healthcare organizations and professionals across the globe.

Bibliography & References

National Institutes of Health Clinical Center. *NIH Chest X-ray Dataset of 14 Common Thorax Disease Categories: Dataset Description and Documentation*. NIH, n.d. https://nihcc.app.box.com/v/ChestXray-NIHCC/folder/36938765345.

Wang, Xiaosong, et al. "ChestX-ray8: Hospital-Scale Chest X-Ray Database and Benchmarks on Weakly-Supervised Classification and Localization of Common Thorax Diseases." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 3462–3471.

Appendix

GitHub Repository for Deep Learning on Chest X-Ray Images for Classification of Thoracic Diseases: https://github.com/andrewmoy3/DSAN_6600_final_project