HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

School of Information and Communication Technology

CHEST X-RAY DISEASES CLASSIFICATION

Introduction to Deep Learning

152474 - IT3320E

Semester 2024.1

Group 18	Student ID
Nguyễn Trọng Phương Bách	20225473
Vũ Minh Hiếu	20225494
Bùi Nguyên Khải	20225501
Trần Ngọc Quang	20225523
Nguyễn Việt Tiến	20225533

Supervisor

Professor. Nguyen Hung Son

Ph.D. Tran Viet Trung

Hanoi, December 2024

Contributions

Name	**Contributions**
Nguyễn Trọng Phương Bách	Report, Training, Slides
Vũ Minh Hiếu	Report, Training
Bùi Nguyên Khải	Report, Modeling, Data Preparation
Trần Ngọc Quang	Report, Modeling
Nguyễn Việt Tiến	GUI, Training

Abstract

Chest X-rays (CXRs) are an essential diagnostic tool for identifying a wide range of medical conditions. However, their interpretation is often limited by variability in radiologist expertise and the complexity of radiographic patterns.

Leveraging advancements in deep learning, this study explores the use of transfer learning with DenseNet-121 and ResNet-101 architectures to develop automated, multi-label classification models for chest X-ray interpretation. Using the CheXpert dataset, we implemented several techniques, including uncertainty label mapping, weighted random sampling, and data augmentation, to address challenges like class imbalance and label ambiguity.

Our models achieved competitive AUROC scores across most pathologies, demonstrating robust performance in diagnosing conditions such as cardiomegaly, pleural effusion, and edema. While the models outperformed baseline methods in some cases, they exhibited limitations in detecting subtle conditions like atelectasis, highlighting areas for future improvement.

This work contributes to the development of AI-assisted diagnostic tools, with the potential to enhance clinical workflows and improve diagnostic accuracy.

Contents

1 Introduction

The interpretation of chest X-rays (CXRs) plays a crucial role in diagnosing and managing a wide range of medical conditions, including pneumonia, heart failure, and lung diseases. Despite their clinical significance, the effective analysis of CXRs remains a challenging task due to the inherent complexity of radiological patterns and the reliance on expert radiologists for accurate interpretation [1]. Variability in expertise, subjective decision-making, and the growing demand for medical imaging in clinical practice further underscore the need for reliable, automated diagnostic tools.

The advent of deep learning has revolutionized the field of medical imaging by enabling the development of models capable of extracting meaningful features from raw image data. In particular, convolutional neural networks (CNNs) have emerged as a powerful tool for image classification tasks [2], achieving state-of-the-art performance in domains ranging from object detection to disease diagnosis. Leveraging these advancements, automated chest X-ray classification systems hold the potential to assist radiologists, improve diagnostic consistency, and enhance healthcare accessibility.

The CheXpert dataset, one of the largest publicly available collections of labeled chest radiographs, serves as a benchmark for developing and evaluating such models. This dataset provides over 220,000 labeled CXRs, with 14 clinically significant diseases, and incorporates uncertainty labels to reflect the nuances of radiological interpretation. However, challenges such as class imbalance, uncertainty in labeling, and the need for computational efficiency present significant obstacles to achieving optimal model performance.

In this project, we explore the use of transfer learning with two prominent CNN architectures, DenseNet-121 and ResNet-101, to address these challenges. By leveraging pretrained models and employing techniques such as uncertainty mapping, weighted sampling, and data augmentation, we aim to build robust multi-label classification models for chest X-ray interpretation. Our study investigates the effectiveness of these methods in improving classification performance and provides insights into the limitations and future directions for AI-driven radiological analysis.

This paper is organized as follows: Section 2 describes the dataset and preprocessing techniques. Section 3 outlines the methodology, including the architecture and training details of the selected models. Section 4 presents the experimental setup and results. Section 5 discusses the implications of our findings, and Section 6 concludes the paper with a summary of contributions and potential areas for further research.

2 Dataset and Preprocessing

2.1 CheXpert Dataset

Chest radiography is among commonly performed tests worldwide, used for diagnosis and management of conditions such as pneumonia, heart failure, and lung diseases. It plays a vital role in routine medical practice as well as large-scale screening programs. Interpreting chest X-rays however is challenging due to varying levels of expertise among radiologists. Such problems have motivated the development of automated tools to enhance diagnostic accuracy, reduce variance, and improve efficiency in clinical workflows. One significant barrier to advancing AI-powered radiograph interpretation has been the lack of large, well-annotated datasets with reliable ground truth for training and benchmarking machine learning models.

The CheXpert dataset was created to address this issue, offering a large-scale collection of 224,316 chest radiographs from 65,240 patients, labeled for 14 clinically relevant observations, including pneumonia, pleural effusion, and cardiomegaly. The CheXpert dataset assigns labels to chest X-rays given the observations in their associated radiology reports. This process consist of 3 steps: identifying mentions of medical conditions using a list of phrases curated by radiologists; classifying those as positive "1", negative "0", or uncertain "-1" based on the wording in the reports (e.g., "no evidence of" is negative, while "cannot exclude" is uncertain). Finally, combining the classifications to assign a label to each condition for the X-ray. For example, if an observation is mentioned positively at least once, it is labeled as positive, and if it is only described uncertainly, then an uncertain label is assigned [3]. The inclusion of uncertainty labels captures the nuances of radiology practice and allows for the development of models that handle ambiguous cases effectively.

To accommodate diverse research needs, CheXpert is available in two versions. The small version we use for this project provides the full set of labeled X-rays in a downsampled format (390×320 pixels) and a reduced size (11 GB), while the large version contains high-resolution images (440 GB)[4].

2.2 Data Preparation

2.2.1 Additional Metadata

The CheXpert small dataset also includes additional patient information, such as gender, age, and imaging details. Imbalances in these factors could potentially introduce bias into the model's performance. Consequently, it is essential to carefully analyze these variables to identify any factors that might influence model predictions or provide useful features that could enhance diagnostic accuracy.

Gender. The gender distribution in the dataset includes 132,764 samples from male patients and 90,883 from female patients (and one patient with unknown gender), reflecting a moderate imbalance. However, this difference is not substantial enough to significantly bias the model.

Age. The dataset includes a wide range of ages, from infants to elderly patients, with the most common ages being in the 60's and 90's. While there is some variation in age distribution, the dataset's diversity allows the model to capture a broad spectrum of chest X-ray characteristics.

Frontal/Lateral. A significantly larger number of frontal images (191,229) compared to lateral images (32,419), making frontal views more representative of the dataset. We decided to focus exclusively on frontal chest X-rays for our classification task, which simplifies the pre-processing pipeline and reduces potential variability introduced by combining different view types.

AP/PA. Most of the chest X-rays in the dataset are taken in the AP (161,759) or PA (29,453) orientations, with only a very small number of images in the LL and RL orientations (16 and 1, respectively). Given the dominance of AP and PA views, which are the most frequently used in clinical settings, the small number of LL and RL images is unlikely to affect model performance significantly.[5]

We opted to exclude additional metadata such as sex, age, view types, and orientation from our analysis, focusing solely on the chest X-ray images. This decision was based on the understanding that the diagnostic features of chest diseases are primarily visual and can be effectively captured directly from the radiographic data. By relying exclusively

on the images, we simplified the model input, reduced potential confounding factors, and maintained focus on the critical diagnostic patterns present in the chest X-rays.

2.2.2 Uncertainty Labels

We opted to handle uncertainty labels by mapping them to binary values (either 0 or 1) representing negative and postive labels due to its simplicity and proven effectiveness, as highlighted in the official CheXpert study. The paper demonstrates that binary mapping approaches, such as U-Zeros (mapping uncertainty to negative) and U-Ones (mapping uncertainty to postive), are straightforward to implement and yield competitive performance compared to more complex strategies. For example, U-Ones performed best for certain pathologies like Atelectasis and Edema, suggesting that uncertain labels often convey likely findings rather than true ambiguity.[3] Mapping uncertainty to binary labels avoids the computational overhead of multi-class uncertainty models and enables direct integration into standard training pipelines, making it an efficient and practical choice for this task.

After mapping uncertainty labels to 0, some rows in the U-Zeros dataset contained no positive labels, resulting in rows where all labels were set to 0. To address this, we assigned the 'No Finding' label a value of 1 for these cases. This approach aligns with the CheXpert dataset labeling methodology, where 'No Finding' is explicitly defined as the absence of all labeled pathologies. By setting 'No Finding' to 1 in these rows, we ensured that the dataset accurately reflects the intended semantics of this label while avoiding any ambiguity during model training.

2.2.3 Train, Validation and Test Set

The CheXpert dataset includes a training set of 223,414 images and a small validation set of only 234 images. Additionally, the official test set is not publicly available, making it challenging to robustly evaluate model performance on an unseen dataset.[3] To address these limitations, we opted to re-split the dataset to create a more balanced and representative validation and test split. For this, we applied stratification techniques tailored for multi-label data called **Second Order Iterative Stratification (SOIS)**, which is based on **Iterative Stratification (IS)** proposed in [6].

The Iterative Stratification (IS) algorithm distributes examples across subsets based on desired proportions. It begins by calculating the number of examples for each subset and label. The algorithm then iterates, selecting the label with the fewest remaining examples and assigning them to subsets that are most underrepresented for that label. If multiple subsets have the same need, the algorithm selects the one with the largest overall discrepancy from the desired distribution. Ties are broken randomly. After each assignment, the desired counts are updated. The process continues until all examples are distributed, which can occur in fewer or more iterations depending on the dataset's structure.

[7] proposed **The Second-Order Iterative Stratification (SOIS)** algorithm, which begins by calculating the required number of samples for each label pair in the dataset. It then iterates over all label pairs, starting with the pair that has the fewest samples, and assigns samples to the fold that requires the most of that label pair, breaking ties randomly. After assigning all samples for a label pair, the algorithm updates the counters and proceeds to the next label pair. Once all label pairs are distributed, the algorithm switches to distributing individual labels using the same procedure, essentially reverting to the **Iterative Stratification (IS)** method. Finally, after assigning positive labels, negative labels are randomly distributed to fulfill fold requirements. SOIS considers both label pairs and individual labels, ensuring balanced stratification across all folds. [7]

To ensure robust model evaluation and avoid overfitting, we chose to split the dataset into 80% for training, 10% for validation, and 10% for testing. Given the large size of the dataset, this 80/10/10 split is considered sufficient to maintain a representative distribution across all subsets while providing enough data for training, tuning, and final performance assessment. Additionally, the stratification techniques employed in the data splitting process ensure that both rare and frequent labels are well-represented across the splits, further enhancing the model's ability to generalize across various classes.

2.2.4 Class Imbalanced

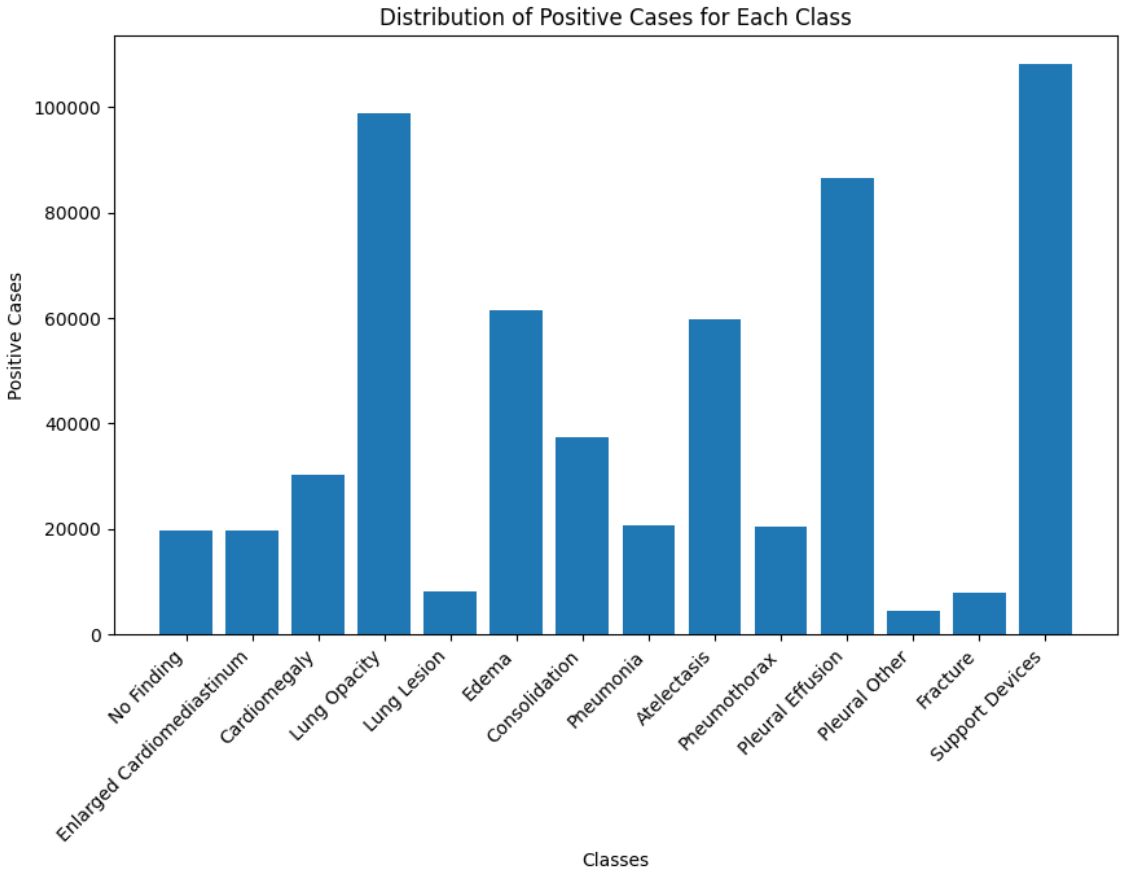

Figure 1: Distribution of Positive Cases for Each Class

The CheXpert dataset exhibits significant class imbalances, as evident from the distribution of positive cases for each class (see Figure 1). This imbalance poses challenges for the model's ability to learn and generalize effectively, as it may be biased towards the majority classes. To address this issue, we employed a weighted random sampling approach during the training process.

Specifically, we first calculated the count of positive cases for each class. We then computed the class weights as the inverse of the class counts, during training, the data loader will draw a batch of 32 samples with replacement, these samples will be randomly selected based on their probabilities (ground truth class weights).

By using this weighted random sampling approach, we ensured that the model was exposed to a more balanced representation of the classes during training. This helps the model focus on the minority classes without completely

ignoring the majority classes, which should improve its ability to learn the underlying patterns and generalize better to unseen data.

2.3 Data Augmentation

As explored in [8], data augmentation is a critical technique in medical image analysis, especially when working with deep learning models. The goal is to artificially increase the size of the training set and improve the generalization of the model by introducing transformations that mimic variations seen in real-world data. For medical images, certain augmentations can help prevent overfitting, especially when the dataset is limited or contains imbalances in class distribution.

Scaling. Scaling helps the model become invariant to small changes in image size. A 5% scaling, where the image size is slightly zoomed in or out, simulates variations in the size of medical structures that might appear in different imaging conditions. This ensures that the model does not become overly sensitive to the precise size of certain features, making it more robust to slight variations in real-world data. Scaling can be particularly helpful in medical images where objects like organs or anomalies may appear at different sizes due to varying patient anatomy or equipment settings.

Rotating. Rotating the image by up to 20 degrees helps the model become invariant to the orientation of medical images. While medical images are usually captured in a standard orientation, slight rotations in clinical settings are common, and models need to handle this variability effectively. In particular, for chest X-rays or other radiographic images, small rotational differences do not significantly change the diagnostic information. The 20-degree range is chosen to introduce moderate rotation while still preserving the anatomical context.

Shearing. Shearing transforms the image by slanting it along one axis (horizontal or vertical), simulating distortions that might occur due to the angle of image capture or patient positioning. A 5-pixel shear is a reasonable choice as it provides variability in the shape of the structures in the image without causing significant loss of information. For example, small shearing can be helpful for images like chest X-rays where slight distortions might happen due to differences in the positioning of the patient.

Translation. Translating images by 5% means shifting the image slightly along the X or Y axis. This operation helps prevent the model from becoming reliant on specific spatial locations of features in the image. It is particularly useful in medical imaging where critical features may appear in different locations within the image (e.g., lesions, organ boundaries) depending on the positioning of the patient during the scan. Translating by 5% ensures that the model learns to focus on relevant features regardless of their location.

Resize. Resizing the images to a standard size of 224x224 pixels ensures that the model receives a consistent input size, which is crucial for efficient training and comparison across different image sizes.

Normalization. Normalizing the images using the dataset's pixel mean and variance is a standard practice in deep learning to ensure that the inputs have consistent statistical properties. This ensures that the model trains more effectively, as it can focus on learning patterns rather than being affected by variations in pixel intensity. Normalization also helps speed up convergence during training by ensuring that the input data lies within a similar range across all images.

Certain augmentation methods were excluded to preserve the clinical integrity of the chest X-ray images. **Flipping** was avoided because it distorts asymmetrical anatomical structures, such as the heart, which are crucial for diagnosis. **Intensity modifications** (like contrast and brightness adjustments) could alter critical diagnostic features, leading to potential loss of important information. **Color modifications** were not used since chest X-rays are typically grayscale, and introducing color could distort pixel values in a way that is not representative of real medical data. Lastly, **random cropping** could result in the loss of vital diagnostic features if important regions are inadvertently removed, potentially hindering the model's ability to learn key patterns. These decisions ensure that the augmentation process maintains realistic, clinically relevant variations.[8]

3 Methodology

3.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) have become the go-to architecture for image classification tasks, including medical image analysis, due to their ability to automatically learn spatial hierarchies of features from raw image data. In our problem, where the goal is to classify chest X-ray images for various diseases, CNNs are particularly well-suited as they can effectively capture important visual patterns, such as textures, shapes, and structures, that are critical for accurate diagnosis.

Main Components of CNNs [9]

Convolutional Layer. The convolutional layer is the core building block of a CNN. It applies a series of filters (or kernels) to the input image, extracting local features such as edges, corners, and textures. These filters slide over the image, performing element-wise multiplication and summation to produce feature maps. This process allows the network to learn spatial hierarchies in the data.

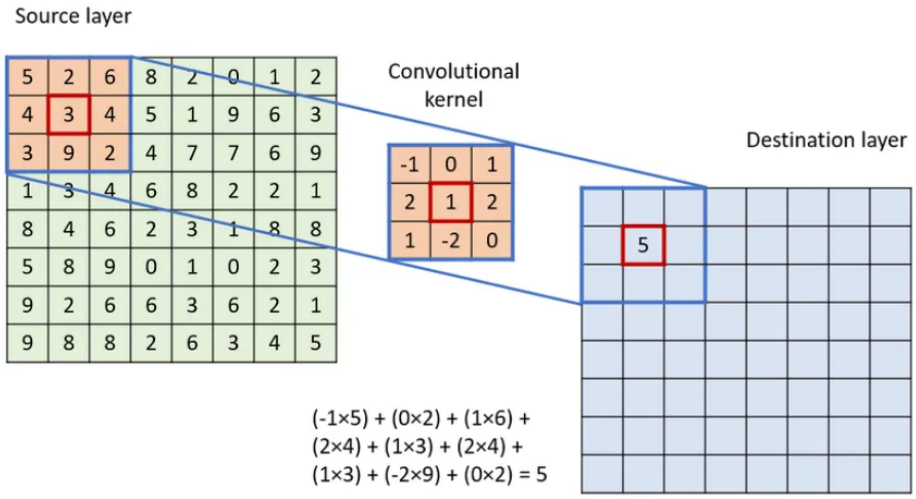

Figure 2: Convolutional operation.

$$f(x,y) * g(x,y) = \sum_m \sum_n f(m,n)\, g(x-m, y-n)$$

- $f(x,y)$: The input signal or feature map.
- $g(x,y)$: The convolution kernel (also known as the filter).
- m,n: Indices for summation, representing the positions in the input and kernel.

Activation Function (ReLU). After each convolution operation, an activation function is applied to introduce non-linearity into the model, enabling it to learn complex patterns. The Rectified Linear Unit (ReLU) is commonly used in CNNs. It replaces all negative values in the feature maps with zero, while leaving positive values unchanged.

$$f(x) = max(0,x)$$

This helps the network learn faster, prevents the vanishing gradient problem, and introduces non-linearity, allowing the model to capture more complex relationships in the data.

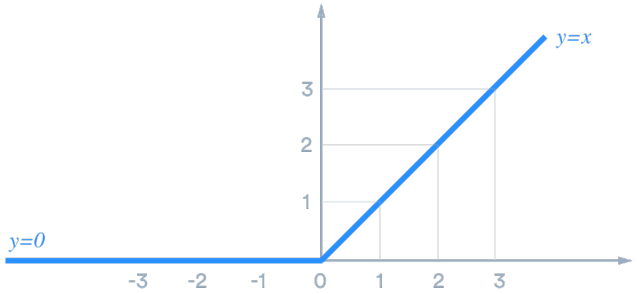

Figure 3: ReLu function

Pooling Layer. Pooling layers, such as max pooling

$$h_i^n(x,y) = \max_{\bar{x},\bar{y} \in N(x,y)} h_i^{n-1}(\bar{x},\bar{y})$$

or average pooling

$$h_i^n(x,y) = \frac{1}{K} \sum_{\bar{x},\bar{y} \in N(x,y)} h_i^{n-1}(\bar{x},\bar{y})$$

- $h_i^n(x,y)$: The output value of the pooled feature map at position (x,y) for channel i at layer n.
- $N(x,y)$: The neighborhood of pixels (e.g., a 3×3 window) centered around position (x,y). This defines the local region over which the pooling operation is performed.
- $\bar{x},\bar{y}$: Positions within the neighborhood $N(x,y)$.
- $h_i^{n-1}(\bar{x},\bar{y})$: The input feature map value at position $(\bar{x},\bar{y})$ in channel i at the previous layer $n-1$.

, are used to downsample the feature maps produced by the convolutional layers. This reduces the spatial dimensions of the data and helps the network become more invariant to small translations of the input image. Pooling layers also reduce the computational complexity and prevent overfitting.

Normalization Layer. Normalization layers, such as Batch Normalization, help stabilize the training process by normalizing the output of the previous layers.

For a layer of the network with d-dimensional input, $x = (x^{(1)}, x^{(2)}, ..., x^{(d)}$. Each dimension of its input is then normalized as:

$$\hat{x}_i^{(k)} = \frac{x_i^{(k)} - \mu_B^{(k)}}{\sqrt{\left(\sigma_B^{(k)}\right)^2 + \varepsilon}}, \text{ where } k \in [1,d] \text{ and } i \in [1,m]$$

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{and} \qquad \sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$$

This can improve convergence and prevent overfitting, especially in deeper networks.

Fully Connected Layer. After several convolutional and pooling layers, the high-level features learned by the network are flattened and passed through one or more fully connected layers. These layers perform classification by mapping the learned features to the output classes. In the context of chest X-ray disease classification, the output layer would typically have neurons corresponding to the presence or absence of specific diseases.

3.1.1 Residual Neural Network

3.1.1.1 Introduction

Residual Networks (ResNet) were introduced in [10] as a solution to the problem of training very deep neural networks. Traditional deep networks suffer from the vanishing gradient problem, where gradients become very small as they propagate back through multiple layers, making it difficult for the network to learn. ResNet addresses this issue through the introduction of **residual connections**, which allow gradients to flow more easily through the network by skipping one or more layers. Additionally, ResNet's strong generalization capabilities help prevent overfitting, while pretrained models on large datasets can be fine-tuned for specific multilabel tasks, reducing training time and data requirements. Its scalability and flexibility make it ideal for handling the complexities of multilabel classification.

This is achieved by using residual blocks. Consider a function $H(\mathbf{x})$ that needs to be approximated by a neural network. Instead of estimating parameters for $H(\mathbf{x})$, the stacked nonlinear layers approximate another mapping $F(\mathbf{x}) = H(\mathbf{x}) - \mathbf{x}$. The original mapping is thus recast into:

$$H(\mathbf{x}) = F(\mathbf{x}) + \mathbf{x}.$$

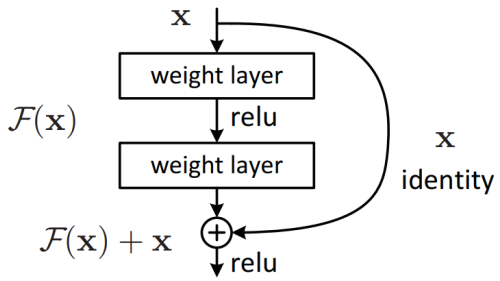

Figure 2. Residual learning: a building block.

Figure 4: Residual Block Structure.

The Residual Network is essentially built by stacking these residual blocks. One prominent advantage of this

method is the mitigation of the vanishing gradient problem. If the output of the l^{th} layer serves as the input to the $(l+1)^{\text{th}}$ layer (assuming no activation function between blocks), then the input at the $(l+1)^{\text{th}}$ layer becomes:

$$\mathbf{x}_{l+1} = F(\mathbf{x}_l) + \mathbf{x}_l.$$

By applying this formulation recursively, we obtain:

$$\mathbf{x}_L = \mathbf{x}_l + \sum_{i=l}^{L-1} F(\mathbf{x}_i),$$

where L is the index of the final residual block, and l is an earlier block. As shown in the equation, at any layer, the signal from the shallower layers always contributes to the deeper layers.

To analyze the effect of residual blocks on backpropagation, consider the partial derivative of a loss function K with respect to some residual block input $\mathbf{x}_l$:

$$\begin{aligned}
\frac{\partial K}{\partial \mathbf{x}_l} &= \frac{\partial K}{\partial \mathbf{x}_L} \frac{\partial \mathbf{x}_L}{\partial \mathbf{x}_l} \\
&= \frac{\partial K}{\partial \mathbf{x}_L} \left(1 + \frac{\partial}{\partial \mathbf{x}_l} \sum_{i=l}^{L-1} F(\mathbf{x}_i) \right) \\
&= \frac{\partial K}{\partial \mathbf{x}_L} + \frac{\partial K}{\partial \mathbf{x}_L} \frac{\partial}{\partial \mathbf{x}_l} \sum_{i=l}^{L-1} F(\mathbf{x}_i).
\end{aligned}$$

This formulation suggests that the gradient computation of a shallower layer $\frac{\partial K}{\partial \mathbf{x}_l}$ always includes a direct term $\frac{\partial K}{\partial \mathbf{x}_L}$. Even if the gradients of the $F(\mathbf{x}_i)$ terms are small, the total gradient $\frac{\partial K}{\partial \mathbf{x}_l}$ resists vanishing due to the added direct contribution of $\frac{\partial K}{\partial \mathbf{x}_L}$.

3.1.1.2 ResNet-101 Architecture

For this project, we selected **ResNet-101** as one of the models to be studied. This model provides a balance between accuracy and computational cost.

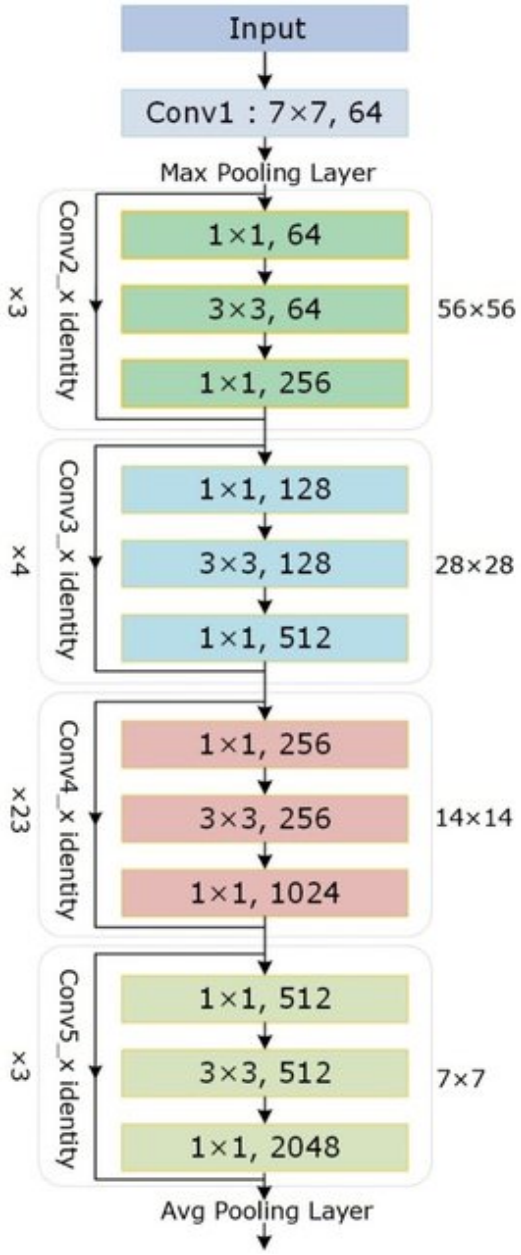

Figure 5: The architectural view of ResNet-101 (Image credit: ResearchGate).

The architecture of ResNet-101, as shown in Figure 5, consists of the following components:

Input and Initial Layers

- **Conv1:** A single convolutional layer with a 7×7 kernel and 64 filters, followed by a Max Pooling layer.
- **Output size:** 56×56.

Residual Blocks

The architecture is divided into four main stages (Conv2_x, Conv3_x, Conv4_x, Conv5_x), each containing multiple residual blocks with 1×1, 3×3, and 1×1 convolutions. The shortcut connections enable residual learning.

- **Conv2_x:** 3 residual blocks, each consist of:

- 1×1 BN-ReLU-Conv with 64 filters (reduces dimensions),
- 3×3 BN-ReLU-Conv with 64 filters (spatial features),
- 1×1 BN-ReLU-Conv with 256 filters (restores dimensions).

Output size: 56×56.

- **Conv3_x:** 4 residual blocks, each consist of:
  - 1×1 BN-ReLU-Conv with 128 filters,
  - 3×3 BN-ReLU-Conv with 128 filters,
  - 1×1 BN-ReLU-Conv with 512 filters.

  Output size: 28×28.
- **Conv4_x:** 23 residual blocks, each consist of:
  - 1×1 BN-ReLU-Conv with 256 filters,
  - 3×3 BN-ReLU-Conv with 256 filters,
  - 1×1 BN-ReLU-Conv with 1024 filters.

  Output size: 14×14.
- **Conv5_x:** 3 residual blocks, each consist of:
  - 1×1 BN-ReLU-Conv with 512 filters,
  - 3×3 BN-ReLU-Conv with 512 filters,
  - 1×1 BN-ReLU-Conv with 2048 filters.

  Output size: 7×7.

Global Average Pooling and Output

After the final residual block (Conv5_x), the feature map undergoes Global Average Pooling, reducing it to a single value per feature map. This is followed by a fully connected layer for classification.

The backbone of the model is pretrained on the ImageNet dataset, which a dataset consists of over 1.2 million images across 1,000 classes, ResNet101 achieves top-tier accuracy. The original training for ResNet101 on ImageNet took several days on multiple GPUs, utilizing techniques like data augmentation and stochastic gradient descent. It demonstrated remarkable performance, with a top-1 accuracy of approximately 76.4% and top-5 accuracy of around 93.5%, making it a popular choice for transfer learning in a wide range of computer vision tasks.

3.1.2 Densely Connected Neural Network

3.1.2.1 Introduction

DenseNet, introduced in [11], is another innovative architecture which further embrace the idea of creating short paths from early layers to later layers. DenseNet introduces the concept of dense connections, where each layer receives inputs from all preceding layers. This approach also inherits the mitigation of vanishing gradient problem from ResNet.

In contrast to ResNet, DenseNet concatenates the output of each layers with the input of all subsequent layers instead of addition, creating a dense connectivity pattern.

$$\mathbf{x}_l = concat(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{l-1})$$

DenseNet utilizes dense blocks, where each block contains several convolutional layers. In each dense block, the input to each layer is a concatenation of the feature maps from all previous layers within the block. This ensures that each layer has access to a richer set of features, making the network more efficient in terms of both parameter usage and learning capacity.

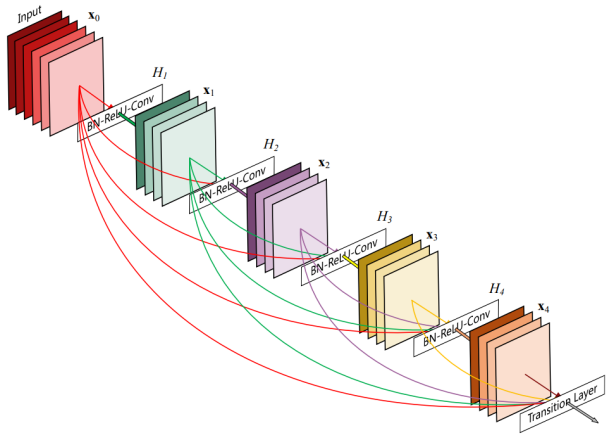

Figure 6: A 5-layer dense block with a growth rate of k = 4. Each layer takes all preceding feature-maps as input.

3.1.2.2 DenseNet-121 Architecture

Another model presented in this project is DenseNet-121, a 121-layer convolutional neural network based on the DenseNet architecture. DenseNet-121 balances model complexity and performance, providing a powerful architecture that avoids the pitfalls of vanishing gradients and excessive parameterization.

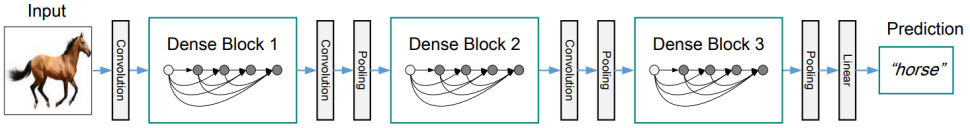

Figure 7: A deep DenseNet with three dense blocks.

Input and Initial Layers

- **Conv1:** A single convolutional layer with a 7×7 kernel and 64 filters, followed by a Max Pooling layer.

- **Output size:** 224×224.

Dense Blocks

The architecture is divided into four main dense blocks (DenseBlock1, DenseBlock2, DenseBlock3, DenseBlock4), each containing multiple layers of dense connections, where each layer receives input from all previous layers. Between the dense blocks, there is a transition layer that performs down-sampling.

- **DenseBlock1:** 6 layers, each consisting of:
  - A 1×1 BN-ReLU-Conv with 32 filters (bottleneck layer),
  - A 3×3 BN-ReLU-Conv with 32 filters (spatial features),
  - Concatenation of the output with the input from all previous layers.

  Output size: 112×112.
- **Transition Layer 1:** A 1×1 convolution with 128 filters, followed by Average Pooling. Output size: 56×56.
- **DenseBlock2:** 12 layers, each consisting of:
  - A 1×1 BN-ReLU-Conv with 64 filters (bottleneck layer),
  - A 3×3 BN-ReLU-Conv with 64 filters (spatial features),
  - Concatenation of the output with the input from all previous layers.

  Output size: 56×56.
- **Transition Layer 2:** A 1×1 convolution with 256 filters, followed by Average Pooling. Output size: 28×28.
- **DenseBlock3:** 24 layers, each consisting of:
  - A 1×1 BN-ReLU-Conv with 128 filters (bottleneck layer),
  - A 3×3 BN-ReLU-Conv with 128 filters (spatial features),
  - Concatenation of the output with the input from all previous layers.

  Output size: 28×28.
- **Transition Layer 3:** A 1×1 convolution with 512 filters, followed by Average Pooling. Output size: 14×14.
- **DenseBlock4:** 16 layers, each consisting of:
  - A 1×1 BN-ReLU-Conv with 256 filters (bottleneck layer),
  - A 3×3 BN-ReLU-Conv with 256 filters (spatial features),
  - Concatenation of the output with the input from all previous layers.

  Output size: 14×14.

Global Average Pooling and Output

After the final dense block (DenseBlock4), the feature map undergoes Global Average Pooling, reducing it to a single value per feature map. This is followed by a fully connected layer for classification.

Output size: 1×1 (after Global Average Pooling), followed by classification layer (e.g., softmax for multi-class classification).

The backbone of DenseNet-121 is pretrained on the ImageNet dataset. DenseNet-121 demonstrated impressive results, with a top-1 accuracy of approximately 74.9% and top-5 accuracy of around 92.3%. Its ability to achieve high accuracy with fewer parameters compared to other architectures makes it an excellent choice for transfer learning and fine-tuning in multilabel image classification.

3.1.3 Transfer Learning

Instead of training the entire network from scratch, which takes a lot of time and resources, transfer learning is the most efficient solution for deep learning problems with limited computational power and data. Transfer learning is a machine learning technique where a model developed for one task is reused as the starting point for a model on a second related task. This technique is often used if the problem domain shares some similarities with the pretrained model's domain, since the model does not have to learn from the beginning all the low-level structures that occur in datasets, and only needs to focus on building the higher-level structures with materials from lower layers. In order to perform transfer learning, a pretrained model on large datasets is often utilized, which provides a strong starting point from all the details and patterns acquired from the initial learning problem (in our case, the ImageNet task of classifying 1000 different object classes from over a million images[cite]). Because of that, the training process with a pretrained model base often achieves better results and serves as a good foundation for the fine-tuning step to adapt the model's general knowledge to specific domains.

For our problem, we leverage available DenseNet and ResNet architectures, which are pretrained on the ImageNet dataset, with slight modifications to the final layer. This is a necessary step, as the original layer of these two models is specifically designed for predicting the 1000 classes of the original problem, whereas our problem requires the classification of 14 different diseases:

- For DenseNet-121 structure: Replacing the fully connected layer after the Global Average Pooling layer on top of the network's DenseBlock4.
- For ResNet-101 structure: The same as above, with the Global Average Pooling layer after the final residual block ($Conv5_x$).

To avoid damaging well-trained layers of the original models, we performed a technique called Layer Freezing, where some layers of a pretrained model are "frozen" and ensure that their parameters are not updated during back-propagation. By blocking the gradient flow across the model at lower levels, only the highest layers are allowed to learn the patterns, and they are obliged to use generic features for the task. This process also helps training become much faster, as the backpropagation calculations require more time than the forward step.

After 3 epochs, we unfreeze the top layers of the model (DenseBlock4 for DenseNet and Layer4 for ResNet), and lower the learning rate and weight decay by a factor of 0.1, but not less than 1e-6.

3.2 Training

3.2.1 Loss Function

In our multi-label classification task, we use the Binary Cross-Entropy with Logits Loss as the loss function. This loss function is well-suited for problems where each class is treated independently, and the goal is to predict probabilities for multiple binary labels. For instance, in our chest X-ray disease classification task, each disease is treated as a separate binary classification problem, where the model predicts the likelihood of the presence or absence of each condition.

During training, the model outputs raw scores (logits) for each label, which are passed through the sigmoid function to produce probabilities between 0 and 1. The Binary Cross-Entropy with Logits Loss then compares these probabilities with the true labels for each class, calculating the error for every label independently. The total loss is computed as the average of the binary cross-entropy losses across all labels and samples in a batch. This loss value is then backpropagated through the network to update the model's parameters, enabling it to minimize the error and improve its predictions over successive iterations.

Binary Cross-Entropy with Logits Loss (BCEWithLogitsLoss) measures the difference between the predicted probabilities and the true labels for a binary classification task. By combining Binary Cross-Entropy with the Sigmoid function into one expression, we make use of the log-sum-exp trick for numerical stability. It is defined as:

$$l(y,\hat{y}) = -\frac{1}{N}\sum_{i=1}^{N}\Big(y_i \log(\sigma(\hat{y}_i)) + (1-y_i)\log(1-\sigma(\hat{y}_i))\Big),$$

- y_i is the true label for the i-th sample (1 for positive and 0 for negative),
- $\hat{y}_i$ is the predicted probability for the i-th example,
- N is the total number of samples.
- $\sigma()$ is the Sigmoid function which takes the raw model outputs (logits) to map them to probabilities in the range [0, 1].

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

The Binary Cross-Entropy with Logits Loss is particularly effective for multi-label classification tasks because each label is treated independently, allowing for overlapping labels (e.g., a patient may have multiple diseases simultaneously). It is also robust to imbalanced datasets, as each label contributes separately to the loss calculation. The combined sigmoid activation and BCE ensure numerical stability and smooth gradients, which are essential for training deep neural networks.

3.2.2 Optimizer

This project uses **AdamW**, which is a variant of Adam (Adaptive Moment Estimation) that integrates a regularization technique called weight decay (introduced in [12]). Classic Adam combines the ideas of momentum optimization and gradient accumulation by keeping track of exponentially decaying average of past gradients (g_t) and past squared

gradients (g_t^2) [13]. AdamW takes the idea a step further by the addition of weight decay, which aims to reduce the size of the model's weights at each training iteration by multiplying them by a decay factor, such as 0.999. In practice, the weight decay can be tuned as a hyperparameter to find the optimal optimizer for the models.

Algorithm 2 Adam with L_2 regularization and Adam with decoupled weight decay (AdamW)

given $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, \lambda \in \mathbb{R}$
initialize time step $t \leftarrow 0$, parameter vector $\boldsymbol{\theta}_{t=0} \in \mathbb{R}^n$, first moment vector $\boldsymbol{m}_{t=0} \leftarrow \boldsymbol{0}$, second moment vector $\boldsymbol{v}_{t=0} \leftarrow \boldsymbol{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
repeat
  $t \leftarrow t + 1$
  $\nabla f_t(\boldsymbol{\theta}_{t-1}) \leftarrow \text{SelectBatch}(\boldsymbol{\theta}_{t-1})$ ▷ select batch and return the corresponding gradient
  $\boldsymbol{g}_t \leftarrow \nabla f_t(\boldsymbol{\theta}_{t-1}) + \lambda \boldsymbol{\theta}_{t-1}$
  $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t$ ▷ here and below all operations are element-wise
  $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t^2$
  $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t / (1 - \beta_1^t)$ ▷ β_1 is taken to the power of t
  $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t / (1 - \beta_2^t)$ ▷ β_2 is taken to the power of t
  $\eta_t \leftarrow \text{SetScheduleMultiplier}(t)$ ▷ can be fixed, decay, or also be used for warm restarts
  $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta_t \left(\alpha \hat{\boldsymbol{m}}_t / (\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon) + \lambda \boldsymbol{\theta}_{t-1} \right)$
until *stopping criterion is met*
return optimized parameters $\boldsymbol{\theta}_t$

Figure 8: Adam with L_2 regularization and AdamW

The optimizer plays a crucial role in training deep learning models by updating the model's parameters to minimize the loss function. During each iteration, AdamW adjusts the weights based on the gradients computed from the loss function. The optimizer's learning rate, along with other hyperparameters like weight decay, governs how quickly the model learns and prevents issues such as overfitting or slow convergence. AdamW's adaptive nature makes it particularly effective for complex models with a large number of parameters, like those used in image classification tasks, by providing efficient parameter updates and regularization.

3.2.3 Scheduler

To adjust the learning rate during training, we decided to use Pytorch's **ReduceLROnPlateau** [14], which is a scheduler that reduces the value of the learning rate if the monitored metric stops improving (reaching a plateau) for a specified number of epochs. This schedule is a suitable choice for many deep learning problems, since models often benefit from reducing the learning rate by a factor of 2-10 once learning stagnates. A large learning rate at the beginning of the training process helps the loss reduce faster and escape local optima, but after that, it tends to jump back and forth around the optimal value, so reducing the learning rate at this point allows further convergence of the training loss. The schedule also has some notable parameters that can be tuned for controlling fine-tuning process of learning rate for optimal performance, including:

- monitor: Metric to track (training loss).
- factor: Factor by which the learning rate will be reduced (usually 0.1).
- patience: The number of allowed epochs with no improvement after which the learning rate will be reduced. Higher patience gives the training loss more chances to escape 'bad' local optima, while lower patience limits

the number of jumps of the loss over 'good' optima. We choose a patience value of less than 5 since 5 is the early stopping patience, allowing us to balance the opportunity for exploration with preventing excessive training time.

3.2.4 Evaluation Metric and Early Stopping

Mean Area Under the Receiver Operating Characteristic (mAUROC) curve is used as the primary evaluation metric. The AUROC score measures the ability of the model to distinguish between the positive and negative classes for each label independently. It is calculated by plotting the True Positive Rate (TPR) against the False Positive Rate (FPR) at various classification thresholds.

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad , \quad \text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

- TP is the number of true positives (correctly predicted positive instances),
- FN is the number of false negatives (actual positives incorrectly predicted as negative).
- FP is the number of false positives (actual negatives incorrectly predicted as positive),
- TN is the number of true negatives (correctly predicted negative instances).

The mean AUROC score aggregates the AUROC scores across all labels, providing an overall measure of the model's performance. For medical applications like chest X-ray disease classification, where each disease is treated as a separate binary classification problem, the mean AUROC score is particularly appropriate because it accounts for imbalanced datasets by focusing on the ranking of predictions rather than the raw class distributions, and it evaluates the model's ability to detect diseases independently, which is critical in multi-label settings where a patient may have multiple conditions simultaneously. A higher mean AUROC score indicates better performance, with a value of 1.0 representing perfect classification and 0.5 representing random guessing.

Early stopping is a regularization technique used to prevent overfitting during model training. In our project we monitor the mean AUROC score on the validation set and stops training if the performance does not improve for 5 consecutive epochs. This allows the model some flexibility to recover from temporary fluctuations in performance while avoiding unnecessary overfitting to the training data and wasteful computational resources. We then save the model with the best validation mAUROC score, ensuring that the model generalizes well to unseen data. By monitoring the AUROC score, which reflects the model's classification quality across all labels, early stopping ensures optimal performance without excessive training.

3.2.5 Hyperparameter Tuning

For searching the optimal hyperparameters for our models, we utilize **Bayesian Hyperparameter tuning** introduced in [15], which is an advanced tuning method based on probability and statistics. Unlike grid search with exhaustive approach or random search (with uniform distribution), Bayesian optimization builds a probabilistic model to predict the performance of hyperparameter combinations and iteratively refines the search to focus on promising areas of the hyperparameter space. Each iteration of the process includes:

- Build a "surrogate" probability model of the objective function, which is cheaper than the actial function, that maps hyperparameters to a probability of a score on the objective function:

$$P(score|hyperparameter)$$

using previously evaluated hyperparameters and their corresponding performance. The model used in this step is often Gaussian Processes or Tree-structured Parzen Estimators (TPE).

- Then, a selection function determines the next set of hyperparameters to evaluate based on the surrogate model's predictions. The most common choice of criteria is Expected Improvement:

$$\mathbb{E}[\mathscr{L}_{y^*}(x)] = \int_{-\infty}^{y^*} (y^* - y)\, p(y|x)\, dy$$

which chooses points with the greatest expected improvement over the current best score, where:

  - y^* is a threshold value of the objective function.
  - x is the proposed set of hyperparameters.
  - y is the actual value of the objective function using hyperparameters x.
  - $p(y|x)$ is the surrogate probability model expressing the probability of y given x.

- After that, the model is trained with the suggested hyperparameters, and the performance is recorded with true objective function (usually the actual metrics used in the validation step, mAUROC for our case).
- The process repeats until a stopping criterion is met (iteration limit, time budget,...).

By focusing on promising regions of the hyperparameter space, Bayesian Hyperparameter tuning significantly reduce the number of evaluations required compared to random or grid search, and is able to find better hyperparameter configurations for the models.

3.3 Optimal Threshold

In our multi-label classification task, determining an optimal threshold for each label is crucial to converting the predicted probabilities into binary decisions (presence or absence of each condition). The threshold defines the cutoff value above which the model classifies a condition as positive. Since our problem involves imbalanced data, a global threshold of 0.5 may not be ideal for all labels. Instead, thresholds can be determined based on metrics such as the Youden's Index (J), which balances sensitivity (True Positive Rate, TPR) and specificity (1 - False Positive Rate, FPR). This approach is widely used in medical diagnostics and is detailed in [16]. The formula for Youden's Index is:

$$J = \text{TPR} + (1 - \text{FPR}) - 1$$

where the formulas for TPR and FPR are already provided in Section 4.1.4. By maximizing J, the optimal threshold for each label can be identified, ensuring a balance between sensitivity and specificity tailored to the characteristics of the dataset and clinical relevance.

4 Experiments and Results

We conduct our experiments on the Kaggle platform using a GPU P100 for accelerated training. The maximum number of epochs is set to 10, and the training process is terminated early if the mAUC score on the validation set does not improve for 5 consecutive epochs. For hyperparameter optimization, we utilize `wandb.ai`'s sweep agent to perform 5 iterations of Bayesian hyperparameter tuning. The hyperparameters tuned include AdamW's learning rate and weight decay, ReduceLROnPlateau's patience, and DenseNet121's drop rate. All experiments were run with a batch size of 32, and training stability was observed with no significant overfitting or vanishing gradient issues. The evaluation metric, mean AUROC (mAUC), is chosen for its effectiveness in multi-label classification tasks. Our results on the test set are benchmarked against the CheXpert baseline model to highlight the effectiveness of our approach.

Model / Label	ResNetTL-U0	ResNetTL-U1	DenseNetTL-U0	DenseNetTL-U1
No Finding	0.86	**0.88**	0.85	0.86
Enlarged Cardiomediastinum	0.63	**0.66**	0.64	0.64
Cardiomegaly	0.82	**0.86**	0.83	0.84
Lung Opacity	0.70	**0.72**	0.69	0.70
Lung Lesion	0.73	**0.79**	0.73	0.74
Edema	0.81	**0.83**	0.81	0.81
Consolidation	0.69	0.68	**0.69**	0.67
Pneumonia	**0.70**	**0.70**	0.69	0.64
Atelectasis	**0.66**	0.65	0.64	0.62
Pneumothorax	0.81	**0.84**	0.80	0.79
Pleural Effusion	0.83	**0.86**	0.83	0.84
Pleural Other	0.78	**0.87**	0.76	0.82
Fracture	0.74	**0.81**	0.72	0.75
Support Devices	0.83	**0.86**	0.82	0.83

Table 1: AUROC value of each label, calculated for each model

The models generally perform well for certain labels such as "No Finding" (ResNetTL-U1: 0.88) and "Lung Opacity" (ResNetTL-U0: 0.81), showing relatively high AUROC values. However, "Enlarged Cardiomediastinum" and "Pneumothorax" exhibit lower AUROC values, around 0.6-0.7, indicating these conditions are harder for the models to classify accurately. ResNet models (especially ResNetTL-U1) tend to outperform DenseNet models for certain labels, such as "No Finding" and "Cardiomegaly". DenseNet models (DenseNetTL-U0 and U1) show competitive performance on "Edema" and "Pleural Effusion," suggesting their strengths in specific conditions. In general, U1 models (uncertainty treated as positive) tend to slightly outperform U0 models (uncertainty treated as negative). This indicates that mapping uncertain labels to positive outcomes may improve classification performance for certain pathologies.

Model / Label	ResNetTL-U0	ResNetTL-U1	DenseNetTL-U0	DenseNetTL-U1	Baseline-U0	Baseline-U1
Atelectasis	0.66	0.65	0.64	0.62	0.81	0.86
Cardiomegaly	0.82	0.86	0.83	0.84	0.84	0.83
Consolidation	0.69	0.68	0.69	0.67	0.93	0.90
Edema	0.81	0.83	0.81	0.81	0.93	0,94
Pleural Effusion	0.83	0.86	0.83	0.84	0.93	0.93

Table 2: Comparison with Baseline-U0 and Baseline-U1 from [3]

There is a notable performance gap between the baseline models and the transfer learning models (ResNet and DenseNet). This indicates that while transfer learning approaches are effective, they may still lag behind well-optimized baseline models trained on high-quality data. The two model Baseline-U0 and Baseline-U1 is configured on the complete CheXpert dataset, whilst ResNetTL-U0, ResNetTL-U1, DenseNetTL-U0, DenseNetTL-U1 is trained on the smaller version due to limited computational power.

5 Discussion

The results demonstrate that transfer learning with DenseNet-121 and ResNet-101, combined with techniques such as uncertainty mapping, weighted random sampling, and data augmentation, can effectively address the challenges of multi-label chest X-ray classification using the CheXpert dataset. Our models achieved decent AUROC scores across most labels, showcasing their capability to handle imbalanced datasets and capture complex visual patterns relevant to diagnostic tasks.

The DenseNet-121 and ResNet-101 models both benefited from pretraining on ImageNet and subsequent fine-tuning for the specific domain of chest X-ray images. However, variations in performance between labels highlight areas for further refinement. For example, the models performed well for conditions like Pleural Effusion and Cardiomegaly, but struggled with more subtle pathologies such as Atelectasis and Consolidation. These discrepancies could be attributed to factors such as class imbalance, noise in the uncertainty labels, and inherent diagnostic difficulty of certain conditions.

The integration of uncertainty mapping strategies (U-Zeros and U-Ones) revealed notable differences in performance. U-Ones, which maps uncertainty to positive labels, generally provided better results for certain pathologies, indicating that uncertain cases may often signify underlying findings rather than true ambiguity. This aligns with clinical insights and underscores the importance of context-aware label processing in medical AI.

While our models performed comparably to the CheXpert baseline on several labels, there remains a performance gap for specific classes, especially in cases with strong class imbalance. Addressing this requires further exploration of advanced data augmentation techniques, alternative uncertainty-handling methods, and architectural innovations tailored for medical imaging. Additionally, larger validation and test sets could provide more robust evaluations and reduce potential overfitting.

6 Conclusion

This study underscores the potential of transfer learning and advanced preprocessing techniques in improving automated chest X-ray disease classification. By leveraging pretrained DenseNet-121 and ResNet-101 models with tailored modifications, we achieved competitive performance on the CheXpert dataset across multiple clinically significant pathologies. The findings validate the effectiveness of uncertainty mapping, weighted random sampling, and optimized thresholds in enhancing model robustness.

However, challenges remain, particularly in addressing class imbalance and improving sensitivity for subtle pathologies. Future work should explore techniques such as multi-task learning, semi-supervised approaches, and leveraging

additional metadata for context-aware predictions. Expanding the dataset with diverse and high-quality annotations could further enhance model performance and generalizability.

In conclusion, this project contributes to the growing field of AI-assisted radiology by demonstrating practical strategies to improve multi-label classification performance. The integration of these methods into clinical workflows has the potential to enhance diagnostic accuracy and efficiency, ultimately benefiting patient care.

References

[1] Louke Delrue et al. "Difficulties in the Interpretation of Chest Radiography". In: Sept. 2011, pp. 27–49. ISBN: 978-3-540-79941-2. DOI: 10.1007/978-3-540-79942-9_2.

[2] Yann LeCun, Y. Bengio, and Geoffrey Hinton. "Deep Learning". In: *Nature* 521 (May 2015), pp. 436–44. DOI: 10.1038/nature14539.

[3] Jeremy Irvin et al. *CheXpert: A Large Chest Radiograph Dataset with Uncertainty Labels and Expert Comparison*. 2019. arXiv: 1901.07031 [cs.CV]. URL: https://arxiv.org/abs/1901.07031.

[4] Christian Garbin et al. *Structured dataset documentation: a datasheet for CheXpert*. 2021. arXiv: 2105.03020 [eess.IV]. URL: https://arxiv.org/abs/2105.03020.

[5] International Society of Radiographers and Radiological Technologists. *eLearning Resources*. Accessed: 2023-12-15. n.d. URL: https://www.elearning.isrrt.org/mod/book/view.php?id=321&fbclid=IwY2xjawHLlnJleHRuA2FlbQIxMAABHeGfg3P2zF9HoR2o5En1KtAwxk1NIKRDvyJ0XzGl2BAECzyrqgMEtgwSdQ_aem_gR95zFIyJfFEr62ZCE7KmA.

[6] Konstantinos Sechidis, Grigorios Tsoumakas, and Ioannis Vlahavas. "On the Stratification of Multi-label Data". In: *Machine Learning and Knowledge Discovery in Databases*. Ed. by Dimitrios Gunopulos et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 145–158. ISBN: 978-3-642-23808-6.

[7] Piotr Szymański and Tomasz Kajdanowicz. *A Network Perspective on Stratification of Multi-Label Data*. 2017. arXiv: 1704.08756 [stat.ML]. URL: https://arxiv.org/abs/1704.08756.

[8] E Goceri. *Medical image data augmentation: techniques, comparisons and interpretations*. 2023. Artif Intell Rev 56: 12561ó ÀÃ12605.

[9] Aston Zhang et al. *Dive into Deep Learning*. https://D2L.ai. Cambridge University Press, 2023.

[10] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[11] Gao Huang et al. *Densely Connected Convolutional Networks*. 2016. arXiv: 1608.06993 [cs.CV].

[12] Ilya Loshchilov and Frank Hutter. *Decoupled Weight Decay Regularization*. 2017. arXiv: 1711.05101 [cs.LG].

[13] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. arXiv: 1412.6980 [cs.LG].

[14] PyTorch. *PyTorch Documentation*. Accessed: 2023-12-15. 2023. URL: https://pytorch.org/docs/stable/index.html.

[15] James Bergstra et al. “Algorithms for Hyper-Parameter Optimization”. In: *Advances in Neural Information Processing Systems*. Ed. by J. Shawe-Taylor et al. Vol. 24. Curran Associates, Inc., 2011. URL: https://proceedings.neurips.cc/paper_files/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf.

[16] WJ Youden. “Index for rating diagnostic tests”. In: *Cancer* 3.1 (1950), pp. 32–35. DOI: 10.1002/1097-0142(1950)3:1<32::aid-cncr2820030106>3.0.co;2-3.