Chest X-ray

Group 24

JON ANDREAS BULL LARSSEN
JON INGVAR JONASSEN SKÅNØY
ISAK KILLINGRØD

Obligatorisk gruppeerklæring

Den enkelte student er selv ansvarlig for å sette seg inn i hva som er lovlige hjelpemidler, retningslinjer for bruk av disse og regler om kildebruk. Erklæringen skal bevisstgjøre studentene på deres ansvar og hvilke konsekvenser fusk kan medføre. Manglende erklæring fritar ikke studentene fra sitt ansvar.

1.	Vi erklærer herved at vår besvarelse er vårt eget arbeid, og at vi ikke har brukt andre kilder eller har mottatt annen hjelp enn det som er nevnt i besvarelsen.	Ja
2.	**Vi erklærer videre at denne besvarelsen:** • Ikke har vært brukt til annen eksamen ved annen avdeling/universitet/høgskole innenlands eller utenlands. • Ikke refererer til andres arbeid uten at det er oppgitt. • Ikke refererer til eget tidligere arbeid uten at det er oppgitt. • Har alle referansene oppgitt i litteraturlisten. • Ikke er en kopi, duplikat eller avskrift av andres arbeid eller besvarelse.	Ja
3.	Vi er kjent med at brudd på ovennevnte er å betrakte som fusk og kan medføre annullering av eksamen og utestengelse fra universiteter og høgskoler i Norge, jf. Universitets- og høgskoleloven §§4-7 og 4-8 og Forskrift om eksamen §§ 31.	Ja
4.	Vi er kjent med at alle innleverte oppgaver kan bli plagiatkontrollert.	Ja
5.	Vi er kjent med at Universitetet i Agder vil behandle alle saker hvor det forligger mistanke om fusk etter høgskolens retningslinjer for behandling av saker om fusk.	Ja
6.	Vi har satt oss inn i regler og retningslinjer i bruk av kilder og referanser på biblioteket sine nettsider.	Ja
7.	Vi har i flertall blitt enige om at innsatsen innad i gruppen er merkbart forskjellig og ønsker dermed å vurderes individuelt. Ordinært vurderes alle deltakere i prosjektet samlet.	Nei

Publiseringsavtale

Fullmakt til elektronisk publisering av oppgaven Forfatter(ne) har opphavsrett til oppgaven. Det betyr blant annet enerett til å gjøre verket tilgjengelig for allmennheten (Åndsverkloven. §2).
Oppgaver som er unntatt offentlighet eller taushetsbelagt/konfidensiell vil ikke bli publisert.

Vi gir herved Universitetet i Agder en vederlagsfri rett til å gjøre oppgaven tilgjengelig for elektronisk publisering:	Ja
Er oppgaven båndlagt (konfidensiell)?	Nei
Er oppgaven unntatt offentlighet?	Nei

Abstract

The growing demand for efficient and accurate medical imaging analysis highlights the importance of deep learning in healthcare. This paper evaluates the performance of popular convolutional neural networks (CNNs), including DenseNet, ResNet, MobileNet, and VGG, on the NIH ChestX-ray14 dataset, which contains 112,120 images across 14 thoracic disease classes. Using pretrained models, we explore the impact of different data preprocessing techniques, data augmentation strategies, and optimizers. The study further investigates architectural variations within DenseNet and evaluates 1-channel grayscale input versus duplicated channel input. The findings provide insights into optimizing CNNs for medical imaging tasks, emphasizing the importance of tailored preprocessing and training strategies for improved diagnostic performance.

Keywords: Deep Learning, Convolutional Neural Networks, Medical Imaging, Chest X-ray Classification, Data Augmentation, Transfer Learning, DenseNet, NIH ChestX-ray14

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The primary goal of this paper is to compare some of the most common models and training techniques on the NIH ChestXray14 dataset.

Motivation In the UK at Queen Alexandra hospital there where 23 000 chest X-rays who where not formally reviewed by a radiologist or clinician. Medical imaging is important for the operation of a hospitals. Radiologists require extensive medical training. The needs for radiologists are currently not being met. At the hospital 3 patients have suffered significantly because of the lack of assessment. This is at only at a hospital in 12 months[1]. We can only image the global consequences of the lack of radiologist, when its not properly addressed in western countries.

Objectives This report seeks to compare models, earlier literature has applied different models with different approaches. This report aims to compare different models on the same approach to gain eliminate potential other factors. This report also aims to evaluate different augmentation methods. This report also aims to evaluate different optimizers with a focus on Adam based optimizers. This will provide a more compressive evaluation.

Limitations Our experiments and comparisons do not compare all combinations. for some of our comparisons we have not ran hyperparameters search on all approaches. For example for our evaluation of data augmentation methods the hyperparameters used for method 2 is optimized for method 1, therefore it is not a fair comparison between the two. For comparisons between models it is possible the models we have compared with DenseNet would outperform the current best result with a different data augmentation, optimizer.

Chapter 2

Background

2.1 NIH Chest X-ray Dataset

The ChestX-Ray14[2] is a dataset for the study of thoracic diseases. It consists of 112.120 frontal chest x-ray images from 30,805 patients, collected from 1992 to 2015. It has 14 disease classes of varying sizes and is one of the largest public datasets of its kind. It expands on ChestX-ray8 by adding six additional thorax diseases: Edema, Emphysema, Fibrosis, Pleural Thickening and Hernia. This makes it an ideal dataset for multi label classification. It has also have some location data for some of its classes but that is not utilized in this report. A limitations of the dataset is that the labels was extracted by text mining of NIH's radiology reports. This NLP techniques may have introduced some label noise, but accuracy is estimated to be $> 90\%$.

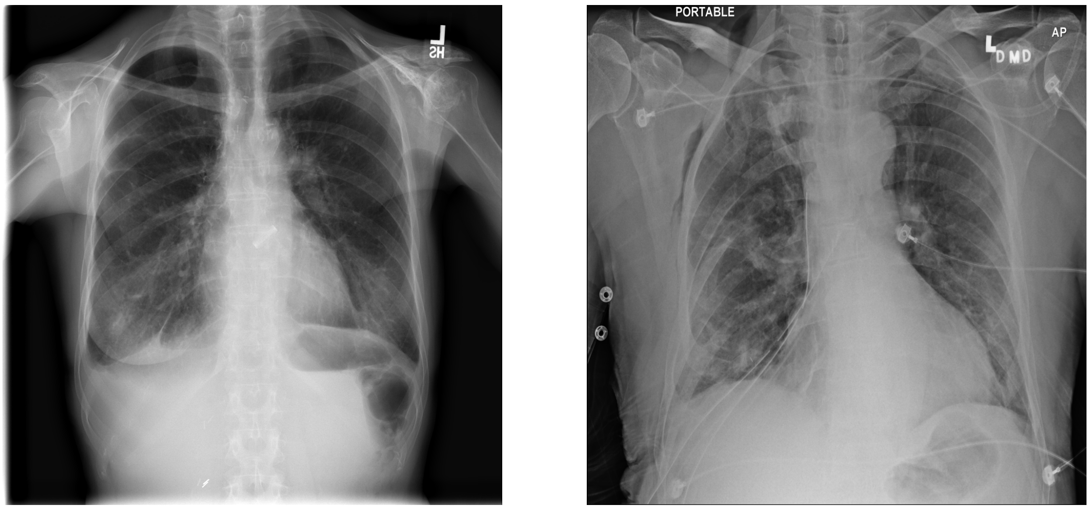


Figure 2.1: Samples from dataset

2.2 Artificial Neural Networks

History Artificial Neural Networks (ANNs) have their roots in the 1940s when Warren McCulloch and Walter Pitts introduced a mathematical model of a neuron, demonstrating its ability to compute logical operations[3]. This foundational work inspired the development of perceptrons by Frank Rosenblatt in the 1950s, which marked the beginning of machine learning systems capable of simple pattern recognition tasks[3]. However, the limitations of early models, as highlighted by Minsky and Papert in 1969, stalled progress until the 1980s when backpropagation algorithms and multilayer perceptrons reignited interest in the field[3].

The modern rise of ANNs owes much to advances in computational power, availability of large datasets, and innovations such as deep learning frameworks[4].

Neurons ANNs are computational models that mimic the behavoiur of biological neurons. In simple terms, biological neurons are responsible for sending and receiving neurotransmitters—chemicals that carry information between brain cells[6]. In the same simplified fashion artificial neurons receive inputs, process the information and transmit the information to other neurons.

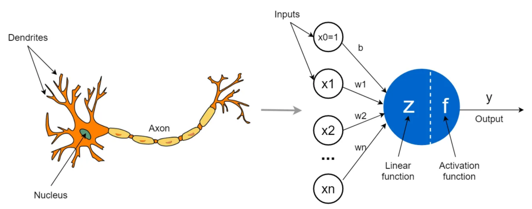


Figure 2.2: Illustration of biological (left) and artificial (right) neurons[5]

Layers ANN's are organized in layers of neurons or nodes. Each node are connected to some or all of the nodes in next layer. The first layer is often called the the input layer. It consist of values that are given to the networks. The values or features can be pixel values in images or other representations of data.

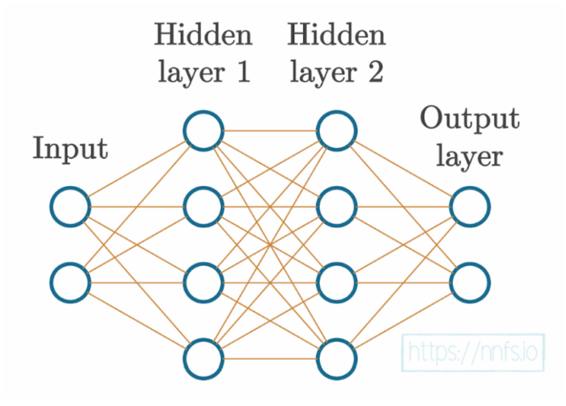


Figure 2.3: Example of basic neural network[7]

The last layer is often called the output layer and it's size is determined by the type of task the network will perform. I.e for multi label classification with 14 labels the output layer will often have 14 nodes, one for each label. The layers between the first and the output layers are often called hidden layers. These layers process the data and may vary in size depending on the network architecture.

Learning ANN's utilize weights, biases and functions in order to learn from inputs it is given.

Weights determine the strength of the connections between nodes in adjacent layers. These weights are adjusted during the training process to minimize the network's error.

A *bias* term is added to the weighted sum of inputs to shift the activation function, allowing the model to fit the data more flexibly. Bias ensures that even when all input features are zero, the neuron can still produce an output.

The *sum operation* calculates the inputs weighted by their connection strengths (weights) and adds the bias term.

$$s_i = w_0 + \sum_{i=1}^{n} w_{ij} x_j$$

This step forms the basis for determining the neuron's activation.

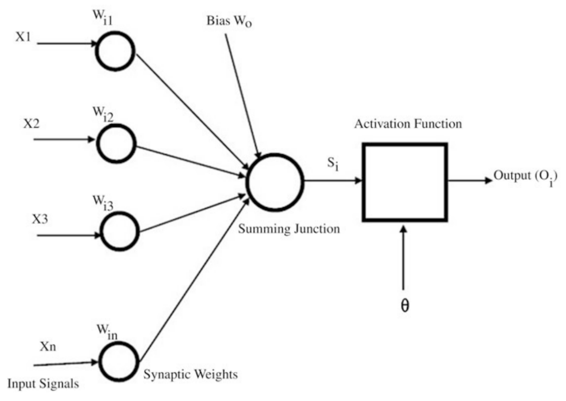


Figure 2.4: Model of a node and connections in ANN[3]

Activation functions introduce non-linearity to the network, enabling it to model complex relationships in data. Common activation functions include ReLU (Rectified Linear Unit), sigmoid, and tanh, each suited to specific use cases. For instance, ReLU is widely used in hidden layers due to its simplicity and computational efficiency, while softmax is popular for output layers in multi-class classification tasks[8].

Backpropagation is the process of optimizing the weights and biases in the network by minimizing the error between predicted and true outputs. It employs gradient descent to calculate the partial derivatives of the loss function concerning each weight. By propagating the error backwards through the network, backpropagation ensures that the parameters are updated iteratively to improve model accuracy. This algorithm has been essential in training deep neural networks.

2.3 Deep Learning

Deep learning (DL) is an extension of ANN's by its use of multiple hidden layers, enabling the modeling of complex, hierarchical data representations[4]. It leverages large datasets and computational power to perform tasks such as image recognition, natural language processing, and autonomous driving with high accuracy. DL models utilize advanced architectures like convolutional neural networks (CNN's) and recurrent neural networks (RNN's), which have proven effective in domains such as computer vision and sequence prediction[8].

2.4 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a specialized type of ANNs designed to process data with a grid-like topology, such as images. Their architecture is well-suited for tasks like image classification, object detection, and segmentation due to their ability to capture nested spatial structures.
CNNs are structured with three primary types of layers: convolutional layers, pooling layers, and fully connected layers, which work together to extract and learn meaningful patterns from the input data [4, 8].

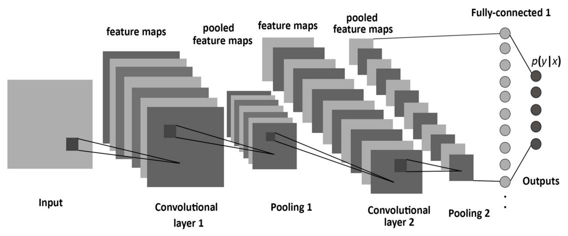


Figure 2.5: Structure of a CNN, with convolutional, pooling, and fully-connected layers.[9]

Convolution layers Convolution layers are responsible for feature extraction. These layers apply learnable filters (kernels) across the input data, performing operations such as edge detection or texture extraction. The convolution operation produces feature maps that highlight the presence of specific patterns in the data.

Pooling layers According to Goodfellow et. al[8] "A pooling function replaces the output of the net at a certain location with asummary statistic of the nearby outputs". Common pooling methods include max pooling, which selects the highest value in a given region, and average pooling, which computes the average value. Pooling also introduces translational invariance, enabling the network to recognize features regardless of their spatial position in the input [8].

Fully connected layers Fully connected layers are typically used towards the end of the CNN to map the extracted features to the final output, such as class probabilities in a classification task. Each node in a fully connected layer connects to every node in the previous layer, allowing the network to integrate the high-level features learned by the convolutional and pooling layers. Activation functions such as softmax or sigmoid are often used in this layer to normalize outputs for classification tasks [8].

2.5 Transfer Learning

Transfer learning is an ML-approach where a model that is trained on a dataset is used alone or as a "base-model" for other tasks. Such models are often called pretrained models because they are trained on large datasets like ImageNet[10]. The pretrained models learn many features from the large datasets, such as edges, textures, and shapes, which are often applicable across other domains as well.
There are several ways to make use of pretrained models:

- **Fine-tuning:** The existing weights in the model might be useful for the new task and might therefor be fine-tuned to better fit the new task or domain.
- **Build upon:** Pretrained models might be used as part of ensemble of several models where the combined outputs of several models perform the task at hand.

The advantages of the use of transfer learning is medical tasks are well evaluated by Shin et al[11]

2.6 Pretrained Models

DenseNet

DenseNet is a deep convolutional network. It utilities more connections between layers, to allow a deeper network. Each layer has connections to every subsequent layer, as apposed to the regular method of only having connections to the next layer. This is to mitigate the vanish-gradient problem. In this manner, DenseNet can have deeper layers, which achieves higher results at a lower computational cost[12].

ResNet

ResNet is a deep convolutional neural network. which consists of residual blocks which utilizes skip connections. This allows for the design of deeper networks by addressing the vanishing gradient problem[13]. ResNet has a number of variants including ResNet-18, ResNet-34, ResNet-50, ResNet-101, and ResNet-152, where the number is the number of layers. ResNet is typically trained on the ImageNet dataset.

Visual Geometry Group (VGG)

VGG is a very deep convolutional neural network designed for large-scale image recognition. The key innovation of VGG is its use of very small (3x3) convolution filters in a deep architecture. The primary focus of the VGG research was to investigate the effect of network depth on accuracy in large-scale image recognition tasks. By increasing the depth of the network to 16-19 layers, VGG achieved a significant improvement over prior-art configurations[14].

MobileNet

MobileNet is a class of efficient convolutional neural networks designed specifically for mobile and embedded vision applications. The primary focus of MobileNet is to create lightweight deep neural networks that can operate effectively under the constraints of mobile devices.
The core architecture of MobileNet is based on depth-wise separable convolutions, which are a key component in building light-weight deep neural networks[15]. This approach allows MobileNet to achieve a balance between model performance and computational efficiency.
MobileNetV2 is a light weight model, is an neural network architecture. It uses an inverted residual structure, there are shortcut connections between bottle-next layers. This allows for a light weight model[16].

InceptionNet

InceptionNet is a deep convolutional neural network architecture designed for computer vision tasks, particularly image recognition. It was introduced as GoogleNet (also known as InceptionV1) and has since evolved through several iterations, including InceptionV3 and InceptionV4. The main characteristic of the Inception architecture is the improved utilization of computing resources inside the network[17]. With focus on efficient computation, multi-scale feature extraction and it's evolution through various versions makes it a relevant architecture in the area of image processing.

2.7 Related Work

The ChestXray14 dataset has been important in advancing automated thoracic disease diagnosis through various deep learning methodologies. Researchers have explored multiple approaches to improve classification performance and address fundamental challenges such as data imbalance and multi-label classification.
There have been many attempts at using pretrained models for the ChestX-ray8 dataset, with most of these employing DenseNet or ResNet models [18]. More generally, in the field of X-rays, other models like VGG16, MobileNetV2, EfficientNet, and variants of DenseNet are frequently used [19]. Some works have also used models trained from scratch, but pretrained models consistently outperform these approaches.

One notable contribution is CheXNet[20]. A 121-layer CNN trained on the ChestXray14 dataset which achieved radiologist-level performance in detecting pneumonia, setting a benchmark in medical image analysis

Ge et al[21] addressed the challenges of multi-label and fine-grained classification within chest X-rays. They introduced a Multi-label Softmax Loss (MSML) function to manage the presence of multiple pathologies and data imbalance, demonstrating improved Area Under the Receiver Operating Characteristic (AUC-ROC) scores on the ChestXray14 dataset.

In the area of ensemble learning, Ashraf et al[22] proposed SynthEnsemble, which combines CNNs, transformers, hybrid and classical models. This ensemble approach achieved an average AUC of 85.4% across 14 disease labels, surpassing previous state-of-the-art methods.

Gozes and Greenspan[23] explored transfer learning by training a DenseNet121 model on the ChestXray14 dataset to extract features relevant to tuberculosis detection in smaller datasets. Their study highlighted the benefits of domain-specific pretraining over generic ImageNet pretraining, leading to enhanced classification performance.

One of the challenges of using pretrained models for Chest X-rays is that they are trained for 3-channel colored images, while Chest X-rays are 1-channel grayscale images. Previous works, such as [1], have tested whether it is better to change the model to receive one channel with mixed results. Alternatively, [24] duplicates the channel to three identical ones, which results in slight differences after normalization.

There have been many attempts at using pretrained models for the ChestX-ray8 dataset, with most of these employing DenseNet or ResNet models [18]. More generally, in the field of X-rays, other models like VGG16, MobileNetV2, EfficientNet, and variants of DenseNet are frequently used [19]. Some works have also used models trained from scratch, but pretrained models consistently outperform these approaches.

More related work can be found at `www.paperswithcode.com` which currently contain 225 papers related to ChestX-ray14 dataset [25].

Chapter 3

Methods

3.1 Hardware setup

The experiments are performed on a combination of local machines and on a shared server. The experiments are primarily performed on either Nvidia RTX 3080 10GB or Nvidia Tesla V100 SXM3 32 GB. More hardware specifications are listed in appendix B.

3.2 Data Analysis

The dataset has a total of 112,120 images taken 30,805 patients, which means that there are multiple images per patient. There is a varying number of images per patient, and as there's a similarity between images of the same patient, there should not be images of the same patient in multiple test or train splits. The dataset has an official train test spilt. Where some of its patients are in the train_val split and some in the test split.

Label	**Train**	**Validation**	**Test**
Number of Samples	77,988	8,536	25,596
Pneumonia	782	94	555
Pneumothorax	2394	243	2665
Pleural Thickening	1997	245	1143
Hernia	122	19	86
Fibrosis	1111	140	435
Emphysema	1317	166	1093
Cardiomegaly	1549	158	1060
Mass	3646	388	1748
Edema	1253	268	925
Infiltration	12480	1302	6112
Consolidation	2584	268	1815
Effusion	7852	807	4658
Nodule	4247	461	1623
Atelectasis	7405	875	3279

Table 3.1: Label distribution across train, validation, and test sets, including the number of samples in each set.

Each patients can have multiple thoracic diseases. Figure 3.1 illustrates the number of patients with multiple diseases, for the ChestXray14 dataset with 14 labels. This demonstrates the complexity of the data.

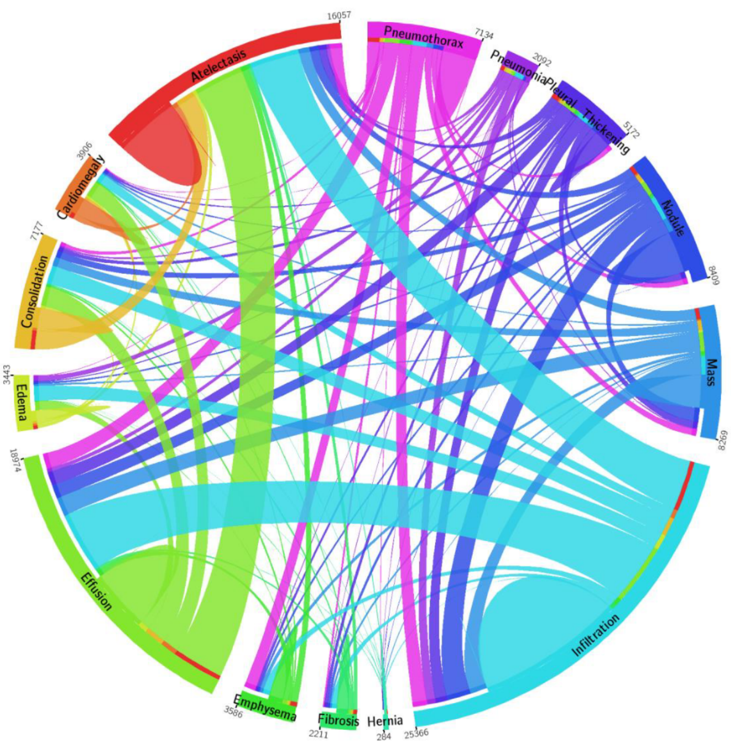


Figure 3.1: A circular diagram which shows the proportions of multi-labels for the 14 labels[2]

3.3 Data preprocessing

As mentioned in section 2.7, the first challenge is that the images are 1 channel, and their pretrained models expect and are trained for 3. The proposed approach has the [18] solution, which is to duplicate the channel in to 3 identical ones, before standardization.

For data augmentation the proposed approach is inspired by [1] with a random rotation of maximum $\pm$ 7° then a random resized crop which creates a patch of 224x224 pixels and represent between 8% and 100% of the image, a random horizontal flip and an added color jitter. Then the image is normalized to between 0 and 1, and then standardized with the ImageNet values.

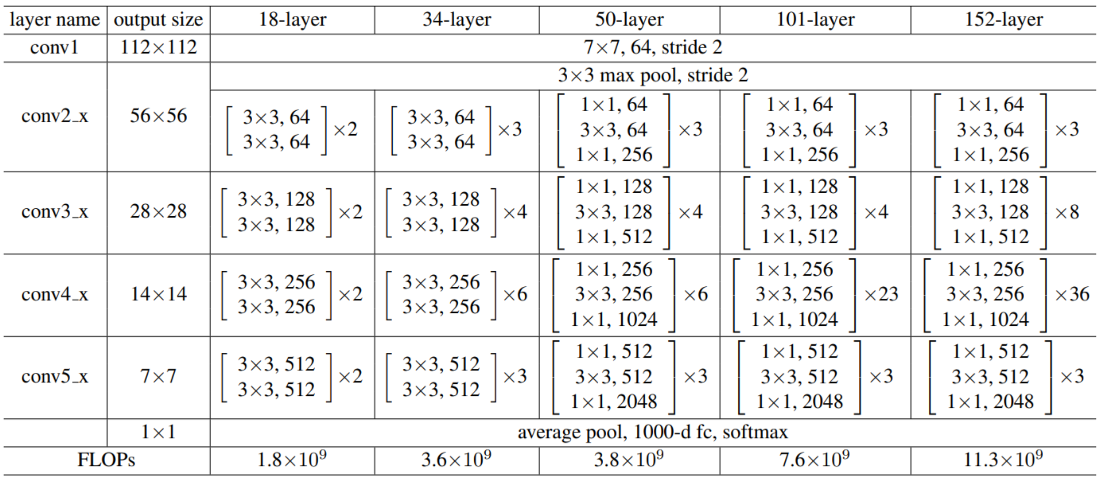
For test and validation data the image is resized to 256x256 and then a center crop. Then the image is normalized to between 0 and 1, and then standardized with the Image Net values.

3.4 Models

PyTorch

This project has made use of preexisting implementations of each model architecture through the python library PyTorch. When initializing each model's PyTorch implementation, the weights of the model can be initialized either with an arbitrary value, or with the weights of a pre-trained model. This project makes use of sets of pre-trained weights for each model which were created by training on the imagenet-1k dataset introduced by Russakovsky et al. in 2015[26]. Having models that already have some prior ability to comprehend images provides a beneficial starting point for training models on the ChestXRay14 dataset due to it's limited size.

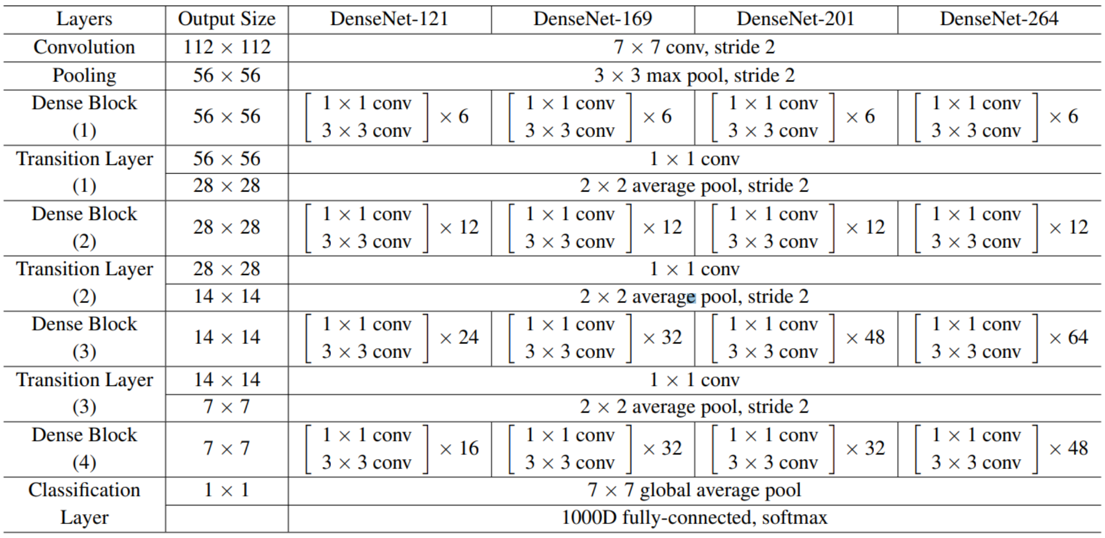
ResNet ResNet is an abbreviation of Residual Neural Network, which is derived from the model's use of residual connections to mitigate the vanishing gradient problem that caused too deep models to deteriorate. Through passing the input data directly into the deeper layers through residual connections, the model can benefit from much a deeper structure than without residual connections, and can therefore learn more complex patterns[27]. In contrast with contemporary models such as VGG, which had 16 or 19 layers, ResNet's residual connections allowed it to train effectively with up to 152 layers.



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 3.2: Architecture of ResNet for ImageNet[27]

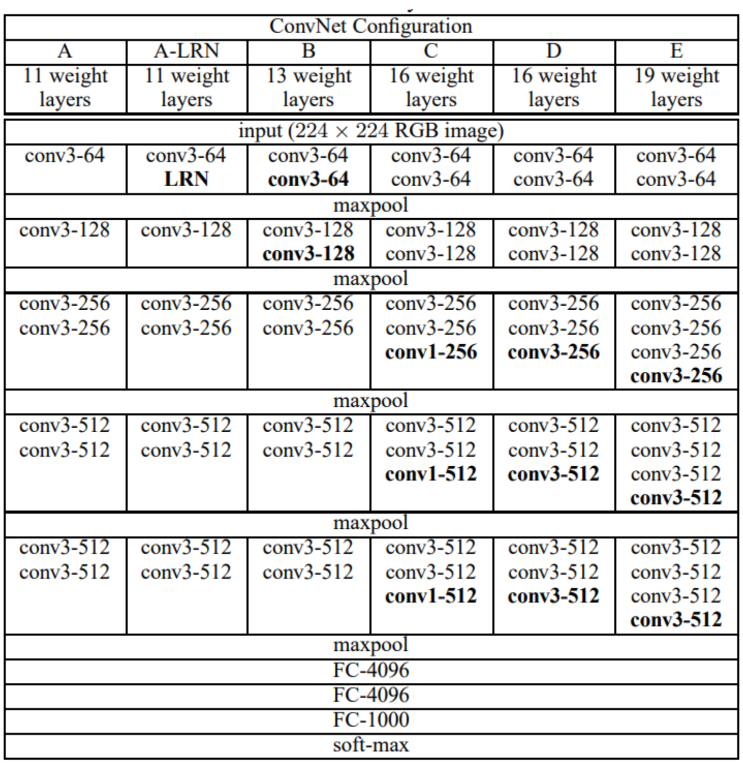
DenseNet DenseNet is another convolutional network that takes advantage of residual connections to enhance performance and benefit from a deeper structure[12]. The residual connections is implemented by connecting each layer directly to every other subsequent layer. It use a composite function of batch normalization, ReLU activation, and convolution. The concatenation operation ensures that each layer has access to feature maps from earlier layers. Additionally, DenseNet employs "dense blocks," where densely connected layers are grouped, mixed with transition layers with operations like convolution and pooling to downsample the feature maps and control the network's computational complexity.



Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	$\begin{bmatrix}1\times1\text{ conv}\\3\times3\text{ conv}\end{bmatrix}\times6$	$\begin{bmatrix}1\times1\text{ conv}\\3\times3\text{ conv}\end{bmatrix}\times6$	$\begin{bmatrix}1\times1\text{ conv}\\3\times3\text{ conv}\end{bmatrix}\times6$	$\begin{bmatrix}1\times1\text{ conv}\\3\times3\text{ conv}\end{bmatrix}\times6$
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	$\begin{bmatrix}1\times1\text{ conv}\\3\times3\text{ conv}\end{bmatrix}\times12$	$\begin{bmatrix}1\times1\text{ conv}\\3\times3\text{ conv}\end{bmatrix}\times12$	$\begin{bmatrix}1\times1\text{ conv}\\3\times3\text{ conv}\end{bmatrix}\times12$	$\begin{bmatrix}1\times1\text{ conv}\\3\times3\text{ conv}\end{bmatrix}\times12$
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	$\begin{bmatrix}1\times1\text{ conv}\\3\times3\text{ conv}\end{bmatrix}\times24$	$\begin{bmatrix}1\times1\text{ conv}\\3\times3\text{ conv}\end{bmatrix}\times32$	$\begin{bmatrix}1\times1\text{ conv}\\3\times3\text{ conv}\end{bmatrix}\times48$	$\begin{bmatrix}1\times1\text{ conv}\\3\times3\text{ conv}\end{bmatrix}\times64$
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	$\begin{bmatrix}1\times1\text{ conv}\\3\times3\text{ conv}\end{bmatrix}\times16$	$\begin{bmatrix}1\times1\text{ conv}\\3\times3\text{ conv}\end{bmatrix}\times32$	$\begin{bmatrix}1\times1\text{ conv}\\3\times3\text{ conv}\end{bmatrix}\times32$	$\begin{bmatrix}1\times1\text{ conv}\\3\times3\text{ conv}\end{bmatrix}\times48$
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

Figure 3.3: Architecture of DenseNet for ImageNet[12]

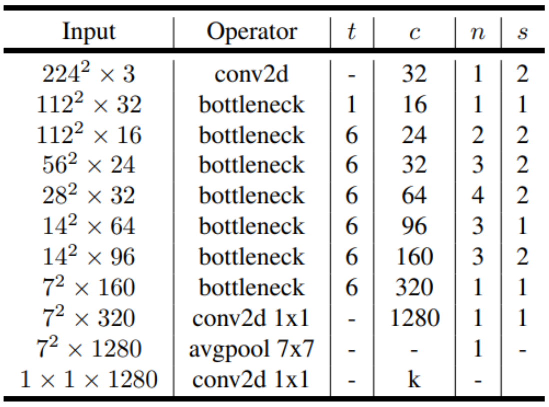
VGG16 VGG16, first introduced by Simonyan et al. in 2015, is a 16-layered convolutional neural network which is characterized by five sets of convolutional layers of increasing size (64, 128, 256, 512, 512) with a max pooling layer after each set of convolutional layer, followed by a final set of three fully connected layers[14].



ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 3.4: Architecture of VGG[14]

MobileNetV2 The MobileNet class of model was first introduced by Howard et al. in 2017 with the purpose of creating light-weight models that can be used on mobile and embedded devices.[28] To accomplish this goal, the MobileNet models prioritize having as efficient as possible of a trade-off between performance and efficiency, in contrast to other models which might prioritize performance in a vacuum. This project makes use of MobileNetV2 in particular, which is a successor to the original set of MobileNet models, which notably performs its computations significantly faster than MobileNetV1 while retaining similar accuracy, further improving the performance to efficiency ratio[29].



Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Figure 3.5: Architecture of MobileNetV2[16]

3.5 Evaluation Metrics

The Area under the ROC curve or AUC, is a metric to evaluate the performance of a binary classier with unbalanced data. For imbalanced data accuracy is not a good representative metric for the performance of the classifier. it considers the relation between the true positive rate and the false positive rate[30].

3.6 Implementation

Hyperparameter Tuning with Optuna When performing comprehensive testing of the performance of several models, a major logistical challenge is to optimize the hyperparameters of each model to ensure they are as performant as possible. If one were to manage the search for the ideal hyperparameters manually, it would become time consuming to return to the results of tests and start new ones on a regular basis. An algorithmic approach also has the possible benefit of approaching the most performant solution in fewer iterations than a manual search.
Optuna is an open source framework for hyperparameter optimization introduced in 2019 by Akiba et al[31]. In practice, Optuna is a simple library to apply to a machine learning training loop by encasing the training and evaluation process in a single function that takes in a study object as its sole parameter. Each trial where a model is trained and evaluated, this study object decides which hyperparameters to use when given a choice of options, and retains information about which hyperparameters have been tried in the past and what performance metric they resulted in.
When searching for the ideal hyperparameters, it could be easy to overfit the hyperparameters towards the contents of the test dataset. To account for this, the Optuna studies were used to find hyperparameters that perform well on the validation set, which were then tested on the test set once training of the best models were completed.

Mean results for every model one trained there is a certain unpredictability in the results. Was this a lucky or unlucky run, therefore determining which of two runs mad the best training approach or model is difficult to say with the degree of certainty this report would prefer. Therefore for each tested approach or model we have run 5 times and display the mean result and the standard deviation.
Since there are 14 classes this report also reports the mean of these classes. This mean is unweighted, meaning *Hernia* with 86 examples in the test data have as much impact as *Infiltration* with 6112 samples.

Scheduler The implementation utilizes a scheduler. Optuna was given a number of possible schedulers with standard parameters, in every study it chose the CosineAnnealingLR scheduler. It is possible that this was the scheduler with the best parameters selected. Since it was the only one selected in later studies we locked the scheduler as CosineAnnealingLR.

Patience factor Training Large CNN models on large datasets take time, therefore this project has utilized patience factor. For each epoch validation loss is tested, if its the new best loss the model is saved and the patience counter is reset. 3 consecutive epochs without a reset stops the training and the model from the best epoch is loaded before testing.

Grad clip As a method of stabilizing training our approach utilizes gradient clipping. This ensures that no single batch changes the model to much, therefore model training is a more stable processes towards the a minima[32].

Loss Function The project utilizes unweighted binary cross entropy loss, this is a loss function which is much used for this dataset[20]. The PyTorch implementation utilized is BCE with logits.

Chapter 4

Results And Discussion

This chapter discusses the results of the experiments performed.

Overview of experiments

4.1 **Comparison of Common Pretrained CNN Architectures**
Evaluates the performance of widely used convolutional neural network architectures, including DenseNet, ResNet, and MobileNetV2, on the ChestX-ray dataset.

4.2 **Comparison of Data Augmentation Methods**
Analyzes the impact of different augmentation strategies, such as random rotations, resized crops, and color jitter, on model performance.

4.3 **Optimization Techniques for DenseNet121**
Compares the effectiveness of optimization algorithms (Adam, NAdam, and AdamW) in training the DenseNet121 model.

4.4 **Evaluation of DenseNet Variants**
Compares the performance of DenseNet models of varying depths, specifically DenseNet121, DenseNet161, and DenseNet201.

4.5 **Effect of Input Channel Configuration**
Assesses the performance differences between using a single-channel input versus duplicating the channel to create a three-channel input.

4.6 **Impact of Image Size**
Investigates how varying image resolutions (e.g., 224x224 vs. 448x448) affect the model's ability to learn and generalize.

4.1 Comparison of common models

This experiment seeks to compare a number of pretrained models. For each model there has been done an Optuna study to find the ideal hyperparameters. A factor here is that each search can be lucky or unlucky, and might not find the absolute best hyperparameters, although the search likely outperforms human efforts in searching for the hyperparameters within the time span.

Condition	**DenseNet121**	**ResNet50**	**MobileNetV2**	**VGG16**
Atelectasis	0.7811 ± 0.0024	$\mathbf{0.7837 \pm 0.0016}$	0.7735 ± 0.0015	0.7532 ± 0.0034
Cardiomegaly	$\mathbf{0.8751 \pm 0.0041}$	0.8729 ± 0.0036	0.8704 ± 0.0013	0.8468 ± 0.0086
Consolidation	0.7543 ± 0.0048	$\mathbf{0.7577 \pm 0.0023}$	0.7514 ± 0.0022	0.7257 ± 0.0020
Edema	$\mathbf{0.8511 \pm 0.0033}$	0.8510 ± 0.0030	0.8481 ± 0.0027	0.8315 ± 0.0036
Effusion	0.8309 ± 0.0025	$\mathbf{0.8317 \pm 0.0012}$	0.8262 ± 0.0012	0.8168 ± 0.0025
Emphysema	0.9106 ± 0.0050	$\mathbf{0.9184 \pm 0.0010}$	0.9052 ± 0.0019	0.8732 ± 0.0093
Fibrosis	0.8333 ± 0.0047	$\mathbf{0.8346 \pm 0.0022}$	0.8293 ± 0.0022	0.7884 ± 0.0111
Hernia	0.9117 ± 0.0053	$\mathbf{0.9234 \pm 0.0081}$	0.8895 ± 0.0037	0.8584 ± 0.0224
Infiltration	0.7046 ± 0.0020	$\mathbf{0.7058 \pm 0.0013}$	0.7026 ± 0.0011	0.6987 ± 0.0025
Mass	0.8291 ± 0.0056	$\mathbf{0.8350 \pm 0.0011}$	0.8127 ± 0.0025	0.7763 ± 0.0057
Nodule	0.7721 ± 0.0078	$\mathbf{0.7772 \pm 0.0017}$	0.7469 ± 0.0047	0.7270 ± 0.0091
Pleural Thickening	0.7814 ± 0.0057	$\mathbf{0.7862 \pm 0.0027}$	0.7727 ± 0.0024	0.7443 ± 0.0065
Pneumonia	$\mathbf{0.7359 \pm 0.0041}$	0.7351 ± 0.0042	0.7193 ± 0.0034	0.7068 ± 0.0031
Pneumothorax	0.8670 ± 0.0029	$\mathbf{0.8710 \pm 0.0012}$	0.8598 ± 0.0030	0.8534 ± 0.0027
Mean	0.8170 ± 0.0015	$\mathbf{0.8203 \pm 0.0018}$	0.8077 ± 0.0010	0.7858 ± 0.0053

Table 4.1: Mean ± Std values for different models across conditions. The best result for each condition is bolded.

As shown in table 4.1 ResNet50 performs best on the average AUC score. It performs best on most classes, where the *Pneumothorax* is the most significant improvement.

In preliminary experiments DenseNet121 performed best, and is therefore used in all other experiments.

4.2 Data augmentation experiment

For any image training, performing data augmentation is important to achieve good results. Its main purpose is to reduce overfitting. With good data augmentation one effectively has a larger dataset than one really has. This is especially important when it comes to medical imaging as good images of certain pathologies are rare, and as such expensive to get a sufficient amount of.

Method 1 is inspired by [1]. This a strong data augmentation which has a strong regularization effect. It is possible there are instances where finding the diagnosis is impossible.
Method 2 has a much weaker augmenting effect which is what the report seeks to compare.
No augmentation is to demonstrate the impact of performing augmentation.

Steps	**Method 1**	**Method 2**	**No Augmentation**
Random Rotation (°)	7	15	-
Random Resized Crop (Pixels)	224x224	None	-
Horizontal Flip	Yes	No	-
Color Jitter	Yes	No	-
Resize	Crop -> 224x224	256x256	-
Center Crop	No	224x224	-
Normalization	Yes	Yes	Yes

Table 4.2: Data Augmentation Methods. See Appendix A for detailed parameter settings.

Condition	**Method 1**	**Method 2**	**No Augmentation**
Atelectasis	$\mathbf{0.7811 \pm 0.0024}$	0.7740 ± 0.0032	0.7696 ± 0.0024
Cardiomegaly	0.8751 ± 0.0041	0.8757 ± 0.0079	$\mathbf{0.8813 \pm 0.0054}$
Consolidation	$\mathbf{0.7543 \pm 0.0048}$	0.7469 ± 0.0018	0.7462 ± 0.0025
Edema	$\mathbf{0.8511 \pm 0.0033}$	0.8422 ± 0.0053	0.8361 ± 0.0066
Effusion	$\mathbf{0.8309 \pm 0.0025}$	0.8261 ± 0.0008	0.8253 ± 0.0026
Emphysema	0.9106 ± 0.0050	$\mathbf{0.9127 \pm 0.0044}$	0.8999 ± 0.0027
Fibrosis	$\mathbf{0.8333 \pm 0.0047}$	0.8231 ± 0.0026	0.8111 ± 0.0098
Hernia	$\mathbf{0.9117 \pm 0.0053}$	0.9036 ± 0.0105	0.8721 ± 0.0089
Infiltration	$\mathbf{0.7046 \pm 0.0020}$	0.6967 ± 0.0067	0.6897 ± 0.0083
Mass	$\mathbf{0.8291 \pm 0.0056}$	0.8234 ± 0.0026	0.8077 ± 0.0066
Nodule	0.7721 ± 0.0078	$\mathbf{0.7727 \pm 0.0046}$	0.7518 ± 0.0035
Pleural Thickening	$\mathbf{0.7814 \pm 0.0057}$	0.7775 ± 0.0027	0.7682 ± 0.0047
Pneumonia	$\mathbf{0.7359 \pm 0.0041}$	0.7170 ± 0.0032	0.7101 ± 0.0053
Pneumothorax	$\mathbf{0.8670 \pm 0.0029}$	0.8610 ± 0.0031	0.8583 ± 0.0020
Mean	$\mathbf{0.8170 \pm 0.0015}$	0.8109 ± 0.0025	0.8019 ± 0.0025

Table 4.3: Comparison of performance metrics (mean ± standard deviation) for DenseNet-121 trained with and without data augmentation. The best results for each condition are shown in bold.

A possible limitation of this experiment is that the different augmentation techniques have a such significant effect that other hyperparameters is needed. Since the hyperparameters is selected by an optuna study with method 1, the performance of the method may be exaggerated as table 4.3 shows. Method 1 perform best of the tested methods.

4.3 Adam, NAdam and AdamW

This experiment tests 3 Adam based optimizers. There are a large number of optimizers which may give improved results on this task. For this experiment focus on Adam based optimizers since they do not require a new hyperparameter search.

Condition	**Adam**	**NAdam**	**AdamW**
Atelectasis	0.7827 ± 0.0017	$\mathbf{0.7832 \pm 0.0009}$	0.7811 ± 0.0024
Cardiomegaly	0.8759 ± 0.0015	$\mathbf{0.8792 \pm 0.0029}$	0.8751 ± 0.0041
Consolidation	0.7581 ± 0.0020	$\mathbf{0.7590 \pm 0.0018}$	0.7543 ± 0.0048
Edema	0.8518 ± 0.0020	0.8506 ± 0.0013	$\mathbf{0.8511 \pm 0.0033}$
Effusion	$\mathbf{0.8324 \pm 0.0011}$	0.8312 ± 0.0009	0.8309 ± 0.0025
Emphysema	0.9132 ± 0.0023	0.9133 ± 0.0019	$\mathbf{0.9106 \pm 0.0050}$
Fibrosis	$\mathbf{0.8339 \pm 0.0045}$	0.8335 ± 0.0047	0.8333 ± 0.0047
Hernia	0.9104 ± 0.0055	0.9128 ± 0.0122	$\mathbf{0.9117 \pm 0.0053}$
Infiltration	$\mathbf{0.7045 \pm 0.0036}$	0.7037 ± 0.0015	0.7046 ± 0.0020
Mass	0.8305 ± 0.0012	$\mathbf{0.8320 \pm 0.0012}$	0.8291 ± 0.0056
Nodule	0.7753 ± 0.0016	0.7765 ± 0.0020	$\mathbf{0.7721 \pm 0.0078}$
Pleural Thickening	0.7852 ± 0.0019	$\mathbf{0.7871 \pm 0.0031}$	0.7814 ± 0.0057
Pneumonia	$\mathbf{0.7339 \pm 0.0063}$	0.7326 ± 0.0042	0.7359 ± 0.0041
Pneumothorax	0.8658 ± 0.0028	0.8652 ± 0.0015	$\mathbf{0.8670 \pm 0.0029}$
Mean	0.8181 ± 0.0016	$\mathbf{0.8186 \pm 0.0028}$	0.8170 ± 0.0015

Table 4.4: Comparison of mean ± std values for DenseNet121 across three optimizers. Best results for each condition are bold.

As shown in table 4.4 There is no significant difference between the results of the optimizers.

The other experiments use the AdamW optimizer, as it performed best in the initial experiments.

4.4 DenseNet 121, 161 and 201

[1] compared model sizes for ResNet and found that smaller models resulted in higher AUC scores. This experiment tests whether the same principles applies to DenseNet. It tests DenseNet 121, 161 and 201. DenseNet121 is the smallest shallowest model, DenseNet161 is the model with the most parameters and DenseNet201 is the deepest model.

Condition	**DenseNet121**	**DenseNet161**	**DenseNet201**
Atelectasis	0.7811 ± 0.0024	$\mathbf{0.7839 \pm 0.0017}$	0.7798 ± 0.0032
Cardiomegaly	0.8751 ± 0.0041	0.8796 ± 0.0015	$\mathbf{0.8810 \pm 0.0037}$
Consolidation	0.7543 ± 0.0048	$\mathbf{0.7551 \pm 0.0027}$	0.7548 ± 0.0027
Edema	0.8511 ± 0.0033	$\mathbf{0.8532 \pm 0.0016}$	0.8503 ± 0.0022
Effusion	0.8309 ± 0.0025	$\mathbf{0.8317 \pm 0.0011}$	0.8279 ± 0.0027
Emphysema	0.9106 ± 0.0050	$\mathbf{0.9189 \pm 0.0017}$	0.9127 ± 0.0033
Fibrosis	0.8333 ± 0.0047	$\mathbf{0.8347 \pm 0.0036}$	0.8340 ± 0.0016
Hernia	0.9117 ± 0.0053	$\mathbf{0.9259 \pm 0.0050}$	0.9234 ± 0.0172
Infiltration	0.7046 ± 0.0020	$\mathbf{0.7056 \pm 0.0019}$	0.7053 ± 0.0011
Mass	0.8291 ± 0.0056	$\mathbf{0.8376 \pm 0.0028}$	0.8341 ± 0.0026
Nodule	0.7721 ± 0.0078	$\mathbf{0.7829 \pm 0.0023}$	0.7816 ± 0.0026
Pleural Thickening	0.7814 ± 0.0057	0.7862 ± 0.0019	$\mathbf{0.7876 \pm 0.0027}$
Pneumonia	0.7359 ± 0.0041	$\mathbf{0.7360 \pm 0.0035}$	0.7342 ± 0.0067
Pneumothorax	0.8670 ± 0.0029	$\mathbf{0.8675 \pm 0.0016}$	0.8653 ± 0.0027
Mean	0.8170 ± 0.0015	$\mathbf{0.8213 \pm 0.0010}$	0.8194 ± 0.0039

Table 4.5: Comparison of mean ± std AUC values for DenseNet of different sizes. Best results for each condition are bolded.

As shown if table 4.5, the best performing model is DenseNet161. There were not a single class where DenseNet121 were the best performing model. DenseNet161 performs best on the majority of classes, with DenseNet201 achieving the highest performance on the remaining classes. This suggest that the increase in parameter count on DenseNet161 causes better performance than the increase in layers in DenseNet201.

DenseNet121 is smaller and faster to train, which makes it the chosen candidate for testing in other experiments.

4.5 Single Channel Compared to Duplicated Channel Input

The used pretrained models operate on 3 channel images, while the Chest X-rays are one channel images. This problem has not been sufficiently investigated. This report evaluates two methods of handling the issue. The first method is to duplicate the channel to 3 identical channels, however after the standardization they will be different. The second method is to modify the model to only receive one channel. Here we initialize the first layer as the average of the existing 3 channels. Using this latter method reduces the image size significantly but nevertheless has a negligible impact on the computational cost.

Condition	**1 channel**	**3 channels**
Atelectasis	**0.7814 ± 0.0043**	0.7811 ± 0.0024
Cardiomegaly	**0.8760 ± 0.0055**	0.8751 ± 0.0041
Consolidation	**0.7566 ± 0.0026**	0.7543 ± 0.0048
Edema	**0.8516 ± 0.0010**	0.8511 ± 0.0033
Effusion	0.8305 ± 0.0008	**0.8309 ± 0.0025**
Emphysema	**0.9137 ± 0.0016**	0.9106 ± 0.0050
Fibrosis	0.8255 ± 0.0065	**0.8333 ± 0.0047**
Hernia	0.9075 ± 0.0145	**0.9117 ± 0.0053**
Infiltration	0.7040 ± 0.0016	**0.7046 ± 0.0020**
Mass	0.8289 ± 0.0044	**0.8291 ± 0.0056**
Nodule	**0.7746 ± 0.0062**	0.7721 ± 0.0078
Pleural Thickening	**0.7830 ± 0.0035**	0.7814 ± 0.0057
Pneumonia	0.7286 ± 0.0049	**0.7359 ± 0.0041**
Pneumothorax	0.8657 ± 0.0018	**0.8670 ± 0.0029**
Mean	0.8163 ± 0.0034	**0.8170 ± 0.0015**

Table 4.6: Comparison of mean ± std AUC values for DenseNet121 with 1 channel and 3 channels input. Best results for each condition are bolded.

As seen in table 4.6 there is no significant difference between the average AUC score. Some classes such as *Pneumonia*, *Fibrosis* and *Hernia* does show advantages to the duplication approach.

The duplicated (3 channels) approach is used in all other experiments unless specified.

4.6 Larger Image Size

This experiments seeks to determine if a larger image size allows the model to detect finer details and therefore increase its performance. This methodology has been shown to increase the average performance of ResNet[1], with particularly notable performance increases on physically smaller pathologies such as *Nodule* and *Mass*, while other pathologies largely retain the same or slightly lower performance.
For this report, 2 experiments have been performed with an increased image size of 448x448. One experiment tested the performance of DenseNet121 with larger images using the single channel method for handling input data. The second experiment used DenseNet121 on larger images, duplicating the single channel image into 3 channels. Due to increased image size, the dataset required more memory, and a lower batch size was therefore used. The experiment using a single channel used a batch size of 60, and the experiment using duplicated channels used a batch size of 32.

Condition	**1 channel**	**3 channels**
Atelectasis	**0.7883 ± 0.0013**	0.7880 ± 0.0011
Cardiomegaly	0.8726 ± 0.0023	**0.8734 ± 0.0024**
Consolidation	**0.7567 ± 0.0020**	0.7557 ± 0.0011
Edema	**0.8529 ± 0.0015**	0.8518 ± 0.0019
Effusion	**0.8362 ± 0.0012**	0.8353 ± 0.0009
Emphysema	0.9290 ± 0.0019	**0.9295 ± 0.0014**
Fibrosis	0.8403 ± 0.0011	**0.8414 ± 0.0039**
Hernia	0.8948 ± 0.0085	**0.9005 ± 0.0032**
Infiltration	0.7082 ± 0.0022	**0.7088 ± 0.0016**
Mass	0.8340 ± 0.0030	**0.8352 ± 0.0013**
Nodule	0.7953 ± 0.0025	**0.7954 ± 0.0005**
Pleural Thickening	**0.7963 ± 0.0017**	0.7951 ± 0.0031
Pneumonia	**0.7393 ± 0.0010**	0.7356 ± 0.0051
Pneumothorax	**0.8788 ± 0.0031**	0.8775 ± 0.0018
Mean	**0.8231 ± 0.0018**	0.8231 ± 0.0013

Table 4.7: Comparison of performance metrics (mean ± standard deviation) for 1 channel and 3 channels. Best results for each condition are shown in bold.

As seen in table 4.7, both implementations performed very similarly. Both implementations of a larger image resolution resulted in a higher average performance. As in prior literature[1] the increase in performance was particularly notable in small features like *Nodule* and *Mass*. The experiments also resulted in lower performance in *Emphysema* and *Hernia*. From these experiments, it seems that the benefits of increased image size apply to other models beside ResNet as well.

Image size	**224x224**		**448x448**	
Condition	**1 channel**	**3 channels**	**1 channel**	**3 channels**
Atelectasis	0.7814 ± 0.0043	0.7811 ± 0.0024	$\mathbf{0.7883 \pm 0.0013}$	0.7880 ± 0.0011
Cardiomegaly	0.8760 ± 0.0055	0.8751 ± 0.0041	0.8726 ± 0.0023	$\mathbf{0.8734 \pm 0.0024}$
Consolidation	0.7566 ± 0.0026	0.7543 ± 0.0048	0.7567 ± 0.0020	$\mathbf{0.7557 \pm 0.0011}$
Edema	0.8516 ± 0.0010	0.8511 ± 0.0033	$\mathbf{0.8529 \pm 0.0015}$	0.8518 ± 0.0019
Effusion	0.8305 ± 0.0008	0.8309 ± 0.0025	$\mathbf{0.8362 \pm 0.0012}$	0.8353 ± 0.0009
Emphysema	0.9137 ± 0.0016	0.9106 ± 0.0050	0.9290 ± 0.0019	$\mathbf{0.9295 \pm 0.0014}$
Fibrosis	0.8255 ± 0.0065	0.8333 ± 0.0047	0.8403 ± 0.0011	$\mathbf{0.8414 \pm 0.0039}$
Hernia	0.9075 ± 0.0145	$\mathbf{0.9117 \pm 0.0053}$	0.8948 ± 0.0085	0.9005 ± 0.0032
Infiltration	0.7040 ± 0.0016	0.7046 ± 0.0020	0.7082 ± 0.0022	$\mathbf{0.7088 \pm 0.0016}$
Mass	0.8289 ± 0.0044	0.8291 ± 0.0056	0.8340 ± 0.0030	$\mathbf{0.8352 \pm 0.0013}$
Nodule	0.7746 ± 0.0062	0.7721 ± 0.0078	0.7953 ± 0.0025	$\mathbf{0.7954 \pm 0.0005}$
Pleural Thickening	0.7830 ± 0.0035	0.7814 ± 0.0057	$\mathbf{0.7963 \pm 0.0017}$	0.7951 ± 0.0031
Pneumonia	0.7286 ± 0.0049	0.7359 ± 0.0041	$\mathbf{0.7393 \pm 0.0010}$	0.7356 ± 0.0051
Pneumothorax	0.8657 ± 0.0018	0.8670 ± 0.0029	$\mathbf{0.8788 \pm 0.0031}$	0.8775 ± 0.0018
Mean	0.8163 ± 0.0034	0.8170 ± 0.0015	$\mathbf{0.8231 \pm 0.0018}$	$\mathbf{0.8231 \pm 0.0013}$

Table 4.8: Tables 4.6 and 4.7 combined for improved clarity. Best results for each condition are shown in bold.

Chapter 5

Conclusions

This study investigates the application of CNNs for classifying thoracic diseases in the NIH ChestXray14 dataset. By comparing performance of pretrained models like DenseNet, ResNet, MobileNet, and VGG, along with various data preprocessing techniques, augmentation strategies, and optimizers on DenseNet121, the results highlight these findings: While the performance of DenseNet121 in initial tests made it seem to be the most promising model for further experimentation, other models have since shown better performance on the dataset. Particularly high performance can be observed by both ResNet50 and DenseNet161, which are models with over 25 million parameters, compared to DenseNet121's relatively small 8 million parameters. DenseNet121 therefore still presents itself as a favorable more lightweight alternative, which has been a valuable attribute in rapid experimentation and iteration.
Experiments on the impacts of data augmentation show that significant data augmentation gives large performance improvements over no data augmentation, and smaller improvements over moderate data augmentation on most classes, with no notable improvements on most remaining classes. The only exception to this is that of the *Cardiomegaly* class, which the models performed noticeably better on when no data augmentation was performed. It seems that the data augmentation is largely beneficial to the performance of a model, although it should be evaluated for each use case with care.
For each class, the change in optimizer between Adam, NAdam and AdamW had minimal impact, indicating that Adam-based optimizers are all similarly performant on DenseNet121. Another experiment resulting in minimally impactful findings is the experiment comparing performance between using a single channel input image or duplicating the single channel of the image into three identical values, as no significant difference between the two methods were found, and either is likely acceptable.
The experiments on image size show a significant increase in performance using 448x448 images compared to using 224x224 images in terms of average AUC, with *Hernia* being the only class to benefit from having smaller images. These results imply that larger images generally allow models to perform much better, though the increased image size requires reduced batch size for classification. One must therefore carefully evaluate the benefits and drawbacks for each use case.

Further work This project has due to time constraints committed early to using DenseNet121 for the majority of experiments. Later results show both ResNet50 and DenseNet161 as more performant, and as such there would be further value in repeating these experiments for these models to ensure that the findings are still applicable to the best models. Beyond using better singular models, there are other means of potentially improving the classification performance on the dataset which warrants investigation, such as the use of ensemble models or passing non-image data such as patient age and gender to the model.

Use of AI-tools

For coding of this assignment ChatGPT 4o[33] and GitHub Copilot[34] was used to write code, debug, and offer suggestions. For the writing of this report ChatGPT has been used to write BibTeX, format latex tables and help write the abstract.

5.1 Code

The code is avalible at https://github.com/Jon-Ingvar-Skanoy/Xrays/tree/delivery. It contain two pipelines, A and B. The main file used in the experiments is A_2_Train_densenet_main.py. with variations for the experiments.

Appendix A

Model hyperparameters

Hyperparameter	Value
`lr`	1.333×10^{-4}
`batch_size`	98
`grad_clip`	0.4784
`scheduler_name`	CosineAnnealingLR
`weight_decay`	1.886×10^{-6}
`betas`	(0.9, 0.999)
`eps`	1×10^{-8}
`amsgrad`	False
`optimizer`	AdamW
`num_epochs`	20
`patience`	3
`random_rotation`	7 degrees
`random_resized_crop`	Scale (0.08, 1.0), Ratio (3/4, 4/3)
`color_jitter`	Brightness=0.2, Contrast=0.2, Saturation=0.2, Hue=0.2

Table A.1: Hyperparameters used for training with AdamW and CosineAnnealingLR.

Hyperparameter	Value
`lr`	1.197×10^{-3}
`batch_size`	21
`grad_clip`	0.675
`random_rotation`	12 degrees
`color_jitter`	0.009
`scheduler_name`	OneCycleLR
`max_lr`	7.242×10^{-5}

Table A.2: Hyperparameters used for VGG16.

Hyperparameter	**Value**
`lr`	2.348×10^{-4}
`batch_size`	42
`grad_clip`	0.9708
`scheduler_name`	CosineAnnealingLR
`weight_decay`	1.3257×10^{-5}
`betas`	(0.9, 0.999) *(default)*
`eps`	1×10^{-8} *(default)*
`amsgrad`	False *(default)*
`optimizer`	AdamW
`num_epochs`	20
`patience`	3
`random_rotation`	7 degrees
`random_resized_crop`	Scale (0.08, 1.0), Ratio (3/4, 4/3)
`color_jitter`	Brightness=0.2, Contrast=0.2, Saturation=0.2, Hue=0.2

Table A.3: Hyperparameters used for ResNet50 with AdamW and CosineAnnealingLR scheduler.

Hyperparameter	**Value**
`lr`	2.291×10^{-4}
`batch_size`	46
`grad_clip`	0.6075
`scheduler_name`	CosineAnnealingLR
`weight_decay`	2.108×10^{-7}
`betas`	(0.9, 0.999)
`eps`	1×10^{-8}
`amsgrad`	False
`optimizer`	AdamW
`num_epochs`	20
`patience`	3
`random_rotation`	7 degrees
`random_resized_crop`	Scale (0.08, 1.0), Ratio (3/4, 4/3)
`color_jitter`	Brightness=0.2, Contrast=0.2, Saturation=0.2, Hue=0.2

Table A.4: Hyperparameters used for MobileNet V2 with AdamW and CosineAnnealingLR.

Appendix B

Hardware Setup

```
Shared JupyterLab Server:
    OS:         Ubuntu 22.04.4 LTS
    CPU:        Intel(R) Xeon(R) Platinum 8168 CPU @ 2.70GHz
    RAM:        1.48TB
    DISK:       1TB
    GPU0:       Tesla V100-SXM3-32GB

Local Machine 1:
    OS:         Windows 11, WSL2 w/Ubuntu 22.04.4 LTS
    CPU:        Intel(R) Core(TM) i5-9600K CPU @ 3.70GHz
    RAM:        32GB
    DISK:       480GB SSD + 22TB HDD
    GPU0:       NVIDIA GeForce RTX 3080 10GB
    GPU1:       NVIDIA GeForce RTX 3070 8GB

Local Machine 2:
    OS:         Windows 11, WSL2 w/Ubuntu 22.04.4 LTS
    CPU:        Intel(R) Core(TM) i7-10700 CPU @ 2.90GHz
    RAM:        16GB
    DISK:       1TB SSD + 7TB HDD
    GPU0:       NVIDIA GeForce RTX 3070 8GB

---------------------------------------

Most essential packages:
    WSL:            2.3.26.0
    Conda:          24.11.0
    Pip:            22.0.2
    Python:         3.10.15 (local) / 3.9.18 (server)
    PyTorch:        2.5.1
    CUDA:           12.4
    TorchVision:    0.20.1
    NumPy:          1.24.3
    Pandas:         2.2.3
    Matplotlib:     3.9.2
    Scikit-learn:   1.5.1
    Seaborn:        0.13.2
    JupyterLab:     4.2.6
```

Bibliography

[1] Ivo M Baltruschat et al. "Comparison of deep learning approaches for multi-label chest X-ray classification." In: *Scientific reports* 9.1 (2019), p. 6381.

[2] Xiaosong Wang et al. "Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 2097–2106.

[3] Neha Yadav, Anupam Yadav, and Manoj Kumar. "History of Neural Networks." In: *An Introduction to Neural Network Methods for Differential Equations*. Dordrecht: Springer Netherlands, 2015, pp. 13–15. ISBN: 978-94-017-9816-7. DOI: 10.1007/978-94-017-9816-7_2. URL: https://doi.org/10.1007/978-94-017-9816-7_2.

[4] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." In: *nature* 521.7553 (2015), pp. 436–444.

[5] Rukshan Pramoditha. *The Concept of Artificial Neurons (Perceptrons) in Neural Networks*. URL: https://towardsdatascience.com/the-concept-of-artificial-neurons-perceptrons-in-neural-networks-fab22249cbfc. (accessed: 02.12.2024).

[6] National Institute of Neurological Disorders and Stroke. *Brain Basics: The Life and Death of a Neuron*. URL: https://www.ninds.nih.gov/health-information/public-education/brain-basics/brain-basics-life-and-death-neuron. (accessed: 02.12.2024).

[7] H. Kinsley and D. Kukieła. *Neural Networks from Scratch in Python*. Harrison Kinsley, 2020. URL: https://nnfs.io/.

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[9] Saleh Albelwi and Ausif Mahmood. "A Framework for Designing the Architectures of Deep Convolutional Neural Networks." In: *Entropy* 19 (2017), p. 242. URL: https://api.semanticscholar.org/CorpusID:31031734.

[10] J. Deng et al. "ImageNet: A Large-Scale Hierarchical Image Database." In: *CVPR09*. 2009.

[11] Hoo-Chang Shin et al. "Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset characteristics and transfer learning." In: *IEEE transactions on medical imaging* 35.5 (2016), pp. 1285–1298.

[12] Gao Huang et al. *Densely Connected Convolutional Networks*. 2018. arXiv: 1608.06993 [cs.CV]. URL: https://arxiv.org/abs/1608.06993.

[13] Kaiming He et al. "Deep residual learning for image recognition." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[14] Karen Simonyan. "Very deep convolutional networks for large-scale image recognition." In: *arXiv preprint arXiv:1409.1556* (2014).

[15] Andrew G. Howard et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications." In: *ArXiv* abs/1704.04861 (2017). URL: https://api.semanticscholar.org/CorpusID:12670695.

[16] Mark Sandler et al. "Mobilenetv2: Inverted residuals and linear bottlenecks." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 4510–4520.

[17] Christian Szegedy et al. "Going deeper with convolutions." In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2014), pp. 1–9. URL: https://api.semanticscholar.org/CorpusID:206592484.

[18] Laleh Seyyed-Kalantari et al. *CheXclusion: Fairness gaps in deep chest X-ray classifiers*. 2020. arXiv: 2003.00827 [cs.CV]. URL: https://arxiv.org/abs/2003.00827.

[19] Seng Hansun et al. "Machine and deep learning for tuberculosis detection on chest x-rays: systematic literature review." In: *Journal of medical Internet research* 25 (2023), e43154.

[20] Pranav Rajpurkar et al. "CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning." In: *ArXiv* abs/1711.05225 (2017). URL: https://api.semanticscholar.org/CorpusID:40094999.

[21] Zongyuan Ge et al. "Chest X-rays Classification: A Multi-Label and Fine-Grained Problem." In: *ArXiv* abs/1807.07247 (2018). URL: https://api.semanticscholar.org/CorpusID:49881466.

[22] S. M. Nabil Ashraf et al. "SynthEnsemble: A Fusion of CNN, Vision Transformer, and Hybrid Models for Multi-Label Chest X-Ray Classification." In: *2023 26th International Conference on Computer and Information Technology (ICCIT)* (2023), pp. 1–6. URL: https://api.semanticscholar.org/CorpusID:265158168.

[23] Ophir Gozes and Hayit Greenspan. "Deep Feature Learning from a Hospital-Scale Chest X-ray Dataset with Application to TB Detection on a Small-Scale Dataset." In: *2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)* (2019), pp. 4076–4079. URL: https://api.semanticscholar.org/CorpusID:173990631.

[24] Laleh Seyyed-Kalantari et al. "CheXclusion: Fairness gaps in deep chest X-ray classifiers." In: (2021).

[25] Paperswithcode community. *ChestX-ray14*. URL: https://paperswithcode.com/dataset/chestx-ray14. (accessed: 04.12.2024).

[26] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge." In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[27] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV]. URL: https://arxiv.org/abs/1512.03385.

[28] Andrew G. Howard et al. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. 2017. arXiv: 1704.04861 [cs.CV]. URL: https://arxiv.org/abs/1704.04861.

[29] Mark Sandler et al. *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. 2019. arXiv: 1801.04381 [cs.CV]. URL: https://arxiv.org/abs/1801.04381.

[30] Tianbao Yang and Yiming Ying. "AUC maximization in the era of big data and AI: A survey." In: *ACM Computing Surveys* 55.8 (2022), pp. 1–37.

[31] Takuya Akiba et al. "Optuna: A next-generation hyperparameter optimization framework." In: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2019, pp. 2623–2631.

[32] Jingzhao Zhang et al. *Why gradient clipping accelerates training: A theoretical justification for adaptivity*. 2020. arXiv: 1905.11881 [math.OC]. URL: https://arxiv.org/abs/1905.11881.

[33] OpenAI. *ChatGPT 4.0*. https://openai.com/chatgpt. Accessed: 2024-12-06. 2024.

[34] GitHub. *GitHub Copilot*. https://github.com/features/copilot. Accessed: 2024-12-06. 2024.