UNIVERSITY OF AMSTERDAM

MSc Artificial Intelligence

Track: Machine Learning

Master Thesis

Data-efficient learning for pulmonary nodule detection

by

Marysia Winkels

10163727

Supervisor:
Prof. Dr. M. (Max) Welling

Assessor:
Dr A Person

Daily Supervisor:
T. S. (Taco) Cohen MSc

Abstract

Convolutional neural networks – the methodology of choice for image analysis – typically require a large amount of annotated data to learn from, which is difficult to obtain within the medical domain. This work attempts to reduce the sample complexity for automated medical image analysis tasks by building prior knowledge regarding 3D symmetries into the network itself through g-convolutions that replace regular convolutional layers. The G-Convolutional Neural Networks were applied to the problem of false positive reduction for pulmonary nodule detection, and proved to be more effective in terms of performance, sensitivity to malignant nodules and speed of convergence compared to a baseline architecture with regular convolutions. For every dataset size, results similar to the baseline were achieved with G-CNNs trained on only a tenth of the data and the trend is that the smaller the available dataset was, the larger the difference in performance became.

Acknowledgments

First and foremost, I would like to thank my supervisor Taco for his guidance throughout this process. He was always available to bounce ideas off of and provided excellent feedback, but also helped me stay focused on the task at hand. Although I was not familiar with any work on group theory before I started this project, he was always quick to answer any question any question I might have. I especially valued his continued positive and encouraging attitude, and it is because of his work on group-convolutions that I even had the opportunity to familiarise myself with this interesting subject, which was a great learning experience.

I would also like to thank the wonderful team at Aidence, where I first became familiar with automated medical image analysis. Aidence provided me with both the resources and the expertise to successfully work on this project and offered an extremely inspiring work environment. I thoroughly enjoyed the many discussions about deep learning and CAD development, and benefited greatly from the provided medical expertise. Most importantly, I am grateful to Mark-Jan and Jeroen for their patience and continued support throughout this process and for offering me this opportunity to learn.

In addition, I would like to express my gratitude the University of Amsterdam student services and the AUF whose efforts greatly added to the completion of this thesis. Lastly, I want to thank my proofreaders and close friends who have been tremendously helpful and supportive.

Contents

1

Motivation

Lung cancer is currently the leading cause of cancer-related deaths worldwide, accounting for an estimated 1.7 million deaths globally *each year* and 270.000 in the European Union alone [1,2], taking more victims than breast cancer, colon cancer and prostate cancer combined. [3] The clinically diagnosed mortality rate is high with an overall survival rate of less than 15% [4], which can largely be attributed to the fact that at present, nearly 80% of lung cancer is diagnosed at an advanced stage. At that point, the cancer has already metastasised to other parts of the body and the patient has little curative treatment options left available. [5] Reason for this late diagnosis is that even the first symptoms – such as a persisting cough or shortness of breath – generally do not present themselves until the cancer is at a later stage [5], making early detection difficult, even though it is evidently crucial if we wish to reduce the mortality rate.

Fortunately, as it is well-known that the single biggest risk factor is long-term tobacco smoking (accounting for over 85% of cases[6]), lung cancer is an ideal candidate for *cancer screening* – the process of testing a seemingly healthy part of the population to identify the disease before any symptoms appear. The effectiveness of lung cancer screening in terms of cost-reduction and increased survival rate has been investigated through randomised controlled trials, most notable of which due to its scale was the National Lung Screening Trial in the U.S.A. The NLST led to the recommendation that screening should be implemented for people in high risk groups for developing lung cancer, and other initiatives came to similar conclusions.[7–13] Additionally, in November 2017, a white paper was released in which the radiologists involved, based on consensus through discussion with experts from eight European countries undertaking trials of lung cancer screening, strongly encouraged the planning for implementation of screening throughout Europe as soon as possible, preferably within an 18 month period.[14]

Nevertheless, a key element to the (cost-)effectiveness of screening is the performance of the reader – in this case the radiologist. Radiologists are the specialists in medical imaging responsible for interpreting the image and communicating the findings to the treating physician. In case of CT images made for lung cancer screening, the radiologist is tasked with identifying suspect lesions in the form of pulmonary nodules, as these may be malignant (*i.e.* cancerous). However, research has revealed a high inter-observer variability between radiologists[15–17], and although medium to large nodules are detected fairly consistently, quality of nodule detection diminishes substantially as nodule size decreases.[18]

A way to reduce these observational oversights would be with *double* or *second readings*, a practice in which two readers independently interpret an image examination and combine findings, thereby reducing errors.[19,20] Although double readings have proven to boost sensitivity, it is not realistically a feasible option due to the steadily growing workload of radiologists[21], and even more so considering the additional number of scans screenings would produce once implemented.

A solution to this, rather than require an extra radiologist to read each chest CT, would be

to introduce computer-aided detection (CADe) technology as the second reader to improve performance.[22,23] CADe systems for the lung aim to assist the reader in reporting pulmonary nodules by presenting its detected findings after the radiologist first examined the image, leaving the ultimate judgement up to the expert radiologist.

CAD(1) systems were initially met with scepticism, as historically their performance was poor, back when they were still largely rule-based systems or created using machine learning with classical feature engineering.[24] Since then, excitement has grown tremendously as deep learning became the methodology of choice for analysing images.[25] With regards to pulmonary nodule image analysis, deep learning techniques unambiguously outperform classical machine learning approaches[26], making CAD a potentially valuable addition to the radiologists. Deep learning techniques, however, typically require a substantial amount of labeled data to learn from – something that is scarce in the medical imaging community both due to patient privacy concerns and the difficulty of obtaining high-quality annotations in large quantities.

The challenge this presents is that of *data efficiency*: the ability to learn in complex domains without requiring large quantities of data. Various approaches exist that attempt to tackle this problem, including semi-supervised learning techniques, active learning or incorporating explicit domain-specific knowledge, but in this work we will try to utilise structural knowledge of the data (specifically related to symmetries of the labels) in an attempt to achieve the same performance with a smaller amount of data. The overall aim is to aid the development of CAD technology in the biomedical imaging domain in general and pulmonary nodule detection systems in particular by exploiting the notion that neural networks will be more data-efficient when prior structural knowledge about symmetries of the data is built into the network itself.

The contribution of this work will first and foremost be an extension of the `GrouPy` python package originally developed by Cohen & Welling[27] to include g-convolutions – a type of layer that can be used as a drop-in replacement for spatial convolutions in

(1) A distinction is often made between CADe (computer aided detection) and CADx (computer aided diagnosis) systems. CAD encompasses both and will be used interchangeably with CADe.

modern network architectures – for 3D, making them suitable for volumetric CT images. Secondly, we aim to provide an analysis as to whether pulmonary nodule detection benefits from group-convolutions in terms of data-efficiency, training speed and performance. For practical purposes, this thesis will focus on the distinction between pulmonary nodules and non-nodules, but the results will hopefully generalise to other 3D biomedical volumes and benefit the overall development of CAD systems.

To help understand the research that has been done, the reader will first be presented with an introduction on deep learning concepts of relevance, in particular the workings of the convolutional layer in Ch. 2. Next, Ch. 3 will provide a medical background on pulmonary nodule to help gain an understanding of the difficulties of the task at hand, the image acquisition protocol and available datasets, as well as a brief historic overview of research on lung CAD systems with deep learning. Ch. 4 will introduce the concepts of invariance and equivariance, related work on deep learning and achieving rotational equivariance, and the basic concepts from group theory required to understand the details with respect to the implementation of group-equivariant neural networks for 3D. Ch. 5 will detail our methodologies and experiments, and Ch. 6 presents the results. Lastly, we will conclude and address issues that may arise from our results in Ch. 7. Additionally, an appendix is provided for the interested reader with medical and mathematical details, additional figures and additional experiment results amongst other things, but will not be necessary to fully understand the scope of this study.

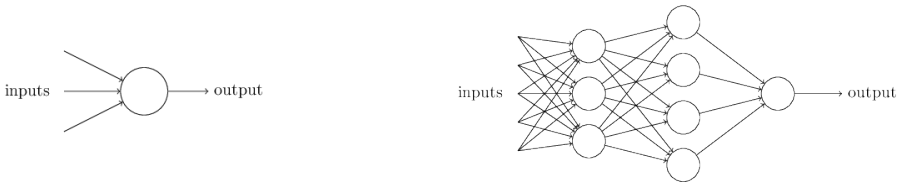
2

Deep Learning

Neural networks are a subset of *machine learning* techniques, which is the science of automated pattern recognition in data. It works by providing an algorithm with examples during the training phase in order to tune the learnable weight parameters in such a way that it can create predictions for unseen data samples. Supervised learning indicates we provide the target class labels for the data (*e.g.* nodule and non-nodule) to help guide the algorithm what to learn.

The simplest example of a neural network is a *perceptron*, as illustrated in figure 2.1a, and is suitable for very simple problems such as the logical OR and AND operations. A neuron (also *node*) is an element of the network where inputs (vector $\mathbf{x}$) are combined with weights (vector $\mathbf{w}$) , a bias (b) and a non-linear activation function (σ) to produce an output value in the form $\texttt{output} = \sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x} + b)$. The weights of the neural net can be

tuned during *training* by providing the network with example data; a combination of input ($\mathbf{x}$) and the output the neural network should provide (label y). The non-linearity is introduced as linear combinations alone will not be able to accurately express problems that are not linearly separable.



(a) A perceptron with a single neuron **(b)** Complex combination of neurons

Figure 2.1: *Neural networks*

A column of neurons stacked together that can receive the same inputs, as illustrated in figure 2.1b, is a *layer*, and the intermediate layers between the inputs and outputs are the *hidden layers*. A neural network is considered *deep* when it contains many hidden layers, and can provide solutions to more complicated and subtle decision problems. Over time, deep neural networks have become the methodology of choice for many intricate decision-making problems, including nodule detection. This chapter aims to provide a historic overview of how deep neural networks gained popularity for image processing tasks and a theoretical background, in particular with regards to the *convolutional* layer, to serve as a foundation for the extension for the rest of this work.

2.1 History

The perceptron, as seen in 2.1a, was one of the first artificial neural networks. It was introduced Frank Rosenblatt in 1958 [28] and was intended to model how the human brain processes visual information, hence the name artificial *neural* network. The weights of the neural net can be adjusted throughout training in an iterative manner to fit the

given data. This is known as *gradient descent*, which is the process of minimising a function by following the gradients. Though initially computationally intractable, a fast algorithm for computing such gradients known as *backpropagation* was originally introduced in the 1970s, and gained popularity due to the 1986 paper by Rumelhart, Hinton and Williams.[29]

However, it wasn't until the mid-2000s that state-of-the-art results could be achieved with neural nets and it was empirically demonstrated that good high-level representations of the data could be learned.[30–32] This is one of a combination of factors that gave rise to the emergence of deep neural networks as the method of choice for many problems in the domain of artificial intelligence. Others include the introduction of GPUs that had the computational power to make the sheer number of computations necessary for the training of large models feasible (leading to a speed increase of nearly a hundred-fold[33]), the increasing availability of large annotated datasets[34] and the introduction of additional effective neural network components such as dropout[35] (regularisation technique) and ReLU[36] (activation function).

Computer vision, and specifically image classification, was one of the fields of artificial intelligence where the use of deep neural networks resulted in an increase in performance, as proven by the outstanding results of Szegedy et al. in 2015,[37] that showed an increase in accuracy of classifying images in the 1000 classes from ImageNet from approximately 75% in 2011[38] to a near perfect score in 2015,[37] even said to exceed human accuracy.[39] This jump in performance is largely to be attributed to the contribution to the ImageNet competition in 2012 by Krizhevsky, Sutskever and Hinton[40] who used *convolutional* layers in their neural network architecture (dubbed AlexNet) to almost half the error rate from for object recognition, which lead to the rapid adoption of deep learning by the computer vision community. Deep learning, and convolutional neural networks in particular, have since also been the method of choice for the development of computer-aided detection and diagnostics systems.[41,42]

2.2 Theoretical background

The process of training a neural net can be summarised as the optimisation of parametric function g (see Eq. 2.1) with respect to its weight parameters $\theta^{(1)}$ to fit data sample $\{x_i\}_{i=1}^{N}$, where N is the number of data samples in the training set. Function g is a composite function of the function $\{f_l\}_{l=0}^{L}$ that calculates the output of hidden layer l, where L is the number of hidden layers:

$$g(x_i|\theta_0, \ldots, \theta_L) = f_L(f_{L-1}(\ldots f_0(x_i|\theta_0)|\ldots\theta_{L-1})|\theta_L) \tag{2.1}$$

A data sample is passed forward through the network, and the optimisation is done by recursively computing the error backwards from the last layer following the chain-rule and updating the weights w.r.t. to the known target output. This notion that the derivative of the error can be computed recursively from the last layer back using the chain-rule is what is known as *backpropagation* and drastically increased feasibility of calculating the gradients considering the number of parameters required to capture the complexity of the problems at hand. However, it does require that all elements of the neural network should be differentiable. The process of following the gradients of the error function towards a minimum value is *gradient descent*.

A forward pass of a data sample through the network combined with a backward pass of the error to update the weights for all data samples is called an *epoch*. A network typically trains for a number of epochs, processing each data sample multiple times and updating the weights accordingly, until it hits some stopping criteria (*e.g.* time, epoch limit). The performance of the model is not only dependent on the chosen structure (*i.e.* types of layers, type of non-linearities, number hidden layers, etc.), but also on the training method used to set the network's parameters (such as the choice of error-function, weight

(1) Previously known as vector $\mathbf{w}$ for a single layer, θ is a matrix that contains all weights in the network and θ_l are the weights of layer l.

initialisation and weight update rule) and the applied regularisation techniques.

2.2.1 Hyperparameters

The choices involved with determining the method of training the network can be described as setting the training hyperparameters. These include how we define determine performance of the network (choice of cost function), how the network's weights are initialised and the manner in which they are updated (update rule and optimizer).

Firstly though, we wish to define *how* the training data is offered to the network and specifically at what point the weights are updated. Gradient descent comes in three distinct flavours regarding data presentation: stochastic gradient descent (SGD), batch gradient descent and mini-batch gradient descent. SGD calculates the error and updates the model for each sample in the training set separately, which can result in faster learning and avoidance of premature convergence in some cases, but is also computationally expensive. Batch gradient descent, on the other hand, computes the error for each sample of the training set individually, but updates the model only after the entire training set has been processed, which may have advantages such as computational efficiency and a more stable error gradient in some cases, but - especially with large datasets - this may become slow or even impossible due to memory constraints.

Mini-batch gradient descent is a compromis between the SGD and batch gradient descent, separating the training set into smaller batches of b samples per batch which are all passed through the network before the model is updated. The choice of batch size b is a trade-off between the time efficiency of training and the noisiness of the gradient estimate. The convention is to use mini-batch gradient descent, rather than SGD or batch gradient descent, though the choice of b may vary (and in itself also influence also hyper parameters related to the weight update rule such as learning rate).

For mini-batch gradient descent, such as all forms of gradient descent, choices with regards to the error function, weight update rule and weight initialisation need to be

specified, which will be expanded upon in the subsections beneath.

Error function

Firstly, the error function (also known as *cost* or *loss* function) evaluates the performance of a neural network during training based on the provided input and expected output and serves as a guidance for the gradient descent algorithm. A common general purpose loss function, most often used for regression problems, is the mean squared error (see Eq. 2.2), where the distance between the predicted output ($\hat{y}_i$) and ground-truth label (y_i) for all data samples x_i are squared to penalise large differences.

$$L = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \qquad \text{(MSE)} \tag{2.2}$$

However, in case of classification the output of the network is often a probability distribution over classes, in which case it is an appropriate choice to use *cross-entropy* as the loss function. Binary cross-entropy (see Eq. 2.3) is a special case of categorical cross-entropy (see Eq. 2.4) for two classes ($c = 2$). In these equations, $\hat{y}_{ij}$ is the output of the network and probability prediction that x_i belongs to the j^{th} class. The sum of probabilities over all classes c for $\hat{y}_i$ is one.

$$L_{\text{binary}} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right] \tag{2.3}$$

$$L_{\text{categorical}} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{c}\left[y_{ij}\log(\hat{y}_{ij})\right] \qquad \text{(Cross-entropy)} \tag{2.4}$$

Cross-entropy is a preferred loss function for classification problems over classification error (or accuracy) as it takes the confidence level of prediction into account and is

therefore a more granular method to compute error. It is also preferred over MSE as classification problems work with a very particular set of possible output values (the labels representing the different classes), which can not be encapsulated with MSE.

Weight updates

As we are now familiar with manners to define the performance of the network in terms of the error or loss, one can understand that the derivative of the loss function can be calculated with respect to the weights after each pass of the batch through the network, and the weights can be adjusted accordingly. However, we will need to define what *accordingly* means in this context.

If we were to add the full value of the derivative to the weights directly, the weights would fluctuate extremely with each update. Instead, the derivative can be multiplied with a small value η typically between 1.0 and 10^{-6}, called the *learning rate*. Eq. 2.5 demonstrates how weight θ is updated by adding the derivative of the current weight, multiplied with the learning rate. If the learning rate η is too small, the model will converge very slowly, while if η is too large, fine-grained modifications to the weights cannot be made, resulting in divergence.

$$\theta = \theta + \eta \cdot \frac{\partial}{\partial \theta} L(\theta) \qquad \text{(weight update with } \eta\text{)} \qquad (2.5)$$

However, with monotonic steps the optimisation remains a slow process that can easily get stuck in local minima. The weight update rule for gradient descent can be improved upon by introducing *momentum* for faster convergence.[43] SGD with momentum not only takes the gradient estimation of the current batch, but also the estimations that were made for the previous batches taken into account. This essentially favours the overall direction over multiple batches combined by computing the current gradient and updating with respect to the accumulated gradient, resulting in a faster approach of the global minimum. Instead of Eq. 2.5 to update the weights, SGD with momentum uses

Eq. 2.6, where $\gamma \in (0, 1)$ (commonly set to 0.9) and v is a velocity vector initialised at zero.

$$\begin{aligned} v &= \gamma v - \eta \cdot \frac{\partial}{\partial \theta} L(\theta) && \textit{so that} \\ \theta &= \theta + v && \text{(Momentum)} \end{aligned} \tag{2.6}$$

An extension of momentum is *Nesterov momentum*, which exploits the notion that by knowing the momentum term, we know the approximate future position and can compute the gradient at that point instead, resulting in update rule Eq. 2.7.[43]

$$\begin{aligned} v &= \gamma v - \eta \cdot \frac{\partial}{\partial \theta} L(\theta + \gamma v) && \textit{so that} \\ \theta &= \theta + v && \text{(Nesterov)} \end{aligned} \tag{2.7}$$

However, other update rules such as AdaGrad[44], AdaDelta[45], RMSprop[46] and Adam[47] exist that alter the update rule by essentially *adapting the learning rate* based on the gradient history. *AdaGrad* scales the learning rate parameter η by dividing the current gradient by the accumulated previous gradients, resulting in a smaller learning rate when the gradient is large and vice-versa. A problem with AdaGrad, however, are the rapidly decreasing learning rates as the accumulated sum of gradients grows throughout training. *AdaDelta* is an extension of AdaGrad aims to fix this by restricting the history of the gradients to a fixed number, using a decaying average. A similar, but slightly different manner to combat the problem of accumulated sum of gradients, known as *RMSprop*, was an unpublished method proposed by Hinton in a lecture of online Coursera Class on Neural Networks for Machine Learning that adapts the learning rate by dividing it by an exponentially decaying average of the squared gradients, rather than the sum like AdaGrad does.

Lastly, *Adam* is an adaptive learning rate method similar to RMSprop with the addition of momentum. The Adam update rule for the weights is specified in Eq. 2.8, where m and v are estimates of the mean and uncentered variance, corrected to $\hat{m}$ and $\hat{v}$ to counter-act the bias towards zero in the initial time steps as they are initialised to zero. The decay rates β_1 and β_2 are typically zet to 0.9 and 0.999, and smoothing term ε that avoids division by zero to 10^8

$$
\begin{aligned}
m &= \beta_1 m + (1-\beta_1)\frac{\partial}{\partial\theta}L(\theta) & \hat{m} &= \frac{m}{1-\beta_1} \\
v &= \beta_2 v + (1-\beta_2)\left(\frac{\partial}{\partial\theta}L(\theta)\right)^2 & \hat{v} &= \frac{v}{1-\beta_2} \\
\theta &= \theta + \frac{\eta}{\sqrt{\hat{v}}+\varepsilon}\hat{m} && \text{(Adam)}
\end{aligned}
\tag{2.8}
$$

To summarise, gradient descent optimisation algorithms are algorithms that ensure a better and faster convergence for gradient descent, either through the introduction of momentum or by adapting the learning rate, and Adam is an adaptive learning rate method that includes momentum for each parameter.

Weight initialisation

Although possible update rules for the weights were discussed, an initial choice of the weight values needs to be made before these updates can occur. If, according to the naive strategy, all the initial values would be set to zero, the network would not be able to learn as weights that are set to the same value will receive the same weight updates.

To break this symmetry, one could for example sample random numbers from a uniform distribution. However, as the number of inputs neurons grow, so will the variance. Instead, an alternative is to scale the variance based on the number of input (n_{in}) and output (n_{out}) neurons, such that the variance remains the same with each passing layer. This is known as Xavier initialisation [48], where the weights can be drawn from a

truncated Gaussian distribution[2] with a zero mean and a standard deviation as indicated in Eq. 2.9, or from a uniform distribution with range $[-\text{limit}, \text{limit}]$, where the limit is also indicated in Eq. 2.9.

A slight alteration to Xavier initialisation was proposed by He *et al.* , who discovered that effectively doubling the size of the weight variance has a beneficial effect for rectified activations in particular, as ReLU and other rectifying activation functions return zero for half of their input. Their proposed standard deviation for the truncated normal distribution and limit for the uniform distribution is specified in Eq. 2.10 and is similar to Xavier initialisation, but only takes the input neurons n_{in} into account.

$$\sigma = \sqrt{\frac{2}{n_{\text{in}} + n_{\text{out}}}} \qquad \text{limit} = \sqrt{\frac{6}{n_{\text{in}} + n_{\text{out}}}} \qquad \text{(Xavier)} \tag{2.9}$$

$$\sigma = \sqrt{\frac{2}{n_{\text{in}}}} \qquad \text{limit} = \sqrt{\frac{6}{n_{\text{in}}}} \qquad \text{(He)} \tag{2.10}$$

2.2.2 Layers

Even when the hyperparameters have been set, there is still some choice to be made with regards to the *architecture* of the neural network. The architecture of a neural network defines the manner in which various layers stack together.

A simple example of a neural network was presented in the introduction, where all nodes of the first and second hidden layer were connected and the output could therefore be computed with $\sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x} + b)$, where σ is the activation function. A layer that is fully-connected like that is known as a *dense* or *fully-connected* layer. However, this does not mean creating a neural network is simply the process of stacking multiple fully-connected layers together. Firstly, there are various choices of activation functions available, each with their own use cases. Secondly, combining multiple fully-connected layers is rarely sufficient to achieve good results for computer vision problems as this

[2] Truncated indicates that values with more than $2 \cdot \sigma$ deviation from the mean are redrawn.

would result in a high number of weights. Even with small images, *e.g.* of size $32 \times 32 \times 3$ (height, width and number of colour channels respectively; standard RGB CIFAR-10 image), a single (fully-connected) neuron in the first hidden layer would already have 3072 weight parameters associated with it, and deep nets consist of many hidden layers.

Instead, other types of layers can be included in the architecture, most notably *convolutional layers* for image-related problems. A network with convolutional layers is a *Convolutional Neural Net* (*ConvNet* or *CNN* for short). The architecture of a CNN will often consist of a combination of convolutional layers, max pooling, and dense layers along with activations and regularisation layers. A common pattern for a CNN to follow is to stack convolution layers followed by an activation (possibly with a normalisation layer in between) and from time to time follow this up with a pooling layer to reduce the spatial dimension, until the output is small enough that it can be followed up with one or two fully-connected layers.

Convolution

A *convolution* operation on an input image is the process of producing an output by combing each pixel of an image with its local neighbours, weighted by a *kernel*. Imagine an input I with convolution kernel K. The value for a position i in the output can be specified as $V_i = \sum_j I_{i+k-j} K_j$, where j is the index over the kernel and k the center position of the kernel, and can be intuitively seen as calculating the dot product between vector I and the reverse of vector K. A simple example of a convolution is that of the identity operation. Figure 2.2a shows how a 3×3 convolution kernel can be used as an identity operator to produce an output value for the center position. As all weights besides that of the center position are zero, these do not contribute towards the output value, whereas the center value does.[3] Quite similarly, figure 2.2b illustrates how a convolution can be used to essentially translate the input by giving a weight to the value at a different position.[4]

[3] $V_4 = \sum_8 I_{4+4-j} K_j = (0 \cdot 1) + (0 \cdot 2) + 0 \cdot 3) + (0 \cdot 4) + (1 \cdot 5) + (0 \cdot 6) + (0 \cdot 7) + (0 \cdot 8) + (0 \cdot 9) = 5$
[4] $V_4 = \sum_8 I_{4+4-j} K_j = (0 \cdot 1) + (0 \cdot 2) + 0 \cdot 3) + (0 \cdot 4) + (0 \cdot 5) + (0 \cdot 6) + (0 \cdot 7) + (0 \cdot 8) + (1 \cdot 9) = 9$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & \underline{5} & 6 \\ 7 & 8 & 9 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} \dots & \dots & \dots \\ \dots & 5 & \dots \\ \dots & \dots & \dots \end{bmatrix}$$

(a) Identity operation

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & \underline{5} & 6 \\ 7 & 8 & 9 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} \dots & \dots & \dots \\ \dots & 9 & \dots \\ \dots & \dots & \dots \end{bmatrix}$$

(b) Translation convolution

Figure 2.2: 3×3 *convolution filters applied to an input.*

The translation example immediately illustrates the problem with convolutions and the edges of an image: how can, for instance, the value for the last position be calculated, which does not *have* a 3×3 neighbourhood, and therefore also no value with which the weight of one can be multiplied? One solution is to discard the edge cases in their entirety, producing no value at all at those positions at which the kernel cannot be validly applied to the position – called *valid padding*. An consequence of this approach, however, is that the output will have less values than the input, which may be undesirable. An alternative therefore is *same padding*, which ensures the output of the convolution is of a similar size to the input by filling in values for non-existing positions, such as zero, the average of the neighbourhood, or the nearest neighbour. The differences between same and valid padding are illustrated in Figure 2.3.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & \underline{5} & 6 \\ 7 & 8 & 9 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} & & \\ & 9 & \\ & & \end{bmatrix} \text{ Valid padding}$$

$$\begin{bmatrix} 5 & 6 & 6 \\ 8 & 9 & 9 \\ 8 & 9 & 9 \end{bmatrix} \text{ Same padding}$$

Figure 2.3: *Translation convolution with same and valid padding*

These mathematical details may understandably leave the reader to wonder how

$$\begin{bmatrix} \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \\ \frac{1}{8} & \frac{1}{4} & \frac{1}{8} \\ \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \end{bmatrix}$$

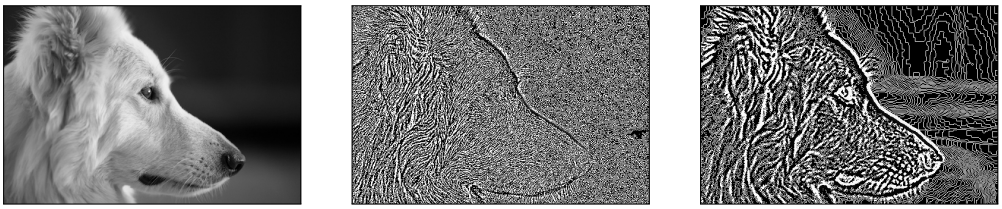
(a) Gaussian blur ($\sigma \approx .8$)

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

(b) Laplacian kernel

Figure 2.4: *Discrete approximations of convolution filters for edge detection.*

convolutions relate to convolutional neural networks for images. Firstly, even before the introduction of convolutional layers in deep neural networks, convolution operators were widely used in the field of computer vision as many simple transformations to an image could be achieved by applying a convolution. Examples include simple operations, such as the previously discussed translation, adjusting image intensity and applying motion blur, but also more sophisticated image alteration techniques such as unsharp masking (subtracting an image blurred with a Gaussian mask from the original image), optical blur (a translation of the original pixel to surrounding pixels with decreasing intensity), and even edge detection. Edges detection, essentially, is the process of highlighting regions of rapid intensity change, which can be achieved applying a Laplacian kernel (see figure 2.4b) that calculates the sum of differences over the neighbours, and can be improved by first smoothing the image (see figure 2.4a) to reduce sensitivity to noise. The combination of smoothing with a Gaussian kernel and applying the Laplacian operator is known as Laplacian of Gaussian, and an application of these techniques to an image is illustrated in Figure 2.5.



(a) Original image **(b)** Laplacian kernel **(c)** Laplacian of Gaussian

Figure 2.5: *Example of convolution filters for edge detection applied to input image.*

The applications of convolutions in image processing have their limits however – rotation,

for example, is not possible to achieve through a convolution operation as not every pixel within the image undergoes the same transformation. Points near the center of rotation c are translated to a point at a lesser distance from their original coordinates than those points further away from center c. For a similar reason, image scaling is impossible with convolutions as well.

Nevertheless, these operators are immensely useful and especially so within deep neural networks precisely for their ability to perform these image transformations. Convolutional layers in neural networks essentially filter out (hence the use of the word *filter* instead of kernel in the context of CNNs) information, and can therefore extract features of the images such as edges. At each layer, the output of the previous layer is used to extract multiple features into *feature maps* (or *channels*), which in turn form the output of that layer. As in each consecutive layer the resulting feature maps are a combination of the previous layer's feature maps, more complex features emerge deeper in the network. The basic geometric shapes detected through convolutions by the earlier layers (*e.g.* edges, curves) are combined to form more complex shapes (*e.g.* outline of an eye) in the deeper layers, thereby learning features of a high abstraction level.

The core observation is that images are stationary – features learned at one part of an image also apply to other parts of the image. This allows for *sparse connections*, where not all input-output pairs have a connection, as the kernel size necessary to detect meaningful features (*e.g.* edges) is only a fraction of the input image size. The learned set of weights for a kernel can be *shared*, which contributes to relieve the problem associated with fully-connected layers of a rapidly increasing number of weights as it reduces the number of weights and computations necessary.

In addition to the *filter shape* that sets the number of weights to be learned and type of *padding* to determine how the edges should be handled, a *stride* needs to be set, which is the step size of the convolution operation. In the earlier examples, a step size of one was assumed – the convolution was applied to each position in the image. However, a larger stride can be set to reduce dimensionality. For example, on a 128×128 image, a convolution with stride 2 (and same padding), will result in an 64×64 image, as at after

each convolution, the next position to apply the convolution at is two steps away rather than one, essentially skipping a pixel.

Pooling

However, setting convolutions to a larger stride are not the only way to downsample the size of the representation, thereby reducing the number of parameters for the network to learn. An alternative is *pooling*. A spatial sliding window (*filter*) is defined along with the step size (*stride*). The filter slides over the input in steps defined by the stride, and outputs one value over that neighbourhood according to the type of pooling: max, average or sum. In practice, max pooling has shown to work best and the step size is typically equal to the size of the sliding window.

An example: using max pooling with a 2×2 filter on a 4×4 feature map with a stride of 2 will result in a 2×2 output, taking the max of the local window region exactly 4 times. Although it is possible to have a stride less than the window size, *e.g.* stride of 1 with filter 2×2 which would result in a 3×3 output, it is not common to do so. The filter size is usually 2×2 for images and $2 \times 2 \times 2$ for three-dimensional volumes.

It should be noted that it is becoming increasingly common to disregard pooling layers in favour of convolutions with a larger stride.[49]

Activation functions

Activation function exist to introduce non-linearity into the network, as a deep network of only linear combinations could be simplified to one single linear equation and for real-world problems linear solutions rarely suffice. Multiple non-linear functions can be used for this purpose, each with their own advantages and disadvantages for different purposes.

Firstly, a *sigmoid* function is a S-shaped monotonic differentiable function with a fixed range of $[0, 1]$, as displayed in Eq. 2.11. As sigmoid is generally outputs one value, it is

used for binary classification where a prediction is determined by defining an arbitrary threshold. *Softmax*, on the other hand, is similar to sigmoid but has the added benefit that the values sum to 1.0 and can therefore be used as probabilities for (binary or multi-class) classification problems. In Eq. 2.12, it is assumed that vector $\mathbf{x}$ is a vector of length c corresponding to the number of classes and the softmax function calculates a value for each element in that vector. Neither sigmoid nor softmax are used as an activation for the hidden units, but can and are often used as the activation for the last layer to return probabilities for each class (softmax) or determine prediction defined by an arbitrary threshold (sigmoid).

$$f(x) = \frac{1}{1 + \exp(-x)} \qquad \text{(Sigmoid)} \qquad (2.11)$$

$$f(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{c} \exp(x_j)} \qquad \text{(Softmax)} \qquad (2.12)$$

The reason neither sigmoid, softmax, nor other traditional activation functions such as tanh are used for the hidden units is the problem of *vanishing gradients*. This occurs when the activation function squashes the input to a relatively small output space, such that large changes in the input will result in small changes in the output and therefore small gradients, especially when multiple layers of activations are stacked and each input is mapped to a smaller region by the proceeding layer. A solution to this was the introduction of the *Rectified Linear Unit*[36] (ReLU for short), defined as in Eq. 2.13. ReLU is not only faster to compute than a sigmoid, but it also does not saturate the gradients.[40] Alternatives to ReLU are, among others, Leaky ReLU and CReLU. Leaky ReLU does not return zero but rather x multiplied by a small positive number close to zero in case that x is negative, such that negative number also have a non-zero gradient. CReLU (Concatenated ReLU[50]) combines the two ReLUs that select the positive and negative part of the activation respectively, thereby doubling the depth of the activations. However, in practice ReLU remains the most commonly used activation function for hidden units and softmax for the final layer to provide probabilities over classes.

$$f(x_i) = \begin{cases} 0 & \text{if } x_i < 0 \\ x_i & \text{otherwise} \end{cases} \qquad \text{(ReLU)} \tag{2.13}$$

Batch normalisation

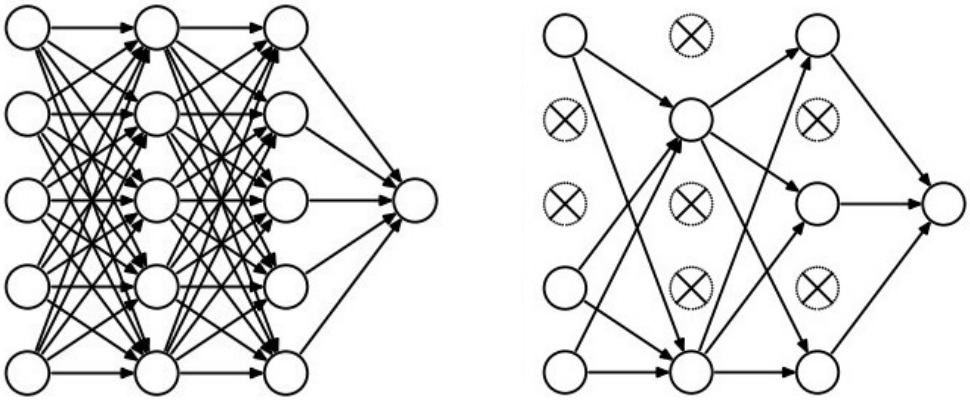
Batch normalisation is a layer typically inserted between the convolutional layer and the activation, that ensures the output of the (convolutional) layer takes on a Gaussian distribution with an initial zero mean and unit variance, that is adjusted throughout the learning process.[51] This is essentially a preprocessing step that can be built directly into the network itself as it is a differentiable operation, and therefore allows for backpropagation of the error. Batch normalisation decreases the difficulty of training the network and allows the network to be more robust against a bad initial choice of weights.

The batch is normalised as specified in Eq. 2.14. The mean and variance (untrainable parameters) of the batch are calculated, where the variance is multiplied with a small floating point number to prevent division by zero. These are then used, optionally with a trainable scale γ and offset β parameter to normalise the batch.

$$\begin{aligned} \mu &= \frac{1}{M}\sum_{i=1}^{M} x_i && \text{Mean} \\ \sigma &= \sqrt{\varepsilon \cdot \frac{1}{M}\sum_{i=1}^{M}(x_i - \mu)^2} && \text{Variance} \\ y_i &= \frac{\gamma(x_i - \mu)}{\sigma} + \beta && \text{Normalised} \end{aligned} \tag{2.14}$$

Dropout

Another layer that is occasionally inserted into the network, though it has no learnable parameters, is the dropout layer.[35] Dropout is a technique that prevents that the network becomes too sensitive to the weight of specific neurons. While training, a node may temporarily be deactivated to ensure the contribution of the node is removed. This ensures predictions do not become too dependent on particular neurons. Dropout is generally applied after the non-linearity, and it is of the utmost important to deactivate dropout when evaluating the model on the validation or test set.



(a) Fully-Connected Neural Network **(b)** Applying dropout with $p \simeq .5$

Figure 2.6: *Dropout*

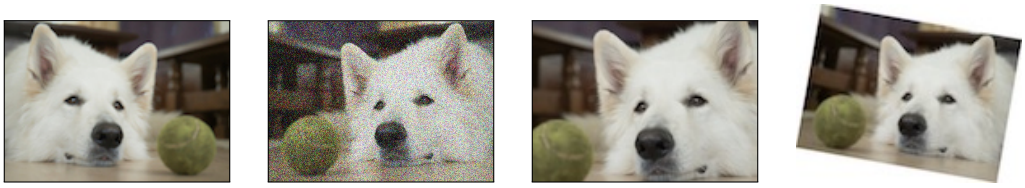
2.2.3 Regularisation

Batch normalisation and especially dropout are layers inserted into the neural network architecture to prevent that the weights of the model are tuned in such a way that the network can accurately reproduce the labels for data it was presented with during training, but cannot do the same for previously unseen new data. This is a central problem in machine learning – the problem of *generalisation*.

The first approach to ensure a trained model generalises well is to divide the available data in three parts; training, validation and test set. The training data is the data fed to the network during training and used to tune the weights and the test data is the dataset on which we report our eventual model performance. However, occasionally we may want to train various models – for instance, to experiment with different hyperparameter settings. If we were to choose the best model according to the performance on the test set, generalisation can still not be guaranteed, which is why model selection is based on the performance on the validation set.

Although this allows us to evaluate how well a model performs, it does not contribute to counter-act *overfitting*, which occurs when a statistical model captures the noise of the data, therefore performing well on the training data, but generalises poorly to unseen data. This is a common problem especially when the size of the training dataset is small compared to the number of model parameters that need to be learned – likely scenario in the medical domain, where the patterns to be learned are complex and there is little data available. Obtaining more data will typically boost performance, but is often a cumbersome or infeasible task. Instead, overfitting can be prevented with *regularisation* techniques, which Goodfellow described to be *"any modification that we make to the learning algorithm that is inteded to reduce the generalization error, but not its training error"*.[52] Batch normalisation and dropout are regularisation techniques that can be immediately inserted into the neural network as layers, but other techniques that have to do with the manner in which we present our data to the network during training

Firstly, when data is not plentiful and the training data alone may not be sufficient to create a model that generalises well, the number of data points for the algorithm to learn from can be artificially increased through *data augmentation*. Data augmentation is the process of altering existing data to create new data that is similar to the original. In terms of images, approaches may include adding noise or applying transformations such as flips or rotations. In practice, we consider the variations we wish our network to be robust against, such as scaling, noise, and rotations, and apply such variations at random during training.



(a) Horizontal flip **(b)** Noise **(c)** 1.1× zoom **(d)** 10^o rotation

Figure 2.7: *Image transformed with data augmentation.*

Secondly, with an iterative method such as gradient descent, the alterations of the weights after each iteration to better fit the training data may at some point come at the expense of performance on unseen data. *Early stopping* rules provide guidance to determine when training should be stopped to achieve better results with respect to generalisation.[53] For example, in *validation-based early stopping*, not only the performance of the model on the training data, but also on the validation data is tracked throughout the iterative training process. Though the error on the training data may continue to decline, a rise in validation error indicates the start of overfitting. The training is to be stopped at the point where the validation error is lowest, but the validation error may fluctuate during training due to local minima. Therefore, one should train for a set number of iterations and create a snapshot of the model parameter weights at point at which the validation error is lowest. These model weights are then to be used for prediction on the test set.

3

Medical Context

Early stage lung cancer manifests itself in the form of *pulmonary nodules* visible on a 2D X-ray or 3D computed tomography (CT) scan. Pulmonary nodules, also commonly referred to as *lung nodules* or simply *nodules* throughout this work, are described as a small, focal, rounded abnormality in the lung visible on a medical image, regardless of presumed histology, that is mostly surrounded by lung parenchyma – the portion of the lung involved in oxygen and carbon dioxide transfer, such as the alveoli, alveolar ducts and bronchioles.[54] While medical images of the chest may visually reveal abnormal cell growth in the lung, not all visible nodules are necessarily associated with cancer. Approximately 20% of nodules represent malignant growths, and other common causes include infections and inflammations (*e.g.* tuberculosis).[55]

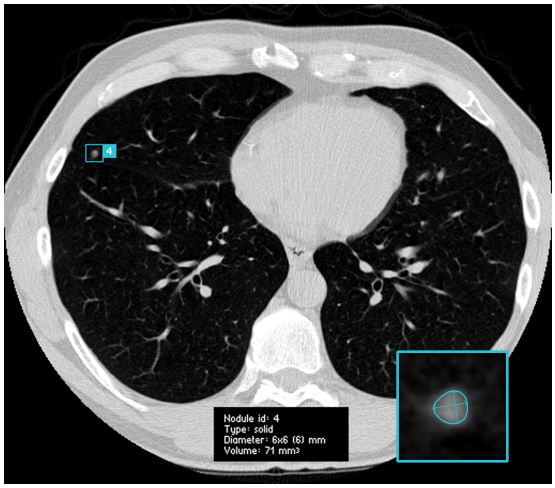
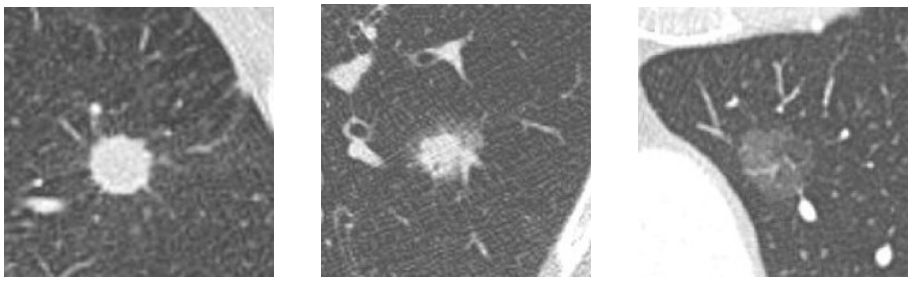


Figure 3.1: *Example of a nodule on a CT thorax slice.*
Source: NLST / Aidence Veye

Pulmonary nodules are characterised to be between 3mm and 30mm in largest axial diameter. Lesions larger than 30mm are *masses* and considered malignant until proven otherwise, while micro-nodules (lesions $<$ 3mm) are considered benign and typically require no follow-up.[56] Detection of nodules, regardless of size, on a three-dimensional CT scan is a complicated, labor-intensive task for the radiologist due to the rich vascular structure in the lung. Though visual examples of pulmonary nodules provided in this thesis are two-dimensional images, it is nearly always impossible to reliably determine whether or not the highlighted structure is a nodule or not based on a single image without three-dimensional context. Additionally, there is a large variation in how nodules visually appear on a CT-scan. *Calcified* nodules, for example, will appear very bright in comparison to non-calcified nodule, though the pattern of calcification (*e.g.* popcorn, diffuse, laminated, etc.) can differ. The radiographic edge characteristic of a nodule may differ as well; some may be *spiculated*, indicating they have linear strands extending from the nodule margin, while others can appear more "bubbly" as a confluent collection of nodules, referred to as *lobulated*. Additionally, nodules can have different degrees of *sphericity*, and their margins may either be irregular or smooth and well-defined.[56]

A commonly distinguished (and reported, due to the differing follow-up management

after detection) differentiating visual feature of a nodule is its *composition*. The nodule can have a solid component – an area within the nodule which is relatively homogeneous and high in pixel value on an image, thereby obscuring the underlying lung tissue. It may also contain a ground-glass component, which – much like the solid component – typically has an increased pixel value in the nodule area compared to the surrounding area, but the underlying (broncho)vascular structure of the lung is still distinguishable. These two components are used to categorise pulmonary nodules as either *solid* (see figure 3.2a) or *sub-solid* nodules. Sub-solid nodules may further be classified as either *part-solid* (containing both a solid and a ground-glass component, see figure 3.2b) and *ground-glass lesion*, that does not obscure the vascular pattern in any way (see figure 3.2c). Solid nodules occur far more often than part-solid nodules and ground-glass, though part-solid nodules typically have a higher malignancy rate – hence the different follow-up management.[57]



(a) Solid **(b)** Part-solid **(c)** Ground-glass

Figure 3.2: *Examples of nodule composition types*
Source: Lederlin *et al.*[58]

The rate of malignancy per nodule composition type is a relevant as various factors can give an indication of the likelihood that a nodule is cancerous, some of which are dependent on the nodules characteristics (such as composition), while others are not.[59] The Brock model, developed by participants of the Pan-Canadian Early Detection of Lung Cancer Study, is an example of a multivariate model that provides an estimation of the risk that a pulmonary nodule spotted on a CT scan is malignant[60], and is recommended for use by the British Thoracic Society as a guideline for determining

appropriate follow-up.[61] Some of the variables for the Brock model's estimation of pulmonary nodule malignancy are directly inferable from the CT scan itself (size[62], location, type, total nodule count on the CT scan, spiculation), while other variables are not. These other variables include family history of (lung) cancer and patient age and sex, which is information that is generally not available for the CAD system developers due to patient privacy concerns. As such, the added value of CAD systems in lung cancer screening would be accurate lung nodule detection – possibly providing values for the inferable variables of interest for malignancy – and serving as a second reader to the radiologist to increase the detection rate, rather than direct diagnosis of lung cancer itself.

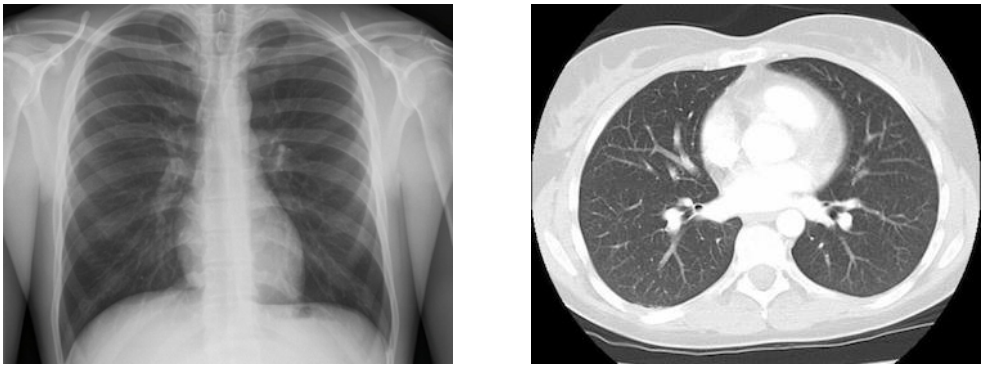
In this introduction, the characteristics of pulmonary nodules in terms of their appearance on medical images and factors that give an indication of their malignancy have been discussed. This should help provide the reader an understanding of the complexities associated with the visual detection of nodules on medical images, as well as why the aim is to assist the radiologist in *detection* rather than *diagnosis*. In the following sections, image acquisition for nodule detection in terms of scanning protocol and data storage will be discussed, as well as details with regards to (semi)publicly available datasets that can be used for developing nodule CAD software and previously done work on automatic nodule detection. A more in-depth account of lung anatomy and lung cancer development is provided in the appendix IV.

3.1 Image acquisition

For the detection of pulmonary nodules, both by a radiologist or CAD software, images of the lung structure are a necessity. Although pulmonary nodules can be visible on an X-ray image, the National Lung Screening Trial compared standard chest X-ray with low-dose helical computed tomography for lung cancer detection and found that nodules were detected more frequently at the earliest stage by low-dose CT.[63] Both methods are

non-invasive, but whereas X-ray produces a single image (see figure 3.3a) in which various structures overlap, CT produces a 3D scan of the thorax, visualising bones, muscles, fat organs, and blood vessels. A chest CT scan is done in one breath-hold and typically visualises the area from the neck to the diaphragm.

CT is one of the most used imaging procedures in radiological practice, during which a large series of 2D X-ray projection images are taken from different directions. Using computer processing, an image – also called *slice* – can be reconstructed from these projection images. In this way, many continuous slices can be obtained, which can be stacked together to form a 3D image of the lung (see figure 3.3b for a single slice). CT scans may be taken with or without contrast. With contrast, a substance is taken by the patient that allows a particular organ or tissue to be seen more clearly on the scan.



(a) X-ray image of chest **(b)** Axial slice of CT chest

Figure 3.3: *X-ray chest and CT chest images*
Source: RadiologyInfo.org

The scanned area of the human body is rarely viewed by the radiologist in software package that allows the viewer to analyse the object of imaging in a three-dimensional manner. Rather, two-dimensional slices are stacked together consecutively in a PACS (*picture archiving and communications system*) viewer, allowing the radiologist to scroll through a series of images. The orientation of these individual images can be from any one of the three orthogonal anatomical planes of the human body: axial, coronal or sagittal. The axial plane is parallel to the ground and is analogous to a view from the top,

the coronal plane divides from front to back and supplies a view from the front, and a sagittal scan provides a view from the left or right side of the body. Figure 3.4 demonstrates a chest CT slice from the various anatomical orientations.

(a) Axial chest CT **(b)** Coronal chest CT **(c)** Sagittal chest CT

Figure 3.4: *Lung CT from the various anatomical planes.*
Source: Radiopedia

For lung nodule detection, exclusively axial scans are considered. In such a CT image, the pixel value represents the mean attenuation of $x \times y \times z$ millimeter of lung tissue, measured in Hounsfield units, which can be windowed to highlight particular structures. The distance between the real world center points of these volumes represented per pixel in the row and column direction[1] in an image is called the *pixel spacing* and has a similar value for both the $x-$ and $y-$direction. The distance in millimeters between the center points of volumes in two consecutive slices is the *slice thickness* and is generally speaking much larger than the pixel spacing. An example of a typical axial lung scan is one with a pixel spacing of ~ 0.5 and slice thickness of ~ 2.5, though these values will often vary.

A CT image and its accompanying metadata, such as pixel spacing and slice thickness, are stored in the DICOM (*digital imaging and communications in medicine*) format. The hierarchical DICOM data model considers four different *information entities*: patient, study, series and instance. A complete examination for a patient (at a certain timestamp) is called a *study*. When a patient undergoes an examination, it is possible that multiple tests are performed, e.g. with different kernels.[2] A single such test is called a *serie*, which

[1] x, y and z will refer to position in the sagittal, coronal and axial plane respectively throughout this work.
[2] The reconstruction kernel, usually set by the vendor, is an algorithm that primarily affects image quality

comprises the entire 3D volume of the lung in case of a CT chest. All series combined form the study. The DICOM format, however, is instance-based and stores each *instance* (synonymous with *slice*) in a series as a `.dcm` file containing both the pixel data associated with the image and metadata with data on all the related information entities, such as unique identifiers for the instance, series, study and patient, patient information, and image acquisition information. This metadata is required to properly reconstruct the stored image data to a recognisable format. All spatial information, such as the position of the patient within the scanner, the relative position of the image plane, and the pixel spacing and slice thickness, are recorded in world millimeters.

3.2 Public datasets

The key element of success in any machine learning task – including those for the development of CAD software – is the quantity and quality of the data at hand. For the quantity of data, we are entirely dependent on openly available datasets of thorax CT images, obtained as described in the previous section, that need to be accompanied by annotations of pulmonary nodules locations in order to provide guidance to our network as to what to learn. The two largest available (semi-)public datasets for lung CTs are the NLST set and the LIDC/IDRI set, which will be expanded upon below to give an indication of dataset quality.

3.2.1 NLST

The NLST dataset is an collection of CT chest images originating from the National Lung Screening Trial, conducted by the American College of Radiology between 2002 and 2010.[13] The NLST was a randomised controlled trial to determine whether screening

and is essentially a trade-off between noise and spatial resolution.

for lung cancer with low-dose helical computer tomography without contrast reduces the mortality from lung cancer in high-risk individuals, relative to screening with chest radiography (X-ray). High-risk group inclusion criteria referred to current smokers (or former smoker who quit smoking within the past 15 years) aged 55 to 74, with 30 or more pack-years[3] of cigarette smoking history, who had not received a chest CT examination 18 months prior to eligibility assessment. 53, 454 participants were enrolled in the trial between August 2002 and April 2004, of which approximately half were randomly assigned to the CT arm of the NLST. The CT arm protocol was for three annual helical CT exams to screen for lung cancer at timestamps *T0, T1* and *T2*. For each screening exam, the image collection contains a localizer image and two to three axial reconstructions of a single helical CT scan of the chest. Approximately 200,000 image series from 75,000 CT exams in 25,000 people are available. However, CT image release is limited to approximately 15,000 scanned participants per project.

For this project, through the courtesy of Aidence, we have anonymised records from 14, 588 individual patients available from the NLST dataset, which each contain a study for each annual timestamp.[4] Adding some constraints, such as a maximum slice thickness of 2.5mm, this provides us with a dataset of 41.383 individual studies. For each study, only the most appropriate series to be used for lung nodule detection is taken into consideration, ensuring a nodule from the same timestamp can never be in the dataset twice, though it can appear multiple times at different timestamps. Of series taken from the 41.383 studies, 14.783 contain one or more nodules and 26.600 do not. The center coordinates of the nodules were initially not provided, only the unique identifier of the instance it was visible on and the lung (left or right). The center coordinates of the nodules were initially not provided, only the unique identifier of the associated instance. Through an annotation process with a team of radiologists lead by Aidence, the coordinates were obtained.

[3] pack years = packs per day * years smoked

[4] A patient identifier is included such that the records from different timestamps can registered as belonging to the same patient.

3.2.2 LIDC/IDRI

The Lung Image Database Consortium was specifically founded to stimulate research in the area of medical imaging for lung by creating an database of CT images, obtained through the contributions of seven academic centers and eight medical imaging companies, in collaboration with the Image Database Resource Initiative (IDRI).[64] Due to the large number of and variety in contributors, the dataset is relatively varied and contains both low-dose and full-dose CTs, taken with or without contrast, and the data was acquired with a wide range of scanner models and acquisition parameters.

The images in the database were combined with a information regarding the content obtained through a two-phase data collection process involving multiple expert thoracic radiologists to combat inter-observer variability. In the initial *blinded* phase, each of the four expert radiologists were asked to review 1018 CT scans and mark detected suspect lesions as *nodule* $\geq$ *3mm*, *nodule* $<$ *3mm* or *non-nodule* $\geq$ *3mm*[(5)], and provide a boundary of the nodule in all dimensions if marked as nodule larger than 3mm. In the second, *unblinded* phase, each individual radiologists was presented with their own marks alongside the anonymized marks of the three others to form a final opinion.[65]

Each of the 1018 cases of the dataset includes the images and is accompanied by an XML file that records outlines and nodule characteristic ratings for each of the 2669 lesions that were marked to be nodules of at least 3mm by at least one out of four radiologists. The marked nodule boundaries were used to calculate characteristics such as volume and maximal diameter, and the radiologists were each requested to subjectively provide an assessment of subtlety, internal structure, spiculation, lobulation, sphericity, texture, margin and (subjective) likelihood of malignancy.[65]

[(5)] This includes masses, but also other clear non-nodule lesions *e.g.* scars, post-surgical changes, etc.

3.3 Related work: nodule detection systems

Considering the advancements in technology since the earliest rule-based CAD systems that used low-level pixel processing, systems for medical image analysis have become an increasingly popular field of research. Convolutional neural networks for nodule detection appeared as early as 1995, where patches suspect of containing nodules were extracted by Lo *et al.* using template-matching, and then classified with a simple two-layer CNN with twelve 5×5 filters.[66] A year later, an improved version of this architecture was used to assess automatically extracted suspect lesions to balance a high number of true positives (desired, relevant information for the radiologist) with a low number of false positives (distracting information).[67]

The idea of Lo *et al.* to separate the detection of lung nodules in a *localisation* step to detect and extract potential suspect lesions and a *false positive reduction* step to remove distracting false positives seems to have become standard procedure – practically all scientific and commercial research into nodule detection since then have adopted this manner of detection. Both Firmino *et al.*[68] and Al Mohammad *et al.*[69] described a generic architecture in their respective reviews of computer-aided dectection systems for pulmonary nodule detection that consists of the following five subsystems:

1. **Data acquisition**: obtaining the medical images, *e.g.* from existing public databases for training and testing purposes or retrieving from private databases from partner hospitals for validation of the software and assistance to the radiologists.

2. **Preprocessing**: techniques applied to improve the quality of the data, *e.g.* windowing to highlight particular structures, normalisation and removal of defects caused by the image acquisition process.

3. **Segmentation**: applying a lung mask or segmentation to separate the lung tissue from other organs and tissues in the scan (to prevent obvious false positives). Many approaches to segmentation exist, varying from simple thresholding

operations or deformable models to neural networks trained to differentiate between lung and non-lung tissue.

4. **Candidate generation**: localisation of abnormal tissue in the lung and marking their locations as potential nodule candidates. This step is also known as *localisation*.

5. **False positive reduction**: binary classification on the generated candidates as either 'nodule' or 'non-nodule', or assigning a probability for each nodule that can be thresholded at an arbitrary point (*e.g.* desired sensitivity/false positive rate trade-off). Also known as *boosting* or *FP-reduction*.

Improving performance of CAD software comes down to improving the results of these individual steps. LUNA16 is a since 2016 running open lung nodule detection challenge using the LIDC-IDRI dataset that attracted both academic and commercial participants. For their running open challenge on nodule detection, they created two separate tracks: a complete *nodule detection* track for CAD systems in their entirety and a specific *false positive reduction* track where a provided set of nodules needs to be assigned a probability regarding the likelihood of the candidate being a nodule. The overview paper published on the results of the LUNA16 challenge so far revealed that the leading solutions in the detection track all separated the challenge into these individual steps and used convolutional neural networks for the false positive reduction.

The LUNA16 challenge was created to provide a reliable comparison of CAD algorithms in order to encourage development of new technology. Although various other approaches to lung CAD systems have been published, both from academic and commercial sources, it is uncertain how these compare as large scale evaluation studies into state-of-the-art lung CAD systems are scarce, development in the field is rapid, and no widely used comparative dataset with an associated performance metric exists other than those proposed by LUNA16 and later the Kaggle Data Science Bowl 2017. Therefore, this section highlights the results and approaches of the highest scoring participants in open challenges related to nodule detection (specifically LUNA16 and the Kaggle Data Science Bowl 2017) as all participants in these challenges were scored

in a similar manner and their approaches have proven to be effective compared to other methods.

In the LUNA16, the current highest ranking solutions for both tracks are by the Fonova team led by Zhouran Lyu.[70] To accommodate the different pixel spacings and slice thicknesses of various scans in the dataset, it is common practice to interpolate the volumes in such a way that each voxel represents $1 \times 1 \times 1$mm of lung tissue. However, Fonova's preprocessing step consists of interpolating the available images to $1\times, 5556 \times .5556$mm of lung tissue. Candidate generation was done with a 3D UNet-like network that takes cropped $128 \times 128 \times 128$ cubes and produces a vector representing the coordinates, radius and probability of a candidate nodule. Non-maximum suppression is used to combine overlapping candidates to calculate the final probability. False positive reduction is used to reduce the generated candidates to the most likely candidates by ensembling probabilities from three different models that are all variations of 3D convolutional neural networks.

Fonova's approach to candidate generation is described to be motivated by the first place solution in the Kaggle Data Science Bowl 2017. Since 2014, Kaggle, in collaboration with Booz Allen, organises the annual Data Science Bowl to tackle the world's challenges with data and technology. The goal of the 2017 challenge was to create algorithms that can improve lung cancer screening technology by determining whether lesions in the lung are cancerous, to ultimately determine a probability to indicate whether the patient is at risk of developing or has developed lung cancer. The winning model by Liao *et al.*[71] separates the problem into two modules: a 3D region proposal network with a modified U-Net for nodule detection which presents all suspicious nodules for a patient, and a module that determines the malignancy of the top five nodules based on detection confidence and combines these findings to produce a likelihood of lung cancer for the patient. Though this team achieved the highest score in the competition, they note a significant difficulty in this challenge was that of overfitting – the number of parameters of a 3D CNN is relatively large compared to the available training data, and a suggested straight-forward manner of improving to results was to increase the number of training samples, once again demonstrating the need for data efficient learning in the medical

domain.

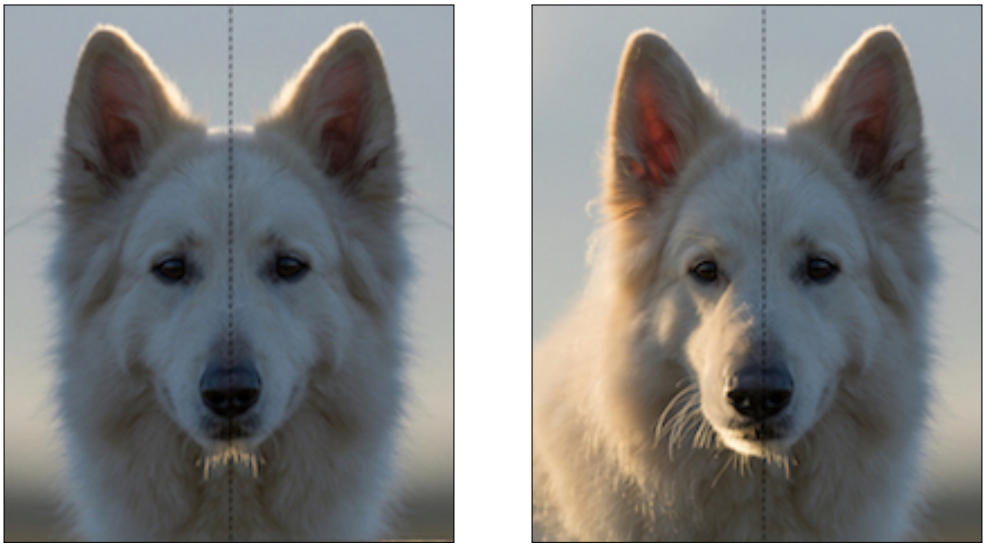
Aidence, the organisation that provided the data as well as supervision and guidance for this thesis, ranked 3rd in the Kaggle Data Science Bowl 2017 challenge out of nearly 2,000 participating teams. Therefore, in addition to best practices from literature on lung nodule detection and false positive reduction, their advice is taken into consideration in setting up and evaluating the experiments.

4

Group Equivariant Convolutional Neural Networks

Examples of symmetry can be seen anywhere in nature, buildings, art and even people and other biological organisms. Consider the dog in Figure 4.1: like many animals, the frontal view of its face exhibits *reflectional* symmetry as a line can be drawn through figure 4.1a to divide it in such a way that the two halves are exact mirror images of each other, despite how uncanny that may seem.

Symmetry, in this case, refers to exact correspondence between two objects after a transformation. Figure 4.1b, however, illustrates that not all properties of an image need to remain the same for it to still exhibit some form of perceptible symmetry – the lighting



(a) Mirror image **(b)** Perceptible symmetry

Figure 4.1: *Reflectional symmetry in the face of a dog*

is different in both halves but the general shape and composition elements such as ears, nose, eyes and whiskers remain the same. We say an object exhibits *symmetry* if a transformation can be applied to it that may change some of its properties, but leaves others in tact.[72]

In machine learning in particular, we are interested in the symmetry of the *labels* that we try to predict – specifically, we wish that label remains the same even when other properties, such as the way that pixels are arranged to form the image, may differ. Consider the example in Figure 4.2, which illustrates that the rotation of the image by various multiples of 90^o does not intrinsically change the image content – whatever the orientation, it remains a picture of a dog. It is said that label is *invariant* under rotation, and the label "dog" that describes the content of the image exhibits a *rotational symmetry*.

For most people, the concept that some transformations such as rotations and reflections may alter the appearance of an object but do not modify the content is a very intuitive notion. Although figure 4.2c is not a dog in an orientation that one would often

(a) Original **(b)** 90^o rotation **(c)** 180^o rotation **(d)** 270^o rotation

Figure 4.2: *Images that all have the label "dog"*

encounter, people are generally still capable of recognising object in the picture for what it is. Neural networks, however, have no such intrinsic notion about which alterations of the data would change the label and which ones would not. Data augmentation is generally the method of choice used to combat this problem,[73] where the transformations we wish the network to be robust against are applied to the input images to provide the network with examples of transformed images it may encounter, but the disadvantage of this approach is that the model will only be (approximately) invariant with respect to the training data and expected transformations and the solution of creating artificial data is not very elegant. Group Equivariant Neural Networks, however, aim to build the invariance and equivariance into the network architecture itself.

To get a full understanding of what this entails, we will first present the reader with a few basic concepts from (symmetry) group theory – the mathematical study of symmetries – before we move on the related work on the topic of encapsulating invariance and equivariance in deep learning. Lastly, we will provide the details of the implementation of 3D group convolutional neural networks.

To get a full understanding of what this entails, we will first familiarise the reader with *invariance* and *equivariance* and how these concepts contribute to the effectiveness of convolutional neural networks in the domain of image processing. Next, we will discuss various approaches that have been investigated to obtain rotation invariance and equivariance in neural networks, before we move on to a theoretical background on group theory – the mathematical study of symmetries – which is necessary to fully understand the theoretical and implementation details of group-convolutional neural networks.

4.1 Invariance & equivariance

Invariance in a mathematical sense is defined as that some property of an object remains the same, while alterations to it have been made. For machine learning problems, *invariance* with respect to some transformation indicates that model produces the same response, regardless of whether the transformation has been applied to the input. Convolutional neural networks, commonly used for analysing images, are approximately *translation invariant* (see Eq. 4.1). A shift of the origin of an image generally does not influence the ability of a trained model to correctly identify its contents.

Whereas a convolutional neural network as a whole is invariant to translation, the individual convolutional layers in the network are *translation equivariant*. In a convolution layer, the same kernel is applied at every position of the input, so the value of a weight is not tied to one single input. Instead of learning a separate weight for every position like in a fully-connected layer, only a single set of weights is learned and these weights are *shared* across inputs.[52] The result of this is that a shift in location simply results the same output (due to the same weights being applied), but shifted. This notion, that a transformation (*e.g.* translation) applied to the input features results in an equivalent transformation of the outputs, is known as *equivariance* and characterised by Eq. 4.2 where function f calculates the output of a hidden layer to which $\mathbf{x}$ is the input and T_g is the operator formed by transforming the input with transformation g. It should be noted that although invariance is commonly used to refer to both invariance and equivariance, these are not interchangeable concepts.[1]

$$f(\mathbf{x}) = f(T_g(\mathbf{x})) \qquad \text{(Invariance)} \tag{4.1}$$

$$f(T_g(\mathbf{x})) = T_g(f(\mathbf{x})) \qquad \text{(Equivariance)} \tag{4.2}$$

[1] Consider input pattern $\mathbf{x} = [0, 1, 2, 0]$ with some function f such that $f(\mathbf{x}) = [0, 1, 0, 0]$. As $\mathbf{x}$ is translated to $\mathbf{x}' = [0, 0, 1, 2]$, invariance means $f(\mathbf{x}') = f(\mathbf{x}) = [0, 1, 0, 0]$ whereas equivariance indicates the result shifts accordingly: $f(\mathbf{x}') = [0, 0, 1, 0]$.

Translation invariance of the network contributes to the effectiveness of CNNs in the domain of image analysis as images exhibit translational symmetry, and is a direct result of the translation equivariance of the convolutional layers combined with dimensionality reduction that removes spatial information. However, perception tasks exhibit various other types of symmetry which CNNs are not naturally invariant to, in particular rotation. The intuition is that, as translational equivariance of the convolution operations make CNNs particularly suitable for image analysis, rotational equivariance of the layers would add to that.

4.2 Related work: rotational equivariance

The intuitive notion that rotational equivariant and invariant properties may be desirable in neural networks was supported by empirical investigative work by Lenc & Vedaldi (2015)[74]. Their aim was to add to the understanding of image representations by measuring their equivariance, and they reported that shallow representations such as HOG and deep representations such as the AlexNet CNN (2012)[40] trained on ImageNet learned representations that are equivariant to horizontal and vertical flips, rescaling by half, and rotation of 90^o, particularly in the earlier layers.

An early approach by Fasel & Gatica-Perez (2006)[75], even before the common adoption of deep learning techniques, involved a neoperceptron for object recognition that achieved rotation invariance by stacking of virtual rotated images (generated from the input image) and introducing blurring layers that reduced the number of virtual image orientations presented to the network. A similar idea that also involved stacking rotated copies of the images was that of Dieleman, Willett and Dambre (2015)[76] who exploited rotational and reflectional symmetry for the purpose of participating in the Galaxy Challenge, an international competition for morphology classification on images from the Galaxy Zoo project. The transformed images (rotated and flipped; called

viewpoints) were processed separately. The output feature maps for these viewpoints were then concatenated and fed to one or more fully-connected layers. Although this does not result in invariance, this approach did allow for better weight-sharing across orientations.

These approaches were limited to applying transformations to input images. However, Gens & Domingos (2014)[77] chose to approximate invariance to the affine symmetry group and allow their deep symmetry networks to generalise to higher dimensional symmetry spaces by evaluating the feature maps at N control points and using kernel interpolation, which proved to reduce sample complexity on the NORB and MNIST-rot datasets. An extension of this network to include symmetry groups in 3D space was suggested as a direction for future work.

Other approaches seem tied to the idea of rotating either the filters or the resulting feature maps. Marcos, Volpi & Tuia (2016)[78], for instance, suggested tying the weights of groups of filters to several rotated versions of the original filter in the first layer of a shallow CNN to encode invariance into the network, and proved its effectiveness on the problem of texture classification. Later that year, Marcos, Volpi and Tuia together with Komadakis (2016)[79] developed a vector field network that encodes rotation equivariance and invariance by applying each filter at multiple orientations resulting in a vector field feature map such that the magnitude and angle of the highest scoring orientation can be used for their modified convolution operator. Interpolation was applied to deal with rotation of various degrees. Particularly interesting is that this approach was tested and proven effective on the biomedical domain, for the purpose of segmenting neuronal structures, though this test only considered two-dimensional images.

A common idea is to introduce an implementation that is easily inserted into existing network architectures. Dieleman, De Fauw & Kavukcuoglu (2016)[80])introduced four operations (*slice*, *pool*, *stack* and *roll*) for this purpose that could be inserted into an existing netwerk to build partially rotation equivariant neural networks for the rotations of 90^o in particular. However, this could easily be extended to include reflections by changing the slicing operation to include reflections and other operations accordingly. Their approach was based on rotated feature maps, although they note rotating filters

would be equivalent, primarily due to the ease of implementation as the operation necessary could be inserted as a separate layer in most deep learning software frameworks. The framework they constructed was suggested to be especially interesting for domains where data scarcity is an issue, such as medical imaging, though it had not yet been applied to this domain. Instead, results were verified on a new version of the Galaxy Zoo project, as well as the Kaggle plankton Data Science Bowl dataset and the Massachusetts buildings dataset.

Advantages Dieleman *et al.* describe to prefer rotating the feature maps over rotating the filters mainly includes ease of implementation, but they do note that this is at the cost of increased memory requirements. Applying the rotation transformation on the filters rather than the feature maps they operate on would reduce memory requirements as the filters instead of the feature maps would need to be copied, but the filters are generally smaller (*e.g.* 3×3 or 5×5) than the feature maps. An alternative approach that included rotating filters were the Oriented Response Networks by Zhou *et al.* (2017) [81], that use Active Rotating Filters (ARFs) that act as a virtual filter bank with the filter and its rotated versions, that both outperform baseline CNNs while using significantly fewer network parameters on various datasets (MNIST, a subset of MNIST-rot, CIFAR-10, CIFAR-100).

Zhou *et al.* noted that the theoretical foundation of their work on ARFs was provided by the work of Cohen & Welling (2016) [27] on Group-Equivariant Convolutional Neural Networks (G-CNNs), as they justified that rotation of images could be reflected in both the feature maps and filters. Similarly, Dieleman *et al.* indicated that the work by Cohen & Welling on G-CNNs described a fairly similar type of model, in addition to a theoretically grounded formalism for exploiting symmetries in CNNs. Whereas Dieleman *et al.* are more focused on efficient implementation for the specific case of 90^o rotations, Cohen & Welling provide a more general framework based on *symmetry groups* that can be easily extended.

Cohen & Welling's work on G-CNNs introduce *group convolution* layers for discrete groups that can be added as a drop-in replacement for spatial convolutions in

convolutional neural network architectures, with negligible computational overhead. Their G-CNNs proved to be effective in achieving state-of-the-art results on both the CIFAR-10 and MNIST-rot datasets and are based on a filter transformation through relatively simple filter indexing. As Cohen & Welling's framework is provides the best theoretical mathematical foundation and is easily extendable to include convolutions for other groups, this work will focus on extending this framework to three dimensions such that it is applicable to volumetric image data. As group convolutional layers are implemented for discrete symmetry groups (generated by translations, reflections and rotation), the following section will first describe some fundamental background to group theory, including a detailed description of the groups implemented by Cohen & Welling as well as the to-be-implemented 3D groups for this thesis, before moving on to the core of this work.

4.3 Group theory

A *group* is defined as a set G together with the group law of G (operation $\cdot$, also referred to as multiplication), that satisfies the following axioms:

- **Closure**: For all elements $a, b \in G$, the result of the operation $a \cdot b$ is also an element in G
- **Associativity**: For all elements $a, b, c \in G$, it is true that $(a \cdot b) \cdot c = a \cdot (b \cdot c)$
- **Identity element**: For every element $a \in G$ there is an element $e \in G$ for which $e \cdot a = a \cdot e = a$ holds. [2]
- **Inverse element**: For each $a \in G$ an element commonly denoted as a^{-1} exists for which $a \cdot a^{-1} = a^{-1} \cdot a = e$ holds. [3]

[2] It should be noted that the identity element of a group is unique. There cannot be more than one identity element, such as e and e', as then both $e \cdot e' = e$ and $e \cdot e' = e'$ should hold and therefore $e = e'$.

[3] Additionally, the inverse of each element of a group is unique as multiplication is associative. Imagine

An simple example of a small group is *Vierergruppe*, named by Felix Klein in 1884 and symbolised by the letter V, which contains the four elements $\{a, b, c, e\}$. The identity element is e; as one can read from the Klein group's Cayley table in table 4.1, the product of every element with e is the original element. Similarly, one can see that the inverse element for every element is itself. The table shows that all elements that are a result of the product between two elements in V are elements in V as well, proving closure, and although it is less easily observable from the table, one can deduce that associativity also applies to group V (*e.g.* $(a \cdot b) \cdot c = c \cdot c = e$ and $a \cdot (b \cdot c) = a \cdot a = e$ from which $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ follows).

·	**e**	**a**	**b**	**c**
e	e	a	b	c
a	a	e	c	b
b	b	c	e	a
c	c	b	a	e

Table 4.1: *Cayley table for Klein's Vierergruppe*

4.3.1 Symmetry groups

The *symmetry group* of an object is the group of all transformations, with composition as the group law, under which the object is invariant with regards to some property. That property in the case of symmetry in machine learning is the class label. The reason we concern ourselves with symmetry groups in this work is that symmetry groups are composed of the transformations we wish to define our network to be robust against.

Symmetry groups consist of combinations of *lossless* transformations. Reflection, a visual of which was provided in the introduction to this chapter, is an example of a lossless transformation, as no information is lost when an image is mirrored. This is opposed to *lossy* transformations, such as a rotation by 10 degrees for which interpolation is required,

both y and z are the inverse of x. Then $y = e \cdot y = (z \cdot x) \cdot y = z \cdot (x \cdot y) = z \cdot e = z$, proving both inverses y and z of x are the same element.

which – although the overall spirit of the image remains the same – does not only change the arrangement of the pixel values but may also influence the values themselves. There are three types of (lossless) transformations that can be used to generate all the elements in the symmetry groups we will focus on: reflection, rotation and translation.

- **Reflection**, also referred to as *flip* or *mirror*, is a mirror operation over a line of reflection. A point $(1, 1)$ in a standard coordinate grid, reflected in the x-axis, will result in point $(1, -1)$. and mirrored in the y-axis will result in point $(-1, 1)$. We denote this by M_L, where line L is the reflection axis in $\mathbb{Z}^d$.
- **Rotation** can be any rotation of an object about a fixed point between 0 and 360 degrees, though rotation is not lossless for all angles. We denote rotation by $R_{c,\theta}$ where c is the point in space that is the center of the rotation and θ the angle.
- **Translation**, denoted by T_V, is the operation of shifting a point to another point, *e.g.* an translation of point $(1, 1)$ with $(1, 0)$ will result in the point $(2, 1)$. The plane is shifted by applying displacement vector v, which is a vector in $\mathbb{Z}^d$.

Important to note is that all these transformations are what are known as *isometries*, as the distances between elements in the image are preserved, unlike with for example scaling – a linear transformation that may enlarge or shrink an object. A *point group* is a group of isometries that keep at least one point fixed. Furthermore, a group that consists of a transformation (*i.e.* reflection or rotation) and a translation is known as a *split group*. A split group in a two-dimensional Euclidean plane is a *wallpaper group* and in a three-dimensional space a *space group*.

4.3.2 Wallpaper groups

An illustrative example of a point group is C_4, the rotational symmetry of the square, consisting of multiples of 90^o clockwise rotations around the center point. C_4 is a group with a finite number of elements – four, to be exact, called the *order* of the group and usually denoted as $|C_4| = 4$. Every element in C_4 can be described in terms of a single

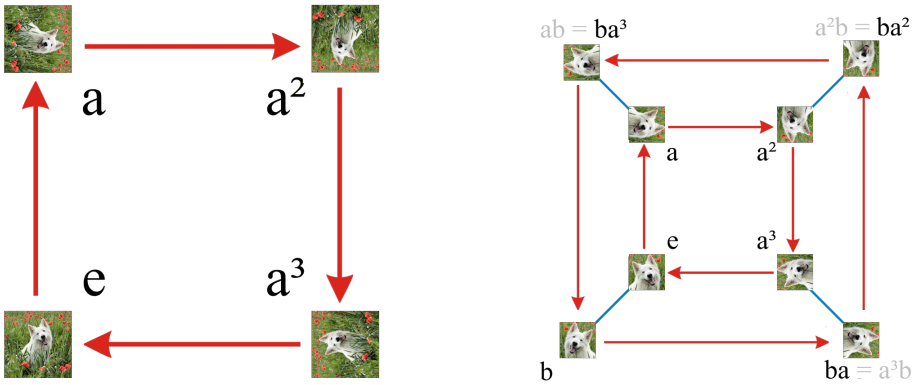
transformation. For instance, a 180^o around the center is simply two consecutive 90^o rotations. The set of transformations that can together generate all elements of the group are the *generators*. If a group can be generated with a single element, like C_4, we say a group is *cyclic*.

Examples of point groups with two generators are the *dihedral* groups. A dihedral group D_n is the rotational symmetry group of a regular polygon with n equal sides, that includes rotations and reflections. One such example is D_4, the square plate. Every element in D_4 can be generated with a combination of 90^o clockwise rotations about the center (also the generator for C_4) and a mirror operation (the second generator). These transformations, referred to as a and b respectively for the time being, are the generators of D_4 denoted as $< a, b >= D_4$. D_4 contains every element in C_4 and the reflection thereof, therefore $|D_4| = 8$.

Cyclic point group C_4 is a *subgroup* of D_4 as the elements in C_4 are a subset of D_4 which in itself forms a group under the group law of D_4. Moreover, cyclic point group C_n is a subgroup of dihedral point group D_n for any $n \in \mathbb{N}$. C_n is also *commutative*, or *abelian*, as $x \cdot y = y \cdot x$ for any two of its elements. D_n for $n \geq 3$, however, is not.

The abstract structure of a group can be encoded and visualised with a Cayley diagram, named after their inventor Arthur Cayley, which shows the group elements and the connections between them.[82] In figure 4.3, a Cayley diagram with a picture of the same dog as in the introduction of this chapter illustrates the effect various combinations of transformations a and b will have on the input image (lower left corner) for group C_4 and D_4. For example, it can be seen from figure 4.3b that a 90^o rotation followed by a reflection ($a \cdot b$; upper left image) does not lead to the same image as a reflection followed by a 90^o rotation ($b \cdot a$; lower right image), providing proof by counter-example that group D_4 is not abelian.

Point groups C_4 and D_4 are isometries of the Euclidean plane, and operate in a two-dimensional space. When combined with two linearly independent translations, in the $x-$ and $y-$axis, they are referred to as *wallpaper groups*. The wallpaper group with



(a) Point group C_4 **(b)** Point group D_4

Figure 4.3: *Cayley diagrams of points groups C_4 and D_4*

point group C_4 as a basis is $p4$, and can be parameterised in terms of (r, u, v) where $0 \geq r < 4$ to indicate the number of clockwise 90^o rotations and a translation $(u, v) \in \mathbb{Z}^2$ from the center point. Similarly, $p4m$ is the wallpaper group that takes D_4 as a point group basis and is parameterised in a similar way, with the addition of boolean parameter $m \in \{0, 1\}$ to indicate a reflection in the y-axis.[4]

4.3.3 Space groups

A symmetry group with translations in a two-dimensional space is a wallpaper group, but a slightly more complex configuration is the symmetry group with translations in the three-dimensional space: the *space group*. Translational symmetry in three-dimensional space is less straight forward than in a two-dimensional space and may for example include lattice centering[5], but for the purpose of this work we solely focus on the natural action of the three-dimensional point group on $\mathbb{Z}^3$. In our case, a space group will therefore consists of a translation $(u, v, w) \in \mathbb{Z}^3$ in combination with transformations like reflection and rotation.

[4] For practical reasons, we choose m for mirror to represent a reflection as the r is already is use.
[5] translational operations derived from the Bravais lattice type, of which there are 14

Cubic Symmetry

A square image, as described in the previous section and used in the Cayley diagram in figure 4.3 to illustrate the elements in the C_4 and D_4, has four distinct symmetry elements if you only consider rotations. Let's only consider rotations for the time being for the simplest three-dimensional equivalent of a square – the *cube*.

First of all, a difference is that we will not be able to rotate around a point, but are to rotate the cube around an *axis* instead. Say we define a line L in the three-dimensional space that runs through the center points of opposite faces of the cube (much like in figure 4.4a). The cube can rotate around this line in steps of 90^o exactly three times before coming back to the original cube position. This is an example of a 4-fold axis, as it provides us with four distinct elements (including the identity element) that exhibit symmetry. As a cube is composed of six faces in total, it has three opposing faces, indicating lines such as in 4.4a can be drawn in three different manners.

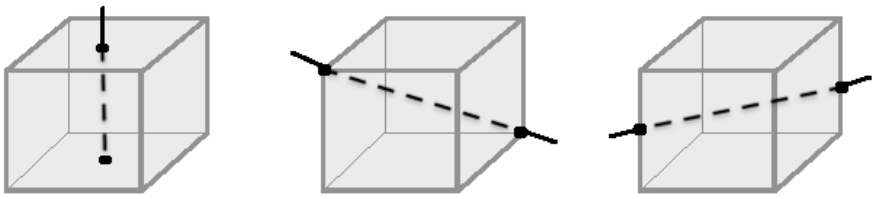
But these are not the only lines a cube can rotate around to achieve symmetry – figure 4.4b shows a line intersecting with the opposing vertices of the cube, which we can construct four of given that there are eight vertices in total. The cube can rotate around the line twice in steps of 120^o in a lossless manner while perserving symmetry. These are the four 3-fold axes of the cube.

Lastly, figure 4.4c has a line drawn through the middle of two opposing edges around which the cube can rotate a 180^o once before returning back to the identity element – a 2-fold axis. Given that there are twelve edges in total, a cube has six opposing edges.

Together, all these rotations around the three types of fold lines as illustrated in 4.4 in addition to the identity element make up the elements of the symmetry group of an octahedron or cube[(6)] – group O. These rotations add up to 24 elements for group O in total, in the following manner:

[(6)] An octahedron can be constructed from a cube by taking the center points of the faces of the cube as the vertices for the octahedron – therefore, the octahedron and cube share their set of symmetries.

- 1 identity element
- 9 elements from the three 4-fold axis of opposite faces (see 4.4a)
- 8 elements from the four 3-fold axis of opposite vertices (see 4.4b)
- 6 elements from the six 2-fold axis of opposite edges (see 4.4c)



(a) Opposite faces **(b)** Opposite vertices **(c)** Opposite edges

Figure 4.4: *Fold axis for symmetries of the cube*

Group O is the point group of symmetries of a cube when we only consider rotations. Much like the extension of C_4 to D_4, group O can be extended by including reflections. This group, called O_h, has the elements of O as a subgroup and the reflections of the elements of O in the center of the cube as additional elements, adding up to 48 elements in total. These 48 elements in total satisfy the group axioms of closure, associativity, identity and inverse. Neither O nor O_h is not abelian.

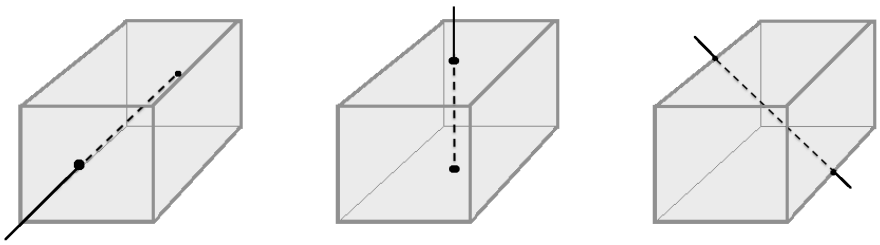
O and O_h are both point groups, as they leave at least one point fixed, and can form the basis for various space groups that include translation. For the purpose of distinguishing between point group G and the space group H with translations in the $\mathbb{Z}^3$ space of which G is a subset, we add suffix t *i.e.* $H = G_t$.

Rectangular Cuboid Symmetry

A cube is not the only object that exists in 3D, another such object is the *rectangular cuboid*, in which all angles are right angles and the opposite faces are equal. Two opposing

faces are squares, while the other four are rectangles. When only rotation is considered, the rectangular cuboid is characterised by symmetry group C_{4h} of order 8: the identity element, one 4-fold axis over the opposing square faces as seen in 4.5a, two 2-fold axis over the opposing rectangular faces as seen in 4.5b and two 2-fold axis over the two opposing long edges as pictured in 4.5c. By including reflections, C_{4h} can be extended to become group D_{4h} of order 16. Whereas is abelian, is not.

Group C_{4h} and D_{4h} are both subgroups of O_h and form the point groups at the basis of the space groups with translations in $\mathbb{Z}^3$ denoted as C_{4ht} and D_{4ht} respectively.



(a) Square faces **(b)** Rectangular faces **(c)** Long edges

Figure 4.5: *Fold axis for symmetries of the rectangular cuboid*

4.3.4 Translation groups

Wallpaper and space groups are extensions of point groups with translation in two- and three-dimensions respectively. However, translations in $\mathbf{Z}^2$ and $\mathbf{Z}^3$ are symmetry groups themselves (denoted as $Z2$ and $Z3$), under composition group law *addition* as $(k, l) + (p, q) = (k + p, l + q)$ for $\mathbf{Z}^2$ and similarly $(k, l, m) + (p, q, r) = (k + p, l + q, m + r)$ for $\mathbf{Z}^3$. The concept of translation as a group is a necessary addition for this work, as standard CNNs that are equivariant and invariant to translation, can be described as G-CNNs with group $Z2$ and $Z3$. How the groups described in this chapter so far can be used to achieve equivariance in convolutional neural networks, in addition to what equivariance is and why it is desirable, will be explained next.

4.4 Implementation

This subsection will elaborate on the proposed Group-Equivariant Convolutional Neural Networks where the convolutional layers are not only equivariant with respect to translation, but also towards rotations and reflections. First, an overview will be presented that describes on how feature maps may be modeled as functions on groups and how this can be used to create equivariance, after which the details with respect to the various group implementations for both two-dimensional and three-dimensional groups will be discussed, the latter of which is the contribution of this work.

4.4.1 Overview

In a traditional CNN with 2D convolutions, function $f : \mathbf{Z}^2 \to \mathbf{R}^K$ maps a pixel coordinate $(x, y) \in \mathbf{Z}^2$ to a K-dimensional vector with a values for each feature map, as K represents the number of channels.

Imagine now that a translation of (u, v) – transformation g – was applied to the feature maps and we wish to retrieve the value of the transformed feature map at point (x, y). To find the value, we can essentially apply the inverse of the transformation to find the unique point (x', y') that gets mapped to (x, y) with transformation g. This point can then be looked up in the original feature map. The inverse of a translation of point (x, y) with (u, v) is $(x - u, y - v)$.

$$\begin{aligned}[T_g f](x, y) &= (f \circ g^{-1})(x, y) \\ &= f(g^{-1}(x, y)) \\ &= f(x - u, y - v) \end{aligned} \tag{4.3}$$

Equation 4.3 applies for any $(x, y) \in \mathbf{Z}^2$, as every coordinate tuple is an element in the group we know as $Z2$. However, this definition similarly applies to any element in groups

other than $Z2$, such as C_4. Given h, an element in group G, we can redefine Eq. 4.3 as $[T_g f](h) = f(g^{-1}(h))$, where $g^{-1}h$ can be interpreted as composition. For C_4 in particular, the value of the feature map transformed with a 90^o rotation can be found by performing a lookup in the original feature map at the inverse of the 90^o rotation – the 270^o rotation, also an element in C_4.

This illustrates that feature maps, originally functions on the plane $\mathbf{Z}^2$ (images) or $\mathbf{Z}^3$ (volumes), can also be interpreted as functions on a group G. This notion forms the basis of the G-CNN, where a feature map consists of a number of adjacent channels according to the number the number of elements in a group that leave at least one point fixed (*e.g.* 4 and 8 respectively for $C_4/p4$ and $D_4/p4m$).

By mapping a set of sampling points to an element in a group, a transformation that maps from one element in the group to another can be applied to it. The resulting element can be mapped back to the set of sampling points, that can in turn be mapped to R. In other words, considering the set of sampling points I, group of transformations G, and real numbers R, we have the mapping $i2g$ that maps the indices from I and produces a group element $g \in G$, the inverse mapping $g2i$, and function v that maps from an index set to real numbers.

$$
\begin{aligned}
v &: I \longrightarrow R \\
i2g &: I \longrightarrow G \\
g2i &: G \longrightarrow I \\
T &: G \longrightarrow G
\end{aligned}
$$

Using these mappings, we can construct a transformed function w that maps I to R, as v implicitly defines v' on G in the form of $v'(g) = v(g2i(g))$ and we can construct w' as $w'(g) = v'(T(g))$. This transformed function w directly maps indices to number, such

that the indices correspond to group elements.

$$w : I \xrightarrow{i2g} G \xrightarrow{T} G \xrightarrow{g2i} I \xrightarrow{v} R$$
$$w : I \longrightarrow R$$

This can be used to transform a set of filters defined on a split plane group G (recall that a split group is a group consisting of a transformation and a translation). A g-convolution uses a transformed filterbank rather than the original filter, which can be retrieved using an array of indices (*gconv indices*). Considering the cost of the filter transformation is negligible, the computational cost of a g-convolution is equal to the computational cost of a regular convolution if the filter bank size for the regular convolution is equal to the augmented filter bank, which in practice may require some tweaking of the number of desired output channels.

The indices used to retrieve the augmented filter bank are specific to the group, and dependent on the group that was used to create the input feature maps (input group) as well as the desired group to use to create the output feature maps. As the input for the first layer of a g-convolutional neural network will be the original image or volume that has not yet undergone any convolutions with transformed filterbanks, the indices for the g-convolution in the first layer will be created based on group $Z2$ for images and $Z3$ for volumes. In the current implementation, the gconv indices constructed to create the transformed filterbank for the consecutive layers are based on the assumption that the all layers in the network will be transformed with the same group (*i.e.* input and output group are equal), although the framework could be extended to support alternative group combinations.

The transformed filterbank is created by reshaping the filter, applying the indexing operation and reordering the axes to collapse the result in a standard shape filter bank. This transformed filter is then used to convolve the input with a regular TensorFlow implementation of 2D or 3D convolution (or alternative deep learning frameworks) as

one would have done with the original filter. The transformed filter has shape $(k, k, \text{ni} * \text{nti}, \text{no} * \text{nto})$ for 2D convolutions and $(k, k, k, \text{ni} * \text{nti}, \text{no} * \text{nto})$ for 3D convolutions, with number of desired output channels (no), number of output transformations (nto), number of input channels (ni), number of input transformations (nti), and the kernel size (k). It should be noted that convolving with the transformed filter bank causes an increase in parameters compared to a convolution with the original filter. The number of parameters can be kept approximately fixed by dividing the number of filters by the square root of the number of stable elements in the group.

The current existing framework for g-convolutions can be extended to include new groups by defining a representation for the elements in the group and implementing the map from indices to the group elements and its inverse. Some small additional adjustments were necessary to construct the gconv indices, and specifically to enable usage of the extended filterbank for a 3D convolution in TensorFlow.

4.4.2 Implementation

To be able to use group-convolutions rather than regular convolutions in 3D, the framework had to be extended to be compatible with 3D convolutions. Additionally, the point groups and space groups associated with the transformations we wish our networks to be equivariant against need to be represented in order to define a manner in which to produce w for each group, which in turn is associated with the creation of the augmented filter bank.

The three fundamental elements to eventually perform a group convolution are `GArray` (for the group element representations), `GFuncArray` (for the mappings) and `GConv` (for transforming the filterbank and applying the convolution). These elements, and how the specific groups relate to these elements, will be discussed in the following sections – first in terms of the original implementation for the two-dimensional groups, after which the details with respect to the space groups will be discussed.

Group representation

The elements of a group are represented in a class that is a subclass of `GArray`, a wrapper of `numpy.ndarray` which stores group elements instead of numbers. The elements must be stored in such a way that operations such as composition and inversion as well as indexing are possible.

Eq. 4.4, Eq. 4.5 and Eq. 4.6 show how a matrix can be used to represent a rotation in order to a transformation on coordinates in $\mathbf{Z}^2$, the explicit representation of a 90^o rotation in $\mathbf{Z}^2$, and how matrix multiplication can be used for composition (composition of 90^o rotation with 180^o rotation, resulting in a 270^o rotation). This illustrates how representing the elements of a group in the form of a matrix can be useful in order to easily combine elements. However, these matrix representation are not ideal when it comes to indexing operations. Take for example group C_4, the group of four 90^o rotations. For the indexing operation, it is only relevant which of these four rotations should be considered and representing this in terms of matrices would be unnecessarily complex.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \qquad \text{(Coordinate conversion)} \quad (4.4)$$

$$R(\theta = 90) = \begin{bmatrix} \cos 90 & -\sin 90 \\ \sin 90 & \cos 90 \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \qquad \text{(Rotation matrix } 90^o\text{)} \quad (4.5)$$

$$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \qquad \text{(Composition of matrices)} \quad (4.6)$$

Therefore, the elements that belong to a group can be represented in two manners: integer parameterisation and matrix parameterisation. However, as this work also concerns *split* groups that consist of a translation in addition to the transformation, matrix parameterisation is not always adequate. Instead, a homogeneous matrix (with an extra axis for the translation) may be used. To summarise, group elements can be parameterised in three manners: `int`, `mat` and `hmat`. Composition and inversion of

elements is more easily performed using the matrix representation (matrix `mat` for point groups, homogeneous matrix `hmat` for wallpaper or space groups)[7], whereas indexing is more easily done with the integer parameterisation. For a given element in the group, regardless of the parameterisation, the inverse, composition with another element, and an alternative parameterisation can be returned.

For each group, a subclass of `GArray` must be created that specifies the reparameterisation process between the integer and matrix parameterisation. We must also state how the identity element is represented and – for finite groups – define the elements that belong to that group. These subclass implementations for various groups of the `GArray` have to abide by the group axioms. The elements of the groups need to satisfy *associativity* and *closure*, and are required to have a *identity* and *inverse* element. These properties are tested for each subclass implementation for the various groups, in addition to being *reparameterizable* (reparameterizing a representation of an element from `int` to `hmat` back to `int` should result in the original element) and *closed under composition*.

The point groups C_4 and D_4 can be encoded using one and two integers respectively: $r \in \{0, 1, 2, 3\}$ to indicate the rotation index and $m \in \{0, 1\}$ to represent the presence of reflection. These `int` parameterisations of C_4 and D_4 and can be extended to $p4$ and $p4m$ (the wallpaper groups with C_4 and D_4 as point groups) by including u and v for translation over the x and y axis respectively. However, these elements can also be represented with a 2×2 matrix for the point groups and a 3×3 homogeneous matrix for the wallpaper groups. The parameterisation from `int` to `hmat` is shown in Figure 4.6. The integer representation (consisting of r for C_4 and r, m for D_4) can be converted to the (homogeneous) matrix representation by inserting r and m into the matrices as shown in Figure 4.6. To convert a given matrix back to the integer representation, the `atan2` function can be used to obtain r and $\frac{1}{2}(1 - \det(\text{mat}))$ for m.

For the 3D point groups and space groups, the matrix and homogeneous matrix representations are 3×3 and 4×4 respectively, to account for the extra axis in the

[7] Composition and inversion are implemented as matrix multiplication and matrix inversion respectively.

$$f(r,u,v) = \begin{bmatrix} \cos(.5r\pi) & -\sin(.5r\pi) & u \\ \sin(.5r\pi) & \cos(.5r\pi) & v \\ 0 & 0 & 1 \end{bmatrix} \qquad (p4)$$

$$f(r,m,u,v) = \begin{bmatrix} (-1)^m \cos(.5r\pi) & -(-1)^m \sin(.5r\pi) & u \\ \sin(.5r\pi) & \cos(.5r\pi) & v \\ 0 & 0 & 1 \end{bmatrix} \qquad (p4m)$$

Figure 4.6: *hmat parametrisation of p4 and p4m as functions of their int parameterisations.*

$z-$direction. The easiest group to parameterise is C_{4h}. Let us recall that the set of transformations that can together generate all elements of the group are the generators and every element in the group can be written as a product of the generators. For C_{4h}, the generators are a 180^o rotation over the y-axis and a 90^o rotation over the z-axis. We can represent rotations around the various axes in matrix form based on Eq. 4.7, Eq. 4.8 and Eq. 4.9. These matrix representations can be converted to homogeneous matrix representations with a 4×4 matrix that has the values for u, v and w (translations in the x, y and z direction respectively) in the last column.

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \qquad (4.7)$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \qquad (4.8)$$

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (4.9)$$

Every element in C_{4h} can be represented as a 3×3 matrix, including the generators, as shown in Figure 4.7. As C_{4h} is abelian, we can define its elements in terms of number of rotations around the rotation axes. This allows us to integer parameterise group C_{4h} with two integers representing the number of rotations around the two axes, where $y \in \{0, 1\}$ and $z \in \{0, 1, 2, 3\}$. Although D_{4h} is not abelian (as a rotation followed by a flip yields a different element than a flip followed by a rotation), we can – for the sake of parameterisation – assume that the flip follows the rotations over both axes, and extend the parameterisation of C_{4h} to represent group D_{4h} by adding an extra integer $m \in \{0, 1\}$ to represent the reflection. For the associated space groups, these parameters are used in addition to parameters u, v and w for translations in the x, y and z direction respectively.

$$g_1 = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \qquad g_2 = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(a) 180^o rotation over y **(b)** 90^o rotation over z

Figure 4.7: *The generators for group C_{4h} (also used for D_{4h})*

Whereas the parameterisation of group C_{4h} is fairly straight-forward, O is a little more complex as it is not abelian. Although O is composed of three fold axes, two generators (as given in Figure 4.8; the 90^o rotation over x and the 90^o rotation over y) are in fact sufficient to generate all 24 elements associated with group O. Instead of defining the number of rotations around the various fold axes, the elements of group O are generated at initialisation and stored in a sorted list that can be accessed to retrieve both the integer and matrix parameterisations. The indices of this sorted list ($0 \geq i < 24$) form the basis for the `int` parameterisation of the elements in group O. Similar to the extension of C_{4h} to D_{4h}, adding parameter $m \in \{0, 1\}$ to indicate reflection allows us to parameterise the 3D point group O_h that includes the rotations of group O and reflection. The associated space groups corresponding to these 3D point groups include a (u, v, w) translation over the $x-$, $y-$ and $z-$axes respectively.

To summarise, Table 4.2 provides an overview of the parameters for the `int`

$$g_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \qquad g_2 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix}$$

(a) 90^o rotation over x **(b)** 90^o rotation over y

Figure 4.8: *The generators for group O (also used for O_{ht})*

parameterisation and shapes associated with the `mat` and `hmat` parameterisations for all groups.

	`int`	`mat`	`hmat`
C_4	(r)	2×2	3×3
$p4$	(r, u, v)	2×2	3×3
D_4	(r, m)	2×2	3×3
$p4m$	(r, m, u, v)	2×2	3×3
C_{4h}	(y, z)	3×3	4×4
C_{4ht}	(y, z, u, v, w)	3×3	4×4
D_{4h}	(y, z, m)	3×3	4×4
D_{4ht}	(y, z, m, u, v, w)	3×3	4×4
O	(i)	3×3	4×4
O_t	(i, u, v, w)	3×3	4×4
O_h	(i, m)	3×3	4×4
O_{ht}	(i, m, u, v, w)	3×3	4×4

Table 4.2: *Overview of parameters for `int` parameterisation and shape for `mat` and `hmat` parameterisations for all groups.*

Mapping

A `GFuncArray` is an array that stores a discretely sampled function v on a group together with a map *i2g* from the set of sampling points I to a element g in a group G.

For each group, mappings *i2g* and *g2i* need to be specified. *i2g* is a meshgrid of all parameterisations of the stable group elements along with the given translation ranges for the three axes and *g2i* is the inverse mapping. For simplicity, as the extra dimensions have

no specific benefit for indexing, the group element parameterisations are flattened, *e.g.* original representation of shape $(2, 4, 2, \dots)$ for D_{4h} (representing the 180^o rotations, 90^o rotations and reflection respectively) becomes $(16, \dots)$.

v is a discretely sampled function based on the total number of stable elements in a group (elements which leave the center unchanged) and the shape of the kernel. For group O, v is an array of shape $(24, 5, 5, 5)$ with kernel size 5. Within the current implementation, the width, height and depth of the kernel are all assumed to be equal and only odd-sized filters are supported, as the origin around which will be rotated is not a point on the grid for even-sized filters.

Similar to `GArray`, the subclass implementations of `GFunc` for the various groups need to abide by the group axioms. Random elements $g, h \in G$ and f (a subclass `GFunc` implementation of G with discretely sampled function v) are used for the tests. In all occasions, g, h and f need to be associative ($(g \cdot h) \cdot f = g \cdot (h \cdot f)$) and invertible ($f_v = (g^{-1} \cdot (g \cdot f))_v$. Additionally, the identity axiom ($f_v = (e \cdot f)_v$) needs to be satisfied as well as that the mappings *i2g* and *g2i* of f need to be invertible.

G-convolution

The 2D g-convolution is available for Chainer and for Tensorflow. The 3D g-convolution is currently only available for TensorFlow. Whereas adding a new group requires implementing both a representation as a subclass of `GArray` as well as group-specific mapping as a subclass of `GFuncArray`, the majority of the necessities with regards to the convolution operation for 3D have been implemented for the purpose of this project and require no extra work for additional 3D groups, besides extending the `gconv3d_util` function that provides the gconv indices and shape information for the convolution operation to ensure the appropriate indices are returned as required by the gconv function for the transformation of the filter bank. As a result, function `gconv3d` that takes the input tensor, a filter, strides, padding style, gconv indices and gconv shape info as input parameters (most of which can be provided by the helper function) can be used as drop-in replacements of regular 3D convolutions.

5

Experiments

The aim of this thesis is to determine whether 3D group-convolutional neural networks are more data-efficient and therefore more useful in the medical domain than their regular 3D convolution counterparts. We attempt to establish this by focusing on the problem of false positive reduction in pulmonary nodule detection on chest CT in particular, which hopefully generalises to other volumetric data tasks. Reasons for conducting this experiment on the problem of nodule detection rather than any other medical image task includes social relevance, considering the increasing need for screening. However, besides social relevance, the most prominent reason is the availability of large annotated datasets for this task.

To investigate whether a G-CNN is indeed a beneficial approach for the problem of false positive reduction of lung nodules, we perform experiments using a network with

g-convolutions for various groups. We compare these results in terms of performance and speed of convergence with a network with a similar architecture, but that uses regular 3D convolutions instead. An additional benefit of a large quantity of available data, besides increased chances of decent performance, is that it allowed us to conduct experiments on how training set size affects the difference in performance between G-CNNs and regular 3D CNNs on a logarithmic scale.

The choice to focus on the last step of the nodule detection process, that of false positive reduction rather than segmentation or candidate generation, was primarily made as this provided is with a reliable, comparative metric for evaluation while ensuring the results were not reliant on third party algorithms (*e.g.* G-CNN detection followed by some other FP-reduction measure) nor compounded (*e.g.* G-CNN detection model followed by G-CNN FP-reduction model). The former would create a dependency on a third-party algorithm, which may influence results in an unpredictable manner, and the latter would create difficulties in pinpointing possible benefits (*e.g.* worse candidate generation combined with improved false positive reduction could still yield an overall worse result). Additionally, G-CNNs in their two-dimensional variant were verified to be effective on classification tasks, which is more similar to FP-reduction than segmentation or object detection tasks.

This chapter describes the essentials of the experiment set-up: the data (Sec. 5.1), the training method (in terms of architecture and hyperparameter settings; see Sec. 5.2) and lastly the evaluation metrics (Sec. 5.3).

5.1 Data

The LUNA16 challenge presents two distinct tracks: nodule detection and false positive reduction. The FP reduction set provided a set of candidate locations that contained a significantly larger amount of non-nodule candidates than a realistic implementation of a

candidate generation model, such as by AIDENCE, would generate. Therefore, the choice was made to disregard this dataset and focus on a realistic output of a candidate generation model instead, which will mostly contain non-nodules that more closely resemble real nodules.

All scans from the NLST and LIDC-IDRI sets have been processed by the AIDENCE candidate generation model and the resulting center coordinates of potential nodules form the basis of our data. These two data sources were chosen due to their availability, volume and annotation quality. Due to the higher volume of available NLST scans and our desire to have a varying training dataset sizes, training and validation is done on processed scans from the NLST dataset. For the test set, we use the LIDC/IDRI dataset, partly due to the higher annotation quality (four expert radiologists vs. one), and partly because this enables us to use the performance metric as described in Sec. 5.3.

The potential nodules from the NLST set have been divided into a validation (10%) and training set (90%). All potential nodules from the LIDC/IDRI set together form our test set. For practical purposes, we offer the training data in a balanced manner to the neural network for training, but both the validation and test set are accurate representations of the ratio positive and negative examples a boosting network may encounter based on the candidate generation model.

A total of 30.326 data samples in total are available for the training. A subset of training data, starting at 30.000 samples, is taken repeatedly, so not all available data is used. 326 data samples (163 random samples for each label) are discarded completely. A data subset of size $2n$ is created by taking the first n positive and first n negative samples from the dataset as a whole, ensuring the subset is balanced with a fifty-fifty positive/negative class label ratio, and each smaller training set is a subset of all larger training sets. The details of the train, validation and test sets are specified in Table 5.1.

Set	Source	Name	Samples	Positive %	Negative %
Training	NLST	nlst-balanced	30.326	50.0	50.0
Validation	NLST	nlst-unbalanced	8.889	20.6	79.4
Test	LIDC/IDRI	lidc-unbalanced	8.582	13.3	86.7

Table 5.1: *Specifics of the training, validation and test set sizes and class ratios.*

5.1.1 Data Preprocessing

All scans from the NLST and LIDC-IDRI sets with a slice thickness of 2.5mm and lower have been processed by the Aidence candidate generation model and resulted in (z, y, x) coordinates of potential nodules. These coordinates have been used to extract *candidate patches* of $13 \times 120 \times 120$ pixels, containing suspect lesions that may or may not be lung nodules. These candidate patches are cut-outs of the original scan around the potential nodule center, resampled with linear interpolation in such a way that each pixels represents 1.25mm $\times$ 0.5mm $\times$ 0.5mm of lung tissue.

However, the values still need to be normalized. The unit of measurement in CT scans is the Hounsfield Unit (HU), a measurement of radiodensity. Values of interest for lung nodule detection and classification lie approximately between -1000 HU (*air*) and 300 HU (*soft-tissue*). To convert from the normal units found in the CT data to Hounsfield Units, we apply the linear transformation in Eq. 5.1, where `slope` and `intercept` are retrieved from the original DICOM using the tags *(0028, 1053) Rescale Slope* and *(0028, 1052) Rescale Intercept* respectively.

$$\texttt{hu} = \texttt{pixel\_value} * \texttt{slope} + \texttt{intercept} \tag{5.1}$$

The values between $[-1000, 300]$ are mapped to a $[-1, 1]$ range in the manner described in Eq. 5.2. Values were not capped as to not lose context information, so patches may still contain values outside of this range.

$$\mathbf{x} = 2 \cdot \frac{\mathbf{x} + 1000}{-1000 + 300} - 1 \tag{5.2}$$

Each dataset was whitened by subtracting the mean and subsequently dividing by the standard deviation to achieve a zero mean with standard deviation of one. This mean and standard deviation depends on the subset of the data currently being trained upon, and varies for different dataset sizes but not between networks trained on the same set size.

The validation- and test set patches are cropped from $13 \times 120 \times 120$ pixels to patches of $12 \times 72 \times 72$ pixels around the nodule center pixel. This patch size was chosen as it represents approximately 15mm $\times$ 36mm $\times$ 36mm of lung tissue – large enough to contain the entire nodule (recall that nodules with a diameter exceeding 30mm are considered masses, not nodules) and to provide some context such as location and possible movement within the lung, but not too big to ensure the patch does not include many distracting context features. The training data is temporarily kept at the original shape to allow for data augmentation techniques that benefit from postponing cropping to a later stage such a scaling, rotating and translating.

It should be noted that many descriptions of approaches for false positive reduction of lung nodules reveal that the convention is to use cubic patches of equal dimensions in which the amount of tissue represented in each direction is equal. However, this presents us with two problems. First, the average slice thickness is 2.5mm while the average pixel spacing is .5mm. Upscaling 2.5mm to represent 0.5mm of lung tissue requires a lot of interpolation, while downscaling of the pixel spacing (*e.g.* to $1 \times 1 \times 1$mm of lung tissue) essentially throws away potentially distinctive information. Secondly, in assessing whether or not a lesion is a pulmonary on an axial lung CT, a radiologist will typically scroll through the slices surrounding the suspect lesion and not only take the physically distinctive features the potential nodule into account, but also deviations of the center position in the $x-$ or $y-$direction throughout the different images, as movement implies a blood vessel. It could therefore be argued that the patch shape should not be cubic, but should be larger in the $x-$ or $y-$direction than in the $z-$direction, which is why this

choice – challenging the convention of cubic patches – was made.

5.1.2 Data Augmentation

As the only distinguishing feature between the baseline and the g-convolutional network is to be the implementation of the convolution layer, the same data augmentation that was applied to the baseline will be applied to all networks, including the g-convolutional networks. Although the expectation is that a group convolutional neural network with symmetry group g will be robust against the elements of g, the network may still benefit from other augmentation methods, such as rotations out of the group and scaling.

The following data augmentations were applied at random to the batch:

- **Rotations**: for each volume in the batch, an angle between $0 - 360$ degrees is chosen and the twelve individual images that make up the volume are rotated at that angle. There are no rotations in the z-axis.
- **Reflection**: each volume in the batch is flipped either in the x, y or z axis, or no reflection operation is applied at all.
- **Translation**: while cropping the volumes in the batch to the desired size, the chosen center point for the cropped volume can deviate 0 to 3 pixels from the true center in the x and y axis and 0 to 1 pixel in the z axis, essentially creating translations of $0 - 3$ pixels in either direction for x and y and a translation of $0 - 1$ pixels in the z.
- **Scaling**: each volume in the batch is scaled between $0.8 - 1.2$ times the original volume using spline interpolation of order 3.
- **Noise**: noise is sampled from a standard normal distribution multiplied by 0.05, to ensure small values, and is added to the original volume for each volume in the batch
- **Value remapping**: a $y = ax + b$ transformation on Hounsfield Unit values, based

on empirically decided upon values to ensure the value changes are not too large. The motivation behind this remapping is that values in the lung are not constant – the same tissue does not always have the same HU value.

Empirically chosen values $x_{[1,2,3,4]}$:

$x_1 = -0.84375$ $\quad$ $x_2 = -0.6875$

$x_3 = 0.5625$ $\quad$ $x_4 = 0.71875$

Helper values $s_{[1,2,3,4]}$:

$s_1 = r_1 \cdot (x_2 - x_1) + x_1$ $\quad$ $s_2 = r_2 \cdot (x_2 - x_1) + x_1$

$s_3 = r_3 \cdot (x_4 - x_3) + x_3$ $\quad$ $s_4 = r_4 \cdot (x_4 - x_3) + x_3$

Values a and b for $y = ax + b$:

$a = (s_4 - s_2)/(s_3 - s_1)$ $\quad$ $b = (s_2 - s_1) \cdot a$

Each augmentation method has a configurable keep probability which determines the chance that that augmentation technique is not applied to the volume, to ensure the augmented mini-batch does not stray away too far from the original data. This is parameter is set at 0.2 during experimentation.

5.2 Method

In addition the data an experiment setup is needed, which this section details both in terms of architecture and training hyperparameters. The architecture used for the false positive reduction of lung nodules was initially inspired by the models used by Cohen & Welling (2016) to test the two-dimensional implementation of group-convolutional neural networks against regular convolutional neural networks on CIFAR-10 and MNIST-rot. Alterations were made accommodate the three-dimensional nature of the dataset and to finetune results for this particular problem on the validation set.

The hyperparameters choices, such as learning rate and optimiser, were based on best practices inspired from other lung CAD systems and nodule false positive reduction applications. Besides academic sources, high-scoring Kaggle Data Science Bowl 2017 competitors and LUNA16 challengers generally provided thorough descriptions of their approach which formed a point of reference for the choices to be made in this project. Again, a hyperparameter search could theoretically improve the results remarkably, but was decided against due to time constraints and deemed unneccessary considering that the score of the system with the current settings was already competitive and therefore sufficient for our goal.

5.2.1 Architecture

The baseline model consists of the following components: *convolutional* layers (g-convolutional or regular 3D convolution), *batch normalisation*, *activation*, *dropout*, 3D *max pooling* and a *dense* layer. The sequence of layers and their corresponding output tensor shapes for the baseline are described in Table 5.2. The baseline architecture with regular convolutions is referred to as $Z3$-CNN. Similarly, the variant of the baseline network with a specific symmetry group is referred to as -CNN preceded by the uppercase name of that (point) group, *e.g.* C_{4h}-CNN.

The activation is ReLU in every activation instance, except for the Softmax activation following the Dense layer. Dropout was applied with a keep probability of .7. 3D Max pooling was done with same padding, and a filter size and stride of $1 \times 2 \times 2$ in the first instance (row 2 in Table 5.2) to reduce only in the x- and y dimensions, and $2 \times 2 \times 2$ in the later two instances (row 5 and 8 in Table 5.2) and was added to separate the dimensionality reduction from the convolution step. The tensor is flattened to provide input for the Dense (fully-connected) layer, which is simply a linear operation on the input vector followed by an activation in the form of $\sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x} + b)$ where σ is the Softmax activation function.

Batch normalisation was done with a trainable offset β and scale γ initialised at a value of

	Layer *(name)*	**Feature maps** *(nr. of out channels)*	**Tensor shape** *(b, z, y, x, c)*
1	Conv-BN-Act	16	(30, 12, 72, 72, 16)
2	MaxPool3D		(30, 12, 36, 36, 16)
3	Conv-BN-Act	16	(30, 12, 36, 36, 16)
4	Dropout		(30, 12, 36, 36, 16)
5	Conv-BN-Act	32	(30, 12, 36, 36, 32)
6	MaxPool3D		(30, 6, 18, 18, 32)
7	Conv-BN-Act	32	(30, 6, 18, 18, 32)
8	Dropout		(30, 6, 18, 18, 32)
9	Conv-BN-Act	64	(30, 6, 18, 18, 64)
10	MaxPool3D		(30, 3, 9, 9, 64)
11	Conv-BN-Act	64	(30, 3, 9, 9, 64)
12	Dense-Softmax		(30, 2)

Table 5.2: *Architecture baseline Z_3*

zero and one respectively, such that the tensor is normalised to $\frac{\gamma(x-\mu)}{\sigma} + \beta$.

A softmax activation function is applied to the last layer of the network, resulting in two values that indicate the probability whether a data sample x_i is a nodule or not. However, the only value of interest for false positive reduction is prediction $\hat{y}_i$ that x_i is a nodule, referred to in Sec. 5.3.2 as the degree of suspicion for x_i. These values, along with the scan metadata, are used to construct a submission file that contains the *Series UID*[1], the x, y and z coordinates of the center of the nodule in world millimeters[2], and the probability the model returns that x_i is in fact a pulmonary nodule. This submission file can be used as input for the evaluation script and results in a FROC curve and score.

Convolution

The implementation of the training process in TensorFlow was done in such a way that by

[1] DICOM tag *(0020,000E) Series Instance UID* that is a unique identifier of the series.

[2] Conversion from pixel coordinates to world millimeters is done with a linear equation using DICOM tag *(0020,0032) Image Position Patient* as slope (offset) and *Pixel Spacing* and *Slice Thickness* as slope (scale).

supplying the appropriate group (one of Z_3, C_{4h}, D_{4h}, O or O_h) as a command line argument, the same architecture and set of hyperparameter settings were used in each training instance, with the only variation being the implementation of the convolutional layer – either a call to the regular TensorFlow implementation of a 3D convolution (with stride 1, kernel size 3, same padding and the addition of a bias variable initialised at zero), or the group convolution with the desired group.

The g-convolutional variant of the baseline network calls a function that requires three parameters besides the tensor itself: the input group, the output group, and the number of desired output feature maps. This function does three things; calculate the number of input and output channels to be used to ensure a similar number of parameters to the baseline, retrieve the static data required for the g-convolution (indices, shape information and shape of the filter tensor) and perform the group convolution.

The number of input channels for the g-convolution operation is determined by taking the number of output feature maps of the previous layer, and dividing this by the order of the input group (see Eq. 5.3[3]).

The output group is always equal to the input group, except for the first instance of the convolution layer (row 1 in Table 5.2) where the input group is Z_3, and determines the number of output feature maps. The number of desired output feature maps for the g-convolutional layer is the number of output feature maps for the corresponding baseline layer at that stage (as can be seen in the *Feature maps* column in Table 5.2) divided by the root of the order of the input group (see Eq. 5.4[4]). The actual output channels is a different following the g-convolution, than the regular convolution.

This is to ensure each model roughly approximates the same number of learnable parameters. Roughly, rather than exactly, as not all group orders are squares (*e.g.* $\sqrt{|C_4|} = \sqrt{4} = 2$ but $\sqrt{|D_4|} = \sqrt{8} \approx 2.83$) and not all number of channels are neatly divisible to integers (*e.g.* $\frac{16}{\sqrt{4}} = 8$, but $\frac{16}{\sqrt{8}} \approx 5.66$). These output values are rounded to the nearest integer.

[3] $|G|$ refers to the order of the group
[4] See footnote ↑

$$\text{in\_channels} = \frac{\text{input channels}}{|G|} \tag{5.3}$$

$$\text{out\_channels} = \frac{\text{input filters}}{\sqrt{|G|}} \tag{5.4}$$

5.2.2 Training set

To compare the data efficiency of the group convolutional neural nets for the different groups in comparison with the baseline, we train the architecture as defined in the previous section on the training set sample sizes of 30, 300, 3.000 or 30.000. These training set sizes were chosen specifically as a logarithmic scale of data set sizes was preferred to evaluate the effects on large differences in data set sizes, and the largest available set size contained 30.326 patches in total.

Each network for a set training set size (*i.e.* same training set size, different group) was trained on the same subset of the data, chosen at random to be the first n samples of both the set of positive patches as well as the set of negative patches, to ensure the training set is *balanced*, indicating that half of the suspect lesions qualify as nodules while the other half does not. This was also represented as such in the mini-batches. This choice for a balanced training set was made out of convenience, as it allowed us to easily resize the training set to smaller subsets, and - most importantly - required no remedies to counteract the negative effects of class imbalance such as oversampling, tuning of the loss function or unequal weights. An unbalanced dataset that includes more negative samples could theoretically improve performance.

5.2.3 Hyperparameters

The data was presented to the neural net in mini-batches of 30 data samples. The smallest dataset was contained only 30 samples and it was desirable to keep configurations

between training runs on different models and different training set sizes as equal as possible. Therefore, a batch size of 30 was used for every training set size.

The *learning rate* was set at 10^{-4} and the CNN was trained by minimising the *cross-entropy*, the most appropriate loss function for classification problems, with *Adam* as an optimisation technique. The weights were initialised using *uniform Xavier initialisation*.

Each model was trained three times to achieve an indication of model performance and deviation between runs. Data augmentation techniques, as specified in Sec. 5.1.2, were applied at random to each individual volume in the mini-batch with a keep probability parameter setting of 0.2.

Although it was desirable to have all hyperparameters set to the same value during training to allow for comparison of scores between different dataset sizes as well, this turned out to be not practically feasible. Competitive scores were achieved for the baseline network with the largest dataset (30k training samples) with approximately 50k gradient updates. However, this many gradient updates seemed excessive and not like a reasonable amount for a dataset size of only 30 samples – a dataset size of which, due to the difficult nature of the problem, one would already expect the system to perform poorly on, even with g-convolutions, and has only been added to the experiment to form an extra basis for comparison. For such a small dataset size, it is expected that convergence, though at a higher loss, happens at a much earlier point and training for a number of gradient updates equal to the models trained on the largest dataset would be a waste of compute resources.

Therefore, instead of using the same number of gradient updates (equivalent to an equal number of epochs, since the batch size is constant), the loss on the training- and validation set is plotted against the number of epochs trained. This visualisation of convergence is then used to determine an appropriate number of epochs to train for each of the four training set sizes. Multiple early runs and plots revealed a tendency of the CNN with regular convolutions to take longer to converge than the G-CNNs, which lead us to determine the appropriate number of epochs based on the convergence of the regular CNN. The number of epochs trained for 400, 200, 100 and 50 with respect to

increasing training set sizes. To ensure optimal performance for all models and as a form of regularization to prevent the model from overfitting, we use *validation-based early stopping* as a stop condition.

5.3 Evaluation metrics

Crucial to any machine learning task, in addition to the data and the model, is the manner with which to evaluate the performance of the trained model. Ideal CAD software will have a high sensitivity and at a low false positive rate. However, it is not straight-forward what the desired trade-off between these two performance measures should be. How would one decide which is better: a system that achieves a higher sensitivity but at a higher false positive rate, or the system with a lower false positive rate overall at the expense of the sensitivity? The answers to these questions may even vary among institutions or users. An even more fundamental problem is that the difficulty in establishing an absolute reference standard for what constitutes a lung nodule. Analysis of the radiologist evaluation of the nodules revealed that there is a large group of findings about which there is no consensus among radiologist, so how would one evaluate the (lack of) detection of that finding by a CAD system? Both aspects present a challenges in evaluation and comparison of our implementation to state-of-the-art lung nodule detection systems. Additionally, to compare the performance of various CAD technologies, the systems will need to be scored in an identical manner, ideally in such a way that it takes into account the difference in preference users may have for the sensitivity/false positive rate trade-off.

Unfortunately, large evaluation studies investigating the performance of different state-of-the-art CAD systems are scarce. However, *LUNA16* is a since 2016 running open lung nodule detection challenge using the LIDC-IDRI dataset that attracted both academic and commercial participants. To score the performance of this thesis' baseline

network (see Sec. 5.2.1) and its group-convolutional counter-parts, we will abide by their performance metrics, that provides both a reference standard and a visualisation of the sensitivity and false-positive rate trade-off. The `python` script that performs the evaluation using the constructed submission file can be found here.

As the overall objective of lung nodule CAD software is to provide assistance in the detection of, above all, *malignant* nodules, we will also want to analyse and evaluate the resulting model predictions with regards to malignancy. Therefore, Sec. 5.3.3 reveals how we reach a consensus from the four radiologist assessments of malignancy on the LIDC-IDRI dataset.

5.3.1 Reference standard

The evaluation method used in LUNA16 was introduced in the earlier *ANODE09* study. Out of the 1018 available thoracic CT scans from the LIDC/IDRI database, only those with a slice thickness of 2.5mm or lower were considered, of which there are 888 in total. The reference standard of the LUNA16 challenge consists of all nodules on these 888 chest CTs of which the two-phase process resulted in acceptance by at least 3 out of 4 radiologists.

Additionally, the reference standard differentiates between *relevant* and *irrelevant* findings. Irrelevant findings refer to the type of findings that, if the CAD software were to detect them, it should neither be applauded nor chastised for it. Three distinct categories of irrelevant findings can be distinguished: findings that mimic a nodule but that an expert observer believes not to be a nodule; nodules with benign characteristic; and nodules that are too small (smaller than $<$ 3mm in diameter in the largest axis). Any CAD marks in regions around irrelevant findings are ignored in the evaluation.

Evaluation is performed by measuring the detection sensitivity of the algorithm and the corresponding false positive rate per scan. *Sensitivity*, also referred to as *true positive rate* or *recall*, expresses the ratio between the existing samples (true positives *and* false negatives) and the detected samples (true positives):

$$\text{Sens} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{5.5}$$

In case of pulmonary nodules CAD, this is the proportion of nodules that have been detected by the CAD software. The LUNA16 considers a nodule candidate to be a true positive or "hit" if that candidate center coordinates are within distance r from the center of a nodule in the reference standard.Distance r is specified to be the diameter of the largest axis of the nodule from the reference standard divided by two as can be seen in Eq. 5.6:

$$r = \frac{\text{nodule diameter}}{2} \tag{5.6}$$

A candidate is considered to be a false positive if the provided candidate center point does not fall within radius r of any of the nodules from the reference standard. Similarly, a false negative is a candidate that the CAD software did not detect as a nodule, even though the it is categorised in the reference standard as a relevant finding. The last category, true negatives (patches that are seen as nodules by neither the CAD software nor the radiologists) have little to no meaning in the context of evaluating nodule detection CAD software due to the overwhelming volume of points that will meet this requirement.

It should be noted that once a finding reported by the CAD software is linked to a finding in the reference set, that finding is removed from the reference set so each nodule can only be marked once. Additionally, if a candidate is not within distance r of a finding in the reference set, but does produce a hit with $1.5 \times r$, it will simply be discarded and not count towards the sensitivity of the algorithm.

5.3.2 Free-Response Operating Characteristic

To combat the differences in preference in the trade-off between sensitivity and false positive rate, the sensitivity is plotted as a function of the average number of false positive markers per scan, calculated as specified in Eq. 5.7.

$$\text{FP-rate} = \frac{\text{total FPs}}{\text{total number of scans}} \tag{5.7}$$

This function is referred to as a Free-Response Operating Characteristic (FROC) analysis. In order to obtain a point on the FROC curve, we provide a degree of suspicion p. For various thresholds t, the CAD findings whose degree of suspicion $p \geq t$ are selected and the sensitivity and false positive rate calculations are done based on the selected candidates. All thresholds that produce a unique point on the FROC curve are connected, and the point associated with the lowest false positive rate is connected to $(0, 0)$, while from the point with the highest false positive rate onward the FROC curve is extended to 8 false positives per scan by a straight horizontal line.

To achieve a general score for a system, allowing us to easily compare multiple systems, we average the sensitivity at seven predefined false positive rate values: $\frac{1}{8}$; $\frac{1}{4}$; $\frac{1}{2}$; 1; 2; 4; and 8. These values can be extracted from the FROC curve in case no threshold t produces the exact desired false positive rate.

It should be noted that not all nodules in the LUNA16 reference standard are included in the test set (1148 out of 1186); therefore, the highest possible sensitivity our model can achieve is approximately 96.8%, as setting threshold t to 0.0 results in the inclusion of all 1148 suspect lesions, regardless of the model performance. Additionally, small adjustments to the evaluation script were made for the resulting visualisation to be appropriate for this project. *E.g.* the current implementation allows multiple FROC curves to be plotted within the same graph for better comparison, and bootstrapping with 1.000 bootstraps to compute a 95% confidence interval was removed to prevent a cluttered, chaotic look when multiple FROC curves are visualised within the same graph.

5.3.3 Issues

An issue with this form of evaluation is that all nodules, big or small, subtle or easily spotted, malignant or benign, are treated and weighted similarly. In practice, for the purpose of assisting in follow-up management and ultimately early detection of lung cancer, it would be beneficial to be able to give weights to the various nodules, or at least analyse the result w.r.t. relevant follow-up measures. It would be interesting to compare results on the various composition types (solid, part-solid or ground-glass lesion), sizes, and in additional features a nodule may exhibit, which were fortunately recorded as part of the creation of the LIDC/IDRI dataset – the most relevant of which for our cause is an estimate of malignancy.

During the annotation process for the LIDC dataset, the readers were - among other things - asked to provide a subjective assessment of the likelihood of malignancy. As no metadata usually required to evaluate malignancy such as family history or patient information (age, sex, etc.) was available to the readers, they were to assume the scan originated from a 60-year old male smoker. Likelihood was assessed on a five point scale, with 1 through 5 representing *Highly Unlikely, Moderately Unlikely, Intermediate, Moderately Suspicious*, and *Highly Suspicious* respectively. For this exercise, we consider a nodule to be malignant (*i.e.* relatively important for a CAD system to report) if at least three radiologists scored the nodule as moderately or highly suspicious. We are more strict with regards to benign nodules, only calling them such if *all* radiologists judged them to be highly unlikely or moderately unlikely to be malignant.

In one case, only two radiologists provided their assessment on the nodule malignancy, one as *moderately unlikely* and the other as *moderately suspicious* – this datapoint was discarded and considered neither malignant nor benign. In 389 cases, only three out of four radiologists provided a malignancy suspicion. In those cases, we require there to be a consensus among all three radiologists to consider a nodule malignant.

Additionally, the average number of false positives per scan is calculated on a $.125 - 8$ interval. However, in clinical practice – the intended usecase for this type of nodule

detection software – 8 false positives per scan on average would arguably be distracting and simply too many. A system overall score benefits from an increase in sensitivity at the higher number of false positives per scan, but although this is theoretically interesting, it is not very useful in practise. An alternative scoring with an avg. FP/S interval $0.125 - 1$ is provided in appendix II.

6

Results

This chapter outlines the results of the conducted experiments and analysis thereof. To recapitulate, the group-convolutional neural network framework was extended to be applicable to three-dimensional data by adding implementations for groups C_{4h}, D_{4h}, O, and O_h. An architecture using these groups as basis for the g-convolutions, as well as a baseline with regular convolutions, was trained on the candidates generated on the NLST dataset and tested on a subset of the LIDC-IDRI dataset for four different training dataset sizes: 30, 300, 3.000 and 30.000. The sections in this chapter specify the results with regards to performance, speed of convergence and alternative metrics.

Unless specified otherwise, all nodules from the LIDC-IDRI dataset between 3mm and 30mm are considered. There are a total of 1186 nodules in the list of to be detected nodules; the dataset used contained 1141 out of these 1186 nodules. As such, the

sensitivity of any given system tested on this dataset is at most 96%. Out of the candidates in the dataset, two were double detections on a single nodule and therefore ignored and thirteen were on the list of to be excluded nodules. The average number of candidates per scan was $\sim$ 9.7.

6.1 Performance

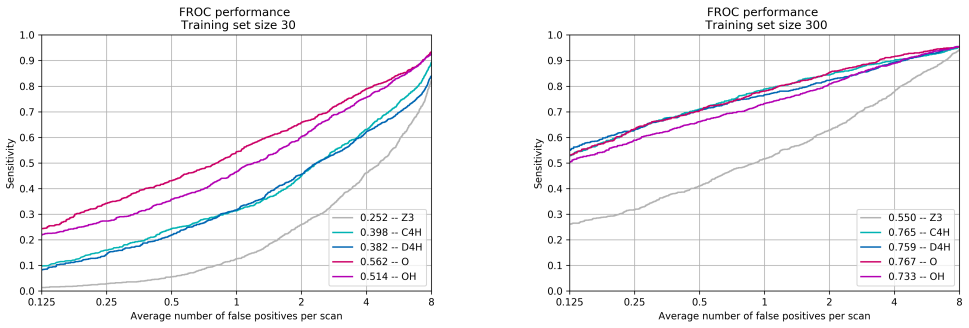
A FROC curve with a 95% confidence interval is provided in Appendix III for each training run on the various dataset sizes as well as a table containing the intermediate points on the curve with .125, .25, .5, 1, 2, 4 and 8 false positives per scan on average respectively and the combined score.

However, the point of interest for this work is to discover how the various models compare, especially with respect to the baseline. For this purpose, Figure 6.1 plots the FROC curve (confidence interval was omitted for clarity) for the various implemented groups for each training set size. As noted before, the training sets among groups for a similar size are identical, as is the test set for all runs. Table 6.1 contains the overall score for each group and training set size combination, and has the highest (**bold**) and lowest (*italic*) scoring model per training set size highlighted.

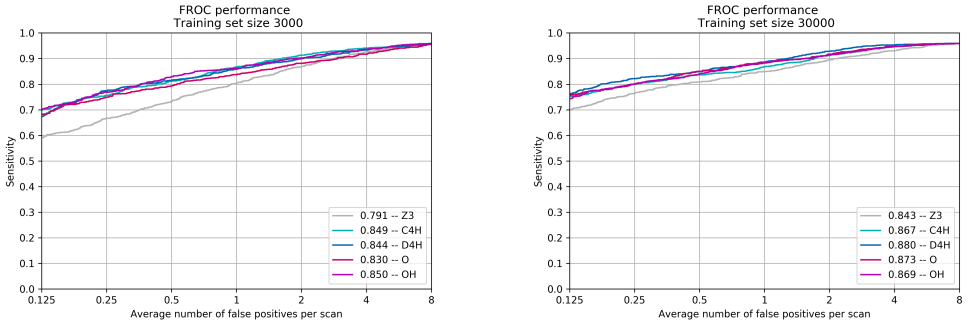
	Z_3	C_{4h}	D_{4h}	O	O_h
30	*0.252*	0.398	0.382	**0.562**	0.514
300	*0.550*	0.765	0.759	**0.767**	0.733
3000	*0.791*	0.849	0.844	0.830	**0.850**
30000	*0.843*	0.867	**0.880**	0.873	0.869

Table 6.1: *Overall score for all training set size and group combinations.*

What is apparent from both the aggregated system scores and the FROC curves per



(a) 30 samples **(b)** 300 samples



(c) 3.000 samples **(d)** 30.000 samples

Figure 6.1: *FROC curve results for all groups per training set size.*

training set size, is that all g-convolutional neural networks all outperform the baseline. The smaller the training set, the larger the increase in performance with respect to the baseline seems to be. In two out of three cases, the baseline score was beaten by a g-convolutional variant of the baseline on a dataset 10% the size of the original (group O on 30 samples outperforms Z3 on 300; group O_h on 3.000 samples outperforms Z3 on 30.000). However, there does not seem to be a consistent best performing group, with the overall systems scores per training set size for the G-CNNs being relatively close. An exception to this are the results of runs with training set size 30, where the cubic symmetry groups (O and O_h) not only appears to significantly outperform the baseline, but also display an increase in performance with respect to the rectangular cuboid symmetry groups (C_{4h} and D_{4h}).

Although Figure 6.1 is by far the most illustrative graph to summarise the results of this

endeavor, interesting additional questions to answer are how the g-convolutional networks compare the regular networks in terms of speed of convergence and by alternative success metrics. Therefore, results with respect to these questions are outlined in the following sections.

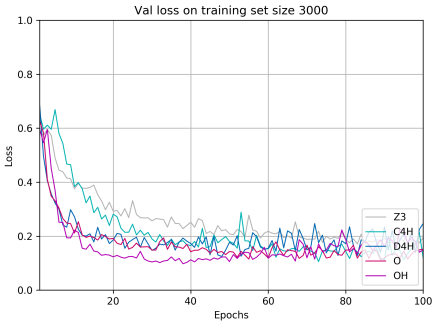
6.2 Speed of convergence

Figure 6.2 plots the loss on the train and validation set throughout training for training runs with dataset size 30.000 and 3.000, recorded at each epoch for 50 and 100 epochs respectively. Furthermore, Table 6.2 illustrates how many epochs of training were necessary for each group to achieve a loss on the validation set that was equal to or lower than the lowest achieved validation loss on the baseline.

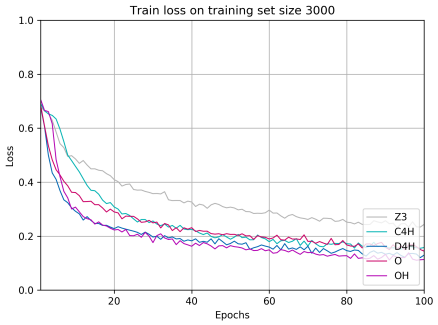
	Z_3	C_{4h}	D_{4h}	O	O_h
3.000 (*100 epochs total*)	*82*	33	22	21	**11**
30.000 (*50 epochs total*)	*41*	4	9	7	**3**

Table 6.2: *Number of epochs after which the loss is equal to or lower than the lowest validation loss achieved on the baseline for each group.*

From Table 6.2, we can conclude that the group-convolutional networks quickly approach the lowest achieved loss for the baseline in a fraction of the number of epochs that was necessary for the baseline. This is consistent with the observation from Figure 6.2 that the group-convolutional models show a faster decline in loss within the early stages of training compared to the baseline. Most group-convolutional models are at a fairly consistent validation loss after approximately 20-30% of training, unlike the baseline which only shows consistency in the later stages.

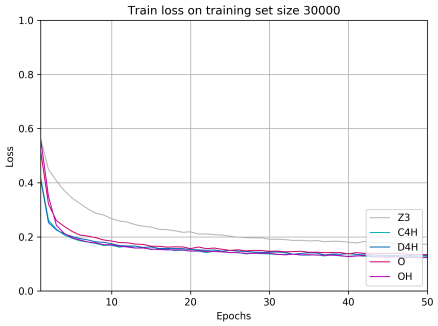


(a) Validation loss on 3.000 samples



(b) Train loss on 3.000 samples

(c) Validation loss on 30.000 samples



(d) Train loss on 30.000 samples

Figure 6.2: *Plots of the train and validation losses throughout training for all groups on training set sizes 3.000 and 30.000.*

6.3 Alternative metrics

As highlighted in Sec. 5.3.3, the current form of evaluation of model does arguably exhibit some issues as all nodules – regardless of type and size or other priorities – are weighed similarly. In practice, CAD software aims to assist a radiologist in detecting and reporting nodules for which follow-up is required, *i.e.* primarily potentially *malignant* nodules. The biggest indicator for malignancy (besides volume doubling time, which cannot be inferred from a single scan) is size, though considering the dataset at hand, we also have an estimation of malignancy available.

Therefore, this section provides an alternative scoring for the model performance based on a selection of most significant nodules, either due to their size or the provided subjective malignancy rating. For clarity, results w.r.t. training set sizes 30 and 300 are omitted, though these can be provided.

6.3.1 Nodule Size

Various guidelines for nodule detection follow-up management exist, most notable of which are the British Thoracic Society (BTS) guidelines and those by the Fleischner Society. Whereas the nodules are generally defined to be within the 3mm and 30mm range, the BTS does not consider nodules smaller than 5mm in diameter to be of interest and the Fleischner Society (and Lung-RADS) require a diameter of at least 6mm for follow-up. Additionally, it could be argued that the added benefit of CAD software would mainly be to detect nodules that do require follow-up, but may be easily missed by the available radiologist due to its relatively small size, *i.e.* in the $5 - 10$mm range. Table 6.3 delivers an overview of system scores within these alternative ranges.

Practically all systems, with the exception of the baseline on nodules > 10mm, benefit from the added constraints of a larger nodule size, in a fairly similar manner. Interesting is that all group-convolutional models benefit most from the constraint that nodules are to

		Z_3	C_{4h}	D_{4h}	O	O_h
3.000	>3mm	*0.791*	0.849	0.844	0.830	**0.850**
	>5mm (BTS)	*0.822*	0.886	**0.892**	0.862	**0.892**
	>6mm (FS)	*0.825*	0.900	**0.913**	0.877	0.900
	>10mm	*0.744*	0.881	**0.894**	0.859	0.875
	>5 & <10mm	*0.856*	0.886	0.890	0.863	**0.899**
30.000	>3mm	*0.843*	0.867	**0.880**	0.873	0.869
	>5mm (BTS)	*0.869*	0.894	**0.907**	0.906	0.900
	>6mm (FS)	*0.873*	0.904	**0.918**	**0.918**	0.910
	>10mm	*0.849*	0.895	0.907	**0.915**	0.905
	>5 & <10mm	*0.877*	0.892	**0.907**	0.902	0.897

Table 6.3: *Overall scores for various minimum nodule diameter sizes.*

be larger than 6mm in diameter, whereas the baseline system performs best on nodules between 5 and 10mm in diameter. This can be explained with the observation that the baseline system hardly seems to benefit from the constraint that nodules must be larger than 10mm – with a relatively similar performance on 30.000 samples and a lower performance on 3.000 samples –, which is slightly counter-intuitive as one would expect a system to be better at recognising relatively large nodules. However, the differences in performance between nodule size constraints do not differ enough to draw conclusions based on these results.

6.3.2 Malignancy

The prime motivation for nodule detection is to find potentially malignant nodules. This section highlights the results of the systems performances with respect to malignancy. To recapitulate, a nodule is considered *malignant* for this exercise if at least three radiologists suspected the nodule to be moderately or highly suspicious. In total, 129 nodules qualify as malignant according to this constraint.

FROC scores

First and foremost, the models were scored based on the sensitivity at the various false positive rate intervals, where all nodules that are in fact nodules but do not qualify as malignant are considered *irrelevant findings*, *i.e.* do not count towards the true positives, nor towards the false positives. Table 6.4 outlines the overall system scores with respect to these constraints, compared with the original scores of the systems on all nodules.

		Z3	C_{4h}	D_{4h}	O	O_h
3.000	*original*	*0.791*	0.849	0.844	0.830	**0.850**
	malignant	*0.808*	0.939	**0.953**	0.917	0.945
30.000	*original*	*0.843*	0.867	**0.880**	0.873	0.869
	malignant	*0.907*	0.964	0.971	**0.977**	0.973

Table 6.4: *Overall system scores for malignant nodules.*

All systems show a higher sensitivity towards malignant nodules compared to the original experiment on all nodules, though the difference is relatively small for for the baseline with 3.000 samples. Although the differences between the score on all nodules and the malignant nodules specifically for all group-convolution based models are considerably larger than the differences between the baseline models, it is not apparent whether this is an overall trend or a coincidence.

Top percentage of candidates

The original dataset contained 8528 candidates in total. Rather than considering this dataset in its entirety, we can choose to focus on the top percentage (w.r.t. probability estimation) of candidates and investigate how many of the nodules (1141 in total) and specifically malignant nodules (129 in total) in the original dataset are present amongst these top predictions. For this purpose, we consider the top 15% of candidates based on

	Z_3	C_{4h}	D_{4h}	O	O_h
total	166	**1141**	**1141**	163	**1141**
malignant	22	**129**	**129**	27	**129**

Table 6.5: *Number of found nodules and malignant nodules in the top 15% of candidates for the various models trained on 30.000 training set samples (max. 1141 nodules and 129 malignant nodules).*

their estimated probability, which amounts to 1287 candidates out of the 8582 in total.

As Table 6.5 makes apparent, a significantly larger number of nodules are among the top 15% of candidates for the majority of the group-convolutional models. Of the 1287 candidates considered for this metric, C_{4h}, D_{4h} and O_h all contained all 1141 nodules (including the 129 malignant nodules), indicating a 100% recall and 88.6% precision – considerably higher than the result for the baseline. However, interesting is that the G-CNN with group O seems to perform similarly to the baseline.

Top probability nodules

However, with the previous metrics regarding malignancy, systems that generally performed better on all nodules could be argued to have an advantage over lesser systems when evaluating their performance on malignant nodules. For example, the top 15% predictions on a better performing system has a higher chance of containing more nodules, and therefore also a higher chance of containing malignant nodules. Therefore, in this subsection, we consider a set number of true positives, and evaluate what percentage of these true positives is not only a nodule, but also malignant. Table 6.6 provides an overview of the number of malignant nodules in the x true positives that received the highest estimated probability for each model, where $x \in \{100, 150, 250\}$.

For example, out of the 100 true positive nodules that were deemed most likely to be nodules by the baseline model trained on 30.000 data samples, 13 were malignant. Similarly, whereas there were 14 malignant nodules in the top 250 nodules for Z_3 trained on 3.000 data samples, there were more malignant nodules in the top 100 for any of the

g-convolutional models trained on the same dataset.

		Z_3	C_{4h}	D_{4h}	O	O_h
	100	5	21	25	20	**37**
3.000	150	*6*	37	40	25	**45**
	250	*14*	59	59	51	**62**
	100	*13*	31	**45**	44	23
30.000	150	*18*	49	60	**61**	38
	250	*31*	61	**77**	**77**	63

Table 6.6: *Number of malignant nodules from the x true positives that received the highest probability estimation.*

The group-convolution based models are considerably more sensitive with respect to malignant nodules in their top rated true positives. The number of malignant nodules in the 250 top probability true positives as judged by the baseline system, are equal to or less than the number of malignant nodules in the top 100 for the group-convolutional systems. This seems to indicate that the trained group-convolutional neural networks do consider highly suspicious nodules more 'nodule-like' than less suspicious nodules.

7

Conclusion

The objective of this work was to aid the development of 3D image analysis software in the medical field, where annotated data is scarce, by creating data efficient neural networks that exploit symmetries and therefore require less data to learn. This was achieved by extending the existing `GrouPy` package originally developed by Cohen & Welling (2016[27]) that implements g-convolutions, a drop-in replacement for regular convolutions, to be applicable to volumetric image data. The convolutional neural networks with 3D g-convolutions were applied to the false positive reduction step of lung nodule detection in particular.

Conclusion

The conclusions that can be drawn from the experiments regarding G-CNNs for false positive reduction of nodule detection are three-fold: firstly, G-CNNs perform better on the given datasets than CNNs with regular convolutions; secondly, the difference in scores between G-CNNs for various groups and the baseline increases as the dataset size decreases; and thirdly, G-CNNs seem to require less epochs to converge than regular CNNs. In the next paragraphs, these conclusions will be discussed in-depth.

First, regarding the performance of the baseline model and the g-convolutional variant, results show that the baseline trained on any training dataset size is outperformed by G-CNNs trained on that dataset size for all groups in every instance. This indicates that on the given dataset of pulmonary nodule candidates, convolutional neural networks with g-convolutions are the better choice. Moreover, an interesting observation is that G-CNNs seem intrinsically more sensitive towards malignant nodules than regular CNNs, deeming nodules that were considered very suspicious by expert radiologists more ”nodule-like” than less suspicious nodules. Although this is arguably a desirable property for a nodule detection system, it was not an objective that was specifically trained on. An explanation for this phenomenon could be that G-CNNs are more receptive towards the visual characteristics commonly associated with malignant nodules (*e.g.* regarding lobulation, spiculation or composition), leading to a higher probability for these nodules when using networks that exploit symmetries.

Secondly, the aim was to create a more data efficient network with a lower sample complexity. By training both the baseline as well as the G-CNNs on various dataset sizes, the difference in performance was shown to increase as the dataset size decreased, indicating that G-CNNs are in fact more equipped to efficiently handle the relatively little data they are provided with. Additionally, in a number of cases, the performance of the baseline was comparable with or even better than the performance of a G-CNN trained on a fraction of the data. For every dataset size, the best G-CNNs perform similarly to the baseline trained on ten times the amount of data, further emphasising that CNNs with

g-convolutions indeed have a far lower sample complexity and are more efficient with the provided data than CNNs with regular convolutions.

Lastly, regarding convergence, plotting the validation loss against the number of epochs trained show that 3D G-CNNs typically experience a fast initial decline towards the eventual loss, leading G-CNNs to converge in less epochs than ordinary CNNs. This faster convergence can be explained in that each parameter receives a gradient signal from multiple 3D feature maps at once. However, it should be noted that the advantage of faster convergence can be counteracted by the fact that the processing of each epoch does take a longer amount of time for a G-CNN than for a typical CNNs, where the processing time required increases with the order of the group.

All in all, analysis of the results of the experiments regarding G-CNNs and pulmonary nodule detection support the idea that G-CNNs are in fact more data efficient than regular CNNs, in that they perform better (especially with smaller dataset sizes) and converge faster. However, for the time being, the bottleneck of group-convolutional neural networks will most likely be the GPU memory requirements. A more extensive Residual Network architecture was experimented with, but limits to the GPU resources prevented the optimal model size from being reached. Fortunately, optimisations to the code or multi-GPU training where smaller batches are parallelised across GPUs could resolve these issues.

To summarise, initial results for G-CNNs on false positive reduction for nodule detection are decidedly promising and suggest g-convolutions are valuable for tasks in the biomedical domain, although they may not immediately be applicable due to memory restrictions. The scope of this project was to implement three-dimensional g-convolutions such that they can be used for volumetric image data and to analyse whether pulmonary nodule detection benefitted from networks with these g-convolutions, which was overwhelmingly demonstrated to be the case.

Future Work

While G-CNNs trained on our datasets have been demonstrated to perform better and converge faster, the intuition was also that if G-CNNs proved to be beneficial, that the performance increase would scale with the order of the group – after all, if including a group of symmetries of some order increased performance, including a group of symmetries of a larger order could hypothetically lead to an even larger performance increase. Interestingly enough, the results do not necessarily seem to support this notion. Variation in scores for the G-CNNs on most dataset sizes are negligible and can be attributed to different weight initialisations, data augmentations, or other small variations within the training process. Remarkably though, there was a notable difference in performance between the groups of cubic symmetry (O, O_h) and rectangular cuboid symmetry (C_{4h}, D_{4h}) for the really small dataset size of 30 samples. This effect was also observed for sample size 300 for cubic patches (see appendix I) and cannot immediately be explained with the knowledge at hand. More research into the effects and differences between the various types of symmetry groups used for the g-convolutions would be an interesting future project.

Another interesting follow-up would be to investigate what the effect would be of combining g-convolutions of various groups within a single network, or combining g-convolutions in a network with regular convolutions. So far, all convolutions within a g-convolutional network have been with the same input and output group, *e.g.* a G-CNN consisting entirely of g-convolutions with symmetry group C_{4h}. However, one might for example create a G-CNN with group convolutions in the lower layers, and regular convolutions in a higher layer, in order to only detect symmetry at small scales.

For pulmonary nodules in particular, an interesting next step would be apply G-CNNs to the problem of nodule composition classification, as composition is relevant in clinical practice for appropriate follow-up management and therefore an interesting aspect to classify correctly. Unfortunately, very little research into nodule texture classification has been published so far and the available datasets are limited. Nodule composition is

provided with the LIDC-IDRI dataset, where all four radiologists reported their estimation of nodule composition on a 1-5 scale (where 1 corresponds to ground-glass lesion, and 5 to solid). Properly aggregating these various radiologist ratings on a five-point scale into three separate classes presents a challenge on its own, but regardless of the aggregation method, the resulting dataset will have an enormous class imbalance. The majority of nodules are solid and only a small fraction are part-solid or ground-glass lesions, while those are far more likely to be malignant. Due to the reduced sample complexity associated with G-CNNs, these networks may also be more successful in dealing with class imbalance.

References

[1] H. Wang et al. *Global, regional, and national life expectancy, all-cause mortality, and cause-specific mortality for 249 causes of death, 1980–2015: a systematic analysis for the Global Burden of Disease Study 2015*. *Lancet*, 388:1459–1544, October 2016. doi:10.1016/S0140-6736(16)31012-1. PMID 27733281.

[2] Eurostat. *Health in the European Union – facts and figures*, September 2017. Data extracted in September 2017. Planned article update: January 2019. Available here.

[3] American Cancer Society Cancer Statistics Center. *Lung Cancer key statistics*. Last update: January 2017. Available here.

[4] World Health Organization. *World cancer report*, 2014. Available here.

[5] American Cancer Society. *Lung Cancer Detection and Early Prevention*. Last revised: February 22, 2016. Available here.

[6] A.J. Alberg; M.V. Brock; J.M. Samet. *Murray and Nadel's Textbook of Respiratory Medicine*. Saunders Elsevier, 6 edition, 2016.

[7] T. Blanchon; J-M Brechot; P.A. Grenier et al. *Baseline results of the Depiscan study: a French randomized pilot trial of lung cancer screening comparing low dose CT scan (LDCT) and chest X-ray (CXR)*. *Lung Cancer*, 58:50–58, 2007.

[8] P.B. Bach; J.R. Jett; U. Pastorino et al. *Computed tomography screening and lung cancer outcomes. JAMA*, pages 308–315, 2007.

[9] Z. Saghir; A. Dirksen; H. Ashraf et al. *CT screening for lung cancer brings forward early disease. The randomised Danish Lung Cancer Screening Trial: status after five annual screening rounds with low-dose CT. Thorax*, 67(4):296–301, 2012.

[10] U. Pastorino; M. Rossi; V. Rosato et al. *Annual or biennial CT screening versus observation in heavy smokers: 5-year results of the MILD trial. Eur J Cancer Prevent,* 21(3):308–315, 2012.

[11] J. Field; S. Duffy; D. Baldwin et al. *UK Lung Cancer RCT Pilot Screening Trial: baseline findings from the screening arm provide evidence for the potential implementation of lung cancer screening. Thorax,* 2015.

[12] Y. Ru Zhao; X. Xie; H.J. de Koning; W.P. Mali; R. Vliegenthart; M. Oudkerk. *NELSON Lung Cancer Screening Study. Cancer Imaging,* 2011.

[13] The National Lung Screening Trial Research Team. *Reduced Lung-Cancer Mortality with Low-Dose Computed Tomographic Screening. New England Journal of Medicine,* 365(5):395–409, 2011. PMID: 21714641.

[14] M. Oudkerk et al. *European position statement on lung cancer screening. The Lancet Oncology,* 18:754–766, November 2017.

[15] Samuel G. Armato et al. *The Lung Image Database Consortium (LIDC): an evaluation of radiologist variability in the identification of lung nodules on CT scans. Academic radiology,* 14(11):1409–21, November 2007.

[16] Paul F. Pinsky et al. *National lung screening trial: variability in nodule detection rates in chest CT studies. Radiology,* 268(3):865–73, September 2013.

[17] Sarah J. van Riel et al. *Observer Variability for Classification of Pulmonary Nodules on Low-Dose CT Images and Its Effect on Nodule Management. Radiology,* 277(3):863–871, December 2015.

[18] Geoffrey D. Rubin. *Lung nodule and cancer detection in computed tomography screening. Journal of thoracic imaging,* 30(2):130–138, March 2015.

[19] P.M. Lauritzen et al. *Radiologist-initiated double reading of abdominal CT: retrospective analysis of the clinical importance of changes to radiology reports. BMJ quality & safety,* 25(8):595–603, August 2016.

[20] D. Wormanns; K. Ludwig; F. Beyer; W. Heindel; S. Diederich.

[21] J.H. Sunshine M. Bhargavan; A.H. Kaye; H.P. Forman. *Workload of radiologists in United States in 2006-2007 and trends since 1991-1992. Radiology,* 252(2):458–467, August 2009.

[22] L. Bogoni; J.P. Ko; J. Alpert J et al. *Impact of a computer-aided detection (CAD) system integrated into a picture archiving and communication system (PACS) on reader sensitivity and efficiency for the detection of lung nodules in thoracic CT exams. J Digit Imaging*, 25(6), 2012.

[23] Y. Zhao et al. *Performance of computer-aided detection of pulmonary nodules in low-dose CT: comparison with double reading by nodule volume. European radiology*, 22(10):2076–84, October 2012.

[24] B. van Ginneken. *Fifty years of computer analysis in chest imaging: rule-based, machine learning, deep learning. Radiological Physics and Technology*, 10(2), February 2017.

[25] H. Greenspan; B. van Ginneken; R.M. Summers. *Guest Editorial Deep Learning in Medical Imaging: Overview and Future Promise of an Exciting New Technique. IEEE Transactions on Medical Imaging*, 35(5), May 2016.

[26] A.A.A. Setio et al. *Validation, comparison, and combination of algorithms for automatic detection of pulmonary nodules in computed tomography images: the LUNA16 challenge. CoRR*, abs/1612.08012, 2016.

[27] T.S. Cohen; M. Welling. *Group Equivariant Neural Networks.* In *International Conference on Machine Learning*, pages 2990–2999, February 2016.

[28] F. Rosenblatt. *The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. Cornell Aeronautical Laboratory, Psychological Review*, 65(6):386–408, 1958.

[29] G.E. Hintor; D.E. Rumelhart; R.J. Williams. *Learning representations by back-propagating errors. Nature*, pages 533–536, 1986.

[30] Y. Bengio; Y. LeCun. *Scaling Learning Algorithms towards AI.* Large-Scale Kernel Machines *by L. Bottou, O. Chapelle, D. DeCoste, J. Weston; MIT Press*, 2007.

[31] Y. Bengio; P. Lamblin; D. Popovici; H. Laorchelle. *Greedy Layer-Wise Training of Deep Networks. Neural Information Processing Systems Conference*, 2007.

[32] G. E. Hinton; S. Osindero; Y.W. Teh. *A fast learning algorithm for deep belief nets. Neural Computation*, 18:1527–1554, 2006.

[33] The Economist. *From not working to neural networking.* 2016.

[34] Wikipedia. *List of datasets for machine learning research.* Extracted December 2017 from wikipedia.

[35] N. Srivastava; G. Hinton; A. Krizhevsky; I. Sutskever; R. Salakhutdinov. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research*, 15:1929–1958, 2014.

[36] R. Hahnloser; R. Sarpeshkar; M.A. Mahowald; R.J. Douglas; H.S. Seung. *Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. Nature*, pages 947–951, 2000.

[37] Y. Jia; P. Sermanet; S. Reed; D. Anguelov; D. Erhan; V. Vanhoucke; A. Rabonivich P.C. Szegedy; W. Liu. *Going deeper with convolutions*. September 2014.

[38] F. Perronnin; J. Sanchez. *High-dimensional signature compression for large-scale image classification. Conference on Computer Vision and Pattern Recognition*, pages 1665–1672, 2011.

[39] A. Karpathy. *What I learned from competing against a ConvNet on ImageNet*. September 2014. Source: blogpost.

[40] A. Krizhevsky; I. Sutskever; G. Hinton. *ImageNet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems*, 2012.

[41] B. van Ginneken. *Fifty years of computer analysis in chest imaging: rule-based, machine learning, deep learning. Radiological Physics and Technology*, 10(1):23–32, February 2017.

[42] G.J. Litjens; T. Kooi; B. Ehteshami Bejnordi; A.A.A. Setio; F. Ciompi; M. Ghafoorian; J.A.W.M. van der Laak; B. van Ginneken; C.I. Sanchez. *A Survey on Deep Learning in Medical Image Analysis*. 2017.

[43] I. Sutskever; J. Martens; G. Dahl; G.E. Hinton. *On the Importance of Initialization and Momentum in Deep Learning*. In *Proceedings of the 30th International Conference on International Conference on Machine Learning*, volume 28 of *ICML'13*, pages III–1139–III–1147, 2013.

[44] John Duchi, Elad Hazan, and Yoram Singer. *Adaptive Subgradient Methods for Online Learning and Stochastic Optimization*. March 2010.

[45] M.D. Zeiler. *ADADELTA: An Adaptive Learning Rate Method*. 2012.

[46] G. Hinton; N. Srivastava; K. Swerksy. Lecture 6a: *Neural Networks for Machine Learning: Overview of mini-batch gradient descent.* 2012. Unofficial method proposed in Coursera class on Neural Networks for Machine Learning. Link to lecture notes.

[47] D.P. Kingma; J. Ba. *Adam: A Method for Stochastic Optimization. CoRR*, 2014.

[48] X. Glorot; Y. Bengio. *Understanding the difficulty of training deep feedforward neural networks. International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

[49] J.T. Springenberg; A. Sosovitskiy; T. Brox; M. Riedmiller. *Striving for Simplicity: The All Convolutional Net. International Conference on Learning Representations*, 2015.

[50] W. Shang; K. Sohn; D. Almeida; H. Lee. *Understanding and Improving Convolutional Neural Networks via Concatenated Rectified Linear Units.* 2016.

[51] S. Ioffe; C. Szegedy. *Batch normalization: Accelerating deep network training by reducing internal covariate shift. International Conference on Machine Learning*, 37, 2015.

[52] I. Goodfellow; Y. Bengio; A. Courville. *Deep Learning.* MIT Press, 2016. URL to book.

[53] L. Prechelt; G. B. Orr. Early stopping – but when? Neural Networks: Tricks of the Trade. *Lecture Notes in Computer Science,* Springer Berlin Heidelberg, pages 53–67, 2012.

[54] D.M. Hansell; A.A. Bankier; H. MacMahon; T.C. McLoud; N.L. Muller; J. Remy. *Fleischner Society: glossary of terms for thoracic imaging. Radiology*, pages 697–722, 2008.

[55] K. Alzahouri; M. Velten; P. Arveux; M.C. Woronoff-Lemsi; D. Jolly; F. Guillemin. *Management of SPN in France. Pathways for definitive diagnosis of solitary pulmonary nodule: a multicentre study in 18 French districts. BMC Cancer*, 2008.

[56] Radiology Assistant. *Solitary pulmonary nodule: benign vs. malignant.* Department of Radiology, Stanford University Medical Center, Stanford, California and the Department of Radiology, Rijnland Hospital, Leiderdorp, the Netherlands.

[57] C.I. Henschke; D.I. McCauley; D.F. Yankelevitz; D.P. Naidich; G. McGuinness; O.S. Miettinen; D.M. Libby; M.W. Pasmantier; J. Koizumi; N.K. Altorki; J.P. Smith. *Early Lung Cancer Action Project: overall design and findings from baseline screening. Lancet*, 354:99–105, 1999.

[58] M. Lederlin; M.P. Revel; A. Khalil; G. Ferretti; B. Milleron; F. Laurent. *Management strategy of pulmonary nodule in 2013. Diagnostic and Interventional Radiology*, 94(11):1081–1093, November 2013.

[59] D. M. Ha; P. J. Mazzone. *Pulmonary Nodules*. 2014.

[60] A. McWilliams; M.C. Tammemagi; J.R. Mayo; H. Roberts; G. Liu; K. Sohrati; K. Yasufuku; S. Martel; F. Laberge; M. Gingras; S. Atkar-Khattra; C.D. Berg; K. Evans; R. Finley; J. Yee; J. English; P. Nasute; J. Goffin; S. Puksa; L. Steward; S. Tsai; M.R. Johnston; D. Manos; G. Nicholas; G.D. Goss; J.M. Seely; K. Amjadi; A. Tremblay; P. Burrowesl P. Macheachern; R. Bhatia; M.S. Tsoa; S. Lam. *Probability of cancer in pulmonary nodules detected on first screening CT. The New England journal of medicine*, 369(10):910–919.

[61] M.E. Callister; D.R. Baldwin; A.R. Akram; S. Barnard; P. Cane; J. Draffan; K. Franks; F. Gleeson; R. Graham; P. Malhotra; M. Prokop; K. Rodger; M. Subesinghe; D. Waller; I. Woolhouse. *British Thoracic Society guidelines for the investigation and management of pulmonary nodules*. 2015. Thorax. 70 Suppl 2: ii1-ii54.

[62] S. J. Swensen *et al. . CT Screening for Lung Cancer: Five-year Prospective Experience. Radiology*, 235:259–265, 2005.

[63] The National Lung Screening Trial Research Team. *Reduced Lung-Cancer Mortality with Low-Dose Computed Tomographic Screening. New England Journal of Medicine*, August 2011.

[64] S.G. Armato; G. McLennan; M.F. McNitt-Gray; C.R. Meyer; D. Yankelevitz; D.R. Aberle; C.I. Henschke; E.A. Hoffman; E.A. Kazerooni; H. MacMahon; A.P. Reeves; B.Y. Croft; L.P. Clarke. *Lung Image Database Consortium Research Group. Lung image database consortium: developing a resource for the medical imaging research community. Radiology*, 232(3):739–748, September 2004.

[65] M.F. McNitt-Gray; S.G. Armato; C.R. Meyer; A.P. Reeves; G. McLennan; R.C. Pais; J. Freymann; M.S. Brown; R.M. Engelmann; P.H. Bland; G.E. Laderach; C. Piker; J. Guo; Z. Towfic; D.P.Y. Qing; D.F. Yankelevitz; D.R. Aberle; E.J.R. van

Beek; H. MacMahon; E.A. Kazerooni; B.Y. Croft; L.P. Clarke. *The Lung Image Database Consortium (LIDC) Data Collection Process for Nodule Detection and Annotation. Radiology*, 14(12):1464–1474, December 2007.

[66] S.B. Lo; S.A. Lou; J.S. Lin; M.T. Freedman; M.V. Chien; S.K. Mun. *Artificial convolution neural network techniques and applications for lung nodule detection.* 14(4):711–718, 1995.

[67] A. Hasegawa; M.T. Freedman; S.K. Mun J.S. Lin; S.B. Lo. *Reduction of false positives in lung nodule detection using a two-level neural classification.* 15(2):206–217, 1996.

[68] M. Firmino; A.H. Morais; R.M. Mendoca; M.R. Dantas; H.R. Hekis; R. Valentim. *Computer-aided detection system for lung cancer in computed tomography scans: review and future prospects. Biomed Eng Online*, 13(1), 2014.

[69] B. Al Mohammad; P.C. Brennan; C. Mello-Thoms. *A review of lung cancer screening and the role of computer-aided detection. Clinical Radiology*, 72(1):433–442, January 2017.

[70] Z. Lyu; Fonova. *3D Deep Convolution Neural Network Application in Lung Nodule Detection on CT images.* November 2017.

[71] F. Liao; M. Liang; Z. Li; X. Hu; S. Song. *Evaluate the Malignancy of Pulmonary Nodules Using the 3D Deep Leaky Noisy-or Network.* November 2017.

[72] W. Miller. *Symmetry Groups and Their Applications*, volume 50. Academic Press, 1 edition, 12 1972.

[73] P.Y. Simard; D. Steinkraus; J.C. Platt. *Best practices for convolutional neural networks applied to visual document analysis.*

[74] K. Lenc; A. Vedaldi. *Understanding image representations by measuring their equivariance and equivalence. In Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[75] B. Fasel; D. Gatica-Perez. *Rotation-invariant neoperceptron. International Conference on Pattern Recognition*, page 336–339, 2006.

[76] S. Dieleman; K. W. Willett; J. Dambre. *Rotation-invariant convolutional neural networks for galaxy morphology prediction. Monthly notices of the royal astronomical society*, 450(2):1441–1459, 2016.

[77] R. Gens; P.M. Domingos. *Deep Symmetry Networks.* In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2537–2545. Curran Associates, Inc., 2014.

[78] D. Marcos M. Volpi; D. Tuia. *Learning rotation invariant convolutional filters for texture classification.* April 2016.

[79] D. Marcos; M. Volpi; N. Komodakis; D. Tuia. *Rotation equivariant vector field networks.* December 2016.

[80] S. Dieleman; J. D. Fauw; K. Kavukcuoglu. *Exploiting cyclic symmetry in convolutional neural networks. International Conference on Machine Learning*, page 1889–1898, June 2016.

[81] Y. Zhou; Q. Ye; Q; Qui; J. Jiao. *Oriented Response Networks. Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[82] A. Cayley. *Desiderata and suggestions: No. 2. The Theory of groups: graphical representation. American Journal of Mathematics*, 1(2):174–176, 1878.

[83] American Society of Clinical Oncology. *ASCO answers: Lung Cancer Fact Sheet.* Retrieved December 2017. Link.

[84] Merck Manual Professional Edition. *Lung Carcinoma: Tumors of the Lungs.* Retrieved December 2017. Link.

[85] CancerCare. *Lung Cancer 101: What is lung cancer?* Retrieved December 2017. Link.

[86] American Cancer Society. *Signs and Symptoms of Lung Cancer.* Retrieved December 2017. Link.

Listing of figures

List of Tables

Appendix

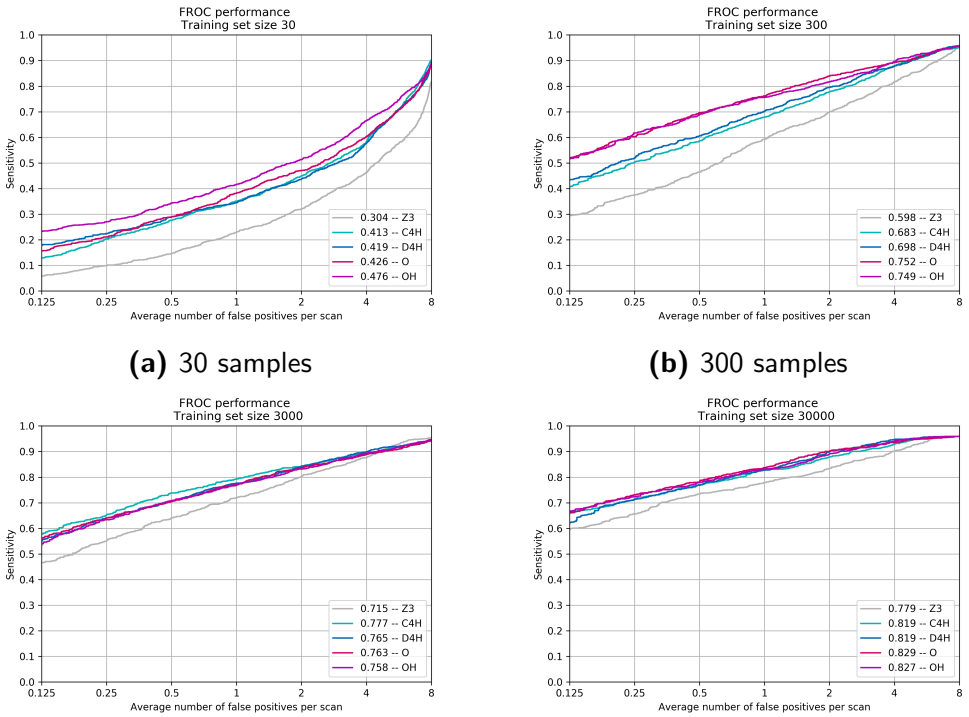
I Cubic patches

An alternative preprocessing method was applied to the scans to create cubic patches of equal dimensions in all axes (x, y and z), where each voxel represents $1 \times 1 \times 1$mm of lung tissue. The reasoning behind this alternative pre-processing method was to investigate whether the g-convolutions based on cubic symmetry groups performed better on cubic patches than on the original rectangular cuboid patches. For this experiment, patches were used of $15 \times 15 \times 15$ where each voxel represents $1 \times 1 \times 1$mm of lung tissue, instead of $12 \times 72 \times 72$ patches where each voxel represents $1.25 \times .5 \times .5$mm lung tissue. Figure I-1 and Table I-1 provide the results for these experiments. It should be noted that the overall system scores are lower as the patches are considerably smaller than those in the original experiments and therefore provide less information and context.

	Z_3	C_{4h}	D_{4h}	O	O_h
30	*0.304*	*0.413*	*0.419*	*0.426*	**0.476**
300	*0.598*	*0.683*	*0.698*	**0.752**	0.749
3000	*0.715*	**0.777**	*0.765*	*0.763*	0.758
30000	*0.779*	*0.819*	*0.819*	**0.829**	0.827

Table I-1: *Score for all training set size and group combinations trained on cubic patches.*

The overall system scores for the G-CNN seem to lie fairly close for most training set sizes, as was the case for the rectangular cuboid patches. Interestingly enough, there is a relatively large difference between the scores for the cubic O and O_h groups for 300



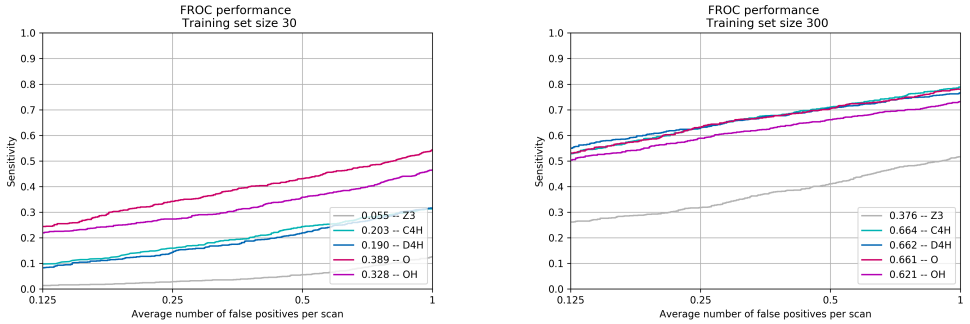
(a) 30 samples **(b)** 300 samples

(c) 3.000 samples **(d)** 30.000 samples

Figure I-1: *FROC curves for all groups per training set size trained on cubic patches.*

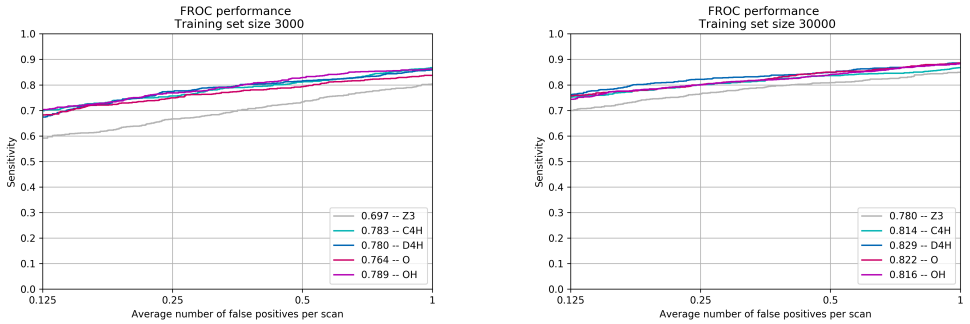
samples compared to the rectangular cuboid groups C_{4h} and D_{4h}. This was also observed for the rectangular cuboid patches on 30 training samples. However, it remains difficult to compare the effect of the two preprocessing approaches as the networks provided with the cubic patches were simply presented with less context and information than the rectangular cuboid patches. On first glance, however, there does not seem to be an obvious difference in relative results for G-CNNs trained on the two patch shapes.

As proposed by the LUNA16 challenge evaluation metric, the system scores and FROC curves are based on at most 8 false positive per scan on average. However, in clinical practice – the intended usecase for this type of nodule detection software – 8 false positives per scan on average would arguably be distracting and simply too many. Figure II-1 and Table II-1 outline the results of the training run on an adjusted average number of false positives per scan interval – at most 1. The scores are calculated by averaging the sensitivity at the following values: $\frac{1}{8}; \frac{1}{4}; \frac{1}{2}; 1$.



(a) 30 samples

(b) 300 samples



(c) 3.000 samples

(d) 30.000 samples

Figure II-1: *FROC curve results for all groups per training set size on the avg. false positive per scan interval* 0.125 – 1.

	Z_3	C_{4h}	D_{4h}	O	O_h
30	*0.055*	*0.203*	*0.190*	**0.389**	0.328
300	*0.376*	**0.664**	*0.662*	*0.661*	0.621
3000	*0.697*	*0.783*	*0.780*	*0.764*	**0.789**
30000	*0.780*	*0.814*	**0.829**	*0.822*	0.816

Table II-1: *Score for all training set size and group combinations on the avg. false positives per scan interval* $0.125 - 1$.

III Results individual training runs

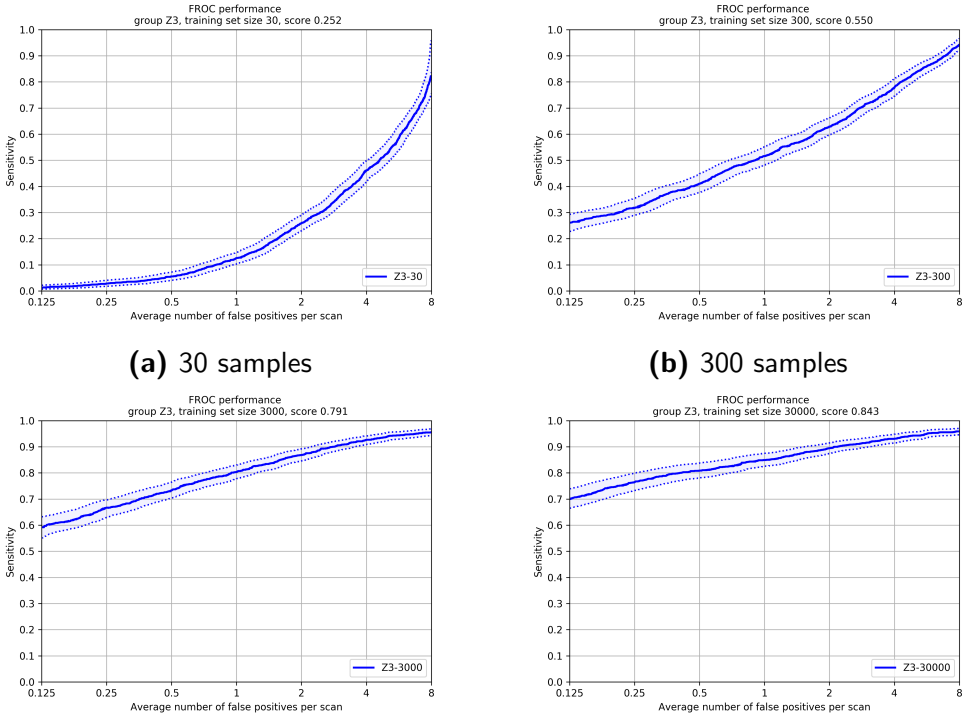
This section contains the results of the individual training runs with a a 95% confidence interval for each training run, as well as a table containing the intermediate points on the curve with .125, .25, .5, 1, 2, 4 and 8 false positives per scan on average respectively and the combined score. These results were omitted in Ch. 6 as the comparison between the various networks per training set size was deemed most relevant.

Group Z_3

Figure III-1 and Table III-1 contain results that concern the original baseline version of the network with regular 3D convolutions.

	.125	.25	.5	1	2	4	8	avg.
30	0.013	0.028	0.055	0.125	0.259	0.460	0.822	*0.252*
300	0.261	0.317	0.410	0.515	0.628	0.777	0.942	*0.550*
3.000	0.588	0.666	0.732	0.803	0.868	0.926	0.955	*0.791*
30.000	0.700	0.763	0.808	0.848	0.895	0.930	0.959	*0.843*

Table III-1: *Score for various training set sizes for baseline*



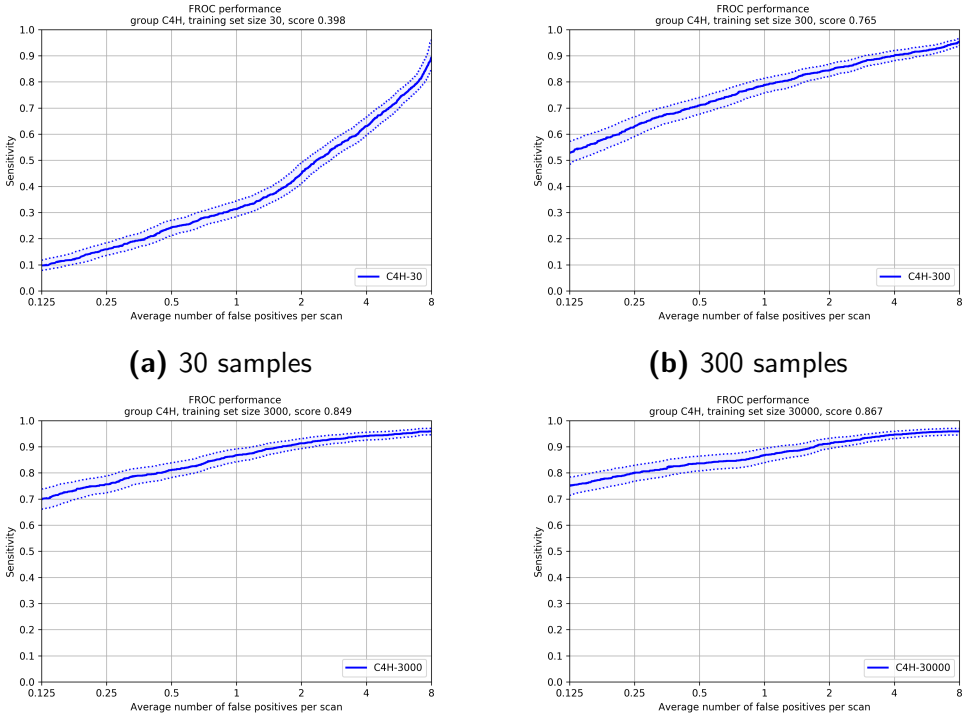
(a) 30 samples **(b)** 300 samples

(c) 3.000 samples **(d)** 30.000 samples

Figure III-1: *Group Z3*

Group C_{4h}

Group C_{4h} is the group of the rectangular cuboid, with rotations and without reflections. Figure III-2 and Table III-2 outline the results with respect to the training runs that use g-convolutions with this group.



(a) 30 samples

(b) 300 samples

(c) 3.000 samples

(d) 30.000 samples

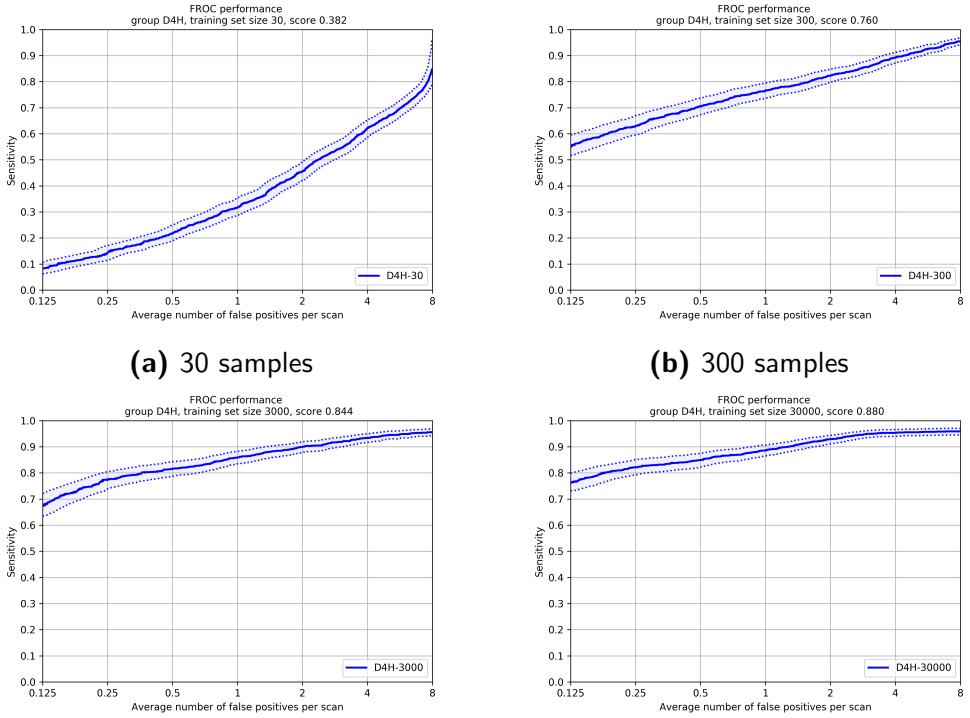
Figure III-2: *Group C_{4h}*

	.125	.25	.5	1	2	4	8	avg.
30	0.098	0.159	0.244	0.313	0.449	0.631	0.893	*0.398*
300	0.529	0.628	0.710	0.788	0.844	0.901	0.954	*0.765*
3000	0.697	0.755	0.810	0.868	0.912	0.940	0.959	*0.849*
30000	0.751	0.800	0.836	0.868	0.911	0.946	0.959	*0.867*

Table III-2: *Score for various training set sizes for group C_{4h}*

Group D_{4h}

D_{4h} is the group of rectangular cuboid symmetry, though different as it considers reflections. Figure III-3 displays the FROC performance of the g-convolutional model with group D_{4h} as its basis, and Table III-3 shows the scores on a logarithmic interval.



(a) 30 samples **(b)** 300 samples

(c) 3.000 samples **(d)** 30.000 samples

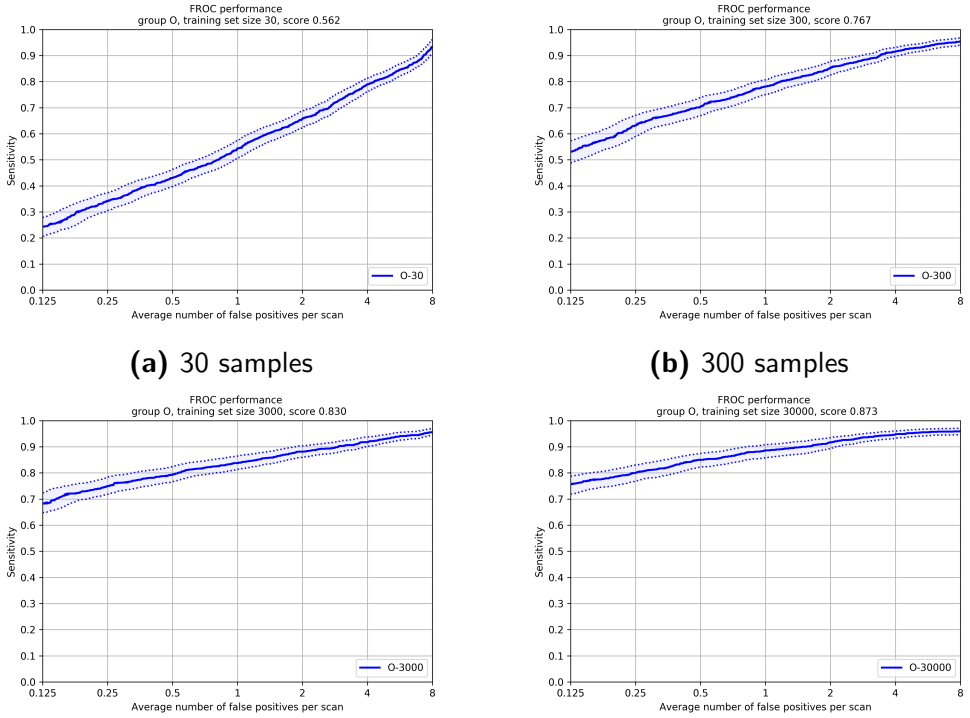
Figure III-3: *Group D_{4h}*

	.125	.25	.5	1	2	4	8	avg.
30	0.083	0.142	0.218	0.316	0.455	0.621	0.840	*0.382*
300	0.550	0.627	0.705	0.766	0.822	0.892	0.954	*0.759*
3000	0.673	0.774	0.815	0.858	0.899	0.933	0.955	*0.844*
30000	0.762	0.821	0.848	0.885	0.928	0.953	0.959	*0.880*

Table III-3: *Score for various training set sizes for group D_{4h}*

Group O

Group O is the group of cubic symmetry with rotations of various degrees over three different fold axes. Figure III-4 and Table III-4 contain the results for the g-convolutional variant of the baseline with this group.



(a) 30 samples

(b) 300 samples

(c) 3.000 samples

(d) 30.000 samples

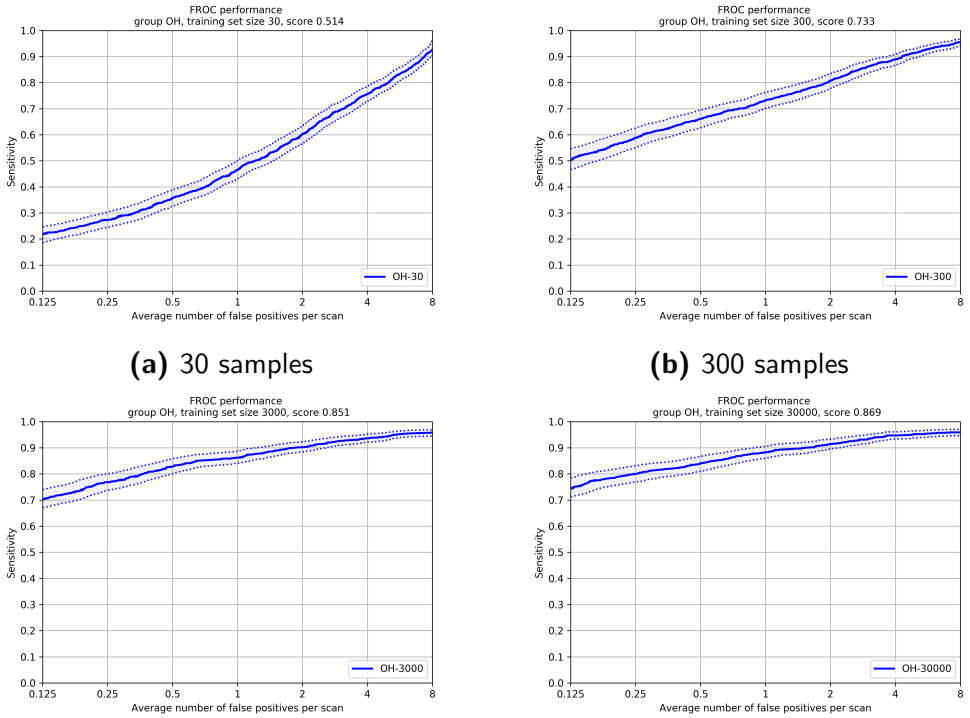
Figure III-4: *Group O*

	.125	.25	.5	1	2	4	8	avg.
30	0.242	0.341	0.430	0.543	0.657	0.788	0.933	*0.562*
300	0.530	0.632	0.703	0.781	0.852	0.915	0.954	*0.767*
3000	0.679	0.749	0.793	0.837	0.881	0.917	0.956	*0.830*
30000	0.755	0.799	0.849	0.885	0.917	0.946	0.959	*0.873*

Table III-4: *Score for various training set sizes for group O*

Group O_h

Group O_h is the group with the largest number of elements of all the groups – 48 in total. This group was also experimented with on the various training set sizes and the result of the model on the test set are presented in Figure III-5 and Table III-5.



(a) 30 samples **(b)** 300 samples

(c) 3.000 samples **(d)** 30.000 samples

Figure III-5: *Group O_h*

	.125	.25	.5	1	2	4	8	avg.
30	0.218	0.273	0.358	0.464	0.602	0.755	0.925	*0.514*
300	0.503	0.588	0.660	0.732	0.808	0.889	0.954	*0.733*
3000	0.702	0.766	0.828	0.862	0.901	0.936	0.958	*0.850*
30000	0.743	0.799	0.840	0.882	0.913	0.947	0.959	*0.869*

Table III-5: *Score for various training set sizes for group O_h*

IV Lung anatomy & cancer development

The lungs are primary organs of the human respiratory system. As air enters the body, it is brought via the *trachea* (windpipe) to the *bronchi* which branches the air into the two lungs on either side of the body. The lungs themselves are composed of small tube-like structures called *bronchioles* that lead the air from the bronchus to tiny elastic sacs called *alveoli*. It is in the alveoli that oxygen and carbon dioxide are transferred between the bloodstream and the air.

Lungs are composed of a variety of types of cells, such as blood, nerve and hormone-producing cells, but most notable due to their numbers are the *epithelial* cells – a type of cell characterised by their cohesive structure that line the surfaces of the body.[83] It is when these epithelial cells no longer abide by their normal growth and death rates, due to a mutation, that tumours in the lung develop. The type of cancer that develops in epithelial cells, in the lungs or otherwise, is known as a *carcinoma*.[84]

A tumour is an uncontrolled growth of abnormal cells that serves no direct purpose, and is considered to be either *benign* or *malignant*.[85] Benign tumours typically have a slower growth rate than their malignant counterparts, and have deviated less from the characteristics of their original cells.(1) Most importantly, though, they lack the ability to invade nearby tissue. Malignant tumours, on the other hand, *can* invade the surrounding tissue and cells are able to spread towards other organs via the lymph nodes or the blood stream. This spread to other body parts is referred to as *metastatis*. A distinction is made between abnormal cell growth that initially starts in the lung (and may metastasize to other places), known as *primary* lung cancer, and cancerous cells that have travelled to the lung to form tumours from other locations, which is *secondary* lung cancer.[85] Lung screening is aimed to detect primary lung cancer in particular.

Symptoms of both primary and secondary lung cancer may include coughing up blood, pain in the chest area or wheezing, and other more generic signs of illness such as weakness, fatigue and weight loss[86], but unfortunately typically do not present themselves until the cancer is in a late stage.(2) However, even when no symptoms have presented themselves as of yet, medical images of the chest may visually reveal abnormal cell growth in the lung in the form of *pulmonary nodules*.

(1) Benign tumours may, however, eventually progress towards malignancy.

(2) Cancer staging is a manner to describe the spread of the cancer, on a 1 to 4 scale. Stage I indicates the tumours are still small and all abnormal cell growth is still contained within the original organ, while stage IV indicates the cancerous cells have already spread to other body parts to form tumours.

V Matrix patterns of rectangular cuboid symmetry

The generators for C_{4h} are the 180^o rotation over y and the 90^o rotation over z. Figure V-1 outlines the matrix patterns that all transformations related to this symmetry have, where the first matrix either has zero or two negative 1's and the second matrix either has one or three negative 1's. The additional eight elements for group D_{4h} are acquired by inverting the matrices.

$$\begin{bmatrix} 1 & \cdot & \cdot \\ \cdot & 1 & \cdot \\ \cdot & \cdot & 1 \end{bmatrix} \begin{bmatrix} \cdot & 1 & \cdot \\ 1 & \cdot & \cdot \\ \cdot & \cdot & 1 \end{bmatrix}$$

Figure V-1: *Patterns of matrix representation for rectangular cuboid symmetry.*

VI Matrix patterns of octahedral symmetry

The elements of group O and O_h can be represented as 3×3 matrix transformations, generated by a 90^o rotation over the y and z axis. All elements in the groups of octahedral are represented by matrices that have their positive or negative 1's in the positions following the patterns from Figure VI-1. For group O, of 24 elements, either *one* or *three* of the 1's in these matrices was *positive*. For the additional 24 elements for group O_h, either *one* or *three* of the 1's in these matrices was *negative*.

$$\begin{bmatrix} 1 & \cdot & \cdot \\ \cdot & 1 & \cdot \\ \cdot & \cdot & 1 \end{bmatrix} \begin{bmatrix} \cdot & 1 & \cdot \\ \cdot & \cdot & 1 \\ 1 & \cdot & \cdot \end{bmatrix} \begin{bmatrix} \cdot & \cdot & 1 \\ 1 & \cdot & \cdot \\ \cdot & 1 & \cdot \end{bmatrix}$$
$$\begin{bmatrix} \cdot & \cdot & 1 \\ \cdot & 1 & \cdot \\ 1 & \cdot & \cdot \end{bmatrix} \begin{bmatrix} \cdot & 1 & \cdot \\ 1 & \cdot & \cdot \\ \cdot & \cdot & 1 \end{bmatrix} \begin{bmatrix} 1 & \cdot & \cdot \\ \cdot & \cdot & 1 \\ \cdot & 1 & \cdot \end{bmatrix}$$

Figure VI-1: *Patterns of matrix representation for octahedral symmetry.*