Deep transfer learning in medical imaging

Western Norway
University of
Applied Sciences

Satheshkumar Kaliyugarsan

Department of Informatics, University of Bergen

Department of Computing, Mathematics and Physics, Western Norway University of Applied Sciences

This thesis is submitted for the degree of
Master's in Software Engineering

June 2019

Acknowledgements

I would like to express my greatest appreciation to my supervisor Dr. Alexander Selvikvåg Lundervold for introducing me to this field and for giving me this golden opportunity to work with this project. I would specially thank him for all the encouragement and guidance throughout this work.
I would also thank my co-students Sindre Eik de Lange and Stian Amland Heilund for all the "deep learning" discussions and the fun we had during this period.
Thanks to the Mohn Medical Imaging and Visualization Centre where the experiments and studies were performed. Especially thanks to Dr. Eli Renate Grüner for the arrangement of the lab facilities.
Finally, I would like to thank my family and friends for their endless support and trust on me throughout this work.

Abstract

A common stumbling block for supervised learning methods based on deep convolutional neural networks (CNNs) is the large number of labeled examples required for training. To alleviate this problem, almost all two-dimensional CNNs designed for tasks involving images initialize their weights from another network trained on a different task for which there is ample data. Typically, the ImageNet challenge dataset is used, a collection of approximately 1.2 million labeled images from 1000 different categories.

The effect of transferring weight between medical imaging tasks is however less well-studied. As creating labeled data for medical images is often a time-consuming, difficult and unreliable process, the amount of training data available is in general very small. This makes successful transfer learning a highly valued prospect.

In this work, we investigate the effect of transferring weights from convolutional neural networks trained to perform medical imaging tasks on large amounts of data, to networks created to solve our target problems.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

The human brain has an impressive ability to acquire knowledge and adapt to changing enviroments. Humans can learn in many different ways, e.g., through pattern recognition, trial and error, demonstration, intuition and introspection. In artificial intelligence (AI), some of these learning approaches are implemented using mathematical functions, making "it possible for machines to learn from experience, adjust to new inputs and perform human-like task" [51]. Even though AI is far behind natural intelligence, it is not a futuristic fiction found only in science fiction books and movies: AI is already having a huge impact on us.

Governments [39, 80], companies [4] and universities [54] are investing vast resources on AI. For instance, last year MIT announced a $1 billion plan for a new college that combines AI, machine learning (ML), and data science with other academic disciplines[54]. Another example is the government in Germany who has planned to spend 3 billion euros within 2025 in AI research and development to close the knowledge gap between them and world leading nations in this field, like China and the United States [39]. The 2018 AI Index report published by experts and researchers from Stanford and other top institutions shows that this field has grown tremendously in recent years, and also that it is expected to continue to grow. For instance, the growth rate of published AI papers on Scopus has increased by more than 8x from 1997 to 2017 (seen in figure 1.1).

AI is one of the most popular technological terms of our time, but it is actually not a new concept. In fact, the term "AI" was invented back in 1955 by John McCarthy in his proposal for a conference on this subject [64].

According to John McCarthy, it is "the science and engineering of making intelligent machines" [98]. Many scientists have been contributing to this field since then, but why is AI so popular today, when it has been around for decades?

In 2006, Geoffrey E. Hinton et al. published a paper [43] demonstrating how to train a deep neural network to recognize handwritten numbers with an error rate of only 1.25.

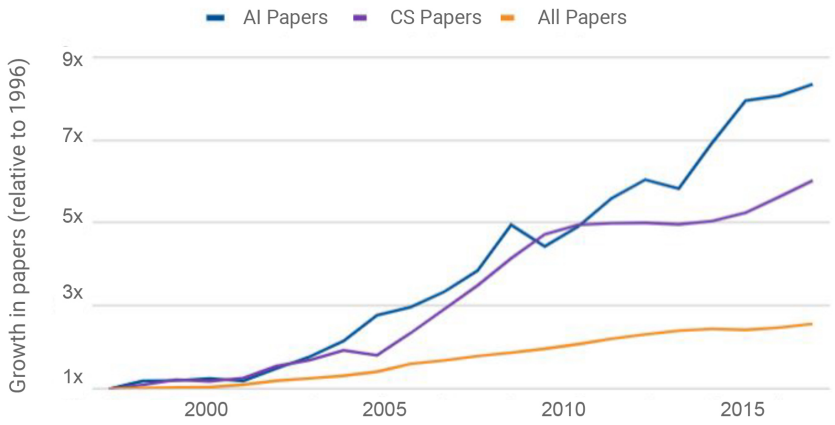


Fig. 1.1 The graph compares the growth rate of published AI papers from 1997 to 2017 with computer science and all other fields.(Source: [85])

The accuracy in itself was not that impressive, since other algorithms, like support vector machines (SVM) had already gotten an error rate of 1.4. The big deal was that up until this point it was considered an impossible task to train deep neural networks to perform useful tasks, and most researchers had left this idea. The study of multi-layer artificial neural networks is today known as "deep learning".

"Deep learning has seen a dramatic explosion in the past 6 years, largely driven by increase in computational power and the availability of massive new datasets"[26]. Deep learning has shown to be applicable to tackle highly complex tasks that no other machine learning (ML) algorithms can compare to, such as translating from one language to another with human-level performance [6], or beating world's best players in games like Go [88] and more recently StarCraft II [5].

Healthcare stands to benefit enormously from recent improvements in deep learning. However, the medical community has only recently begun to realize what can be achieved with this technology.

Several applications have already shown some impressive results in a wide range of medical tasks, including occlusion detection in stroke imaging [10], classification of abnormalities on chest radiographs [50, 78, 99], and melanoma recognition in dermoscopy images [25, 37, 105]. A survey conducted by Geert Litjens et al. in 2017 provides an overview of papers using deep learning methods for different medical imaging tasks [61]. Lundervold [62] gives an updated overview, focusing on applications related to magnetic resonance imaging (MRI).

Such applications can be used to mitigate the workload of radiologists. An article published by the Norwegian newspaper Bergens Tidende in 2016 pointed out that the Radiology Department at Haukeland University Hospital had more than 7000 examinations queued due to shortage of radiologists [74].

The situation only gets worse as the workload of the Radiology Department continues to increase each year (see figure 1.2).

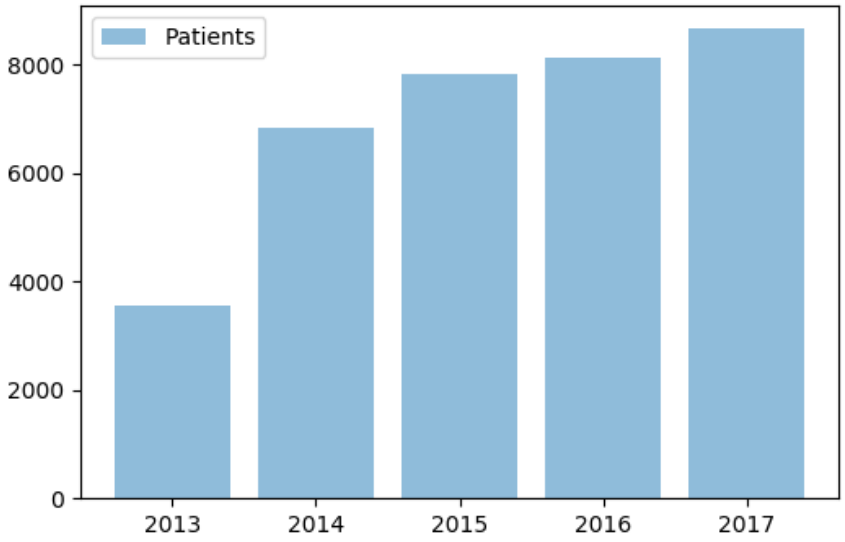


Fig. 1.2 The growth rate of breast examination from 2013 to 2017 at the Radiology Department at Haukeland University Hospital. (Data source: [74])

Shortage of radiologists has led them to work more overtime to get their job done. This situation is particularly worrisome considering that long workdays reduces radiologists diagnostic accuracy [56, 57].

Short supply of radiologists is a problem we see in all other developed countries as well. This problem underlines the need for new technologies like deep learning, to reduce the heavy workload of clinicians [94].

Implementing deep learning algorithms in hospitals can improve patient care, reduce the workload of clinicians, and assist them in making better decisions. In order to develop deep learning algorithms that works well, large amount of quality data are required. To give you an idea, a group of researchers at Stanford who developed an algorithm to detect pneumonia on frontal chest radiographs had access to 112.120 X-ray images, each annotated with up to 14 different diseases [78].

A common problem when applying deep learning methods in medical imaging is that you often do not have access to such an amount of labeled data [26], as creating labeled data for medical images is often challenging.

One way to mitigate this problem is to apply a technique called transfer learning [104]. It is a very natural idea, informed by our own approach to learning: People often learn something in one context which informs how well they learn and perform in another context. For instance, a programmer who knows Java can apply that knowledge in learning C# faster.

Nowadays, almost all two-dimensional convolutional neural networks (CNN) designed for tasks involving images utilize this idea: knowledge is transferred from a network trained on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) dataset [83], containing more than 1.2 million labeled images from 1000 different categories.

The effect of transferring knowledge from other tasks is however less well-studied. The idea investigated in this work is the effect of transferring knowledge from CNNs trained to perform medical imaging tasks, where there is an ample supply of training data, to a network created to solve our target problems.

The key contributions of this thesis are:

- A new transfer learning approach for 2D medical images, involving indirect transfer from ImageNet to the target network by first passing through a network trained on data more similar to the target dataset.
- Analysis about the impact of the data set size and the value of the layers in a CNN when using transfer learning in volumetric medical images.

This thesis will be split into two parts. Part 1, *Background*, will give the reader an overview of the fundamental theory behind transfer learning in computer vision. Chapter 2 introduces the basics of machine learning theory. The following chapter 3 will present the theory behind neural networks and how they are utilized in computer vision. Chapter 4 presents the technical details around transfer learning and the advantages of using this approach in computer vision tasks.

In part 2, *Experiments*, we will present our experiments in transfer learning for medical image analysis. In chapter 5, we describe an approach to transfer learning in 2D medical images, discuss the methods and materials used, and explain the results obtained. In chapter 6, we will look at the value of transfer learning in 3D medical images by presenting two different experiments. Similar to chapter 5, we will discuss the methods and materials and present the findings at the end of the chapter. Chapter 7 summarizes our results and addresses potential future work.

All the figures in this thesis have been created using yEd [106] and Matplotlib [48].

Part I

Background

Chapter 2

Introduction to machine learning

In this machine learning (ML) introduction, we will give you an overview of some basic concepts and techniques in ML. It is essential to understand the fundamental concepts introduced in this chapter before continuing to the rest of the thesis.

2.1 What is machine learning?

Machine learning (ML) is a sub-field of AI, based on computer science, mathematics, and statistics. In 1959, Arthur Samuel, one of the pioneers in this field, defined machine learning as, "A Field of study that gives computers the ability to learn without being explicitly programmed"[31]. This definition is slightly vague. In 1997, Tom Mitchell provided a more operational and engineering-oriented definition: "A computer program is said to learn from experience 'E' with respect to some class of tasks 'T' and performance measure 'P' if its performance at tasks in 'T' as measured by 'P' improves with experience 'E'"[12].

Put simply, the idea of machine learning is to enable computer programs to learn from data and make decisions based on the knowledge they acquire.

To give you an idea of the concept, spam filters typically utilizes machine learning algorithms to distinguish between spam and non-spam emails (T). These algorithms learn from previous examples of data (E). The performance can be measured in different ways, reflecting users end goal. For instance, in this case, one can use the ratio of true results to say how well the program works (P). This performance metrics is called accuracy and is commonly used in classification problems. We will discuss performance metrics in more detail later in this chapter.

2.2 Traditional programming vs machine learning

In traditional programming, the data and set of rules are executed on the computer to create the result. While in machine learning, the data and the result is executed on the computer to create the program (see figure 2.1). This machine learning program can then be used to predict unknown data.

So, instead of creating rules for spam filtering yourself or use existing rules (e.g., white-listing or blacklisting), machine learning algorithms learn these rules all by themselves. You do not have to maintain large complex lists of rules as these algorithms can figure them out for you.

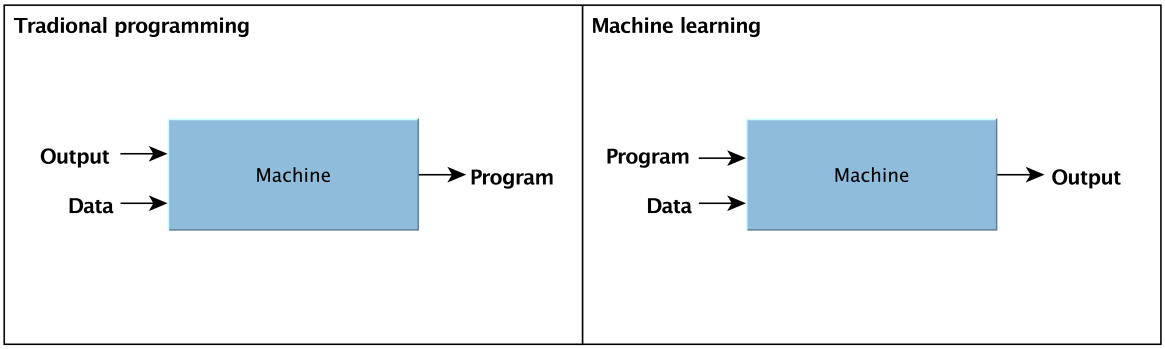


Fig. 2.1 Traditional programming versus machine learning.

2.3 Different types of machine learning

Machine learning approaches can roughly be divided into three categories: supervised learning, unsupervised learning and reinforcement learning. In addition, algorithms that can handle unlabeled and labeled data together are categorized as semi-supervised learning. These algorithms are usually a combination of supervised and unsupervised learning.

Supervised learning

In supervised learning, the algorithms learns from training data where the label is included, and the goal is to make predictions on future data based on these examples. This type of machine learning has had great success in recent years and is behind essentially all the economic value of AI [75]. The tasks within supervised learning are divided into classification and regression problems. In regression you want to predict real-values, called continous values. For example, the price of a house given the location, size, etc.

In classification you want to predict the category of the input data. For instance, whether an email is spam or not.

Common algorithms within supervised learning are k-nearest neighbor, logistic regression, linear regression, support vector machines, decision trees, and neural networks. Some of these algorithms are constrained to do either classification or regression, like linear regression. Others, like decision trees, can be used for both problems with minor modifications.

Unsupervised learning

In unsupervised learning, we want to find patterns and relationships from unlabeled data. Typical tasks within unsupervised learning are clustering, dimensionality reduction and association rule learning [31]. Common algorithms for these tasks are k-means, Kernel Principal Component Analysis (PCA) and the Apriori algorithm for association rule learning.

Unsupervised learning can actually be used as preprocessing for supervised learning. For instance, say you have to work with high dimensional data with lots of information, like the health status of patients. These types of data are often difficult to visualize, and algorithms are prone to overfit the data due to the high complexity. People with domain knowledge can reduce the complexity by doing feature engineering, where they create new features and remove unnecessary features manually, but this can be a time-consuming work.

An alternative for feature engineering is feature extraction, where new features are learned directly from the data. Kernel PCA is a well-known algorithm for feature extraction. It projects high dimensional data to lower dimensions without losing too much information. However, there is a risk that important feature are removed since it learns without labels. A main challenge with unsupervised learning is that there is no right or wrong answer. Therefore, it is difficult to evaluate the performance of the algorithm.

Reinforcement learning

This is probably what people think about when they hear the word AI. In real life, as well as in most video games, you get rewards and penalties based on your decisions. By cleverly using these feedback signals, one can learn how to improve ones performance.

Reinforcement learning works the same way. Here you have an agent that tries to maximize a reward by taking actions in an environment. Based on the effect of the actions, the agent learns what is wise to do. Many of the most high profile AI achievements in later years are based on reinforcement learning (typically combined with supervised learning).

For example, Google DeepMind's AlphaStar used this technique to beat human professionals in StarCraft II [5], and AlphaZero to beat world champions at Go and chess [87, 88].

2.4 Data: a limiting factor in machine learning

Machine learning algorithms performance are confined by the quality and quantity of the data. In addition to making sure the training data contains the signals necessary to produce useful predictions, you must always make sure that future data sufficiently is similar to the training data. It is necessary to use a training set that is representative for the future cases you want to generalize to. In general, in order for the machine learning algorithms to perform well, it is important that the provided data is of high quality. If the training data is full of errors and noise, then the insights from these data will be flawed.

According to a report published by CrowdFlower in 2016, data scientist spend most of their time on cleaning and organizing data [22]. "Data cleaning, is the process of detecting and correcting (or removing) corrupt or inaccurate records from a record set, table or database. The term refers to identifying incomplete, incorrect, inaccurate, irrelevant, etc. parts of the data an then replacing, modifying, or deleting this dirty data"[38].

For example, missing values must typically be handled in some way before it can be passed into a machine learning algorithm since most algorithms cannot deal with missing values. There are several methods for dealing with missing values, one simple way is to drop columns or instances with missing values. This is usually not the best solution as it might lead to information loss and significantly reduce the size of your data set. By carefully imputing, or replacing, missing values it is sometimes possible to deal with this issue without losing too much performance.

2.5 Understanding features

The success of machine learning models often depends on how you utilize the features and their types (e.g., continuous and categorical). This is part of what is called feature engineering, one of the most crucial parts of building machine learning systems. "Coming up with features is difficult, time-consuming, requires expert knowledge. 'Applied machine learning' is basically feature engineering"[68].

This process typically involves concepts such as feature extraction and feature selection:

- In feature selection, we select a subset of useful futures and ignore the irrelevant ones (e.g., constant and redundant features). This is actually a part of many learning algorithms, for example, decision trees do it implicitly.
- In feature extraction, new useful features are learned from the original data. The idea is to project existing features into a new feature space with lower dimensionality. Examples of feature extraction methods include Linear Discriminant Analysis (LCA), and Principal Component Analysis (PCA) (note that these methods rely on linearity assumptions).

2.6 Overfitting and underfitting

A complex model usually fits the training data better than a simple model. For example, increasing the depth of a decision tree will not decrease the accuracy in the training set. However, the best fitting model is not necessarily the final model you want to use. The end goal in machine learning is to find a model that generalizes well.

When a model performs well on training examples and poorly on new instances, we say that the model overfits. Overfitting refers to models that are too closely adapted to the training data and makes predictions on new data based on details and noise from the training examples. The opposite of overfitting, when the models are too simple to capture the underlying relationship in the training data, is called underfitting. These fitting issues are very common, and often the main reason for poor performance of machine learning models.

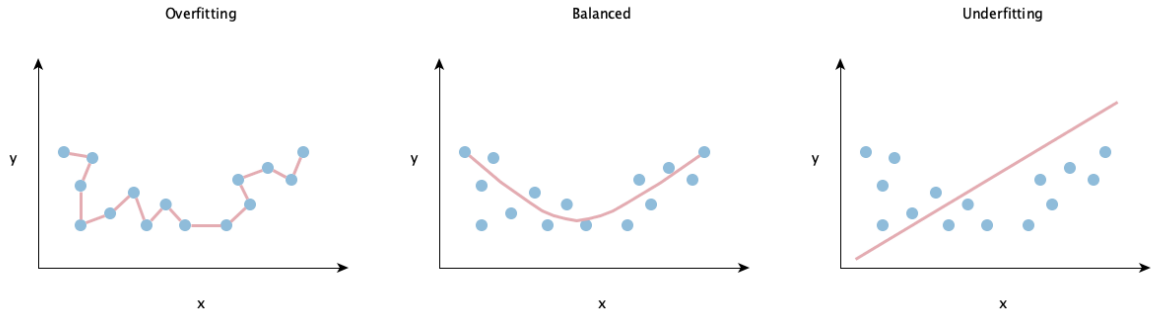


Fig. 2.2 An example of overfitting, good balance, and underfitting

2.7 Bias-variance tradeoff

The only way to find out how well your model generalizes to new data is by testing it on unseen data. A simple way to do this in practice is by splitting the dataset into two sets: training set and test set. Generally, people use 80 % of the data for training and the rest for testing (note that this split depends on the amount of data you have available). The training set is used to fit the model, and the test set is used to estimate the generalization error of the final model. An alternative approach is to use cross-validation, which uses multiple splits into training and test. This provides a more robust measure of generalization performance and is particualry useful when there is little data available. A model's generalization error can be expressed as the sum of three kinds of prediction errors [31]:

- Bias refers to the error due to modeling assumptions. A model with high-bias has a tendency to underfit the training data.
- Variance refers to the error due to variations in the training data. Models with high-variance tend to overfit the training data.
- Irreducible error refers to error due to noise in the data. This error can only be reduced by cleaning the dataset.

High variance models tend to be simple. Working on decreasing the variance will increase the bias, similarly, reducing the bias will increase the variance. Reducing just one of the error, will not improve the model, so you need to find the optimal balance between bias and variance that minimizes the generalization error. Hence the name tradeoff (see figure 2.3). An optimal balance between them would never overfit nor underfit.

How can we find the optimal balance between bias and variance?

In practice, it is not possible to calculate the exact bias and variance error. However, there is actually a common diagnosing method that can be used to determine whether a model is prone to high bias or high variance. This method involves comparing the training error and the generalization error estimate from the test set. When both of these errors are high, the model is susceptible to high bias and tends to underfit the data. If the training error is significantly lower than the generalization error, the model is prone to high variance and has most likely overfit the training data.

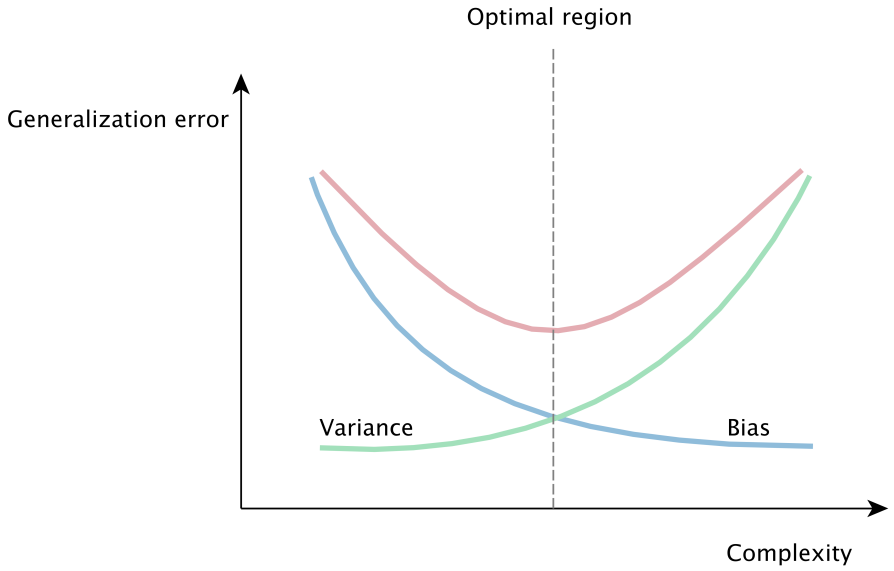


Fig. 2.3 Bias-variance tradeoff

How can we prevent these problems?

In case of overfitting, the ideal solution is to collect more training data. Unfortunately, in most cases this is not an option. Other alternatives are to use a simpler model or apply regularization to the model (L1, L2, dropout, etc). Regularization is a collection of techniques used to penalize complexity in a model (we will discus this further in chapter 3).

To prevent underfitting, the simplest solution is to train a more complex model. If this does not work, other options are to perform feature engineering (e.g., create new features and remove irrelevant features) or, if applicable, use less regularization.

A general practice when working with reducing overfitting and underfitting is to divide the data set into three subsets: training, validation and test set. The validation set is used to tune the hyperparameters (e.g., regularization) and select the "best" performing model for your problem. If the data set is too small, you can use other methods such as cross-validation or bootstrap.

Note that performance measures based on the validation set will be biased since the validation set is used to select the best performing model. Therefore, it can only be used to give a biased estimate of the generalization error. If you have several models that perform equally well, you should always select the simplest one.

Once you have found the model and hyperparameters to use for your task, a final model is trained on the entire training set (training and validation data) with these settings. This newly trained model is then evaluated on the test data to make sure that the model is neither overfitting nor underfitting.

You should never reuse the same test set if you are not satisfied with your final model and want to tune your model or swap it with another one. If you do this, it will no longer be unbiased, which might lead to poorer performance than expected once you deploy the new model into production. Your only option is to gather more data to create a new test set.

2.8 Model evaluation

The performance evaluation of models is an essential part of any machine learning project. After all, this is what you look at the end of the project to see if you have achieved the final goal. There are several evaluation measures that can be used to assess your models. The selection, however, is bound by the specific machine learning problems you are facing (e.g., classification, regression, clustering, etc.) [109]. In this section, we will focus on the ones used for classification tasks.

We have already mentioned the accuracy metric earlier in this chapter. This measure alone does not always give you a clear picture of the performance. For example, a well-known problem in medical datasets is that they often consist of a majority class and a minority class (e.g., more normal samples than abnormal events) [97]. Classifying everyone as the majority group will give you a high accuracy straight away, even though it is a bad classifier.

A better way to evaluate the performance of a classification problem is to use a confusion matrix. A confusion matrix is a table that shows the number of correct and incorrect classifications for each class. The rows in a confusion matrix correspond to the true class label, and the columns correspond to the predicted class label. For binary classification, there are four possible cases in a confusion matrix:

- True positives (TP): The actual class is positive, and the predicted class is positive
- False positives (FP): The actual class is positive, but the predicted class is negative
- True negatives (TN): The actual class is negative, and the predicted class is negative
- False negatives (FN): The actual class is negative, but the predicted class is positive

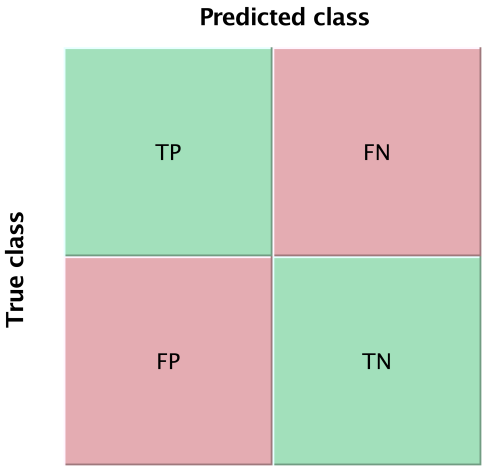


Fig. 2.4 Confusion matrix for binary classification.

These outcomes give us a better understanding of the performance and are useful to calculate various metrics such as accuracy, precision, recall, and AUC-ROC curve [27] for binary classifiers.

Accuracy

Accuracy is a useful metric if you have a relatively balanced dataset. The formula for accuracy is:

$$\frac{TP+TN}{TP+FP+TN+FN}$$

However, when we have an unbalanced dataset, we need other options.

Precision

"The precision is the ability of the classifier to not label as positive a sample that is negative"[13]. The formula for calculating precision score is:

$$\frac{TP}{TP+FP}$$

Recall

"The recall can be described as the ability of the classifier to find all the positive samples"[13]. The formula for calculating recall score is:

$$\frac{TP}{TP+FN}$$

To find a good balance between precision and recall, one can use the F1 score, which gives you the harmonic average between them [31]. The formula for F1 score is:

$$F_1 = \frac{TP}{TP + \frac{FN+FP}{2}}$$

In practice, a classifier cannot have high recall and high precision at the same time. Increasing the recall will decrease the precision, and vice versa. Thus, you have to look at what is important in the classification task before choosing what you want to try to minimize. For instance, in a cancer detection task, it is typically better to classify a healthy patient as cancerous than classifying a cancerous patient as healthy.

Chapter 3

Deep learning for computer vision

Deep learning is a small subfield of machine learning, but it is what draws the huge interest in artificial intelligence and machine learning these days. The idea of deep learning is nothing new, but has actually been around for decades. Deep learning algorithms are based on a class of machine learning algorithm called artificial neural networks (ANNs). The popularity of deep learning has exploded in the last decade, thanks to the rapid advances in computing power and the availability of tremendous amounts of data. Deep learning algorithms currently form the state-of-the-art machine learning models in many domains, including healthcare, finance, self-driving cars and more.

In this chapter, we will present the basic concepts of ANNs, and continue with deep neural networks and look at how they are used in the field of computer vision.

3.1 Neural networks

ANN models are very loosely inspired by the biological neural networks in the brain. Similar to the biologicalneural networks, ANNs are made up of a set of computational units, called neurons (also known as nodes). Figure 3.1 shows the structure of an artificial neuron.

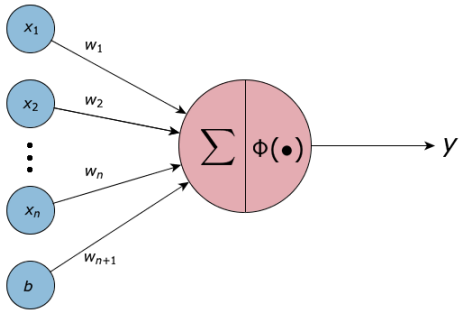


Fig. 3.1 Artificial neuron.

A neuron calculates the dot product between the input signals $X = [x_1, x_2, \ldots, x_n]$ and its corresponding weights $W = [w_1, w_2, \ldots, w_n]$ as follows:

$$s = \bar{X} \cdot \bar{W} = \sum_{i=1}^{n} x_i w_i$$

An additional constant called a bias is often added to the weighted sum to give ANN models more flexibility. We use b to denote the bias parameter:

$$s = \bar{X} \cdot \bar{W} = \sum_{i=1}^{n} x_i w_i + b$$

This value is then passed through an activation function Φ that calculates the output signal (we will discuss activation functions in section 3.2). That is the basic concept of an artificial neuron.

Modern ANNs are just a group of neurons stacked together in layers. They consist of an input layer, one or more hidden layers, and an output layer. ANN with more than one hidden layer is commonly referred to as a deep neural network (DNN) [31].

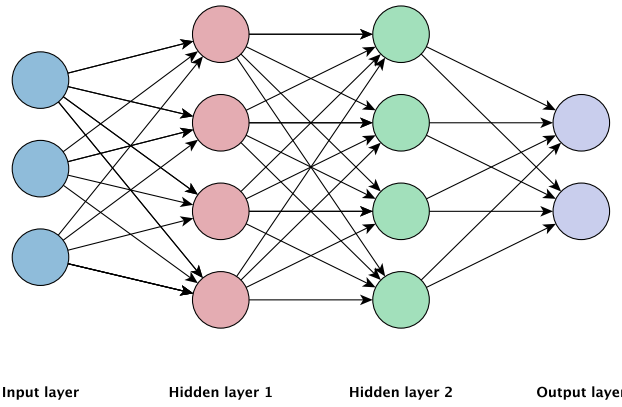


Fig. 3.2 Example of a deep neural network with two hidden layers

There are various types of architectures for ANNs, such as feedforward neural networks, recurrent neural networks, convolutional neural networks, and more. The ANN model shown in figure 3.2 is called a feedforward network since the data is only passed in one direction, from the input layer through the hidden layers to the output layer. The bias neurons and the weights we see between the neurons in this figure are the part of the network that actually learns during training. The weights are usually randomly initialized before training, often by using something called He initialization from He et al. [41]. During the training process,

described in section 3.3, these weights get updated so that the network becomes able to correctly predict the true value Y for every (or most) training example X.

ANNs with more than one layer of learnable weights can represent nonlinear relationships between input variables and output variables [86], thanks to nonlinear activation functions.

3.2 Activation functions

Activation functions are crucial for making ANNs able to learn and tackle complex tasks. As mentioned in section 3.1, they are functions used in neurons to calculate output signal based on input signals. The output signal of these functions decides whether the data given to the neuron is deemed relevant to what extent it should be passed on to the next neuron [11].

In order to have the ability to learn complex tasks, ANNs must add nonlinear activation functions to their neurons. If ANNs did not use nonlinear activation functions, then they would only be able to solve linearly separable problems, regardless of how many hidden layers they use [107]. This is because compositions of linear dot product operations are still linear.

There are various types of nonlinear activation functions, some of them are shown in figure 3.3.

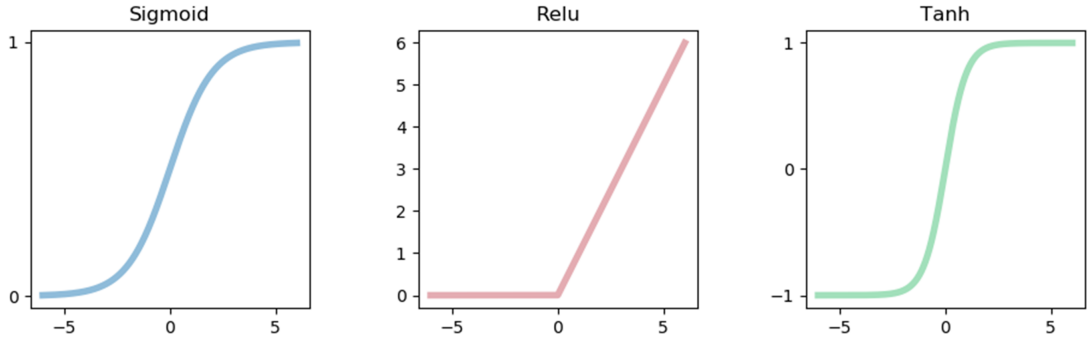


Fig. 3.3 Activation functions.

Historically, the most common activation functions to use were the sign, sigmoid and the hyperbolic tangent functions [17]:

$$\Phi(s) = sign(v)$$
$$\Phi(s) = \frac{1}{1+e^{-s}}(sigmoid function)$$
$$\Phi(s) = \frac{e^{2s}-1}{e^{2s}+1}(tanh function)$$

However, an activation function called rectified linear unit (ReLU) has become more popular in the last few years because it was found to significantly accelerate the training of multilayered neural networks compared to other functions [55]. ReLu simply sets all negative outputs to zero and leaves the rest untouched:

$$\Phi(s) = max\{s, 0\}\,(ReLU)$$

3.3 Training multilayer neural networks

Preprocessing image data

In general, it is a good practice to normalize the data (between 0 and 1) before we feed them into the network. We do this to increase the speed of the learning process. For image data, this can be achieved by subtracting the mean from each pixel and then dividing the result by the standard deviation.

Backpropagation

In section 3.1 we mentioned that the weights in ANN models gets updated during the training so that the ANN can correctly predict the true value for each training example.

But how do we train a neural network?
First, for each training example, we propagate the data forward through the network to the output layer to make a prediction. A loss function (see section 3.3) is then used to calculate the network's prediction error (the difference between the true value and the predicted value).

This prediction error is then propagated back through the network to calculate the effect each weight had on the error. This technique is commonly known as backpropagation.

The weights are then updated according to the impact they had on the error using an optimization method (such as gradient descent, discussed later in this section). When the

network has iterated once through all training elements, we say that it has completed an epoch. Figure 3.4 illustrates the entire training process.

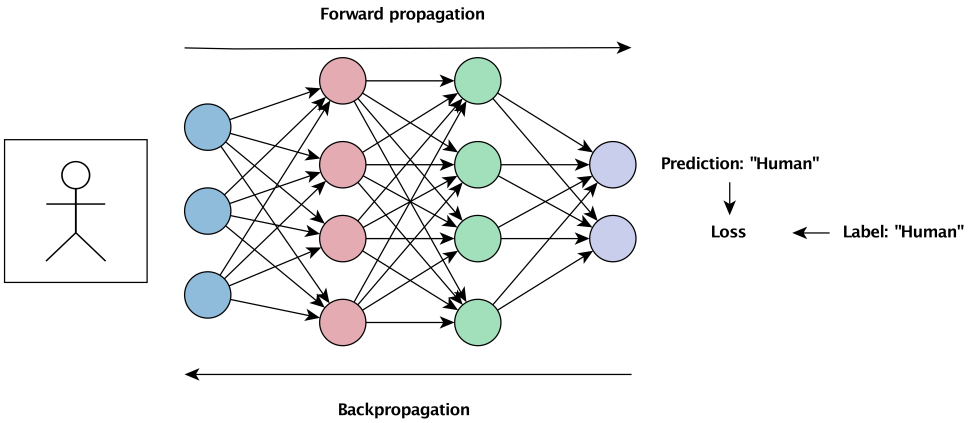


Fig. 3.4 Neural network training. Note that this illustration is inspired by the figure on p. 4 in *A performance and power analysis* [69].

Loss functions

As mentioned before, a loss function tells us how well the network manages to predict the correct output for the provided data. In this section, we will go through some commonly used loss functions.

Cross-entropy

Cross-entropy is a loss function often used in classification tasks. The equation for cross-entropy is as follows:

$$loss = -\sum_{i=1}^{n} y_i log(\hat{y}_i)$$

Where n is the number of instances, y is the true value, and $\hat{y}$ is the predicted value. Cross-entropy gives you an output value between 0 and 1, where high values denote bad performance, and vice versa.

Root mean square error

Root mean square error (RMSE) is a loss function usually utilized for regression tasks. It measures the difference between the predicted value and the actual value, as shown in the following equation:

$$loss = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Dice coefficient

The dice coefficient (Dice), is a commonly used loss function in medical image segmentation tasks [91]. Segmentation can be seen as a form of classification, where the task is to classify each pixel within an image. The equation for Dice loss is as follows:

$$loss = \frac{y \cap \hat{y}}{|y| + |\hat{y}|}$$

Dice loss gives you an output value between 0 and 1, where the dice score of 1 indicates perfect prediction.

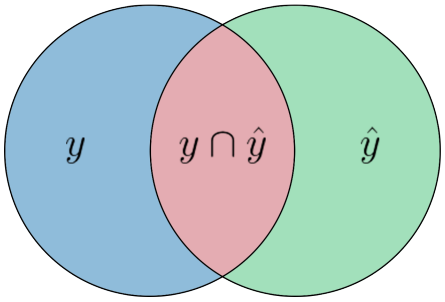


Fig. 3.5 Dice coefficient

Gradient descent

Gradient descent is a general optimization algorithm used to find the minimum value of a function. In neural networks, gradient descent is used to iteratively update the weights in such a way that reduces the prediction error of the network. The algorithm is fed the derivative of the loss function with respect to all the weights in the network from backpropagation. Then, it updates the weights in each layer by moving them a step in the opposite direction of their gradients. This entire process is repeated until we have reached a local or global minimum, as shown in figure 3.6. The size of each step is determined by a hyperparameter called learning rate. A small learning rate will cause the algorithm to converge slowly. On the other hand, with a large learning rate, the algorithm might overshoot the minimum.

There are three variants of gradient descent: batch gradient descent, stochastic gradient descent, and mini-batch gradient descent [81]. Batch gradient descent utilizes the entire training set to calculate the gradient of the loss function in each iteration. This means that this method, in general, is very slow and computationally intensive. On the other hand, stochastic gradient descent uses a random training sample at each iteration, which makes this method significantly faster than batch gradient descent. However, the movement on the loss surface will be very irregular, making it difficult to find the optimum value. Mini-batch gradient descent is a combination of batch and stochastic gradient descent, where one selects a random subset of the training set at each iteration to calculate the gradients. More information about gradient descent and other optimization algorithms can be found in Sebastian Ruder's paper titled *An overview of gradient descent optimization algorithms* [81].

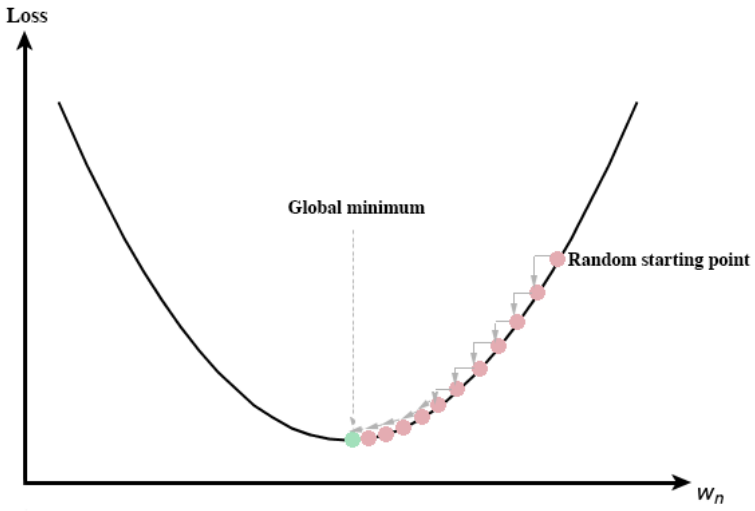


Fig. 3.6 Gradient descent

3.4 Convolutional neural networks

There are various types of neural networks, and each one has its own advantages and disadvantages. In this section, we will look at a class called convolutional neural networks (CNNs) that have been shown to work particularly well for computer vision tasks. Instead of receiving the input data as a vector of pixel values (as in feedforward networks 3.2), CNNs receives them as a matrix of pixel values (with width, height, and depth). In this way, we can keep the spatial relationships within the images. CNNs are like the artificial neural networks described in the introduction, but each neuron is only connected to a selection of neurons in

the previous layers (local connectivity), and weights are shared, in a manner motivated by principles used by the mammalian visual cortex. They were first introduced by Fukushima in 1980 [30], and famously used together with backpropagation by LeCun in 1989 [58] in a work that has inspired their use in computer vision ever since. In recent years, several CNN architectures such as AlexNet [55], ResNet [42], and DenseNet [47] have demonstrated state-of-the-art performance in various computer vision task.

Components of CNNs

In this subsection, we will go through the three primary layers in a CNN: the convolution layer, the pooling layer, and the fully connected layer.

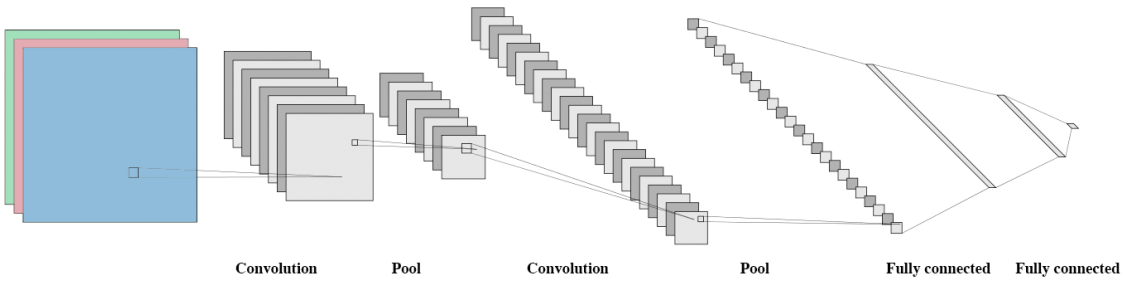


Fig. 3.7 Architecture of a convolutional neural network. (Source: [59])

Convolutional layer

The convolutional layer is the core building block in a CNN. Its function is to automatically extract useful features from input images. Typically, in the first layer, it finds the edges, colors, and other low-level features. As you get deeper in the network, the convolutional layers will learn more complex features (see figure 3.8). In order to represent these features, convolutional layer uses a set of matrices called kernels (also known as filters). Each of these kernels has the same number of dimensions as the input image, by design with the same depth size (e.g., the number of color channels), but smaller height and width size. During the training phase, when we propagate the data forward, each kernel slides over the activations produced by the previous layer with a predefined step size called a stride. It calculates the dot product between the kernel and the receptive field (i.e., a restricted region in the input that has the same size as the kernel). The output matrix from this process is called an activation

map (or a feature map). Figure 3.9 shows an example of how a 2x2x1 kernel with stride of 1 slides over a 3x3x1 input.

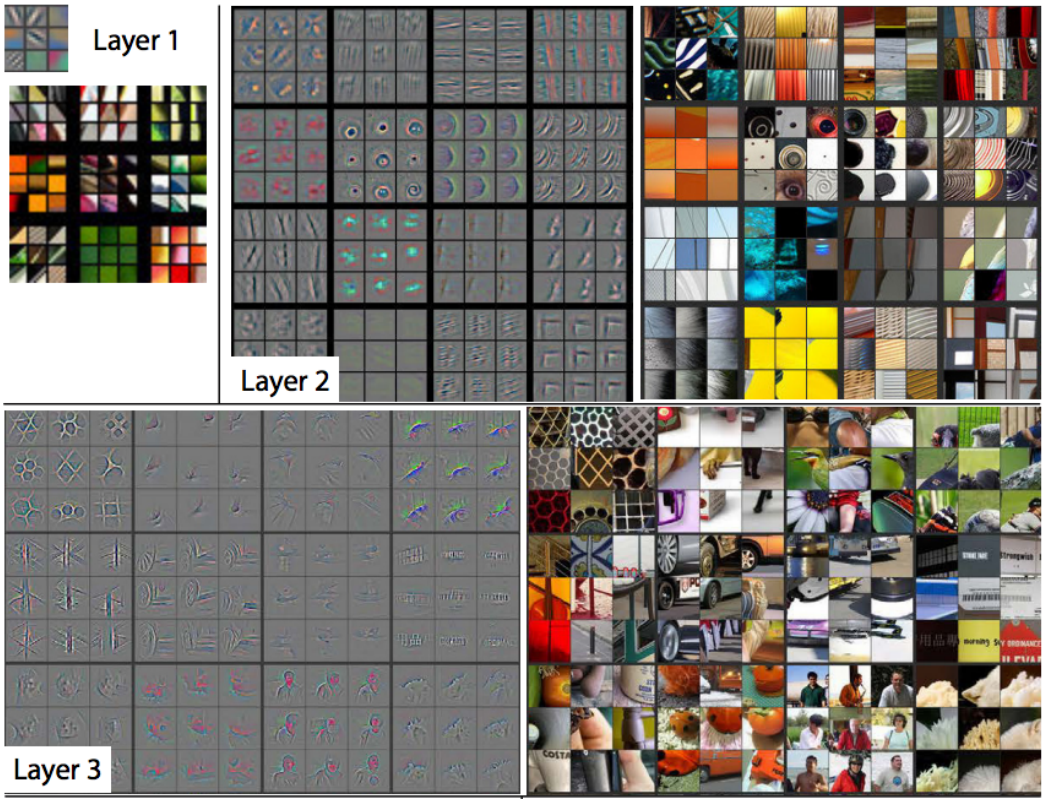


Fig. 3.8 Visualization of features learned from the ImageNet dataset. Adapted from p. 4 in *Visualizing and Understanding Convolutional Networks* [108].

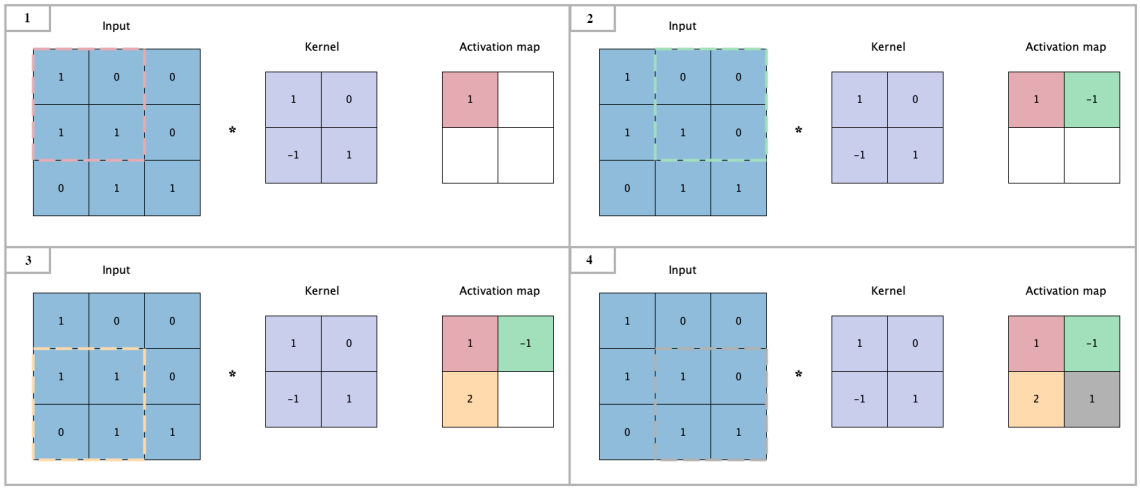


Fig. 3.9 An illustration of convolutional operation

Pooling layer

As shown in figure 3.7, after a convolutional layer we usually have a pooling layer however, in many modern architectures these are replaced by strided convolution. The main purpose of a pooling layer is to compress the activation maps to reduce the number of parameters in the network, reducing computational and memory requirements. The two most widely used pooling approaches in practice are max pooling and average pooling. As shown in figure 3.10, max pooling works by sliding a window across the input data (i.e., activation map), similar to a convolution, and at each step, we select the largest value in the pooling window. Average pooling, on the other hand, calculates the average value in the pooling window at each step. Note that pooling layers do not have any learnable parameters, which means that important information can be thrown away in this process.

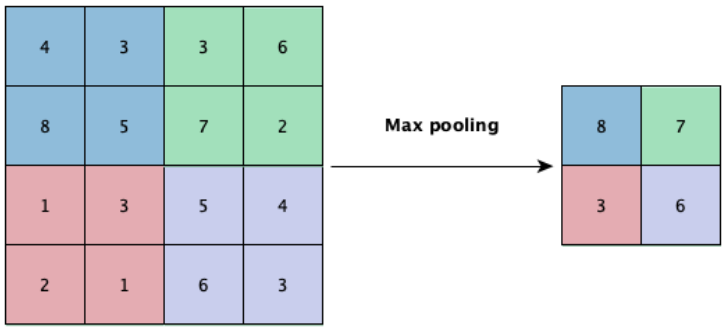


Fig. 3.10 Max pooling with 2x2 pooling window and stride 2

Fully connected layer

Fully connected layers work the same way as feed-forward networks. As the name implies, all the neurons in a fully connected layer have a connection to all the neurons in the next following layer (see figure 3.2). Fully connected layers are usually added at the end of the network to learn the mapping between the high-level features and the output classes. Again, modern architecture often drop these and replace them by convolutions, giving rise to fully-convolutional CNNs.

3.5 Regularization

CNN architectures are normally comprised of millions of parameters, giving them extremely high capacity and also making them prone to overfitting the training data. To deal with the

problem of overfitting, one can use various types of regularization techniques such as L1 and L2 regularization, dropout, early stopping, and data augmentation.

Early stopping

Early stopping is simply the process of stopping the training before it starts to overfit the training data. This technique is widely used in practice, often in combination with other regularization techniques to improve the generalization error. Overfitting is detected by measuring the training and validation loss, stopping the training process when the validation loss becomes significantly worse than the training loss.

L1 and L2 regularization

The basic idea of L1 and L2 is to add a penalty to the prediction error based on the complexity of the model. In practice, we do this by adding one of these terms to the loss function.
L1 regularization treats each weight similarly and encourages them to become zero. The equation for L1 regularization is as follows:

$$Loss = error(y - \hat{y}) + \lambda \sum_{i=1}^{N} |w_i|$$

On the other hand, L2 regularization penalizes larger weights harder but does not force them to become zero. The equation for L2 regularization is as follows:

$$Loss = error(y - \hat{y}) + \lambda \sum_{i=1}^{N} w_i^2$$

Dropout

The last regularization technique we will look at in this section is called dropout. Hinton et al. introduced this technique in a paper [44] in 2012, and it has shown to improve the performance of state-of-the-art neural networks in various domains [89]. The idea behind dropout is that at each training iteration, we ignore a set of neurons randomly with a predefined probability value (usually 50%). In other words, this means that the randomly selected neurons will not be considered during a certain training iteration. In this way, we end up with a simpler network at each iteration, and each neuron becomes less finely-tuned to the particularities of other neurons.

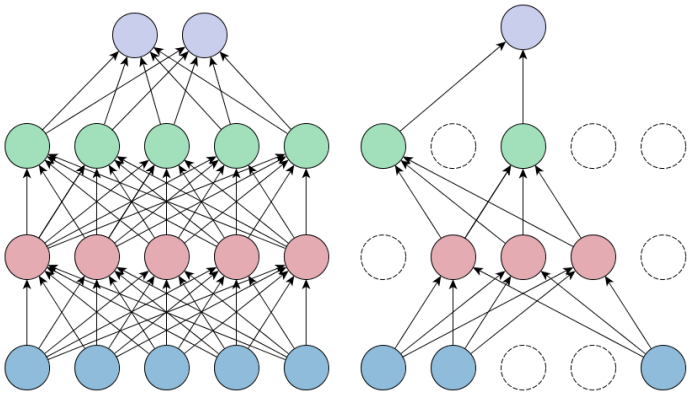


Fig. 3.11 An example of how dropout works in a feed-forward network

Data augmentation

The ideal solution to prevent overfitting is to collect more training data. In practice, this can be expensive and is often not an option. However, one way to obtain more data essentially for free is by utilizing a technique called data augmentation. This technique consists of creating new training data from existing ones by applying simple transformations such as flipping, rotating, scaling, zooming, or more advanced, domain specific transformations. Note that data augmentation should be used with caution since it can change the meaning of the images (see figure 3.12).

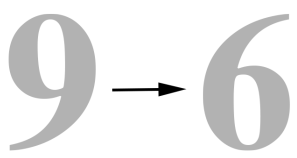


Fig. 3.12 Rotating the image on the left 180 degrees will change the semantic meaning of the image, but will still have the same label value.

Chapter 4

Transfer learning for deep neural networks

Transfer learning is one of the hottest topics in deep learning. Andrew Ng said at NIPS 2016 that "transfer learning will be the next driver of ML success"[82].

As mentioned in the introduction, transfer learning is the process of using the knowledge learned in one process and applying it to a different task. Although this is a very is a simple idea, it has shown to be a very powerful technique for deep neural networks in domains such as computer vision [24, 40], and natural language processing [46]. Nowadays, it is often shown to be advantageous to use pre-trained models for computer vision tasks rather than training them from scratch [65]. However, the transfer learning performance is dependent upon the similarity between the source task and the target task; the more similar tasks are, the more value is provided [23, 104].

In this chapter, we will present the technical details around transfer learning and the advantages of using this approach in computer vision tasks.

4.1 Transfer learning in computer vision

As we have seen earlier, the first few layers of CNNs often end up learning low-level features, such as edges, colors, shapes, etc. Such features appear to be common across different kinds of images [104]. Thus, it is often better to use the weights from another network trained to perform a similar task as a starting point instead of training from scratch. This is a form of transfer learning, also known as fine-tuning or pre-training.

Transfer learning methods

In practice, there are various types of approaches to transfer learning in computer vision tasks. The two most commonly used strategies are:

Using a pre-trained model

Training CNNs from scratch is a difficult task due to the enormous amount of training data and computational power required to train them. Thus, people usually initialize the weights for their target network from an already trained network. For instance, most deep learning frameworks provides a set of state-of-the-art 2D models pre-trained on the ImageNet dataset [83].
Note that if the purpose of the target task is different from the source task, as in figure 4.1, then you have to remove the output layer and add a new one with random weights to fit your problem. Or potentially more than one layer, depending on whether you need additional capacity for your target task. In addition, how many layers you should retrain (unfreeze) depends on the similarity between the tasks, and the amount of data that are available for the target problem. For example, if you have a small dataset you should probably not unfreeze many layers due to the risk of overfitting. On the other hand, if you have a large dataset, you can fine-tune the entire network without concerns for overfitting. As the weights in the earlier layers are typically more valuable after transfer than those on later layers, one can also use discriminative learing rates. That is, lower learning rates for early layers than for later layers.

Using a pre-trained model as fixed feature extractor

Instead of doing all the feature engineering manually as we mentioned in chapter 2, one can use pre-trained models to find the most important features for us. In order to do this, it is required that the pre-trained model at least partially contains your target problem. For example, if you want to classify cats and dogs, one can use a pre-trained model on ImageNet as a fixed feature extractor. As shown in figure 4.2, we can do this by removing the last fully connected layer and feeding the extracted features into a new ML model that learns to solve the target problem. It is also possible to use features extracted from one or more earlier layers, making this approach less dependent on the closeness of the tasks as earlier layers often contain generally useful features.

As you can see, this approach reduces the size of the features, which means that the extracted dataset can fit in a traditional ML algorithm such as SVM, random forest, etc. Tradional ML algorithms do not require the same amount of computational power and data as CNNs; thus, you may want to use this approach when you are low on data and computational

power. Or if you want to combine features automatically extracted by neural networks together with manually designed features, perhaps based on a different data source, in a single model.

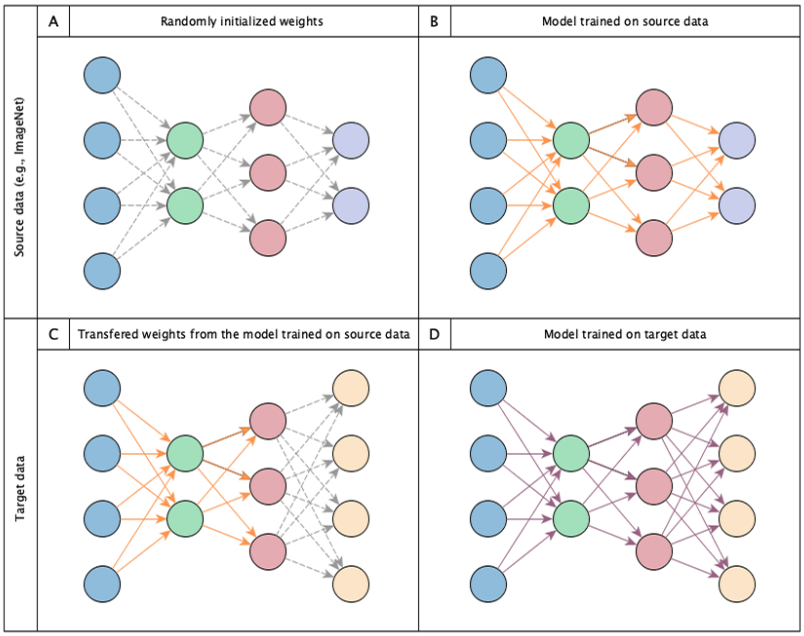


Fig. 4.1 An example of transferring weights from a model trained to classify two classes to a target model fine-tuned to classify four different classes. Dotted lines indicates randomly initialized weights

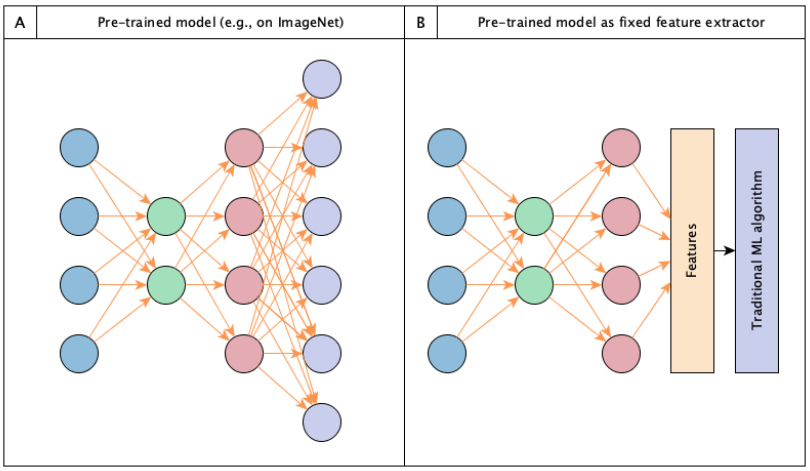


Fig. 4.2 An example of using a pre-trained model as a fixed feature extractor

Let us turn to some experiments in transfer learning for medical image analysis.

Part II

Experiments

The purpose of this thesis is to investigate the value of transferring knowledge from medical tasks that have a vast amount of annotated training data to other tasks with less training examples, and constructing techniques and software facilitating such transfer. The final goal is to enable application-specific 3D segmentation in medical data with a relatively modest need for manual input. In this part we present our experiments as follows:

1. **Transfer learning in 2D medical image classification tasks**
2. **The impact of layers in transfer learning in 3D medical image segmentation tasks**
3. **The effect of dataset size in transfer learning in 3D medical image segmentation tasks**

Chapter 5

Transfer learning for 2D medical images

5.1 Introduction

The advantage of applying supervised deep learning algorithms in medical imaging tasks has been demonstrated thoroughly across many applications. In order to develop such algorithms, large amounts of data are required. Unfortunately, in the medical domain, the main challenge for machine learning methods is access to such amounts of annotated data [26].

To overcome this problem, a couple of studies have investigated the benefits of transferring weights from a model pre-trained on the ImageNet dataset to medical tasks in recent years, including thoraco-abdominal lymph node detection and interstitial lung disease classification [84], Intima-media boundary segmentation, polyp detection, pulmonary embolism detection, colonoscopy frame classification [92], ultrasound kidney detection [79], and more recently, classification of cellular morphological changes [53]. All of these studies have demonstrated that transfer learning from ImageNet outperforms models trained with random weights, or in the worst case, provides the same performance.

In this chapter, we propose a technique that involves indirectly transfer from ImageNet to the target network, by first passing through a network more similar to the target dataset. The idea is illustrated in figure 5.1. Independeltly from our work, an article [23] published by Yin Cui et al. in 2018 demonstrated that one could improve the transfer learning performance with such an approach. We are not aware of any studies that have applied this approach for transfer learning in 2D medical images. The investigations described in this chapter were also presented by the author at NVIDIA's GTC Europe 2018 conference in Münich [52].

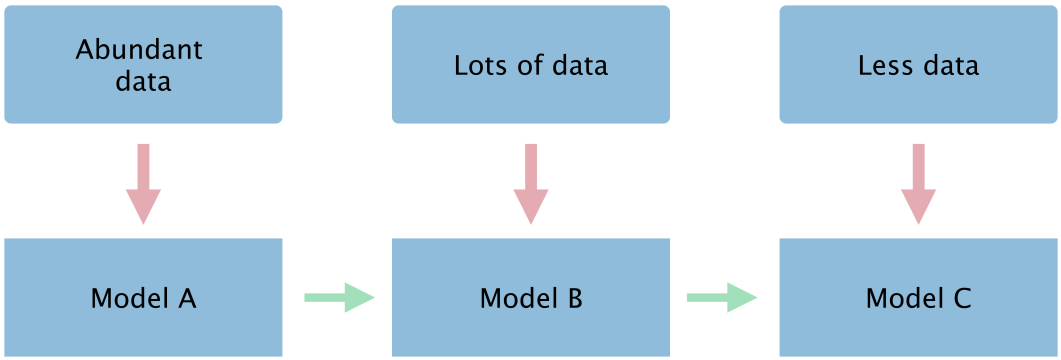


Fig. 5.1 Transferring knowledge from one task to another

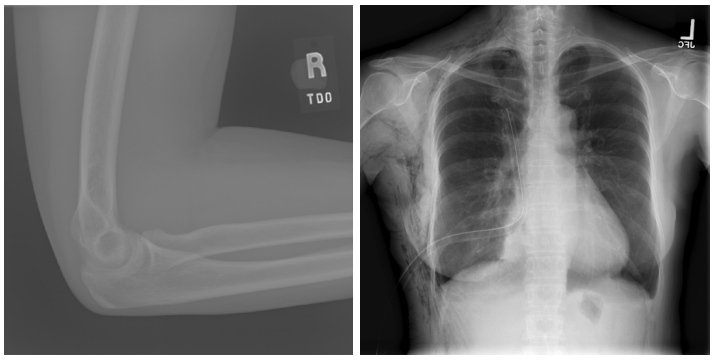
5.2 Methods and materials

Datasets

In this case study, we used the following two publicly available X-ray datasets:

- MURA dataset (figure 5.2a) provided by Andrew Ng's Standford ML group consist of 40,895 upper limb musculoskeletal X-ray images from 14,982 studies, where each study is manually labeled as either normal or abnormal(5,915 abnormal studies, containing a total of 15,117 images) [77]. In our research, we used 13,773 of these studies for training and validation (8402 no finding and 5371 abnormality) and left out the rest for testing (665 no finding and 544 abnormality).

- NIH ChestX-ray dataset (figure 5.2b) contains 112,120 automatically annotated frontal-view X-ray images from 30,805 unique patients, each of them labeled with up to 14 different diseases such as pneumothorax, nodule, and effusion [99]. Of these, we used 53137 for training and validation (50500 no diagnosis and 2637 pneumothorax) and 12526 for testing (9861 no diagnosis and 2665 pneumothorax). Note that detecting pneumothorax from frontal-view X-ray images alone is not a very realistic task from a radiologists perspective, but it is a very good test-case for machine learning.



(a) Mura image (b) Chest X-ray image

Network architecture

Following the approach taken by the CheXNet study from Andrew Ng's Standford ML group [78], we used a DenseNet121 network with some additional layers. An illustration of the architecture used in our study is shown in figure 5.3 (see **experimental setting** for details). A well-known problem when training deep neural networks is the challenge of propagating valuable gradients to the early layers (e.g., gradients gets smaller as we backpropagate), which makes it hard to train these layers. This problem is commonly referred to as the vanishing gradient problem [34]. The Dense blocks shown in figure 5.3 alleviates the vanishing gradient problem by connecting all the layers to each other as a directed acyclic graph. Note that these blocks are an extension of residual blocks [103], that will be explained in chapter 6.

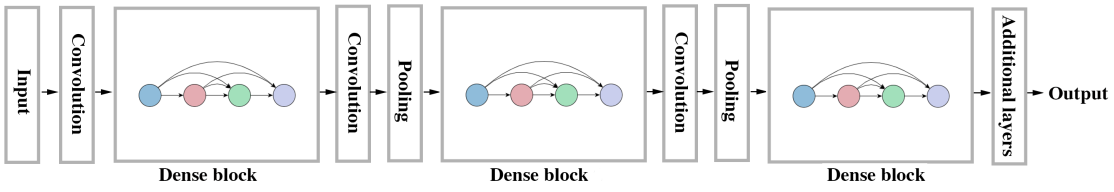


Fig. 5.3 An illustration of the architecture used in our transfer learning study. Note that this illustration is inspired by the figure on p. 3 in *Densely Connected Convolutional Networks* [47].

Deep learning frameworks

In recent years, several open-source deep learning frameworks have been developed and backed by major technology companies, such as Google (TensorFlow) and Facebook (PyTorch). This has made the process of implementing and modifying DNN easier. In order

to construct and train our networks in this study, we used Pytorch through the fast.ai deep learning library. A short summary of these libraries is given below.

Pythorch

Pytorch is a popular open-source python deep learning framework built by Facebook's AI group [67]. It is a flexible framework based on a machine learning library called Torch [20] and offers among other things support for running tensor calculation such as NumPy [71] on Graphical Process Units (GPUs) and algorithmic differentiation operations on tensors which makes it easier to train DNN [72, 73].

Fastai

Fastai is a deep learning library built on top of PyTorch developed with the aim of simplifying the process of training DNN, utilizing state-of-the-art deep learning approaches in various domains (e.g., computer vision, natural language processing, etc) [45].

Experimental setting

Our models were all trained using the following regime:

Optimizer: Adam

Learning rate: Base learning rate 0.0001, with scaled down learning rates for earlier layers when using transfer learning (i.e. discriminative learning rates)

Data augmentation: Random horizontal flips

Image sizes: 299 x 299

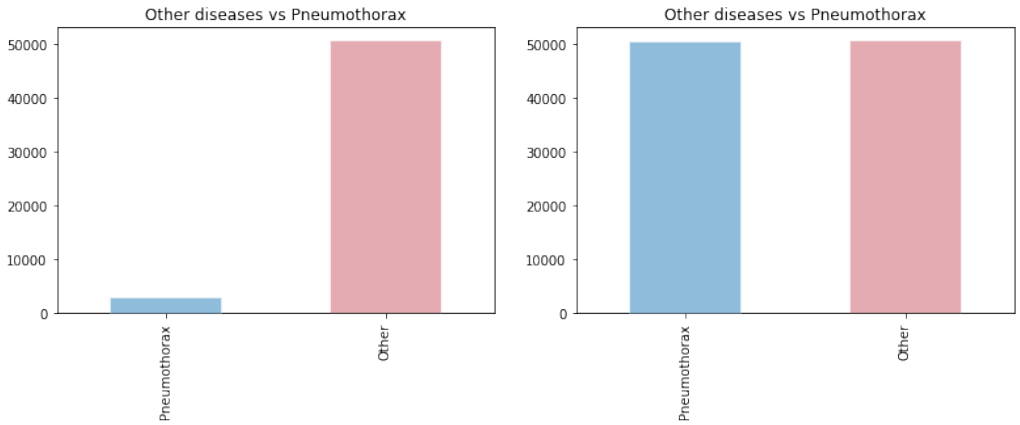
Batch size: 16

GPU: NVIDIA GTX 1080 Ti

Model: Pytorch's DenseNet121, with the last layer removed and the following added: *AdaptiveConcatPool2d, AdaptiveMaxPool2d, BatchNorm, Linear, BatchNorm, Linear, LogSoftmax*

Preprocessing

As shown in figure 5.4a, we encountered the problem of imbalanced classes in the ChestX-ray dataset (ratio difference 1:19). In order to deal with this problem, we decided to oversample the pneumothorax class by duplicating each sample 18 times (see figure 5.4b). Note that we could have combined this method with another technique called undersampling to lower the risk of overfitting. As the name implies, undersampling is the process of reducing the size of the majority class, for instance, by selecting a subset of the majority class.



(a) Imbalanced (b) Balanced

Fig. 5.4 Oversampling imbalanced data

Model evaluation

In order to assess our approach, we choose as our target objective detection the detection of pneumothorax from chestX-ray dataset, pre-trained on the MURA dataset. We compare the training loss with a model trained from scratch on the chestX-ray and with another model pre-trained on ImageNet. As we have seen earlier, training loss represents the price paid for the inaccuracy of predictions during the training.

5.3 Experimental results

MURA

The pre-trained MURA model used for our transfer learning approach achieved a study-wise testing accuracy of 82.66%. By study-wise we mean that if an image in a study gets classified as abnormal, every other image in the study receives the same prediction even though they

could have been classified as normal beforehand. The reason for study-wise prediction is, in a real case scenario, a radiologist would likely have classified an X-ray examination as abnormal if one of the images showed an abnormality.
The confusion matrix depicted in Figure 5.5 and accuracy measurement for each study type shown in Table 5.1 provides further detail about the obtained classification result.

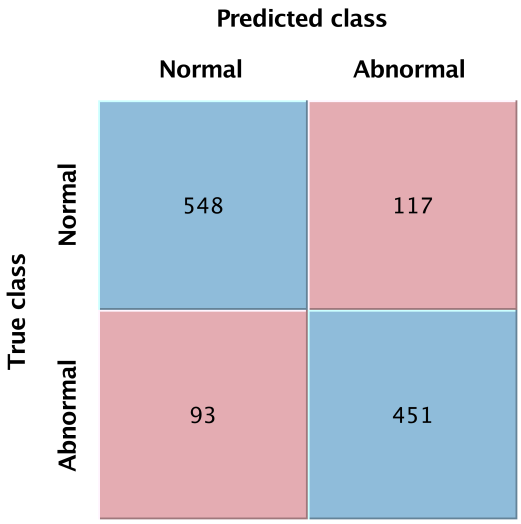


Fig. 5.5 Confusion matrix of the prediction result for the MURA dataset

Type	Accuracy
1: Wrist	86.97%
2: Forearm	83.33%
3: Hand	82.63%
4: Humerus	86.02%
5: Shoulder	74.36%
6: Elbow	83.75%
7: Finger	81.71%

Table 5.1 Accuracy for each X-ray type in the MURA dataset

Transferring to ChestX-ray

As we can observe in figure 5.6, the pre-trained models learn significantly faster than the model trained from scratch. In addition, the model whose weights were initialized from the model trained on MURA reaches a lower training loss quicker. This illustrates how our approach to transfer learning leads to less need for training data: a given loss value is reached earlier for the network pre-trained on ImageNet and then MURA compared to the network pre-trained only on ImageNet.

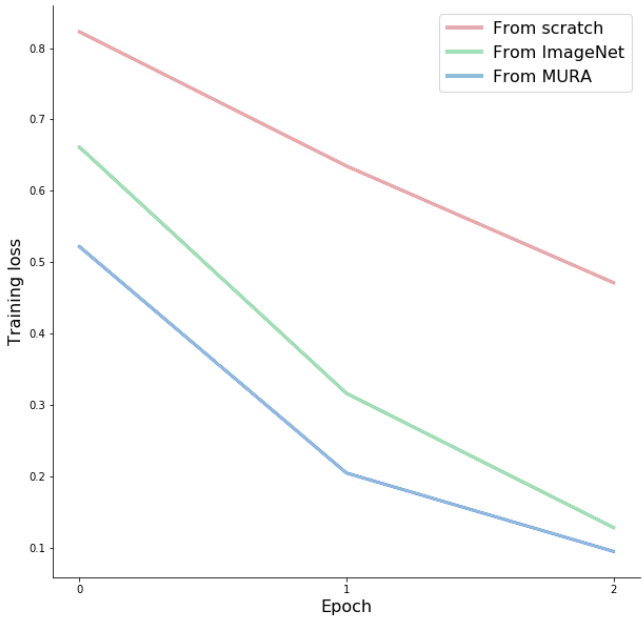


Fig. 5.6 Training loss for our models

The difficulty of this medical task led to prediction accuracy of only 82.09% on the test set. The confusion matrix of the classification results is shown in figure 5.7 Our source code for this study is publicly available at: https://github.com/skaliy/Deep_Learning_Kaliy.

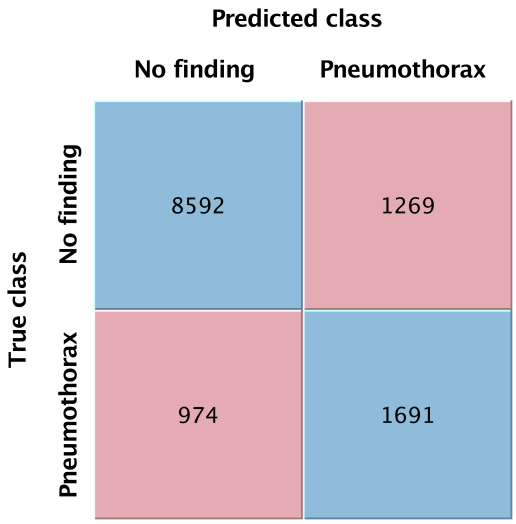


Fig. 5.7 Confusion matrix for the ChestX-ray dataset

5.4 Discussion

In this study, we have presented an approach for transfer learning in 2D medical images, where we first train a network pre-trained on the ImageNet dataset to detect abnormality in bone X-rays, before fine-tuning this network for the primary task of classifying pneumothorax in chest X-rays. Upon completing the study, we learned that the ChestX-ray dataset contains a lot of label noise (e.g., examples of chest drain treated patients annotated as pneumothorax) [70]. In addition, we expect that transferring weights from a more accurate network would lead to a more significant effect. This project is an important step in our larger-scale effort to investigate the value of transfer learning for medical images. We are particularly interested in 3D medical images, where ImageNet is less relevant, making the idea of transferring from other, medical tasks more important, as we will study in the next chapter.

Chapter 6

Transfer learning for 3D medical images

6.1 Introduction

The challenge of having small numbers of training subjects is particularly prevalent for segmentation of regions in 3D medical images such as magnetic resonance imaging (MRI). Here, manual delineation is difficult, time-consuming, and expensive, and the established automatic or semi-automatic methods are slow. In order to overcome this problem, a couple of studies have converted 3D medical images to 2D to be able to use CNNs pre-trained on ImageNet data [101, 102]. However, the downside of going to 2D is that we might lose informative spatial relationships between the pixels in the images. Due to the fact that there are no good transfer learning strategies in 3D medical images, it is still common to use networks trained from scratch. However, the potential of using transfer learning is mentioned in a couple of papers [60, 96].

In this chapter, we will investigate the effect of transferring weights from a network trained on a large-scale 3D medical dataset to another 3D medical task with a smaller number of training samples. As far as we are aware, there are only a few studies that have looked into the value of this approach. Lundervold et al. created a 3D CNN for segmentation of left and right kidneys from DCE-MRI, pre-trained on the task of left and right segmentation of the hippocampus in T1-weighted MR images [63]. More recently, Chen et al. created a large-scale 3D medical dataset called 3DSeg-8, and utilized models pre-trained on this dataset for various 3D medical imaging tasks (e.g., lung segmentation, pulmonary nodule classification, and liver segmentation) with great success. [18].

6.2 Methods and materials

Datasets

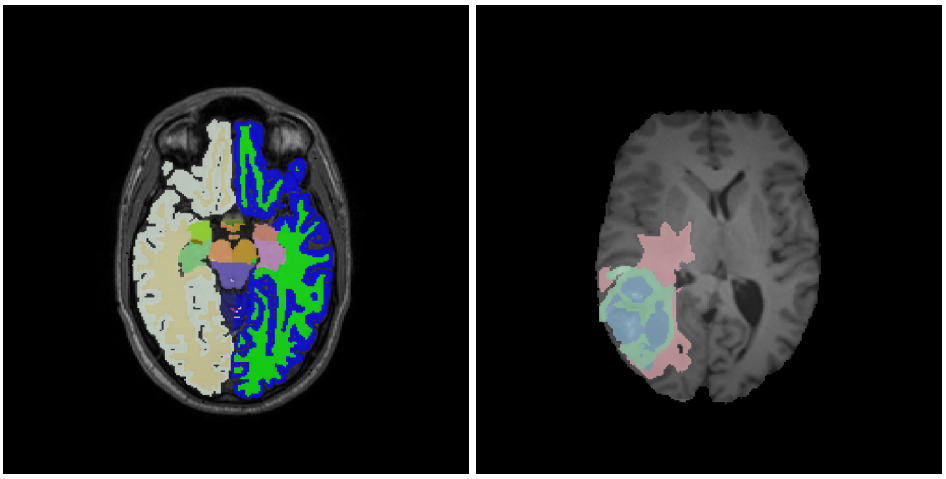
The following two datasets were used in our experiments:

- The IXI dataset (figure 6.1a) contains 581 T1 weighted scans and 579 T2-weighted scans of 581 different healthy subjects [Group]. Each image has a volume dimension of 256 x 256 x 150. The T1 weighted scans highlight structures with fat and is typically used to look at anatomy. On the other hand, T2 weighted scans highlight areas with water and is useful to detect pathology [95].

  We have already co-registered and automatically labeled the images with up to 42 different features (e.g., hippocampus, cerebellum, amygdala, etc.) using FreeSurfer 6.0 [28], NumPy [71], and Nibabel [14] as a part of another project. Segmentation of all the 42 different regions proved to be a difficult task due to difficulties in the ground truth labels (see **preprocessing** for further details). An essential part of transfer learning is to have a source network that performs well on a particular task. Therefore, we decided to reduce the complexity of the task by only looking at the hippocampus. This part of the brain has an important function in learning and memory [3]. Changes in the volume of the hippocampus are often associated with various neuropsychiatric diseases (e.g., epilepsy, Alzheimer's disease, etc.) [19, 29], but manual segmentation of the hippocampus to measure volumes is quite difficult [15]. In order to alleviate this problem, several automatic techniques have been developed, such as multi-atlas methods [49, 76]. However, these methods still require a lot of processing time [16]. Over the past year, a couple of studies have shown that the computation time can be significantly reduced by using CNNs, and still achieve high segmentation performance [16, 93, 100].

- The Multimodal Brain Tumor Image Segmentation Benchmark Challenge 2018 (BraTS) training dataset consist of 285 studies, where 75 subjects are low grade gliomas(LGG) and 210 subjects are high grade gliomas (HGG) [7–9, 66]. Each study contains four MRI modalities (T1, T1ce, T2, FLAIR) of size 240 x 240 x 155, and are manually segmented by one to four neuroradiologists [66]. Since the IXI dataset only consists of T1 weighted and T2 weighted scans, we restriced ourselves to the same modalities in this dataset. As shown in figure 6.1b, the tumor is annotated into three different subregions: Edema (the pink region), non-enhancing core (green region), and enhancing core (blue region). Edema is usually represented in FLAIR images [9], and because of this, we have decided to exclusively look at the tumor core (non-enhancing core and

enhancing core together) in our experiments. This led us to exclude the LGG images, where the major part of the tumor is comprised of edema and non-enhancing core [21]. According to an article published by Tidsskrift for Den Norske Legeforening in 2011, HGG affects around 200 Norwegian patient's every year, and has a poor prognosis (e.g., a five-year survival rate of 6.1 %) [90]. In addition, HGG treatments make great demands on follow-up care (e.g., regular medical checkups). Deep learning algorithms that can segment tumor regions accurately from the early stages can, therefore, be a highly valuable tool for clinicians to improve the treatment of HGG patients.



(a) IXI (b) BraTS

Library

Tensorflow

Tensorflow is a widely used open source library for numerical computation and uses a structure known as a dataflow graph to describe the computational data flow in an application. It was initially developed by the Google Brain team for internal use in their machine learning and deep learning research [2]. TensorFlow is a flexible library that can run on both GPUs and CPUs across a wide range of platform including Windows, MacOS, Linux, Android, and iOS [2, 31]. In addition, Google has provided experimental support for running TensorFlow on Tensor Processing Units (TPUs), which are Google's own application-specific integrated circuits (ASICs) specialized for machine learning tasks [Google]. The TensorFlow library is constantly evolving with frequent updates. Recently, Google released TensorFlow 2.0 alpha that aims to simplify the process of using this library [ten].

NiftyNet

Niftynet is a deep learning library built on top of Tensorflow designed for research in medical image analysis [33]. The purpose of this library is to simplify the process of developing deep learning algorithms for 2D, 3D, and 4D medical imaging tasks, such as image segmentation, image registration, and image generation [33]. Note that CNNs utilized for 3D, and 4D medical imaging tasks have usually larger amounts of learnable parameters compared to CNNs used for 2D images, which means that they require more computational power. In order to moderate this problem, NiftyNet uses a patch-based image analysis approach as shown in figure 6.2. A window sampler samples image windows from the input data, and a window aggregator decodes the output from the network. There are different types of window samplers that can be used to generate these windows, such as weighted sampling, where the sampling probability of each window depends on the frequency of the classes. Less frequent classes are sampled more often. As a result of this window sampling, NiftyNet does not use the term epochs.

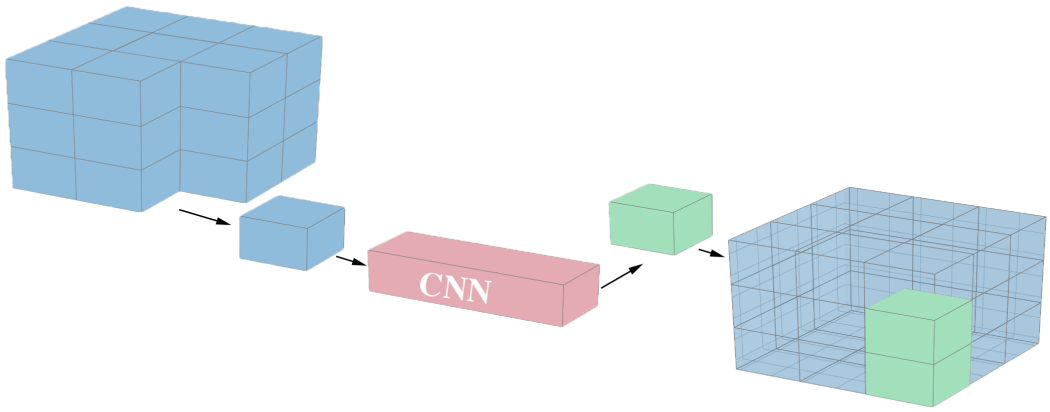


Fig. 6.2 Patch-based image analysis.

Network architecture

Following the approach by Wenqi Li et al. for segmentation of 155 neuroanatomical regions from brain MRI images [60], we used a high-resolution, 3D convolutional network (HighRes3DNet). An illustration of the HighRes3DNet used in our experiments is presented in figure 6.3, and the number of learnable parameters for this network is shown in table 6.1. Note that each residual block in this network utilizes skip-connections, as illustrated in figure 6.4. Skip connections were introduced in ResNet in 2015 by Kaiming He. et al., where they demonstrated state-of-the-art performance in the ILSVRC with a 152 layer deep neural network that utilized such connections [42]. Before skip connections, no one had managed to

train such deep neural networks due to the vanishing gradient problem (discussed in chapter 5).

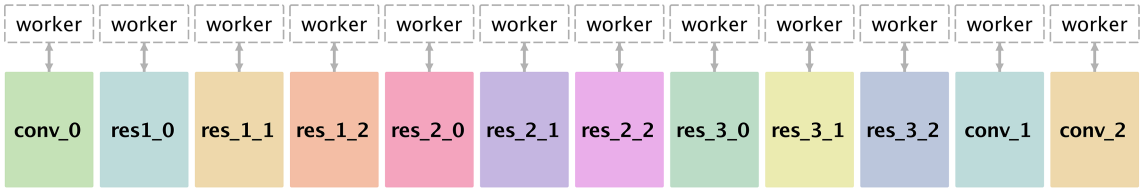


Fig. 6.3 Visualization of the high-resolution, 3D convolutional network architecture utilized in our experiments

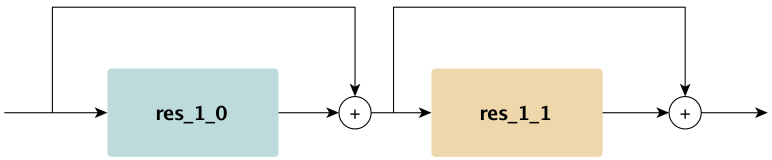


Fig. 6.4 An illustration of skip connection between two residual blocks in high-resolution, 3D convolutional network.

Layers	**Learnable parameters**
1. Conv_0	964
2. Res_1	42060
3. Res_2	153132
4. Res_3	610380
5. Conv_1	5602
6. Conv_2	174
Total	812312

Table 6.1 Total number of learnable parameters for the high-resolution, 3D convolutional network used in our experiments.

Experimental setting

Our models were all trained using the following regime:

Optimizer: Adam

Window sampling: Weighted

Learning rate: Base learning rate 0.001

Regularization: L2

Data augmentation: Rotation (-10.0, 10.0) and scaling (-10.0, 10.0)

Patch sizes: 96 x 96 x 96

Batch size: 2

GPU: NVIDIA Tesla V100

Model: NiftyNets's HighRes3DNet

Preprocessing

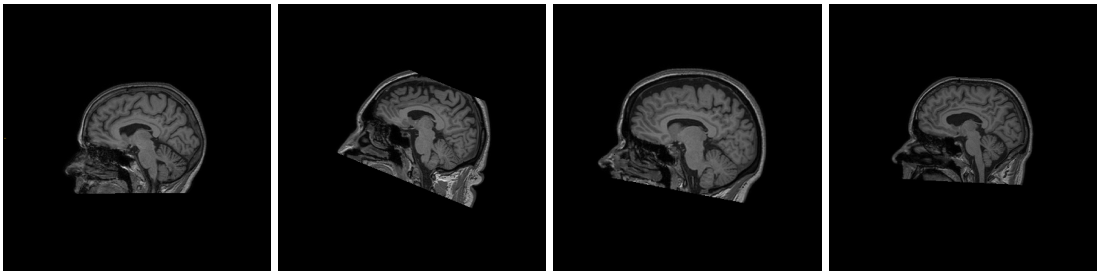
Normalization

All the MRI modalities for IXI and BraTS were normalized by subtracting the mean value and dividing by the standard deviation.

Removing abnormal training data

A problem we encountered in the IXI dataset was that some of the images were cropped differently than the rest, for instance, the IXI642 image shown in figure 6.5. This was caused by our co-registration of the T1 and T2 images. To handle this problem, we decided to train a network on the entire dataset, and remove the images that the network predicted poorly on. By applying this approach, we ended up removing 145 images from the dataset. Note that our source network trained on this modified dataset can slightly overfit these data due to this approach, making it impossible to compare our results from the hippocampus segmentation with other similar studies that we have mentioned earlier. But as we are mainly interested in creating a well-performing source network, this is not deemed problematic. The cropping

issue could have been handled by building a more robust network. However, this is out of the transfer learning studies scope.



IXI416 IXI642 IXI325 IXI192

Fig. 6.5 Examples of T1-weighted images in the IXI dataset

Data split

- We split the modifed IXI dataset randomly into 364 training subjects, 44 validation subjects and 44 testing subjects.
- The HGG cases from the BraTS dataset is split randomly into 105 training samples, 63 validation samples, and 42 test samples to assess the value of our transfer learning approach. No subject was present in more than one of these dataset.

Model evaluation

The impact of the pre-trained layers

In order to study the value of layers transferred from the pre-trained IXI model, we have decided to fine-tune five unique models on the BraTS dataset with freezing different layers (e.g., conv_0 frozen, conv_0 and res_1 frozen, etc.). In addition, we will train a model from scratch that will be our baseline model. The value of each layer will be assessed through the training- and validation loss, and the average Dice score of the segmentation results on the training- and validation data.

The effect of training set size in the target task

The best performing models from the previous study will be used in this experiment to investigate the effect of the BraTS training set size. Note that, if the model trained from scratch is not one of them, then it will get a free pass into this experiment since it is our baseline model. To analyze how each model is affected by the size of the training data, we

partition the training set into four sets (e.g., S1, S2, S3, S4), and train all the models on each of them. Every set except the first one builds on the previous set, as shown in figure 6.6. The performance of the models will be evaluated based on the training time and the average Dice score of the segmentation result on the test data.

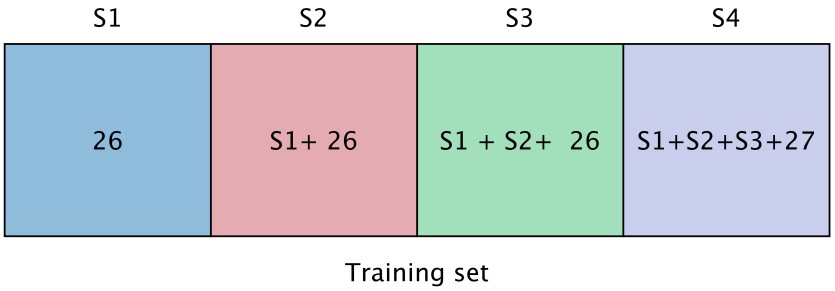
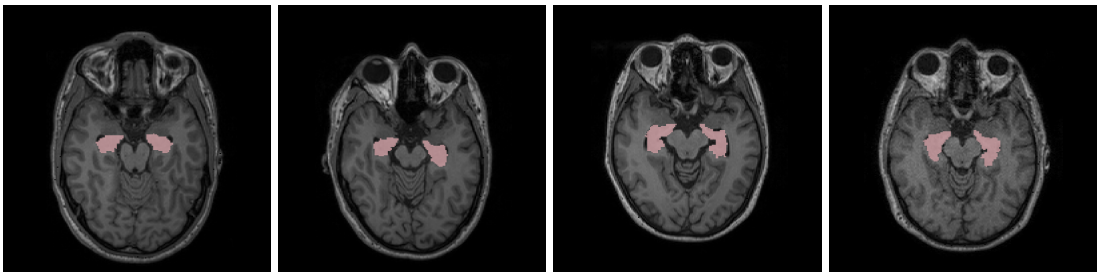


Fig. 6.6 An illustration of how we partition the training set

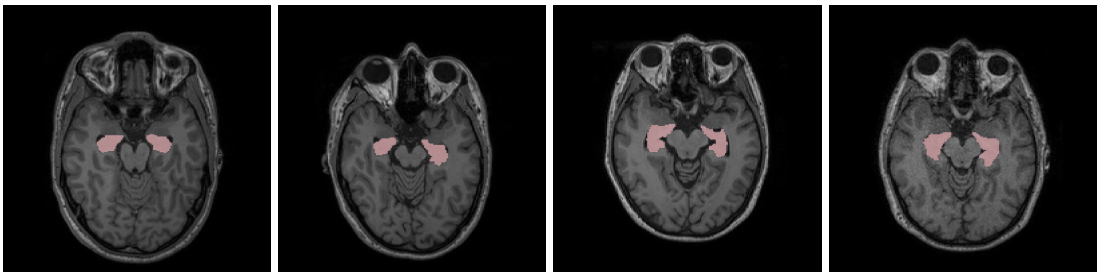
6.3 Experimental results

IXI

The source model trained on the training set and the validation set achieved an average Dice score of 0.87 on the test set. Some of these segmentation results and their corresponding ground truth labels are presented in figure 6.7. Because we dropped a couple of subjects from the dataset in the preprocessing phase, we decided to train the final model on the entire modified dataset for a couple of iteration with a small learning rate. This approach might give a performance and robustness boost. However, there is no way to evaluate the performance without bias since the model has seen all the available data.



Ground truth labels

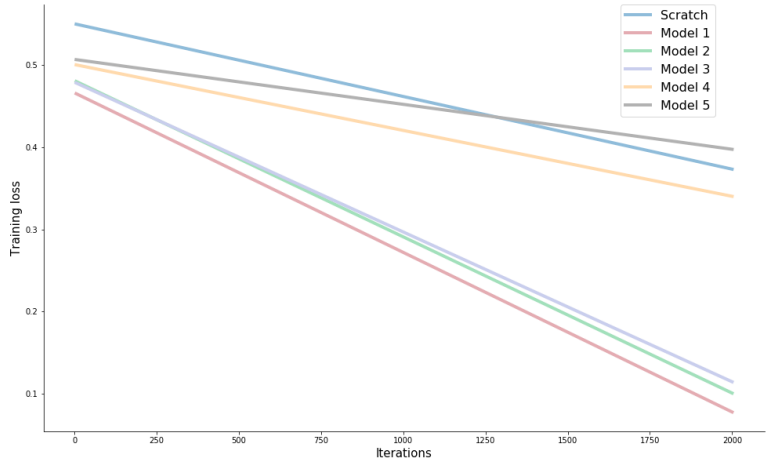


Segmentation results

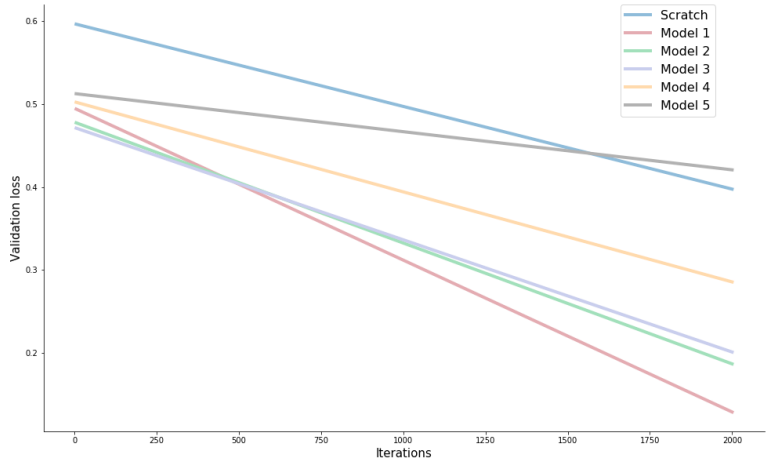
Fig. 6.7 Comparison of ground truth labels and segmentation results from four test cases in the modified IXI dataset

The impact of layers in transferring from IXI to BraTS

The training loss and the validation loss obtained by our models are presented in figure 6.8. Note that the loss functions are actually very spikey, but have been smoothed to visualize the value of each layer. The reason for this unstable fluctuations is likely caused by the small network architecture, the batch size and the patch-based image analysis approach shown in 6.2. As we can deduce from the loss functions, the pre-trained models with the early layers frozen converge significantly faster than the other models. This indicates that the early layers from our pre-trained model are more useful than the higher layers when transferring weights to solve our task in the BraTS dataset. These results are consistent with the theory we have seen earlier that the early layers learn more task-general features (e.g., edges, patterns, etc.), and higher layers learn more task-specific features (see figure 3.8).



Training loss



Validation loss

Fig. 6.8 The training and validation loss for our models. Note that the specified model number implies how many layers that are frozen.

Although the loss is very low for some of our pre-trained models, the segmentation performance of these models is still poor, as shown in table 6.2. From the obtained results we can see that the pre-trained models 1, 2, and 3 overfits. From the actual loss functions, it was difficult to say when the models began to overfit. However, note that the purpose of this experiment is to investigate the value of the transferred layers. Hence, this is not something we need to worry about in this experiment, but it will be taken into consideration in the

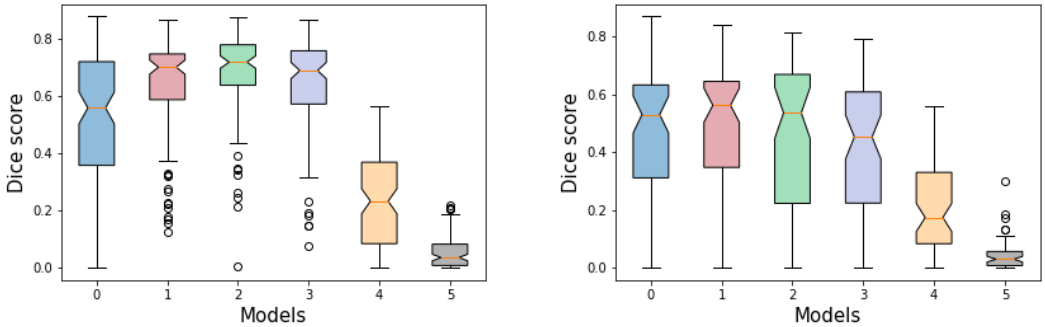
following experiment, where we will evaluate the generalization performance of the best performing models in this experiment, based on the size of the training set.

	Training Dice	Validation Dice
Scratch	0.5334	0.4602
Model 1	0.6389	0.4785
Model 2	0.6748	0.4377
Model 3	0.6406	0.4071
Model 4	0.2371	0.2109
Model 5	0.0556	0.0447

Table 6.2 The average Dice score on the training and validation set for each of our models.

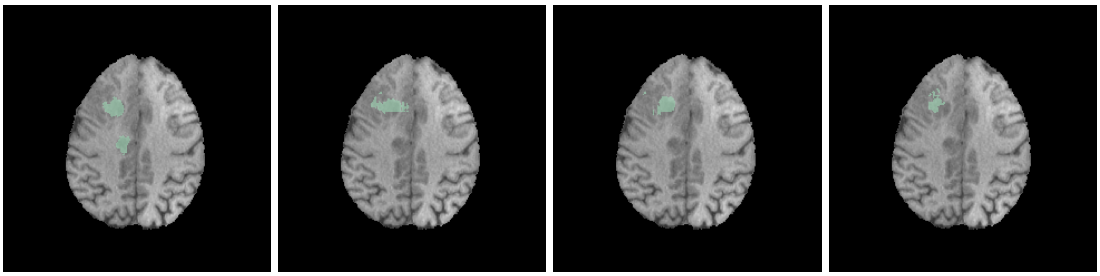
Based on the box plots shown in figure 6.9, we assume the poor segmentation performance in this experiment is mainly caused by not having enough training samples for particular kinds of cases, such as training subject Brats18_CBICA_AQG_1 with multiple tumor cores. The pre-trained models 1, 2, and 3 are the only models that are able to segment this subject with the dice score of 0.1580, 0.2152, and 0.1470 as shown in figure 6.10. Even though the segmentation results are bad, it is interesting to observe the performance of the pre-trained models on this subject. It suggests that the pre-trained models have learned something valuable from the IXI dataset that can be used in this task.

Training subject Brats18_CBICA_AUQ_1 is another example that suggest the same thing. Model 1, 2, and 3 achieves the Dice score of 0.2103, 0.6887, and 0.4989 (see figure 6.11), while the model trained from scratch is not able to segment the tumor core. This observation is also indicated in the box plot of the segmentation results on the training set in figure 6.9a. As we can see, the lower quartiles (e.g., 25% of the lowest values) for model 1,2, and 3 are significantly higher than the model trained from scratch.



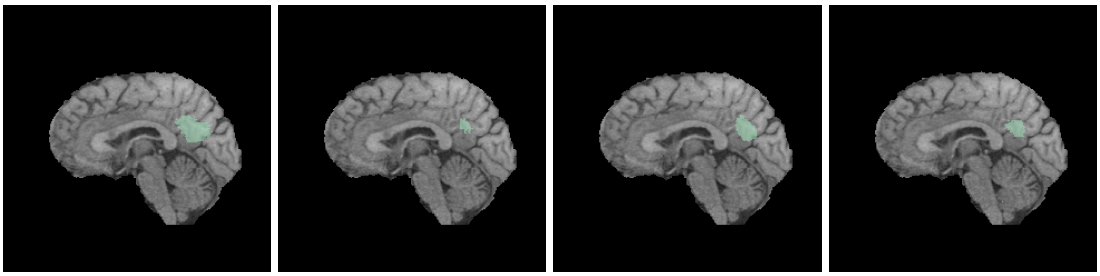
(a) Training Dice score (b) Validation Dice score

Fig. 6.9 Box plot of the Dice scores obtained by each of our models on the training set and the validation set. The dots indicate outliers.



Ground truth label Model 1 Model 2 Model 3

Fig. 6.10 Ground truth and segmentation results for training subject Brats18_CBICA_AQG_1



Ground truth label Model 1 Model 2 Model 3

Fig. 6.11 Ground truth and segmentation results for training subject Brats18_CBICA_AUQ_1

The effect of training set size in the target task

Based on the results and observations from the previous experiment, we decided to utilize three different pre-trained models where layer 1, 2 and 3 were frozen respectively, in addition to a model initialized with random weights. The models with more layers frozen seem to have too low capacity to be trained, as seen from the Dice scores on the training data in table 3.5. In order to alleviate the problem of overfitting as we saw in the previous experiment, we chose to compare the average Dice score of the segmentation results on the training set and the validation set at an iteration where the loss functions started to indicate overfitting. If the average Dice score on the validation set was significantly lower than the average Dice score on the training set, we chose a model from an earlier iteration and evaluated again. This can be seen as a form of early stopping performed manually. However, this approach was very time-consuming and computationally expensive, which made it challenging to find a good balance between overfitting and underfitting. The inference (segmentation) time on each subject was around 17 seconds. For instance, for the smallest training set S1 plus the validation set (e.g., 26 training subjects and 63 validation subjects), the inference could take

up to 25 minutes. Calculation of Dice score on each image came in addition, so it could easily take up to 40 minutes per evaluation.

The segmentation performance on the training, validation, and test sets for each model trained on the four different training cases are presented in tables 6.3, 6.4, 6.5, and 6.6.

	Training Dice	Validation Dice	Test Dice	Number of iterations
Scratch	0.5821	0.3744	0.4261	500
Model 1	0.7500	0.3920	0.4312	500
Model 2	0.7476	0.3780	0.3611	500
Model 3	0.6867	0.3214	0.3388	500

Table 6.3 Segmentation results obtained using training set S1

	Training Dice	Validation Dice	Test Dice	Number of iterations
Scratch	0.5000	0.3810	0.4526	1000
Model 1	0.6421	0.4169	0.4819	500
Model 2	0.6319	0.4143	0.4708	500
Model 3	0.6178	0.3887	0.4677	500

Table 6.4 Segmentation results obtained using training set S2

	Training Dice	Validation Dice	Test Dice	Number of iterations
Scratch	0.4623	0.3712	0.4175	1000
Model 1	0.6390	0.3900	0.4540	1000
Model 2	0.6567	0.4257	0.4918	1000
Model 3	0.6040	0.3993	0.4675	1000

Table 6.5 Segmentation results obtained using training set S3

	Training Dice	Validation Dice	Test Dice	Number of iterations
Scratch	0.5924	0.5175	0.5103	5000
Model 1	0.6830	0.4450	0.5565	1500
Model 2	0.6437	0.4088	0.4550	1500
Model 3	0.6439	0.4382	0.5019	1500

Table 6.6 Segmentation results on the test set obtained using training set S4

Even though the segmentation performance is still poor (discussed in the previous experiment), we can see some interesting indications in the obtained results. In all the training cases, the best generalization error is achieved by one of the pre-trained models. In addition, in case S2 and S4 we can see that the pre-trained models are selected from an earlier training iteration compared to the model trained from scratch, which means that the training time is decreased for these models. Note that the training time is dominated by the (CPU-based) sampling of class-weighted windows. The actual updating of network weights through gradient descent and backpropagation happening on the GPU is a relatively small portion of the process. For other sampling configurations, the time saved per iteration by reducing the number of parameters in the network through freezing would be much greater, further increasing the value of transferring from pre-trained networks.

6.4 Discussion

In this study, we presented the potential of using transfer learning in 3D CNN for volumetric medical images, by first training a network on the task of segmenting the left and right hippocampus on a large-scale MRI dataset of healthy patients, and fine-tuned this network to segment tumor core in a smaller MRI dataset. Although the overall segmentation performance of our models was bad, we got some interesting suggestions. The obtained results indicate that our approach to transfer learning can decrease the training time and might improve the segmentation performance compared to a model trained from scratch. We expect that our findings would have been more significant with better preprocessing techniques for splitting the data into training, validation, and test sets. We further believe that using modern best practices, such as decreasing the learning rate during the training would give better results. But this is beyond the scope of this investigation since it would require a lot of experimentation to find a good learning rate schedule for each mode. In addition, note that using all the MRI sequences in the brain tumor dataset would also lead to better segmentation,

but would not be as suitable for our transfer experiments from the T1 + T2 images of IXI. We have also experimented with using different CNN architectures and found better results using e.g. DenseVNet [32]. However, these architectures are not as well-suited for our transfer learning experiments.

Our source code for this study is publicly available at: https://github.com/skaliy/niftynet_brain. Future transfer learning investigations will be added to this repository.

Chapter 7

Conclusion and future work

In this thesis, we described a new approach for transfer learning in 2D CNN for medical images. We proposed a technique that involves first training a network pre-trained on the ImageNet dataset to the medical domain, before fine-tuning this network for the primary task in the medical domain. Our obtained results indicated that this approach to transfer learning leads to less need for training data.

In addition, we investigated the potential of transfer learning in 3D CNN for volumetric medical images, an area that is not well-studied. Although the obtained segmentation performance was poor due to the difficulty in the target dataset and limitations in the framework, we got some interesting indications. Our findings suggest that the training time can be reduced and might improve performance slightly as well. We expect that better preprocessing methods and using state-of-the-art techniques would give better results.

In the future work, we would like to develop further on this study and create preprocessing methods and utilize state-of-the-art techniques that can be used to improve the segmentation performance of the networks in various volumetric medical image segmentation tasks. In addition, we want to create a sort of ImageNet dataset for 3D MRI, and pre-train a couple of state-of-the-art models on this dataset than can be used in local projects at Haukeland University Hospital.

References

[ten] TensorFlow Core. https://www.tensorflow.org/alpha. Accessed: 2019-05-13.

[2] Abadi, M., Agarwal, A., Barham, P., et al. (2015). TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Software available from tensorflow.org.

[3] Anand, K. S. and Dhikav, V. (2012). Hippocampus in health and disease: An overview. *Annals of Indian Academy of Neurology*, 15(4):239.

[4] anonymous (2017). Battle of the brains. https://www.economist.com/business/2017/12/07/google-leads-in-the-race-to-dominate-artificial-intelligence.

[5] Arulkumaran, K., Cully, A., and Togelius, J. (2019). AlphaStar: An Evolutionary Computation Perspective. *arXiv preprint arXiv:1902.01724*.

[6] Awadalla, H. H., Aue, A., Chen, C., Chowdhary, V., Clark, J., Federmann, C., Huang, X., Junczys-Dowmunt, M., Lewis, W., Li, M., et al. (2018). Achieving human parity on automatic chinese to english news translation.

[7] Bakas, S., Akbari, H., Sotiras, A., Bilello, M., Rozycki, M., Kirby, J., Freymann, J., Farahani, K., and Davatzikos, C. (2017a). Segmentation labels and radiomic features for the pre-operative scans of the TCGA-GBM collection. The Cancer Imaging Archive (2017).

[8] Bakas, S., Akbari, H., Sotiras, A., Bilello, M., Rozycki, M., Kirby, J., Freymann, J., Farahani, K., and Davatzikos, C. (2017b). Segmentation labels and radiomic features for the pre-operative scans of the TCGA-LGG collection. *The Cancer Imaging Archive*, 286.

[9] Bakas, S., Akbari, H., Sotiras, A., Bilello, M., Rozycki, M., Kirby, J. S., Freymann, J. B., Farahani, K., and Davatzikos, C. (2017c). Advancing the cancer genome atlas glioma MRI collections with expert segmentation labels and radiomic features. *Scientific data*, 4:170117.

[10] Barreira, C. M., Bouslama, M., Haussen, D. C., Grossberg, J. A., Baxter, B., Devlin, T., Frankel, M., and Nogueira, R. G. (2018). Abstract WP61: Automated Large Artery Occlusion Detection IN Stroke Imaging-ALADIN Study. *Stroke*, 49(Suppl_1):AWP61–AWP61.

[11] Bell, J. (2014a). *Machine learning: hands-on for developers and technical professionals*. John Wiley & Sons.

[12] Bell, W. (2014b). *Machine learning: for developers and technical professionals*. Wiley.

[13] Bonnin, R. (2017). *Machine Learning for Developers*. Packt Publishing.

[14] Brett, M., Markiewicz, C. J., Hanke, M., et al. (2019). nipy/nibabel: 2.4.1.

[15] Carmichael, O. T., Aizenstein, H. A., Davis, S. W., Becker, J. T., Thompson, P. M., Meltzer, C. C., and Liu, Y. (2005). Atlas-based hippocampus segmentation in Alzheimer's disease and mild cognitive impairment. *Neuroimage*, 27(4):979–990.

[16] Carmo, D., Silva, B., Yasuda, C., Rittner, L., and Lotufo, R. (2019). Extended 2D Volumetric Consensus Hippocampus Segmentation. *arXiv preprint arXiv:1902.04487*.

[17] CHARU, C. A. (2019). *NEURAL NETWORKS AND DEEP LEARNING: A Textbook*. SPRINGER.

[18] Chen, S., Ma, K., and Zheng, Y. (2019). Med3D: Transfer Learning for 3D Medical Image Analysis. *arXiv preprint arXiv:1904.00625*.

[19] Chen, W., Li, S., Jia, F., and Zhang, X. (2011). Segmentation of hippocampus based on ROI atlas registration. In *2011 IEEE International Symposium on it in Medicine and Education*, volume 1, pages 226–230. IEEE.

[20] Collobert, R., Kavukcuoglu, K., and Farabet, C. (2011). Torch7: A matlab-like environment for machine learning. Technical report.

[21] Crimi, A., Bakas, S., Kuijf, H., Keyvan, F., Reyes, M., and van, W. T. (2019). *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries: Fourth International Workshop, BrainLes 2018, Held in Conjunction with MICCAI 2018, Granada, Spain, September 16, 2018, Revised Selected Papers, Part II*, volume 11384. Springer.

[22] CrowdFlower (2016). Data Science Report. Technical report.

[23] Cui, Y., Song, Y., Sun, C., Howard, A., and Belongie, S. (2018). Large scale fine-grained categorization and domain-specific transfer learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4109–4118.

[24] Dai, J., He, K., and Sun, J. (2016). Instance-aware semantic segmentation via multi-task network cascades. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3150–3158.

[25] Esteva, A., Kuprel, B., Novoa, R. A., Ko, J., Swetter, S. M., Blau, H. M., and Thrun, S. (2017). Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115.

[26] Esteva, A., Robicquet, A., Ramsundar, B., Kuleshov, V., DePristo, M., Chou, K., Cui, C., Corrado, G., Thrun, S., and Dean, J. (2019). A guide to deep learning in healthcare. *Nature medicine*, 25(1):24.

[27] Fawcett, T. (2006). An introduction to ROC analysis. *Pattern recognition letters*, 27(8):861–874.

[28] Fischl, B. (2012). FreeSurfer. *Neuroimage*, 62(2):774–781.

[29] Frisoni, G. B., Fox, N. C., Jack Jr, C. R., Scheltens, P., and Thompson, P. M. (2010). The clinical use of structural MRI in Alzheimer disease. *Nature Reviews Neurology*, 6(2):67.

[30] Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202.

[31] Géron, A. (2017). *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, Inc.

[32] Gibson, E., Giganti, F., Hu, Y., Bonmati, E., Bandula, S., Gurusamy, K., Davidson, B., Pereira, S. P., Clarkson, M. J., and Barratt, D. C. (2018a). Automatic multi-organ segmentation on abdominal CT with dense v-networks. *IEEE transactions on medical imaging*, 37(8):1822–1834.

[33] Gibson, E., Li, W., Sudre, C., Fidon, L., Shakir, D. I., Wang, G., Eaton-Rosen, Z., Gray, R., Doel, T., Hu, Y., et al. (2018b). NiftyNet: a deep-learning platform for medical imaging. *Computer methods and programs in biomedicine*, 158:113–122.

[34] Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323.

[Google] Google. Cloud Tensor Processing Units (TPUs). https://cloud.google.com/tpu/docs/tpus. Accessed: 2019-05-13.

[Group] Group, B. I. A. IXI Dataset. https://brain-development.org/ixi-dataset. Accessed: 2019-05-12.

[37] Haenssle, H., Fink, C., Schneiderbauer, R., Toberer, F., Buhl, T., Blum, A., Kalloo, A., Hassen, A. B. H., Thomas, L., Enk, A., et al. (2018). Man against machine: diagnostic performance of a deep learning convolutional neural network for dermoscopic melanoma recognition in comparison to 58 dermatologists. *Annals of Oncology*, 29(8):1836–1842.

[38] Haldorai, A. and Ramu, A. (2018). *Cognitive Social Mining Applications in Data Analytics and Forensics*. IGI Global.

[39] Hansen, H. (2018). Germany plans 3 billion in AI investment: government paper. https://www.reuters.com/article/us-germany-intelligence/germany-plans-3-billion-in-ai-investment-government-paper-idUSKCN1NI1AP.

[40] He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2017). Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969.

[41] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.

[42] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[43] Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554.

[44] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

[45] Howard, J. et al. (2018). fastai. https://github.com/fastai/fastai.

[46] Howard, J. and Ruder, S. (2018). Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*.

[47] Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708.

[48] Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3):90–95.

[49] Iglesias, J. E. and Sabuncu, M. R. (2015). Multi-atlas segmentation of biomedical images: a survey. *Medical image analysis*, 24(1):205–219.

[50] Irvin, J., Rajpurkar, P., Ko, M., Yu, Y., Ciurea-Ilcus, S., Chute, C., Marklund, H., Haghgoo, B., Ball, R., Shpanskaya, K., et al. (2019). Chexpert: A large chest radiograph dataset with uncertainty labels and expert comparison. In *Thirty-Third AAAI Conference on Artificial Intelligence*.

[51] Joiner, I. A. (2018). *Emerging Library Technologies: It's Not Just for Geeks*. Chandos Publishing.

[52] Kaliyugarasan, S. and Lundervold, A. S. (2018). Transfer learning in medical images: a case study.

[53] Kensert, A., Harrison, P. J., and Spjuth, O. (2018). Transfer learning with deep convolutional neural networks for classifying cellular morphological changes. *SLAS DISCOVERY: Advancing Life Sciences R&D*, page 2472555218818756.

[54] Knight, W. (2018). MIT has just announced a $1 billion plan to create a new college for AI. https://www.technologyreview.com/the-download/612293/mit-has-just-announced-a-1-billion-plan-to-create-a-new-college-for-ai/.

[55] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

[56] Krupinski, E. A., Berbaum, K. S., Caldwell, R. T., Schartz, K. M., and Kim, J. (2010). Long radiology workdays reduce detection and accommodation accuracy. *Journal of the American College of Radiology*, 7(9):698–704.

[57] Krupinski, E. A., Berbaum, K. S., Caldwell, R. T., Schartz, K. M., Madsen, M. T., and Kramer, D. J. (2012). Do long radiology workdays affect nodule detection in dynamic CT interpretation? *Journal of the American College of Radiology*, 9(3):191–198.

[58] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551.

[59] LeNail, A. (2019). NN-SVG: Publication-Ready Neural Network Architecture Schematics. *The Journal of Open Source Software*, 4:747.

[60] Li, W., Wang, G., Fidon, L., Ourselin, S., Cardoso, M. J., and Vercauteren, T. (2017). On the compactness, efficiency, and representation of 3D convolutional networks: brain parcellation as a pretext task. In *International Conference on Information Processing in Medical Imaging*, pages 348–360. Springer.

[61] Litjens, G., Kooi, T., Bejnordi, B. E., Setio, A. A. A., Ciompi, F., Ghafoorian, M., Van Der Laak, J. A., Van Ginneken, B., and Sánchez, C. I. (2017). A survey on deep learning in medical image analysis. *Medical image analysis*, 42:60–88.

[62] Lundervold, A. S. and Lundervold, A. (2019). An overview of deep learning in medical imaging focusing on MRI. *Zeitschrift für Medizinische Physik*, 29(2):102 – 127.

[63] Lundervold, A. S., Rørvik, J., and Lundervold, A. (2017). Fast semi-supervised segmentation of the kidneys in DCE-MRI using convolutional neural networks and transfer learning.

[64] Maglogiannis, I. G. (2007). *Emerging artificial intelligence applications in computer engineering: real word ai systems with applications in ehealth, hci, information retrieval and pervasive technologies*, volume 160. Ios Press.

[65] Mahajan, D., Girshick, R., Ramanathan, V., He, K., Paluri, M., Li, Y., Bharambe, A., and van der Maaten, L. (2018). Exploring the limits of weakly supervised pretraining. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 181–196.

[66] Menze, B. H., Jakab, A., Bauer, S., Kalpathy-Cramer, J., Farahani, K., Kirby, J., Burren, Y., Porz, N., Slotboom, J., Wiest, R., et al. (2014). The multimodal brain tumor image segmentation benchmark (BRATS). *IEEE transactions on medical imaging*, 34(10):1993–2024.

[67] Mishra, P. (2019). Introduction to PyTorch, Tensors, and Tensor Operations. In *PyTorch Recipes*, pages 1–27. Springer.

[68] Ng, A. (2013). Machine Learning and AI via Brain simulations. *Andrew Ng*.

[69] NVIDIA (2015). GPU-Based Deep Learning Inference: A Performance and Power Analysis. *Whitepaper, November*.

[70] Oakden-Rayner, L. (2017). Exploring the ChestXray14 dataset: problems.

[71] Oliphant, T. E. (2006). *A guide to NumPy*, volume 1. Trelgol Publishing USA.

[72] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in PyTorch. In *NIPS-W*.

[73] Paszke, A., Gross, S., Chintala, S., and et al., C. (2016). pytorch. ://github.com/pytorch/pytorch.

[74] Pedersen, K. (2016). 7000 undersøkelser i kø hos røntgenlegene. ://www.bt.no/nyheter/lokalt/i/eLBba/7000-undersokelser-i-ko-hos-rontgenlegene.

[75] Peng, T. and Sarazen, M. (2018). Andrew Ng Warns of Centralized AI Power. :/www.syncedreview.com/2018/05/29/andrew-ng-warns-of-centralized-ai-power/.

[76] Pipitone, J., Park, M. T. M., Winterburn, J., Lett, T. A., Lerch, J. P., Pruessner, J. C., Lepage, M., Voineskos, A. N., Chakravarty, M. M., Initiative, A. D. N., et al. (2014). Multi-atlas segmentation of the whole hippocampus and subfields using multiple automatically generated templates. *Neuroimage*, 101:494–512.

[77] Rajpurkar, P., Irvin, J., Bagul, A., Ding, D., Duan, T., Mehta, H., Yang, B., Zhu, K., Laird, D., Ball, R. L., et al. (2017a). Mura: Large dataset for abnormality detection in musculoskeletal radiographs. *arXiv preprint arXiv:1712.06957*.

[78] Rajpurkar, P., Irvin, J., Zhu, K., Yang, B., Mehta, H., Duan, T., Ding, D., Bagul, A., Langlotz, C., Shpanskaya, K., et al. (2017b). Chexnet: Radiologist-level pneumonia detection on chest x-rays with deep learning. *arXiv preprint arXiv:1711.05225*.

[79] Ravishankar, H., Sudhakar, P., Venkataramani, R., Thiruvenkadam, S., Annangi, P., Babu, N., and Vaidya, V. (2017). Understanding the mechanisms of deep transfer learning for medical images. *arXiv preprint arXiv:1704.06040*.

[80] Rosemain, M. and Rose, M. (2018). France to spend $1.8 billion on AI to compete with U.S., China. https://www.reuters.com/article/us-france-tech/france-to-spend-1-8-billion-on-ai-to-compete-with-u-s-china-idUSKBN1H51XP.

[81] Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

[82] Ruder, S. (2017). Transfer Learning - Machine Learning's Next Frontier. Accessed: 2019-05-25.

[83] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015). Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252.

[84] Shin, H.-C., Roth, H. R., Gao, M., Lu, L., Xu, Z., Nogues, I., Yao, J., Mollura, D., and Summers, R. M. (2016). Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset characteristics and transfer learning. *IEEE transactions on medical imaging*, 35(5):1285–1298.

[85] Shoham, Y., Perrault, R., Brynjolfsson, E., Clark, J., Manyika, J., Niebles, J. C., Lyons, T., Etchemendy, J., Grosz, B., and Bauer, Z. (2018). The AI Index 2018 Annual Report. Technical report, AI Index Steering Committee and Human-Centered AI Initiative and Stanford University.

[86] Shukla, A. (2010). *Intelligent Medical Technologies and Biomedical Engineering: Tools and Applications: Tools and Applications*. Igi Global.

[87] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2017a). Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*.

[88] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017b). Mastering the game of go without human knowledge. *Nature*, 550(7676):354.

[89] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

[90] Storstein, A., Helseth, E., Johannesen, T. B., Schellhorn, T., Mørk, S., and van Helvoirt, R. (2011). Høygradige gliomer hos voksne. *Tidsskrift for Den norske legeforening*.

[91] Taha, A. A. and Hanbury, A. (2015). Metrics for evaluating 3D medical image segmentation: analysis, selection, and tool. *BMC medical imaging*, 15(1):29.

[92] Tajbakhsh, N., Shin, J. Y., Gurudu, S. R., Hurst, R. T., Kendall, C. B., Gotway, M. B., and Liang, J. (2016). Convolutional neural networks for medical image analysis: Full training or fine tuning? *IEEE transactions on medical imaging*, 35(5):1299–1312.

[93] Thyreau, B., Sato, K., Fukuda, H., and Taki, Y. (2018). Segmentation of the hippocampus by transferring algorithmic knowledge for large cohort processing. *Medical image analysis*, 43:214–228.

[94] Topol, E. J. (2019). High-performance medicine: the convergence of human and artificial intelligence. *Nature medicine*, 25(1):44.

[95] van der Plas, A. (2015). MRI Technique. Accessed: 2019-05-24.

[96] Wachinger, C., Reuter, M., and Klein, T. (2018). DeepNAT: Deep convolutional neural network for segmenting neuroanatomy. *NeuroImage*, 170:434–445.

[97] Wan, X., Liu, J., Cheung, W. K., and Tong, T. (2014). Learning to improve medical decision making from imbalanced data without a priori cost. *BMC medical informatics and decision making*, 14(1):111.

[98] Wang, F.-Y. (2012). A big-data perspective on AI: Newton, Merton, and analytics intelligence. *IEEE Intelligent Systems*, 27(5):2–4.

[99] Wang, X., Peng, Y., Lu, L., Lu, Z., Bagheri, M., and Summers, R. M. (2017). Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2097–2106.

[100] Xie, Z. and Gillies, D. (2018). Near Real-time Hippocampus Segmentation Using Patch-based Canonical Neural Network. *arXiv preprint arXiv:1807.05482*.

[101] Xu, Y., Géraud, T., and Bloch, I. (2017). From neonatal to adult brain MR image segmentation in a few seconds using 3D-like fully convolutional network and transfer learning. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 4417–4421. IEEE.

[102] Yang, Y., Yan, L.-F., Zhang, X., Han, Y., Nan, H.-Y., Hu, Y.-C., Hu, B., Yan, S.-L., Zhang, J., Cheng, D.-L., et al. (2018). Glioma grading on conventional MR images: a deep learning study with transfer learning. *Frontiers in neuroscience*, 12.

[103] Yoshida, K. and Lee, M. (2018). *Knowledge Management and Acquisition for Intelligent Systems*, volume 11016. Springer.

[104] Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328.

[105] Yu, Z., Jiang, X., Zhou, F., Qin, J., Ni, D., Chen, S., Lei, B., and Wang, T. (2019). Melanoma Recognition in Dermoscopy Images via Aggregated Deep Convolutional Features. *IEEE Transactions on Biomedical Engineering*, 66(4):1006–1016.

[106] yWorks GmbH (n.d.). yED.

[107] Zafar, I., Tzanidou, G., Burton, R., Patel, N., and Araujo, L. (2018). *Hands-on Convolutional Neural Networks with TensorFlow: Solve Computer Vision Problems with Modeling in TensorFlow and Python*. Packt Publishing Ltd.

[108] Zeiler, M. D. and Fergus, R. (2013). Visualizing and understanding convolutional networks (2013). *arXiv preprint arXiv:1311.2901*.

[109] Zheng, A. (2015). Evaluating Machine Learning Models. Technical report.