FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

ASSIGNMENT OF BACHELOR'S THESIS

Title:	Detecting abnormalities in X-Ray images using Neural Networks
Student:	Jan Rudolf
Supervisor:	Ing. Jakub Žitný
Study Programme:	Informatics
Study Branch:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	Until the end of winter semester 2021/22

Instructions

Research current state-of-the-art techniques that are used for prediction, classification and segmentation tasks in the medical imaging domain, focus on X-Ray images.
Implement your own prototype model that will work on one of the open datasets provided by the supervisor.
Compare the performance of your model with reference results from literature or existing models and discuss the pros and cons.
Publish your prototype code and make sure your results are reproducible.

References

Will be provided by the supervisor.

Ing. Karel Klouda, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague February 22, 2020

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Detecting abnormalities in X-Ray images using Neural Networks

Jan Rudolf

Department of Applied Mathematics
Supervisor: Ing. Jakub Žitný

June 4, 2020

Acknowledgements

I would like to thank my family for their support during my work on the thesis.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on June 4, 2020 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Rudolf, Jan. *Detecting abnormalities in X-Ray images using Neural Networks.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2020.

Abstrakt

Tato práce pojednává o základech strojového učení a dále se zaměřuje na moderní techniky strojového vidění v oblasti lékařských zobrazovacích metod. Součástí práce je implementace prototypu pro klasifikaci rentgenových snímku z MURA datasetu.

Klíčová slova Strojové vidění, lékařské zobrazovací metody, prototyp, MURA

Abstract

This work apprises the reader with the basics of machine learning. Furthermore researches the current state-of-the-art approaches used in machine learning in medical imaging. As a part of the theis, the prototype for classification of MURA dataset is trained.

Keywords Computer vision, medical imaging, Prototype, MURA

Contents

List of Figures

List of Tables

Introduction

Deep artificial neural networks have gained great popularity in recent years. Advancements of recent years in computational hardware, big data collection and learning algorithms allowed their development which contributed to machine learning spread into our daily life. Also many industries have adopted machine learning to improve quality or efficiency or reduce personnel costs.

Healthcare industry is not an exception, we can see growth of machine learning applications used for prediction of diseases and computer-aided diagnosis. Advances have been made in tasks of breast cancer detection, prediction of eye diseases, early detection of acute diseases and more. Nevertheless, deployment of machine learning in such a delicate industry, where a mistake might cost the highest price, raises a lot of ethical and legal questions[1, 2, 3].

Determining whether X-Ray study is normal or abnormal is crucial for patients' health and requires an expert to carry out its analysis. Application of deep learning models in Computer aided diagnosis can reduce risk of error caused by human expert's mistake.

This thesis researches current state-of-the-art machine learning techniques used in medical imaging. Theoretical part of this thesis introduces the reader to techniques of machine learning, and to its applications in medical imaging domain. Practical part describes training and results of my own models made for classification of MURA dataset samples. The MURA dataset comprises 14,863 X-Ray studies of seven different parts of upper the limb classified as normal or abnormal [4].

Goal of thesis

There are four goals of the thesis.

Research of current state-of-the art techniques that are used for prediction, classification and segmentation tasks in medical imaging domain, focus on X-Ray images.

Implementation of prototype model for classification of MURA[4] dataset. Performance of the model will be compared with other models trained on the same dataset. Thesis will contain discussion over their attributes, pros and cons.

Compare the performance of your model with reference results from literature or existing models and discuss the pros and cons.

Publish your prototype code and make sure your results are reproducible.

Chapter 1

Machine learning

The goal of Machine learning (ML) is development of algorithms which are being improved through experience. Machine learning is often applied in cases where conventional algorithms are very hard to implement like computer vision, spam filtering, etc. [5].

ML algorithms extracts patterns from the data and projects them into mathematical model (trains model). Trained model makes predictions of input data based on similarities and differences between input data and the dataset.

1.1 Supervised × Unsupervised learning

ML algorithms are divided into several classes by the type of task they are meant to solve. Most common classes are supervised and unsupervised learning.

1.1.1 Supervised learning

Supervised learning process is done with labeled data. It means the dataset consists not only samples, but also desired output (label) for each sample.

In this thesis samples will be referred as X, labels as Y, predictions as $\hat{Y}$ and Loss (Cost) function as $C : (Y, \hat{Y}) \rightarrow \mathbb{R}$.

The relation between X and Y can be expressed as a mapping function $f : X \rightarrow Y$. Loss function is used as a measure of difference between labels and predictions. Task of supervised learning is to infer a mapping function $\hat{f}$ which $\hat{f} \approx f$, not only for the dataset but for all inputs relevant to the problem. That is done by minimization of loss function C[6].

Classification represents methods which train a model to make predictions of input sample's belonging to one of discrete classes. For example, determining the type of patient's diabetes based on his anamnesis is a classification problem. Well known methods for classification are: Decision tree, Logistic regression.

Regression represents methods which train a model to make non-discrete predictions. One of regression problems is determining risk of infant's suffering from autism based on it's EEG or/and his parents anamnesis [7]. Well known methods of regression are: Linear regression, Polynomial regression.

1.1.2 Unsupervised learning

Unlike Supervised learning, Unsupervised learning dataset does not contain labels. Since there are no labels there is also no mapping function f to approximate. In general, task of Unsupervised learning algorithms is to analyse patterns in dataset. Since patterns are known, algorithm can determine whether a sample fits some of patterns or is an outlier (anomaly) [8, 6].

Clustering or Cluster analysis represents methods used for creation groups in the dataset based on similarities between samples. The principle of Clustering is often used in recommendation systems where a sample of dataset represents user's ratings of products. Method is trying to find groups of similar users (users with similar ratings). Assuming that users in a group have similar desires, the system recommends user products which were favoured by other users in the group. Well known examples are Hierarchical clustering, K-means [6].

1.2 Testing and Validation

Once the model is trained, it is required to evaluate it's performance.

Let's assume following example. We want to predict Alice's test score based on the time she takes to study. We have got a labeled dataset consisting of information about students who already took the test. Using Polynomial regression (PR) we've trained a model whereas least mean squared roots loss is used. The value loss function is almost zero. We assume that the model is well trained and we can use it to make the prediction. Although the loss function was almost zero, the prediction model made is a big negative number. Therefore the prediction is not sane.

The problem is that the model is **overfitted**. Instead of finding general patterns from the field of the problem, the model was learnt to predict well only data from the dataset. Thus being said the model performs well only while evaluating data from training dataset, while predictions of other inputs are often very poor as it is displayed on figure 1.1 left.

Common practice is splitting dataset into **training dataset** and **testing dataset**. While the model is trained on the training dataset, the testing dataset is used solely for it's evaluation. Because the model was never trained on the testing set, loss function calculated over predictions of the testing set, serves a true evaluation of model's ability to predict general cases (if the dataset is a proper generalized specimen from the field of the problem).

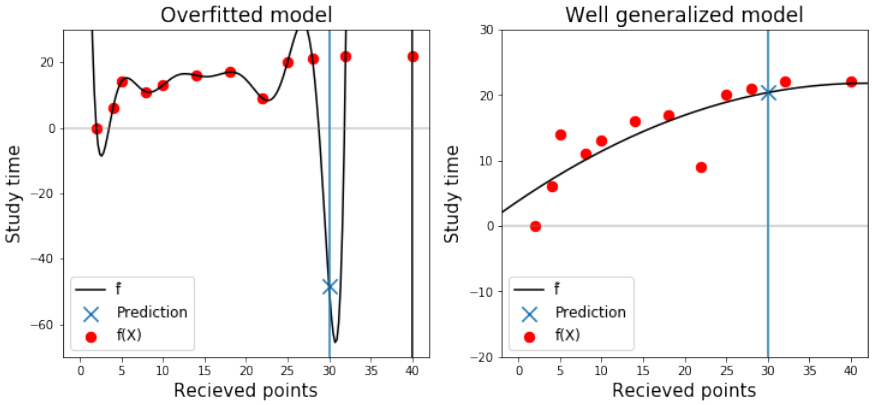

Figure 1.1: Graphs displays behaviour of the mapping function $\hat{f}$ of overfitted model (left) and well generalized model (right) based on training dataset. Note that datasets on both images are the same, but the scales are different for representational purpose

Difference between errors of training and testing datasets is called a generalization gap. Two conclusions can be made, based on the generalization gap . (i) If the generalization gap is small the model well generalizes the problem domain. (ii) If test loss is considerably bigger than train loss the model suffers from overfitting [6].

Let's return to the previous example. We randomly split our samples into train and test datasets in ratio 3 to 1. We have trained the model in same the fashion as before but using solely the train dataset. Afterwards, we've evaluated losses of both datasets. Training error is again ≈ 0, while test error is very big. It is very clear that the model is overfitted. Training process of Polynomial regression has a parameter n which states the degree of polynom used in the PR model. In previous training we used $n = 13$ which caused heavy overfitting. We will address overfitting by adjusting $n = 2$.

Model trained with $n = 2$ has no longer train loss ≈ 0. Although, the model is considered better because testing loss is much lower. The model works well even for samples not included in the training process (model is generalized). Also testing loss is close to training loss (the decision whether two losses are close to each other depends on the domain problem), thereby the overfitting problem is fixed. Yet, the example contains a new problem. Value of parameter n was determined by testing loss. Doing so, the model is no longer trained on testing data only, therefore testing loss is not objective.

Parameters like n which are set before training are called **Hyperparameters (HP)**. Two basic techniques are used to evaluate the quality of hyperparameters.

The basic one is a simple split of the dataset into three sets. Next to train and test sets, the **validation set** is created, solely for evaluation of

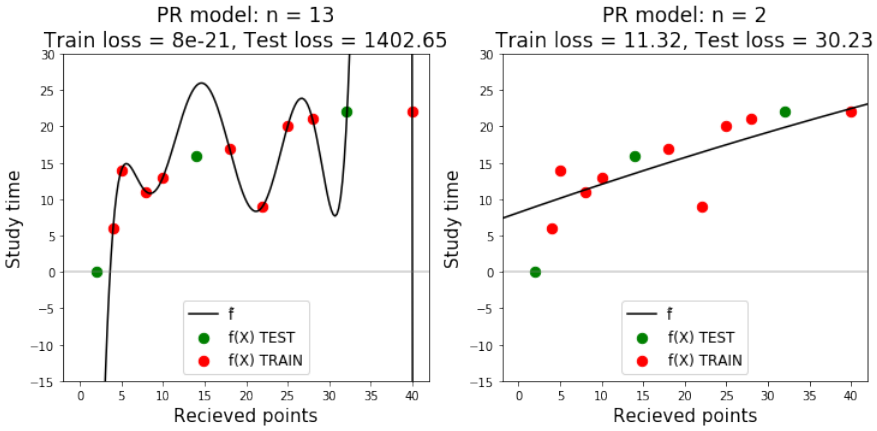

Figure 1.2: Graphs display behaviour of the mapping function $\hat{f}$ based on training dataset and parameter n

hyperparameter configuration quality.

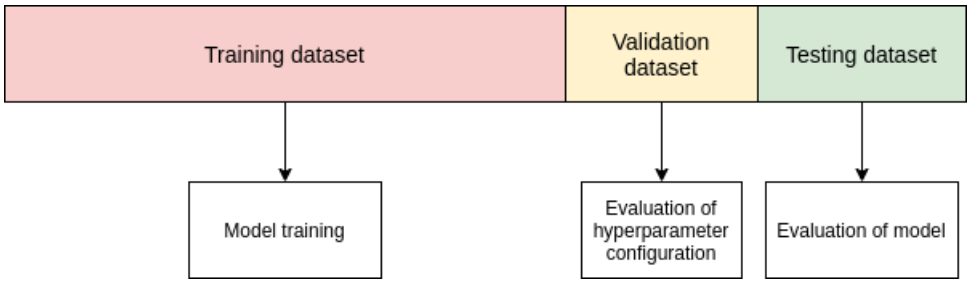

Figure 1.3: Segmentation of a dataset with individual segments purpose

The other approach is called **k-fold cross-validation**. The training set is randomly split into k equal subsets called folds. The validation loss is expressed as an average of k different estimates. Each estimate is done by evaluation of the model on one fold, while remaining folds are used for training. Approach when k = number of samples is known as leave-one-out cross-validation [6].

The advantage of cross-validation is, all data are gradually used for validation thus lowers chance of validation loss being based on misfortune selection of validation samples which doesn't generalize the dataset properly. Disadvantage is, the training process has to be performed k times. Repeated training represents a problem for methods with a long training process where high k will cause even longer training and smaller k drastically lowers the number of samples.

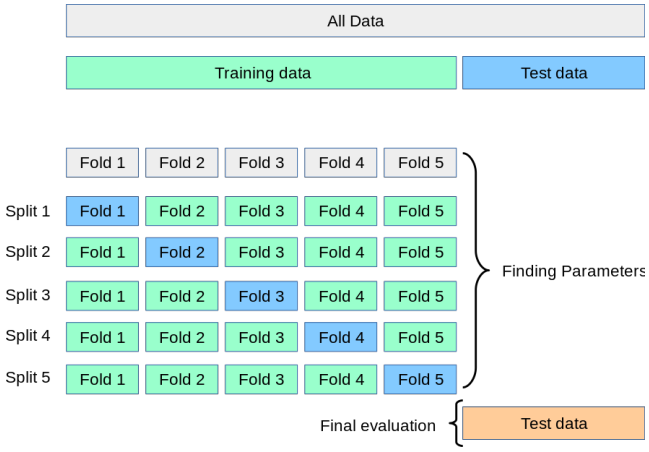

Figure 1.4: Illustration of cross-validation folds [9]

1.3 Hyperparameter optimization

Selection of hyperparameters is crucial for the training process. The process in which hyperparameters are selected is called **Hyperparameter optimization** (HPO). HPO algorithm determines which configurations of hyperparameters should be tried. For those configurations the model is learnt on a training dataset and their evaluated on a separate data samples.

The space of all hyperparameter configurations is called **Hyperparameter space**, where each hyperparameter represents one dimension. There are several algorithms used for hyperparameter optimization.

Grid search is an algorithm used for full exploration of Hyperparameter space. The space is represented as a grid where each column in the grid represents one configuration of hyperparameters. The algorithm explores every single column. Grid search is usually used for optimization of problems where single trial is computationally cheap and/or where are few configurations to try.

Random search evaluates predefined number of random hyperparameter configurations. Although it is a primitive way of hyperparameter optimization, it often performs well for problems with an expensive trial where it is not possible to do full exploration of Hyperparameter space using Grid search.

Hyperband is the state-of-the-art algorithm for hyperparameter optimization. It extends the idea of the Successive halving and addresses its problematic need of a priory selection of parameters (so-called ”n versus B/n” problem) [10].

Successive halving uniformly allocates budget B (might be time, epochs, etc.) among n random hyperparameter configurations used for iterative training methods. Once the budget is depleted, half of configurations are thrown away and the budget is redistributed among remaining configurations. This is

iteratively done until only one configuration remains. The problem is that for fixed budget B it is not a priori clear whether should be used big n (many configurations with small budget) or small n (few configurations with big budget) [10].

Hyperband runs the successive halving algorithm for more times, with different n for each iteration. It accepts parameters R (maximum amount of resource per configuration) and η (controls portion of discarded configurations in each iteration of Successive halving) [10]. See Hyperband original paper [10] for further details.

1.4 Data Augmentation

Data augmentation is a technique used for production of more training samples. Based on existing data the technique derives more data samples which might be different in properties minor for the task while keeping relevant key features present. Additional samples can lead to improvement of model generalization. Data augmentation is widely, but not exclusively used in Computer vision. This section is dedicated to image data augmentation.

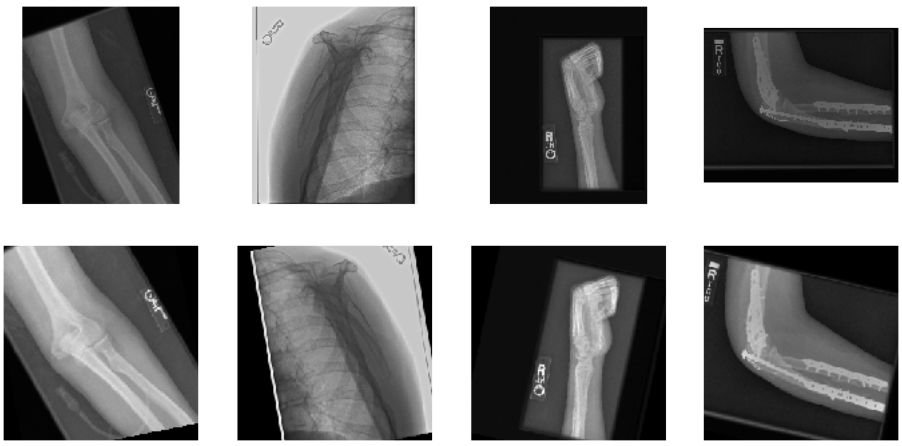

Figure 1.5: Example of data augmentation. First row displays raw images from the MURA dataset [4]. Second row shows images after preprocessing and augmentation (The used augmentation technique is described in Section 2.2.2 as strong augmentation)

Rotation is probably the most commonly used augmentation technique. Rotating images in relevant range makes model rotational invariant - different angle of features does not affect model outcome. Shifting image makes model positional invariant. Flipping along axis is also very useful, since it creates very different image without a risk of key features loosing. Zoom provides feature size independence and shear can help with features recorded in different angles.

Augmentations are not limited only to geometric transformations. There are operations with colors like brightness shift or channel shifts. Another used technique is adding noise to image. These are commonly used techniques, nevertheless any image operation can be used as long as it does no harm to a relevant features.

There are two approaches to augmentation called online and offline. Using online augmentation samples are augmented during training process, requiring more computational resources, while offline augmentation uses samples augmented before training began and saved, requiring extra storage capacity [11].

Although the data augmentation is meant to create more training samples, it can be also used for validation/testing data. This application adds more robustness to model since augmenting test samples makes them "closer" to augmented samples from the training dataset. Yet augmenting input comes with a trade-off in the speed of the model, therefore it can not be effectively used in real-time systems, like autonomous cars where the reaction time is required to be as low as possible [11].

1.5 Artificial neural networks

Artificial neural networks (ANN) represents a machine learning model based on the idea of the neural network of an animal's brain. In their early days Artificial neural networks raised a big wave of enthusiasm, since it presents a powerful model which can be used for a wide selection of problems. Yet, the lack of a good training algorithm and big computational power did not allow effective use of large neural networks and the field ANN became stagnant. Interest in ANNs was restored in 1975 by introduction of Werbos's backpropagation algorithm [12].

Today there are lot of types of Artificial neural networks e.g. Convolutional neural network, LSTM, Generative adversarial network etc. ANNs are used in variety of fields like Computer vision, Medical diagnosis, and many others [13, 4].

ANNs are often seen as a "black box" model, since the reasoning of its output is often inexplicable. Nevertheless ANN is the state-of-the-art approach for many areas of machine learning, since it can be used for solving very sophisticated tasks and sometimes it even achieves better results than humans.

Yet, learning of a big artificial neural network is often challenging, since it requires a lot of training samples, computational power and time. Even meeting these requirements does not guarantee a smooth learning process, since there are many decisions to make that are crucial for the learning process and can't be made a priori, since there is no way to proclaim with certainty that the decision will be beneficial. Among such decisions are selection of

ANN architecture, hyperparameters, preprocessing techniques and many others. Sometimes all of these decisions are considered hyperparameters. However, development of ANN is not a complete wander in the dark type of process. There are lot a of good practices and rules of thumb that are commonly applied which might lead to an improvement of the model.

General properties of artificial neural network, it's evaluation and it's learning process are described on an example of Multi-layer perceptron, yet these properties are applicable even for different feed-forward multi-layer Artificial neural networks.

1.5.1 Perceptron

The perceptron (or single-layer perceptron) represents the simplest form of Artificial neural network, since it practically consists of a single neuron. Perceptron is a linear classifier, i.e. it can be used for classification of linear separable sets. The algorithm learns the model vector of weights $W = (w_0, w_1, ..., w_n)$. Activation (output) of a single neuron is expressed as:

$$\hat{Y} = f(\xi) \quad \text{where} \quad \xi = w_0 + \sum_{i=1}^{n} w_i \cdot x_i,$$

f is an activation function, $x = (x_1, ..., x_n)^T$ is an input vector; see Figure 1.6. ξ is called a potential of a neuron or an intermediate quantity. Weight w_0 is commonly referred to as bias (b). The bias performs a linear shift of neural potential.

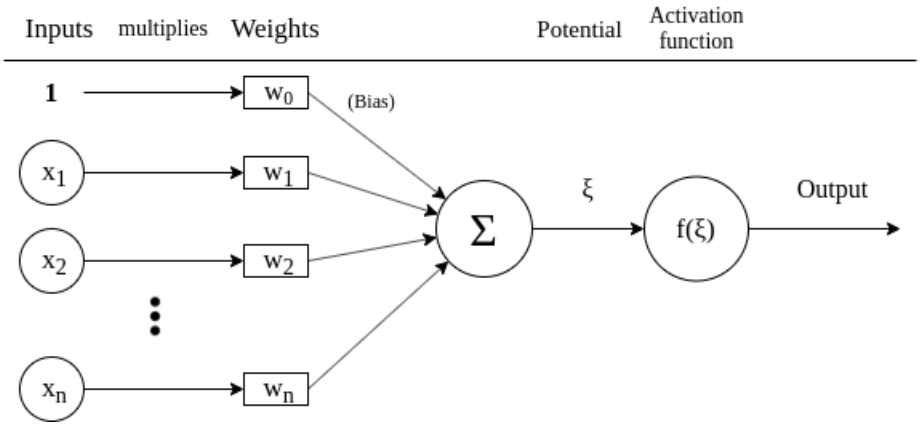

Figure 1.6: Neuron model

Perceptron uses a threshold function as an activation function:

$$f(\xi) = \begin{cases} 1 & \text{if} \quad \xi \geq 0 \\ 0 & \text{if} \quad \xi < 0 \end{cases}$$

1.5.2 Multi-layer Perceptron

Multi-layer Perceptron (MLP) is a feed-forward ANN sometimes referred to as a "vanilla" ANN since it uses the basic idea of ANN.

It consists of neurons arranged into more then two ordered fully connected layers (also called Dense layers). First layer consists of input neurons while the last layer consists of output neurons. Therefore they are called Input and Output layers. Note that model inputs are projected as activation of the Input layer without any change. Layers in between are called hidden layers. Depth of network is denoted as L and it represents a number of layers (including input and output layers). Number of neurons in a single layer is called layer width. Every neuron is connected by connections to all neurons of the previous layer and accepts their activations as inputs. Unlike single-layer perceptron which uses solely threshold function as an activation function, activation function of MLP is selected from more options.

i-th neuron in layer l is referred to as $a_i^{(l)}$ (the same expression may also refers to an activation of the neuron). Inputs $x = a^{(l-1)} = (a_1^{(l-1)}, ..., a_n^{(l-1)})^T$ (where n is a width of layer $l-1$) into every neuron in layer l are activations of layer $l-1$ (the previous layer). Therefore all neurons in one layer share same input yet they react differently according to their weights and bias. Weight of connection from neuron $a_i^{(l)}$ into neuron $a_j^{(l)}$ is represented as $w_{j,i}^{(l)}$.[1] Input weights of neuron $a_i^{(l)}$ are denoted as $W_i^{(l)} = (w_{i,0}^{(l)}, ..., w_{i,n}^{(l)})$. Bias of neuron $a_i^{(l)}$ is referred to as $b_i^{(l)}$ (unlike w_0 in context of single-layer perceptron). See Figure 1.7 for better overall understanding of connections in ANN.

1.5.2.1 Evaluation

Evaluation of the Neural network is called forward pass. For evaluation of each neuron works the same rules as for the single-layer perceptron. Yet there are small adjustments in expressions. May depth of layers l and $l-1$ be d_l and d_{l-1}. Then:

$$a_j^{(l)} = f(b_j^{(l)} + \sum_{i=1}^{d_{l-1}} w_{j,i}^{(l)} \cdot a_i^{(l-1)})$$

The expression can be altered into expression of matrix operations:

$$a_j^{(l)} = f(\xi_j^{(l)}) \quad \text{where} \quad \xi_j^{(l)} = W_j^{(l)} \cdot a^{(l-1)} + b_j^{(l)}$$

Therefore the matrix expression for evaluation of one layer is:

$$a^{(l)} = f(\xi^{(l)}) = f(W^{(l)} \cdot a^{(l-1)} + b^{(l)}) =$$

[1] Order of neuron indices might seem strange, yet it's justified by it's later usage in matrix expression

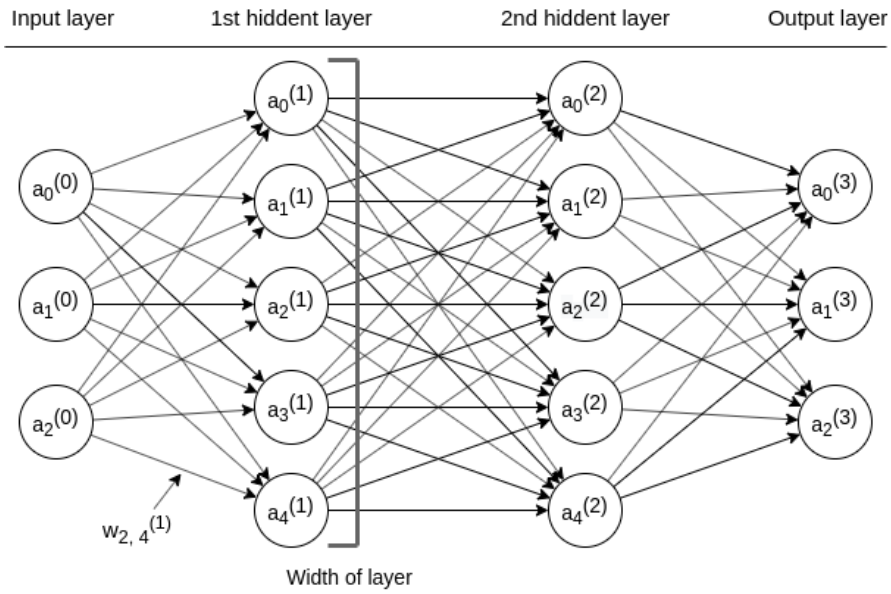

Figure 1.7: Illustration of Multi-layer perceptron with 2 hidden layers of width=5

$$= f\left(\begin{bmatrix} w_{1,1}^{(l)} & w_{1,2}^{(l)} & \dots & w_{1,d_{l-1}}^{(l)} \\ w_{2,1}^{(l)} & w_{2,2}^{(l)} & \dots & w_{2,d_{l-1}}^{(l)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{d_l,1}^{(l)} & w_{d_l,2}^{(l)} & \dots & w_{d_l,d_{l-1}}^{(l)} \end{bmatrix} \cdot \begin{bmatrix} a_1^{(l-1)} \\ a_2^{(l-1)} \\ \vdots \\ a_{d_{l-1}}^{(l-1)} \end{bmatrix} + \begin{bmatrix} b_1^{(l)} \\ b_2^{(l)} \\ \vdots \\ b_{d_l}^{(l)} \end{bmatrix} \right)$$

where the activation function f is applied element-wise: $f(\xi^{(l)}) = (f(\xi_1^{(l)}), \dots, f(\xi_{d_l}^{(l)}))^T$

By this recurrent expression, the network is evaluated layer by layer up to the output layer.

Matrix way of expression is not only shorter and well-arranged, but most importantly, it is much more efficient in practice, since computational hardware is very well optimized to perform matrix operations.

1.5.2.2 Learning

Artificial neural networks learn by adjusting their parameters. There are two types of learnable parameters in Multi-layer perceptron, which affects output of the model: Weights and Biases. Learning of feed-forward Artificial neural networks is done by Backpropagation algorithm and some form of Stochastic gradient descent (SGD) [14].

At first, the ANN is provided training samples and they are used for evaluation of loss function C. Backpropagation is used for finding gradient ∇C

with respect to all weights and biases. After that the gradient descent modifies weights and biases to minimize the loss with respect to ∇C [14, 15].

In order to calculate gradient ∇C_X for training sample X it's necessary to calculate partial derivative of C_X with respect to all weights w and biases b:

$$\nabla C_X = \Big(\frac{\partial C_X}{\partial b_1^{(1)}}, \frac{\partial C_X}{\partial w_{1,1}^{(1)}}, \ldots, \frac{\partial C_X}{\partial w_{d_L,d_{L-1}}^{(L)}}\Big)^T$$

In following text $\delta_j^{(l)}$ will represent error in $\xi_j^{(l)}$ of neuron $a_j^{(l)}$. Backpropagation offers procedure to calculate this error. For neuron in output layer it is expressed as:[2]

$$\delta_j^{(L)} = \frac{\partial C_X}{\partial a_j^{(L)}} f^{(L)\prime}(\xi_j^{(L)}) \implies \delta^{(L)} = \nabla_a C \odot f^{(L)\prime}(\xi^{(L)})$$

where $\nabla_a C$ represents vector of $(\frac{\partial C_X}{\partial a_1^{(L)}}, \ldots, \frac{\partial C_X}{\partial a_{depth(L)}^{(L)}})^T$ and $\odot$ Hadamard product (element-wise multiplication of matrices). Next recurrent equation represents dependency of error $\delta^{(l)}$ on next layer's error $\delta^{(l+1)}$.

$$\delta^{(l)} = ((W^{(l+1)})^T \delta^{(l+1)}) \odot f^{(l)\prime}(\xi^{(l)})$$

Error of any neuron can be evaluated using these two equations. Therefore:

$$\frac{\partial C_X}{\partial b^{(l)}} = \delta^{(l)}$$

This equation seems natural, since δ is difference between ξ and desired neuron's potential and bias performs independent linear shift. At last:

$$\frac{\partial C_X}{\partial w_{j,k}^{(l)}} = a_k^{(l-1)} \delta_j^{(l)}$$

These four equations express everything needed for construction of gradient ∇C_X. Gradient ∇C is then calculated as an element-wise average of gradients of all samples from the training dataset [15].

Once the gradient ∇C is calculated it is applied to the weights. This is done by Gradient descent algorithm which adjusts weights in a direction of the gradient in order to minimize the loss function. Size of these adjustments is controlled by hyperparameter η which is in the context of ANNs called learning rate. Thus for combination θ_n:

$$\theta_n = \theta_{n-1} - \eta \cdot \nabla C(\theta_{n-1})$$

[2] In the expression activation function f has layer index. It was omitted until now, yet it is common for an ANN to use more different activation functions

where n denotes number of step. Selection of learning rate is crucial and often intricate. High learning rate results in divergence of the search, while low learning rate makes requires lot of training iterations and is vulnerable to ending in suboptimal result; See Figure 1.8. There are techniques that address this problem like decaying learning rate during training, application of momentum in descent, adaptive learning algorithms that adjust learning rate during training, etc. or their combinations.

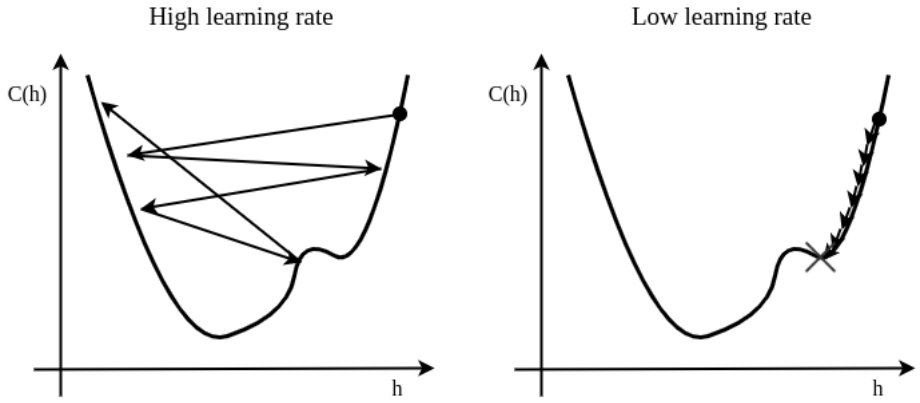

Figure 1.8: Illustration of issues which comes with suboptimal selections of learning rate for learning of one parameter h

Once all training samples are used, the whole process starts all over again. One such an iteration is called an **epoch**.

Since datasets used for training of ANNs are usually large and their learning process takes a very long time, the model is learnt on subsets of the dataset called mini-batches (or batches). Therefore the weight changes are applied more often which leads to faster convergence of loss and also increases overfitting resistance. Application of gradient for every mini-batch is called Stochastic Gradient descend (SGD).

1.5.2.3 Adam

Over the years, many optimization algorithms were designed to adjust steps of SGD to make it robust against stranding in suboptimal minimum and make the training quickly converging to optimum, yet all of optimization algorithms has their cons and there is no optimization algorithm that performs best in all cases. The adaptive momentum estimation algorithm known as Adam, combines RMSProp algorithms and basic SGD with momentum. It is often used because of its ability to find optimum very fast, however in some cases it tends to converge into some local minimum [16].

Momentum technique works similar as the momentum in the world of physics. If the ∇C is steep down it enlarges the step size in the direction of

the gradient for next iterations while if it is uphill the momentum makes the step size smaller, yet it can preserve previous direction for further iterations (evade small local minimum). Basic step of SGD with momentum looks:

$$\theta_n = \theta_{n-1} - \eta \cdot z_n \quad , \quad z_n = \beta z_{n-1} + (1-\beta) \cdot \nabla C(\theta_{n-1})$$

This technique requires only parameter β, that determines how strong momentum impact is.

RMSProp optimizes learning rate for each individual parameter, therefore the learning step is accelerated in the desired direction. It observes second raw momentum (uncentered variance) of parameters and accelerates parameters with low variance (those which behaves steadily). To estimate the parameter momentum for the dataset (only the current mini-batch is available) it uses moving average v of squared cost function:

$$p_n = p_{n-1} - \frac{\eta}{\sqrt{v_i} + \epsilon} \frac{\partial C(\theta_{n-1})}{\partial p} \quad , \quad v_n = \beta v_{n-1} + (1-\beta)(\frac{\partial C(\theta_{n-1})}{\partial p})^2$$

where p_n is the parameter in step n, β is hyperparameter that controls the exponential decay rate of the moving average and ϵ guarantees computational stability. Therefore, if the parameter oscillates then the moving average of variance is high and further steps of the parameter are small while monotonous changes causes small variance and bigger steps [17, 16].

Adam takes up on the idea of RMSProp but also uses first momentum (mean) with all parameters. The step Adam makes is:

$$p_n = p_{n-1} - \frac{\eta \hat{z_n}}{\sqrt{\hat{v_n}} + \epsilon}$$

where and are bias-corrected moment estimates[3] and z_n is has modified form for a single parameter ($\frac{\partial C(\theta_{n-1})}{\partial p}$ replaces $\nabla C(\theta_{n-1})$).

There is also variation of Adam that incorporates Nesterov momentum called **Nadam**. This algorithm turned out to decrease cost even faster than its predecessor. See Nadam original paper for more information [18].

1.5.3 Activation function

Until now, form of activation functions (AF) were omitted with the exception of single-layer Perceptron. There are several requirements for activation function. It has to be differentiable. Otherwise it would not be possible to apply backpropagation algorithm for learning. Activation function should be non-linear. Although it is possible to use linear activation function it would demote the model into form, which could be represented as a model with a

[3]Technique that prevents momentum estimate from being dragged to zero too harshly; See original paper [16] section 3 for further details.

single hidden. Such network would not be able to identify high level features, therefore it is not suitable for complicated problems [14].

One of activation functions used in hidden layers is **hyperbolic tangent**. It was often used in past.

$$f(\xi) = \frac{e^{\xi} - e^{-\xi}}{e^{\xi} + e^{-\xi}}$$

ReLU (rectified linear unit) is currently the most usual activation function for hidden layers.

$$f(\xi) = max(\xi, 0)$$

For regression tasks identity function $f(\xi) = \xi$ is used in the output layer as an activation function.

Sigmoid function (logistic function) is an activation function used in the output layer for binary classification models.

$$f(\xi) = \frac{1}{1 + e^{\xi}}$$

Output of model is $P(Y = 1|X = x) = f(\xi)$ and $P(Y = 0|X = x) = 1 - P(Y = 1|X = x)$.

Softmax function is an activation function used in the output layer for multi-class classification models. It guarantees that all activations of the output layers will be in interval $(0; 1)$ and sum of all outputs will be equal 1.

$$f(\xi_k) = \frac{e^{\xi_k}}{\sum_{j=1}^{K} e^{\xi_j}}$$

for all $k \in \{1, \dots K\}$ where K denotes number of classes. Output of model is then $P(Y = k|X = x) = f(\xi_k)$ (The likelihood of x's belonging to class k). Prediction of model is class with highest likelihood of belonging [6].

1.5.4 Loss function

Loss function (or cost function) measures the difference between real labels Y and output of the model $\hat{Y}$. Selection of loss function is significant since it determines what the model is actually learning. Like an activation function, it has to be differentiable. Otherwise the backpropagation would not be applicable. There are commonly used losses for different tasks [15, 19].

Squared error is typical for regression tasks. It harshly penalizes big deviations.

$$L(Y, \hat{Y}) = (Y - \hat{Y})^2$$

Binary cross-entropy is used of binary classification tasks.

$$L(Y, \hat{p}) = -Y \log \hat{p} - (1 - Y) \log(1 - \hat{p})$$

where $\hat{p} = \hat{P}(Y = 1|X = x)$.

Categorical cross-entropy is used for multi-class classification tasks:

$$L(Y, \hat{p}) = -\sum_{j=1}^{K} \mathbb{1}_{Y=j} \log \hat{p}_j$$

where K is number of classes, $\hat{p} = (\hat{p}_1, \dots \hat{p}_K)$, $\hat{p}_i = \hat{P}(Y = i)$ and $\mathbb{1}_{Y=j} = 1$ for $Y = j$, 0 otherwise.

1.5.5 Droupout

Droupout is a one of techniques applied to ANNs to address overfitting. During the training process droupout causes deactivation of randomly selected neurons in hidden layers. Thus, hidden layers can not completely rely on the presence of other neurons. This prevents learning of complex features found in only one single training sample. Dropout is often used between fully connected layers since it provides a good way for regularization of the model [20].

1.6 Convolutional neural network

Convolutional neural networks (CNN) were used for visual tasks since the late 1980s. However their application was not used widely becouse of a lack of big computational power and insufficient sizes of available image datasets. These both conditions changed for the better in the mid 2000s. The most significant rise of interest they encountered in 2012, when CNN model won the ImageNet challenge (ILSVRC) and with record breaking results. Nowadays CNNs represents state-of-the-art approach for computer vision tasks since they significantly outperforms other known methods [20].

Convolutional neural networks represent a type of feed-forward ANN and it's the state-of-the-art method for Computer vision, yet it has found an application in other fields like natural language processing [20].

The principle of CNNs is based on an assumption that individual inputs of a neural network are positionally related, thus inputs which are close together share some context. Image is a textbook example of such input.

In the following sections, principles are explained with 2D image data, yet they can be generalized for N-dimensional data.

1.6.1 Layers of CNN

Unlike multi-layer perceptron, CNNs are built from several different types of layers.

1.6.1.1 Convolutional layer

The convolutional layers serve as feature extractors. Neurons in a convolutional layer are arranged into feature maps. Each neuron has its receptive field

(kernel). The neuron is connected to neurons within the receptive field in all feature maps of the previous layer. All neurons within a single layer share the same weights. Therefore feature map can be seen as product of application of convolution filter (represented by weights) on feature maps of previous layer [20].

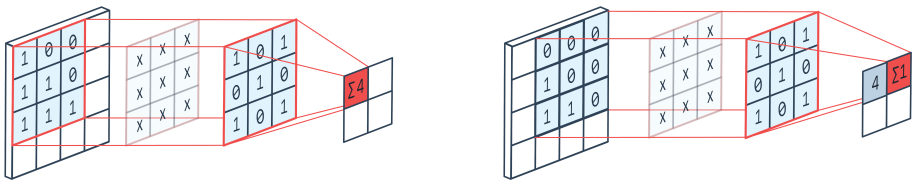

Figure 1.9: Illustration of 2D convolution application on feature maps. Kernel shape = (3, 3), stride = 1 [21]

Stacking of convolutional layers results in behaviour where early layers learn to extract low level features like edges and corners while deeper layers infer more complicated features from combinations of previous feature maps [13].

Convolutional layer has several attributes like number of feature maps (kernels), shape of the receptive field (kernel shape), stride, padding. Size of the receptive field determines how many and which "surrounding" neurons from previous layers are connected to the neuron. Stride can be seen as a size of convolution shift. Applying stride down-samples feature maps. Padding determines how neurons should be interpreted if the receptive field overflows outside the previous layer feature map [22].

1.6.1.2 Pooling layer

Pooling layers are used to reduce the resolution of feature maps (down-sampling). This leads to positional/rotational invariant feature extraction and reduction of computational difficulty. Initially, Average pooling was used as a common practice, yet recent models usually use Max pooling which leads to better results in practice [20].

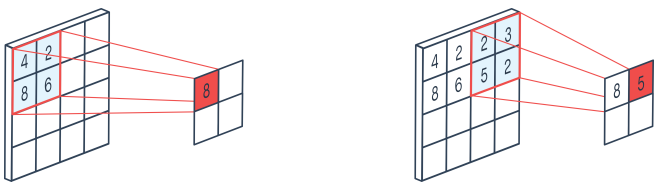

Figure 1.10: Illustration of max pooling applied on feature maps. Pool shape = (2, 2), stride = 2 [23]

Special case of pooling layer is global pooling layer. Such a layer aggregates the whole feature map into a single neuron. Global pooling layer is often used as a last layer which uses feature maps and the flow is followed up by fully connected layers [20, 13, 22].

Attributes of the pooling layer are pool shape, which determines the field of neurons for aggregation. Stride works just like in case of Convolutional layer[20, 22].

1.6.2 Deep convolutional neural network

Development revealed that CNN's depth plays a key role in improving model performance. Deeper CNNs naturally integrates higher level features, therefore are applicable even to more complex tasks [24].

Unfortunately training of deep networks suffers from a problem called **Vanishing gradient problem**. Vanishing gradient refers to a phenomenon in which the early layers have gradients near zero because high layers are almost saturated. Even opposite phenomena can appear. It's called **Exploding gradient** and it is characterized by propagation of huge gradient into early layers, making them completely unstable [20].

1.6.3 Batch normalization

Although batch normalization can be applied even in other types of neural networks, it is listed in CNN section because it presents inseparable part of Deep CNNs today [20, 13, 25].

Another problem of deep ANN learning is a phenomenon called Internal covariate shift. It is caused by changes of distributions of each layer's inputs caused by weight changes in the previous layers. Therefore training such network requires use of a lower learning rate which results in slower training [26, 13].

This problem is addressed by Batch normalization (BN). Batch normalization normalizes each output of the layer with respect to mean and variance of the mini-batch in the output. The normalized values are then applied to outputs before application of non-linearity (activation function) in order to keep input distribution of the next layer the same. Yet plain normalizing of output might bring some undesired phenomenons, for example change of bias in the neuron alone does not change its distribution at all, therefore bias would affect the potential in any way. To avoid this, batch normalization is applied by linear transformation: $\hat{\xi} = \gamma \cdot BN(\xi) + \beta$, where γ and β are trainable parameters [26].

The batch normalization also works as a regularization method, improving generalization of the model. In experiments it was found that BN can substitute or reduce the need of usually used dropouts. Application of batch normalization was proposed to be done before activation function, yet later

research showed that the batch normalization of the flow just before convolutional layer can is a more effective approach [26, 27].

1.6.4 Architectures

Since the introduction of CNNs many architectures have been derived. "Basic" CNN architecture comprises several convolutional layers continuously down-sampled by pooling layers or a stride. At the end feature maps are either reduced by global pooling or passed as they are to a stack of fully-connected layers.

In the following sections are presented architectures which brought groundbraking novels to the field.

1.6.4.1 VGG-16

The VGG-16 architecture was introduced in 2014. It consists of 16 layers, 13 convolutional and 3 fully connected. Usual CNN's had about 5 layers by that time. The novel is an idea that increasing depth of CNN leads to model improvement [28].

1.6.4.2 ResNet

ResNet is the network that popularized the use of batch normalization and residual mapping and introduced very deep networks [13].

It is built from residual blocks. Every block contains stacked non-linear convolutional layers with ReLU activation function and batch normalization. Original order of these operations is $Conv > BN > AF$, yet further research invented different arrangements which led to better results [27]. Every residual block comes with a so-called shortcut connection[4]. This connection represents a flow of activations, parallel to the stack of layers and their feature maps are added together; See Figure 1.11. Therefore every block applies residual mapping $x_l = F(x_{l-1}) + x_{l-1}$ where x_l is vector of block activations, x_{l-1} is previous block activations (output of the shortcut connection) and $F(x)$ represents activations of stacked layers based on the x_{l-1}. Down-sampling through network is done by 1×1 Convolution with stride [13].

Let $H(x)$ be desired mapping done by a few stacked layers and x input activations to them. Idea of ResNet assumes if the stacked layers can learn mapping $H(x)$ then it can also learn $F(x) := H(x) - x \implies H(x) = F(x) + x$. Therefore it should be possible to add blocks without enlargement of training error since the optimizer can drive weights of new stacked layers to zero in order to perform as an identity mapping. Residual mapping speeds up learning the process by increasing resistance to the vanishing gradient problem [13]. See original paper [13] for closer residual mapping description.

[4] Do not confuse with connections between individual neurons

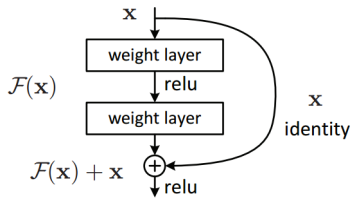

Figure 1.11: Illustration of residual block [13]

Shortcut connection is usually done by identity. Yet it can't be used if the stack of layers does a down-sampling because feature maps of the stack's activation and shortcut would not match. In that case is used a projection which down-samples shortcut to the same dimension by 1×1 Convolution with stride [13].

There are several models of different depth presented in the original ResNet paper [13], based on the idea of residual connections. From all architectures the 152 layer deep ResNet had the best performance in most cases. ResNet paper also introduced 1202 deep architecture. It performed well but suffered more from overfitting [13].

1.6.4.3 DenseNet

Nowadays DenseNet represents state-of-the-art architecture which takes up the idea of shortcut connections and extends it furthermore [25].

Dense net is built of dense blocks which consists of several layer stacks. Each stack has a shortcut connection (identity), yet feature maps are not added to feature maps of the stack but they are contacted unlike ResNet. Therefore each stack produces feature maps which are propagated into next layers of dense block without any change, providing better flow between all layers and resistance to the vanishing gradient problem. Number of feature maps with which a stack contributes to the flow (Growth rate) is denoted as k and the number of output feature maps in l-th block is expressed as $k_0 + k \cdot l$, where k_0 is the number of feature maps passed to the block. Each stack comprises $BN > AF > 1 \times 1\ Conv > BN > AF > 3 \times 3\ Conv$ operations. The 1×1 convolution, also referred to as bottleneck layer, reduces the number of feature maps (concatenation of all stack outputs may lead to a great number of feature maps in later layers, depending on k) for 3×3 convolution which contributes its features to the flow [25].

All dense blocks are separated by a transition layer which down-samples feature maps and comprises $BN > AF > Pool$ or $Conv$. Usual choice is pooling, yet convolution might be used if compression of model is desired (the DenseNet which uses convolution compression is denoted as DenseNet-C) In that case parameter $0 < \theta \leq 1$ denotes compression rate and the convolutional

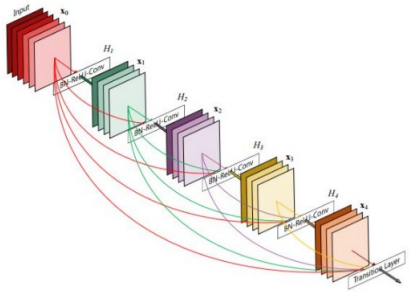

Figure 1.12: Illustration of a dense block [25]

layer reduces the number of feature maps to $\lfloor m \cdot \theta \rfloor$, where m denotes the number of feature maps of the previous dense block.

1.6.5 Grad-CAM

The reasoning behind a prediction made by a artificial neural network is often unclear. Therefore the user can not tell whether the prediction is based on a relevant feature in data or for example some kind of anomaly unknown to the from training data. Such model is completely unacceptable in many fields including medical imaging where performing treatment based on "black-box" diagnosis is unthinkable.

Class activation map (CAM) is a map of the analyzed image what highlights places in the image based on the influence they contributed to the decision [29].

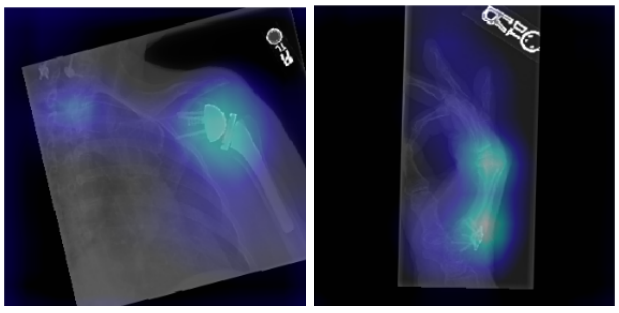

Figure 1.13: Examples of Grad-CAM visualization applied to sample from MURA dataset evaluated by one of the implemented prototypes, highlighting areas that most affected the decision [4]

Grad-CAM is an algorithm for classification CNNs used to find Class activation maps. The algorithm watches the gradient flowing into the last convo-

lutional layer and assigns importance of values to each neuron for the desired class [29].

1.7 Ensemble model

Ensemble techniques are methods used to combine more different machine learning models into one model in order to get more robust predictions. This can be especially useful for good models that are varied in the way they perceives the input data and makes predictions based on different features, thus can mutually compensate weaknesses. Ensemble models are very popular nowadays since they often reduce the generalization error. Nevertheless there are no guarantees that the ensemble model will perform better then its individual submodels. In practice ensembles are built from three to ten models, usually the more the better, nevertheless it is good to keep in mind that with 10 models, the time needed for evaluation will also rise 10 times (assuming that individual models have similar prediction time requirements) [30].

There are several ways to implement ensembling in practice. The basic one, also called "committee of networks", is simple averaging of the model outputs. Despite its simplicity, this approach is still most commonly used to this day. Another possible approach is weighted averaging of model predictions, where the weights are represented by success rate of the individual models. Lastly, the Stacking ensemble needs training unlike previously presented methods. Stacking ensemble uses the outputs of submodels and learns from them a new model on top of them to combine their predictions into the prediction of the ensemble [30].

1.8 Machine learning in Medical imaging

Medical imaging is a set of tests used to create internal images of patient body parts. These techniques greatly improved healthcare, physicians can perform patients internal diagnosis without invasive intervention and provided us valuable information about the human body. Among medical imaging techniques are X-ray, computed tomography (CT), magnetic resonance imaging (MRI), positron emission tomography (PET), mammography, ultrasound and others [31, 32].

Finding informative features is a key to a successful medical analysis. In early applications of machine learning in medical imaging meaningful features were designed mostly by human medical experts on the basis of their knowledge. Thus did not allowed non-experts to contribute to the field. Although, there has been attempts of feature extraction based solely on data, true breakthrough was done by expansion of deep learning, incorporating the feature extraction into a learning step [32].

Nevertheless, deep learning methods usually requires large dataset to learn. That represents a problem, since the medical imaging field does not collect such data in an employable form (the data are unlabeled). There are techniques that can help overcome this problem. For example labels can be inferred from the associated radiological reports using natural language processing, like it was done with ChestX-ray8, the currently biggest open dataset of X-ray images [32, 33, 34].

Data augmentation is a matter of course for practically any computer vision tasks, yet the impact of basic augmentations on training for smaller datasets is limited. Generative neural networks (GAN) gained a lot of attention lately, since they are able to create new data samples that represents the target domain well [35, 34].

Another approach to compensate a small number of samples is using a model learnt on different dataset, for example a dataset of natural images. Although such a model does not know features from the task domain, it has been found that low level features are fairly similar across different domains. Therefore continuing model training with different dataset can reduce overfitting for the target domain and speed up the training process. This approach is referred to as Fine tuning. There are some networks specifically designed to benefit from pre-training like DeTraC, which is used for classification of X-ray images. Such a network requires only a small number of samples from the target domain to work well [36, 11, 37].

Lately the U-net network architecture has gained great popularity as a model for image segmentation[5] in the medical imaging area, since the model is capable of learning from very little data and resistant against overfitting [38].

Nevertheless in these days new bigger medical imaging datasets arise (especially in cases of more common disorders like fractures), making training of CNNs much easier [32].

[5]The computer vision task with the goal of splitting the image into several segments where every one of them consists a desired feature

Chapter 2

Analysis and design

In this chapter I will present MURA dataset - the labeled dataset comprises more than 40,000 X-ray shots. Using this dataset I have trained several CNN prototypes and conducted experiments with the prototypes that are also described in this chapter. Since the dataset is fairly big (especially for a medical imaging dataset) it is possible to use more conventional models that leverages from a large number of samples [4].

2.1 MURA dataset

MURA (musculoskeletal radiographs) dataset is a large open dataset containing 40,561 images from 14,863 radiograph studies of upper extremity, where each study is labeled as either normal or abnormal. Classified parts of limb are elbow, finger, hand, humerus, forearm, shoulder and wrist [4]. See distribution of studies at Table 2.1.

Study	Train		Validation		Total
	Normal	Abnormal	Normal	Abnormal	
Elbow	1094	660	92	66	1912
Finger	1280	655	92	83	2110
Hand	1497	521	101	66	2185
Humerus	321	271	68	67	727
Forearm	590	287	69	64	1010
Shoulder	1364	1457	99	95	3015
Wrist	2134	1326	140	97	3697
Total No. of Studies	8280	5177	661	538	14656

Table 2.1: Distribution of studies with respect to limb part, set type and label [4]

MURA dataset is one of the biggest open dataset of radiographic imaging.

Thus makes the problem Unfortunately, the test set is not publicly available, nor is it possible to get your own model evaluated on test data at the moment.

The data was collected from the Picture Archive and Communication System of Stanford Hospital with respect to patients personal data. Images in the dataset were scaled down, while preserving their aspect ratio. Original classification of all samples was manually done by 6 board-certified radiologists from the Stanford Hospital. To get a better understanding of the nature of injuries, the radiologists also described findings of 100 abnormal studies. Results were: "53 studies were labeled with fractures, 48 with hardware, 35 with degenerative joint diseases, and 29 with other miscellaneous abnormalities, including lesions and subluxations" [4].

2.2 Prototypes

As prototypes I chose 4 Convolutional neural network architectures. I compared their attributes, advantages and disadvantages with each other and with the model proposed in the MURA original paper [4].

The baseline model is a 8 layer deep of basic CNN architecture. It sets a baseline for other models and demonstrates differences in the training process of basic and residual architectures.

Another architecture I used was ResNet-34. Its fast and undemanding training process allowed me to perform many experiments with data augmentation, down-sampling strategies and more, before bringing the technique to the deeper models.

At last I trained ResNet-101 and DenseNet-C-169 models, deep networks that represent state-of-the-art approaches for image classification of large sets.

In order to get a better understanding of model predictions I use Grad-CAM.

2.2.1 Metric

As a metric of quality I use Cohen's kappa as it is used in the original MURA paper. Cohen's kappa is a metric of inter-rater reliability (agreement) between two raters for categorical predictions. It's value is in range -1 to 1, where 1 represents a perfect match of predictions, -1 complete contradiction in predictions and 0 represents plain random inter-rater reliability. Value lower than 0.4 indicates poor agreement, between 0.4 and 0.75 indicates good agreement and more than 0.75 represents great agreement. Cohen's kappa is expressed as:

$$K = \frac{Pr(a) - Pr(e)}{1 - Pr(e)}$$

where $Pr(a)$ is agreement among raters (Accuracy) and $Pr(e)$ is an estimate of random agreement among raters [39].

Unlike accuracy Cohen's kappa is well suited even for evaluation of unbalanced distribution of labels (In case of MURA it is 6:4). For example if all samples in the dataset of rare disease that consists of 1 positive to 99 negative cases are predicted as negative then the accuracy is 0.99 while Cohen's kappa is 0.

2.2.2 Pre-processing and augmentation

The images of MURA dataset vary in resolution and aspect ratio, yet it is guaranteed that the longer dimension is 512 pixels long. However CNN with Global pooling layer is able to process images of any shape, the batch training of more than 1 sample per batch would not be possible.

During pre-processing the images are resized to 224 x 224 or 320 x 320 (based on application), preserving the aspect ratio. Areas uncovered by image are padded with zeros. Values of pixels are re-scaled by 1/255 as it is a known good practice for neural networks.

The augmentation is applied online – during training. As augmentation transformations I applied rotation, shift, zoom, brightness change, shear and horizontal flip. The possible extent of the transformations (like angle of rotation) is defined as a value from a given range of values. Every image during training is alternated by the transformations with randomly generated parameters in the bounds of the ranges for every epoch.

Transformations	Rotation	Shift	Zoom range	Brightness range	Shear	Horizontal flip
Modest augmentation	30°	0.15	0.9–1.1	0.9–1.1	0.05	yes
Strong augmentation	30°	0.2	0.7–1.1	0.7–1.2	0.05	yes

Table 2.2: Used augmentation strategies

Two strategies I used are described in Table 2.2. The Modest augmentation is designed to apply transformation only to the point where the augmented image is still very similar yet brings invariance to the trained model. The Strong augmentation strategy applies more radical changes to images that can make the dataset more flexible and represent a bigger variety of cases from the field. For the Strong strategy I have considerably expanded the ranges of zoom in and brightness changes. The decision to magnify zoom in range is based on facts that the key features are usually in the middle of an image and the images pre-processed with padding are shifted closer to the center. Nevertheless, in extreme cases the radical changes to an image might make features harder to find, for example by big a decrease of brightness in an outlier image that is already very gloomy.

Impacts of the augmentations strategies had on learning are described in

2.2.3 Training

To keep the training process simple I train models on individual images with labels corresponding to its original study. Nevertheless, to predict studies as a whole, images of the study are evaluated independently and the average of the outcomes represents the output of the model for the study.

The imbalance of MURA dataset is addressed by application of class weights. Weights are calculated to make the sample weighted average of the dataset equal 0.5, therefore $w_0 = N_1/N$, $w_1 = N_0/N$ where N is number of samples and N_i is number of samples labeled as i.

As a lost function I used weighted binary cross-entropy as it is a standard for weighted binary classification tasks.

2.2.4 Test set

To overcome the problem of the missing test dataset I have randomly picked 201 studies from the training dataset - approximately the same size as the testing dataset used in the original paper. Test dataset distribution of sample results and limb part is same as the distribution of training dataset [4].

2.2.5 Architectures

In this section are closer described used architectures and their initial performance.

2.2.5.1 Baseline model

The baseline model is a 8 layer deep of basic CNN architecture. It sets a baseline for other models and demonstrates the differences in training process of basic and residual architectures.

Properties of layers adopts trends from ResNet/DenseNet in the sense of convolution size, numbers of convolutional filters and feature map downsampling based in relation to feature map size to make difference between baseline and other networks slighter to stress out the differences of shortcut connections and batch normalization. The baseline is regularized mostly by strong dropouts to prevent overfitting. See Table 4.1.

I have trained the baseline model using Nadam optimizer with learning rate $= 1e-4$ with batch size $= 32$, until the learning curve[6] plateaued. The images were resized to the 224×224 resolution and augmented using the Modest strategy.

[6]Learning curve is a curve representing the history of model training. Rising (for maximization) curve suggests improvements in the model while the plateaued curve indicates stagnating learning

Layer	Properties	Output size*	Dropout rate
1. Conv layer	7×7 conv $\times 64$, stride $= 2$	$112 \times 112 \times 64$	0
Max pooling	3×3 pool, stride $= 2$	$56 \times 56 \times 64$	0
2. Conv layer	3×3 conv $\times 128$	$56 \times 56 \times 128$	0.3
3. Conv layer	3×3 conv $\times 128$	$56 \times 56 \times 128$	0.3
Max pooling	2×2 pool	$28 \times 28 \times 128$	0
4. Conv layer	3×3 conv $\times 256$	$28 \times 28 \times 256$	0.4
5. Conv layer	3×3 conv $\times 256$	$28 \times 28 \times 256$	0.4
Max pooling	2×2 pool	$14 \times 14 \times 256$	0
6. Conv layer	3×3 conv $\times 512$	$14 \times 14 \times 512$	0
G. Avg. pooling	global pool	512	0.5
7. Dense layer		1000	0.3
8. Dense layer		1	0

Table 2.3: Baseline model architecture
*order size represents: Width×Height×Number of feature maps

The model achieved 0.521 CK for individual images and 0.568 CK for studies on the test set.

2.2.5.2 ResNet-34

34 layer deep ResNet, same as the proposed architecture for ImageNet in the original paper [13] but instead of standard residual block the 2 layer full pre-activation block is used as it is introduced in paper [27].

In order to get a better understanding of hyperparameter space I conducted a grid search for the learning rate for Nadam optimizer and batch size. Every combination of hyperparameters as they are shown in Figure 2.1 was tried at least 5 times. The images were resized to the 224×224 resolution and augmented using the Modest strategy. The training was limited to 50 epochs and used 10 epochs early stopping[7]. The results represent the average maximum score achieved by the configuration.

For further training of ResNet-34 I used the most successful configuration which is learning rate 0.0001 and batch size equal 64. This configuration not only achieved the best score but also its learning curve was still steadily increasing. I trained this model further until the curve plateaued.

The model achieved 0.567 CK for individual images and 0.568 CK for studies on the test set.

[7]Early stopping stops model training if the model does not show any improvement in last n epochs

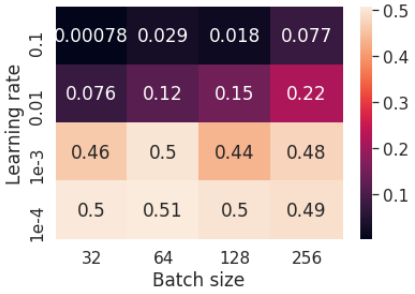

Figure 2.1: Heatmap representing results of the ResNet-34 grid search

2.2.5.3 ResNet-101

101 layer deep ResNet, that is just likethe previous model inspired by original paper architecture for ImageNet, comprises full pre-activation, 3 layer bottleneck blocks. Yet I utilized several amendments that I found useful during experiments with the ResNet-34. They are closer discussed in Chapter 3.Unlike original architecture [13] this architecture uses only 2 down-sampling blocks instead of 3.

For training I used SGD optimizer with momentum which converges slower then Nadam but in many cases is able to achieve a smaller generalization gap. The initial learning rate is set to 0.1, the momentum is 0.9, and weights are decayed by 0.95 factor every 5 epochs. Moreover the learning rate schedule is applied that divides learning rate by 10 if the model does not report improvement for 13 epochs. The images are in 320×320 resolution, augmented by Strong augmentation strategy for training samples and the Modest for validation dataset. The model was trained until the learning curve plateaued and achieved 0.51 (95% CI 0.498 to 0.522) CK for individual images and 0.573 (95% CI 0.567 to 0.579) CK for studies on the test set.

2.2.5.4 DenseNet-C-169

169 layer deep DenseNet-C with parameters $k = 12$ and $m = 0.5$. It's other properties are same as it is proposed in the original DenseNet paper

For training I used SGD optimizer with initial learning rate = 0.01 and momentum 0.9. Learning rate was decayed by 0.95 factor every 5 epochs. Moreover the learning rate schedule is applied that divides learning rate by 10 if the model does not report improvement for 13 epochs. The images are in 320×320 resolution, augmented by Strong augmentation strategy for training samples and the Modest for validation dataset like the ResNet-101. The model was trained until the learning curve plateaued. The model achieved 0.502 (95% CI 0.483 to 0.521) CK for individual images and 0.562 (95% CI 0.543 to 0.581) CK for studies on the test set.

layer name	output size	34-layer	34-layer lds	101-layer lds
conv1	112×112	7×7, 64, stride 2		
conv2_x	56×56	3×3 max pool, stride 2		
		[3×3, 64; 3×3, 64] ×3	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×4	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4
conv4_x	14×14	[3×3, 256; 3×3, 256] ×6	[3×3, 256; 3×3, 256] ×9	[1×1, 256; 3×3, 256; 1×1, 1024] ×26
conv5_x	7×7	[3×3, 512; 3×3, 512] ×3		
	1×1	average pool, 1-d fc		

Table 2.4: Table displaying architecture of ResNet used in this work

Chapter 3

Experiments

This chapter describes experiments I have conducted, mostly on the ResNet-34 model. These experiments are performed on the ResNet-34 architecture with properties that are described in Section 2.2.5.2 to make their application fast.

3.1 Augmentation

The way I tried to reduce overfitting most is image data augmentation. There are several techniques I tested.

Augmentation strategies – application of Modest and Strong that are described in section 2.2.2 were the first technique I tried. Figure 3.1 shows the learning process of models with different strategies.

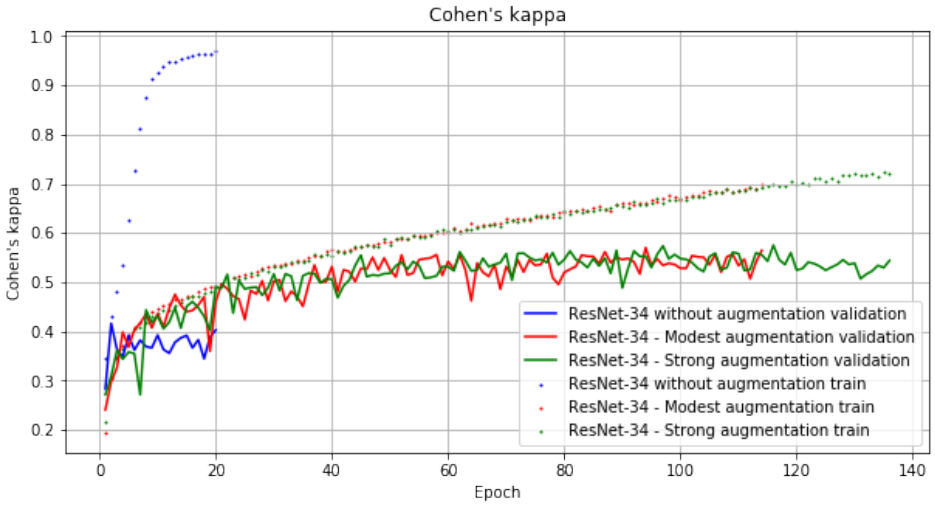

Figure 3.1: Comparison of augmentation strategies

Unsurprisingly, the model without augmentation is very quickly drastically overfitted. Nevertheless Modest and Strong strategy performs very similarly without big differences with exception of the initial training error, although this can be only a coincidence. Since both strategies performed similarly I used Strong strategy in next prototypes since it should provide better generalization for classification of samples outside the defined boundaries of the dataset.

Test augmentation – This experiment addresses the option to augment images outside the training cycle. At Figure 3.2 I applied Strong augmentation to training data and Modest augmentation to validation. This training is compared with a model that applies Strong augmentation for samples.

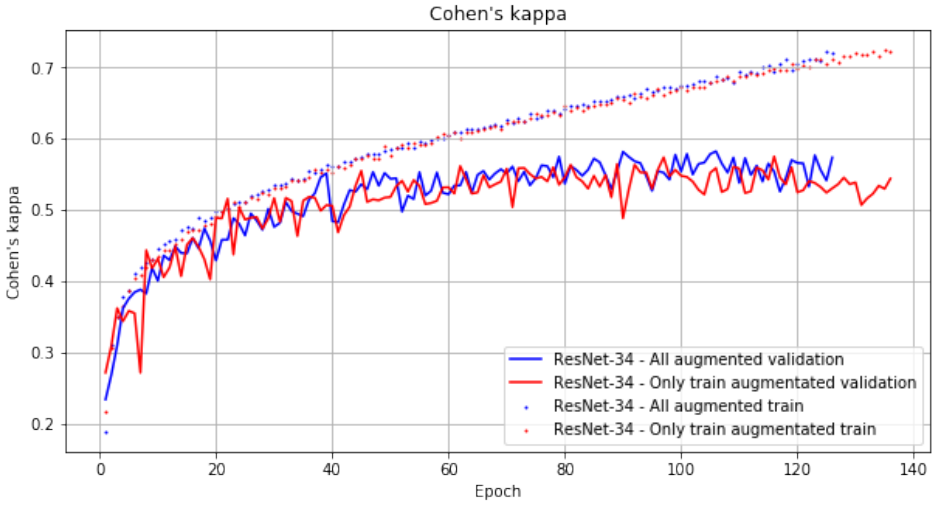

Figure 3.2: Comparison of basic augmentation approach with test time augmentation

The augmentation of all samples turned out to be beneficial, although slightly. Therefore I used this technique in next prototypes.

Image histogram equalization – Image histogram is a representation of a contrast distribution in the image. Histogram equalization is a technique that redistributes contrast uniformly , therefore makes the image features easier to distinguish. Small contrast is often a problem of X-ray shots.

Therefore I tried to apply Modest strategy with histogram equalization (without brightness shift). Despite high hopes, the model did not perform well and achieved a poor score (0.51 CK on individual images from valid dataset). Closer examination of equalized images revealed that their dull noise is often magnified by augmentation into extreme noise, severely damaging image features. See Figure 3.3.

There is another approach to enhance contrast of the image. CLAHE is an algorithm for contrast enhancement. Unlike standard HE, CLAHE does not equalize histogram of the whole image but is concerned about individual

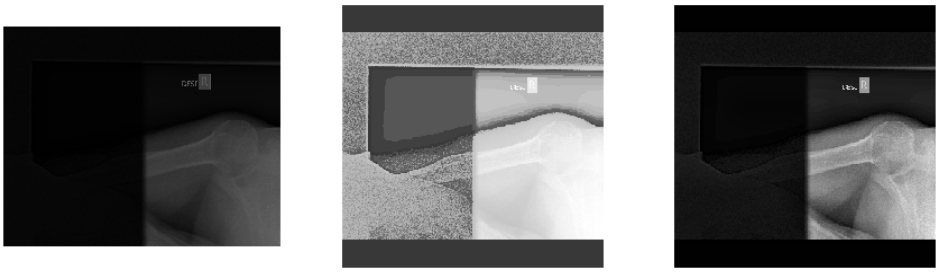

Figure 3.3: Impact of histogram equalization. Left to right: Raw image, Equalized image, CLAHE equalized image

image parts. Figure 3.3 shows the difference where CLAHE nicely brings out the features without causing any collateral damage.

Unfortunately the CLAHE application is computationally expensive and using it with online augmentation turned out to be unthinkable.

3.2 Less down-sampling (lds)

While I've been watching CAMs using Grad-CAM for ResNet I came across phenomena where the produced CAM was highlighting area too big – even wide surroundings of the abnormality. The problem is that the last feature map, a key component for Grad-CAM, was too down-sampled. Therefore I tried to removed the last down-sampling layer. This not only helped Grad-CAM to perform better but also the resultant performed better then previous ResNets. Therefore I applied this adjustment to ResNet architectures used in the thesis. They are denoted as ResNet lds.

3.3 Aggregation of images using maximum

Each study comprises 1 or more images. Previous presented networks used to combine predictions from individual images using average into the final prediction. This comes with one pitfall. Let's assume a study of 5 images, 4 of them are completely fine while 1 contains significant abnormality. Naturally, this study should be declared abnormal, nevertheless a good classifier for individual images with average would proclaim the study as normal, since there is no way for 1 abnormal image to outweigh 4 normal images.

. As a reference I tried to evaluate the ResNet-34 model with maximum as aggregation. It came out with 0.572 CK on the validation set (for comparison it achieved 0.606 CK for studies on the validation set using average aggregation)

To make maximum a decent option I tried to train a model with higher specificity at the expense of sensitivity. To achieve this I used weighted binary

cross-entropy as a function with adjusted class weights. I performed a search for the threshold parameter t that states class weights: $w_0 = t$ and $w_1 = 1 - t$ using a Successive halving algorithm, with 80 epochs as β and $n = 8$ with uniformly distributed values from 0.55 to 0.9. They were compared by their CK score for study. Threshold 0.7 came out as the most successful one.

Nevertheless the final model achieved only 0.49 for studies on the test set.

3.4 Ensemble

At last I built a stacking ensemble model using a simple 4 layer deep MLP where the hidden layers are 25 neurons wide. I used the 5 best performing models with CK on validation set from 0.592 to 0.607. These models are 2 of ResNet-101 type and 3 of ResNet-34 type, trained for images with 320×320 resolution. The model was trained using SGD optimizer with learning rate 0.1, batch size = 128 for 5 epochs.

The model achieved 0.562 (95% CI 0.553 to 0.571) CK for individual images and 0.605 (95% CI 0.588 to 0.622) CK for studies on the test set.

Chapter 4

Results

In this chapter I will summarize results achieved by selected prototypes and compare them with the model presented by MURA original paper [4] called Stanford baseline. For clarity, I will call the baseline model proposed by me in Section 2.2.5.1 My baseline.

Model	Test CK (studies)
My baseline	0.568
ResNet-34	0.568
ResNet-34 lds	0.568
ResNet-101	0.573 (95% CI 0.567 to 0.579)
DenseNet-C-169	0.562 (95% CI 0.543 to 0.581)
Ensemble	0.605 (95% CI 0.588 to 0.622)
Stanford baseline model	0.705 (95% CI 0.700, 0.710)

Table 4.1: Summary of results

4.1 Discussion

Stanford baseline is an averaging ensemble model of 5 169 layer deep DenseNets. The submodels pre-trained on ImageNet dataset[8]. The submodels are trained by Adam optimizer with learning rate 0.0001, batch size = 8 and the learning rate is divided by 10 every time learning curve plateaus. The submodels are trained like mine – from individual images without a study context. Training samples are augmented by rotations of up to 30 degrees and random lateral inversions.

[8]ImageNet dataset is a large dataset of more than 14 million images from 20 thousand classes

Unlike models presented by me, the Stanford model is fed by network whole studies. Yet, since the Stanford model averages predictions of the images in the study, it does not work with the context of the study.

The Stanford model outperformed the models presented by me. In my opinion the better performance of the Stanford model is mainly caused by an application of fine-tuning and probably by more careful setting of parameters.

Most other implementations of CNNs trained on MURA dataset I found online were not very different from the Stanford model. Using ensemble models created from CNNs pre-trained on ImageNet seems to be a common practice. [40].

Interesting approach I found is a using unsupervised learning with GANs using samples classified as normal to get an understanding of hand form in order to make an abnormal study outlier to the domain. See original paper [41] for further info.

Nevertheless it is unfortunate that the models can not be compared on the same dataset, since I have some doubts regarding the quality of labels of the original training dataset, from which I took testing samples. While browsing the dataset I found some samples that look very abnormal for me, as a non-doctor, yet they are classified as normal. See Figure 4.1.

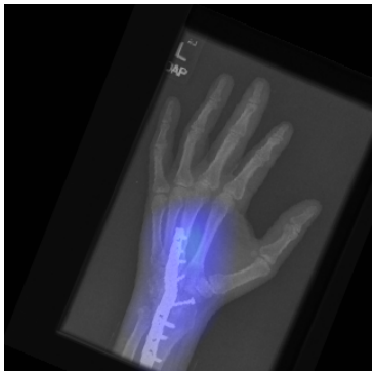

Figure 4.1: Possible wrongly labeled sample from training dataset. The image contains hardware that is highlighted by Grad-CAM.

Chapter 5

Used technologies

This section summarizes the technologies I used for development and analysis of the practical part.

The practical part is written exclusively in Python. Python does not need to be compiled and a new code can be passed to a running interpreter, making Python very suitable for experimenting. Another great advantage is the fact that Python offers a great variety of libraries very well optimized for Machine learning, data processing and visualizations and more.

Development of prototypes was done using Keras. Keras is an API written in Python, running on top of a machine learning platform, with focus on Neural networks. With Keras it is possible to quickly develop high level components of the Neural network, train or evaluate it effectively and its ecosystem provides handy features for data pre-processing and augmentation, hyperparameter tuning and more. The platform under Keras used in this thesis is Tensorflow 2.2. It is a machine learning platform for dataflow and differentiable programming, that has gained great popularity in Neural network development, although it can be used for a variety of tasks. Tensorflow computations can use GPU as well as distribute computations among more processing units, to considerably speed up its computations.

As an IDE I used a web application called Jupyter Notebook that allows the user to pass new code segments to a running Python interpreter, generate visualizations and more via an interactive environment. Although JN is very effective for experimentation, its usage for long time running, computationally demanding tasks is unsuitable. Therefore for model designing, visualizations and other undemanding features the JN is used, while computationally hard tasks are written as a plain Python script.

Chapter 6

Conclusion

6.1 Contribution

There were four goals of the thesis.

The first goal was research of current state-of-the-art techniques used for Computer vision tasks. First Chapter apprises the reader to machine learning basics focusing Artificial neural networks and Computer vision. The end of the chapter describes current aims to meet the task with the support of previously introduced basics.

The second goal was to implement a prototype of a neural network for prediction of MURA studies, a task I focused on the most. Although the proposed prototypes did not match best performing methods, they performed quite well achieving 0.6 CK. Their predictions can be visualized using Grad-CAM to understand . Second and third chapters are devoted to model designing and investigate possible approaches to reduce the generalization gap.

The third goal was comparison of proposed prototypes with models trained on the same literature. This goal was problematic since there are not many published and well documented models and those who do use very similar techniques.

The fourth goal was to publish the prototype code. All scripts used for creation of the prototypes, training, evaluation and visualization are uploaded on the enclosed medium, along with trained prototypes and the results are reproducible.

6.2 Future work

There are a lot of possible options that could further improve the proposed models. Among most promising I consider fine-tuning since it was frequently used in other well performing prototypes I found online. Initially I considered the size of the MURA dataset sufficient for training CNNs from scratch,

yet I was proved to be wrong. Another possibility I find tempting to try is application of CLAHE as an augmentation technique.

As for the domain of computer vision in medical imaging there are some interesting new techniques on the horizon. In my opinion capsule networks represent one of the possible directions to which the field can go. Unlike conventional CNNs, capsule networks perceive the world more like humans, thus giving them great potential.

Bibliography

[1] Bazazeh, D.; Shubair, R. Comparative study of machine learning algorithms for breast cancer detection and diagnosis. In *2016 5th International Conference on Electronic Devices, Systems and Applications (ICEDSA)*, 2016, pp. 1–4.

[2] Predicting eye disease with Moorfields Eye Hospital. *DeepMind*, 2018, [Online; accessed 2-May-2020]. Available from: `https://deepmind.com/blog/article/predicting-eye-disease-moorfields`

[3] Suleyman, M.; King, D. Using AI to give doctors a 48-hour head start on life-threatening illness. *DeepMind*, 2019, [Online; accessed 2-May-2020]. Available from: `https://deepmind.com/blog/article/predicting-patient-deterioration`

[4] Rajpurkar, P.; Irvin, J.; et al. MURA: Large Dataset for Abnormality Detection in Musculoskeletal Radiographs. 2017, `1712.06957`.

[5] Alpaydin, E. *Introduction to machine learning*. MIT press, 2020.

[6] Simeone, O. A Brief Introduction to Machine Learning for Engineers. 2017, `1709.02840`.

[7] Stahl, D.; Pickles, A.; et al. Novel Machine Learning Methods for ERP Analysis: A Validation From Research on Infants at Risk for Autism. *Developmental Neuropsychology*, volume 37, no. 3, 2012: pp. 274–298. Available from: `https://doi.org/10.1080/87565641.2011.650808`

[8] Barnett, V. The Study of Outliers: Purpose and Model. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, volume 27, no. 3, 1978: pp. 242–250, ISSN 00359254, 14679876. Available from: `http://www.jstor.org/stable/2347159`

[9] Pedregosa, F.; Varoquaux, G.; et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, volume 12, 2011: pp. 2825–2830.

[10] Li, L.; Jamieson, K.; et al. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. 2016, 1603.06560.

[11] Shorten, C.; Khoshgoftaar, T. A survey on Image Data Augmentation for Deep Learning. *Journal of Big Data*, volume 6, 12 2019, doi:10.1186/s40537-019-0197-0.

[12] Wikipedia contributors. History of artificial neural networks — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=History_of_artificial_neural_networks&oldid=950342019, 2020, [Online; accessed 28-April-2020].

[13] He, K.; Zhang, X.; et al. Deep Residual Learning for Image Recognition. 2015, 1512.03385.

[14] da Silva, I. N.; Hernane Spatti, D.; et al. *Artificial Neural Network Architectures and Training Processes*. Springer International Publishing, 2017.

[15] Nielsen, M. A. Neural Networks and Deep Learning. 2015, [Online; accessed 29-April-2020]. Available from: http://neuralnetworksanddeeplearning.com/

[16] Kingma, D. P.; Ba, J. Adam: A Method for Stochastic Optimization. 2014, 1412.6980.

[17] Bushaev, V. Understanding RMSprop — faster neural network learning. *towards data science*, 2018, [Online; accessed 14-May-2020]. Available from: https://towardsdatascience.com/understanding-rmsprop-faster-neural-network-learning-62e116fcf29a

[18] Dozat, T. Incorporating nesterov momentum into adam. 2016.

[19] Klouda, K.; Vašata, D. Neuronové sítě. *Vytěžování znalostí z dat*, 2020, [Online; accessed 30-April-2020]. Available from: https://courses.fit.cvut.cz/BI-VZD/lectures/files/BI-VZD-11-cs-handout.pdf

[20] Rawat, W.; Wang, Z. Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review. *Neural Computation*, volume 29, no. 9, 2017: pp. 2352–2449.

[21] 2D Convolution block. *Peltarion*, 2020, [Online; accessed 30-April-2020]. Available from: https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/blocks/2d-convolution-block

[22] Chollet, F.; et al. Keras. https://keras.io, 2015.

[23] 2D Max pooling block. *Peltarion*, 2020, [Online; accessed 30-April-2020]. Available from: https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/blocks/2d-max-pooling-block

[24] Zeiler, M. D.; Fergus, R. Visualizing and Understanding Convolutional Networks. 2013, 1311.2901.

[25] Huang, G.; Liu, Z.; et al. Densely Connected Convolutional Networks. 2016, 1608.06993.

[26] Ioffe, S.; Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. 2015, 1502.03167.

[27] He, K.; Zhang, X.; et al. Identity Mappings in Deep Residual Networks. 2016, 1603.05027.

[28] Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. 2014, 1409.1556.

[29] Selvaraju, R. R.; Cogswell, M.; et al. Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. *International Journal of Computer Vision*, volume 128, no. 2, Oct 2019: p. 336–359, ISSN 1573-1405, doi:10.1007/s11263-019-01228-7. Available from: http://dx.doi.org/10.1007/s11263-019-01228-7

[30] Brownlee, J. Ensemble Learning Methods for Deep Learning Neural Networks. 2018, [Online; accessed 17-May-2020]. Available from: https://machinelearningmastery.com/ensemble-methods-for-deep-learning-neural-networks/

[31] Kim, D. Introduction to Computer-Aided Diagnosis in Medical Imaging (Radiology). 2016, [Lecture; Online; accessed 3-May-2020]. Available from: https://www.youtube.com/watch?v=46Oz5JNFR24

[32] Shen, D.; Wu, G.; et al. Deep learning in medical image analysis. *Annual review of biomedical engineering*, volume 19, 2017: pp. 221–248.

[33] Wang, X.; Peng, Y.; et al. ChestX-Ray8: Hospital-Scale Chest X-Ray Database and Benchmarks on Weakly-Supervised Classification and Localization of Common Thorax Diseases. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul 2017, doi:10.1109/cvpr.2017.369. Available from: http://dx.doi.org/10.1109/CVPR.2017.369

[34] Mei, S.; Yang, H.; et al. Unsupervised-Learning-Based Feature-Level Fusion Method for Mura Defect Recognition. *IEEE Transactions on Semiconductor Manufacturing*, volume 30, no. 1, 2017: pp. 105–113.

[35] Yi, X.; Walia, E.; et al. Generative adversarial network in medical imaging: A review. *Medical Image Analysis*, volume 58, 2019: p. 101552, ISSN 1361-8415, doi:https://doi.org/10.1016/j.media.2019.101552. Available from: `http://www.sciencedirect.com/science/article/pii/S1361841518308430`

[36] Shin, H.; Roth, H. R.; et al. Deep Convolutional Neural Networks for Computer-Aided Detection: CNN Architectures, Dataset Characteristics and Transfer Learning. *IEEE Transactions on Medical Imaging*, volume 35, no. 5, 2016: pp. 1285–1298.

[37] Abbas, A.; Abdelsamea, M. M.; et al. Classification of COVID-19 in chest X-ray images using DeTraC deep convolutional neural network. 2020, `2003.13815`.

[38] Ronneberger, O.; Fischer, P.; et al. U-Net: Convolutional Networks for Biomedical Image Segmentation. 2015, `1505.04597`.

[39] Hsu, L. M.; Field, R. Interrater agreement measures: Comments on Kappan, Cohen's Kappa, Scott's π, and Aickin's α. *Understanding Statistics*, volume 2, no. 3, 2003: pp. 205–219.

[40] Banga, D.; Waiganjo, P. Abnormality Detection in Musculoskeletal Radiographs with Convolutional Neural Networks(Ensembles) and Performance Optimization. 2019, `1908.02170`.

[41] Davletshina, D.; Melnychuk, V.; et al. Unsupervised Anomaly Detection for X-Ray Images. 2020, `2001.10883`.

Appendix A

Acronyms

AF Activation function

ANN Artificial neural network

BN Batch normalization

CNN Convolutional neural network

CK Cohen's kappa

Conv Convolution / Convolutional layer

HP Hyperparameter

HPO Hyperparameter optimization

ML Machine learning

MLP Multi-layer perceptron

PR Polynomial regression

SLP Single-layer perceptron