

Université de Bordeaux,
Pázmány Péter Katolikus Egyetem,
Universidad Autónoma de Madrid

Master Thesis

Training of CNN model for classifying radiological scans

Miklós Tóth

Supervisor:
Dr. Bence Németh
Dr. András Horváth

May 21, 2019

I, the undersigned Miklós Tóth, student of the Pázmány Péter Catholic University, Faculty of Information Technology and Bionics, declare that I have written this diploma thesis solely myself, without any unauthorized help, and I have only used the sources referenced. Every part quoted word by word or in a paraphrased manner is indicated clearly, with a reference made. I have not submitted this diploma thesis in any other training program.

Signature

Kivonat

A Pázmány Péter Katolikus Egyetem Image Processing and Computer Vision szakirányos mesterképzés hallgatójaként az óralátogatásokkal töltött félévek után a diplomatervem megírására visszatértem Magyarországra. A tanulmányaim folyamán szerzett tapasztalatok alapján a gépi tanulás témakörét láttam a legizgalmasabbnak. Ez a szakterület az, aminek fejlődése azt hiszem, hogy a leginkább fogja tudni a jövőnket befolyásolni, és aminek idővel minden más szakterületre és iparágra is komoly hatása lesz. A félév során elvégzett munkámmal szerettem volna magamnak is megteremteni a lehetőséget, hogy az eddigieknél jobb rálátást kapjak erre a szakterületre, illetve, hogy olyan munkát végezhessek ez idő alatt, amivel a már most is jelentkező ipari igényeken keresztül is megvizsgálhatom a gépi tanulás, taníthatóság problémakörét és világát.

Egyetemünk korábbi hallgatójának, Dr. Németh Bencének a vállalkozásában találtam erre lehetőséget. Vele és lelkes kollégáival dolgozva íródott ez a diplomaterv is, és talán nem is ér itt véget majd a munkakapcsolatunk. A félév során ugyan voltak kisebb változtatások abban, milyen konkrét elvárásaik is vannak a munkám kimenetelével, minthogy ők maguk is piaci igényekhez alkalmazkodnak, ez azt hiszem nem meglepő. A dolgozat címe „CNN model készítése radiológiai felvételek osztályozására”, s az első dolog amit megbeszéltünk, hogy milyen feladatok megoldására lenne szükségük. A kórházaktól és radiológiai központokból, úgynevezett távleletező központokba, majd onnan lekérve az ő eszközeikre is beérkező radiológiai felvételeket szeretnék automatikusan vizsgálni, és a megfelelő osztályokhoz tartozó képeken lefuttatni egy speciálisan azokhoz tartozó következő modult. A képfelismerés automatizálása viszonylag megoldott problémának tekinthető, a kérdés az volt, milyen eszközökkel tudok nekik segíteni abban, hogy egy jól működő, gyors, stabil modellt kapjanak a félév végére, és én hogyan tudom összefoglalni a munkámat és tanulmányaimat egy diplomaterv megírása közben.

Az osztályozó konvolúciós neurális hálózatok képelemzésre, azon belül is kategorizálásra használt hálózatok fejlődésének megismerése után a munkámnak helyt adó cég eszköztárát felhasználva a félév során megismerkedtem közelebbről a távleletező központok működésének hátterével, az ott tárolt radiológiai leletek struktúráival és kezelhetőségével. Ezután megkerestem a megbízás elvégzéséhez legalkalmasabbnak tűnő keretrendszereket, megoldottam néhány felmerülő problémát majd néhány teszt segítségével bizonyítottam, hogy egy általam tanított modell is mire lehet képes. A mindeközben megszerzett tapasztalatokról és az ide kapcsolódó és szükséges tanulmányaimból megszerzett tudás összefoglalójaként írtam ezt a diplomatervet.

Abstract

As a student of the Pázmány Péter Catholic University on the Image Processing and Computer Vision specialization, after I finished the semesters abroad and all of our classes, I returned to Hungary. With all the experiences I have gained while learning, I choose to continue with machine learning. This topic will have a tremendous impact on our future, there will be no other field or industry, which will not be effected by it in one way or another. In this semester, I was trying to make it possible to have a better understanding of this field, while I am doing a work, which is able to show the present state and needs of the industrial applications of the machine learning and deep learning.

A former student of our university, Dr. Bence Németh gave me an opportunity in his startup where I was able to do just that. With him, and his enthusiastic colleges I had the opportunity to wrote this thesis, and with the end of this semesters it may will not be the end of our working relationship. As we were going through the semester the expected results of my work have changed sometimes a bit, but I think as they have to work in sight of their marker, it is not a surprise. The title of my thesis is „Training of CNN model for classifying radiological scans", and the first thing we agreed on, was what they need, and what I should deliver. The hospitals and radiological centers produce medical imaging, these findings will be stored in teleradiology stations and can be sent to their system. However they need to be classified, and based on that categorization, automatically start the follow up algorithms. The question of classification is a well written down subject, as I was trying to show this in the literature review, however the better question was that what kind of tools will I be able to find, to help to solve their task, and give them a well working, stable, fast model at the end of the semester. Meanwhile I have to collect and write down what I was working on in a thesis work for the university.

First I was trying to get a better understanding of the convolutional neural networks used for image classification. Then I was focusing on the background of the teleradiology stations, and how do they work. How do they store, and handle medical images. Then I have tired to find possible frameworks to actually do my works. After that I made some effort to prove, that the model I am able to produce is working quiet well

Contents

1 Introduction

As we aim at minimizing the gap between how computers and humans can solve problems when introduced to new problems, we have been witnessing a monumental growth in the field of artificial intelligence. Computer vision is one of the key fields in this domain, as we would like to enable machines to view the world as humans do, perceiving it in a similar manner, and using the knowledge for image recognition, classification, etc. This thesis tries to give a deeper understanding of this field, both theoretically and after that practically. I will present my solution to the problems that I had to face this semester.

The expression "machine learning" was first used by Arthur Samuel in 1985 [18] and the most famous definition was given by Tom M. Mitchell in 1997 [13] as the following: "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."

2 Medical Imaging Classification

2.1 Medical Imaging

Advances in medicine over the past few decades have improved healthcare immensely, allowing doctors to more efficiently diagnose and threat patients. However doctors are only human beings, so they commit mistakes. The potential of this field is obvious and enormous. Tasks like radiological image classification and potential critical area enhancements can free up a lot of time and energy from the doctors hands. Radiologists regularly disagree on their respective interpretations of medical images. A well trained machine can potentially soon be able to work just as effectively as a human doctor, but way more efficiently. It can learn from thousands of images, testing and analyzing of patient data in a matter of milliseconds and make decisions way faster and way more consistently than any human competitor ever could. With the decrease of computational costs these effects will only increase. The first steps towards these goals have already been taken, and even if we are not there yet, with every passing day, we are getting closer and closer to these systems. Until we get there the decision support, and learning from these successful and false predictions will make these systems better.

Medical imaging means the technique of making possible of the interior analysis of a body by visual representation. For thousands of years we only viewed organs and tissues if they were removed. Now with the technical enhancements we are possible to make them viewable and more importantly diagnosable while they are functioning in their place. With the growth of our databases we are able to establish and differentiate normal anatomy and physiology, and from this knowledge we can identify abnormalities as well.

Medical imaging can be seen as a noninvasive technique - no instrument is introduces to the patient's body - where we can produce images of the internal aspect of the patient's body. In the next few subsections I would like to shortly introduce the main techniques available and used today, to have a better understanding what kind of images I have had to work with while writing my master thesis.

2.1.1 X-Ray

Diagnostic x-rays are well known and most frequently used as they are fastest and easiest medical imaging techniques of the present day. To be able to view the internal form of an object, we have to place it between an X-ray generator and a detector field. Both of these essential tools can be various types of developments depending on mostly the time of production. Detectors may be photographic films or digital detectors. As the object absorbs the radiation while it moves through it, the different density levels of the internal

organs have different absorption levels and they will make different shades on the black and white final image. Bones contain calcium, which have the highest absorption level in the human body, therefor those appear white, fat is gray, as it absorbs some of the radiation, and air absorbs almost nothing, so the lungs usually almost black. However we have to remind ourselves, that most usually X-ray images are projections of radiography images. They will be two dimensional molds of the body on the film, therefore sometimes the identification of the individual body parts on top of each other can be very difficult, if not even impossible.

2.1.2 CT

The computer tomography (CT scanning) is very similar to the original X-ray diagnostic approach, however we take multiple images right after each other, and more importantly the also the X-ray radiation source and the detector film is moving around the patient. With these technique, the biggest disadvantage of the Diagnostic x-rays can be reconditioned, namely the overlapping of the objects. As the patient moves through the examination any given part of it will result pictures from multiple different angles and times. The attenuation of the beams is collected and with computer aided computation two dimensional slice images are formed on three planes (axial, coronal, and sagital). It is easier to imagine this period of the image as slices of a bread. However, with even more computation, the almost exact three dimensional model can be generated without a problem. The term "computed tomography" (CT) is often used to refer to X-ray CT, because it is the most commonly known form. But, many other types of CT exist, such as positron emission tomography (PET) and single-photon emission computed tomography (SPECT)

2.1.3 MRI

In contrast to the previously shown X-rays, the magnetic resonance imaging uses strong magnetic fields to generate the radiological images. This means that this is the only radiation free imaging technology. Therefore it is wildly used, even against the fact that the actual examination takes more time and can be considered as a more pricey and not so comfortable experience compared with the previously shown CT. MRI was originally called NMRI (nuclear magnetic resonance imaging), but the use of 'nuclear' in the acronym was dropped to avoid negative associations with the word. Certain atomic nuclei are able to absorb and emit radio frequency energy when placed in an external magnetic field. In clinical and research MRI, hydrogen atoms are most often used to generate a detectable radio-frequency signal that is received by antennas in close proximity to the anatomy being examined. As our body is mostly composed from water, we can see why most MRI scans essentially map the location of water and fat in the body.

2.2 Image Classification

When we are talking about image classification, or image recognition, we have to have the understanding, that computers have no intelligence on their own. For us, humans it seems extremely easy to identify objects on images based on all the knowledge that we have from our surroundings, however even if it seems like sometimes, this was not a given. Before we can even speak or just walk, we play games to identify animals in children books, and we were constantly told by a supervisor how to handle the task. From our part this process involve a great deal of generalization, finding and identifying key characteristics of objects. We separate the irrelevant background from the foreground, we identify the most important objects on the picture, and start to look for those features. A dog have four legs, maybe visible tail, and furry skin. The cats have pretty much the same characteristics, but they are smaller, and their ears are usually pointy. So usually with pretty much confidence, we can make a good decision.

Let's imagine a computer's case, we have an input image, and we know that there is a given set of objects, that can appear on the picture. However to gain the same results with high confidence is a really complex task, given only numerical and spatial data. Usually, in the case of pictures a matrix of pixel intensity values either in two or three dimensions. Somehow, we have to give the ability to a machine, that from the row numerical data, it must find patterns, meaningful features of classes that we are looking for. The computer will have to use low level features first, all of them can be found with fast mathematical matrix calculations, and able to highlight edges and curves. From here with the combinations of these findings it builds up more and more abstract concepts through a series of convolutional layers.

3 DICOM - Digital Imaging and Communications in Medicine

DICOM is a communication protocol and a file format for storage of medical imaging and information. It is able to contain additional patient information beside the single or multiple usually radiological (such as CT scans, MRIs, and ultrasound) images. This way the communications and transmission between workstations, scanners, archiving or printing systems is much easier and the format ensures that all the data stays together throughout the whole process securely.
The copyright holder of the public standards are held by the National Electrical Manufacturers Association (NEMA). The need of standardization appeared with the first electrical acquiring systems in hospitals in the early 80's, and the format has got it's name at 1993. The DICOM Standard is still updated four to five times each year and republished approximately once every year or two. The images stored in the files with the .DCM extension, let's say the pixel data, can be compressed with the use of many different standards, including lossless JPEG, JPEG 2000, or even simple JPEG, and also it can be run-length encoded (RLE).
It is easy to see why this standardization is a success as it gives flexibility to the users, and provides valuable advantages, like the inseparability of the patient and examination information and the actual finding. This way, there can not be a mismanagement of textual data, and precious information can not be lost or swapped.
However, as one huge concern still rises, and partially this was the starting point of my thesis work as well, namely what happens if the fields of the DICOM files that arrive to a system are just missing. It is possible that the file had too many optional fields, and therefore they are just left blank, or in an even worst case, they are filled with incorrect data. [2]

3.1 DICOM tags

The attribute or data element of the DICOM files are a composition of the following four attributes: a tag, a value representation (VR), a value length, and an actual field, which will contain each individual element's own value.

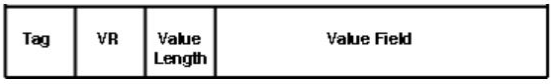


Figure 3.1. DICOM Attribute (Data Element) Structure

Each data element is described by a pair of numbers, usually in the format (XXXX,XXXX) with hexadecimal numbers: DICOM Group Number and DICOM Element Number. Even numbered groups are elements defined by the DICOM standard and are referred to as public tags. Odd numbered groups can be defined by users of the file format, but must conform to the same structure as standard elements. These are referred to as private tags.

For example:

Tag	VR	Name
(0038,0050)	LO	Special Needs
(0010,2297)	PN	Responsible Person
(0010,21A0)	CS	Smoking Status
(0010,4000)	LT	Patient Comments
(0028,1080)	CS	Gray Scale
(0040,A032)	DT	Observation DateTime

... and many many more. Thousands of these kind of DICOM Tags can be present in the output of an examination. Some of them are essentially useful, some of them are not just optimal, but they can be testing device specific, only locally used tags also. When I did my research, I was using the following link to have a solid, well searchable database of DICOM codes and information on them: https://www.dicomlibrary.com/dicom/dicom-tags/

In the previous table there can be seen some of the examples I picked semi randomly from the thousands of examples and perhaps are useful tags. If we want to look for the meaning of the VR (value representations) fields, we get a small description of every second character coded formats meaning. For example

- **LO** means ***Long String***: Long String A character string that may be padded with leading and/or trailing spaces. The character code 5CH (the BACKSLASH "ïn ISO-IR 6) shall not be present, as it is used as the delimiter between values in multiple valued Default Character Repertoire and/or as defined by (0008,0005). Its length can be 64 chars maximum.

- **LT** means ***Long Text***: A character string that may contain one or more paragraphs. It may contain the Graphic Character set and the Control Characters, CR, LF, FF, and ESC. It may be padded with trailing spaces, which may be ignored, but leading spaces are considered to be significant. Data Elements with this VR shall not be multi-valued. Its length can be 10240 chars maximum.

- **DT**: ***Date Time*** common data type. Indicates a concatenated date-time ASCII string in the format: YYYYMMDDHHMMSS.FFFFFFZZZZ The components of this string, from left to right, are YYYY = Year, MM = Month, DD = Day, HH = Hour, MM = Minute, SS = Second, FFFFFF = Fractional Second, and ZZZZ = Hours and Minutes of offset. ZZZZ is an optional suffix for plus/minus offset from Coordinated Universal Time. A component that is omitted from the string

is termed a null component. Trailing null components of Date Time are ignored. Non-trailing null components are prohibited, given that the optional suffix is not considered as a component.

3.2 Images in DICOM

DICOM supports a wide range of image formats for storing medical image pixel volumes. The formats can be loosely broken into two main groups:

1. DICOM-specific : the formats that are used by DICOM only. They are typically the oldest ones, introduced at the dawn of the computer era before better imagevstoring approaches had been developed. They resemble raw bitmaps with varying ways of packing the pixel bytes.

2. Independent standard formats accepted by DICOM . These include such well known formats as JPEG , RLE (run-length encoding), ZIP , and the less known (but getting more and more popular) JPEG2000 , and JPEG-LS . All these standards are also associated with various image compression techniques, reversible and not, which makes them particularly useful in medical imaging (reducing image data size is important). When DICOM includes other standards to be used for particular tasks (such as JPEG for image encoding), the embedded standard approach is very convenient, consistent, and makes good practical sense.

Image modality	Typical image matrix (height, weight, bytes per pixel)	Image size (KB)	Typical number of images in a study	Typical study size (MB)
Computed tomography, CT	$512 \times 512 \times 2$	512	500	250
Magnetic resonance, MR	$256 \times 256 \times 2$	128	200	25

4 Machine Learning

As I have stated in the Introduction as well, there is two well known definition out there for Machine Learning. As Arthur Samuel described it informally informal in '85: "the field of study that gives computers the ability to learn without being explicitly programmed." [18]

Or how Tom Mitchell tried to explain: "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." [13].

Example: playing checkers.
E = the experience of playing many games of checkers
T = the task of playing checkers.
P = the probability that the program will win the next game.

Not so surprisingly neither of them can be seen as a perfect definition. For every problem, every question leads us to a new solution, and with that, maybe a slightly different definition could be find. In general, let's say any machine learning problem can be assigned to one of two subsets: classifications: Supervised learning or unsupervised learning.

4.1 Supervised Learning

We talk about Supervised learning, when the set of the output have a relationship with the output, and we already know that what values can be expected and valid. Therefor we have the ability to validate our output.

Supervised learning problems can be subdivided into "regression" and "classification" problems. In a classification problem, we are trying to predict results in a discrete output. In other words, we are trying to map input variables into discrete categories like a finite set of neutral numbers. Instead when we talking about a regression problem, we are predicting the results within a range of continuous output, meaning that we are trying to map input variables to some continuous function.

Example 1:

Given a real estate market where we try to predict the price of a house, given the data about the size of the place. Price as a function of size is a continuous output, so this is a regression problem.

We are able to turn this example into a classification problem. Instead of a continuous output, let's make our output about whether the house "sells for more or less than the

asking price." This way we created exactly 2 well separated sets of discrete classes based on price.

4.2 Unsupervised Learning

In case of Unsupervised learning we approach the problem with little to no idea what our results should look like at the end of the learning process. We can derive structure from data where we don't necessarily know the effect of the variables, how many outputs we can find, or what is the relationship between input and output if there is any. This means that at the end of any learning cycle, we can not have the feedback based on the given results and the performance of the system. We have to derive structure by clustering the data based on previously may unseen relationships among the variables in the data in a trial and error way.

Example:

Non-clustering: The "Cocktail Party Algorithm", allows you to find structure in a chaotic environment. (i.e. identifying individual voices and music from a mesh of sounds at a cocktail party).

Clustering: Take a collection of 1,000,000 different genes, and find a way to automatically group these genes into groups that are somehow similar or related by different variables, such as lifespan, location, roles, and so on.

4.3 Model Representation

If we would like to give an explanation with notation to a previously shown example, we should use: x^i to denote the "input" variables (living area in this example), also called input features, and y^i to denote the "output" or target variable that we are trying to predict (this was the price of the house). A pair, like

$$(x^{(i)}, y^{(i)})$$

is called a training example, and the dataset that we'll be using to learn. The list of m training examples $(x(i), y(i)); i = 1, ..., m$ is called a training set. Note that the superscript "(i)" in the notation is simply an index into the training set, and has nothing to do with exponentiation. We use X to denote the space of input values, and $Y$to denote the space of output values. In this example, $X = Y = R$.

To describe the supervised learning problem slightly more formally, our goal is, given a training set, to learn a function $h : X \to Y$ so that $h(x)$ is a "good" predictor for the corresponding value of y. For historical reasons, this function h is called a hypothesis. Seen pictorially, the process is therefore like this:

When the target variable that we're trying to predict is continuous, such as in our housing example, we call the learning problem a regression problem. When y can take

on only a small number of discrete values (such as if, given the living area, we wanted to predict if a dwelling is a house or an apartment, say), we call it a classification problem.

5 CNN - Convolutional Neural Networks

A Convolutional Neural Network is a deep learning algorithm, which is able to assign importance with learnable biases and weights to various aspects or objects on an input image, and this way it is able to identify, classify, or in any other way differentiate one from another. While in other more primitive algorithms the pre-processing is hand-engineered and therefor more lengthy, after enough training a CNN is able to learn the same characteristics on it's own. CNNs can be used in a lot of different fields, like image and video recognition, image classification, medical image analysis, and even from recommender system to natural language processing.
Convolutional Neural Networks are inspired by biological processes. The original idea is coming from the 60's research done by Hubel and Wiesel, regarding vision of mammals, like cats and monkeys. [6] They shewed in this paper, that in a human like eye and nervous system individual neurons respond to visual stimuli only in the receptive field. A very restricted region. However these receptive fields of different regions partially overlap so that the entire field of vision can be covered.

CNNs have an input layer, and output layer, and hidden layers. The hidden layers usually consist of convolutional layers, ReLU layers, pooling layers, and fully connected layers.

- Convolutional layers apply a convolution operation to the input. This passes the information on to the next layer.
- Pooling combines the outputs of clusters of neurons into a single neuron in the next layer.
- Fully connected layers connect every neuron in one layer to every neuron in the next layer.

This way we can eliminate the need of feature extraction by hand, therefore the efficiency of the algorithm can be significantly better.

5.1 About the layers

5.1.1 Convolutional layer

To give a far overview of a general CNN architecture we have to start with the usually first layer in a CNN which is a Convolutional Layer. Here we use a so called filter (or sometimes referred to as a neuron or a kernel), which will highlight and select a receptive field from our input. The filter itself is just as an array on numbers, as the input image,

and these numbers called weights, or parameters. The depth of the filter has to match with the depth of the input image, otherwise we will get a mathematical error. After we have chosen the filter size, we also have to choose the stride and the padding.

We will move the filter matrix thought the input image, and convolving, multiplying the values in the filter with the original pixel values of the image, basically computing element wise multiplications. These multiplications will be summed up to get a single number. Now, we repeat this process for every location on the input volume. Every unique location on the input volume produces a number. After sliding the filter over all the locations, we will get an array of numbers, which we call an activation map or feature map.

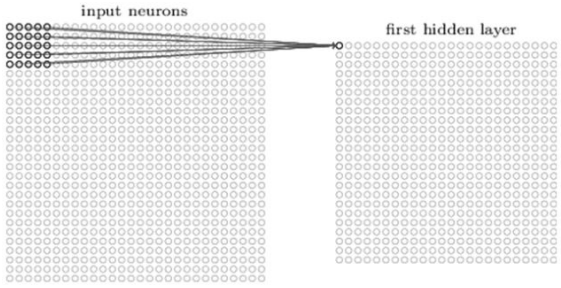


Figure 5.1. Visualization of a 5x5 filter convolving around a 32x32 input volume and producing a 28x28 activation map [16]

In these convolutional layers we will find the feature extraction, or identification itself. With the help of different filter matrices things like straight edges, simple colors, or curves of any kind can be found and highlighted to pass onto the next layer. Basically when we make the convolution with a part of the input image which has similar features what we have in our specific filter, then we will get higher values in our activation map. And similarly if we do not have a specific feature in the examined region if the input then the results for the corresponding area in the feature map will be insignificant. The more filters we use, the greater the depth of the activation map, and the more information we have about the input image at the end of a convolution. [16]

We have to mention here, that in this example the input of the imaginary convolution was an actual image, however in a real CNN, convolutional layers can follow each-other. That means we will have activation maps as inputs instead of images to make the multiplication and summation with the filters and get another activation map. This way of stacking of convolutional layers we are able to get activation maps that represents higher level features. Types of these features could be semicircles (combination of a curve and straight edge) or squares (combination of several straight edges). As we go through the network and go through more convolutional layers, we get feature maps that represent more and more complex features. After only a few layers deep in our network, we may have some filters that activate when there is handwriting in the image, or filters that activate when they see specific coloured objects, and so on. [23]

There are two main parameter that we have to be able to understand to have a proper control over the differences between the individual convolutional layers we will use in our

network. Namely, until now we only mentioned that the filter matrix have a size, however the movement of this smaller matrix of filter values in the input volume is still a question. To understand this, we will introduce the stride and padding definitions.

5.1.2 Stride

With the volume of stride we can control how the filter convolves around the input, that amount by which the filter shifts is the stride. Ideally stride will be set to get an output in the integer volume, and not a fraction

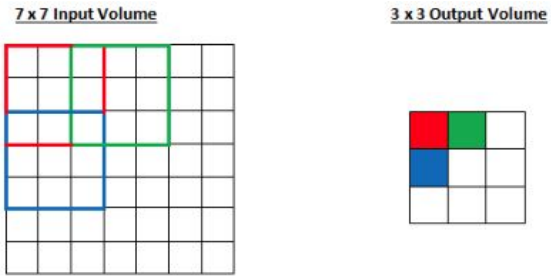


Figure 5.2. Visualization of a 4x4 filter convolving around the input volume and producing a 3x3 output volume with stride set to 2

The formula for calculation the output size for any given convolutional layer would be the following:

$$OP = \frac{IP - F + 2P}{S} + 1$$

Where the OP is the output height divided by the width, IP is the input height/width, F is the filter size, P is the padding, and S is the stride.

As we can see, the output size will shrink with the bigger stride, as the inspection of the original input volume will be less frequent. This means that with a higher stride, we can save computational time, and even storage, however we lose some accuracy, as the overlap of the investigated filters will be smaller and smaller.

5.1.3 Padding

To avoid the shrinking of the size of the output we can use padding. This technique introduces new pixels around the edge of the original input image. The border provides space for annotations or acts as a boundary when using advanced filtering techniques. For example if we make a multiplication in the frequency domain it will be corresponding with circular convolution in the spatial domain. This means that without padding the image properly, results from one side of the image will wrap around to the other side of the image. If our filter is hanging over the right side of the image, it is wrapping back on the left side, so the opposite side of the image will have an effect on the bordering pictures. We want to avoid that, and it is possible with padding.

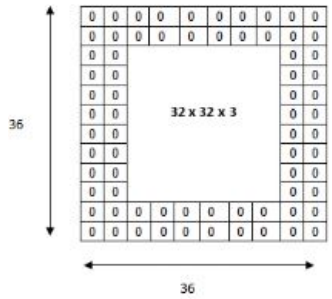


Figure 5.3. Zero padding of a 32x32 pixel wide image to have 2 borders of zeros

5.1.4 Activation function

After Convolutional layers, it is convention to continue with the application of a nonlinear layer (or activation layer). So far our system only computed linear operations in the previous convolutional layers like element wise multiplications and summations. In previous generations nonlinearity was achieved with the help of more complex functions, like tanh $f(x) = \tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$ and sigmoid $f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$, however researchers have found out that there is a just as effective, but much more efficient solution with the help of the ReLU (Rectified Linear Units) $f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ layer. It also helps to alleviate the vanishing gradient problem, which is the issue where the lower layers of the network train very slowly because the gradient decreases exponentially through the layers. With basic terms we can say, that this layer will change the negative activation values to zero, and with that it increases the nonlinear properties of the model and the overall network without affecting the receptive field of the convoltutional layer.[15]

5.1.5 Pooling layer

After the so called activation layer we usually have pooling layers also. Here we have to make a simple downsampling of our original input data. Similar to the previous layers, here we also have a lot of different choices to make this possible, and from those the maxpooling is the most popular. Here basically all that happens is that we take a smaller filter and as we apply this filter to the input image, we measure the selected region with it, and generate a downsampled output matrix. In case of the maxpooling, this means, that we select the highest value from the selected region of the filter, and we discard the rest of the elements.

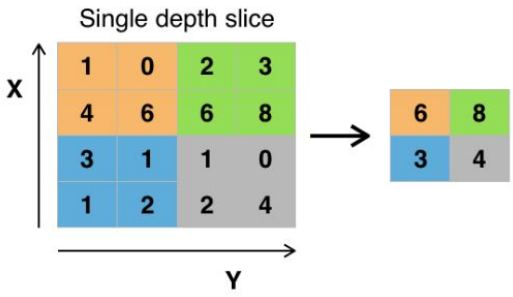


Figure 5.4. Example of Maxpooling with a 2x2 filter and a stride of 2

It is obvious, that this layer have the ability, to drastically decrease the size of the spatial dimensions, however have the potential, to consistently keep the specific feature that we have found in the previous steps. This is good for us in respect of the computational cost of the network, and it is able to control overfitting. This term means that it is possible to get a real good results while in training, so to say to model is tuned to the training data. Unfortunately when proposed to new and never before seen validation or test scenarios, the results will significantly decrease. The extraction of some of the spatial qualities of the features, we can show that the exact location of a given feature is not as important as the features relative location to each-other, which will be preserved after the maxpooling operation.

5.1.6 Dropout layer

This layer also provides some protection from overfitting, mentioned in the previous section. During the training steps, this layer is used to drop out a random set of activations from that layer by setting them to zero. This way we are able to build more robust and redundant networks, as we force the network to generalize, even if some of the outputs missing. Due to the randomness of the process, every time we take another training turn, other neurons will be missing, therefore the individual utility of a single neuron will not be reduced significantly, but the result of the raining can provide a less sensitive network to overfitting overall. [20]

5.1.7 Fully connected layer

Basically this is usually every classifier's last layer, where we take the input volume (whatever the we got as an output for all the previously used layers) and outputs an N dimensional vector, where N is the number of classes that we would like to choose from. (As an example we can imagine if we working on the classification of the handwritten digits, we will have an N = 10, as there are 10 different possibly digits out there.) Each number in this N dimensional vector represents the probability of a certain class. Basically, a FC layer looks at what high level features most strongly correlate to a particular class and has particular weights so that when you compute the products between the weights and the previous layer, you get the correct probabilities for the different classes.

5.2 Backpropagation

Until now we have not really talked about how all these layers actually learn. How do the optimize their weights to be able to successfully recognize edges or more complex structures like a shape of a human body. The most popular optimization strategy in machine learning is called gradient descent. When gradient descent is applied to neural networks, its called back-propagation. Backpropagation algorithms are a family of methods used to efficiently train artificial neural networks. The main feature of backpropagation is its iterative, recursive and efficient method for calculating the weights updates to improve in the network until it is able to perform the task for which it is being trained. [3]

A neural network is a function. We posses our input data, and after a series of matrix operations that are applied to that data we will produce an output. We can imagine this, if we look at a image classification problem, it must be obvious, that after a long serious of randomly set weights used in all the neurons, the output will be also random. If we have a lot of classes from where we would like to identify the input image. Data is propagated forward through the computational graph of operations. After one round in our network, the result vector should contain pretty much even values, as it was unable to make a proper guess on the right class. However in case of learning steps, we have the labeled solution, we can measure the received results and compute the error compared to the desired solution from the label.

The measurement of the the accuracy of the hypothesis function, or we can say that the initial weight values of any neuron in training is done by using a cost function. Basically it is a measure to show how far away a particular solution is from an optimal solution to the problem to be solved. There are many different functions what we can use as cost functions. In a general case we can say that: let's takes an average difference of all the results of the hypothesis with inputs from x's and the actual output y's.

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2 = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2$$

The most important to see here is the halved mean of the squares of $h_\theta(x_i) - y_i$ part, the difference between the actual and the predicted output.

This function is otherwise called the "Squared error function", or "Mean squared error". The mean is halved $\left(\frac{1}{2}\right)$ as a convenience for the computation of the gradient descent, as the derivative term of the square function will cancel out the $\frac{1}{2}$ term.

After the error is computed, it will be propagated backwards by computing gradients recursively for every set of layers in the neural network.

Let's see the algorithm with a small example. If we imagine a one of the simplest network that is possible, each layer with a single neuron in it. if we have the input layer two hidden layers and 1 output layer that will give us three weights and 3 biases to work with in this example network.

$$w_1\ b_1, w_2\ b_2, w_3\ b_3$$

Our goal is to understand how sensitive this a sot function to the changes in these

variables, and change them accordingly, and decrease the cost function with them, shall we say: study. Now we will focus our attention to the last two neurons activation functions namely $a^{(L)}$ and $a^{(L-1)}$ and the desired output, the given labeled training example y. The cost of this simple network for a single training example is $C_0 = (a^{(L)} - y)^2$. However we know, that the $a^{(L)}$ activation here is only the product of the previous activation like this:

$$a^{(L)} = \sigma(w^{(L)}a^{(L-1)} + b^{(L)})$$

A special nonlinear function, the weight and the bias. We want to understand how sensitive the cost function is to small changes in our weight $w^{(L)}$, or phrase differently, what is the derivative of the cost function in respect to the weight.

$$\frac{\partial C_0}{\partial w^L} = ?$$

However with the use of the chain rule, we are able to split this ratio, into a more manageable ratios multiplication as follows:

$$\frac{\partial C_0}{\partial w^L} = \frac{\partial w^{(L)}a^{(L-1)} + b^{(L)}}{\partial w^L} * \frac{\partial a^L}{\partial w^{(L)}a^{(L-1)} + b^{(L)}} * \frac{\partial C_0}{\partial a^{(L)}}$$

This right here, the multiplication of this three ratio gives us the sensitivity of the C_0 cost function to any small changes in the $w^{(L)}$ weights. This way we showed how one component of the gradient vector can be calculated. We can build that up from the derivatives of the cost function with respect to all those wights and biases that are concerned here. With this oversimplified example I was trying to show how the algorithm work, and with the application of the Chain Rule, how can be propagate forth and back we can calculate how sensitive the cost function is to previous weights and previous biases.

5.3 Transfer Learning

When we are trying to do image classification, or any other type of learning based algorithm with a computer, we have to start from ground zero every time. However, it is easy to see, that this is rather inefficient. If we wanted to model our brain, and the working of the neurons, we realize that our learning processes are always merged with experiences we had before, even from different fields. We had played before we touched a pencil, and we have made a lot of silly drawings before we even started to learn how to form a character. Same can be said about machine learning. All image processing tasks should start with the ability to find edges on a picture. After that curves, objects, background and foreground can be revealed. These are all crucial parts no matter what kind of images we are looking at, therefore we do not need to start our training with our precious, and maybe not so widely available data, but on whatever images we can find, and after learning the basics, we can further specialize to our explicit field of study.

The definition of transfer learning is given in terms of domain and task. The domain D consists of: a feature space X and a marginal probability distribution $P(X)$,

where X=$(x_1, ..., x_n)$ in X. Given a specific domain, D=$(X, P(X))$, a task consists of two components: a label space Y and an objective predictive function $f(\cdot)$ (denoted by T=$(Y, f(\cdot))$, which is learned from the training data consisting of pairs , which consist of pairs (x_i, y_i), where x_i in X and y_i in Y. The function $f(\cdot)$ can be used to predict the corresponding label, $f(x)$, of a new instance x. [12]

Given a source domainD_S and learning task T_S, a target domain D_T and learning task T_T, transfer learning aims to help improve the learning of the target predictive function $f_T(\cdot)$ in D_T using the knowledge in D_S and T_S, where $D_S = D_T$, or $T_S = T_T$.

5.4 Data Augmentation

Usually one of the hardest things in deep learning is to get an appropriate, well labeled, and good quality input data in big amount for training purpose. With data augmentation we are able to extend our limited dataset, and multiply the number of potential inputs of the training function. In case of pictures, if we imagine the exact same picture with a horizontal or vertical rotation, we can just as easily recognize the same object on it, however, for a learning algorithm, with limited input, this small change can carry lot of new and valuable information for future test cases. We have the same image, the same unchanged label, however, from a mathematical point of view we have an entirely different matrix representation for that information, and this can be almost as good as entirely new labeled images to train from. Some artificial ways to expend our dataset can be grayscales, horizontal flips, vertical flips, random crops, color jitters, translations, rotations, and much more.

6 CNN Literature Review

6.1 AlexNet

This paper is from the authors Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton, and its title is "ImageNet Classification with Deep Convolutional Networks" [10], but everyone knows it by the short name: AlexNet. It is one of the most cited article in the field of machine learning with a total of 14,000 times and is widely regarded as one of the most influential publications in the field. It was created back in 2012, and with the presentation of this work, the convolutional neural networks have been presented to the world of image classification, and with the available computational power at the time they were able to perform significantly better then any other competitor until then. They won the ILSVRC (ImageNet Large-Scale Visual Recognition Challenge), the Olympics of machine learning, and ever since the success of the CNN type architectures is unquestionable. Alex and his team was able to perform on the tests with the error rate of 15.4% while the second best competitor had 26.2% for the same error. The difference is significant, that is obvious even now, however with today's knowledge, the 15.4% error rate seems to be pretty high. Especially considering the fact, that it is the so called Top 5 error which is the rate at which, with a given image, the model does not output the correct label with its top 5 predictions.

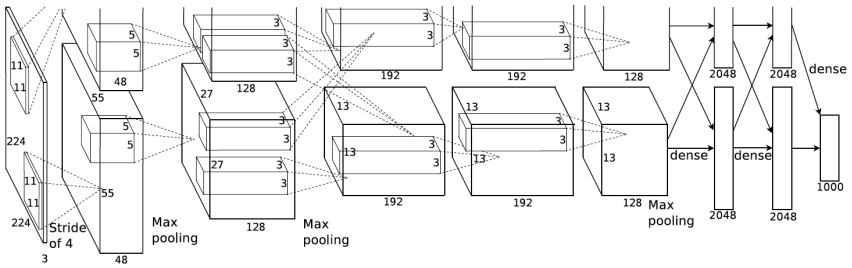


Figure 6.1. Schematic diagram of the Alexnet with two different streams at the same time, as due to computational restrictions they used two GPUs at the same time

Looking through the diagram of the architecture it is also obvious, that they used a pretty simple layout compared to modern systems. It only contains five convolutional layers, dropout and maxpooling layers, and three fully connected layers. Still to save on computation, they managed to run two streams on two separate GTX 580 GPU at the same time for five to six days.
Beside showing the power of CNNs they introduced in this paper some of the special

techniques of machine learning even used today. For example the use of ReLu for the nonlinearity function showed that even at the first sight too simplistic function can free up valuable computational time and power while still producing acceptable results. They used data augmentation, the technique I also presented in more details previously. With this, they were able to multiply the number of available training images, as after the translations, horizontal reflections, and patch extractions all training data was introduced as almost new pictures to train from to the system. Lastly but not least they have showed that adding some random dropout to the layers the overfitting is successfully manageable.

6.2 ZF Net

Right after the success of AlexNet, the writers of the paper called "Visualizing and Understanding Convolutional Networks"[23] won the ILSVRC 2013. They used a pretty much similar architecture as presented in the previous section, however they made some interesting modifications worth to talk about here. As the title also suggest, they were able to visualize the previously never seen before filters and weights. It is called DeConvNet because it maps features to pixels (the opposite of what a convolutional layer does).

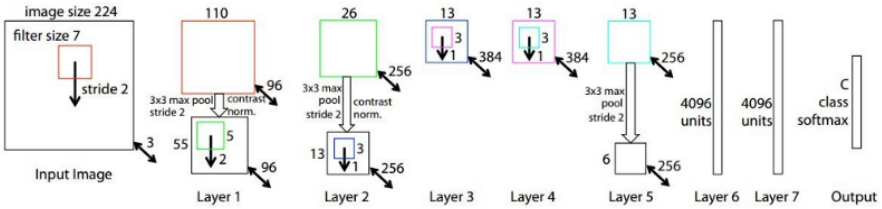


Figure 6.2. The ZFNet architecture was similar to what we have seen in the AlexNet, but with tweaked hyper-parameters, and slight modifications.

Matthew D. Zeiler and Rob Fergus used only 1.3 million image to train their network, which is approximately only the one third of the images with the augmentation, used to train the AlexNet. However they achieved better results due to the small changes they made on the architecture and its working. First of all the convolutions used smaller filter sizes. From 11x11, they reduced these to only 7x7, which means bigger activation maps, therefore more detailed feature extraction, more detailed original pixel information can be retained. They also gave much more time to their system to make adjustments, the AlexNet was trained for 6 days, whilst the ZF Net for the double of that, twelve days.

As I mentioned before this paper[23] had another important impact, the never before have seen visualizations they were able to provide from the inside of their architecture. In every layer of the trained network they had a path back to the image plane with the state of the activation maps.

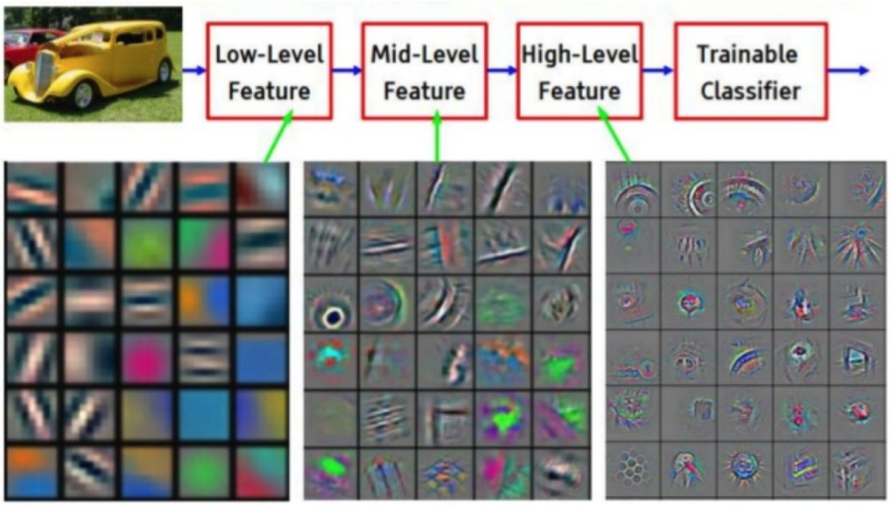


Figure 6.3. Visualization of three different level of layers

As we can see here, as we going deeper and deeper into the layers, the more complex activation maps can we work with. While at the beginning we only have simper edges or colours, later on the shapes start to appear, and even further the more circular features are being detected, and on the last one, the highest level special forms of the car, like tires can be noticed. This new visualization technique made this paper really stand out from its competitors.

6.3 VGG Net

From the year 2014, the paper called "Very Deep Convolutional Networks for Large-Scale Image Recognition"[19] written by Karen Simonyan and Andrew Zisserman. The trend what was started in the previous years continued with the success of this paper. However in this year the system represented here was not the winner of the ILSVRC 2014, it is till worth to have a better understanding about the importance of this paper.

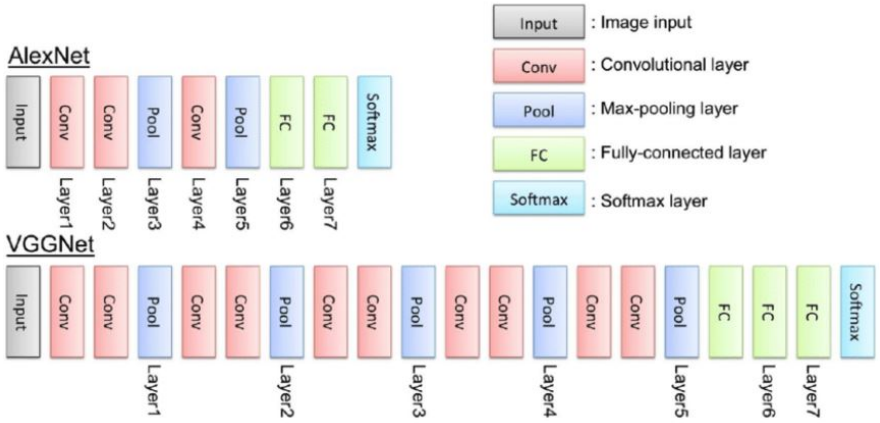


Figure 6.4. A schematic comparison between the smaller architecture of the AlexNet, and the similarly build but deeper and bigger VGG Net

The number of layers has grown to 19, whilst the filter size shrink to a really small 3x3, with only 1 stride and padding. However they started to combine multiple filters at the same time, so they were able to keep the benefits of having a rally small filter size, but three times the 3x3 filter have the simulation of the effective receptive size of a 7x7 filter. Also the number of filters doubles after each maxpool layer. This reinforces the idea of shrinking spatial dimensions of the activation maps, therefore winning runtime, but growing in depth. The most important takeaway from this article is that convolutional neural networks have to have a deep network layer in order for the better representation abilities of visual data. They made a bigger, deeper system, while they made it more simple the inside, and the results were quiet impressive with 7.3% error rate in the ILSVRC 2014 tests.

6.4 GoogLeNet

GoogLeNet is a 22 layer CNN and was the winner of ILSVRC 2014 with a top 5 error rate of 6.7%. The paper itself is called "Going Deeper With Convolutions"[21], however, despite the title may suggest otherwise, the most important factor of this article that the writers has proven that the bigger is not necessarily better, and we do not have to only grew the number of the layers. We have to measure the computational costs, the power any memory usages, and they presented a sophisticated architecture which is able to perform exceptionally good, while does not inevitably much bigger in size.

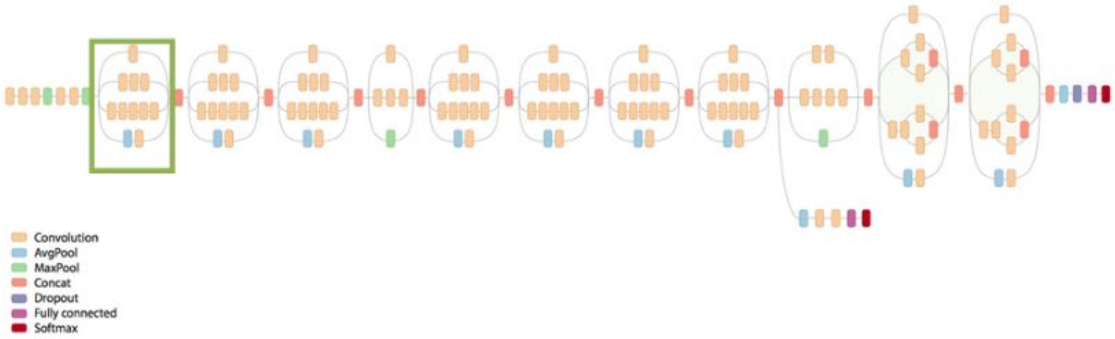


Figure 6.5. A schematic architecture of the GoogLeNet, were the green box highlighting one of the Inception module out of the 9 they used.

They used the so called Inception module, which will be an architectural solution to present parallel computation to the CNN's architecture. Until so far we only have seen sequential working, convolutional layers followed by pooling and so on. In this paper Christian and her colleges were able to perform these operations in parallel.

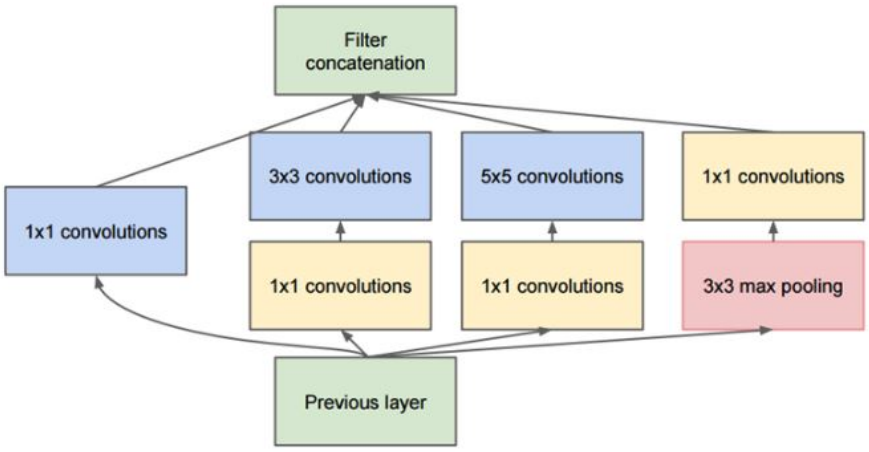


Figure 6.6. The inside architecture of one of the inception modules whit the parallel computations of the different filters and the pooling

As we can see, at the place of a single convolutional layer, we are able to perform more then one convolutions and a pooling at the same time. The authors were possible to do this with adding the 1x1 convolution before the 3x3 and the 5x5 layers, thus provide a method of dimensionality reduction. This means that the medium sized filter convolutional and the large size filter convolutional does not have to deal with as large of the volumes. This is basically a pooling of features. After all the convolutions inside one inception module, all the individual outputs will be concatenated together to form the input of the next layer.

6.5 ResNet

Microsoft ResNet is the architecture presented in the paper from Kaiming He and his team, titled as "Deep Residual Learning for Image Recognition"[4] in 2015. Whit this architecture, they were able to score a stunning 3.6% error rate, which is obviously better then anything we have sen so far, but we have to mention that depending on the skill level and experiences of the test person, a human generally gets an error rate around 5-8%. The ILSVRC 2015 has been won by the architecture, which contains the most layers so far: 152.

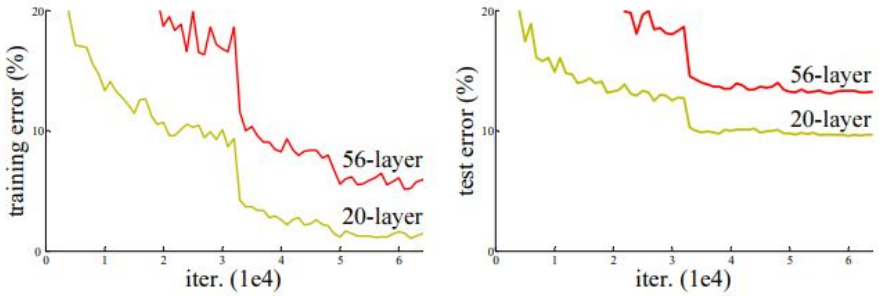


Figure 6.7. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer “plain” networks. The deeper network has higher training error, and thus test error[4]

Interesting to note what the authors claim that a naïve increase of layers in plain nets results in higher training and test error. They even tried their own network which is able to handle 152 layers effectively, to how does it performs with 1202 layers, but their test accuracy decreased. After this the presumption of the previous paper is proved, that we can not just stack layers on-top of each other blindly, groundbreaking architectural solutions have to support the growth as well.

6.6 MobileNet

'MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications'[5] is an article from 2017. It is presenting a lightweight architecture. The most interesting part of the presented research is that it is using a depthwise separable convolution solution on each, individual colour channel rather the flattening of the combination of all three together. Whit this they achieve kind of a filtering of the input channels. Or as the authors explain: "For MobileNets the depthwise convolution applies a single filter to each input channel. The pointwise convolution then applies a 1×1 convolution to combine the outputs the depthwise convolution. A standard convolution both filters and combines inputs into a new set of outputs in one step. The depthwise separable convolution splits this into two layers, a separate layer for filtering and a separate layer for combining. This factorization has the effect of drastically reducing computation and model size."

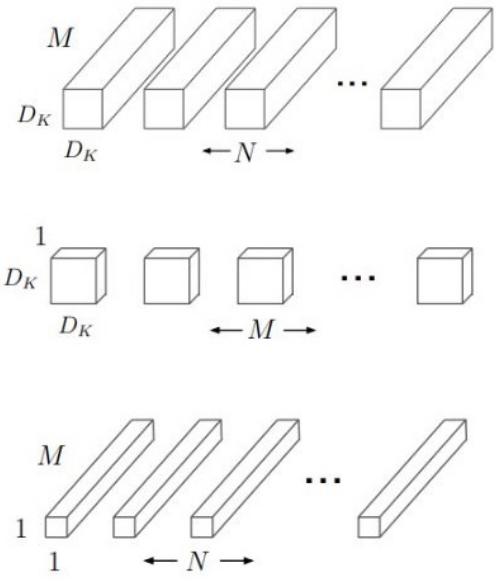


Figure 6.8. The first row the standard convolution filter, the second row the depthwise convolutional filter, and the third row would be the pointwise convolution

The overall architecture of the MobileNet is the following. It has 30 overall layers, and after the normal convolution it has the depthwise layers, pointwise layers following each other one by one. these last two layer is basically replacing the standard convolution to build a depthwise separable filter. It is able to work really fast and compared to that with low maintenance costs.

7 Implementation

7.1 Introduction

With the explosive development of medical imaging technologies, as their incidence grew and prices reduced the number of radiological images increased exponentially. Radiologist have to report the finding, however it is getting more and more difficult with the ever expanding load. In recent years the number of images waiting for inspection more than a month increased drastically that may mean a serious risk for the patients too. Radiological departments of hospitals request support of the teleradiology stations.

Data sciences and convolutional neural networks perform more and more successful in image processing and their potential use in the field of radiological imaging seems to be optimal to reduce the workload of radiologists and diagnostic errors.

The images can be processed automatically by AI models as they arrive to the teleradiology station. This way the sorting, potential prioritization by the probability of the presence of lesions on the given image is possible. Unfortunately the DICOM labels of the images can be absent or nonstandardized. An image processing model therefore is indispensable - which is able to automatically classify images as they arrive to the centers - before the test-specific modules can start the analyzing of the images. This work of thesis tries to give a solution for this problem, and in the system of Sineko company, feed and AI module with classified images.

7.2 Sineko

The Sineko Global Informatikai Ltd. was started by the present COE, Dr. Németh Bence as a startup after he finished his studies at Pázmány Péter Catholic University on the Computer Science Engineer branch and Image Processing and Computer Vision (IPCV) specialization. As he also has qualifications and personal connections on the field of medicine and medical community, he was able to experience firsthand what limitations and needs doctors have now days. With his company he tries to offer solutions to some of these.

Right now in his team he has 15 people, from various background, centred around medical professionals, business developers and engineers. The work that was most interesting to me is of course from the engineering part, and we could subdivide that into four more categories, front-end, back-end development, solutions for the integration with the hospitals, and experimentation with machine learning. As before the Master program I had the opportunity to work for the General Electric Healthcare here in Budapest, I had some experiences with development works and structures in multinational compa-

nies. My work involved working mostly on back-end, on the data access layer of a piece of a medical software, where we had to frequently switch to work out solutions for the presentation layer, the so called front-end.

Whit these in mind, and considering I am at the last semester of the IPCV specialization myself, I felt the urge to choose to orient myself towards the other two possible branches, and after the theoretical classes in Budapest and Bordeaux to have a closer look on how hands on solutions can be developed in a real life environment. And as there can not be any solution without integration, and can not be deep learning without proper amount of clean data I had to work on some of the problems and challenges with the integration with the medical centers also. That is what is presented in this section of my thesis.

7.3 GRAID

Sineko Global main software is the GRAID. It is a web-based application, where radiologists will be able to get a better user experience while working on their otherwise repetitive tasks. Instead of dictation or typing they will be able to make their findings with only a few clicks. Whit this they will hopefully regain some time and will to focus on solving more meaningful problems. The already existing teleradiology station architecture can use this application as a module. Tat is why one of my task was to get more familiar with the technological background and the working of these stations. The GRAID software is supporting the structured creation and handling of medical findings, lowers the rate of error, and opens the possibilities of the use of artificial intelligent projects, which needs clear and well labeled data. This software aims to produce that much faster and easier than anything ever before.

7.4 The environment

After I decided to join to the Sineko team, and I got the details of the kind of work I had to be able to make for them, and for my thesis, I was starting to look for the possible and available hardware and software solutions. First I made some small exploratory works for the possible libraries and APIs. It was pretty clear from the beginning, that I would like to keep things as simple as possible. Both in sense of programming, and before that environment wise. And I had to realize, that right after I pass the toy problem solving level with Python, Windows will not be reliable enough for more serious application development. Switching to a 18.04.2 LTS Ubuntu, the latest long term support system, I had a lot of pleasure working with Python[11]. I also used for the development JET BEANS IDEs, for the Python the PyCharm, and for Java, the IntelliJ latest community versions. PyCharm handles virtual environments for all new projects. It has some extra features, like making a venv folder for one project, but that can be imported to another one any time. For my Machine Learning module I was using Keras[8] 2.2.4 front-end with tensorflow 1.13.1 back-end as it is user friendly, modular and runs seamlessly on CPU and GPU if necessary. To try out the integration with the medical centers I was in need of a medical center simulation. This is provided by a software called Orthanc the latest

version is the 1.5.4+dfsg-1 with the release: disco (19.04). It is a simple, yet powerful standalone DICOM server, which I will use to mock a real server in a medical center, containing DICOM data, and able to handle requests as presented later. To handle the AI side of these requests I will make a REST API based application, developed in Java with openjdk 11.0.2

7.5 Simulation of the DICOM Server

After I installed the Orthanc, it has became a startup program of my Ubuntu system, so looking at the http://localhost:8042/ we can find the application's explorer.html page. This is where the DICOM depository can be manage, plugins can be set up and selected, and even to try the REST calls with a basic graphical interface under the menu "Query/Retrieve". I needed to set up a few important things in the configuration file of the Orthanc, located at /etc/orthanc/orthanc.json. The first thing was under the "DicomModalities" glossary, to modify the list of the known DICOM modalities, the exact name, IP address and the port number of the future DICOM client, that will be running on my computer form the Java application presented later. Beside this the Configuration of teh HTTP server, where I put the HTTP port for the REST services and for the GUI to the "HttpPort" : 8042, and the Configuration of the DICOM server remained. Here the port had to be set up as the following: "DicomPort" : 4242 and "DicomAet" : "ORTHANC" this last entry will be the name on what the application will be able to call the server. After having these necessary configuration steps, the java programming cloud be started, to test out the API, and the Server responses.

7.6 Requests from Java

As I presented in the previous section the most important setup parameters to start my own DICOM server, I started to look for possibly already existing solutions for the REST API calls, which will be started from Java. I found one of the repositories called dcm4che/quickstart from the author Gunter Zeilinger <gunterze@gmail.com>, which he produced and publicised in Jul 2018. All the presented works where I try to explain how it is working will be Java pseudo code, and are based on that module.

What I was doing here, making a connection to the locally running Orthanc server, then sending out a request to it to get a list of the available images stored on the server by type. After that I ask for these findings one by one, and the AI module can be started based on what we have got back. These are the main steps that the Sineko have been working on, and wanted me to have a better understanding on. I try to show the interesting parts here:

Listing 7.1. Defining DICOM endpoints

```
private Connection conn;
private Device device;
```

```
conn = new Connection("127.0.0.1",
    "DICOM_PORT", "PROTOCOL", "AE_TITLE:ORTHANC");
device = new Device("127.0.0.1", "11112", "DICOM", "MLKSTTH");

device.setDimseRQHandler(createCStoreSCP());
```

In the Listing 1 I am making the connection, the fields are filled to reflect the setup that was given in the Orthanc json file. We need the local host as an address, a DICOM port, the used protocol, and the AE title. Essentially the Connection is where this code is looking for the image server, and the Device will be our Java code, which will try to connect.

Listing 7.2. Getting the list of records from the server

```
AAssociateRQ assocReq = new AAssociateRQ();
assocReq.addPresentationContext(new PresentationContext(
        UID.StudyRootQueryRetrieveFIND, UID.ImplicitVR))

atts = new Attributes();
atts.setString(Tag.QueryRetrieveLevel, VR.LO, "STUDY");
atts.setString(Tag.Modality, VR.CS, "DX");
atts.setString(Tag.StudyInstanceUID, VR.UI);

DimseRSP response;
response = assocReq.cfind(UID.StudyRootQueryRetrieveFIND, 0x0000,
            atts, UID.ImplicitVR);
```

Whenever we need something from the Orthranc, we have to set that what is the purpose of the given connection, that is what happening in the AAssociateRQ. After that we have to set the exact parameters, what we would like to get from the server. The setString methods third parameter controls that the given parameter will or will not be a filter criteria. In this example we ask from the "study" level of the DICOM architecture, the images with "DX" modalities, which represents the "**D**igital **X**-ray". In the response we get the StudyInstanceUIDs of all the available findings that matched with the previous criterion, as in the case of the third setString, there was no third parameter passed to the function. This is how it knows what to gave back. Into the response, we will get the server's response after the cfind call.

Listing 7.3. Getting the list of records from the server

```
List<String> studyUids = new ArrayList<>();
response.getDataset().forEach(study -> .... {
      studyUids.add(study.getString("StudyInstanceUID"));
});
```

After the response arrived, we have to extract the "StudyInstanceUID"-s for each available DICOM, that are stored on the server. This is needed, as the actual transfer request will have to be called individually specified whit this parameter.

Listing 7.4. Requesting of the studies from the Orthanc

```
assocReq.addPresentationContext(new PresentationContext(
    UID.StudyRootQueryRetrieveMOVE, UID.ImplicitVR));

atts.clear();
atts.setString(Tag.QueryRetrieveLevel, VR.LO, "STUDY");

for (sIUid : studyUids) {
    atts.setString(Tag.StudyInstanceUID, VR.UI, sIUid);

    assocReq.cmove(UID.StudyRootQueryRetrieveMOVE, atts,
        UID.ImplicitVRLittleEndian, "localAE=MLKSTTH");
```

In this block the most important call would be the "cmove", as this will be the actual request, which now knows what to ask from the server, actually starts to send out the call. However before that the "assocReq" has to be changed accordingly. In the second listing it was created as a UID.StudyRootQueryRetrieveFIND type, and here, after the use of the cfind we need it as a StudyRootQueryRetrieveModelMOVE, to be able to use the two connected system for cmove. The UID is an enumeration class in the toolkit, and these elements return a DICOM code. That is what the Orthanrec expects and understands.

Listing 7.5. Requesting handler that can accept the incoming data

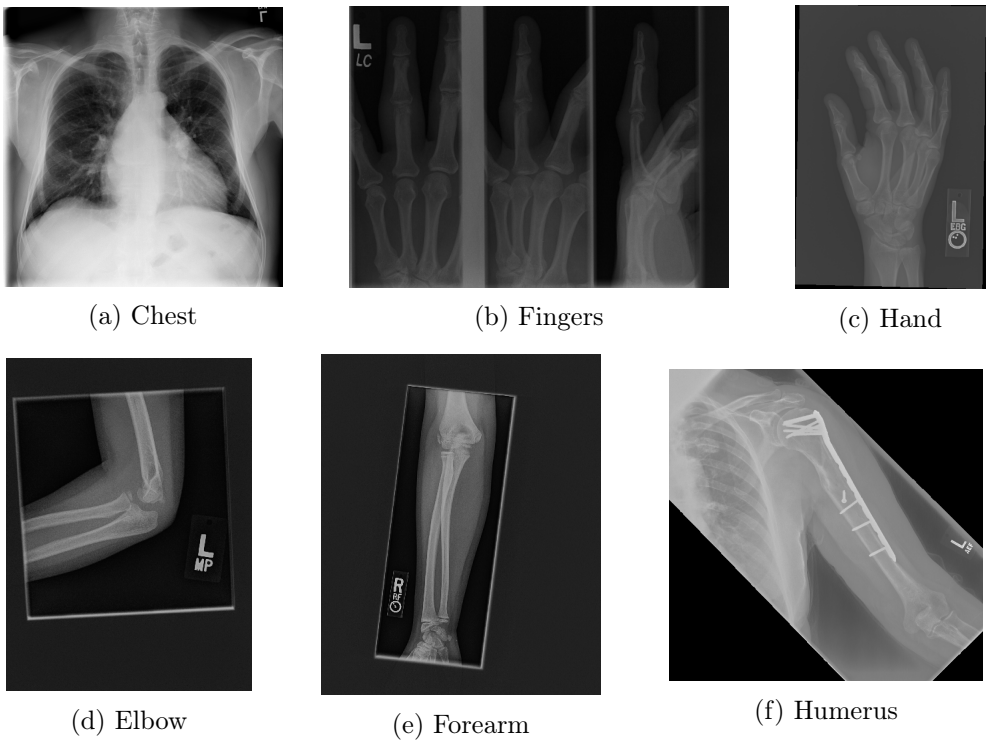
```
private DimseRQHandler createCStoreSCP() {

    protected void store(Association as, PresentationContext pc,
        Attributes req, PDVInputStream body) {
            cStore(pc, req, body);
        }
    return dicomServReg;
}
```

Listing 5 would be a separately running listener, that is responsible to handle the requests. It was instantiated in the Listing 1. Here is the code snippet that would run on creation. The response have similar attributes to a HTTP request, the one named body will be used to retrieve the image information.

7.7 Datasets

7.7.1 X-Ray

First I started to work with the MURA (musculoskeletal radiographs) dataset[17], which is a collection of roughly 50000 bone X-rays. The original goal of the creation was to try to find differentiation signs of abnormal and normal recordings. Therefor the labeling of the dataset is done by placing the JPG files into different folders, not only by the normal and abnormal images, but also by what is on the image. That is why this set of pictures was easy to start whit for me. After a registration it is publicly available, and can be train and test with with pretty much ease. However it turned out that one of the most important task for that the Seniko would like to be solved, is to have a classifier for images with chest data on them. The seven available class that I was able to get with the MURA was the following: Elbow, Finger, Forearm, Hand, Humerus, Shoulder and Wrist. I had to find another source, however the solution was not so hard, as there is a similar Stanford Machine Learning Group handled dataset called CheXpert [7]. This contained the similar, around half a megabyte sized X-Rays, and I only had to pick a similar amount, to match the number of images in all the folders representing the individual classes of my learning algorithm.



(a) Chest (b) Fingers (c) Hand

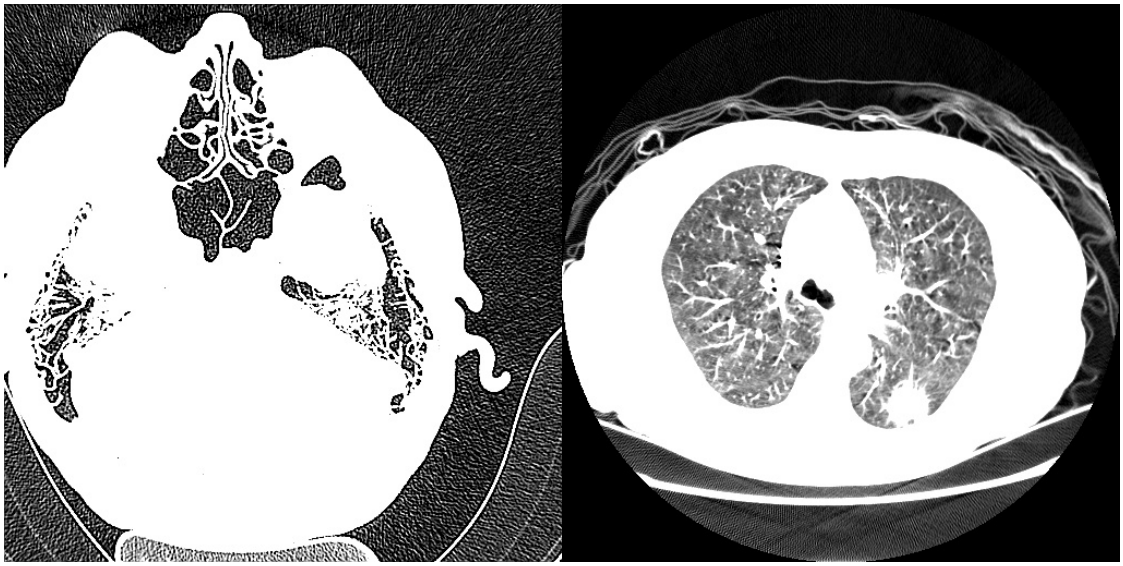
(d) Elbow (e) Forearm (f) Humerus

Figure 7.1. Samples from the x-ray images of the MURA and CheXpert database

7.7.2 CT

Beside the x-rays the classification of 3 dimensional data types like CT or MRI were also a part of my task. Due to the lack of proper training and test datasets, I had only time to

work with CTs until I finished this paper. These CTs however they carry information that can be turned to the third dimension, they are stored as slices. This means much more image, from a certain examination, and every consecutive image is a slightly different picture from the examined body part. As we did not had capacity, to work on three dimensional convolutional networks, I had to stick with the processing of the slices. In my case, this meant a more time consuming preprocessing step for the CT images, as they had to be reorganized, and sorted out. They are stored in a structurally similar way to what I have showed in the case of x-rays, and what I will explain further in the next section. CT imaging data also usually contain a few exploratory x-ray imaging, in same other cases, only a few slices of a body part, to give the radiologists a better understanding what kind of setup parameters they should use for the real, more time consuming examination.



(a) Head (b) Lung

Figure 7.2. Samples from the CT images I used for the training of my classifier

7.8 Pre-processing

Beside the Kears own data augmentation possibilities provided by the ImageDataGenerator I also had to write some own code. Working with the DICOMs, that were publicly available thought the CANCER Image Archive (https://www.cancerimagingarchive.net/), I had ended up with a lot of differently structured and questionably reliable data. The before mentioned ImageDataGenerator's flow_from_directory() function was quiet useful. It is able to read the input through complex library structures. However, as it gets the path of a directory, it automatically assumes, that inside that directory, it will find the sub-directories of all the classes that it have to work with.

DICOM images have the following directorial structure:

```
Patient identifier
└── Study
    └── Sirious
        └── DICOM files
```

As I have presented earlier, the DICOM files contains many different kind of information beside image data. Therefore before I could had them to the Keras API, I had to extract these graphical data. My code is able to decompile PNG and JPG files from the DICOMs through this method:

```
for n, image in enumerate(images_path):
    ds = dicom.dcmread(os.path.join(DICOM_path, image))
    pixel_array_numpy = ds.pixel_array
    if PNG == False:
        image = image.replace('.dcm', '.jpg')
    else:
        image = image.replace('.dcm', '.png')
    cv2.imwrite(os.path.join(OP_path, image), pixel_array_numpy)
```

It goes thought all DICOM files in the given folder, and with the pydicom's dcmread function it makes a pixel array from each file. After that, it concatenates the file with the proper extraction, based on the user selection at the beginning of the script (this means that we have to know before running the code, what kind of output we would like to get).

(a) Head CT first slice (b) Lung exploratory image for setup

Figure 7.3. Samples from CT images that I had to get rid of, as they are either just not useful, or even not necessary pictures

As I mentioned I also had to get rid of some of the folders, containing unwanted, or otherwise not related picture. For example head CT-s usually have some blank images,

a few slices before the examination reach the head of the patient, or where it is only barely touches. Similarly, the exploratory images were a problem, that I had to get rid of. Partially i did these by hand, looking through what information value and usefulness do some of the pictures have, or by a short Bash script, like the deletion of the .xlm file from the folder containing all of the slices. These kind of files often used as a list, with which the iteration thought the images made possible, however I did not needed these, as the Keras have a convenient solution for this exact problem.

7.9 Classifier code overview

In the AI module that I have been working I was using a pretrained network for feature extraction. The first one that I have found inside the Keras, was the MobileNet:

```
base_model = MobileNet(weights='imagenet',include_top=False)
```

With this I import the mobilenet model, with the preset weights for the imagenet and discards the last 1000 neuron layer. Whit this I can have a well trained network, and the time and computing power I have to spare for the runs are significantly lower. The lower and middle level feature will be extracted, and I only change the dense layer.

```
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024,activation = 'relu')(x) #dense layer 1
x = Dense(1024,activation = 'relu')(x) #dense layer 2
x = Dense(512,activation = 'relu')(x) #dense layer 3
preds = Dense(8,activation = 'softmax')(x) #final layer
```

I add dense layers so that the model can learn more complex functions and classify for better results. The final 4 layer changes to specifically be trained for my input data at the end of the network. The last layer has to be a softmax, with the equal number of layer that we would like to see as output for the network. I was training here for 8 different kind of X-Ray image types.

```
model = Model(inputs=base_model.input,outputs = preds)
```

This is how the actual modal is created, based on the pre-trained feature extractor part, and my own dense layers together.

```
train_datagen = ImageDataGenerator(preprocessing_function=preprocess_input)
train_generator = train_datagen.flow_from_directory('/train',
    target_size = (64,64),
    color_mode = 'rgb',
    batch_size = 32,
    class_mode = 'categorical',
    shuffle = True)
```

Keras ImageDataGenerator is a build-in preprocessing tool which is able to open and multiply the image data if necessary for making possible better learning capabilities of the system with data augmentation. Most often the Image datasets available on the internet are either has images placed under folders which has their respective class names (this was the case for me, as I have explained it before) or placed under a single folder along with a CSV or JSON file which maps the image filenames with their corresponding classes. Both of these can be handles with this tool with ease.

```
model.load_weights('mod_wChest_4lay_VGG19.h5')
```

```
model.save('mod_wChest_4lay_VGG19.h5')
```

These are the load and save function of the Keras model representation. I was usually running only one epoch of training at a time, to be always able to change the hyperparameters according to the most recent results. If the weights are loaded back to the same structured CNN model, then the next run can refine the results in each loop.

```
adam = Adam(1e-4)# Adam optimizer
model.compile(optimizer=adam,loss='categorical_crossentropy',
                metrics=['accuracy'])
```

Adam[9] is an adaptive learning rate optimization algorithm that's been designed specifically for training deep neural networks. Its name is derived from adaptive moment estimation, and the reason it's called that is because Adam uses estimations of first and second moments of gradient to adapt the learning rate for each weight of the neural network. Usually for the first epochs I was initializing it with 1e-3, and later on, as I was getting closer to the top precision, I tried to change it to be smaller.

```
step_size_train = train_generator.n // train_generator.batch_size
hist = model.fit_generator(generator = train_generator,
                   steps_per_epoch = step_size_train,
                   epochs = 1)
```

The Keras deep learning library includes three separate functions that can be used to train your models, and however all three of these functions can essentially accomplish the same task, there are some small differences between how versatile are they:

- .fit: Is the most lightweight from the three. We use it if we know that the raw data itself will fit into memory — we have no need to move old batches of data out of RAM and move new batches of data into RAM. Therefor we only use it, if we know before runtime, that the whole dataset will fit into the memory. And furthermore, we will not be manipulating the training data on the fly using data augmentation.
- .fit_generator: The one I used usually, and a more flexible one but still pretty easy to handle. It handles the built in data augmentation, and epochs if necessary.

Notice that here we need to supply a step_size_train parameter when calling the fit_generator function. The Keras data generator is meant to loop infinitely — it should never return or exit. Therefore, we compute the step_size_train value as the total number of training data points divided by the batch size. Once Keras hits this step count it knows that it's a new epoch and it can exit the loop if necessary.

- .train_on_batch: function accepts a single batch of data, performs backpropagation, and then updates the model parameters. The typically use of the .train_on_batch function when we have very explicit reasons for wanting to maintain our own training data iterator, such as the data iteration process being extremely complex and requiring custom code. It was unnecessary to use in my case.

```
with open('history_wChest_4lay_VGG19.json', 'w') as f:
    json.dump(hist.history, f)
```

If the returned history volume is saved after the .fit_generator function as it is on my previous code snippet, it is possible to save out the results in these arrays to a standard json file and evaluate it if necessary.

8 Results

8.1 X-ray

The CNN I first used to train on 8 classes of x-rays was using a feature extracting module, therefore I had not build all the layers of a complex convolutional neural network architecture. I used the VGG19 architecture[19], with pretrained set of weights called 'imagenet'. The ImageNet project was aiming to provide a large image database for research purposes. More then 14 million image is inside it right now, with 20000 classes to identify. This is the best available pretrained set of weights, as the purpose was also classification, however on a much wider range of classes in RGB colour space. With these in mind the generalization abilities inherited from training on these tremendous set should be giving us a great generalization capability. I changed the dense layers to represent my own 8 classes (chest, finger, hand, elbow, forearm, humerus, shoulder, wrist), the feature extraction in the lower levels of the architecture proved most useful.

In the dense layer, I added a global average pooling layer, and I started the training with a layer containing 1024 neurons with ReLu activation functions, in the next layer 512 neurons and sigmoid activation, and for the last fully connected layer I had 8 neurons, to produce the 8 probability percentage of the classes. The network was able to use totally 3,228,864 parameters without my dense layers, and with them it grow to a number of 4,771,464 trainable parameters. From the sum total of the found 38608 images belonging to 8 classes. This means that in each of the classes I had an average of 4800 individual image. This is still can not be considered a huge dataset however originally from the Mura and CheXpert folders I only had roughly 1000 x-ray per class. I let Keras do the data augmentation, and that was able to provide me with this impressive almost 5 times more images to train from.

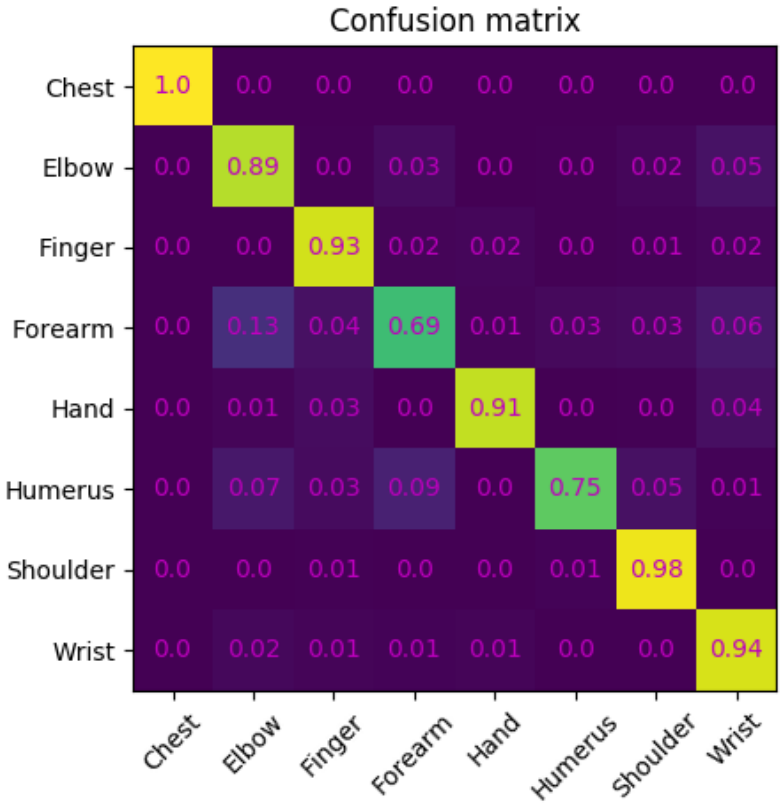


Figure 8.1. Confusion matrix of my implementation with the VGG19 architecture.

I was doing the training with a target size 64x64, so all the input images will be resized to this value in pixel height and width. This allows me to be able to gain speed, and as I have made several tries, this, or even 32x32 pixels are enough to see similar results. I even tired 16x16 pixel, however that seemed to be too small, some of the valuable information must have been lost, as the train usually stacked about a 50% accuracy. The batch size was chosen to be to 32, with shuffled input images, to avoiding the procession of same patient's images together and in categorical class mode, which is a necessary pick next to the categorical crossentropy loss function and the Adam optimizer set to a bit lower then the default (1e-3), to (1e-4), as even with this, the speed of the network's convergence was too fast.

I also tried the MobileNet[5] architecture, with pretrained weight on the imagenet, and with all the same other parameters, to have a better understanding of the effect of the underlying built of the system. It seemed to be an optimal choice, as its speed and power consumption is very low, compared to other networks, while still performed well in this classification tasks.

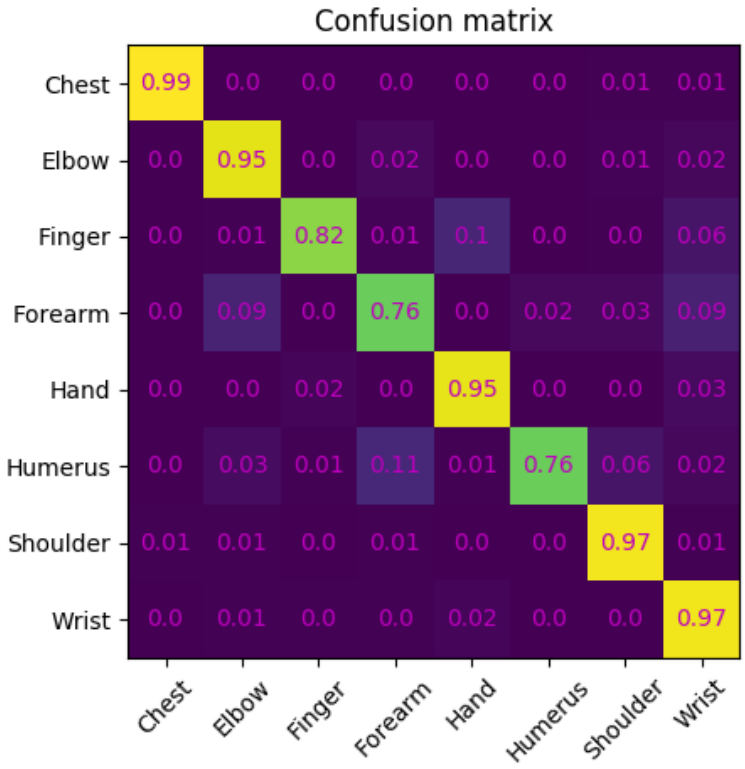


Figure 8.2. Confusion matrix of my implementation with the MobileNet architecture.

Both of 8.1 and 8.2 matrices shows, that I was able to achieve high accuracy in the main diagonal, where we see the true positive results. The validation set was a separable image collection made by the previously mentioned two x-ray data provider, MURA, and CheXpert. The number of images per class is between 200 and 600, and their quality should be considered similar to what we had for training. For a more detailed presentation here are the precise numbers:

Classes	**#image**	**Ratio of data**	**VGG19 acc**	**Mobilenet acc**
Chest	155	0,04624	1,0000	0,98709
Elbow	465	0,13872	0,8881	0,94623
Finger	461	0,13753	0,9305	0,81561
Forearm	301	0,08979	0,68770	0,76079
Hand	460	0,13723	0,90652	0,95217
Humerus	288	0,08591	0,75347	0,76388
Shoulder	563	0,16795	0,97513	0,97158
Wrist	659	0,19659	0,94081	0,96509

From these if we summarize the validation accuracy, and compare it to the accuracy I got during the training of the model:

valid:	0,89707	valid:	0,90662
train:	0,94840	train:	0,97120
diff:	0,05132	diff:	0,06457

Here we can see that my training did some overfitting. I was running 3 epochs, so the training have seen all images 3 times, however in different batches (32 picture together), and with shuffle on, so the other 31 must have been from different examination on different patients. Still, the VGG19 had 5% the MobileNet 6.5% of overfitting to the training data.

However the final validation accuracy of my model with transfer learning based on the MobileNet architecture was higher by 1% and shows a total of 90% accuracy on the validation data, even together with the plus 1.5% more overfitting on training. Therefor I choose to stick with this architecture, and started to look for different optimization algorithms, to reduce the number of epochs needed for the training process, so the overfitting can stay under control hopefully.

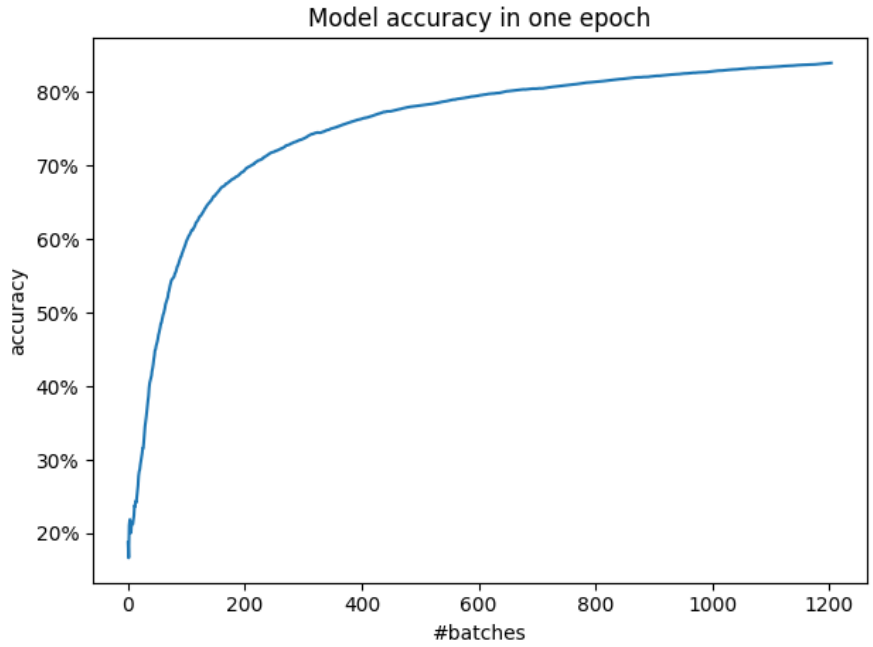
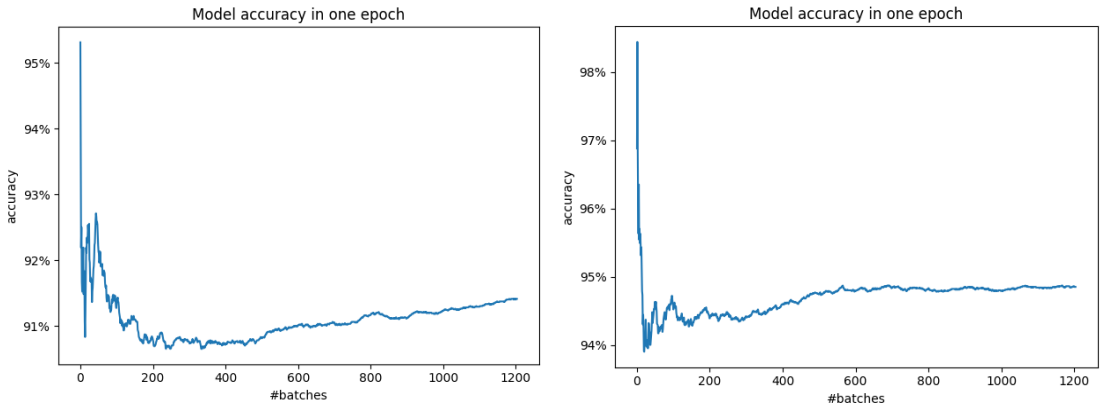


Figure 8.3. Learning rate of the MobileNet architecture with the Adam optimizer

Adam is an optimizer that is usually used for much bigger dataset then mine, and it needs a certain amount of time to get velocity towards an optimum, therefore it is possible that it jumps over the minimum of a function, and needs time to stop and turn back again. On much bigger datasets it is less of a problem, however, here it resulted that I needed to make 3 epochs, to make it converge.



(a) 2nd epoch (b) 3rd epoch

Figure 8.4. The following 2 epochs of the same MobileNet run with Adam optimizer

All of the presented methods have a really fast approach to a high accuracy on the training data. This is not surprising if we consider how nice results these architectures had on much complex RGB image classification problems, like the ILSVR Challenge, with thousands of classes.

Therefore the 8.3, and the following 2 figures shows the accuracy of the model after each batch. This means that we see a cumulative summation of the results, the nth batch accuracy is influenced by all the previous results in the same epoch. Therefor while looking at the 8.4a and 8.4b figures, if in the first batches we are "lucky", we can have better results then what we can consider realistic. On the 8.4b picture, at the 3rd epoch the accuracy clearly stuck at 94% what I have showed earlier, that is why the training have come to a stop.

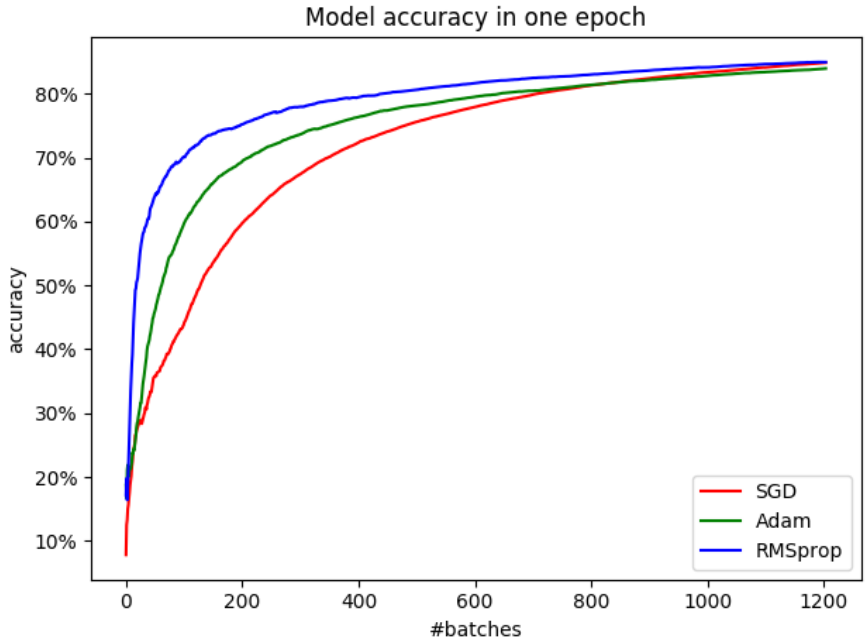


Figure 8.5. The accuracy of 3 different optimizer going through the batches in one epoch

Interesting to compare here the learning rate of another optimization method, I tried the stochastic gradient descent (SGD)[1], and RMSprop[14]. SGD is an older method, so I had not have high expectations, however it is popular and widely used as it is a simple and effective method. The RMSprop optimizer is similar to the gradient descent algorithm with momentum. The value of momentum is denoted by beta and is's default setting is 0.9.

All the above examples used categorical crossentropy which is widely considered the best available loss function for image classification problems. I made same test runs, checking other possibilities available in Keras, however I found poor results, like the categorical hinge which resulted a disappointing convergence at 68% accuracy after the same 1204 batches that I always used for x-ray classification.

8.2 CT

Building on the results of the x-ray classifier, I made some experiments on other medical imaging data like computer tomography images. The model was able to produce exceptionally nice results. Using only 1 epoch learning the training batches will never contain the same image twice, therefor the overfitting is out of question.

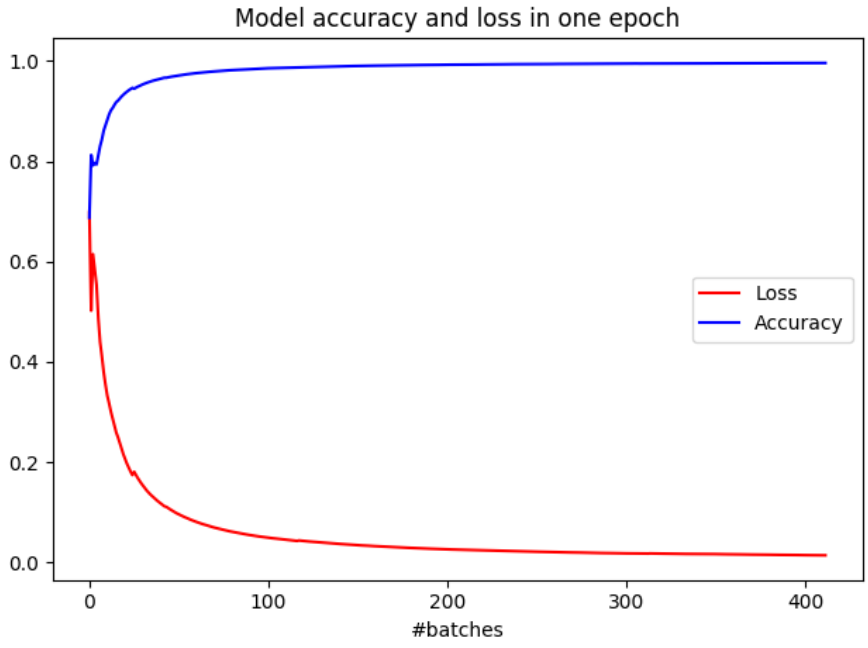


Figure 8.6. The accuracy and loss function values after 1 epoc on a binary CT image classifier

As we can see here, the same architectural model behave extremely well. The accuracy rises rapidly the first batch (with size of 32 images) have a 68% accuracy, as it is a binary classifier, however in not more then 14 steps it passes the 90% and does not stop until the end of the epoch at an impressive 99.6%.

In the training set I had 13223 images belonging to 2 classes. That is why the number of batches on the 8.6 figure is more than 400. This is an unfortunately small training

sample and in the future I will have to try to get more data, to further analize this architecture for CT images.

	#images	accuracy
Lung	778	1
Head	556	0,920863

valid acc: 96.7%

As we can see the accuracy is high, I implemented a binary classification here due to the limited amount of CT pictures that was available for me. The CT pictures needed an extra amount of time spent with preprocessing, as it was frequent that I encountered with bad labels even in the training sets, and generally too small number of images both for training and validation purposes.

9 Future Works

9.1 One-Shot Learning

One of my tasks during this thesis work was to suggest a solution for the company to use a pretrained model without new learning phases, even in cases where new classes would be introduced to their system. In case of what we have seen so far that sounds pretty much impossible, the dense layers of the CNN learn the specific features of the training data images from the preciously labeled dictionary system representing the possible cases, and generate a vector with probability values for all possible classes, no more, no less. If the number of desired outputs change, classic neural network must be retrained. However, we humans work differently, we are able to generalize, to learn from only a handful of examples, or even only from one, hence the title of this section. One-shot learning tries to achieve just that. With the help of a so-called similarity function, we would be able to decide the degree of difference between two images.[22]

With the help of this function, we can start to measure distances between our existing class of representative images (the labeled data), and the input we would like to recognize. When examining and comparing the results, we can see either that all the of the volumes are too big, and this must be a new image from an unknown class, or that one of the classes have a given probability to be the right answer for the recognition problem. It is easy to see that this process does not necessitate a lot of retraining, even when we introduce new classes These kind of networks will never be as good and confident in classifying our existing classes with enough data. Nonetheless, it is possible for not just huge companies with seemingly unlimited source of data in private internal sets to be able to train their models well.

9.2 Better Results

While working on this thesis, I always used transfer learning, and with this approach I was able to save time and power consumption on my part. However, after seeing the results, I am sure that one of the best next step towards making the real models that Sineko will be able to use in their software will be running my scripts on serious hardware. I mostly done both of the coding and test runs in my home, therefore the privilege of the usage of the pretrained weights, the refusal of tenability on most of the network layers was necessary to be able to progress. Whenever the hardware environment and availability enables, after loading in the pretrained weights, the whole network should keep learning on a possibly bigger dataset. This is a way by which I could have achieved better results compared to the presented ones.

10 Summary

I examined the state of the art concerning architectures from the scientific publications on the historically most successful and interesting CNNs, and from these experiences and from what I have learned through the lectures during my study, I made a small overview about them and about the most important parts of the convolutional neural networks in general.

I got hands on knowledge about the architecture of the DICOM standards, and how they are used in the present to collect, store, and share medical imaging information. I made a small overview on how the mentioned imaging hardware works, and what results they can provide. I made an effort to try the server and application side programming of these complex structures, to see how the Sineko's own solutions are working, and what is the route, on which my classifier algorithm got the images it had to work with.

I trained several classifier models, and based on the results, I was trying to make them even better. Faster, less hardware consuming, and able to work with the small amount of data I had to use for train and evaluate. I showed that the model I use is able to work on multiple different set of medical images, like x-ray and CTs, and it can be used as a solid starting point for an industrial application.

Bibliography

[1] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[2] Bernard Gibaud. The dicom standard: a brief overview. In *Molecular imaging: computer reconstruction and practice*, pages 229–238. Springer, 2008.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[5] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[6] David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106–154, 1962.

[7] Jeremy Irvin, Pranav Rajpurkar, Michael Ko, Yifan Yu, Silviana Ciurea-Ilcus, Chris Chute, Henrik Marklund, Behzad Haghgoo, Robyn Ball, Katie Shpanskaya, et al. Chexpert: A large chest radiograph dataset with uncertainty labels and expert comparison. In *Thirty-Third AAAI Conference on Artificial Intelligence*, 2019.

[8] Nikhil Ketkar. Introduction to keras. In *Deep Learning with Python*, pages 97–111. Springer, 2017.

[9] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[11] Dave Kuhlman. *A python book: Beginning python, advanced python, and python exercises*. Dave Kuhlman, 2009.

[12] Yuan-Pin Lin and Tzyy-Ping Jung. Improving eeg-based emotion classification using conditional transfer learning. *Frontiers in human neuroscience*, 11:334, 2017.

[13] T Mitchell. Machine learning. In *McGraw Hill.*, page 2. 1997.

[14] Mahesh Chandra Mukkamala and Matthias Hein. Variants of rmsprop and adagrad with logarithmic regret bounds. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2545–2553. JMLR. org, 2017.

[15] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[16] Michael A Nielsen. *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA, USA:, 2015.

[17] Pranav Rajpurkar, Jeremy Irvin, Aarti Bagul, Daisy Ding, Tony Duan, Hershel Mehta, Brandon Yang, Kaylie Zhu, Dillon Laird, Robyn L Ball, et al. Mura: Large dataset for abnormality detection in musculoskeletal radiographs. *arXiv preprint arXiv:1712.06957*, 2017.

[18] Arthur Samuel. Some studies in machine learning using the game of checkers. In *IBM Journal of Research and Development*, pages 210–229. 1959.

[19] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[20] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[21] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[22] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in neural information processing systems*, pages 3630–3638, 2016.

[23] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.