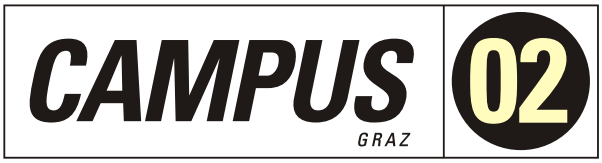
Master's thesis

Detection of Epiphyseal plate fractures by machine learning

ausgeführt am



Studiengang
Informationstechnologien und Wirtschaftsinformatik

Von: Philipp Kahr
Pers. Kennz. 0810319004

Graz, am February 24, 2021

.....
Philipp Kahr

Ehrenwörtliche Erklärung

Ich erkläre ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die benutzten Quellen wörtlich zitiert sowie inhaltlich entnommene Stellen als solche kentlich gemacht habe.

.....
Philipp Kahr

Danksagung

Der größte Dank gilt meiner Familie, vor allem meiner Mutter und Oma, welche mich ständig während des gesamten Studium unterstützt haben. Ohne die beiden wäre es nicht möglich gewesen diesen Weg erfolgreich zu beschreiten. Zudem, gilt auch mein Dank, meiner Freundin, Annalena, die mich des Öfteren bei den vielen Lernstunden unterstützte. Emil Spreitzer, MSc. war Tag und Nacht erreichbar und machte mehr als einfach nur Korrekturlesen. Zu guter letzt, möchte ich mich bei meinem Masterthesisbetreuer Dr. Michael Georg Grasser, bedanken.

Graz, am February 24, 2021

Kurzfassung

Jeder fünfte Mensch leidet an muskoskeletalen Erkrankungen. Gesundsheitssysteme auf der ganzen welt sind durch die Diagnostik, akute Behandlung und Nachbehandlung unter extremen Druck. Radiologieabteilungen sind durch die immense Anzahl an Patienten überlastet und sind von Leistungsinduzierten Erschöpfungen geprägt. Zwischen 11% und 27% aller Knochenbrüche werden fehldiagnostiziert. Dies ist besonders schädlich wenn es um eine falsche Entscheidung in der Pediatrie geht, da das Knochenwachstum beeinträchtigt wird. Dementsprechend sind besonders gefährdet Wachstumsfügenbrüche da die direkt das Knochenwachstum beeinflussen. In den letzten Jahren haben sich immer mehr digitale Lösungen in Krankenanstalten etabliert. Machine learning und neurale Netzwerke sind zwei spannende Themen die mehr und mehr Aufmerksamkeit auch in klinischen Applikationen finden.
In dieser Arbeit vergleichen wir derzeitige state-of-the-art machine learning Modelle für die Anwendbarkeit mit Wachstumsfügenbrüche. Unser Datensatz erstreckt sich über 21557 Röntgenbilder, welche in ein Training und Validierung Datensatz. Insgesamt sieben verschiedene Modelle wurde überprüft und getestet. Ein misch Ansatz aus ShapeMask und SpineNet erbrachte die beste Leistung mit einer korrekten Vorhersage von 80,7%. Alle Modelle, inklusive SpineNet, ResNet, ein eigens erstelltes convolutional neural network, faster R-CNN und Azure Custom Vision AI, konnten im 70% bis 80% Rahmen die Vorhersagen korrekt treffen. Nur MobileNetV2 zeichnete sich als nicht anwendbar für diesen Anwendungsfall ab, da es lächerliche 40% erreichte.
Keines der angewendeten Modelle konnte akkurater als ein Radiologe sein. Allem in allem, haben wir eine gute Übersicht über die möglichen Anwendungen von aktuellen Modellen in der Bilderkennung für Wachstumsfügenbrüche, erstellt.

Abstract

Every fifth human suffers from musculoskeletal disorders. Diagnosis, treatment and aftercare of those conditions provide a severe problem for the healthcare system around the world. Radiology units are highly occupied with patient demands and in consequence, radiologists are especially prone to suffer from fatigue. Overall 11% to 27% fractures are misdiagnosed. This is especially harmful during childhood since inappropriate treatments are futile for further bone development. Certain care in this perspective is epiphyseal fractures, which are directly linked to bone growth.

In recent years more and more computational driven diagnosis methods find their way into clinical diagnosis due to hardware and software advancements. Particularly the evolvement of machine learning algorithms for image analysis in any kind of application is intriguing for clinical usage to support radiologists.
Here, we compare state-of-art imaging analysis machine learning modells for their applicability to predict epiphyseal fractures. Using 21557 X-ray images, split into training and validation datasets, from the Radiology Department of the Medical University of Graz were used to train seven different commonly used models. We found that A custom ensemble model of ShapeMask and SpineNet yielded the most accurate prediction with 80,7%. Most tested models, including SpineNet, ResNet, MobileNetV2, a custom convolutional neural network, faster R-CNN, and Azure Custom AI provided an accuracy of 70% to 80%. Only MobileNetV2 turned out to be inapplicable for this use case, resulting in a meagre 40% accuracy.
None of the tested models was able to outperform the accuracy of radiologists. All in all, we provide a comprehensive overview of the possible utilization of currently available imaging analysis machine learning models and their possible use for epiphyseal fracture diagnosis.

Contents

1 Introduction

Around 1.7 billion people are affected by musculoskeletal disorders, which is a summarization of pain in ligaments, joints, bones, nerves, tendons. The WHO categorized musculoskeletal disorders into more than 150 diagnosis (Briggs et al., 2016). An example musculoskeletal disorder is the carpal tunnel syndrome. (Andersson & Watkins-Castillo, 2014) This number increased over time, and the amount of workload to be handled by all sorts of healthcare specialists is increasing, especially prone, are radiologists. Therefore, physician fatigue is becoming more of a prominent problem each day, which negatively impacts diagnostic accuracy. (Fitzgerald, 2001) This stretches even further when pediatric trauma care is present. Not only are pediatric fractures different from those found in adults, but they also have a greater variety. Reviewing orthopedic and surgical trauma literature reveals rates between 11% and 27%. The most common misdiagnosed fractures involve distal, radius, fingers, elbow, and proximal fibula. (Soundappan, Holland, & Cass, 2004) Around 15% to 18% of all pediatric fractures involve the epiphyseal plate. (Roehrborn et al., 1999)(Rogers, 1970)(Mizuta, Benson, Foster, Paterson, & Morris, 1987) In any case, when injuries are left undiagnosed and untreated, the cosmetic and functional consequences are futile. (George & Bixby, 2019) Physeal fractures themselves reveal to be challenging to diagnose correctly, as they take many shapes in X-Rays, even gold standard radiologists have difficulties finding them. Fractures classified by Salter-Harris I and II are the most missed epiphyseal fractures due to their occult appearances in X-Rays, (Rogers & Poznanski, 1994) as Bone development and growth plate might mimic injuries (George & Bixby, 2019). They are also known as the growth plate which is responsible for longitudinal bone growth. As those fractures interfere with proper bone development, a correct and timely diagnose is mandatory. (Mounts, Clingenpeel, McGuire, Byers, & Kireeva, 2011) Supporting radiologists by applying machine learning to radiographs is an up and rising trend. Current research shows a proper machine learning model can compete with board-certified radiologists, in exceptional cases, even beating them. (Rajpurkar, Irvin, Bagul, et al., 2017) while providing support to the radiologist in detecting abnormalities, such as fractures, is the utmost priority to tackle physician fatigue.

The first artificial neural networks from the 1950s vanished quickly to various reasons, such as insufficient data, lack of computing power, overfitting, and missing algorithms. In the 1980s, numerous Machine learning (ML) algorithms were designed and become accessible for classification tasks. (Lee et al., 2017) In the early 2000s, the first commercial applications of ML were introduced into health care. How-

ever, Computer-aided detection (CAD) systems produced more false-positive than humans. Therefore, CAD systems were viewed critically (Lehman et al., 2015), and the expected benefit of CAD systems was not found. (Fenton et al., 2007) Nowadays, ML is capable of overcoming the limitations of CAD systems. Soon, radiologists will be augmented by ML applications. We want to create an ML model that helps to identify one of the most common misdiagnosed fractures in pediatric emergency care and around 80% of misdiagnosis in the emergency room are fractures, which can result in increased pain, prolonged treatment, and loss of function. (Jarraya et al., 2013)

As mentioned above 15% to 18% of all pediatric fractures involve the epiphyseal plate, making it the single biggest type of fracture. (Mizuta et al., 1987) There are various studies that deal with classifying growth plate fractures in plain radiographs versus 3D images such as CT or MRI. Most of them have a common conses that 3D imaging is better than plain radiographs. (Lippert, Owens, & Wall, 2010)(Petit et al., 1996)(Smith, Rand, Jaramillo, & Shapiro, 1994) Diagnosing plain radiographs is challenging due to the pose any individual can have.

The current research leads to a vastly growing community with more and more data sets becoming open. Open datasets allow any individual to compete in finding the best solution. The MURA dataset (Rajpurkar, Irvin, Bagul, et al., 2017) allowed for an open competition. This resulted in a prospering community trying to outperform each other with a finely tuned ML model. We will work on a dataset and compare out of the box machine learning frameworks on their capabilities.

1.1 Research question, objectives, methodology

The primary goal of this research is to answer the research question: 'What can be seen as a successful machine learning framework in the context of medical image analysis, regarding Epiphyseal fractures?'.

Answering our hypotheses 'H1: Machine learning can be employed to diagnose Epiphyseal fractures.' and the corresponding antitheses 'H0: Machine learning cannot be employed to diagnose Epiphyseal fractures.' Will yield results to answer the research question. Multiple questions arise when answering the hypotheses.

- Current frameworks are sufficient to recognize and diagnose Epiphyseal fractures.
- Bone tissue segmentation is required to receive high accuracy.

A literature review is used to answer the most prominent questions and to get an overview of the current research and trends. Answering the research question is

done by using the κ mean error unit, which is widely used to compare ML models (Rajpurkar, Irvin, Bagul, et al., 2017).

As described above, by leveraging the power of machine learning, we want to build a model that is capable of detecting Epiphyseal fractures. Analyzing the current status of machine learning, artificial intelligence and deep learning models will yield that there are already some frameworks that exist and may be better suited than open source frameworks from the community. (Erickson, Korfiatis, Akkus, Kline, & Philbrick, 2017)

Targeting a specific children's disease for machine learning is challenging due to the limited available sample size. The data set provided by the Child Radiology Department of the Medical University of Graz, including X-Ray images of long bones. The machine learning model should be developed in multiple steps. In the first step, the data set has to be processed, standardized, and analyzed. The second step involves the applying of different already existing frameworks on the existing data set of epiphyseal fractures. Based on the results of already existing frameworks, further development of the model has to be done.

Due to heterogeneity of the data, it is necessary to standardize the data for further development and analysis. Standardization of the data involves multiple steps (1) alignment of the image information (2) trimming the size of the images (3) reduction of color (4) enhancement of the data set by applying common transformations. By implementing commonly used machine learning frameworks, a decision can be made on which model is the most promising to answer the research question with the accessible data at hand. Depending on the results the most suitable framework will be adapted to fully cover our needs. This will, for example, involve testing whether an unsupervised or supervised learning method is more appropriate. Once the implementation has reached a certain level, the model will be trained on a partitioned data set containing a training data set. For validation the model will be tested upon a data set it has never seen before.To evaluate our model, the data is compared to the data from already diagnosed images. The best will be defined as the successful machine learning framework and their implementation details will be referenced to answer the research question.

Comparing the most prominent open source ML frameworks and models will allow for a good evaluation basis.

1.2 Structure

A brief medical introduction into the field of Epiphyseal fractures can be found in the Chapter 2 Medical Background. This serves only as a primer for rudimentary

understanding of the medical implications that Epiphyseal fractures can have and why a proper detection is needed. It covers the types of Epiphyseal fractures and classes that they are divided into. It also highlights the error rate and other pediatric complications in relation.

The third chapter Machine learning is all about the state of the art machine learning techniques and frameworks. It covers the brief history of machine learning and explains current trends and depicts the most prominent scientific research. It covers topics from detecting lung cancer to removing bones on X-Rays or rebuilding entire skull shapes from MR images. Two sections target specific neural network designs that power machine learning technologies, while other summarizes existing machine learning applications and their outcome. Additionally, it describes three of the most prominent machine learning frameworks, which will later be used to compare their performance on epiphyseal plate fractures.

Fourth chapter Method is a deep dive into the configuration, programming, and management of the different ML frameworks. We are looking at three different open source ML frameworks that are the most prominent ones. Code snippets and some images from the working of the ML frameworks are displayed.

The fifth chapter Results discussses the results and the future outlook and additionally research that can be done in the field.

At the end Appendix contains various listings, figures and tables that can be selectively noted during the thesis.

2 Medical Background

Most children in emergency care are treated in consequence of traumatic experiences. The diagnosis process usually involves radiographic procedures, which can be immensely difficult in children. The major diagnostic challenge is to recognize subtle signs of osseous injury, which results in missed spotting of occult fractures on initial radiographs. Negative initial radiographic diagnosis is then confirmed by advanced imaging such as Computer tomography (CT) and Magnetic resonance imaging (MRI), as well as ultra sound and or nuclear medicine. Untreated fractures lead to prolonged pain loss of function and disability. (Jarraya et al., 2013)

2.1 Uniqueness of pediatric bones

An axiom often encountered in pediatric care is "children are not just small adults". This is especially true regarding the number of underdeveloped bones in the skeleton. Adult bones distribute any force put onto them in the following fashion. The force is allowed to travel through the bone, hopefully distributing it enough, so that the point of fracture is not reached. When the bone is fractured, it represents itself as a cortical discontinuity. However, children have more collagen and cartilage. Thus their bones are not as rigid as the ones from adults. One might even compare them to be a bit rubbery. (Frost & Schönau, 2000) This increase in elasticity and the underdeveloped bone structure makes children bones more prone to fractures. Depending on the circumstances, these fractures do not propagate similar to adult ones. (Little, Klionsky, Chaturvedi, Soral, & Chaturvedi, 2014) Another uniqueness occurring only in children bones is that the outer shell, known as the periosteal sleeve is proportionally stronger than the inner bone structure, known as the inner fibrous cortex. This fact causes fractures known as "the torus" or "buckle" fracture. Those fractures involve cortical deformity instead of discontinuity. The highly vascular structure is needed for the growth of the bone and is placed longitudinally. Usually, it manifests as a straight or undulating lucent band in X-Rays. (Wattenbarger, Gruber, & Phieffer, 2002) The entire physis is enclosed by a tough fibrous ring of Lacroix. Lacroix acts as a connector between the epiphyseal and metaphysis periosteum. The periosteum is the weakest link, and thus fractures often occur through the metaphysis. (Rathjen & Kim, 2014)(Dwek, 2010)

2.2 Fracture types

Pediatric fractures, particularly epiphyseal plate fractures, are diagnosed using the Salter-Harris style. In this approach fractures were originally categorized in five groups. Similar to any classification task Salter Harris depends on a reproducible and inter and intraobserver reliability. As any classification task must be based on a reproducible and inter and intraobserver reliability, the Salter-Harris method relies on landmarks. The impact of a physeal injury depends primarily on the localization of the fracture. Some impose a risk to stop growth and lead to deformity; however, not all of them impose the same high risk. Therefore a classification system which identifies patterns that might lead to any of those risks is desireable. (Cepela, Tartaglione, Dooley, & Patel, 2016) Salter-Harris purpose is to accurately describe physeal injuries with a prognosis relating to premature physeal closure (Salter & Harris, 1963). There are five types of Salter-Harris, explained in figure 2.1.

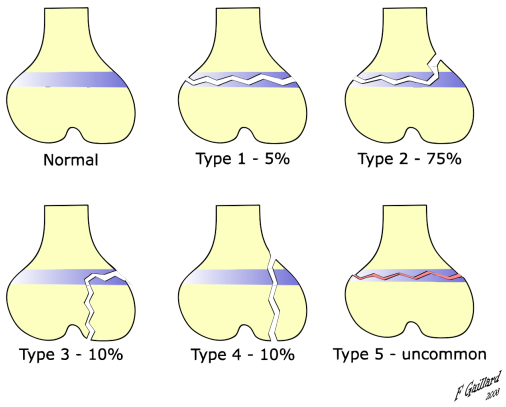


Figure 2.1: The Five basic fracture types of the Salter-Harris classification are shown. A Type I fracture is a separation through the physis. A Type II fracture enters in the plane of the physis and exits through the metaphysis. The resulting metaphyseal fragment is called the Thurston-Holland fragment (*). A Type III fracture enters in the plane of the physis and exits through the epiphysis. A Type IV fracture crosses the physis, extending from the metaphysis to the epiphysis. A Type V fracture is a crush injury resulting in injury to the physis.(Cepela et al., 2016, 2) Taken from `https://en.wikipedia.org/wiki/Salter%E2%80%93Harris_fracture#/media/File:SalterHarris.svg`on 21.09.2019 at 19:13 UTC+2, CC did by Dr. Frank Gaillard `http://www.frankgaillard.com/`

Cuts directly through the growth plate in a horizontal manner represent the first type and are more common in young patients, due to the thicker physis as shown in figure 2.2. (Cepela et al., 2016)

The second type is the most common one as it accounts for 74% of the physeal fractures. Figure 2.3 shows a fracture line that travels through the plane physis and exists at the metaphysis. The resulting metaphyseal fragment is called Thurston-Holland fragment.

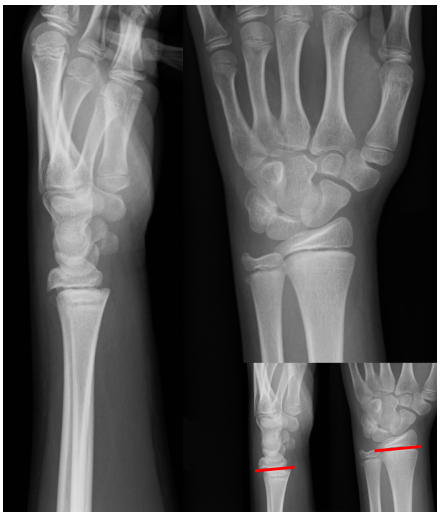


Figure 2.2: Salter-Harris Type I fracture of the distal radius.
Taken from `https://upload.wikimedia.org/wikipedia/commons/0/01/Salter_Harris_1_demo%281%29.jpg`on 21.09.2019 at 19:13 UTC+2, CC.

Figure 2.3: Salter-Harris Type II fracture of the ring finger proximal phalanx.
Taken from `https://upload.wikimedia.org/wikipedia/commons/7/76/Salter_Harris_2_demo.jpg`on 21.09.2019 at 19:13 UTC+2, CC.

The third type is a bit similar to the second type as it also enters in the plane of the physis and exits through the epiphysis. This results in an intra-articular fracture. The imposed risks of posttraumatic arthritis are directly connected to potential growth arrest. However, Salter-Harris Type III fractures are far less common than Type II. An example is shown in figure 2.4 (Cepela et al., 2016)

Figure 2.4: Salter-Harris Type III fracture of the big toe proximal phalanx.
Taken from `https://upload.wikimedia.org/wikipedia/commons/7/75/Salter_Harris_3_demo.jpg`on 21.09.2019 at 19:13 UTC+2, CC.

The fourth type fracture crosses the physis and runs through metaphysis and epiphysis. This fracture imposes a high risk for long-lasting effects such as growth deformity and subsequent asymmetric growth due to the formation of a transphyseal bony bar. As Salter-Harris Type IV fractures disrupt the entire physis and articular surface, the longitudinal stability cannot be ensured. Additionally, this might lead to a complete physeal arrest. Figure 2.5 shows an example. (Cepela et al., 2016)

Figure 2.5: Salter-Harris Type IV fracture of big toe proximal phalanx.
Taken from `https://upload.wikimedia.org/wikipedia/commons/b/b1/Salter_Harris_4_demo.jpg`on 21.09.2019 at 19:13 UTC+2, CC.

The fifth type are not fractures by definition; instead, they are defined as to crush injuries due to compressive force. There is a bit of discussion amongst authors if such fractures exist. Peterson et al. questioned their existence (Peterson & Burkhart, 1981) while Rathjen et al. concluded that such occult fractures are possible (Rathjen & Kim, 2014). Salter-Harris Type V fractures might occur due to stress-related load, such as a young gymnast performing stunts and repetitive pressuring the overly extended wrist. Figure 2.6 shows a radiograph with such a Salter-Harris Type V fracture. (Cepela et al., 2016)

Figure 2.6: Salter-Harris Type V fracture near the proximal radius. The small arrows mark the fracture line, whilst the bigger one represents the path of the applied force. Reprinted with permission from Wolters Kluwer Health, Inc. Reprinted with permission from (Cepela et al., 2016)

3 Machine learning

Machine learning (ML), Deep learning (DL), Artificial intelligence (AI) are keywords used in every aspect of life. Artificial neural networks (ANN) has been around since the 1950s. They quickly vanished due to the lack of sufficient training data, as well as insufficient computing resources. Since big data and the enhanced computing power has increased over the last few years, this trend has reappeared. Additionally, today, ANN involve novel algorithms that surpass human capabilities in visual and auditory recognition. This suggests that ANN is a leading technology that will potentially revolutionize the way health care is working. ANN is capable of detecting, classifying, and finding references. As health care has to deal with more and more medical imaging techniques and data, ANN may be ported to help with the workload. (Lee et al., 2017)

ML is a specific set of methods that detect patterns in data. This detection allows ML to predict future data and enables decision making. Decision making is not bound to full information anymore, because based on the detected patterns, the ML is deciding which direction is best. ML itself is a subset of AI. (Murphy, 2012) ML is data-driven and needs minimum intervention from a human.

Deep learning is part of ML and a particular type of ANN. Deep learning is the most promising approach to health care. As described above, in 1950, the ANN suffered from overfitting and vanishing gradient, both impacting the learning capabilities of the neural network. Nowadays, most of those limitations do not exist anymore. Deep learning already beat humans in the field of biomedical images. (Rajpurkar, Irvin, Bagul, et al., 2017)(Nakata, 2019)

As ML leverages big medical data sets, questions around ethics arouse. ML has the capabilities to diagnose, predict diseases, and optimize treatments. However, those applications are highly personalized and need to leverage a massive amount of personalized data. Thus legal and ethical issues arise, this thesis will only focus on some ethical aspects. While one might assume that the data sharing leads to a benefit for the society, tempering with sharing patient data for a particular reason such as AI training is a threat to patient privacy and confidentiality. There are tensions between who owns the data, as the health care provider has properties interests, while a patient is more concerned about privacy. Additionally, public communities are willing to go great lengths to improve medical care, thus aching for more open access. (a) Assuming there is a minimal risk associated with data sharing, (b) explicit consent

is impractical, and (c) data security in the form of anonymization can lead to altruism. (Jaremko et al., 2019) Most of the data consists of radiology images, which are classified as highly personal and sensitive. A particular image can contain enough information to track a group of persons. Figure 3.1 shows an example reconstruction of an axial fluid-attenuated inversion recovery image. There is always the possibility of a data breach that results in suffering patients as they might experience discrimination, increased costs in insurance, and many more. In the European Union, the General Data Protection Regulation opts patients to perform a general consent to research (Regulation, 2016).

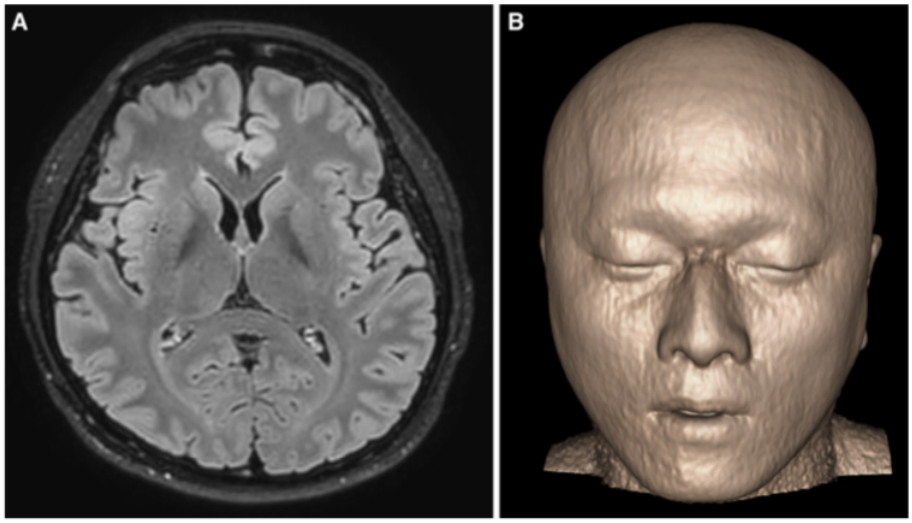


Figure 3.1: (A) Axial fluid-attenuated inversion recovery image. (B) Coronal 3-dimensional reconstruction using skin threshold. Reprinted with permission from SagePub (Jaremko et al., 2019)

The two typical tasks a radiologist performs involve the detection of structural abnormalities and classifying into disease categories. (Lee et al., 2017) Both tasks are well suited for ML in combination with big medical data.

Generally speaking, ML tries to mimic the behavior of neurons. Figure shows an artificial neuron figure 3.2. A neuron consists of the synapse, cell body, axon, and dendrites. This single neuron can cope with various input signals and outputs them based on conditions to other neurons. Interconnected artificial neurons are the foundation of each ANN. Artificial neurons output a signal based on the weighted sum of proof. Compared to biological neurons, the weighted sum of proof are the dendrites. Those computing units are glued and establish an ANN. Learning algorithms teach the artificial neuron on what to base the decision. There are various training methods available such as back-propagation, where input is directly mapped to the desired output. (Haykin, 1994)

Neural networks build the basis of every learning algorithm, and they are composed of neurons. The network uses neuron activation α and parameters $\Theta = W, \beta$. W

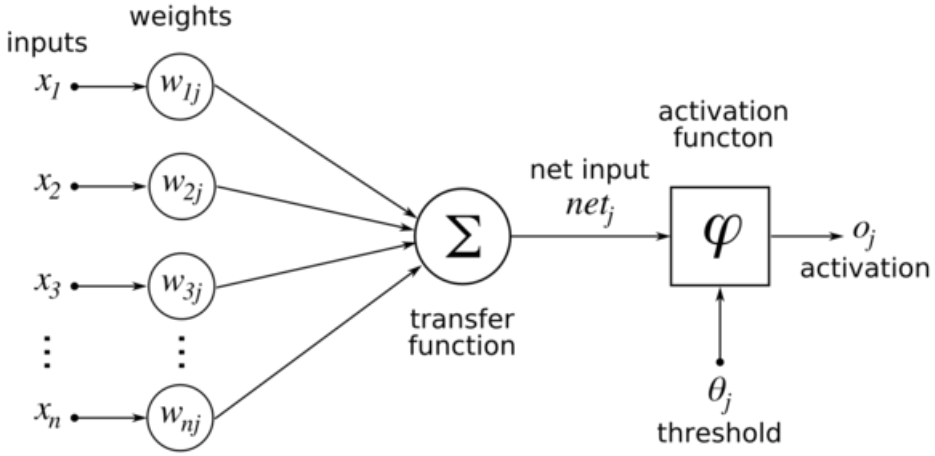


Figure 3.2: Artificial neurons. Reprinted with permission from Springer. (Gopinath et al., 2019)

represents the weights, and β is a set of biases. To activate the neural network, first, an input x must be given to the neurons which then perform an element-wise non-linearity $\sigma(\cdot)$. This is referred to as a transfer function $a = \sigma(W^T x + b)$.

3.1 Technologies

There are two approaches for training the ML model, the unsupervised, and supervised method. Unsupervised learning encompasses unlabeled data and does not consider output data. It focuses primarily on the vector space representing some hidden structure. $D = \{x, y\}\frac{N}{n=1}$ where x are the input features and label pairs y. y can take several forms. In classification tasks, y is a scalar representing a class label. In regression, it is typically a vector that endlessly stores variables. y can even become a multi-dimensional label image in image classification tasks. In contrast to unsupervised learning, supervised learning creates an ML model that deduces from training data. It is optimized for using loss function $L(y, \hat{y})$ based on model parameters Θ. The $\hat{y}$ is obtained by feeding data x to the function $f(x; \Theta)$. Training data has to be prepared with a numerical or nominal vector, that outlines the features of input data and the same output data. Depending on the output data, it is divided into regression and classification. Regression occurs when the output data is continuous; classifications have categorized values. (Litjens et al., 2017)

Naïve Bayesian model is a subtype of supervised learning. The algorithm itself is relatively simple, yet performs excellently in specific classification tasks such as RNA sequence assignment. (Q. Wang, Garrity, Tiedje, & Cole, 2007) The Support vector machine (SVM) in general, is the most popular classification algorithm. The performance

is similar to a naïve Bayesian model, whilst having advantages in regularization and convex optimization. (Byvatov, Fechner, Sadowski, & Schneider, 2003)

Linear and logistic regression systems are used in a variety of regression tasks as they offer pure architecture. A straight line is placed in the space between the data. Fitting that line optimally to the data is a vital task. Logistic regression systems perform classification. (Drucker, Burges, Kaufman, Smola, & Vapnik, 1997) Relying on a naïve Bayesian model such as SVM for financial data or weather forecasting is a challenge. Support vector regression is showing reliable performance for such tasks. (Yu, Chen, & Chang, 2006) (Tay & Cao, 2001)

The following figure 3.3 presents a summarized view of the different ML tasks.

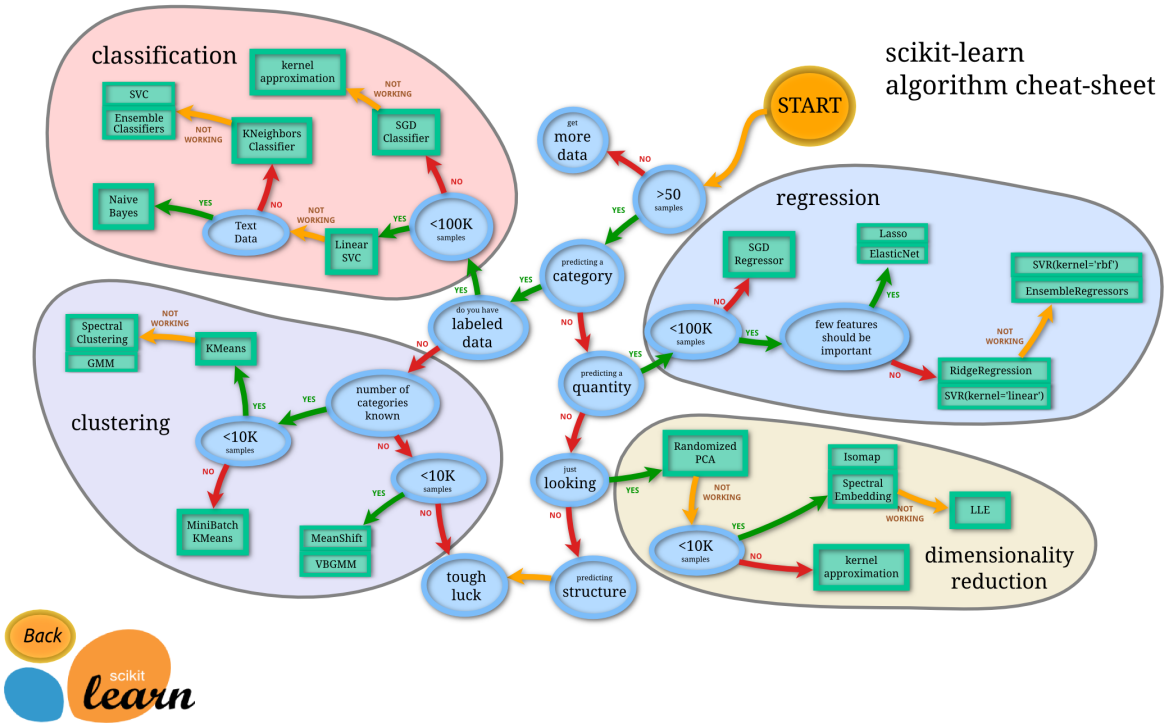


Figure 3.3: Taken from: `https://scikit-learn.org/stable/tutorial/machine_learning_map/`Machine learning categories, classification, regression, dimension reduction, clustering. Support vector regression (SVR), Gaussian mixture model (GMM), principal component analysis (PCA), Support vector regression (SVR), variational Bayesian Gaussian mixture model (VBGMM), locally-linear embedding (LLE), stochastic gradient descent (SGD)

To train an ANN nominal or numerical values must be defined in the input data, also known as a feature. Defining features is an essential task in ML. In the first steps, data scientist and domain experts interpret the data statistically before handcrafting features. Accuracy and performance are in direct union with the quality of the features present. Electing features also depend on the type of ML algorithms utilized;

they might vary depending on whether unsupervised learning or a classification task is planned. (Litjens et al., 2017)

To tackle difficulties presented by ANN such as overfitting, a deep neural network (DNN) was introduced. DNN diverge from ANN by applying multiple hidden layers in order. Thus DNN is costly in operation time, performance, and training. Figure 3.4 shows a typical ANN with one layer. The input layer describes the values going inside the ANN. The output layer outputs a value or class prediction. The layers between input and output are hidden layers. They are hidden because the state of those does not match to input or output data. DNN has more hidden layers see figure **??**, allowing them to create more puzzling decisions. Each hidden layer is targeted with a specific task. (Deng, Hinton, & Kingsbury, 2013) A three hidden layer DNN might be able to discover and classify a tumor. The edges in such complicated networks necessitate optimized weights from explicit training samples. Billions of parameters are randomly initialized and progressively configured by optimization algorithms such as gradient descent to find the optimal minimum. DNN outperforms ANN in tasks such as prediction and recognition, as those involve complicated decision trees. (Kiwiel, 2001) DNN are researches have obtained a way to implement them using unsupervised restricted Boltzmann machine. By training the layers independently in an unsupervised manner, limitations such as overfitting and local minimum optimization where overcome. Instead of handcrafting features, the DNN delivers the features. GPUs are becoming more and more powerful and prominent, which also helped to shorten the computational time. (Hinton, Osindero, & Teh, 2006)

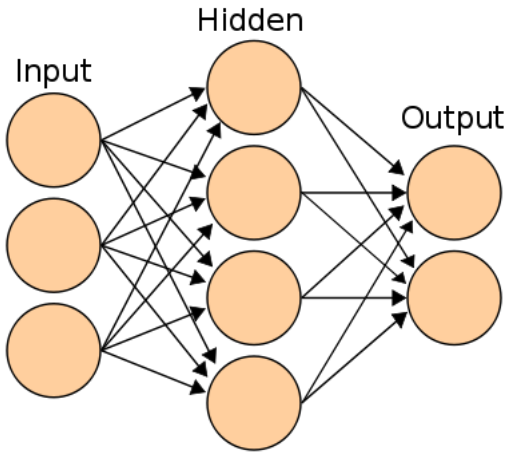


Figure 3.4: ANN architecture showing input, hidden and output layer. Taken from `https://fa.wikipedia.org/wiki/%D9%BE%D8%B1%D9%88%D9%86%D8%AF%D9%87:Artificial_neural_network.svg`on 21.09.2019 at 19:13 UTC+2, CC.

The restricted Boltzmann machines (RBM) and deep belief network (DBN) are based upon the Markov Random Field which establishes an input layer, often called visible layer $x = (x_1, x_2, \ldots, x_N)$ and hidden layer $h = (h_1, h_2, \ldots, h_M)$ which holds the internal feature representation. Input vector x can receive feature description h at any

given time since the nodes are bidirectionally joined. Sampling and generating new data points are routine tasks for RBM. DBN leverage RBM as a layer instead of relying on autoencoders. RBM is trained in an unsupervised manner, usually finished with a linear classifier.

Backpropagation is a term describing a learning process for ML models, which is not complicated, yet simple. On a high-level view, it is simple to set up and works out of the box very well. However, challenging backpropagation with some difficult tasks might lead to a non-working ML model. Fixing this involves tedious tasks and designing a neural network that fits backpropagation. The design is not based on scientific facts; instead, it relies more on choices, and there is no foolproof recipe. (Y. A. LeCun, Bottou, Orr, & Müller, 2012)

3.1.1 Recurrent neural networks

Another form of neural networks is Recurrent neural networks (RNN). RNNs capabilities are limited to an effective training procedure. Common methods are backpropagation and real-time recurrent learning. (Williams & Zipser, 1995)(Werbos, 1988) (Pineda, 1987) The limitation RNNs face is that the magnitude of weights defines the decision. Unlearned data flows back exponentially due to the weights, resulting in two possible ways an RNN can work. Either error blows up quickly, or they vanish quickly. This leads to the issue of learning in RNNs with time delays. Usually, RNNs tend to be unable to learn when the discrete-time is greater than 5 - 10. The discrete-time is defined as a step between input events and output event. (Cummins, 1999) The long short-term memory (LSTM) is a model solving exactly that problem. It was introduced by Hochreiter et al. in 1997 and has since been one of the default algorithms for learning in RNNs. (Hochreiter & Schmidhuber, 1997) One of the main differences from LSTM to the other training methods such as backpropagation and real-time recurrent learning, is a constant flow of errors that allows dealing with time lags going even bigger than 1000 discrete-time. The constant flow of errors is known as constant error carrousels (CECs) which happens in distinct units that are called cells. Gate units deal with opening and closing of the cells. (Hochreiter & Schmidhuber, 1997) With this LSTM, it was possible to solve complex tasks that took a long time to complete. The figure 3.5 explains primary architectural style of a LSTM.

The figure 3.5 shows a standard LSTM network and their way of work. The LSTM consists of a simple unit in the hidden layer that is known as the memory block, which is a summarization of memory cells. Those memory cells are shielded by gates that deal with the input and output to any cell inside the same memory block. Each cell has a defined state inside the memory block, due to the constant error carousel, which deals with a recurrent self-connected linear learning. CEC solve the issue of vanishing errors, by feeding the input to the cell in a loop. This continuously circulating activation is protected from forwarding and backward errors with the help of

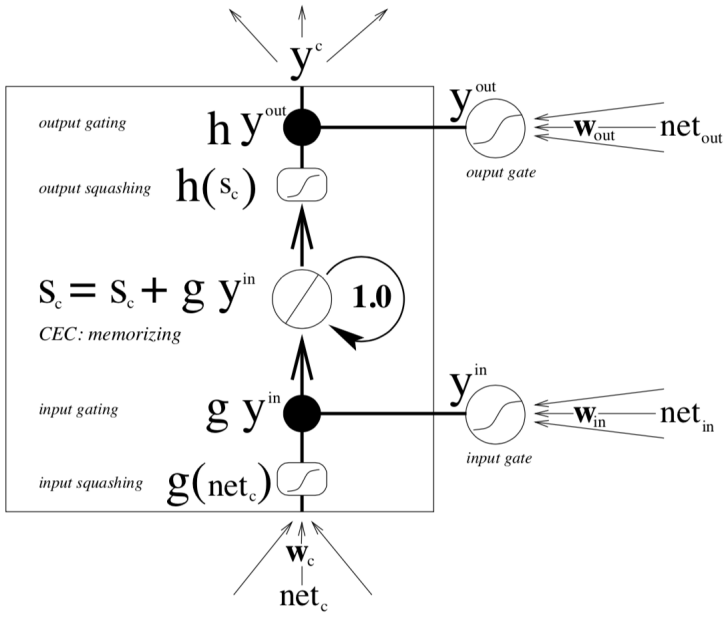


Figure 3.5: A recurrent self learning unit is connected with a weight of 1. Gates such as input and output regulate the flow of data, to define the state of the cell (s_c) g targets the input to be smashed and h targets output to be smashed. Reprinted with permission from Springer Nature (Cummins, 1999)

the gates. The figure 3.5 shows a single memory block with a single cell, where the state of the cell s_c is determined, and changes based on three different sources. net_c describes the input of the cell, net_in and net_out target the input and output gates. As mentioned above, already discrete time steps such as $t = 1, 2, \ldots$ are defined by the following definition. One step in an LSTM forces every unit to be updated as well as the weights to be computed. In and output gates are activated based on the following computation. (Cummins, 1999)

$net_{out_j}(t) = \sum_m w_{out_j m} y^m(t-1); y^{out_j}(t) = f_{out_j}(net_{out_j}(t))\ net_{in_j}(t) = \sum_m w_{in_j m} y^m(t-1); y^{in_j}(t) = f_{in_j}(net_{in_j}(t))$

The notation is the following: j is the index of the memory block. v is the cell inside the memory block j. w_{lm} represents the connection from the cell m to cell l. m is the index from all units in the entire network. f denotes as the logistic sigmoid function which has a range between $[0, 1]$. (Cummins, 1999)

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

The cell itself receives the following input, where c_j^v is the v the cell of the j memory block.

$net_{c_j^v}(t) = \sum_m w_{c_j^v m} y^m(t-1)$

Flattening the received input is done by g, which is a logistic sigmoid function with range $[-2, 2]$.

$g(x) = \frac{4}{1+e^{-x}} - 2$

To calculate the state of the memory cell $s_c(t)$ the squashed input is added at the last possible step $s_c(t-1)$.

$s_{c_j^v}(0) = 0; s_{c_j^v}(t) = s_{c_j^v}(t-1) + y^{in_j}(t) g(net_{c_j^v}(t)) for\ t > 0$

Last but not least, the cell output y^c is calculated by applying the squashing function h to the internal state s_c and using the gate function y^{out}.

$y^{c_j^v}(t) = y^{out_j}(t) h(s_{c_j^v}(t))$

The squashing function h is another sigmoid function that centers it's values around a $[-1, 1]$ range.

$h(x) = \frac{2}{1+e^{-x}} - 1$

The final step assumes a network topology with one input layer, a hidden layer with memory blocks, and a default output layer. The output is defined as k.

$net_k(t) = \sum_m w_{km} y^m(t-1), y^k(t) = f_k(net_k(t))$

With this formula, a single forward pass is done. Accounting for backpropagation, backward passes are more complex and go beyond the scope of this thesis. The backward pass itself can be summarized to a one time pass of any input to the memory cell where it runs through a gate and stays insides the CEC until the gate opens up. This is essentially the reason why LSTM can deal with arbitrary time gaps. (Hochreiter & Schmidhuber, 1997)

However, they do face challenges in particular situations, such as growing s_c linearly during a time series. Assuming an infinity and continuous input stream to the cells, the output squashing function h is saturated quickly, due to the cell states s_c growing in a boundless manner. LSTM fed with a problem that might reset cell states when a new input is received, quickly leads to a derivate vanishing of h which results in blocking error flows. Additionally, the memory cell will shrink down to a normal backpropagation, as the output the cell generates is equal to the output of the gate activation. Thus any of the above-mentioned benefits from an LSTM are lost completely. (Hochreiter & Schmidhuber, 1997) Hochreiter et al. 1997 did not discover this issue in their proposed LSTM they reset all their cell states to zero before starting a new input sequence. (Cummins, 1999) There are some ideas on dealing with this exactly problem. One approach would be a method that calculates the inverse of the

output function h, to learn which cells are not adequately trained and need to be reset. However, this is not successful in actual real-world applications as the network is not automatically resetting itself in any neutral state at any given moment. Felix et al. introduced forget gates that focus on solving exactly that problem. (Cummins, 1999) They exchanged the CEC weight function with a forget gate activation function y^{φ}. φ is calculated the same way that any other gate activation function is. Figure 3.6 shows that the activation function φ replaces the CEC function, thus allowing it to reset the inner cell state s_c at any given moment. In contrary to the function defining a cell above, a y^{φ_j} is added in the beginning.

$$s_{c_j^v}(t) = y^{\varphi_j}(t)\ s_{c_j^v}(t-1) + y^{in_j}(t) g(net_{c_j^v}(t)) for\ t > 0$$

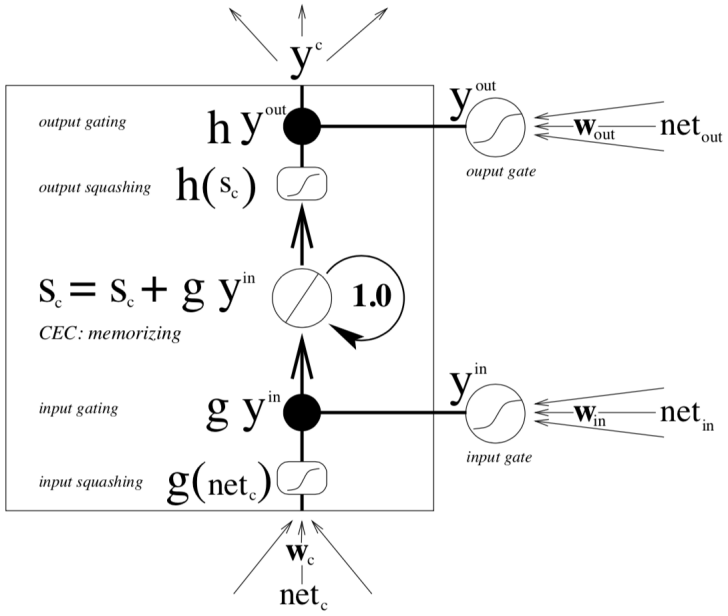


Figure 3.6: A memory block with the forget gate. Reprinted with permission from Springer Nature (Cummins, 1999)

3.1.2 Convolutional Neural Networks

Convolutional Neural Networks (CNN) consist of multiple layers that imitate an animal visual cortex. (Hubel & Wiesel, 1968) With CNN computer vision has perceived a new level and is fastly improving. Image classification is a primary task performed by CNNs, as those can detect the edge-shape-component-object structure. At first, the CNN receives a feature map which consists of a three-dimensional matrix where the first two correspond to the height and width of the image's pixels. The last one contains the color channel, red, green, or blue. After that, CNN performs three steps. CNN build convolution layers which are effective at detecting low-level features such as lines that can be interpreted to a defined shape. (Druzhkov & Kustikova, 2016) Every layer deals with the input layer where a set of K kernels $W = W_1, W_2, \ldots, W_K$ and

biases $B = b_1, b_2, \ldots, b_K$ generate a feature map X_k. Inside each convolution layer the following process is repeated: $X_k^l = \sigma(W_k^{l-1} * X^{l-1} + b_k^{l-1})$. (Litjens et al., 2017)

There are other contestants that expand the functionality of CNNs. To benchmark different CNNs the ImageNet challenge arouse. LeNet introduced by LeCun et al. in 1998 (Y. LeCun, Bottou, Bengio, Haffner, et al., 1998) and AlexNet proposed by Krizhevsky et al. in 2012 are relatively alike (Krizhevsky, Sutskever, & Hinton, 2012). As this architecture is shallow and consisting only of two to five convolutional layers, deep learning architectures were explored. Going a step further, a model-based even 19 layers (OxfordNet) won the ImageNet challenge in 2014. (Simonyan & Zisserman, 2014) The initial complex building blocks were introduced by Szegedy et al. where they proposed a 22 layer network GoogLeNet. (Szegedy et al., 2015) This marks the first occurrence of inception blocks. Instead of relying on a single convolutional mapping, it works with a set of convolutional with different sizes and shapes. By stacking more and more smaller convolutions and kernels, the need for parameterization reduces. The most impressive example is the winner from the ImageNet challenge in 2015, where the neural network was built using ResNet blocks. Those blocks work on a residual manner, and thus the next block only learns the residual and thereby is pre-conditioned. (He, Zhang, Ren, & Sun, 2016)

A layer in any neural network serves a distinct purpose. In CNN, multiple layers can be combined and serve as input layers. They can be merged at any given point. Multi-stream architectures are used for multi-scale image analysis. To detect abnormalities, the context is a primary cue. Feeding larger batches improves the context information. However, this directly impacts the memory requirements of a network. The first application of a multi-stream multi-scale architecture was made by Farabet et al. (Farabet, Couprie, Najman, & LeCun, 2012).

Convolution: Is performed as the first of those three steps and extracts tiles from the feature map. Whilst extracting the CNN applies a filter so that a new feature map or convolved feature is outputted. The output must not resemble the shape and size of the input feature map. Convolutions can be parameterized in two settings:

- Extraction size of tiles (usually 3x3 or 5x5 pixels)
- Depth of the output feature map

As shown in 3.7, the convolution is performed with a 3x3 filter that runs over a 5x5 input feature map. In a 5x5 map are nine possible locations for a 3x3 map. Thus the output feature map is itself a 3x3 map. There are two example steps shown in 3.7 and in 3.8.

In each 3x3 filter pair, an element-wise multiplication of the filter and tile matrix is performed. The sum of this multiplication is the sum of the square in the output

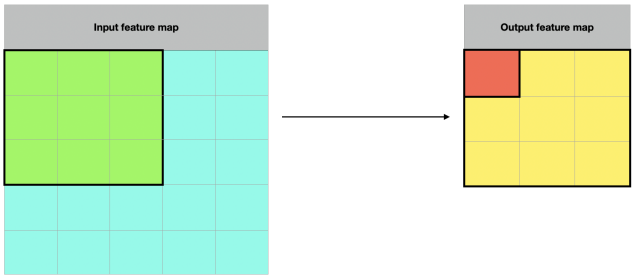


Figure 3.7: Step 1 3x3 convolution of depth 1 performed over a 5x5 input feature map with depth 1.

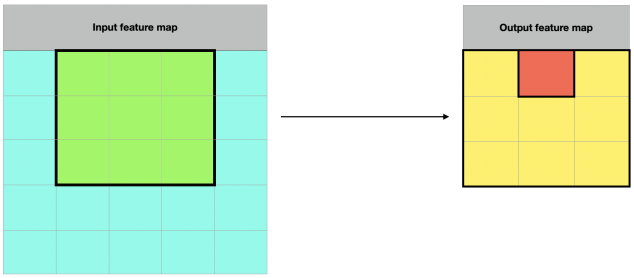


Figure 3.8: Step 2 3x3 convolution of depth 1 performed over a 5x5 input feature map with depth 1.

feature map. The figure 3.9 has the input feature map on the left side and the convolutional filter map on the right side. As explained element-wise multiplication is done, this is explained in figure 3.10.

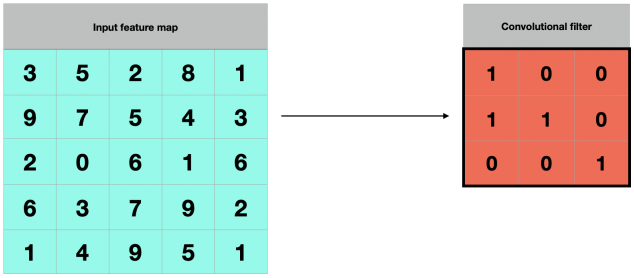


Figure 3.9: Left 5x5 input feature map depth 1. Right a 3x3 convolutional map depth 1.

For each applied filter, CNN can extract more meaningful features (shapes, edges) from the input feature map. The number of filters should be as low as possible as each applied filter does not linearly increase the extracted features. Instead, the tradeoff for adding filters is increased training time. There is no correct answer to how many filters need to be used, and it often depends on the use case.

ReLU: Usually, the next step in CNN involves using activation functions such as 'ReLU' short for Rectified Linear Unit (ReLu). This function affects the nonlinear-

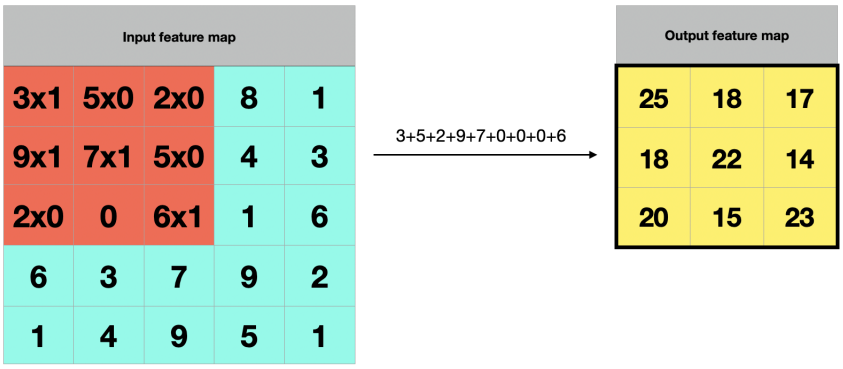


Figure 3.10: Left the 3x3 convolution map is applied to the 5x5 input feature map resulting in an element-wise multiplication, thus outputting the output feature map. In this output feature map all other elements are calculated.

ity in the model. As $F(x) = max(0, x)$ returns 0 for all values where x $\leq$ 0 and returns x for all values where x > 0.

Pooling: CNN has a vast feature map that needs downsampling before it can be processed further. This step is needed to save up on processing time. There are various pooling algorithms, and the most common one is 'max-pooling'. Pooling is ensured to maintain the most crucial feature data.

Fully connected layers: The final classification and decision are happening in fully connected layers. Fully connected is defined as every node from layer one is connected to every node from layer two. In this fully connected layer, an activation function 'softmax' outputs a value between zero and one. Figure 3.11 is an example of a fully connected CNN.

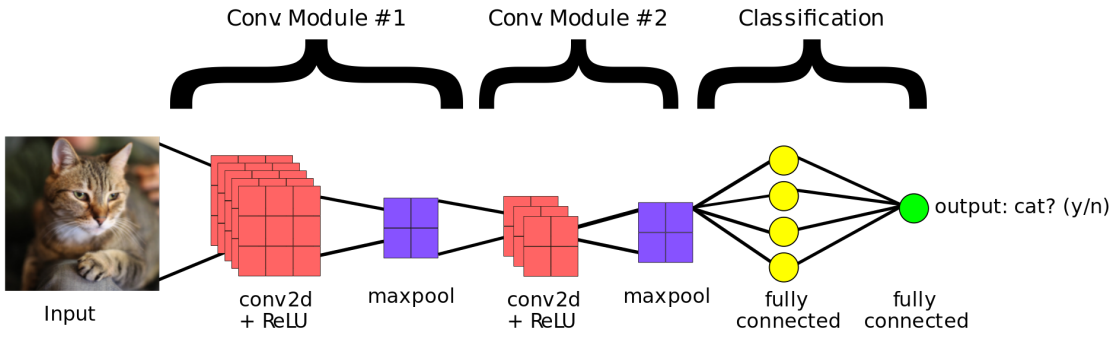


Figure 3.11: CNN with two modules for feature extraction and two fully connected layers. Taken from `https://developers.google.com/machine-learning/practica/image-classification/images/cnn_architecture.svg`on 21.09.2019 at 19:13 UTC+2, CC.

CNN in 3D faces another challenge. Milletari et al. proposed a way to account for 3D images in a 3D convolutional layer. They are allowing CNN to have the context of the location. Using prostate MRIs, which are immensely difficult to analyze/classify, because of the heterogeneous phenotype and artifacts of the MRI, they achieved 82.39% in the PROMISE2012 challenge. They had to overcome artifacts, that arose from field inhomogeneity, which distorts the entire MRI data and measurement of the volume and setting the volume in relation to the anatomical boundaries, which is a complex task as well. (Roehrborn et al., 1999) Despite all challenges, the V-net achieved 82.39% in the PROMISE2012 challenge. (Milletari, Navab, & Ahmadi, 2016) To solve these convolutions are performed for feature extraction and reduction the resolution by applying stride. At a specific point, the resolution is decompressed until the original size is reached. While sequencing two images as a foreground and background image to achieve 3D views, a softmax layer was added. This layer gives a probability for the voxel to belong to the foreground or background. Since prostates can be relatively small. The network can easily be trapped in local minima, that bias the entire network towards the background. Instead of on relying re-weighting of foreground, a new dice loss layer was introduced. (Milletari et al., 2016)

Softmax takes transformations such as $f(x;\Theta) = \sigma(W^L\sigma(W^{L-1}\dots\sigma(W^0x+b^0)+b^{L-1}+b))$. W^n is a matrix of rows w_k combined with the activation k. L represents the final Layer, while n indicates the current layer. Softmax maps the final layer of activations to a distribution over classes $P(y|x;\Theta)$. w_i^L combines the weight L and the class i

$$P(y|x;\Theta) = softmax(x;\Theta) = \frac{e(w_i^L)^T x+b_i^L}{\sum_{k=1}^{K} e(w_k^L)^T x+b_k^L}$$

3.2 Biomedical image applications

In the last couple of years, all forms of machine learning were introduced to radiologic images. First, DNN dealt with basic image segmentation and registration. CNN is now commonly used in medical images, as they deal with the obstacles of image segmentation by default. Various examples demonstrate the application of CNNs for segmenting structures inside the human body. (Middleton & Damper, 2004) Among them are Lungs, tumors (Pereira, Pinto, Alves, & Silva, 2016), brain structure (Moeskops et al., 2016), cells and membranes (Ciresan, Giusti, Gambardella, & Schmidhuber, 2012), bone tissue (Cernazanu-Glavan & Holban, 2013), tibial cartilage (Prasoon et al., 2013), cell mitosis (Cireşan, Giusti, Gambardella, & Schmidhuber, 2013) and many more successful of 2D image segmentation. Coping with volumetric data from 3D images such as MRI or CT resulted in a large number of parameters. Feeding the network with different streams originating from slices of the Volume of Interest, circumvented the need for parameters. (Prasoon et al., 2013) The CNN takes the 2D images as input and applies spatial consistency or other post-processing computations. Those inflict a huge performance impact, which makes this approach

unfeasible when big data is used. To overcome this issue, Architectures based on CNN are introduced, that use the entire image, instead of segmenting it. The most noticeable CNN is the fully CNN (fCNN) developed by Kang and Wang (Kang & Wang, 2014). They segmented crowds in surveillance videos. Whilst applying convolutional and pooling layers in a looping manner, the output resolution shrunk. (Kang & Wang, 2014) Brosch et al. (Brosch et al., 2015) suggested a way to leverage an fCNN to keep the identical resolution. By utilizing convolutional and deconvolutional layers, to reconstruct convolutional fragments to the original resolution (Zeiler, Taylor, Fergus, et al., 2011) With such high-quality images, an fCNN is proficient of detecting lesions. (Mottaghi, Xiang, & Savarese, 2015) The U-net architecture is based on the fCNN by introducing an up-sampling part with up-convolutions to increase the image size, expansive paths, and coined contractive. (Ronneberger, Fischer, & Brox, 2015) By leveraging skip connections and directly connecting opposing layers, performance is boosted. The suggested approach from (Ganea, Burdescu, & Brezovan, 2011) determine some major regions. It leverages a hexagonal representation of the pixels and builds a hyper graph structure in which the hierarchical structure is processed. (Ganea et al., 2011)

There are a few studies showing real-time image segmentation for diagnosis and surgery guidance purposes. (Wohlhart & Lepetit, 2015)(Dollár, Welinder, & Perona, 2010)(Zach, Penate-Sanchez, & Pham, 2015) Currently, none of them meet the high accuracy standards. However, there are some intriguing approaches to a CNN that was suggested by Shun Miao et al. as Pose Estimation via Hierarchical Learning (PEHL). (Miao, Wang, & Liao, 2016) This requires multiple layers in which the first layer is reliable for accurately classifying the rotational parameter, then normalizing it and feeding it into other layers that deal with scale (is it zoomed in or zoomed out).

Several examples of annotation approaches using ML. Figure 3.12 shows an example labeling over a chest X-Ray. The first labeling relied heavily on datasets, where the target object is annotated. (Kulkarni et al., 2013) Analyzing the cortex in brain MRIs is required to perform tumour segmentation (Subbanna, Precup, & Arbel, 2014). A combination of CNNs and RNNs lead to automatically annotated chest radiographs. (Karpathy & Fei-Fei, 2015) Adding captions with gradient descent models oppose a difficulty. Therefore CNNs and RNNs are preferred. (Bengio, Simard, Frasconi, et al., 1994) It is possible to segment, locate, and identify affected organs. The feature the CNN outputs is a context that the RNN then interprets and labels as a disease. (Shin et al., 2016)

Segmentation is a task that allows a quantitative analysis of parameters like shape and volume. Usually, the first step in a pipeline involves segmentation, before object detection and classification can be applied. Segmenting a biomedical image means splitting it into voxels that represent a distinct region. As mentioned above the U-net architecture by (Ronneberger et al., 2015) added two innovative features. Combining the up-sampling and downsampling layer has been proposed before. However, Ronneberger et al. took it a step further and added skip connections that allowed

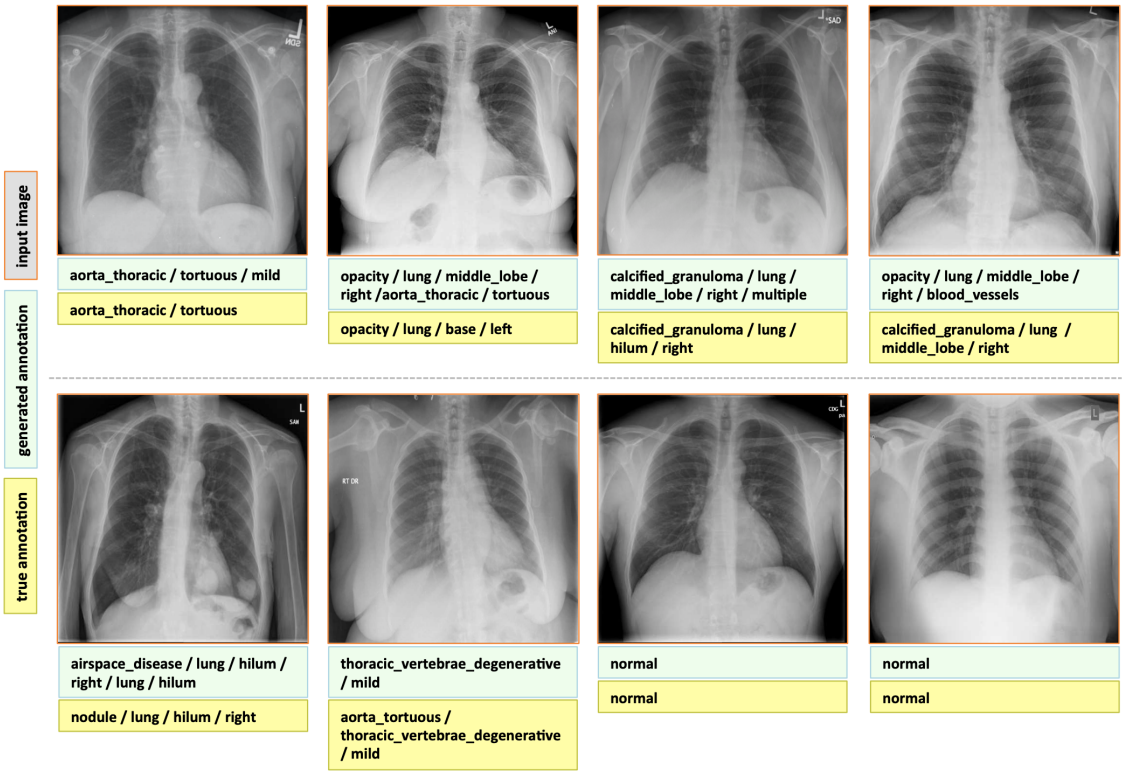


Figure 3.12: Examples of annotations (light green box) versus the true annotations in the yellow box. Reprinted with permission from ©(Shin et al., 2016) IEEE

to concatenate features to adjust the paths. This means that it is possible to feed the entire image in one forward pass into the U-net. Instead of relying on patch-based networks, that means that the CNN has full knowledge of the context. (Ronneberger et al., 2015) There are a few papers that deal with processing 3D images such as MRI and CT scans as 3D images. One most noticeable is a 3D version of the U-net architecture that is called V-net is proficient in segmenting a 3D image with 3D convolutional layers. (Milletari et al., 2016)

Computer-aided detection (CAD) has been in use since 2000. Detection and false-positive reduction are the primary two tasks accomplished by CAD. ML often performs the latter task. Unfortunately, current CAD systems perform not good enough, and the clinical usage is dropping as a consequence. (Fenton et al., 2007) There are some deep learning-based CAD systems used for diagnosing and staging of breast cancer (D. Wang, Khosla, Gargeya, Irshad, & Beck, 2016) and lung cancer (Kumar, Wong, & Clausi, 2015). Around 13,5% of diagnosed cancer in 2012 was breast cancer. Identifying high-risk patients and treating them early on diminishes the risks of mortality. In mammograms, the mammographic density is visible, which is a common indicator for breast cancer. There are case studies trying to incorporate object or lesion detection with multiple biomedical imaging technologies such as CT and

Positron Emission Tomography (PET) at the same time. Teramoto et al. suggested a multi-stream CNN. (Teramoto, Fujita, Yamamuro, & Tamaki, 2016)

Kallenberg et al. introduce a sparse convolutional autoencoder that learns a deep hierarchy from unlabeled data. This additionally extends CNNs and feed the found features and classifiers into a CNN for proper detection. The architecture is shown in figure 3.13 (Kallenberg et al., 2016) This is a new approach as unsupervised pre-training with sparse autoencoders enables to create features automatically. Xu et al. compared unsupervised feature learning approaches, supervised and handcrafted features, finding that unsupervised methods perform far superior. This procedure is inevitable to handle the lack of high-quality annotations in medical image data. (Xu et al., 2014)

Autoencoder and stacked autoencoders used by Kallenberg et al. (Kallenberg et al., 2016) in unsupervised learning are simple networks that reconstruct the input x to the output x' with one hidden layer h. They do use a weight matrix $W_{x,h}$ and bias $b_{x,h}$. By applying a non-linear function in the hidden activation the reconstruction is successful: $h = \sigma(W_{x,h}x + b_{x,h})$. The hidden layer h must have fewer dimensions than x. To represent a subspace with the most dominant latent structure from the input, which is necessary to deny the model to learn a trivial solution easily. (Vincent, Larochelle, Lajoie, Bengio, & Manzagol, 2010) Autoencoders are stacked at the final layer of neural networks.

Lung cancer screening within DL has the potential to predict it as well as classify lung nodules. (J. Chen et al., 2015) The challenge is to allow each CNN to get local information and contextual information such as localization. In a standard CNN, this is not possible. Shen et al. dealt with three different CNN interconnected that take the patches of nodules at different sizes as input. (Shen, Zhou, Yang, Yang, & Tian, 2015) A poor prognosis is given when lung cancer is not detected early on. On average, the survival rate for 5-year is less than 20%. (Hua, Hsu, Hidayati, Cheng, & Chen, 2015) Thus lung lesion detection is utterly important. Small lung nodules on CT images are controversial as there is no guideline on interpreting the tumor characteristics. Current CAD systems do not perform well enough, as they depend heavily on optimizing and are tedious to develop. Hua et al. developed a deep belief network in combination with a CNN that is able of finding and classifying pulmonary nodules. (Hua et al., 2015)Even highly complicated biomedical imaging techniques such as single-photon emission computed tomography and positron emission tomography have been incorporated into DL approaches to diagnose Alzheimer's disease. (Suk & Shen, 2013)(Liu et al., 2014)(Suk, Lee, Shen, Initiative, et al., 2014)

Localization of anatomical objects such as organs, landmarks, regions, diseases, is a prerequisite task. They are often needed to be done in pre-processing before a diagnosis can be created. As biomedical images need to be interpreted with the 3D volume information, multiple suggestions have been introduced that cope with a 3D to 2D transformation. Applying 3D space on 2D orthogonal planes, Yang et al. were

able to identify landmarks on distal femur surface with three independent sets of 2D MRI slices within a single CNN and defining the 3D intersection as the landmark, where the CNN had the highest classification output. (Yang et al., 2015) A landmark is defined as a distinctive feature that allows for comparison. On a human skull that would be the jar or the tip of the nose as an example. One other successful model by de Vos et al. who added regions of interest to organs. By adding a 3D bounding box to the 2D parsing of the 3D CT volume, the aortic arch and descending aorta were identified with a heart CT. (de Vos, Wolterink, de Jong, Viergever, & Išgum, 2016)

Still, as object detection is an important task, it proves as a challenging one. Sliding pixel windows or voxels introduce redundant learning. This is solved by leveraging the capabilities of an fCNN. However, the majority of samples given involve injuries that are easy to discriminate, meaning that the neural network will have trouble when tested with complicated tasks. There is a suggestion to select falsely classified data and sent it to network more often. Van Grinsven et al. used this technique to train a deep learning method to detect retinas properly. (Van Grinsven, van Ginneken, Hoyng, Theelen, & Sánchez, 2016)

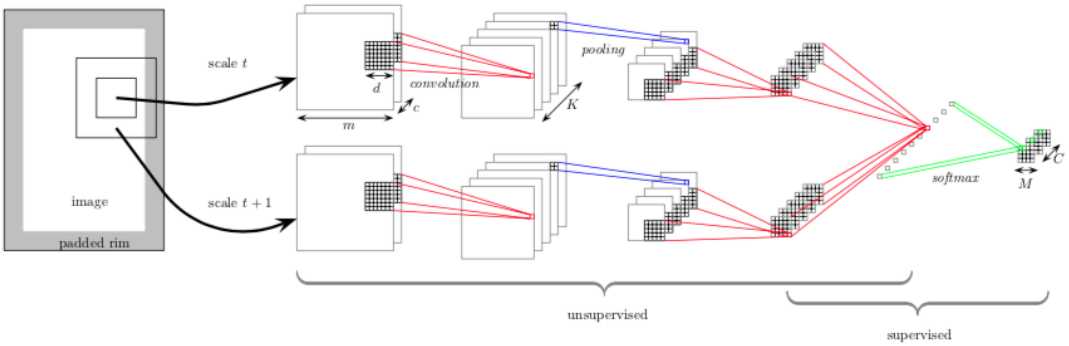


Figure 3.13: The CNN consists of convolutional, pooling, and softmax layers. The image is extracted in patches. The correct feature maps are automatically selected depending on the pixels in the patch. The sparse convolutional autoencoder is the unsupervised CNN. The last layer, supervised CNN, uses a fine-tuned softmax regression with pre-trained weights and bias terms. Reprinted with permission from ©(Kallenberg et al., 2016)
IEEE

3.2.1 Radiographs

As mentioned above already, there are some applications of ML, DL, AI in the field of radiographs present. This section focuses on the existing applications that are worth mentioning. Most papers target adult radiographs. However, there is one neural network that specifically targets pediatrics by Larson et al. (Moore, Slonimsky, Long, Sze, & Iyer, 2019) Larson et al. proposes a CNN that is fitted of estimating bone ages

as good as an expert radiologist and any other automated model. They were able to close the gap between the estimation of their CNN to 0 years, the κ was still around 0.8 years. Bone age is examined by using left-hand X-rays, due to simplicity, minimal radiation exposure, and multiple ossification centers. A radiologist takes the unknown age image and compares it to a reference atlas. Inter and intra-observer differences span from 0.07 to 1.25 years (Berst et al., 2001). BoNet: features multiple layers, with one convolution max-pooling layer, three hidden layers, again a max-pooling layer that extracts additional feature maps in smaller slices. A deformation layer that deals with the localization of the extracted feature maps. After that, a convolutional feature map is again resized, and max pooling applied. In the last layer, around 2048 neurons are fully interconnected. (Larson et al., 2017)

MURA, the large dataset of musculoskeletal radiographs, consists of 40.561 images from other 14.863 studies. Each study was manually labeled by radiologists as abnormal or normal, sinceo ne of the critical radiological tasks is to classify an image. Around 1.7 billion people are diseased with musculoskeletal conditions (Andersson & Watkins-Castillo, 2014). Any long-term conditions such as pain, disability, deformity are directly connected to such conditions (Woolf & Pfleger, 2003). The dataset consists of 9.045 normal and 5.818 abnormal studies of the upper extremity, making it one of the largest publicly available datasets. Rajpurkar et al. created a test site contain 207 studies that were labeled by three board-certified radiologists. Their goal was to match the ML performance. Classifying an image as abnormal or normal is a binary classification $y \ \epsilon \ 0, 1$ task. By creating a 169-layer CNN and using an arithmetic mean of the probabilities from the CNN, the designated performance was reached. Additionally, their last layer is a single output that directly feeds into a sigmoid non-linearity function. Each layer is connected to each other layer in an only feed-forward manner, also known as Dense Convolutional Network Architecture introduced by Huang et al. (Huang, Liu, Van Der Maaten, & Weinberger, 2017). Furthermore, they optimized the weighted binary cross-entropy loss for each image. (Rajpurkar, Irvin, Bagul, et al., 2017)

$$L(X, y) = -w_{T,1} * y \log p(Y = 1|X) - w_{T,0} * (1 - y) \log p(Y = 0|X)$$

X is the image in the study type T from the training set. y is the label, $p(Y = i|X)$ is defined by the network and represents the probability. $i, w_{T,1} = |N_T|/(|A_T| + |N_T|)$ and $w_{T,0} = |A_T|/(|A_T| + |N_T|)$ is defined by $|A_T|$ and $|N_T|$ being the abnormal and normal image sets. (Rajpurkar, Irvin, Bagul, et al., 2017)

They were able to outperform the board-certified radiologist on specific tasks. On all seven tasks, elbow, finger, forearm, hand, humerus, shoulder, and wrist, the model was only best performing in one category wrist. Diagnosing complex parts of the body, such as the wrist, the model received a κ 0.931, while failing on simple tasks such as the forearm, where it only received a κ 0.737, with every radiologist being above κ 0.796. Additionally, it underperformed in 4 categories elbow, forearm, humerus, and shoulder. In hand and finger, it was better than one radiologist but not

better than all three. (Rajpurkar, Irvin, Bagul, et al., 2017) Publishing their dataset to the public allowed other contestants to perform deep learning analysis. Banga et al. tried to improve the performance and consistency of the CNN proposed by Rajpurkar et al. Banga et al. used an ensemble model within a single trained model-based. Their basic idea is to leverage existing CNN applications such as ChexNet to detect pneumonia in Chest X-Rays (Rajpurkar, Irvin, Zhu, et al., 2017). The ChexNet network developed by Rajpurkar et al. was trained on over 100.000 frontal Chest X-Ray and detected pneumonia on an expert radiologist level. The proposed ensemble model performed better than the MURA model. In the finger category, where it exceeded by a κ 0.653, while the best radiologist had κ 0.410. However, it was not capable of surpassing all radiologists. Ensemble learning shows that this is a possible way to stitch specific neural networks together. (Banga & Waiganjo, 2019)

A paper by Cernazanu-Glavan et al. focuses heavily on segmentation of bone structure in X-Rays. (Cernazanu-Glavan & Holban, 2013) Until now, segmentation tasks needed a priori human intervention. The prerequisite to any ML is segmentation, as this is the only automatic method for successfully extracting information from any image. In X-Rays bone tissue is segmentable as this is differentiated from the rest. Until now, segmentation tasks needed a priori human intervention. By performing a pixel segmentation method, the pixel is labeled as either bone tissue or none bone tissue. In their sense, a pixel is a 128x128 pixel slice of an image. They feed their image in a normal forward pass to the CNN which then applies a 128x128 convolutional map. The starting X-Rays have a resolution of 2492x1984, which resulted in approx. 10.000 128x128 pairs. The architecture of CNN is shown in figure 3.14. The pixel classifier in the segmentation process is an approach that allowed CNN to get exact delimitation of a specific bone, which results in a completely new possible way to leverage CNN in X-Ray images. This is relevant in regards to Chest X-Rays with lung nodules. When CNN proposed by Kumar et al. (Kumar et al., 2015) would only need to dissect the parts of the picture without any unrelated bone tissue, the performance might increase. (Cernazanu-Glavan & Holban, 2013)

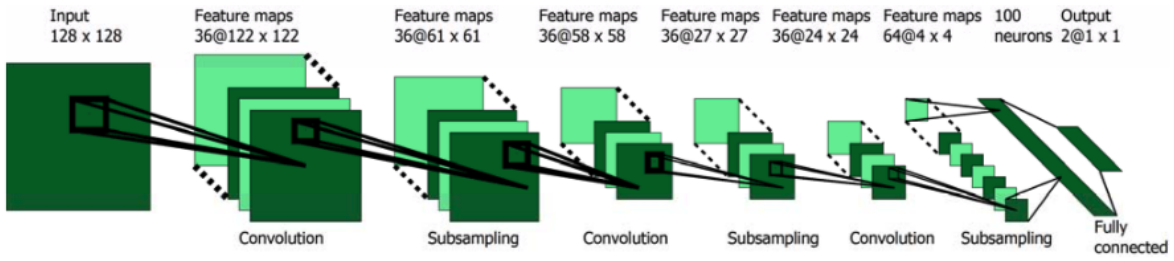


Figure 3.14: CNN architecture proposed by Cernazanu-Glavan. Reprinted with permission from (Cernazanu-Glavan & Holban, 2013)

Additionally, leveraging the segmentation of bone structure into possible bone tissue or not, might boost the performance of networks such as ChexNet (Huynh, Nguyen, & Tran, 2018). Furthermore, Gordienko et al. introduced a bone suppressing CNN

that helps radiologists see uncluttered Chest X-Rays, showing such a suppressed bone Chest X-Ray in figure 3.15 (b). (Gordienko et al., 2018)

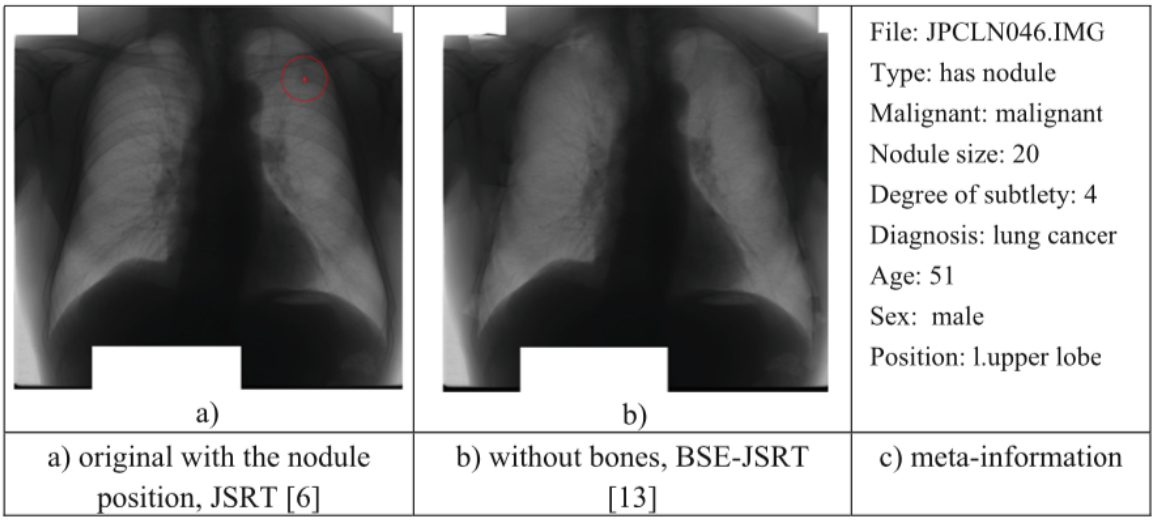


Figure 3.15: (a) unsuppressed Chest X-Ray (b) suppressed bone Chest X-Ray. Reprinted with permission from Springer (Gordienko et al., 2018)

The Medical University of Graz recently published a paper where they were able to build a neural network that reached 86% of classification accuracy. Hržić et al. (Hržić, Štajduhar, Tschauner, Sorantin, & Lerga, 2019) had the novel idea to segment the bone tissue away from the bones and classify the segmented part. As shown above, many ML approaches involve a large quantity of data and correct labeling, which is time-consuming and requires medical experts. Furthermore, most ML models target specific bones or sections of the body to develop enough accuracy. The paper proposes a hybrid X-Ray image segmentation and bone fracture solution. It involves 4 steps, (1) alignment of the tilted X-Ray to a correct direction, (2) using an entropy-based correction method to extract the bone contour, additionally de-noising the entire X-Ray, (3) leveraging graph theory to detect fractures based on the bone contour, (4) localization and classification of the detected fractures. This method can deal with occult fractures and helps radiologists to deal with easily overlooked fractures. Correct alignment is required for an ML model to function correctly. The alignment of pictures is not an issue for a human as it is tilted to the correct position. Radius and Ulna fractures are often misaligned due to pain and impaired mobility of patients. The novel approach used in this paper involves enlarging the border of the X-Ray image, performing a black and white image transformation, utilizing the PCA method to calculate a vector describing the orientation of the image, lastly, adjusting the image orientation to the correct vertical axis. Pediatric X-Rays differ from mature in the post-processing that the extremities must not be cropped beyond the borders of the physical radiation-beam collimation (Bomer, Wiersma-Deijl, & Holscher, 2013). The local entropy-based tissue removal is one of the fundamental approaches used in this novel approach. Bone segmentation often struggles with deciding whether it is bone or tissue. The Shannon entropy, also known as information entropy, is used on various other prob-

lems and has shown an excellent performance overall (Bandyopadhyay, Chanda, & Bhattacharya, 2011). The article introduces a slightly altered Shannon entropy, which added a local short-term measurement. The idea originated from the sliding window, similar to the functions of a CNN. The sliding window strides with 1 pixel, and the entropy is calculated for the middle pixel of a $n\ x\ n$ dimension. The entropy can be classified into two different intensities, namely low and high. The intensity is low when the sliding window catches no bone, whilst the intensity is high when the sliding window captures bone due to the surrounding tissue, as the pixel intensity is diverging. Taking it a step further, the standard deviation of multiple applied local entropy algorithms is calculated, which results in an image like figure 3.16. Detecting the Ulna and Radius bone is done by extracting the region of interest with the help of the boundaries. The bottom boundary is quick to discover as it is the bottom of the X-Ray. The top boundary is more difficult to detect as it involves cropping the image beneath the wrist. The entropy helps here again with the showing of considerable peaks in the line where the bones end — applying local maxima to the Ulna bone results from incorrect localization of the end of both bones. Building a bone graph differs not from other graph solutions in a significant way. It still resides on a root node, which is represented by the root pixel and searches for the next approximate pixels. Selecting the correct neighbor pixels allows building a graph with parent-child relationships. Finding the last possible node, which, by definition, cannot have any further children, is done by calculating the difference in paths while only choosing those whose lengths are higher than the image height. After successfully applying the graphing algorithm to the pictures, it looks like this figure 3.18. Sometimes, X-Rays contains blurred bones; thus, the graphing algorithm will create multiple lines. One must know that blurred bones create multiple contours that are proximate to each other. To tackle this particular challenge, a merging algorithm that searches for contouring a vicinity of the initial ones, in the paper, this was tuned to a window of 15 pixels, then merging them on a per-row basis. The last part involves the detection of a fracture. By applying the above-mentioned methods to pictures without a fracture, the ideal bone graph is created. An X-Ray that contains a fracture will advert to this ideal graph path, and localization of the fracture is possible. This proposed algorithm was capable of classifying and marking the position of occult fractures with an accuracy of 91,16%. (Hržić et al., 2019)

3.3 Algorithms

3.3.1 Pooling

Is a particular algorithm primarily used in CNNs. CNNs produce convolutional layers which summarize features in an input image. The output feature map is sensitive to the location of the found feature. As images are not identically pixel by pixel, there

Figure 3.16: Shannon local entropy filter applied to a X-Ray. Published under CC By license (Hržić et al., 2019)

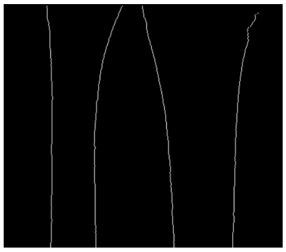
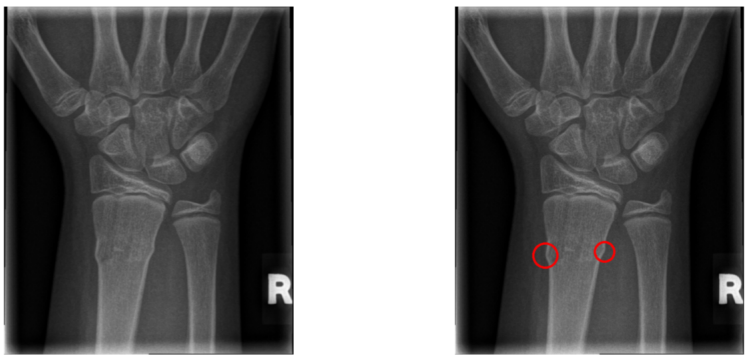


Figure 3.17: Graphing algorithm shows the bone contours(white). Published under CC By license (Hržić et al., 2019)



(**a**) Input raw image. (**b**) Proposed fracture areas..

Figure 3.18: The proposed algorithm shown in (b) marking the proposed fracture with red circles. (a) is the input image with a fracture. Published under CC By license (Hržić et al., 2019)

is a need to make the feature map more robust against changes in the input image, this is called 'local translation invariance'. The task of a CNN is to identify an object as an object, and it is trained by using translation invariance, rotation/viewpoint invariance, size invariance, and illumination invariance. Translation does account for size and shape of the detectable object as it shifts it pixel by pixel to another location. When making the object smaller or bigger with size invariance, the proportion is kept. Downsampling is a task derived from signal processing which reduces the resolution whilst keeping the most crucial information and dropping details that seem irrelevant. There are multiple ways to downsample in CNNs, such as padding and striding an image; still, the most robust solution is pooling. (Cherian, Fernando, Harandi, & Gould, 2017)

Pooled feature maps consist of a 2x2 matrix usually. Thus the pooling process always produces a smaller feature map by the factor of two. A 'ReLU' feature map of 6x6 (36 pixels in total) will result in a 3x3 (9 pixels) pooled feature map. (Cherian, Fernando, et al., 2017)

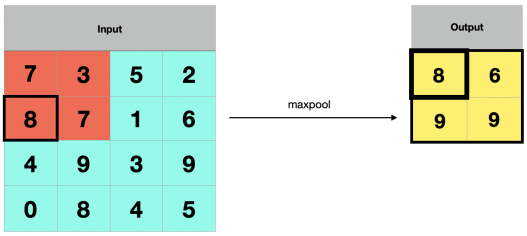


Figure 3.19: Left a 4x4 matrix is filtered by a 2x2 pooling filter with max. Right shows the output from the max pool algorithm applied to all possible fields.

Maximum pooling: This algorithm selects the highest number in the 2x2 matrix. As shown in figure 3.19 applying a 2x2 in the first and second row and column will result in 8. As this is the highest number in this 2x2 matrix. This algorithm seems to perform better in classification tasks than average pooling, as it only collects the most present feature. (Ketkar et al., 2017)

Average pooling: Opposed to maximum pooling, average pooling calculates the average from the 2x2 matrix. In figure 3.19 the average of the 2x2 matrix is $(7+3+8+7)/4 = 5$. (Cherian, Koniusz, & Gould, 2017)

Global pooling: Average pooling and maximum pooling both downsample the input feature map with a 2x2 or larger matrix. Global pooling downsamples everything at once. As shown in figure 3.19 the input feature map is 4x4. Thus the pooling map is also 4x4, and the highest feature is saved. This aggressively shows the presence of a feature. It can be leveraged to produce a model without building fully connected layers. In combination with other methods like region matching, it achieves excellent results. (Tolias, Sicre, & Jégou, 2015)

3.4 Toolkits and libraries

There are a lot of toolkits and libraries available to deal with ML tasks. Some libraries are more focused on building the network, while others focus on the models. The following list of toolkits contains just the most popular and widely spread frameworks.

3.4.1 Toolkits

Theano: Instead of relying on basic python implementation, it sits above numpy and takes advantage of the efficiency that numpy provides. (Van Der Walt, Colbert, & Varoquaux, 2011) Theano is more focused on providing a proper toolkit to build networks, instead of building complete solutions. (Bastien et al., 2012) Thus allowing a high degree of flexibility for modifying and tuning architecture in research phases.

Keras: Theano and TensorFlow can be leveraged as a backend to Keras. Keras is developed by Google, and TensorFlow announced that it would be using Keras as their high-level language.

Caffe: Caffe is one of the biggest and oldest ML toolkit available. It is actively developed by Berkeley Vision and Learning Center and uses JSON - like syntax to describe neural networks.

DLTK: The Deep Learning Toolkit for medical image analysis is built on TensorFlow. DLTK is focused on providing abstraction layers, on dealing with the different image file types, as well as the labeling of such. As many DL libraries expose low-level features, they lack functions like sequencing MRI images. Dealing with biomedical images such as computed tomography can vastly introduce performance challenges. Storing 10.000 of computer tomography images with a 512x512x256 dimension (height, width, color depth) in float32 are around 2,6 terabyte of data. Additionally, special treatment is needed for particular kinds of images, such as de-noising, spatial normalization, bringing down the intensity, bias-field correction. Usually, images are stored in the DICOM standard, which saves volume information in a 2D image. However, 3D,4D,5D images need additional information. Brain MRI is stored in NifTI. DTLK deals with various image types and is capable of converting them on the fly. It outperformed other CNNs in the imageNet challenge with achieving an 81.5 coefficient on the average dice test. At the time given, the accuracy of the winner was 79. (Pawlowski et al., 2017)

PyTorch: PyTorch is widely used in the scientific community as it allows for distributed training. PyTorch ML models are written in TorchScript which allows to write in eager mode and move to production quickly.

3.4.2 TensorFlow

TensorFlow is actively developed and maintained by Google. It supports multiple CPU, and GPUs is capable of distributing the load. As TensorFlow is built upon the idea of Tensor, a short mathematical deep dive is needed.

Tensor A Tensor is an algebraic object that defines vector space. It can represent scalar products which are basically just numbers, as well as multilinear vectors. Alternatively, more straightforward, it is a generalization of vectors and matrices into higher dimensions. Tensors have ranks, and those define how a function is mapped. In table 3.1 the different ranks a tensor can have are defined.

Rank	Math entity
0	Scalar
1	Vector
2	Matrix
3	3 - Tensor (cube of numbers)
n	n - Tensor

Table 3.1: Tensor ranks

Understanding Tensor as a programmer is more easily explained with using example code from TensorFlow itself.

```
mammal = tf.Variable("Elephant", tf.string)
```

Listing 3.1: Zero rank Tensor

As we can see, this Tensor contains a scalar which is Elephant, and this string is treated as a single object and not as an array of characters. In TensorFlow Tensors always have a defined datatype, thus 'tf.string' must be appended.

```
mammals = tf.Variable(["Elephant","Delphin"], tf.string)
```

Listing 3.2: One rank Tensor

In this example, a rank one Tensor is produced as it takes two input arguments, more precise an Array of string objects.

```
animals = tf.Variable([["Elephant","Delphin"],["Goldfish","Dog"]], tf.↩
    string)
```

Listing 3.3: Two rank Tensor

A second rank Tensor consists of at least one column and one row. As in table 3.1 describe, a second rank tensor maps a matrix.

```
animals = tf.Variable([["Elephant","Delphin"],["Goldfish","Dog"],["cat"]], tf.string)
```

Listing 3.4: Three rank Tensor

A third rank Tensor is mapped to a cube of numbers or an n-dimensional array.

```
image = tf.zeros([10, 299, 299, 3]) # batch x height x width x color
```

Listing 3.5: Four rank Tensor

An even higher ranked Tensor is used in image processing, to represent the batch, height, width, and color channel.

3.4.3 PyTorch

PyTorch is programmed in Python and available as LibTorch, which is a C++ re-implementation and as PyTorch as a Python Library with Nvidia CUDA support. The main difference between PyTorch and Tensorflow / Keras is that PyTorch does not use a computational graph representing mathematical expressions. Tensorflow is more like a define-compile-run, while PyTorch is a dynamic define-by-run framework. It does not have a compilation step, which allows any user to directly execute a mathematical expression and calculate the gradient of such expression. PyTorch is widely used in the scientific community as it allows us to easily manipulate the underlying deep learning architecture as it is more intuitive than Tensorflow. Additionally, all existing debugging tools are integrated into PyTorch, and debugging is more easy with the dynamic nature. PyTorch lacks the optimization part, as this requires a define-compile-run paradigm. Similar to Tensorflow, PyTorch uses tensors for their definitions. Furthermore, PyTorch is integrated well with existing Python implementations such as NumPy, Pandas, and many more, making it more easy to port the required code to PyTorch. (Ketkar et al., 2017)

3.4.4 Azure Cognitive Services

The Microsoft Azure platform is gaining more and more popularity. Current studies suggest that Microsoft Azure is the fastest growing cloud platform. Whilst Amazon still holds the first position at around 33%. Microsoft has grown from 8% in 2015, to 16% in 2018. Alibaba, Google and IBM are the third big players in this market and fluctuate between 5% and 8%. (Dutta & Dutta, 2019)(Khan, Dewangan, Meena, & Birthare, 2020)

Microsoft has deployed various services that can be used out of the box. One of the main components are the Azure Cognitive Services. This service allows anyone to leverage already trained machine learning and deep learning algorithm, by simply submitting their image that needs to be analyzed via an API call. Azure Cognitive services are described as a set of RESTful services that can recognize, understand, and interpret the content of various inputs such like text, images, live videos and much more. The Microsoft Cognitive service includes a separate vision category, which is a lowno code version of deep neural networks. It provides the user with a simple web browser interface, where they can train and validate their model, as seen in 3.20. (Del Sole, 2018)

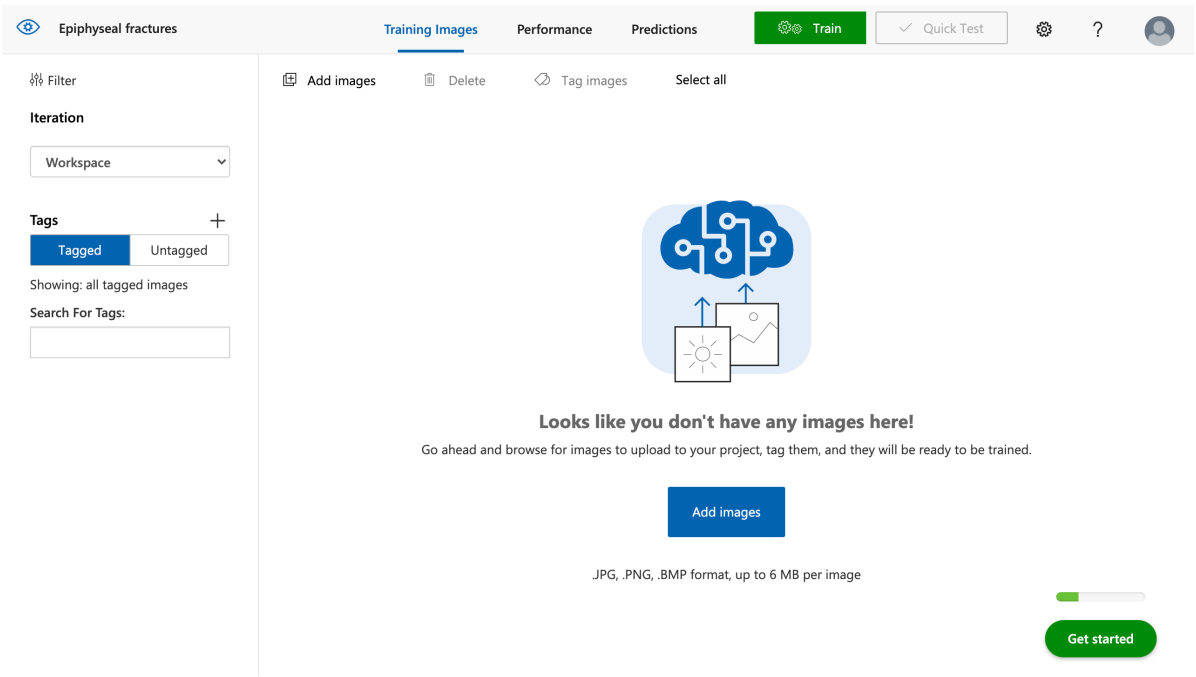


Figure 3.20: Custom vision portal from Microsoft. Screenshot from 07th June 2020.

As Azure is an entire ecosystem that is growing more and more and provides various sets of services that all tie together perfectly. This allows hospitals to reduce their costs and implementation resources to a minimum. This is one of the reasons, why we choose to include Azure custom vision ai as a resource that must be tested. (Soh, Copeland, Puca, & Harris, 2020)(Shaikh, n.d.)

4 Method

We are developing with Jupyter Notebooks running in Python 3.7 with Tensorflow 2 in AzureML using Standard_NC6 instances. Those contain 6x vCPU Intel Xeon E5-2690 v3, 56 GiB RAM, 1 GPU Nvidia Tesla K80 with 12 GiB VRAM.

Our images are distributed as followed:

Total training images: 20082 Total validation images: 1475

Whereas those are categorized into two classes, fractured and unfractured. See table 4.1 for more information.

Type	Class	Count
training	fractured	9261
training	unfractured	10821
validation	fractured	893
validation	unfractured	582

Table 4.1: Distribution of classes in training and validation dataset

The amount of total unfractured images is 11403 (52,89%) the amount of fractured images is 10154 (47,09%), which allows for an unbiased machine learning model as the ratio for unfractured to fractured X-Ray is 1 to 1,12. X-Rays are not always correctly aligned, which leads to difficulty for the ML algorithm to detect the fracture. This misalignment will not be resolved in our model. Adding the black border around the images to resolve any scaling issue. Our training data set always consists of 20082 images and our validation data set contains 1475 images. That amount of data can directly be used and does not need any additional augmentation such as rotating the images or zooming in. The model is validated by running it against the validation dataset. The model has not seen the validation dataset prior. The growth plate fractures are labelled as such.

The data preparation phase underwent the following steps. First of all the images were classified using the supervisely standard. For ResNet, spinenet and MobileNetV2 transformation to another format such as pascalVOC, coco and TFRecord were needed. Tensorflow and Roboflow provide a service to transform the dataset.

The transformation had little to no impact on the quality of the images and the bounding boxes. The supervisley dataset contains the annotations such as metal, axis, text, fracture, fracture student. However, except for the SpineNet implementation, none of the annotations localisation is leveraged. The figure 4.1 and figure 4.2 show examples of the annotations.

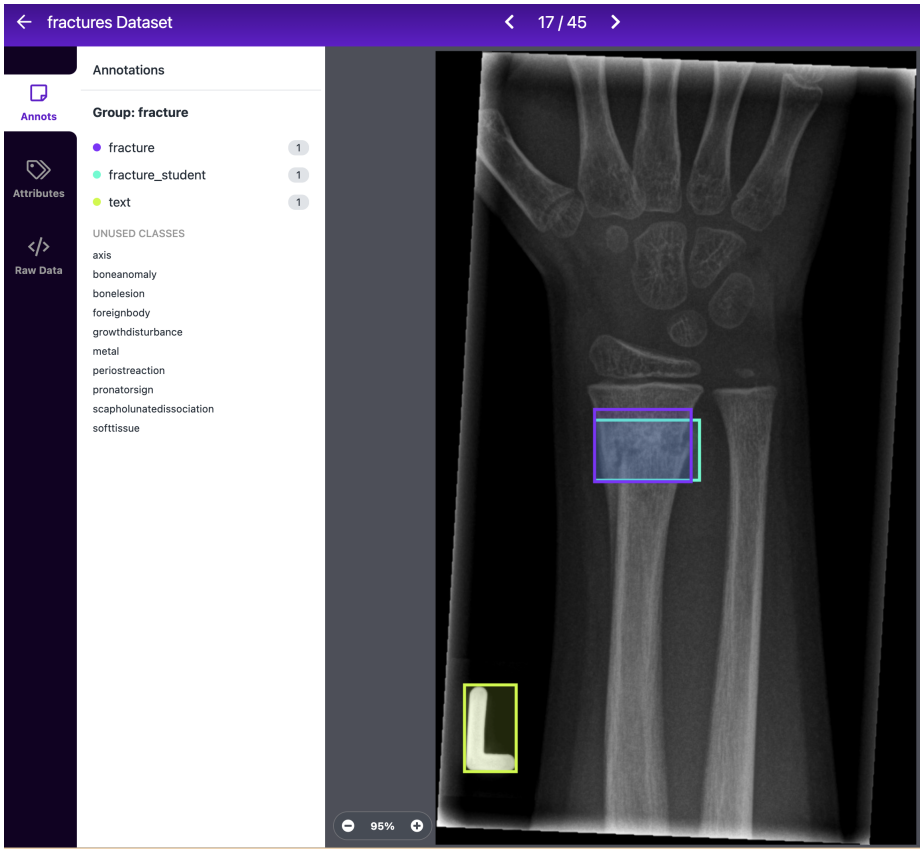


Figure 4.1: The following boxes are displayed. In violett fracture, in blue fracture student, and yellow text.

4.1 Tensorflow

As the industry leader in machine learning, Tensorflow is evaluated using various already existing models. The most known models such as Faster R-CNN, Mask R-CNN, MobileNetV2, ResNet, SpineNet, and ShapeMask seem to be capable of detecting fractures properly. Each model provides it's own advantages and disadvantages. One of the major disadvantages of every existing model is the pre-trained weights. None contain any weights for fractures, thus the model must be trained again with frozen weights. In the second stage of the training, the weights are unfrozen and merged with the preexisting. Creating a new model based on the original one. For this to properly work the images are annotated in either way the model needs. All existing models are taken from the Tensorflow Model Garden (C. Chen et al., 2020).

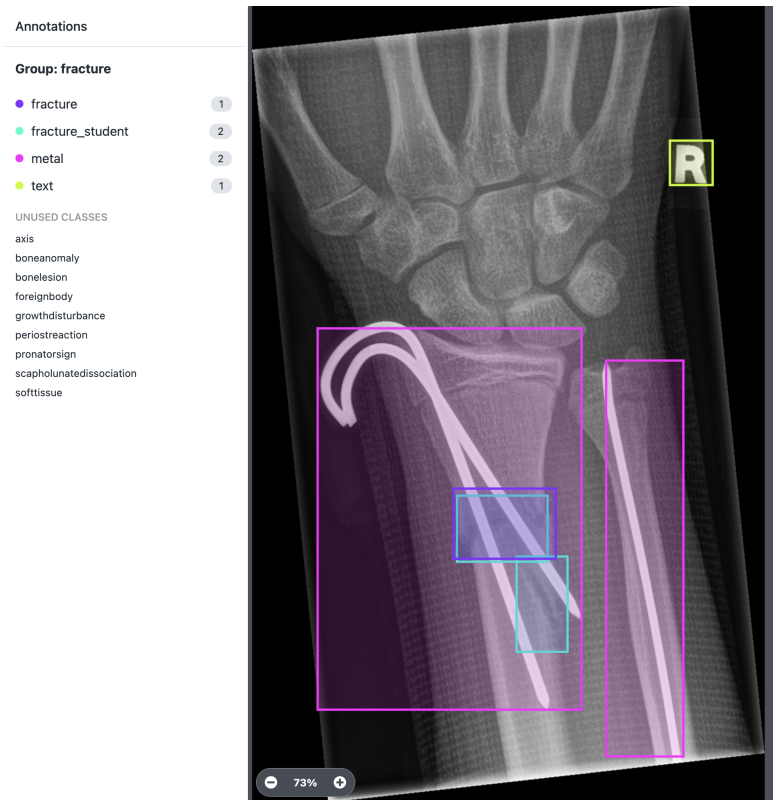


Figure 4.2: The following boxes are displayed. In violett fracture, in blue fracture student, yellow text, and pink metal.

Custom CNN: The primary goal of this is to create a CNN with 128 fully connected layers. Feeding around 64 images per batch and using the full resolution without any downsizing.

MobileNetV2: is limited to square pictures which will stretch our images as those have a 2:1 ratio as opposed to a 2:2 ratio. The implementation stretches the width to the height parameter. This opposes a huge disadvantage, still, MobileNetV2 was once one the fastest and best models available. (Sandler, Howard, Zhu, Zhmoginov, & Chen, 2018)

Faster R-CNN: is the leading model as of today. This model does not only allow to quickly validate and train a model, it has low memory consumption and be used in mobile phones with a lightweight version. The expected performance is quite high, however, R-CNN opposes the issue of overfitting due to the recursion learning process. One picture might be feed to it multiple times. (Ren, He, Girshick, & Sun, 2015)

ResNet: only provides the residual information from one layer to the next one. This provides a faster learning approach and reduces the memory footprint. ResNet was the winner of the 2015 picture challenge. (He et al., 2016)

SpineNet: is based on a permutation of ResNet which modifies the architecture of the entire deep neural network. Networks usually get more performance and accuracy by adding width and height to them, more layers, more fully connected blocks. As opposed to MobileNetV2, SpineNet takes multiple decoders and encoder layers and feeds them to each other, which increased the resolution of the image fed. As shown in Figure 4.3 blocks can have various resolutions as inputs, which allows for better localization and recognition of small objects due to feature size. (Du et al., 2019)

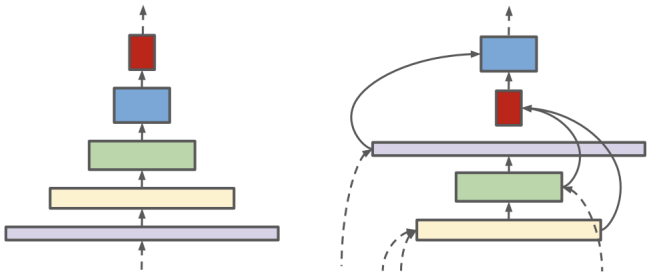


Figure 4.3: On the left side a typical deep neural network with a forward approach is shown. On the right side, a scale-permuted network is shown. The width and height of the blocks show the resolution of the image. Dotted arrows represent incoming and outgoing connections to blocks not shown in this figure. Reprinted with permission from ©(Du et al., 2019) IEEE

Hržić et al. (Hržić et al., 2019) supposed to remove the bone tissue. Mask R-CNN, ShapeMask are public available models that solve this approach by detecting either bone material or removing the unnecessary picture data.

Mask R-CNN: is built on the faster r-cnn and uses a similar approach instead of detection the object it creates a mask to segment the object. (Abdulla, 2017)(He, Gkioxari, Dollár, & Girshick, 2017)

ShapeMask: is a newer implementation of a masking model as Mask R-CNN and built on top of the well known RetinaNet (MobileNetV2). ShapeMask segments the images by first detecting a box, then creating a detection prior. Based on that prior a coarse mask is applied and this mask is refined until a fine mask is found. The performance of ShapeMask is around 20x faster than mask r-cnn. (Kuo, Angelova, Malik, & Lin, 2019)

ShapeMask or Mask R-CNN are the preprocessors to the models defined above. Segmenting the image and providing the mask and original image allows either network to perform more efficiently. By providing only the detected bones and removing all unnecessary clutter the resolution of the fed image can be increased.

4.2 Azure custom vision

Azure custom vision features a RESTful API that either validates a dataset based on already existing models or trains a new model. We decided to run both tests, we leverage the same pictures as mentioned above. The maximum learning duration of azure custom vision is 24 hours. The model trained for the exact amount of time. Figure 4.4 shows that the precision nearly reached 3/4. Multiple test runs with validation images were done. Two good examples are shown in figure 4.5 and 4.6 both being near 75% accuracy.

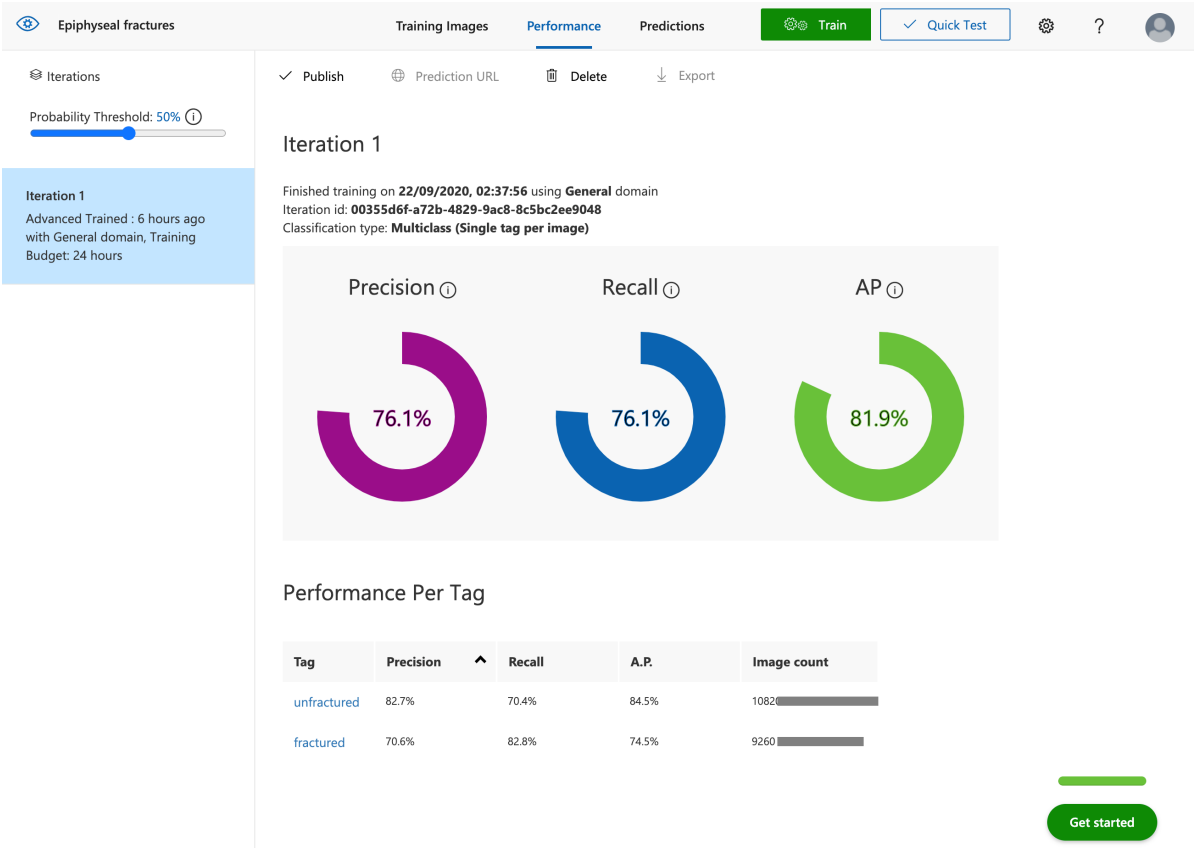


Figure 4.4: Azure custom vision ai iteration using 24 hour budget.

Figure 4.5: Azure custom vision ai quick test of fractured x-ray.

Figure 4.6: Azure custom vision ai quick test of fractured x-ray.

5 Results

All of the listed methods have been implemented using Tensorflow and or Azure custom vision ai. More and detailed implementations can be found in the 7. A short overview of the findings is presented in the table 5.1.

Model	Accuracy
ShapeMask and SpineNet	80,9%
Custom CNN	74,6%
Azure custom vision ai	73,7%
SpineNet	73,3%
Faster R-CNN	71,8%
ResNet	69,1%
MobileNetV2	46,1%

Table 5.1: Accuracy results of different models.

The best single performer is the custom made CNN which combines multiple convolution layer with various feature map sizes. When combining multiple frameworks in a sort of ensemble learning technique that resulted in an even better accuracy.

The custom made CNN has many fully connected layers, can be seen in listing 5.1 and reached a high accuracy whilst being ressource hoggy. The VRAM usage was at the maximum and training took nearly 7 to 8 hours. The accuracy could be improved when adding more potent hardware.

```
model = Sequential([
    Conv2D(16, 3, padding='same', activation='relu', input_shape=(↩
    img_height, img_width ,3)),
    MaxPooling2D((2, 2)),
    Conv2D(32, 3, padding='same', activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(64, 3, padding='same', activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, 3, padding='same', activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(256, 3, padding='same', activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(256, 3, padding='same', activation='relu'),
    MaxPooling2D((2, 2)),
```

```
    Conv2D(512, 3, padding='same', activation='relu'),
    Flatten(),
    Dense(2048, activation='relu'),
    Dense(3, activation='softmax')
])
```

Listing 5.1: custom made CNN

MobileNetV2 struggled with the data the most as it can only deal with a 224x224 image size. This alone reduces the detection rate of any good deep neural network. Even when feeding multiple 224x224 images using autoencoders, the detection rate did not improve.

Custom vision was able to detect 73,7% correctly which places it second. Preprocessing the images to be better compliant for such a non-modifiable network it might even yield higher results. Additionally, this method does not cost any local hardware ressources and instead is designed as a neural network as a service, which in turn allows health care systems to reduce their maintenance and development costs.

SpineNet is the best public available performer and tops in with 73,3% accuracy. Interestingly this model trained for a short amount of time and resource usage was low. Additionally, it outperformed the direct competitor ResNet by nearly 4% and resource usage was 10% to 20% down.

ShapeMask and SpineNet in combination outperformed the rest. This is mainly due to the possibility of feeding higher quality images to SpineNet. ShapeMask was trained using the labelled boxes and did a region detection of a possible fracture location, extracted this part of the image and fed it to the SpineNet.

To answer the research question at hand 'What can be seen as a successful machine learning framework in the context of medical image analysis, regarding Epiphyseal fractures?' which lists the following two questions.

- Current frameworks are sufficient to recognize and diagnose Epiphyseal fractures.
- Bone tissue segmentation is required to receive high accuracy.

A successful machine learning framework in the current state of technology is defined as something that outperforms human skills, e.g. MURA (Rajpurkar, Irvin, Bagul, et al., 2017). The baseline for epiphyseal fractures is around 82% to 85%, since studies suggest that 15% to 18% are miss-diagnosed.

- Current frameworks are sufficient to recognize and diagnose Epiphyseal fractures.

Current frameworks are often not optimized for such images. Most of them struggle with wrong ratios on the input picture. Furthermore, the convolutional approaches used are often defined by time and used resources. This is particularly relevant for commercial applications, opposed to clinical use where quality is the main driver. Combining different models often introduces the thread of biased behavior and reduces the understanding of the network.

One thing that all open source frameworks and implementations have in common is reducing the image size. This is especially a known issue for frameworks such as MobileNetV2, since that requires square input with a maximum resolution of 244 by 244.

The drawback of normal scale-decreased backbone networks is that either the feature or the localisation might be detected. As the encoder computes features from the input, the backbone looses information on where being. Additionally more connected layers directly decreases the ability to properly detect features. SpineNet solves this and the results support that.

Out of the box frameworks alone might not be accurate enough. Combining multiple frameworks and creating an ensemble learning technique as with ShapeMask and SpineNet does yield the best results and has the most complexitiy.

Basically summarizing is that the current frameworks are not sufficient in detection accuracy to beat professional doctors.

- Bone tissue segmentation is required to receive high accuracy.

As multiple papers suggested removing unnecessary data from any image automatically improves the detection rate of any neural network. Medical University Graz, Hržić et al. (Hržić et al., 2019) showed that up to 91.16% where segmented and 86.22% classified correctly. Leveraging such networks as ShapeMask allows for quick and easy segmentation. Further studies are needed whether masking the bone itself and feeding the bone to the network versus masking everything except the bone is the better way to go.

The ensemble learning technique implemented using the ShapeMask model examplifies that proper annotations and labelled boxes are worth checking. The implementation trained ShapeMask to detect the possible fracture location and feed a cropped image in the highest possible resolution to ResNet. This high resolution image allows more features to be extracted which results in better accuracy. The most noteable achievement hereby is, that ShapeMask does not impact the runtime as much as expected. The runtime totaled in with 8 hours and RAM usage was up to 12GB. Compared with non ensemble learning techniques, this RAM usage was regularly observed. However, the hardware was used in a more efficient way since 4/5 of the image are redacted.

In conclusion current frameworks and models are good enough for mass use, they still lack the information and high quality approach needed for medical data. Most networks cannot deal with large and resource intensive images. As of today and for this use custom build CNN are required.

6 Discussion

1.7 billion people suffer from musculoskeletal diseases which are diagnosed by professional radiologists. Their diagnosis precision is around 85%, which still is superior to machine learning-based classification.

In the current study, we chose Epiphyseal plate fracture due to their high abundance in pediatric emergency care, to study the current state of the art machine learning models for real-world X-Ray image analysis. We found that generally speaking custom made CNN outperform the public available models for basic classification tasks. This is to be expected since public available models seldomly are optimized for greyscale biomedical images. None of the models tested have pre-trained weights for fractures, bones and/or metal. One candidate model that overcomes those issues would be the MURA model (Rajpurkar, Irvin, Bagul, et al., 2017) which is specifically trained for the detection of abnormal and normal X-Rays. However, it was not part of the testing as it is not publically available and is rarely used in the global community.

The method developed proposed by Hržić et al. (Hržić et al., 2019) yielded a precision of 91,16% with wrist fractures in pediatric cases. This novel method removed the bone tissue and detected the outer bone contour which then was used to detect any fractures. However, computational resource needs are very demanding due to the design of the neural network. Future optimisation either within the architecture itself, or the hardware, could make this method the gold standard inaccurate classification.

Furthermore, our research showed that ensemble learning techniques can be leveraged to reduce the memory footprint whilst increasing detection accuracy. A combination of out-of-the-box ShapeMask and SpineNet achieved 80,9% correct predictions. Future implementations could improve accuracy when running on more potent hardware as well as using higher resolution X-Rays.

The most disappointing candidate was the MobileNetV2, whilst once winning the ImageNet challenge, it did poorly with roughly 40% accurate predictions. This is due to the underlying architecture as it only works with a square ratio with a maximum resolution of 224x224. X-Rays usually are 2:1 ratios and have at least 1000 pixel per side, thus the transformation causes loss of information which in turn harms the prediction accuracy.

Summarizing the study provides a comparison of all current state of the art machine learning models that are widely used in the community for general image analysis purposes in the application of X-Ray based Epiphyseal plate fracture classification. Basically, none of the models outperformed radiologists by any stretch. Nonetheless, machine learning-based image analysis could still accommodate physicians in their day to day work and help them discover possible fractures. The machine learning field is vastly evolving and growing since 2015 and future developments will eventually provide methods to be on par with the best radiologists.

7 Appendix

7.1 Tensorflow

7.2 Dataset conversions

The next part is the conversation from coco to tfrecords, which is part of the base implementation of Tensorflow. More can be seen here: `https://github.com/tensorflow/models/blob/master/research/object_detection/dataset_tools/create_coco_tf_record.py`

```
python voc2coco.py \
--ann_dir '/Users/philipp/Documents/pascalVOC/wrist/Annotations' \
--ann_ids '/Users/philipp/Documents/pascalVOC/wrist/dataset_ids/train.txt'
--labels '/Users/philipp/Documents/pascalVOC/wrist/labels.txt' \
--output '/Users/philipp/Documents/pascalVOC/train.json'
--ext xml


python voc2coco.py \
--ann_dir '/Users/philipp/Documents/pascalVOC/wrist/Annotations' \
--ann_ids '/Users/philipp/Documents/pascalVOC/wrist/dataset_ids/val.txt'
--labels '/Users/philipp/Documents/pascalVOC/wrist/labels.txt' \
--output '/Users/philipp/Documents/pascalVOC/val.json'
--ext xml
```

Listing 7.1: pascalVOC to Coco

```
import os
import argparse
import json
import xml.etree.ElementTree as ET
from typing import Dict, List
from tqdm import tqdm
import re


def get_label2id(labels_path: str) -> Dict[str, int]:
    """id is 1 start"""
```

```
    with open(labels_path, 'r') as f:
        labels_str = f.read().split()
    labels_ids = list(range(1, len(labels_str)+1))
    return dict(zip(labels_str, labels_ids))


def get_annpaths(ann_dir_path: str = None,
                 ann_ids_path: str = None,
                 ext: str = '',
                 annpaths_list_path: str = None) -> List[str]:
    # If use annotation paths list
    if annpaths_list_path is not None:
        with open(annpaths_list_path, 'r') as f:
            ann_paths = f.read().split()
        return ann_paths

    # If use annotaion ids list
    ext_with_dot = '.' + ext if ext != '' else ''
    with open(ann_ids_path, 'r') as f:
        ann_ids = f.read().split()
    ann_paths = [os.path.join(ann_dir_path, aid+ext_with_dot) for aid ↩
    in ann_ids]
    return ann_paths


def get_image_info(annotation_root, extract_num_from_imgid=True):
    path = annotation_root.findtext('path')
    if path is None:
        filename = annotation_root.findtext('filename')
    else:
        filename = os.path.basename(path)
    img_name = os.path.basename(filename)
    img_id = os.path.splitext(img_name)[0]
    if extract_num_from_imgid and isinstance(img_id, str):
        img_id = int(re.findall(r'\d+', img_id)[0])

    size = annotation_root.find('size')
    width = int(size.findtext('width'))
    height = int(size.findtext('height'))

    image_info = {
        'file_name': filename,
        'height': height,
        'width': width,
        'id': img_id
    }
    return image_info


def get_coco_annotation_from_obj(obj, label2id):
    label = obj.findtext('name')
    assert label in label2id, f"Error: {label} is not in label2id !"
    category_id = label2id[label]
    bndbox = obj.find('bndbox')
    xmin = int(bndbox.findtext('xmin')) - 1
```

```
    ymin = int(bndbox.findtext('ymin')) - 1
    xmax = int(bndbox.findtext('xmax'))
    ymax = int(bndbox.findtext('ymax'))
    assert xmax > xmin and ymax > ymin, f"Box size error !: (xmin, ymin, xmax, ymax): {xmin, ymin, xmax, ymax}"
    o_width = xmax - xmin
    o_height = ymax - ymin
    ann = {
        'area': o_width * o_height,
        'iscrowd': 0,
        'bbox': [xmin, ymin, o_width, o_height],
        'category_id': category_id,
        'ignore': 0,
        'segmentation': []  # This script is not for segmentation
    }
    return ann


def convert_xmls_to_cocojson(annotation_paths: List[str],
                             label2id: Dict[str, int],
                             output_jsonpath: str,
                             extract_num_from_imgid: bool = True):
    output_json_dict = {
        "images": [],
        "type": "instances",
        "annotations": [],
        "categories": []
    }
    bnd_id = 1  # START_BOUNDING_BOX_ID, TODO input as args ?
    print('Start converting !')
    for a_path in tqdm(annotation_paths):
        # Read annotation xml
        ann_tree = ET.parse(a_path)
        ann_root = ann_tree.getroot()

        img_info = get_image_info(annotation_root=ann_root,
                                  extract_num_from_imgid=extract_num_from_imgid)
        img_id = img_info['id']
        output_json_dict['images'].append(img_info)

        for obj in ann_root.findall('object'):
            ann = get_coco_annotation_from_obj(obj=obj, label2id=label2id)
            ann.update({'image_id': img_id, 'id': bnd_id})
            output_json_dict['annotations'].append(ann)
            bnd_id = bnd_id + 1

    for label, label_id in label2id.items():
        category_info = {'supercategory': 'none', 'id': label_id, 'name': label}
        output_json_dict['categories'].append(category_info)

    with open(output_jsonpath, 'w') as f:
        output_json = json.dumps(output_json_dict)
```

```
        f.write(output_json)

def main():
    parser = argparse.ArgumentParser(
        description='This script support converting voc format xmls to coco format json')
    parser.add_argument('--ann_dir', type=str, default=None,
                        help='path to annotation files directory. It is not need when use --ann_paths_list')
    parser.add_argument('--ann_ids', type=str, default=None,
                        help='path to annotation files ids list. It is not need when use --ann_paths_list')
    parser.add_argument('--ann_paths_list', type=str, default=None,
                        help='path of annotation paths list. It is not need when use --ann_dir and --ann_ids')
    parser.add_argument('--labels', type=str, default=None,
                        help='path to label list.')
    parser.add_argument('--output', type=str, default='output.json', help='path to output json file')
    parser.add_argument('--ext', type=str, default='', help='additional extension of annotation file')
    parser.add_argument('--extract_num_from_imgid', action="store_true",
                        help='Extract image number from the image filename')
    args = parser.parse_args()
    label2id = get_label2id(labels_path=args.labels)
    ann_paths = get_annpaths(
        ann_dir_path=args.ann_dir,
        ann_ids_path=args.ann_ids,
        ext=args.ext,
        annpaths_list_path=args.ann_paths_list
    )
    convert_xmls_to_cocojson(
        annotation_paths=ann_paths,
        label2id=label2id,
        output_jsonpath=args.output,
        extract_num_from_imgid=args.extract_num_from_imgid
    )


if __name__ == '__main__':
    main()
```

Listing 7.2: pascalVOC to Coco python code

7.3 Custom CNN

The first model proves that Tensorflows default CNN is capable of detecting 2/3 of all X-Rays correctly.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 540, 203, 16)      448
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 270, 101, 16)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 270, 101, 32)      4640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 135, 50, 32)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 135, 50, 64)       18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 67, 25, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 67, 25, 128)       73856
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 33, 12, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 33, 12, 256)       295168
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 16, 6, 256)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 16, 6, 512)        4719104
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 8, 3, 512)         0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 8, 3, 1024)        18875392
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 4, 1, 1024)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 4, 1, 2048)        75499520
_________________________________________________________________
flatten (Flatten)            (None, 8192)              0
_________________________________________________________________
dense (Dense)                (None, 128)               1048704
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 387
=================================================================
Total params: 100,535,715
Trainable params: 100,535,715
Non-trainable params: 0
_________________________________________________________________
```

Listing 7.3: First model summary

```
Epoch 1/15
```

```
627/627 [==============================] - 241s 385ms/step - loss: ↩
    0.6946 -accuracy: 0.5571 - val_loss: 0.7410 - val_accuracy: 0.5387
Epoch 2/15
627/627 [==============================] - 208s 331ms/step - loss: ↩
    0.6485 -accuracy: 0.6234 - val_loss: 0.7212 - val_accuracy: 0.5618
Epoch 3/15
627/627 [==============================] - 207s 329ms/step - loss: ↩
    0.6306 -accuracy: 0.6541 - val_loss: 0.6928 - val_accuracy: 0.5686
Epoch 4/15
627/627 [==============================] - 220s 352ms/step - loss: ↩
    0.6218 -accuracy: 0.6633 - val_loss: 0.7229 - val_accuracy: 0.5543
Epoch 5/15
627/627 [==============================] - 223s 355ms/step - loss: ↩
    0.6126 -accuracy: 0.6758 - val_loss: 0.7043 - val_accuracy: 0.5727
Epoch 6/15
627/627 [==============================] - 223s 355ms/step - loss: ↩
    0.6060 -accuracy: 0.6833 - val_loss: 0.7218 - val_accuracy: 0.5625
Epoch 7/15
627/627 [==============================] - 223s 355ms/step - loss: ↩
    0.5950 -accuracy: 0.6920 - val_loss: 0.6794 - val_accuracy: 0.6230
Epoch 8/15
627/627 [==============================] - 223s 355ms/step - loss: ↩
    0.5900 -accuracy: 0.6987 - val_loss: 0.6975 - val_accuracy: 0.5992
Epoch 9/15
627/627 [==============================] - 223s 355ms/step - loss: ↩
    0.5839 -accuracy: 0.7004 - val_loss: 0.6863 - val_accuracy: 0.6148
Epoch 10/15
627/627 [==============================] - 222s 355ms/step - loss: ↩
    0.5766 -accuracy: 0.7090 - val_loss: 0.7094 - val_accuracy: 0.6073
Epoch 11/15
627/627 [==============================] - 223s 355ms/step - loss: ↩
    0.5698 -accuracy: 0.7111 - val_loss: 0.6785 - val_accuracy: 0.6202
Epoch 12/15
627/627 [==============================] - 223s 356ms/step - loss: ↩
    0.5603 -accuracy: 0.7185 - val_loss: 0.6921 - val_accuracy: 0.6338
Epoch 13/15
627/627 [==============================] - 223s 355ms/step - loss: ↩
    0.5483 -accuracy: 0.7261 - val_loss: 0.7098 - val_accuracy: 0.6257
Epoch 14/15
627/627 [==============================] - 225s 359ms/step - loss: ↩
    0.5329 -accuracy: 0.7355 - val_loss: 0.7193 - val_accuracy: 0.6352
Epoch 15/15
627/627 [==============================] - 223s 355ms/step - loss: ↩
    0.5092 -accuracy: 0.7497 - val_loss: 0.7208 - val_accuracy: 0.6223
```

Listing 7.4: First model run

Some papers suggest using dense fully connected layers between the convolution layer as a bridge.

```
model = Sequential([
    Conv2D(16, 3, padding='same', activation='relu', input_shape=(↩
    IMG_HEIGHT, IMG_WIDTH ,3)),
    MaxPooling2D((2, 2)),
```

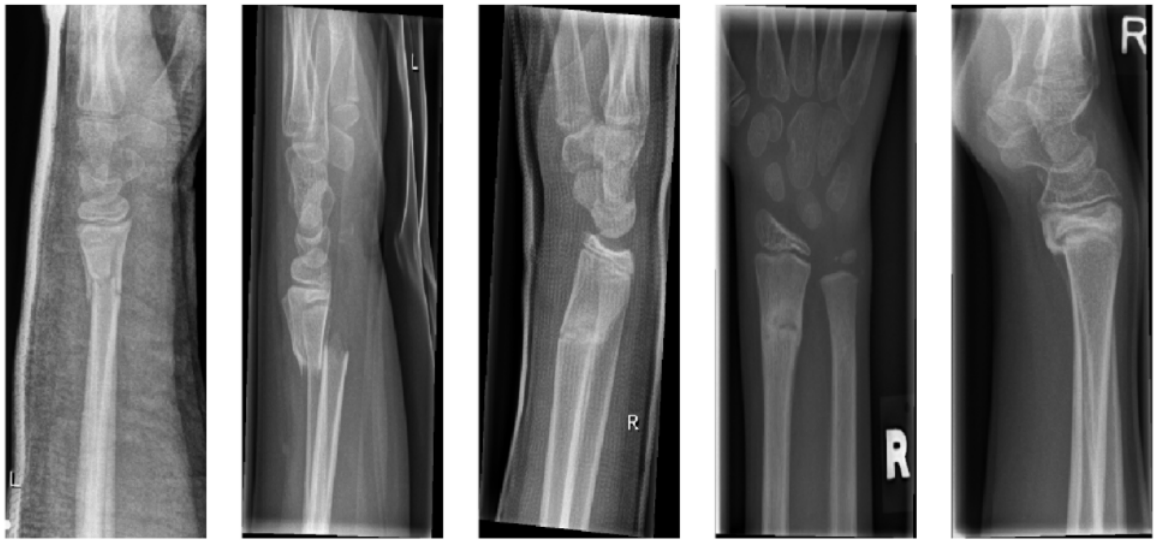


Figure 7.1: First run showing five resized sample images.

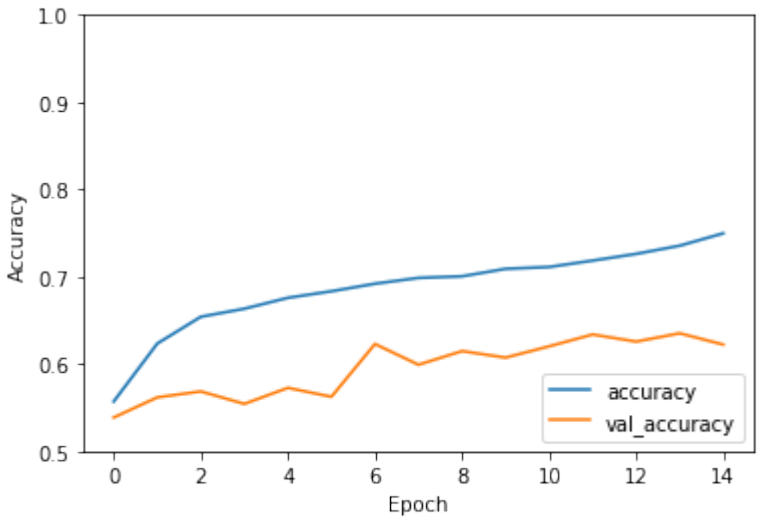


Figure 7.2: First run showing acuracy of training images and validation accuracy.

```
    Conv2D(32, 3, padding='same', activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(64, 3, padding='same', activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, 3, padding='same', activation='relu'),
    Dense(128, activation='relu'),
    Conv2D(256, 3, padding='same', activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(256, 3, padding='same', activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(512, 3, padding='same', activation='relu'),
    Flatten(),
    Dense(2048, activation='relu'),
```

```
    Dense(3, activation='softmax')
])
```

Listing 7.5: Second model definition

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_14 (Conv2D)           (None, 540, 203, 16)      448
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 270, 101, 16)      0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 270, 101, 32)      4640
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 135, 50, 32)       0
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 135, 50, 64)       18496
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 67, 25, 64)        0
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 67, 25, 128)       73856
_________________________________________________________________
dense_5 (Dense)              (None, 67, 25, 128)       16512
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 67, 25, 256)       295168
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 33, 12, 256)       0
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 33, 12, 256)       590080
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 16, 6, 256)        0
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 16, 6, 512)        1180160
_________________________________________________________________
flatten_3 (Flatten)          (None, 49152)             0
_________________________________________________________________
dense_6 (Dense)              (None, 2048)              100665344
_________________________________________________________________
dense_7 (Dense)              (None, 3)                 6147
=================================================================
Total params: 102,850,851
Trainable params: 102,850,851
Non-trainable params: 0
_________________________________________________________________
```

Listing 7.6: Second model summary

```
Epoch 1/15
627/627 [==============================] - 195s 311ms/step - loss: ↩
    0.7000 - accuracy: 0.5219 - val_loss: 0.7190 - val_accuracy: 0.3933
Epoch 2/15
627/627 [==============================] - 194s 309ms/step - loss: ↩
    0.6929 - accuracy: 0.5354 - val_loss: 0.7334 - val_accuracy: 0.3933
Epoch 3/15
```

```
627/627 [==============================] - 192s 307ms/step - loss: ↩
    0.6921 - accuracy: 0.5380 - val_loss: 0.7316 - val_accuracy: 0.3933
Epoch 4/15
627/627 [==============================] - 193s 307ms/step - loss: ↩
    0.6922 - accuracy: 0.5371 - val_loss: 0.7174 - val_accuracy: 0.3933
Epoch 5/15
627/627 [==============================] - 195s 310ms/step - loss: ↩
    0.6918 - accuracy: 0.5371 - val_loss: 0.7309 - val_accuracy: 0.3933
Epoch 6/15
627/627 [==============================] - 192s 307ms/step - loss: ↩
    0.6920 - accuracy: 0.5378 - val_loss: 0.7342 - val_accuracy: 0.3933
Epoch 7/15
627/627 [==============================] - 192s 307ms/step - loss: ↩
    0.6916 - accuracy: 0.5370 - val_loss: 0.7185 - val_accuracy: 0.3933
Epoch 8/15
627/627 [==============================] - 194s 309ms/step - loss: ↩
    0.6919 - accuracy: 0.5360 - val_loss: 0.7400 - val_accuracy: 0.3933
Epoch 9/15
627/627 [==============================] - 195s 311ms/step - loss: ↩
    0.6917 - accuracy: 0.5388 - val_loss: 0.7239 - val_accuracy: 0.3933
Epoch 10/15
627/627 [==============================] - 195s 311ms/step - loss: ↩
    0.6917 - accuracy: 0.5389 - val_loss: 0.7135 - val_accuracy: 0.3933
Epoch 11/15
627/627 [==============================] - 199s 317ms/step - loss: ↩
    0.6918 - accuracy: 0.5386 - val_loss: 0.7499 - val_accuracy: 0.3933
Epoch 12/15
627/627 [==============================] - 203s 325ms/step - loss: ↩
    0.6918 - accuracy: 0.5385 - val_loss: 0.7122 - val_accuracy: 0.3933
Epoch 13/15
627/627 [==============================] - 200s 318ms/step - loss: ↩
    0.6916 - accuracy: 0.5366 - val_loss: 0.7069 - val_accuracy: 0.3933
Epoch 14/15
627/627 [==============================] - 199s 318ms/step - loss: ↩
    0.6912 - accuracy: 0.5398 - val_loss: 0.7307 - val_accuracy: 0.3933
Epoch 15/15
627/627 [==============================] - 200s 319ms/step - loss: ↩
    0.6915 - accuracy: 0.5379 - val_loss: 0.7326 - val_accuracy: 0.3933
```

Listing 7.7: Second model run

The additional dense layer does not positively impact the detection rate of epiphysis fractures. The model is more complex and computational heavy to calculate, as shown in figure 7.3.

7.4 Faster R-CNN

7.5 mobileNetV2

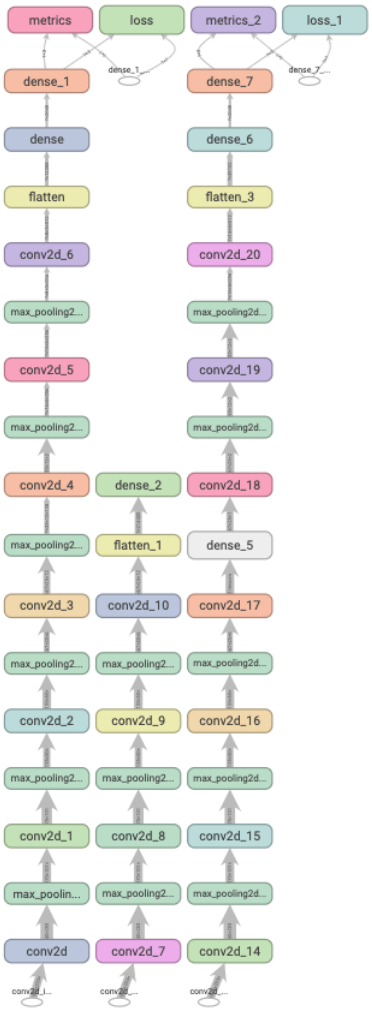


Figure 7.3: Second model CNN architecture

```
Epoch 1/30
627/627 [==============================] - 159s 254ms/step - loss: nan ↩
    - accuracy: 0.4605 -val_loss: nan - val_accuracy: 0.6052
Epoch 2/30
627/627 [==============================] - 160s 255ms/step - loss: nan ↩
    - accuracy: 0.4611 -val_loss: nan - val_accuracy: 0.6052
Epoch 3/30
627/627 [==============================] - 158s 253ms/step - loss: nan ↩
    - accuracy: 0.4611 -val_loss: nan - val_accuracy: 0.6052
Epoch 4/30
627/627 [==============================] - 158s 252ms/step - loss: nan ↩
    - accuracy: 0.4611 -val_loss: nan - val_accuracy: 0.6052
Epoch 5/30
627/627 [==============================] - 159s 253ms/step - loss: nan ↩
    - accuracy: 0.4611 -val_loss: nan - val_accuracy: 0.6052
Epoch 6/30
```

```
627/627 [==============================] - 158s 252ms/step - loss: nan ↩
    - accuracy: 0.4611 -val_loss: nan - val_accuracy: 0.6052
Epoch 7/30
627/627 [==============================] - 158s 252ms/step - loss: nan ↩
    - accuracy: 0.4611 -val_loss: nan - val_accuracy: 0.6052
Epoch 8/30
627/627 [==============================] - 158s 252ms/step - loss: nan ↩
    - accuracy: 0.4611 -val_loss: nan - val_accuracy: 0.6052
Epoch 9/30
627/627 [==============================] - 158s 252ms/step - loss: nan ↩
    - accuracy: 0.4611 -val_loss: nan - val_accuracy: 0.6052
Epoch 10/30
627/627 [==============================] - 158s 252ms/step - loss: nan ↩
    - accuracy: 0.4611 -val_loss: nan - val_accuracy: 0.6052
Epoch 11/30
627/627 [==============================] - 158s 253ms/step - loss: nan ↩
    - accuracy: 0.4611 -val_loss: nan - val_accuracy: 0.6052
Epoch 12/30
627/627 [==============================] - 158s 252ms/step - loss: nan ↩
    - accuracy: 0.4611 -val_loss: nan - val_accuracy: 0.6052
Epoch 13/30
627/627 [==============================] - 158s 252ms/step - loss: nan ↩
    - accuracy: 0.4611 -val_loss: nan - val_accuracy: 0.6052
Epoch 14/30
627/627 [==============================] - 158s 252ms/step - loss: nan ↩
    - accuracy: 0.4611 -val_loss: nan - val_accuracy: 0.6052
Epoch 15/30
627/627 [==============================] - 158s 252ms/step - loss: nan ↩
    - accuracy: 0.4611 -val_loss: nan - val_accuracy: 0.6052
Epoch 16/30
627/627 [==============================] - 158s 252ms/step - loss: nan ↩
    - accuracy: 0.4611 -val_loss: nan - val_accuracy: 0.6052
Epoch 17/30
627/627 [==============================] - 158s 252ms/step - loss: nan ↩
    - accuracy: 0.4611 -val_loss: nan - val_accuracy: 0.6052
Epoch 18/30
627/627 [==============================] - 158s 252ms/step - loss: nan ↩
    - accuracy: 0.4611 -val_loss: nan - val_accuracy: 0.6052
Epoch 19/30
627/627 [==============================] - 158s 252ms/step - loss: nan ↩
    - accuracy: 0.4611 -val_loss: nan - val_accuracy: 0.6052
Epoch 20/30
627/627 [==============================] - 158s 252ms/step - loss: nan ↩
    - accuracy: 0.4611 -val_loss: nan - val_accuracy: 0.6052
Epoch 21/30
627/627 [==============================] - 158s 252ms/step - loss: nan ↩
    - accuracy: 0.4611 -val_loss: nan - val_accuracy: 0.6052
Epoch 22/30
627/627 [==============================] - 158s 252ms/step - loss: nan ↩
    - accuracy: 0.4611 -val_loss: nan - val_accuracy: 0.6052
Epoch 23/30
627/627 [==============================] - 158s 252ms/step - loss: nan ↩
    - accuracy: 0.4611 - val_loss: nan - val_accuracy: 0.6052
Epoch 24/30
```

```
627/627 [==============================] - 158s 252ms/step - loss: nan - accuracy: 0.4611 -val_loss: nan - val_accuracy: 0.6052
Epoch 25/30
627/627 [==============================] - 158s 252ms/step - loss: nan - accuracy: 0.4611 -val_loss: nan - val_accuracy: 0.6052
Epoch 26/30
627/627 [==============================] - 158s 253ms/step - loss: nan - accuracy: 0.4611 -val_loss: nan - val_accuracy: 0.6052
Epoch 27/30
627/627 [==============================] - 158s 253ms/step - loss: nan - accuracy: 0.4611 -val_loss: nan - val_accuracy: 0.6052
Epoch 28/30
627/627 [==============================] - 159s 253ms/step - loss: nan - accuracy: 0.4611 -val_loss: nan - val_accuracy: 0.6052
Epoch 29/30
627/627 [==============================] - 160s 256ms/step - loss: nan - accuracy: 0.4611 -val_loss: nan - val_accuracy: 0.6052
Epoch 30/30
627/627 [==============================] - 160s 255ms/step - loss: nan - accuracy: 0.4611 -val_loss: nan - val_accuracy: 0.6052
```

Listing 7.8: MobileNetV2 freezed weights run

```
Epoch 31/60 627/627 [==============================] - 167s 266ms/step - loss: nan - accuracy: 0.4611 - val_loss: nan - val_accuracy: 0.6052
Epoch 32/60 627/627 [==============================] - 168s 268ms/step - loss: nan - accuracy: 0.4611 - val_loss: nan - val_accuracy: 0.6052
Epoch 33/60 627/627 [==============================] - 166s 265ms/step - loss: nan - accuracy: 0.4611 - val_loss: nan - val_accuracy: 0.6052
Epoch 34/60 627/627 [==============================] - 168s 268ms/step - loss: nan - accuracy: 0.4611 - val_loss: nan - val_accuracy: 0.6052
Epoch 35/60 627/627 [==============================] - 168s 268ms/step - loss: nan - accuracy: 0.4611 - val_loss: nan - val_accuracy: 0.6052
Epoch 36/60 627/627 [==============================] - 168s 268ms/step - loss: nan - accuracy: 0.4611 - val_loss: nan - val_accuracy: 0.6052
Epoch 37/60 627/627 [==============================] - 168s 268ms/step - loss: nan - accuracy: 0.4611 - val_loss: nan - val_accuracy: 0.6052
Epoch 38/60 627/627 [==============================] - 168s 268ms/step - loss: nan - accuracy: 0.4611 - val_loss: nan - val_accuracy: 0.6052
Epoch 39/60 627/627 [==============================] - 168s 268ms/step - loss: nan - accuracy: 0.4611 - val_loss: nan - val_accuracy: 0.6052
Epoch 40/60 627/627 [==============================] - 165s 268ms/step - loss: nan - accuracy: 0.4611 - val_loss: nan - val_accuracy: 0.6052
```

```
Epoch 41/60 627/627 [==============================] - 168s 268ms/step - loss: nan - accuracy: 0.4611 - val_loss: nan - val_accuracy: 0.6052
Epoch 42/60 627/627 [==============================] - 167s 268ms/step - loss: nan - accuracy: 0.4611 - val_loss: nan - val_accuracy: 0.6052
Epoch 43/60 627/627 [==============================] - 168s 268ms/step - loss: nan - accuracy: 0.4611 - val_loss: nan - val_accuracy: 0.6052
Epoch 44/60 627/627 [==============================] - 166s 268ms/step - loss: nan - accuracy: 0.4611 - val_loss: nan - val_accuracy: 0.6052
Epoch 45/60 627/627 [==============================] - 168s 268ms/step - loss: nan - accuracy: 0.4611 - val_loss: nan - val_accuracy: 0.6052
Epoch 46/60 627/627 [==============================] - 168s 268ms/step - loss: nan - accuracy: 0.4611 - val_loss: nan - val_accuracy: 0.6052
Epoch 47/60 627/627 [==============================] - 168s 268ms/step - loss: nan - accuracy: 0.4611 - val_loss: nan - val_accuracy: 0.6052
Epoch 48/60 627/627 [==============================] - 167s 268ms/step - loss: nan - accuracy: 0.4611 - val_loss: nan - val_accuracy: 0.6052
Epoch 49/60 627/627 [==============================] - 168s 268ms/step - loss: nan - accuracy: 0.4611 - val_loss: nan - val_accuracy: 0.6052
Epoch 50/60 627/627 [==============================] - 168s 268ms/step - loss: nan - accuracy: 0.4611 - val_loss: nan - val_accuracy: 0.6052
Epoch 51/60 627/627 [==============================] - 164s 268ms/step - loss: nan - accuracy: 0.4611 - val_loss: nan - val_accuracy: 0.6052
Epoch 52/60 627/627 [==============================] - 168s 268ms/step - loss: nan - accuracy: 0.4611 - val_loss: nan - val_accuracy: 0.6052
Epoch 53/60 627/627 [==============================] - 168s 268ms/step - loss: nan - accuracy: 0.4611 - val_loss: nan - val_accuracy: 0.6052
Epoch 54/60 627/627 [==============================] - 168s 268ms/step - loss: nan - accuracy: 0.4611 - val_loss: nan - val_accuracy: 0.6052
Epoch 55/60 627/627 [==============================] - 165s 268ms/step - loss: nan - accuracy: 0.4611 - val_loss: nan - val_accuracy: 0.6052
Epoch 56/60 627/627 [==============================] - 168s 268ms/step - loss: nan - accuracy: 0.4611 - val_loss: nan - val_accuracy: 0.6052
Epoch 57/60 627/627 [==============================] - 168s 268ms/step - loss: nan - accuracy: 0.4611 - val_loss: nan - val_accuracy: 0.6052
Epoch 58/60 627/627 [==============================] - 169s 268ms/step - loss: nan - accuracy: 0.4611 - val_loss: nan - val_accuracy: 0.6052
```

```
Epoch 59/60 627/627 [==============================] - 168s 268ms/step ↩
    - loss: nan - accuracy: 0.4611 - val_loss: nan - val_accuracy: ↩
    0.6052
Epoch 60/60 627/627 [==============================] - 168s 268ms/step ↩
    - loss: nan - accuracy: 0.4611 - val_loss: nan - val_accuracy: ↩
    0.6052
```

Listing 7.9: MobileNetV2 unfrozen weights run

7.6 ResNet

```
__________________________________________________________________
Layer (type)                 Output Shape
==================================================================
input_4 (InputLayer)         [(None, 224, 224, 3)
__________________________________________________________________
zero_padding2d_3 (ZeroPadding2D (None, 230, 230, 3)
__________________________________________________________________
conv2d_159 (Conv2D)          (None, 112, 112, 64)
__________________________________________________________________
batch_normalization_159 (BatchN (None, 112, 112, 64)
__________________________________________________________________
activation_147 (Activation)  (None, 112, 112, 64)
__________________________________________________________________
max_pooling2d_3 (MaxPooling2D)  (None, 55, 55, 64)
__________________________________________________________________
conv2d_160 (Conv2D)          (None, 55, 55, 64)
__________________________________________________________________
batch_normalization_160 (BatchN (None, 55, 55, 64)
__________________________________________________________________
activation_148 (Activation)  (None, 55, 55, 64)
__________________________________________________________________
conv2d_161 (Conv2D)          (None, 55, 55, 64)
__________________________________________________________________
batch_normalization_161 (BatchN (None, 55, 55, 64)
__________________________________________________________________
activation_149 (Activation)  (None, 55, 55, 64)
__________________________________________________________________
conv2d_162 (Conv2D)          (None, 55, 55, 256)
__________________________________________________________________
conv2d_163 (Conv2D)          (None, 55, 55, 256)
__________________________________________________________________
batch_normalization_162 (BatchN (None, 55, 55, 256)
__________________________________________________________________
batch_normalization_163 (BatchN (None, 55, 55, 256)
__________________________________________________________________
add_48 (Add)                 (None, 55, 55, 256)
__________________________________________________________________
activation_150 (Activation)  (None, 55, 55, 256)
__________________________________________________________________
conv2d_164 (Conv2D)          (None, 55, 55, 64)
```

```
__________________________________________________________________________
batch_normalization_164 (BatchN (None, 55, 55, 64)
__________________________________________________________________________
activation_151 (Activation)     (None, 55, 55, 64)
__________________________________________________________________________
conv2d_165 (Conv2D)             (None, 55, 55, 64)
__________________________________________________________________________
batch_normalization_165 (BatchN (None, 55, 55, 64)
__________________________________________________________________________
activation_152 (Activation)     (None, 55, 55, 64)
__________________________________________________________________________
conv2d_166 (Conv2D)             (None, 55, 55, 256)
__________________________________________________________________________
batch_normalization_166 (BatchN (None, 55, 55, 256)
__________________________________________________________________________
add_49 (Add)                    (None, 55, 55, 256)
__________________________________________________________________________
activation_153 (Activation)     (None, 55, 55, 256)
__________________________________________________________________________
conv2d_167 (Conv2D)             (None, 55, 55, 64)
__________________________________________________________________________
batch_normalization_167 (BatchN (None, 55, 55, 64)
__________________________________________________________________________
activation_154 (Activation)     (None, 55, 55, 64)
__________________________________________________________________________
conv2d_168 (Conv2D)             (None, 55, 55, 64)
__________________________________________________________________________
batch_normalization_168 (BatchN (None, 55, 55, 64)
__________________________________________________________________________
activation_155 (Activation)     (None, 55, 55, 64)
__________________________________________________________________________
conv2d_169 (Conv2D)             (None, 55, 55, 256)
__________________________________________________________________________
batch_normalization_169 (BatchN (None, 55, 55, 256)
__________________________________________________________________________
add_50 (Add)                    (None, 55, 55, 256)
__________________________________________________________________________
activation_156 (Activation)     (None, 55, 55, 256)
__________________________________________________________________________
conv2d_170 (Conv2D)             (None, 28, 28, 128)
__________________________________________________________________________
batch_normalization_170 (BatchN (None, 28, 28, 128)
__________________________________________________________________________
activation_157 (Activation)     (None, 28, 28, 128)
__________________________________________________________________________
conv2d_171 (Conv2D)             (None, 28, 28, 128)
__________________________________________________________________________
batch_normalization_171 (BatchN (None, 28, 28, 128)
__________________________________________________________________________
activation_158 (Activation)     (None, 28, 28, 128)
__________________________________________________________________________
conv2d_172 (Conv2D)             (None, 28, 28, 512)
__________________________________________________________________________
conv2d_173 (Conv2D)             (None, 28, 28, 512)
__________________________________________________________________________
```

```
batch_normalization_172 (BatchN (None, 28, 28, 512)
_________________________________________________________________
batch_normalization_173 (BatchN (None, 28, 28, 512)
_________________________________________________________________
add_51 (Add)                    (None, 28, 28, 512)
_________________________________________________________________
activation_159 (Activation)     (None, 28, 28, 512)
_________________________________________________________________
conv2d_174 (Conv2D)             (None, 28, 28, 128)
_________________________________________________________________
batch_normalization_174 (BatchN (None, 28, 28, 128)
_________________________________________________________________
activation_160 (Activation)     (None, 28, 28, 128)
_________________________________________________________________
conv2d_175 (Conv2D)             (None, 28, 28, 128)
_________________________________________________________________
batch_normalization_175 (BatchN (None, 28, 28, 128)
_________________________________________________________________
activation_161 (Activation)     (None, 28, 28, 128)
_________________________________________________________________
conv2d_176 (Conv2D)             (None, 28, 28, 512)
_________________________________________________________________
batch_normalization_176 (BatchN (None, 28, 28, 512)
_________________________________________________________________
add_52 (Add)                    (None, 28, 28, 512)
_________________________________________________________________
activation_162 (Activation)     (None, 28, 28, 512)
_________________________________________________________________
conv2d_177 (Conv2D)             (None, 28, 28, 128)
_________________________________________________________________
batch_normalization_177 (BatchN (None, 28, 28, 128)
_________________________________________________________________
activation_163 (Activation)     (None, 28, 28, 128)
_________________________________________________________________
conv2d_178 (Conv2D)             (None, 28, 28, 128)
_________________________________________________________________
batch_normalization_178 (BatchN (None, 28, 28, 128)
_________________________________________________________________
activation_164 (Activation)     (None, 28, 28, 128)
_________________________________________________________________
conv2d_179 (Conv2D)             (None, 28, 28, 512)
_________________________________________________________________
batch_normalization_179 (BatchN (None, 28, 28, 512)
_________________________________________________________________
add_53 (Add)                    (None, 28, 28, 512)
_________________________________________________________________
activation_165 (Activation)     (None, 28, 28, 512)
_________________________________________________________________
conv2d_180 (Conv2D)             (None, 28, 28, 128)
_________________________________________________________________
batch_normalization_180 (BatchN (None, 28, 28, 128)
_________________________________________________________________
activation_166 (Activation)     (None, 28, 28, 128)
_________________________________________________________________
conv2d_181 (Conv2D)             (None, 28, 28, 128)
```

```
__________________________________________________________________________________________________
batch_normalization_181 (BatchN (None, 28, 28, 128)
__________________________________________________________________________________________________
activation_167 (Activation)     (None, 28, 28, 128)
__________________________________________________________________________________________________
conv2d_182 (Conv2D)             (None, 28, 28, 512)
__________________________________________________________________________________________________
batch_normalization_182 (BatchN (None, 28, 28, 512)
__________________________________________________________________________________________________
add_54 (Add)                    (None, 28, 28, 512)
__________________________________________________________________________________________________
activation_168 (Activation)     (None, 28, 28, 512)
__________________________________________________________________________________________________
conv2d_183 (Conv2D)             (None, 14, 14, 256)
__________________________________________________________________________________________________
batch_normalization_183 (BatchN (None, 14, 14, 256)
__________________________________________________________________________________________________
activation_169 (Activation)     (None, 14, 14, 256)
__________________________________________________________________________________________________
conv2d_184 (Conv2D)             (None, 14, 14, 256)
__________________________________________________________________________________________________
batch_normalization_184 (BatchN (None, 14, 14, 256)
__________________________________________________________________________________________________
activation_170 (Activation)     (None, 14, 14, 256)
__________________________________________________________________________________________________
conv2d_185 (Conv2D)             (None, 14, 14, 1024)
__________________________________________________________________________________________________
conv2d_186 (Conv2D)             (None, 14, 14, 1024)
__________________________________________________________________________________________________
batch_normalization_185 (BatchN (None, 14, 14, 1024)
__________________________________________________________________________________________________
batch_normalization_186 (BatchN (None, 14, 14, 1024)
__________________________________________________________________________________________________
add_55 (Add)                    (None, 14, 14, 1024)
__________________________________________________________________________________________________
activation_171 (Activation)     (None, 14, 14, 1024)
__________________________________________________________________________________________________
conv2d_187 (Conv2D)             (None, 14, 14, 256)
__________________________________________________________________________________________________
batch_normalization_187 (BatchN (None, 14, 14, 256)
__________________________________________________________________________________________________
activation_172 (Activation)     (None, 14, 14, 256)
__________________________________________________________________________________________________
conv2d_188 (Conv2D)             (None, 14, 14, 256)
__________________________________________________________________________________________________
batch_normalization_188 (BatchN (None, 14, 14, 256)
__________________________________________________________________________________________________
activation_173 (Activation)     (None, 14, 14, 256)
__________________________________________________________________________________________________
conv2d_189 (Conv2D)             (None, 14, 14, 1024)
__________________________________________________________________________________________________
batch_normalization_189 (BatchN (None, 14, 14, 1024)
__________________________________________________________________________________________________
add_56 (Add)                    (None, 14, 14, 1024)
__________________________________________________________________________________________________
```

```
activation_174 (Activation)     (None, 14, 14, 1024)
_________________________________________________________________
conv2d_190 (Conv2D)             (None, 14, 14, 256)
_________________________________________________________________
batch_normalization_190 (BatchN (None, 14, 14, 256)
_________________________________________________________________
activation_175 (Activation)     (None, 14, 14, 256)
_________________________________________________________________
conv2d_191 (Conv2D)             (None, 14, 14, 256)
_________________________________________________________________
batch_normalization_191 (BatchN (None, 14, 14, 256)
_________________________________________________________________
activation_176 (Activation)     (None, 14, 14, 256)
_________________________________________________________________
conv2d_192 (Conv2D)             (None, 14, 14, 1024)
_________________________________________________________________
batch_normalization_192 (BatchN (None, 14, 14, 1024)
_________________________________________________________________
add_57 (Add)                    (None, 14, 14, 1024)
_________________________________________________________________
activation_177 (Activation)     (None, 14, 14, 1024)
_________________________________________________________________
conv2d_193 (Conv2D)             (None, 14, 14, 256)
_________________________________________________________________
batch_normalization_193 (BatchN (None, 14, 14, 256)
_________________________________________________________________
activation_178 (Activation)     (None, 14, 14, 256)
_________________________________________________________________
conv2d_194 (Conv2D)             (None, 14, 14, 256)
_________________________________________________________________
batch_normalization_194 (BatchN (None, 14, 14, 256)
_________________________________________________________________
activation_179 (Activation)     (None, 14, 14, 256)
_________________________________________________________________
conv2d_195 (Conv2D)             (None, 14, 14, 1024)
_________________________________________________________________
batch_normalization_195 (BatchN (None, 14, 14, 1024)
_________________________________________________________________
add_58 (Add)                    (None, 14, 14, 1024)
_________________________________________________________________
activation_180 (Activation)     (None, 14, 14, 1024)
_________________________________________________________________
conv2d_196 (Conv2D)             (None, 14, 14, 256)
_________________________________________________________________
batch_normalization_196 (BatchN (None, 14, 14, 256)
_________________________________________________________________
activation_181 (Activation)     (None, 14, 14, 256)
_________________________________________________________________
conv2d_197 (Conv2D)             (None, 14, 14, 256)
_________________________________________________________________
batch_normalization_197 (BatchN (None, 14, 14, 256)
_________________________________________________________________
activation_182 (Activation)     (None, 14, 14, 256)
_________________________________________________________________
conv2d_198 (Conv2D)             (None, 14, 14, 1024)
```

```
_____
batch_normalization_198 (BatchN (None, 14, 14, 1024)
_____
add_59 (Add)                    (None, 14, 14, 1024)
_____
activation_183 (Activation)     (None, 14, 14, 1024)
_____
conv2d_199 (Conv2D)             (None, 14, 14, 256)
_____
batch_normalization_199 (BatchN (None, 14, 14, 256)
_____
activation_184 (Activation)     (None, 14, 14, 256)
_____
conv2d_200 (Conv2D)             (None, 14, 14, 256)
_____
batch_normalization_200 (BatchN (None, 14, 14, 256)
_____
activation_185 (Activation)     (None, 14, 14, 256)
_____
conv2d_201 (Conv2D)             (None, 14, 14, 1024)
_____
batch_normalization_201 (BatchN (None, 14, 14, 1024)
_____
add_60 (Add)                    (None, 14, 14, 1024)
_____
activation_186 (Activation)     (None, 14, 14, 1024)
_____
conv2d_202 (Conv2D)             (None, 7, 7, 512)
_____
batch_normalization_202 (BatchN (None, 7, 7, 512)
_____
activation_187 (Activation)     (None, 7, 7, 512)
_____
conv2d_203 (Conv2D)             (None, 7, 7, 512)
_____
batch_normalization_203 (BatchN (None, 7, 7, 512)
_____
activation_188 (Activation)     (None, 7, 7, 512)
_____
conv2d_204 (Conv2D)             (None, 7, 7, 2048)
_____
conv2d_205 (Conv2D)             (None, 7, 7, 2048)
_____
batch_normalization_204 (BatchN (None, 7, 7, 2048)
_____
batch_normalization_205 (BatchN (None, 7, 7, 2048)
_____
add_61 (Add)                    (None, 7, 7, 2048)
_____
activation_189 (Activation)     (None, 7, 7, 2048)
_____
conv2d_206 (Conv2D)             (None, 7, 7, 512)
_____
batch_normalization_206 (BatchN (None, 7, 7, 512)
_____
```

```
activation_190 (Activation)     (None, 7, 7, 512)
_________________________________________________________________
conv2d_207 (Conv2D)             (None, 7, 7, 512)
_________________________________________________________________
batch_normalization_207 (BatchN (None, 7, 7, 512)
_________________________________________________________________
activation_191 (Activation)     (None, 7, 7, 512)
_________________________________________________________________
conv2d_208 (Conv2D)             (None, 7, 7, 2048)
_________________________________________________________________
batch_normalization_208 (BatchN (None, 7, 7, 2048)
_________________________________________________________________
add_62 (Add)                    (None, 7, 7, 2048)
_________________________________________________________________
activation_192 (Activation)     (None, 7, 7, 2048)
_________________________________________________________________
conv2d_209 (Conv2D)             (None, 7, 7, 512)
_________________________________________________________________
batch_normalization_209 (BatchN (None, 7, 7, 512)
_________________________________________________________________
activation_193 (Activation)     (None, 7, 7, 512)
_________________________________________________________________
conv2d_210 (Conv2D)             (None, 7, 7, 512)
_________________________________________________________________
batch_normalization_210 (BatchN (None, 7, 7, 512)
_________________________________________________________________
activation_194 (Activation)     (None, 7, 7, 512)
_________________________________________________________________
conv2d_211 (Conv2D)             (None, 7, 7, 2048)
_________________________________________________________________
batch_normalization_211 (BatchN (None, 7, 7, 2048)
_________________________________________________________________
add_63 (Add)                    (None, 7, 7, 2048)
_________________________________________________________________
activation_195 (Activation)     (None, 7, 7, 2048)
_________________________________________________________________
average_pooling2d_3 (AveragePoo (None, 4, 4, 2048)
_________________________________________________________________
flatten_3 (Flatten)             (None, 32768)
_________________________________________________________________
dense_3 (Dense)                 (None, 1)
=================================================================
Total params: 23,620,481
Trainable params: 23,567,361
Non-trainable params: 53,120
```

Listing 7.10: ResNet 50 model definition

7.7 SpineNet

```
MODEL_DIR="~/models/spinenet49"
```

```
TRAIN_FILE_PATTERN="~/masterthesis/train/training.tfrecord"
EVAL_FILE_PATTERN="~/masterthesis/eval/eval.tfrecord"
VAL_JSON_FILE="~/masterthesis/val.json"
python3 ~/models/official/vision/detection/main.py \
    --strategy_type=tpu \
    --model_dir="${MODEL_DIR?}" \
    --mode=train \
    --model=spinenet49 \
    --params_override="{architecture: {backbone: spinenet, ↩
   multilevel_features: identity}, spinenet: {model_id: 49}, ↩
   train_file_pattern: ${TRAIN_FILE_PATTERN?} }, eval: { val_json_file:↩
    ${VAL_JSON_FILE?}, eval_file_pattern: ${EVAL_FILE_PATTERN?} } }"
```

Listing 7.11: SpineNet49

7.8 ShapeMask

```
train:
    train_file_pattern: ~/masterthesis/train/training.tfrecord
    total_steps: 8000
    batch_size: 128
eval:
    eval_file_pattern: ~/masterthesis/eval/eval.tfrecord
    val_json_file: ~/masterthesis/val.json
    batch_size: 64
shapemask_head:
    shape_prior_path: ~/masterthesis/shape/shape.tfrecord"
```

Listing 7.12: shapemask

7.9 Azure custom vision

The azure custom vision ai provides an easy user interface. The learning methods did work out well. Around 3/4 of all images where classified correctly using a 24 hour learning budget and 7 hour learning budget. An alternativ was trained on a 7 hour budget 7.4.

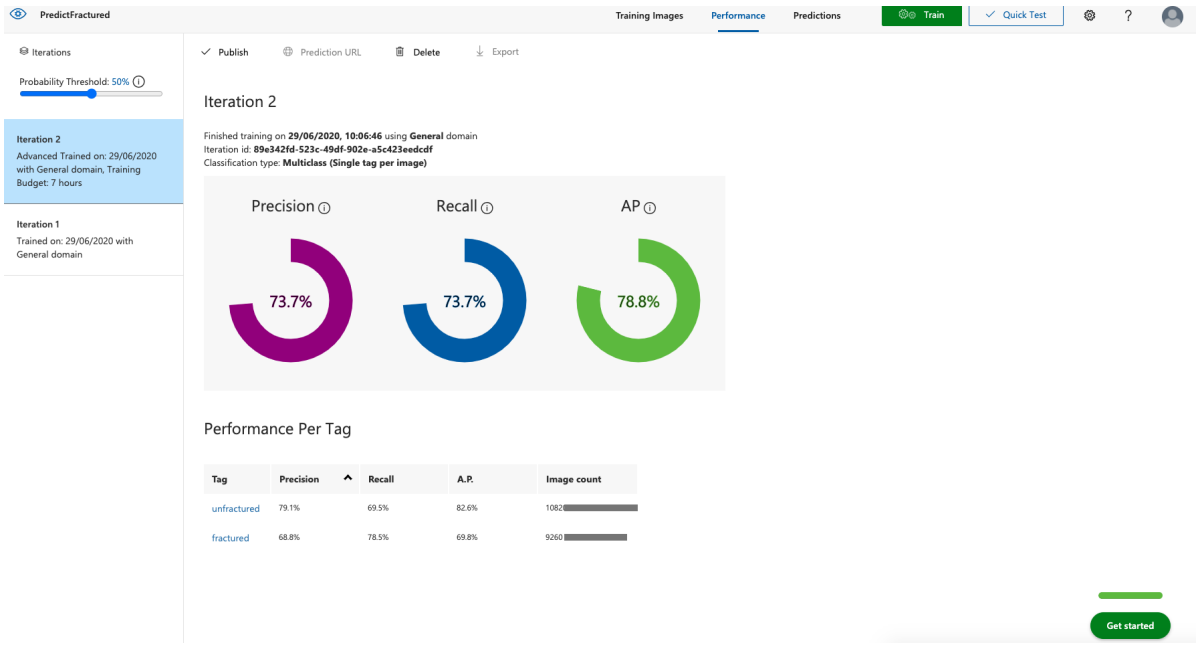


Figure 7.4: Azure custom vision ai iteration using 7 hour budget.

[type=acronym]

List of Figures

List of Tables

Listings

References

Abdulla, W. (2017). *Mask r-cnn for object detection and instance segmentation on keras and tensorflow.* `https://github.com/matterport/Mask_RCNN`. Github.

Andersson, G., & Watkins-Castillo, S. (2014). The burden of musculoskeletal diseases in the united states (bmus). *United States Bone and Joint Initiative, Rosemont, IL*.

Bandyopadhyay, O., Chanda, B., & Bhattacharya, B. B. (2011). Entropy-based automatic segmentation of bones in digital x-ray images. In *International conference on pattern recognition and machine intelligence* (pp. 122–129).

Banga, D., & Waiganjo, P. (2019). Abnormality detection in musculoskeletal radiographs with convolutional neural networks (ensembles) and performance optimization. *arXiv preprint arXiv:1908.02170*.

Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I., Bergeron, A., ... Bengio, Y. (2012). Theano: new features and speed improvements. *arXiv preprint arXiv:1211.5590*.

Bengio, Y., Simard, P., Frasconi, P., et al. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, *5*(2), 157–166.

Berst, M. J., Dolan, L., Bogdanowicz, M. M., Stevens, M. A., Chow, S., & Brandser, E. A. (2001). Effect of knowledge of chronologic age on the variability of pediatric bone age determined using the greulich and pyle standards. *American Journal of Roentgenology*, *176*(2), 507–510.

Bomer, J., Wiersma-Deijl, L., & Holscher, H. C. (2013, Oct). Electronic collimation and radiation protection in paediatric digital radiography: revival of the silver lining. *Insights Imaging*, *4*(5), 723–727. doi: 10.1007/s13244-013-0281-5

Briggs, A. M., Cross, M. J., Hoy, D. G., Sanchez-Riera, L., Blyth, F. M., Woolf, A. D., & March, L. (2016). Musculoskeletal health conditions represent a global threat to healthy aging: a report for the 2015 world health organization world report on ageing and health. *The Gerontologist*, *56*(suppl_2), S243–S255.

Brosch, T., Yoo, Y., Tang, L. Y., Li, D. K., Traboulsee, A., & Tam, R. (2015). Deep convolutional encoder networks for multiple sclerosis lesion segmentation. In *International conference on medical image computing and computer-assisted intervention* (pp. 3–11).

Byvatov, E., Fechner, U., Sadowski, J., & Schneider, G. (2003). Comparison of support vector machine and artificial neural network systems for drug/nondrug classification. *Journal of chemical information and computer sciences*, *43*(6), 1882–1889.

Cepela, D. J., Tartaglione, J. P., Dooley, T. P., & Patel, P. N. (2016, Nov 01). Classifications in brief: Salter-harris classification of pediatric physeal fractures.

Clinical Orthopaedics and Related Research®, *474*(11), 2531–2537. Retrieved from https://doi.org/10.1007/s11999-016-4891-3 doi: 10.1007/s11999-016-4891-3

Cernazanu-Glavan, C., & Holban, S. (2013). Segmentation of bone structure in x-ray images using convolutional neural network. *Adv. Electr. Comput. Eng*, *13*(1), 87–94.

Chen, C., Du, X., Hou, L., Kim, J., Jin, P., Li, J., ... Yu, H. (2020). *Tensorflow official model garden.* Retrieved from https://github.com/tensorflow/models/tree/master/official

Chen, J., Chen, J., Ding, H.-Y., Pan, Q.-S., Hong, W.-D., Xu, G., ... Wang, Y.-M. (2015). Use of an artificial neural network to construct a model of predicting deep fungal infection in lung cancer patients. *Asian Pac J Cancer Prev*, *16*(12), 5095–5099.

Cherian, A., Fernando, B., Harandi, M., & Gould, S. (2017). Generalized rank pooling for activity recognition. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 3222–3231).

Cherian, A., Koniusz, P., & Gould, S. (2017). Higher-order pooling of cnn features via kernel linearization for action recognition. In *2017 ieee winter conference on applications of computer vision (wacv)* (pp. 130–138).

Ciresan, D., Giusti, A., Gambardella, L. M., & Schmidhuber, J. (2012). Deep neural networks segment neuronal membranes in electron microscopy images. In *Advances in neural information processing systems* (pp. 2843–2851).

Cireşan, D. C., Giusti, A., Gambardella, L. M., & Schmidhuber, J. (2013). Mitosis detection in breast cancer histology images with deep neural networks. In *International conference on medical image computing and computer-assisted intervention* (pp. 411–418).

Cummins, F. (1999, January). Learning to forget: continual prediction with lstm. *IET Conference Proceedings*, 850-855(5). Retrieved from https://digital-library.theiet.org/content/conferences/10.1049/cp_19991218

Del Sole, A. (2018). Introducing microsoft cognitive services. In *Microsoft computer vision apis distilled* (pp. 1–4). Springer.

Deng, L., Hinton, G., & Kingsbury, B. (2013). New types of deep neural network learning for speech recognition and related applications: An overview. In *2013 ieee international conference on acoustics, speech and signal processing* (pp. 8599–8603).

de Vos, B. D., Wolterink, J. M., de Jong, P. A., Viergever, M. A., & Išgum, I. (2016). 2d image classification for 3d anatomy localization: employing deep convolutional neural networks. In *Medical imaging 2016: Image processing* (Vol. 9784, p. 97841Y).

Dollár, P., Welinder, P., & Perona, P. (2010). Cascaded pose regression. In *2010 ieee computer society conference on computer vision and pattern recognition* (pp. 1078–1085).

Drucker, H., Burges, C. J., Kaufman, L., Smola, A. J., & Vapnik, V. (1997). Support vector regression machines. In *Advances in neural information processing systems* (pp. 155–161).

Druzhkov, P., & Kustikova, V. (2016). A survey of deep learning methods and soft-

ware tools for image classification and object detection. *Pattern Recognition and Image Analysis*, *26*(1), 9–15.

Du, X., Lin, T.-Y., Jin, P., Ghiasi, G., Tan, M., Cui, Y., ... Song, X. (2019). *Spinenet: Learning scale-permuted backbone for recognition and localization.*

Dutta, P., & Dutta, P. (2019). Comparative study of cloud services offered by amazon, microsoft & google. *International Journal of Trend in Scientific Research and Development (ijtsrd)*, *3*, 981–985.

Dwek, J. R. (2010). The periosteum: what is it, where is it, and what mimics it in its absence? *Skeletal radiology*, *39*(4), 319–323.

Erickson, B. J., Korfiatis, P., Akkus, Z., Kline, T., & Philbrick, K. (2017). Toolkits and libraries for deep learning [Journal Article]. *J Digit Imaging*, *30*(4), 400-405. Retrieved from https://www.ncbi.nlm.nih.gov/pubmed/28315069 doi: 10.1007/s10278-017-9965-6

Farabet, C., Couprie, C., Najman, L., & LeCun, Y. (2012). Learning hierarchical features for scene labeling. *IEEE transactions on pattern analysis and machine intelligence*, *35*(8), 1915–1929.

Fenton, J. J., Taplin, S. H., Carney, P. A., Abraham, L., Sickles, E. A., D'Orsi, C., ... others (2007). Influence of computer-aided detection on performance of screening mammography. *New England Journal of Medicine*, *356*(14), 1399–1409.

Fitzgerald, R. (2001, 2019/08/11). Error in radiology. *Clinical Radiology*, *56*(12), 938–946. Retrieved from https://doi.org/10.1053/crad.2001.0858 doi: 10.1053/crad.2001.0858

Frost, H. M., & Schönau, E. (2000). The" muscle-bone unit" in children and adolescents: a 2000 overview. *Journal of pediatric endocrinology and metabolism*, *13*(6), 571–590.

Ganea, E., Burdescu, D. D., & Brezovan, M. (2011). New method to detect salient objects in image segmentation using hypergraph structure. *Advances in Electrical and Computer Engineering*, *11*(4), 111–116.

George, M. P., & Bixby, S. (2019). Frequently missed fractures in pediatric trauma: A pictorial review of plain film radiography. *Radiologic Clinics of North America*, *57*(4), 843 - 855. Retrieved from http://www.sciencedirect.com/science/article/pii/S0033838919300260 (Trauma and Emergency Radiology) doi: https://doi.org/10.1016/j.rcl.2019.02.009

Gopinath, M. P., Aarthy, S. L., Manchanda, A., & Rishabh. (2019). Machine learning on medical dataset. In S. C. Satapathy, V. Bhateja, R. Somanah, X.-S. Yang, & R. Senkerik (Eds.), *Information systems design and intelligent applications* (pp. 133–143). Singapore: Springer Singapore.

Gordienko, Y., Gang, P., Hui, J., Zeng, W., Kochura, Y., Alienin, O., ... Stirenko, S. (2018). Deep learning with lung segmentation and bone shadow exclusion techniques for chest x-ray analysis of lung cancer. In *International conference on computer science, engineering and education applications* (pp. 638–647).

Haykin, S. (1994). *Neural networks: a comprehensive foundation*. Prentice Hall PTR.

He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). *Mask r-cnn.*

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the ieee conference on computer vision and pattern recognition*

(pp. 770–778).

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

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Hržić, F., Štajduhar, I., Tschauner, S., Sorantin, E., & Lerga, J. (2019). Local-entropy based approach for x-ray image segmentation and fracture detection. *Entropy*, *21*(4). Retrieved from `https://www.mdpi.com/1099-4300/21/4/338` doi: 10.3390/e21040338

Hua, K.-L., Hsu, C.-H., Hidayati, S. C., Cheng, W.-H., & Chen, Y.-J. (2015). Computer-aided classification of lung nodules on computed tomography images via deep learning technique. *OncoTargets and therapy*, *8*.

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

Hubel, D. H., & Wiesel, T. N. (1968). Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, *195*(1), 215–243.

Huynh, M.-C., Nguyen, T.-H., & Tran, M.-T. (2018). Context learning for bone shadow exclusion in chexnet accuracy improvement. In *2018 10th international conference on knowledge and systems engineering (kse)* (pp. 135–140).

Jaremko, J. L., Azar, M., Bromwich, R., Lum, A., Alicia Cheong, L. H., Gibert, M., ... Tang, A. (2019, May). Canadian association of radiologists white paper on ethical and legal issues related to artificial intelligence in radiology. *Can Assoc Radiol J*, *70*(2), 107–118. doi: 10.1016/j.carj.2019.03.001

Jarraya, M., Hayashi, D., Roemer, F. W., Crema, M. D., Diaz, L., Conlin, J., ... Guermazi, A. (2013). Radiographically occult and subtle fractures: a pictorial review. *Radiol Res Pract*, *2013*, 370169. doi: 10.1155/2013/370169

Kallenberg, M., Petersen, K., Nielsen, M., Ng, A. Y., Diao, P., Igel, C., ... others (2016). Unsupervised deep learning applied to breast density segmentation and mammographic risk scoring. *IEEE transactions on medical imaging*, *35*(5), 1322–1331.

Kang, K., & Wang, X. (2014). Fully convolutional neural networks for crowd segmentation. *arXiv preprint arXiv:1411.4464*.

Karpathy, A., & Fei-Fei, L. (2015). Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 3128–3137).

Ketkar, N., et al. (2017). *Deep learning with python*. Springer.

Khan, I., Dewangan, B., Meena, A., & Birthare, M. (2020). Study of various cloud service providers: A comparative analysis. In *5th international conference on next generation computing technologies (ngct-2019)*.

Kiwiel, K. C. (2001). Convergence and efficiency of subgradient methods for quasiconvex minimization. *Mathematical programming*, *90*(1), 1–25.

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

Kulkarni, G., Premraj, V., Ordonez, V., Dhar, S., Li, S., Choi, Y., ... Berg, T. L.

(2013). Babytalk: Understanding and generating simple image descriptions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *35*(12), 2891–2903.

Kumar, D., Wong, A., & Clausi, D. A. (2015). Lung nodule classification using deep features in ct images. In *2015 12th conference on computer and robot vision* (pp. 133–138).

Kuo, W., Angelova, A., Malik, J., & Lin, T.-Y. (2019). *Shapemask: Learning to segment novel objects by refining shape priors.*

Larson, D. B., Chen, M. C., Lungren, M. P., Halabi, S. S., Stence, N. V., & Langlotz, C. P. (2017). Performance of a deep-learning neural network model in assessing skeletal maturity on pediatric hand radiographs. *Radiology*, *287*(1), 313–322.

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

LeCun, Y. A., Bottou, L., Orr, G. B., & Müller, K.-R. (2012). Efficient backprop. In *Neural networks: Tricks of the trade* (pp. 9–48). Springer.

Lee, J. G., Jun, S., Cho, Y. W., Lee, H., Kim, G. B., Seo, J. B., & Kim, N. (2017). Deep learning in medical imaging: General overview [Journal Article]. *Korean J Radiol*, *18*(4), 570-584. Retrieved from `https://www.ncbi.nlm.nih.gov/pubmed/28670152` doi: 10.3348/kjr.2017.18.4.570

Lehman, C. D., Wellman, R. D., Buist, D. S., Kerlikowske, K., Tosteson, A. N., & Miglioretti, D. L. (2015). Diagnostic accuracy of digital screening mammography with and without computer-aided detection. *JAMA internal medicine*, *175*(11), 1828–1837.

Lippert, W. C., Owens, R. F., & Wall, E. J. (2010). Salter-harris type iii fractures of the distal femur: plain radiographs can be deceptive. *Journal of Pediatric Orthopaedics*, *30*(6), 598–605.

Litjens, G., Kooi, T., Bejnordi, B. E., Setio, A. A. A., Ciompi, F., Ghafoorian, M., ... Sanchez, C. I. (2017). A survey on deep learning in medical image analysis [Journal Article]. *Med Image Anal*, *42*, 60-88. Retrieved from `https://www.ncbi.nlm.nih.gov/pubmed/28778026https://www.medicalimageanalysisjournal.com/article/S1361-8415(17)30113-5/pdf` doi: 10.1016/j.media.2017.07.005

Little, J. T., Klionsky, N. B., Chaturvedi, A., Soral, A., & Chaturvedi, A. (2014, Mar-Apr). Pediatric distal forearm and wrist injury: an imaging review. *Radiographics*, *34*(2), 472–490. doi: 10.1148/rg.342135073

Liu, S., Liu, S., Cai, W., Pujol, S., Kikinis, R., & Feng, D. (2014, April). Early diagnosis of alzheimer's disease with deep learning. In *2014 ieee 11th international symposium on biomedical imaging (isbi)* (p. 1015-1018). doi: 10.1109/ISBI.2014.6868045

Miao, S., Wang, Z. J., & Liao, R. (2016). A cnn regression approach for real-time 2d/3d registration. *IEEE transactions on medical imaging*, *35*(5), 1352–1363.

Middleton, I., & Damper, R. I. (2004). Segmentation of magnetic resonance images using a combination of neural networks and active contour models. *Medical engineering & physics*, *26*(1), 71–86.

Milletari, F., Navab, N., & Ahmadi, S. (2016). V-net: Fully convolutional neural networks for volumetric medical image segmentation. *CoRR*, *abs/1606.04797*. Retrieved from `http://arxiv.org/abs/1606.04797`

Mizuta, T., Benson, W., Foster, B., Paterson, D., & Morris, L. (1987). Statistical analysis of the incidence of physeal injuries. *Journal of pediatric orthopedics*, *7*(5), 518–523.

Moeskops, P., Viergever, M. A., Mendrik, A. M., de Vries, L. S., Benders, M. J., & Išgum, I. (2016). Automatic segmentation of mr brain images with a convolutional neural network. *IEEE transactions on medical imaging*, *35*(5), 1252–1261.

Moore, M. M., Slonimsky, E., Long, A. D., Sze, R. W., & Iyer, R. S. (2019, Apr). Machine learning concepts, concerns and opportunities for a pediatric radiologist. *Pediatr Radiol*, *49*(4), 509–516. doi: 10.1007/s00247-018-4277-7

Mottaghi, R., Xiang, Y., & Savarese, S. (2015). A coarse-to-fine model for 3d pose estimation and sub-category recognition. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 418–426).

Mounts, J., Clingenpeel, J., McGuire, E., Byers, E., & Kireeva, Y. (2011). Most frequently missed fractures in the emergency department [Journal Article]. *Clin Pediatr (Phila)*, *50*(3), 183-6. Retrieved from `https://www.ncbi.nlm.nih.gov/pubmed/21127081` doi: 10.1177/0009922810384725

Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press.

Nakata, N. (2019, Feb). Recent technical development of artificial intelligence for diagnostic medical imaging. *Jpn J Radiol*, *37*(2), 103–108. doi: 10.1007/s11604-018-0804-6

Pawlowski, N., Ktena, S. I., Lee, M. C., Kainz, B., Rueckert, D., Glocker, B., & Rajchl, M. (2017). Dltk: State of the art reference implementations for deep learning on medical images. *arXiv preprint arXiv:1711.06853*.

Pereira, S., Pinto, A., Alves, V., & Silva, C. A. (2016). Brain tumor segmentation using convolutional neural networks in mri images. *IEEE transactions on medical imaging*, *35*(5), 1240–1251.

Peterson, H. A., & Burkhart, S. S. (1981). Compression injury of the epiphyseal growth plate: fact or fiction? *Journal of pediatric orthopedics*, *1*(4), 377–384.

Petit, P., Panuel, M., Faure, F., Jouve, J., Bourliere-Najean, B., Bollini, G., & Devred, P. (1996). Acute fracture of the distal tibial physis: role of gradient-echo mr imaging versus plain film examination. *AJR. American journal of roentgenology*, *166*(5), 1203–1206.

Pineda, F. J. (1987). Generalization of back-propagation to recurrent neural networks. *Physical review letters*, *59*(19), 2229.

Prasoon, A., Petersen, K., Igel, C., Lauze, F., Dam, E., & Nielsen, M. (2013). Deep feature learning for knee cartilage segmentation using a triplanar convolutional neural network. In *International conference on medical image computing and computer-assisted intervention* (pp. 246–253).

Rajpurkar, P., Irvin, J., Bagul, A., Ding, D., Duan, T., Mehta, H., ... others (2017). Mura: Large dataset for abnormality detection in musculoskeletal radiographs. *arXiv preprint arXiv:1712.06957*.

Rajpurkar, P., Irvin, J., Zhu, K., Yang, B., Mehta, H., Duan, T., ... others (2017). Radiologist-level pneumonia detection on chest x-rays with deep learning. *arXiv preprint arXiv:1711.05225*.

Rathjen, K. E., & Kim, H. K. (2014). Physeal injuries and growth disturbances. In *Rockwood, green, and wilkins fractures in adults and children: Eighth edition.* Wolters

Kluwer Health Adis (ESP).

Regulation, P. (2016). Regulation (eu) 2016/679 of the european parliament and of the council. *REGULATION (EU)*, *679*, 2016.

Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems* (pp. 91–99).

Roehrborn, C. G., Boyle, P., Bergner, D., Gray, T., Gittelman, M., Shown, T., ... others (1999). Serum prostate-specific antigen and prostate volume predict long-term changes in symptoms and flow rate: results of a four-year, randomized trial comparing finasteride versus placebo. *Urology*, *54*(4), 662–669.

Rogers, L. F. (1970). The radiography of epiphyseal injuries. *Radiology*, *96*(2), 289–299.

Rogers, L. F., & Poznanski, A. K. (1994, May). Imaging of epiphyseal injuries. *Radiology*, *191*(2), 297–308. doi: 10.1148/radiology.191.2.8153295

Ronneberger, O., Fischer, P., & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *International conference on medical image computing and computer-assisted intervention* (pp. 234–241).

Salter, R. B., & Harris, W. R. (1963). Injuries involving the epiphyseal plate. *JBJS*, *45*(3), 587–622.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). *Mobilenetv2: Inverted residuals and linear bottlenecks.*

Shaikh, K. (n.d.). Demystifying azure ai.

Shen, W., Zhou, M., Yang, F., Yang, C., & Tian, J. (2015). Multi-scale convolutional neural networks for lung nodule classification. In *International conference on information processing in medical imaging* (pp. 588–599).

Shin, H.-C., Roberts, K., Lu, L., Demner-Fushman, D., Yao, J., & Summers, R. M. (2016). Learning to read chest x-rays: Recurrent neural cascade model for automated image annotation. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 2497–2506).

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Smith, B. G., Rand, F., Jaramillo, D., & Shapiro, F. (1994). Early mr imaging of lower-extremity physeal fracture-separations: a preliminary report. *Journal of pediatric orthopedics*, *14*(4), 526–533.

Soh, J., Copeland, M., Puca, A., & Harris, M. (2020). Ethical ai, azure ai, and machine learning. In *Microsoft azure: Planning, deploying, and managing the cloud* (pp. 67–84). Berkeley, CA: Apress. Retrieved from `https://doi.org/10.1007/978-1-4842-5958-0_5` doi: 10.1007/978-1-4842-5958-0_5

Soundappan, S. V. S., Holland, A. J. A., & Cass, D. T. (2004, Jul). Role of an extended tertiary survey in detecting missed injuries in children. *J Trauma*, *57*(1), 114–118. doi: 10.1097/01.ta.0000108992.51091.f7

Subbanna, N., Precup, D., & Arbel, T. (2014). Iterative multilevel mrf leveraging context and voxel information for brain tumour segmentation in mri. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 400–405).

Suk, H.-I., Lee, S.-W., Shen, D., Initiative, A. D. N., et al. (2014). Hierarchical feature representation and multimodal fusion with deep learning for ad/mci diagnosis.

NeuroImage, *101*, 569–582.

Suk, H.-I., & Shen, D. (2013). Deep learning-based feature representation for ad/mci classification. In *International conference on medical image computing and computer-assisted intervention* (pp. 583–590).

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 1–9).

Tay, F. E., & Cao, L. (2001). Application of support vector machines in financial time series forecasting. *omega*, *29*(4), 309–317.

Teramoto, A., Fujita, H., Yamamuro, O., & Tamaki, T. (2016). Automated detection of pulmonary nodules in pet/ct images: Ensemble false-positive reduction using a convolutional neural network technique. *Medical physics*, *43*(6Part1), 2821–2827.

Tolias, G., Sicre, R., & Jégou, H. (2015). Particular object retrieval with integral max-pooling of cnn activations. *arXiv preprint arXiv:1511.05879*.

Van Der Walt, S., Colbert, S. C., & Varoquaux, G. (2011). The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, *13*(2), 22.

Van Grinsven, M. J., van Ginneken, B., Hoyng, C. B., Theelen, T., & Sánchez, C. I. (2016). Fast convolutional neural network training using selective data sampling: Application to hemorrhage detection in color fundus images. *IEEE transactions on medical imaging*, *35*(5), 1273–1284.

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., & Manzagol, P.-A. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, *11*(Dec), 3371–3408.

Wang, D., Khosla, A., Gargeya, R., Irshad, H., & Beck, A. H. (2016). Deep learning for identifying metastatic breast cancer. *arXiv preprint arXiv:1606.05718*.

Wang, Q., Garrity, G. M., Tiedje, J. M., & Cole, J. R. (2007). Naive bayesian classifier for rapid assignment of rrna sequences into the new bacterial taxonomy. *Appl. Environ. Microbiol.*, *73*(16), 5261–5267.

Wattenbarger, J. M., Gruber, H. E., & Phieffer, L. S. (2002). Physeal fractures, part i: histologic features of bone, cartilage, and bar formation in a small animal model. *Journal of Pediatric Orthopaedics*, *22*(6), 703–709.

Werbos, P. J. (1988). Generalization of backpropagation with application to a recurrent gas market model. *Neural networks*, *1*(4), 339–356.

Williams, R. J., & Zipser, D. (1995). Gradient-based learning algorithms for recurrent. *Backpropagation: Theory, architectures, and applications*, *433*.

Wohlhart, P., & Lepetit, V. (2015). Learning descriptors for object recognition and 3d pose estimation. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 3109–3118).

Woolf, A. D., & Pfleger, B. (2003). Burden of major musculoskeletal conditions. *Bulletin of the World Health Organization*, *81*, 646–656.

Xu, Y., Mo, T., Feng, Q., Zhong, P., Lai, M., Eric, I., & Chang, C. (2014). Deep learning of feature representation with multiple instance learning for medical image analysis. In *2014 ieee international conference on acoustics, speech and signal processing (icassp)* (pp. 1626–1630).

Yang, D., Zhang, S., Yan, Z., Tan, C., Li, K., & Metaxas, D. (2015). Automated anatomical landmark detection ondistal femur surface using convolutional neural network. In *2015 ieee 12th international symposium on biomedical imaging (isbi)* (pp. 17–21).

Yu, P.-S., Chen, S.-T., & Chang, I.-F. (2006). Support vector regression for real-time flood stage forecasting. *Journal of Hydrology*, *328*(3-4), 704–716.

Zach, C., Penate-Sanchez, A., & Pham, M.-T. (2015). A dynamic programming approach for fast and robust object pose recognition from range images. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 196–203).

Zeiler, M. D., Taylor, G. W., Fergus, R., et al. (2011). Adaptive deconvolutional networks for mid and high level feature learning. In *Iccv* (Vol. 1, p. 6).