UNIVERSITATEA DIN BUCUREȘTI
FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

LUCRARE DE LICENȚĂ

Coordonator:
Prof. Dr. Ionescu Radu Tudor

Absolvent:
Vultur Cristina

București
Iunie 2021

UNIVERSITATEA DIN BUCUREȘTI
FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

2D vs 3D Convolutional Neural Networks in Hemorrhage Detection

Coordonator:
Prof. Dr. Ionescu Radu Tudor

Absolvent:
Vultur Cristina

București
Iunie 2021

Rezumat

Tomografia computerizată este o tehnică de imagistică medicala ce este utilizată pentru a evalua severitatea unei hemoragii cerebrale. Interpretarea ei este consumatoare de timp și nu poate fi realizată ușor chiar și de experți foarte bine pregătiți. Învățarea profundă a arătat un potențial imens de a extrage informații valoroase din imagistica medicală. Cu toate acestea, cantitatea de date de antrenare necesare pentru a construi proiecte de învățare profundă în imagistica medicală 3D are un impact substanțial asupra performanței modelului, precum și resursele computaționale necesare.

În această lucrare prezentăm două arhitecturi diferite de rețele neuronale convoluționale de mică adâncime: una ResNet-18 2D și una 3D, pe care le-am folosit în sarcina de a detecta hemoragia și subtipurile sale pe baza setului de date RSNA 2019 Brain CT Hemorrhage. Scopul nostru a fost de a experimenta compensarea între puterea de calcul și nevoia de cantități mari de date, așa că am antrenat rețeaua noastră 2D cu mai multe date decât cea mai costisitoare din punct de vedere al resurselor de calcul, 3D. Am plecat cu premisa că rețeaua 3D va funcționa mai bine datorită anumitor avantaje ale sale, chiar dacă a primit mult mai puține date de intrare.

Prima rețea a primit ca intrare secțiuni individuale din imaginea volumetrică care au fost preprocesate eficient prin extragerea ferestrelor de intensitate diferită și alipirea lor ca imagine asemănătoare RGB. De asemenea, am redus dimensiunea secțiunilor obținute pentru ca modelul nostru să se antreneze mai repede. Cu această abordare se obține o pierdere medie de **0,0824** pe setul nostru de validare. După terminarea testelor noastre cu această rețea, am estimat că primirea scanărilor întregi ca intrări a modelului va îmbunătăți și mai mult performanța.

Apoi, pentru a doua rețea, am reconstruit scanările la scara lor inițială și am antrenat rețeaua cu un subset din acestea. Rețeaua noastră convoluțională 3D a primit de patru ori mai puține date decât cea 2D și a obținut o pierdere de a **0.4947** pe setul nostru de date de validare. Modelul a avut o performanță scăzută, contrar așteptărilor, așa că am încercat să-l îmbunătățim dând doar cinci secțiuni consecutive ca intrare. Obținem o pierdere de **0,2031** cu această abordare, măsurătorile alese îmbunătățindu-se semnificativ.

La final încheiem prin analizarea modelelor și facem recomandări pentru dezvoltări viitoare.

Abstract

Computed tomography is an imaging technique that is used to evaluate the severity of a brain hemorrhage. It is very time-consuming and cannot be easily interpreted even by highly trained experts. Deep learning has shown immense potential to extract valuable information from medical imaging. However, the amount of training data necessary to construct deep learning projects in 3D medical imaging has a substantial impact on model performance, as well as the computational resources needed.

In this thesis, we propose two different shallow Convolutional Neural Network (CNN) architectures: a 2D and a 3D ResNet-18 that we used in the task of detecting hemorrhages and their subtypes. We trained our network on the dataset presented by the "*RSNA 2019 Brain CT Hemorrhage*" competition. Our goal was to experiment with the compensation between computational power and needing large amounts of data, so we fed our 2D Network more data than our more computational expensive one, the 3D. We left with the premise that the 3D will perform better due to its certain advantages, even though it was fed significantly fewer data.

The first network received as input individual CT slices, that were efficiently preprocessed by extracting different intensity windows and stacking them as an RGB-like image. We also reduced the size of the CT slices for our model to train faster. With this slice-by-slice approach, we obtain a mean log loss of 0.0824 on our validation set. After finishing our tests with this network,

we considered that having as input whole CT slices is likely going to improve performance.

Then, for the second network, we reconstructed the scans at their original scale and fed a subset of them to the network. Our 3DCNN network received four times fewer data than our 2D one and obtain a loss of **0.4947** on our validation dataset. The model performed poorly, which was unexpected, so we tried to improve it by giving only five consecutive slices as input. We obtain a **0.2031** loss with this approach, with significant improvements to the metrics chosen.

In the end, we conclude by analyzing the models and make recommendations for future improvements.

Content

List of tables and figures

Tables:

Figures:

Introduction

Computed tomography (CT) is a worldwide used method to examine areas affected inside the human body, being the number one imaging technique that is designed to assess brain hemorrhage and the severity of it in case of traumatic injury. For hemorrhage to be diagnosed experts have to go through and interpret tens of scans. This can easily lead to missing life-threatening details. The diagnosis given by experts is also extremely time-critical because the delay of it can cause the patient to lose his life or have traumatic damage. Traditionally, experts visualize tens of slices for abnormalities which can be extremely time-consuming and extends the possibility to miss certain critical details. In the emergency department it is reported that the CT health examinations can last up to 1.5-4 hours causing a delay in diagnosis, which can often result in hemorrhage expansion. With that being said, an accurate and fast diagnosis for the Intracranial hemorrhage is needed.

Deep learning has shown immense potential to extract valuable information from medical imaging data. Its ability to perform complex cognitive tasks with fast interference has led it to gain popularity for the use of medical analysis in recent years.

Even if the diagnosis could be done extremely fast using deep learning there are still some major problems that these automated diagnoses have. Firstly, one of the top common challenges in artificial intelligence is computing power. Machine learning algorithms demand even increasing numbers of power to work efficiently. Secondly, the need for large amounts of data that is hard to obtain is another problem in the field of medical imaging.

Convolutional neural networks are the most favored when it comes to visual tasks and extremely used when it comes to medical images. When you hear about a 3D image, naturally you think of a 3D convolutional neural network. The problem with them is that they can be extremely time-consuming when it comes to their training and fine-tuning. To search the optimal parameters, you will need to train it multiple hours which is not efficient at all.

In this thesis, we wanted to experiment with the compensation between computational power and needing large amounts of data. Firstly, we wanted to compare the 2D approach with the 3D one. Our goal is to see if the computational resources needed for the latter are worth it or there can be a similar solution or better given by a 2D neural network that receives more data. Secondly, if

considering the hole 3D Scan, pretrained on medical imaging, that receives significantly less data, can compensate the problem of needing large amounts of data. We will be working with the same dataset and the same shallow architecture for both of our networks to analyze the accuracy given and the differences. In this thesis, we will feed our 2D model mini-batches of different intensity windows from the same slice, following a slice-by-slice approach that does not consider neighboring slices. For our 3D model, we follow a scan approach, feeding our model with single whole scans.

The thesis is structured as follows: Chapter 1 presents the Theoretical Concepts needed to understand this thesis, Chapter 2 presents Recent approaches found in scientific literature, Chapter 3 contains the description of all the technologies used in developing our project, Chapter 4 presents our approach on the task at hand, Chapter 5 discusses the experiments we made and results we obtained after them, and Chapter 6 contains the detailed description for our whole program: Labeling the data, Preprocessing and Implementation details. The thesis closes with conclusions and further development.

Chapter 1 Theoretical concepts Convolutional Neural Networks

Machine learning is a subtype of artificial intelligence that has brought major contributions to the world, especially over the last decade. However, as any revolutionizing tool, it comes with limitations. An interesting example is the chess game between Garry Kasparov, the famous champion, and IBM's Deep Blue supercomputer that took place in 1996. The machine successfully won against the Russian grandmaster, but it was until recently that computers were unable to perform certain fundamental tasks for the human brain, including the identification of an object in an image. [1]

A problem that comes with using fully connected networks on image tasks is that even with a slight change, for example, a few pixels shift to the left, the output changes. Along with the network, every intermediate layer will change, which means the prediction will slightly change as well. There are going to be variations for the same image, but we do not want to have intermediary representations that differ for translations because is not relevant for image applications. The representation is inefficient, the middle layers have to learn the same thing even for an image that is translated. Even though the input is different it should be capable of processing it the same. This requires hard optimization and is inefficient.

A convolutional neural network(CNN) is an artificial neural network(ANN) that has had significant impacts in various fields related to pattern and image recognition. More intuitively, we can consider it a network that has a skill in picking up or identifying patterns and finding meaning in them. CNN's ability to recognize patterns is what makes it so valuable for image analysis. They make use of certain properties from the data because they have inductive biases implicit in their architecture. For visual data, CNN is more efficient than a fully connected network due to its translation equivariance.

1.1 The architecture of the visual cortex

The CNN's architecture was influenced by the structure of the Visual Cortex and is comparable to the connection patterns of biological neurons. [1]

In 1958 [2]and 1959 [3] David H. Hubel and Torsten Wiesel carried out series of experiments on cats, providing valuable insights into the structure of the visual cortex. Some of the visual cortex's neurons can only respond to stimuli in a limited area of the visual field, which is known as the receptive field. The researchers also discovered that some neurons can only respond to horizontal lines, while others only respond to lines of varying orientations(two neurons may have the same receptive field but react to different line orientations). These fields overlap and are combined to form the entire visual area. It has been suggested that higher-level neurons are capable of responding to more complex patterns due to their connection with low-level ones. (on Figure 1.1, notice the relation between the neurons, how each of them is only linked to a few from the preceding layer). This elaborate architecture can detect a wide range of complex patterns in any part of the visual field. [1]

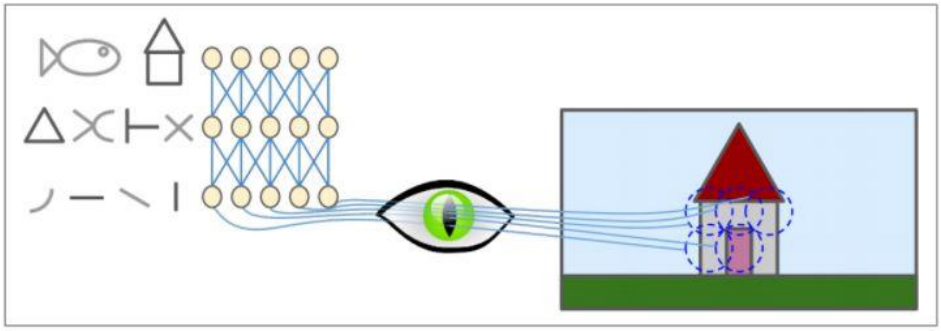


Figure 1.1 Local receptive fields in the visual cortex

These visual cortex experiments influenced the neocognitron that was implemented in 1980, which eventually developed into what we now call the Convolutional Neural Networks (CNN). [1]

1.2 Building blocks of CNN architecture

A convolution network is formed by stacking multiple convolutional layers, with non-linear activations in between. In the end, fully connected layers could be used. Going from the first layers to the last, the features obtained by the network have different meanings. The firsts layers represent edges, textures, and other low-level features, while the features found in the upper ones could represent entities such as parts or whole objects. One single output point corresponds to an area of

the image. The more convolutions we stack at a time, the bigger is the area that is analyzed from the input.

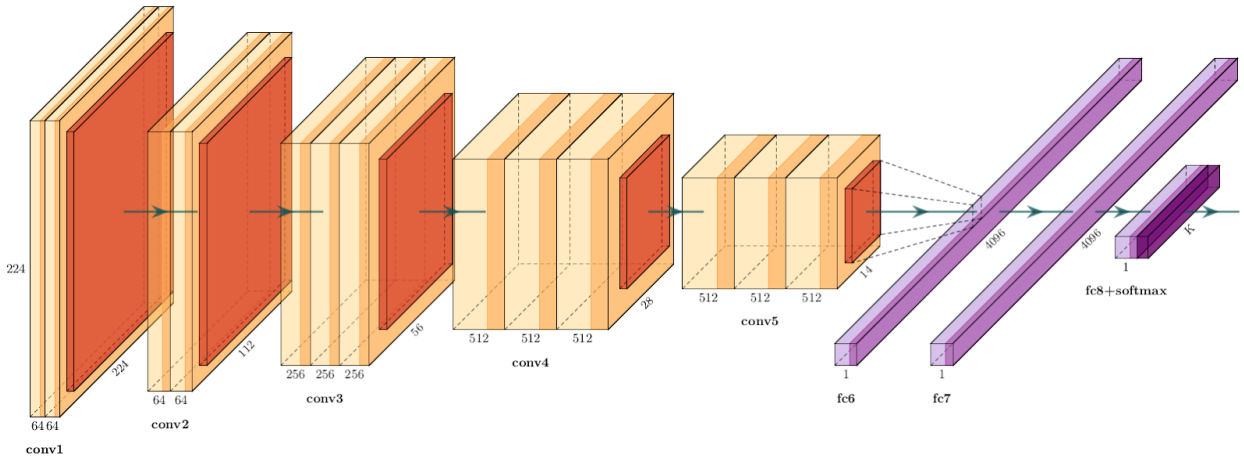


Figure 1.2 Convolutional Neural Network Architecture Example [4]

1.2.1 Convolutional layer

The convolutional layer is the central component of a CNN, which makes most of the computational work of the network. The convolutional layer, as any other layer, receives an input and then processes it. The operation, in this case, is called *'convolution'* and is pointwise multiplication, which combines two functions. [1] The weight vector *(kernel)* is used to generate a feature map by sliding over the input. These operations take a number of features in an image and extract them into a single layer. [5] To put it simply, for an image to be classified, certain features need to be extracted from the image, and that is the objective of the convolution. [6]

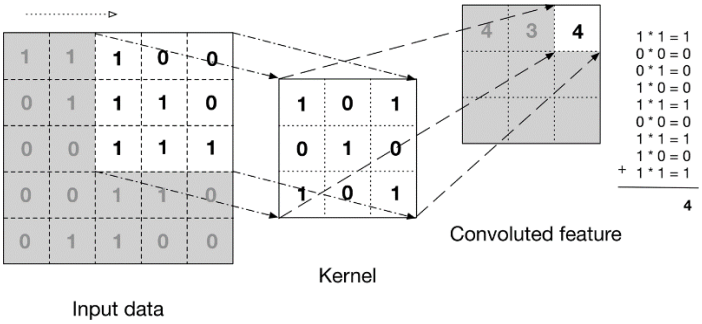


Figure 1.4 Convolutional layer

1.2.2 Pooling

A pooling layer situates usually between a subsequence of one or more convolutional layers. Its main purpose is to compress or generalize feature representations and decrease the number of trainable parameters as well as the output resolution. This helps the model to be equivariant to small local translations. Also, it helps make computations lighter. This works on the principle that when we find a considerable activation it does not matter exactly where we find it. There exist few pooling techniques but the most commonly used are Max pooling and average pooling. [5]

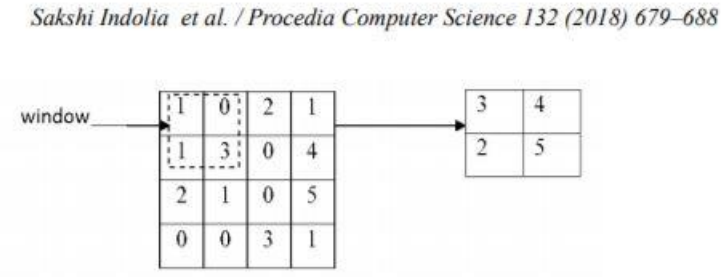


Figure 1.5 Pooling operation example [7]

1.3 Regularization and Dropout:

Regularization refers to any strategy used to reduce its generalization error. It is a technique that makes slight improvements to the learning algorithm, even though the training accuracy remains the same. To enhance generalization, constraints and penalties are used to encode past knowledge

or to establish a preference for simpler models. Deep Learning models are prone to overfitting, so these kinds of strategies are used to prevent it from happening.

Dropout is a regularization method that makes our models more robust by making each neuron not rely on fixed features. At each step in the optimization algorithm, it makes a percent of neuron activations zero. The network will have to adjust because it will not receive complete data during training. At testing time, we use all the output but scaled with that percent.

1.4 Architectures Evolution

1.4.1 Convolutional Networks in 1998: LeNet-5

LeNet-5 has played a significant role in the evolution of deep learning. It was named after the network that first presented the concept of neural networks. LeNet is a structure proposed by Yann LeCun and he collaborates in the year 1998, in the research paper *"Gradient-Based Learning Applied to Document Recognition"* [8]. LeNet is a feed-forward neural network and can run smoothly in large-scale image processing. At Bell Labs, Yann LeCun and his colleagues first presented the backpropagation algorithm in 1989, followed by a prototype of a neural network that could identify handwritten numbers. They believed that it might be improved by imposing task limitations. In 1990, they described the use of backpropagation networks for recognizing handwritten digits. The pre-processing done to the data was minimal, and their model was built to be highly constrained.

Being a prototype for the initial development of convolutional networks, LeNet has the following staple units: convolutional layer, full connection layer, pooling layer, and sub-sampling layer.

LeNet-5 remains the name of the network that defined the essential components of CNNs. Nowadays, CNN models differ greatly from LeNet, yet they are all built on its foundation. [9]

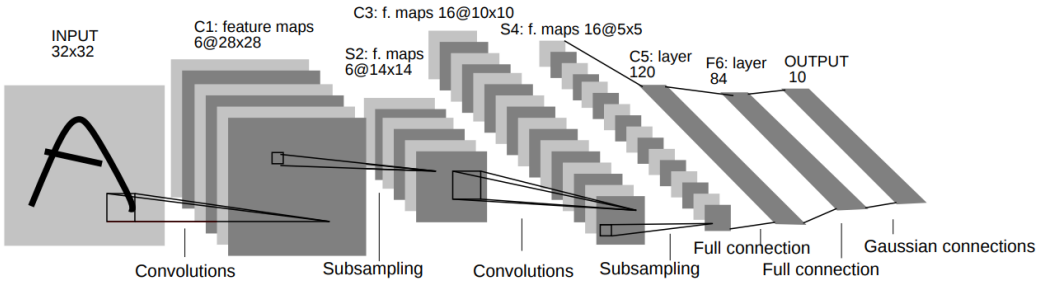


Figure 1.6. LeNet architecture.

1.4.2 Convolutional Networks in 2012: AlexNet

AlexNet is a leading architecture in the Computer Vision domain, giving the restart of neural network applications for the current research. Designed by Alex Krizhevsky, it was proposed in "*ImageNet classification with deep convolutional neural networks".* [10] In 2012, it achieved a top-five error rate of 17% on *the "ImageNet Large-scale Visual Recognition Challenge"* [11], thus winning the competition. The error obtained was significantly better than the second-best performer, which only achieved a 26% error rate. [1] The depth of a model's complexity was the reason why it was computationally expensive to perform. The paper shows that the model can be trained without sacrificing its high-performance characteristics, through the use of GPUs during training. [10]

The researchers used the Relu function to accelerate the training process. They also used the dropout layers to prevent the model from overfitting. [12]

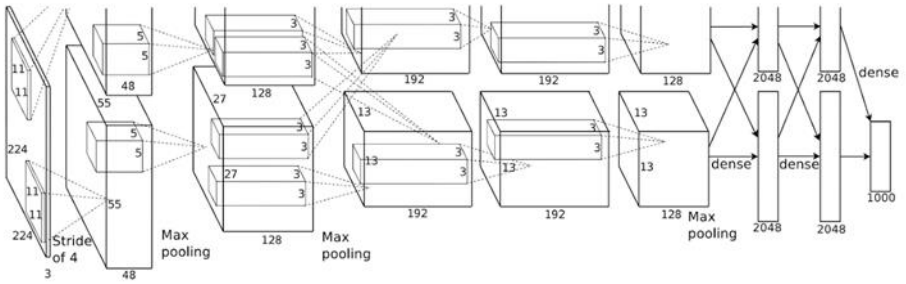


Figure 1.7: AlexNet architecture

1.4.3 Convolutional Networks in 2014: GoogleNet - Inception V1

Christian Szegedy et al. from Google Research has developed the GoogLeNet [13] architecture, which was successful in the 2014 ILSVRC challenge and achieved a 7% error rate. Inception overcomes the limitations of existing frameworks for classification and detection. Its main goal is to increase the efficiency with which available computer resources are used while maintaining the computational budget. Its performance was greatly improved by the fact that the network was deeper and wider than previous CNNs. The ability to use more parameters more efficiently was made possible by the introduction of inception modules. These modules were inspired by the 2010 Inception movie, where characters keep going deeper and deeper into multiple layers of dreams. Figure 1.8 shows the architecture of an inception module. [1]

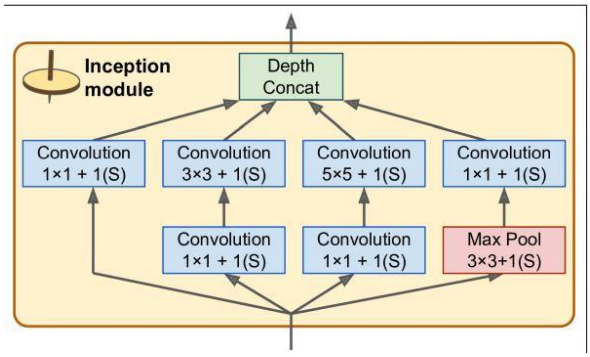


Figure 1.8 Inception module 1

What is interesting is that several kernels of different sizes were applied independently at the same time, later concatenating the result of all branches. This allows capturing patterns from multiple scales, resulting in a more flexible level of complexity. [14]

Unlike previous architectures, that end with a fully connected layer, GoogleNet uses global average pooling instead. This layer that averages each feature map is followed by a final linear layer with SoftMax activation. Additionally, auxiliary classifiers were added to the intermediate layers. Their goal was to combat vanishing gradients and act as regularizers.

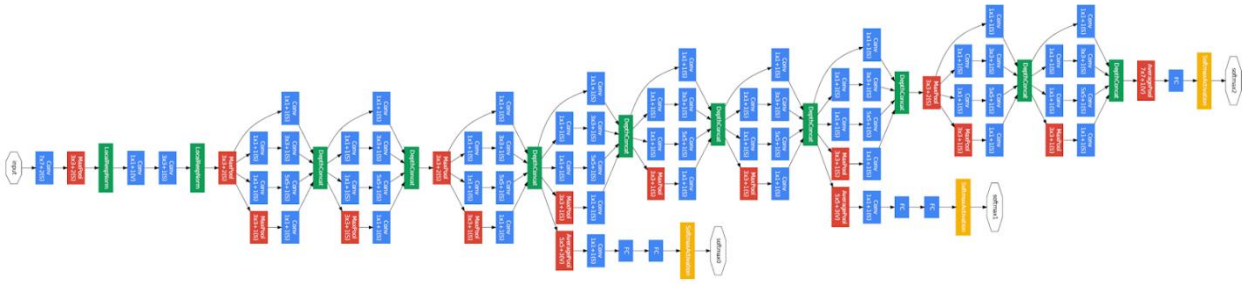


Figure 1.9 GoogleNet - Inception V1

1.4.4 Convolutional Networks in 2016: ResNet

After the appearance of AlexNet, deep convolutional networks have made significant advances in image classification. They can integrate low-to-mid-level features and can provide high-level classifiers with deep models with a depth of over sixteen to thirty. However, each consecutive winning architecture employs additional layers to reduce the error rate. [15] It has been shown that with the increase in the number of layers that the neural network is made of, training gets more difficult, and accuracy begins to saturate and eventually declines. [16]Figure 1.10 shows how deep 'plain' networks with an increased count of layers obtain worse results.

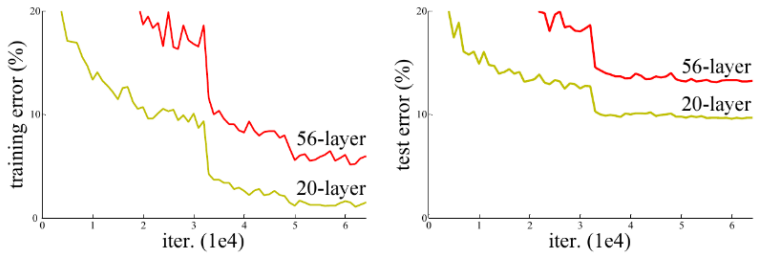


Figure 1.10 how deep 'plain' networks obtain worse results.

This degradation issue was solved by Kaiming He et al. [16] which developed The Residual Network (or ResNet), which won the *"ILSVRC 2015 competition"*. Using an extraordinarily deep CNN built of 152 layers they achieved an incredible error rate lower than 3.6%. The key to training such a deep network was by implementing skip connections (also called shortcuts) to jump over some layer. Most ResNet models are implemented with multiple parallel skips. [16]

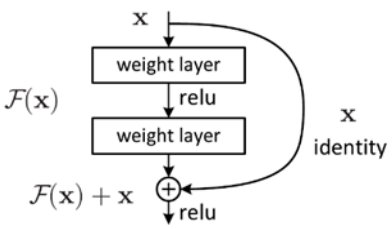


Figure 1.11 Residual block

When a network has too many levels it is hard to propagate gradients through all of them. An efficient method of fixing this issue is by gathering an intermediate level from a particular layer and add it to a greater depth. Skipping helps minimize the impact of vanishing gradients, which slows the learning process. It also helps minimize the need for additional training data as the network learns the feature space. In most cases, the weights for the adjacent layer are adapted to minimize the upstream layer's complexity.

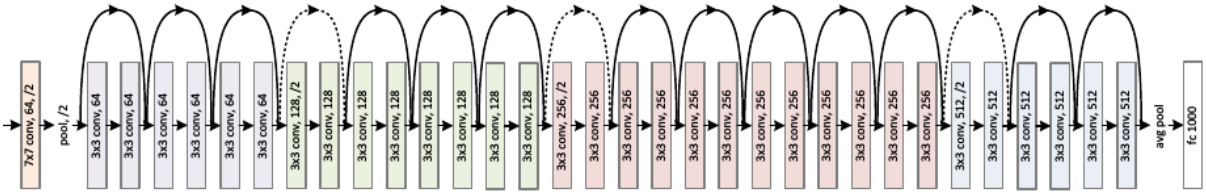


Figure 1.12 ResNet architecture

1.5 2D VS 3D Convolutional Neural Networks

The different dimension that the kernel slides along the data is the fundamental distinction between 2D and 3D convolutions. [17] For the Conv2D, the kernel moves in two directions. The model has a 3-dimensional input (depth, height, and width), it is mainly used on image data. Contrary to popular belief, simple 2D images are three-dimensional, having width, height, and color channel. When it comes to Conv3D, the kernel slides in three dimensions, which result in the model having an input that is 4-dimentional such as 3D medical images(height, width, depth, color channel) or videos (height, width, depth, frames). In the images bellow we have a visual representation of the two operations.

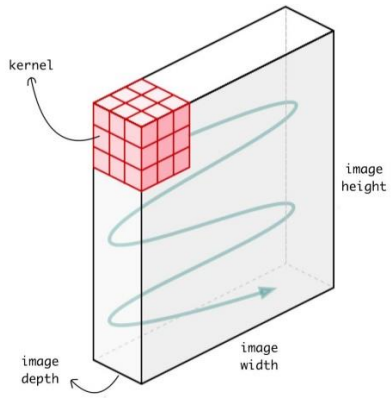


Figure 1.13 2DConv

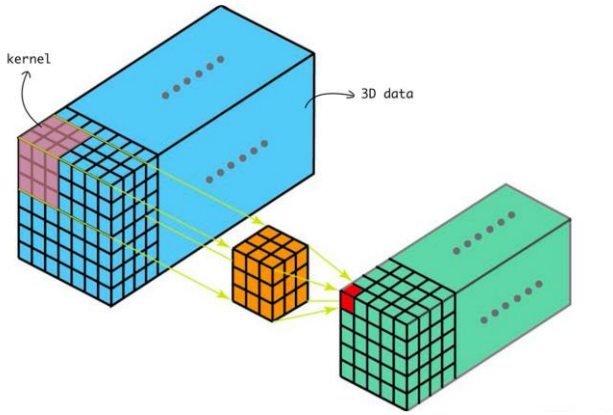


Figure 1.14 3DConv

1.6 Transfer Learning

When learning a new skill, humans reuse knowledge and practices that they already learned before. For example, when learning physics, we use knowledge learned from mathematics. How can we make ML models benefit from previously learned tasks, to learn more efficiently just as humans use prior knowledge? They could be given a start from a model that already learned on a previous task. It would output new representations for a new problem, by building on the old representations. This is referred to as transfer learning. This method is done by taking a network already trained and reuse its parameters. This is commonly used when the data available for the task is not enough. The easiest way is to retrain only the last level without changing anything from the ones before. This approach is not always useful, but we can retrain different levels of the network. The more the domains are similar to each other, the more we benefit from the use of transfer learning. Even though the task might seem different, the lower layers look at the texture, edges so we might use something.

Chapter 2 Recent Approaches

For quite some time now, deep learning has gained popularity due to its success in object recognition from images [18] As a result, many areas that use deep learning for real-world application tasks have developed, such as facial recognition [19], visual product search [20] and more [21, 22, 23]. One area that has been gaining a lot of interest is medical imaging analysis. [24] With all the complex tasks that medical imaging requires from specialists, deep learning-based radiology tools support them, easing their workload of interpreting various medical images. We briefly discuss some prior work related to hemorrhage detection and classification [25, 26, 27, 28, 29, 30, 31, 32], as well as transfer learning for these applications. [33]

Since Phong et al. [33] showed in his paper that 2DCNNs pretrained on ImageNet [34] can increase the performance of diagnosing intracranial hemorrhage, more papers have started to use this approach.

Nguyen, Nhan T. et al. [31] and Burduja et al. [25] suggest a CNN model pretrained on natural imaging, for feature extraction from slices. They integrate it with a long short-term memory (LSTM) that takes in the features given by the CNN and predicts intracranial hemorrhage subtypes by looking at the whole CT scan. This method is practical because the scans given as input for the LSTM do not need to have the same size, as it needs for a 3DCNN, also considering neighboring slices provides a performance boost. These papers also present models that used only certain Hounsfield Units(HU) windows that were previously chosen, such as brain, subdural and soft tissue windows, stacked as an RGB image. This gives an increase in performance compared to the models that train over the full range of HU values. For our 2D CNN model, we decided to follow the same window approach.

Another paper that uses this method is presented by Hojjat et al. [35]. In addition, it feeds these stack windows to two deep CNN pretrained models: SE-ResNeXt-50 and SE-ResNeXt-101, then averaging the result given by both, it follows with a sliding window module.

For an even better performance of deep learning in computed tomography, Lee et al. [30] developed a Window setting optimization (WSO) fully trainable module with CNNs. This module aims to find the optimal window settings from a CT image to increase the performance of intracranial hemorrhage detection. The models that were fed images that contained the full range

of HU values, as well as whose windows were manually chosen, were surpassed by the WSO model.

Majumdar et al. [36] offers a DCNN for feature extraction and classification, to eliminate multiple preprocessing steps, and improved its performance by calculating the average result for the different image's rotations outputs.

Grewal et al. [27] presented a model that gives radiologists level accuracy by reproducing the steps that are made by experts to analyses 3D CT scans. The model is a 40-layer DenseNet, that takes as input full range HU values 2D slices, to which they added three segmentation auxiliary tasks to push the model to focus on hemorrhagic regions. Then similar to [31, 25], they incorporated a bidirectional LSTM after the CNN, that uses the 3D context from neighboring slices.

A paper presented by Weicheng et al. [28] follows an approach based on a pixel-level supervision method that exceeds the performance of radiologists by detecting even features missed by experts. The authors built a patch-based fully CNN which performs joint classification and segmentation.

Chang et al. [26] presented a version of mask R-CNN, whose backbone is a hybrid 3D and 2D derived from the feature pyramid network, that can make use of collected data from five slices that surround the ROI.

The process that Saab et al [32] introduced uses a weak supervision process with the purpose of reducing the amount of time spent manually labeling the data. They start by extracting scan labels from what is found in written reports which are compared to the output of their classifier. They employ a shared 3D shallow image encoder, followed by an attention layer.

Singh et al. [29] proposed a shallow architecture for the classification of brain hemorrhages that scored higher accuracy than the optimized 3D VGGNet and 3D ResNet, due to their method. They employed a feature-enhancing strategy to create crisp edges and curves around anomalous ROIs to make feature extraction easier.

We can see that there have been multiple recent studies involving the use of 2D and 3D convolutional neural networks, specifically in hemorrhage detection, have been published. The problem with using a 2DCNN instead of a 3D one is that the whole CT scan should be sliced to get the data in the shape that the model accepts and that can cause loss of significant information. For our task, we have the exact opposite, we need to reconstruct the scans from the slices given in the dataset, so the data loss from the original form will be the same in both of our cases. We decided

to opt for a shallow architecture for both of our models while implementing some preprocessing presented in these papers for better performance.

Chapter 3 Technologies used

3.1Python

Getting started with AI and ML algorithms can be challenging and time-consuming. Having a well-structured and tested environment is very important to enable developers to implement algorithms successfully. Python is preferred for programming tasks related to artificial intelligence and machine learning due to its strong technological stack which includes a large number of libraries designed for these areas of programming. [37]

The next subchapters will present the libraries used for developing the program presented in this thesis.

3.2 PyCharm

For this thesis, we chose PyCharm as our Integrated Development Environment, for programming with Python. The 1.0 version was released in October 2010, without its open-source version (PyCharm Community Edition), which became available later in October 2013. [38] It features a graphical debugger, an embedded unit tester, and a web development environment with built-in extensions. [39] It can run on Windows, macOS and Linux versions.

3.3 Pydicom

Digital Imaging in Medicine(DICOM) was designed in early 1980 [40] and has since become the international standard for digital imaging and communications in medicine. It has enabled the replacement of film with a fully digital workflow. This has changed the way diagnostic imaging is done. DICOM sets the format and quality requirements for medical images and related data, and it is implemented in almost every cardiology, radiology, and radiotherapy imaging device. It is also used in other medical domains such as dentistry and ophthalmology. [41]

With its evolution, modern technology is needed to support the digital workflow of medical imaging. This includes tools that can handle the immense volume of images and metadata that are required to support the imaging process.

Pydicom [42] is an open-source library used for working with DICOM datasets. It was developed by Darcy Mason and contributors [43] to encourage the development of methods that can help people, by making it easier to build software that uses DICOM data and protocol. It allows for to easily reading, modifying, and writing data in this form.

This library was chosen for data handling because of its useful structure. By reading a DICOM file, you can have access to all the metadata of the slice, as seen below.

```
dicom = pydicom.dcmread('../rsna-intracranial-hemorrhage-detection/stage_2_train/' +
'ID_000012eaf' + '.dcm')
print(dicom)
plt.imshow(dicom.pixel_array)
plt.show()
```

```
Dataset.file_meta -------------------------------
(0002, 0000) File Meta Information Group Length  UL: 188
(0002, 0001) File Meta Information Version       OB: b'\x00\x01'
(0002, 0002) Media Storage SOP Class UID         UI: CT Image Storage
(0002, 0003) Media Storage SOP Instance UID      UI: 1.2.840.4267.32.337944818669776895705763408052798539612
(0002, 0010) Transfer Syntax UID                 UI: Explicit VR Little Endian
(0002, 0012) Implementation Class UID            UI: 1.2.40.0.13.1.1.1
(0002, 0013) Implementation Version Name         SH: 'dcm4che-1.4.35'
-------------------------------------------------
(0008, 0018) SOP Instance UID                    UI: ID_000012eaf
(0008, 0060) Modality                            CS: 'CT'
(0010, 0020) Patient ID                          LO: 'ID_f15c0eee'
(0020, 000d) Study Instance UID                  UI: ID_30ea2b02d4
(0020, 000e) Series Instance UID                 UI: ID_0ab5820b2a
(0020, 0010) Study ID                            SH: ''
(0020, 0032) Image Position (Patient)            DS: [-125.000000, -115.897980, 77.970825]
(0020, 0037) Image Orientation (Patient)         DS: [1.000000, 0.000000, 0.000000, 0.000000, 0.927184, -0.374607]
(0028, 0002) Samples per Pixel                   US: 1
(0028, 0004) Photometric Interpretation          CS: 'MONOCHROME2'
(0028, 0010) Rows                                US: 512
(0028, 0011) Columns                             US: 512
(0028, 0030) Pixel Spacing                       DS: [0.488281, 0.488281]
(0028, 0100) Bits Allocated                      US: 16
(0028, 0101) Bits Stored                         US: 16
(0028, 0102) High Bit                            US: 15
(0028, 0103) Pixel Representation                US: 1
(0028, 1050) Window Center                       DS: "30.0"
(0028, 1051) Window Width                        DS: "80.0"
(0028, 1052) Rescale Intercept                   DS: "-1024.0"
(0028, 1053) Rescale Slope                       DS: "1.0"
(7fe0, 0010) Pixel Data                          OW: Array of 524288 elements
```

Figure 3.1 Pydicom Metadata

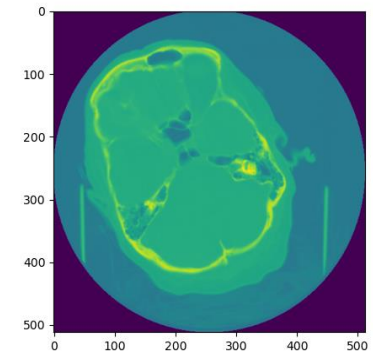


Figure 3.2 Pydicom Pixel Array

3.4 PyTorch

PyTorch is an open-source library that enables developers to create deep learning projects in Python, used primarily for applications in computer vision and natural language processing. Its flexible approach and ease of use make it an ideal tool for developing deep learning models. [44]

PyTorch is an excellent choice for anyone wanting to learn how to develop deep learning. Its clean syntax, streamlined API, and ease of debugging make it an excellent first choice for developers. Also, it offers two key features: accelerated computation, through GPUs, and numerical optimization. [45] This library can be used in various scientific applications, such as rendering, modeling, and simulation. Its goal is to provide all the necessary building blocks for developing deep learning networks and training them. [45] Let us describe more detailed how PyTorch supports deep learning projects. Figure 3.3 depicts a typical configuration for a PyTorch model.

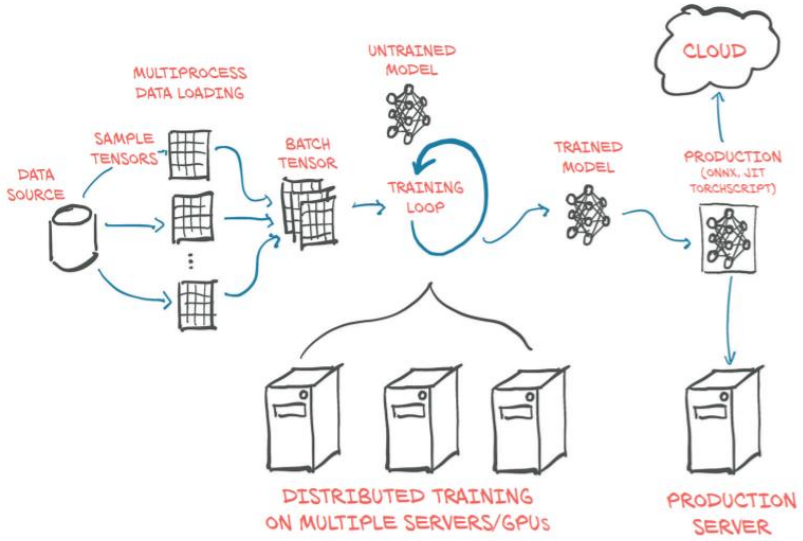


Figure 3.3 PyTorch project structure [45]

Firstly, PyTorch has a data structure in the form of a multi-dimensional matrix that coordinates all the inputs and outputs in a network. It is called a *tensor* [46]. In addition, the torch module provides various functions that can run on them. This makes programming deep learning in PyTorch very natural. Both the operations and the tensors can be used on the CPU and the GPU. One of the core features that PyTorch provides is its ability to move operations from the CPU to the GPU. [45]

Most architectural components needed for building and initializing a model are provided by the torch.nn module. At the left part of figure 3.3, we notice that before we can even start working on our model, we need to process the data we have collected. This is typically done in the form of converting our custom data to tensors. The Dataset class, located in torch.utils.data [47, 45] is used to build the link between our data and a standardized PyTorch tensor.

Due to the nature of data storage, it is often slow to handle. So, we need to parallelize the data loading by assembling it into tensors that contain multiple samples, called *batches*. PyTorch offers a simple implementation and that can handle both the initial and continuous loading of data from the dataset in the background using the DataLoader class. [45]

The training loop, represented in the center of figure 3.3, is typically implemented as a standard for loop. At every step, we will be evaluating the output from the model. Then, a loss function or a criterion is used to compare the output to the target. The objective is to push the model's output to better match the target. This is done with the use of PyTorch's *autograd* engine and an optimizer that does the updates. The training loop is an important part of a deep learning project, which often consumes a lot of time and energy. However, its reward is a model optimized to the task at hand(depicted at the right of the training loop) [45]

Chapter 4 Proposed approach

4.1 Dataset

For this thesis, the dataset used is the rich image dataset from the "*RSNA Intracranial Hemorrhage Detection Competition"* [48, 49] hosted on Kaggle. It was a classification competition, where the task was to detect hemorrhages found in head CT images and classify them into subtypes. The dataset contains multi-labeled data: about 874k human anatomical brain CT slices of the head region labeled with different types of hemorrhage. The sub-types consist of hemorrhages in the intraparenchymal, intraventricular, subarachnoid, subdural, epidural, and a label that should be true if any hemorrhage exists.

The format for all provided brain CT scans is DICOM, which is the outcome of the *Digital Imaging and Communications in Medicine standard*. [50] In addition to the image data, it includes metadata alongside the pixel data, which normally includes the patient's data (name, age, condition) and scan's data(pixel size, thickness). What differentiates the image from a DICOM file from the usual is the meaning of each pixel's value which is measured in Hounsfield units. This measurement shows the density of the tissue. Some examples of HU values and what they mean can be found in Figure 4.1. This is useful for medical imaging analysis because radiologists can focus on different intensity windows of HU, for visualizing different tissue types. [51]

Substance	HU
Air	−1000
Lung	−500
Fat	−100 to −50
Water	0
CSF	15
Kidney	30
Blood	+30 to +45
Muscle	+10 to +40
Grey matter	+37 to +45
White matter	+20 to +30
Liver	+40 to +60
Soft Tissue, Contrast	+100 to +300
Bone	+700 (cancellous bone) to +3000 (cortical bone)

Figure 4.1 HU values [51]

Before going into describing our chosen architectures, we need to understand the structure of the images from the dataset. Each DICOM file represents a slice from a CT scan. In the dataset, there are multiple slices that come from the same scan, approximately 800k slices from 25.000 scans. [25] We can group them using the metadata that we previously mentioned. The dataset is split into training and testing, but since we do not have access to the labels from the testing dataset, we will be using a percentage from the training data as our validation dataset to test our models' performance.

4.2 Model and Feature Selection

In this thesis, we wanted to experiment with the tradeoff between needing large amounts of data and computational costs. We chose two Pre-trained Convolutional Models: A 2D ResNet18 and a 3D ResNet18, which we adapted to our task. We wanted to view how a shallow architecture can perform on detecting hemorrhages in these two forms.

4.2.1 2DCNN Pretrained ResNet18 by PyTorch

As we previously stated, we wanted to use a pretrained model for our classification task. We opted for a model from TorchVision due to its ease of use and understanding.

The TorchVision.Models [52] subpackage, provided by PyTorch, includes model definitions for a wide range of purposes, such as object recognition, instance segmentation, video classification, etc. Some examples of model architectures for image classification that can be found here are AlexNet, GoogleLeNet, ResNet, (which we discussed in Chapter 3), and many more [52].

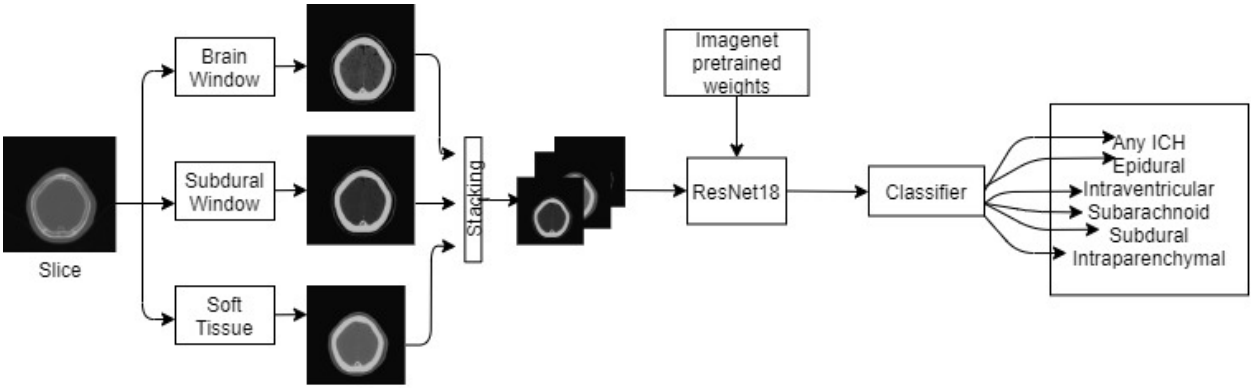
The Resnet models found in the subpackage [53] were presented in "*Deep Residual Learning for Image Recognition"* [16]. There are five variations of ResNet architectures distinguished by their number of layers: 18, 34, 50, 101, and 152. Each deep residual network is pre-trained on ImageNet [34] and expect as input batches of RGB-like images, normalized in the same way as the model was trained. [53] In figure 4.2 the detailed model architectures is described.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

ures for ImageNet. Building blocks are shown in brackets (see also Fig. 5), with the numbers of block

Figure 4.2 ResNet model architectures

Due to the large dataset, we opted for a shallower architecture: the 2D ResNet18. For this model, we are going to follow a slice-by-slice approach. As we have previously stated in Chapter 2, it has been proved that ImageNet-pretrained CNNs can aid in a successful diagnosis of cerebral hemorrhage [33] . The model is going to be fed with all the training data (730k slices from our dataset), predicting without considering neighboring slices. We are going to extract different intensity windows from the slice and stack them as an RGB image, which will be our model's input.



4.3 The representation for our 2DCNN approach

4.2.2 3DCNN Pretrained ResNet18 by MedicalNet

MedicalNet is an open-source library, provided by Tencent, a Chinese multinational leading technology company [54]. The repository contains a PyTorch implementation based on the paper *"Med3D: Transfer Learning for 3D Medical Image Analysis"* [55]**.**

The volume of training data that is required to develop medical analysis projects significantly affects the performance of the models. Due to the complexity of 3D medical imaging data acquisition and annotation, it is really hard to construct a dataset that is large enough for training models properly. As a solution, Sihong Chen et. al [55] compiles various medical challenges into a single 3DSeg-8 dataset. The paper also introduced Med3D, a heterogeneous 3D network that enables the co-training of multi-domain 3DSeg-8 with pre-trained models that can extract general 3D features. Then transfer these features to various 3D medical image tasks, such as lung segmentation, pulmonary nodule classification, and liver segmentation. The goal was to develop a pre-trained 3D model that can be used to improve other medical analysis tasks with limited training data.

The ability of the Med3D to accelerate the training of 3D medical tasks was demonstrated in the experiments presented in the paper. What we found interesting is that the training convergence speed was twice faster than models pre-trained on the Kinetics dataset. It also has the potential to boost accuracy by up to 20%. [55]

In MedicalNet we can find a series of 3D-ResNet models pretrained on the dataset from the *"Grand Challenge on MR Brain Segmentation at MICCAI 2018"* [56]. The task of this challenge was the segmentation of brain parts in MRI images.

To view the advantages of using MedicalNet, the authors provide a table with the accuracies that ResNet models achieved on two challenges trained from scratch and pretrained.

Network	Pretrain	LungSeg(Dice)	NoduleCls(accuracy)
3D-ResNet10	Train from scratch	71.30%	79.80%
	MedicalNet	87.16%	86.87%
3D-ResNet18	Train from scratch	75.22%	80.80%
	MedicalNet	87.26%	88.89%

Figure 4.4 Accuracies using MedicalNet

After looking at the results for our chosen ResNet18 model, we can observe that it improved the accuracy on both tasks. Naturally, since it was pretrained for a segmentation problem, it increased the accuracy more on the LungSeg than on the nodule classification task. However, an 8% increase in accuracy is considerable in any deep learning problem.

With this 3D ResNet18, we will follow a scan approach, considering neighboring slices. The model takes as input multiple consecutive slices that are stacked to create a whole scan. Because 3DCNNs are computationally expensive, we cannot 'afford' to feed it all the 20.000 scans. And since the paper that backed this pretrained model states that using it will remove the need for a huge dataset, we chose a smaller subset of 6.700 scans. We wanted the labels to refer to the whole scan instead of each slice, so we manually labeled every reconstructing scan by looking at its slices. If a subtype of hemorrhage is present in any of them, then its label will be 1 for the entire scan. Since the model is already pretrained on Medical Imaging, not on natural images, we expect it to perform better, even though the data is less than the one we want to feed to the 2DCNN.

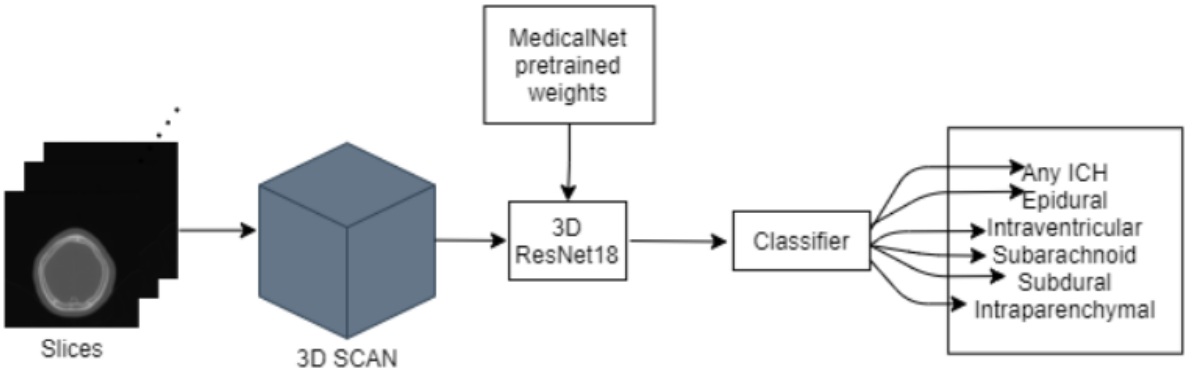


Figure 4.5 The representation for our 3DCNN approach

4.2.3 Adjusting the models

Our goal is to detect the presence of hemorrhage, as well as classifying it into subtypes. For both of our approaches we can have many forms of hemorrhage present in a single data sample and the general label 'any' must be true for all the scans/slices that contain hemorrhage, no matter the type. By looking at this structure of the labels, both of our tasks can be considered as a multilabel classification problem.

The 2D Resnet18 model from PyTorch is trained to classify natural images in one of 1000 classes, while the 3D ResNet18 from MedicalNet is trained for a brain segmentation task. Since both of our models are not built to fit a multilabel classification problem we need to adjust them to suit our task's needs. To do so, we replaced their last layer with a fully connected layer that outputs six classes. We want the model to predict the probability that a subtype of hemorrhage is present, meaning we want the output for each class to be a value ranging from 0 to 1. Also, being a multilabel classification problem, the outputs are not mutually exclusive. Having this in mind, we used a Sigmoid activation function to follow the last layer. Sigmoid, unlike SoftMax, does not give a probability distribution around n classes as output, but independent probabilities.

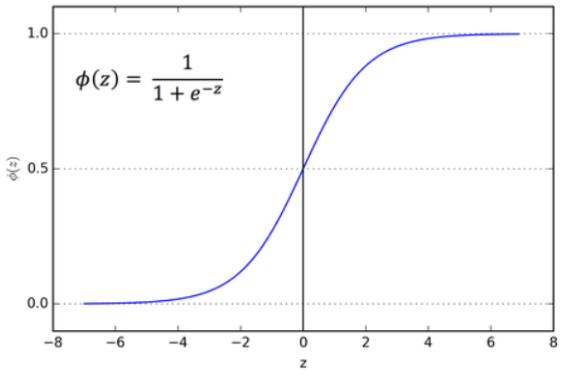


Figure 4.6 Sigmoid function graph

4.3 Loss and Optimization

We train the models to detect hemorrhage and to classify it into subtypes simultaneously. For all the labels: intraparenchymal, intraventricular, subarachnoid, subdural, epidural, and 'any' we need to predict its label: present (1) or not (0). As discussed above, our last layer is succeeded by a sigmoid activation function, meaning our model returns six values between 0 and 1 indicating the independent probabilities that each subtype of hemorrhage is present as well as being any subtype. Bearing this in mind, we can treat the problem as six different binary and independent classification problems. Usually, the loss function suitable for this kind of task is Binary Cross-Entropy(BCE). It sets up a binary classification task between two classes (present or not) for every class that we have (all the subtypes and detection). It is also class-independent, which means that the loss

estimated for a component from the output vector is unaffected by the values of other components. [57, 58]

$$\boldsymbol{Binary\ cross\ entropy} = -\frac{\mathbf{1}}{\boldsymbol{C}}\sum_{\boldsymbol{i=1}}^{\boldsymbol{C}} \boldsymbol{y_i} \cdot \boldsymbol{log}(\boldsymbol{\hat{y}_i}) + (\mathbf{1} - \boldsymbol{y_i}) \cdot \boldsymbol{log}(\mathbf{1} - \boldsymbol{\hat{y}_i})$$

$where\ \hat{y}_i\ is\ the\ value\ in\ the\ output\ vector\ on\ the\ ith\ position,$
$y_i\ is\ the\ ith\ target\ value, and\ C\ is\ the\ number\ of\ classes.$ [59]

For our models we chose to use Adam as the optimizer, with different learning rates: 1e-5 for our 2DCNN model, and 3e-5 for our 3DCNN. We fine-tuned them for six, respectively five epochs.

Chapter 5 Experiments and Results

The performance of our models on RSNA is measured by the log loss (Binary Cross Entropy), both during training and validation. Also, to have a better understanding of our models' performance we used metrics specific for multilabel classification problems. Both models will predict a total of six probabilities for each sample, five hemorrhage subtypes, and the detection label. For the rest of the evaluation metrics used we converted the output probabilities to strict class labels by rounding them to either 0 or 1. The first metric that we used is accuracy, this considers that a model has given a correct output only if all the labels match the target one. Then for a better measurement that takes into consideration even partially correct outputs, we used the Hamming score. This "accuracy" is defined for each output as the proportion of predicted accurate labels to the overall number of predicted and actual labels for that output. The score is the mean of the total cases and is calculated as so:

$$\boldsymbol{Hamming\ score} = \frac{\mathbf{1}}{\boldsymbol{n}} \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \frac{|\boldsymbol{y_i} \cap \boldsymbol{\hat{y}_i}|}{|\boldsymbol{y_i} \cup \boldsymbol{\hat{y}_i}|}$$

The last metric that we used is the Hamming loss. It calculates the average number of times an example's relationship to a class label is predicted incorrectly. As a result, hamming loss takes into account both the prediction error and the missing error. [60] Ideally, we would anticipate the hamming loss to be zero, implying no mistake; realistically, the lower the value, the greater the efficiency.

$$Hamming\ Loss = \frac{1}{nL} \sum_{i=1}^{n} \sum_{j=1}^{L} I(y_i^j \neq \widehat{y_i^j})$$

where I refers to the indicator function. [60]

We also used several other metrics such as: Precision, Recall/Sensitivity, F1 score and Specificity. For the Precision, F1, Sensitivity, and Specificity we used 'macro' averaging, not considering zero division (when de denominators were zero).

5.1 2D Model

5.1.1 Experimental Setup for our 2D model

The RSNA training dataset contains about 25.000 non-contrast brain CT images, each with 20 to 60 slices of size 512x512. [31] After removing the invalid images from our dataset, we used the remaining slices for our model's predictions. We split the data into training(80%) and validation(20%). The training set has 602.242 slices of which 515.840 have no hemorrhage, and 86.402 have one of the 5 subtypes present. The validation set has a total of 150.561 slices, of which 129.030 are healthy, and 21.531 have a positive hemorrhage diagnosis. The distribution of hemorrhage subtypes in our sets can be seen below.

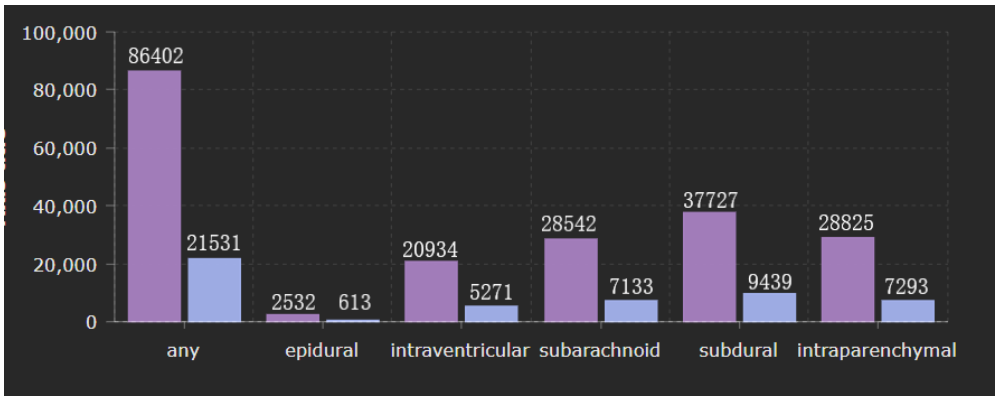


Figure 5.1 Distribution of hemorrhage types in our sets for the 2DCNN

We trained our 2D ResNet18 model for 57 hours on a NVIDIA GeForce GTX 1060 with Max-Q design, for 6 epochs, using a *1e-5* learning rate for our Adam optimizer. The overall architecture was pre-trained on ImageNet [34]. For training time-reducing purposes, all images were downscaled to 256×256 pixels. Also, for increasing our model's generalization capacity, during training, we performed various augmentations. This method was used in several research papers that we presented in Chapter 2. [31, 25]

5.1.2 Results of our 2D model

After the training, we obtained a **0.0823** validation loss. We experimented with the model not pre-trained and obtained a best of **0.1133** loss after 4 epochs, compared with **0.0927** that we obtained with the pretrained after that number of epochs.

We were curious to try to see also how a larger architecture would perform, so we also trained the ResNet50 pretrained model from Torch Vision. To our surprise, the convergence speed was much slower at the beginning epoch, but at the end, it obtained a better loss of **0.0723** after all the epochs.

Table 1. Values obtained after four epochs of training, for the ResNet18 both pretrained and not, and ResNet50 pretrained.

Model	Loss after 4 epochs
ResNet18 pretrained	0.0927
ResNet18 from scratch	0.1133
ResNet50 pretrained	0.0845

For our pretrained model ResNet18 we calculated certain evaluation metrics after it has finished training.

Table 2. Metrics obtained on the validation set after our chosen model has finished training.

Metric	**Value**
Accuracy	0.8982
Hamming Loss	0.0281
Hamming Score	0.9249
Precision	0.8808
F1	0.7937
Sensitivity/Recall	0.7224
Specificity	0.9910

As we can observe the strict accuracy obtained was almost 90%, with a very small Hamming Loss, and a Hamming Score of 0.9249, we can say that our model has outperformed our expectations.

However, the data in our set is not equally distributed among our classes: there are five times more healthy slices than there are brains with hemorrhages of any kind, so the accuracy is not that insightful. If we look at the specificity, we can see that the model managed to diagnose 99% percent of the healthy brains correctly. Because for our task, it is worse if a brain with hemorrhage is miss-diagnosed as a healthy one than if a healthy one is considered ill, we will focus on Recall. Because we used a 'macro' average to calculate it, the number of false negatives could have come from the model misdiagnosing the slice with different types of hemorrhages, not necessarily as healthy. With that being said, the model successfully diagnosed the majority of the hemorrhages subtypes present but there is room for improvement.

5.2 3D Model

5.2.1 Experimental Setup for our 3D model

For our 3D Model, we followed a scan approach, each scan being labeled manually. In this case, a scan can have multiple subtypes of hemorrhages present.

Because 3D models are computationally expensive, we only used a subset of 6,700 scans from the RSNA dataset, with 4058 having no hemorrhages and 2692 having at least a subtype present. The distribution of hemorrhage subtypes for the chosen subset can be seen below.

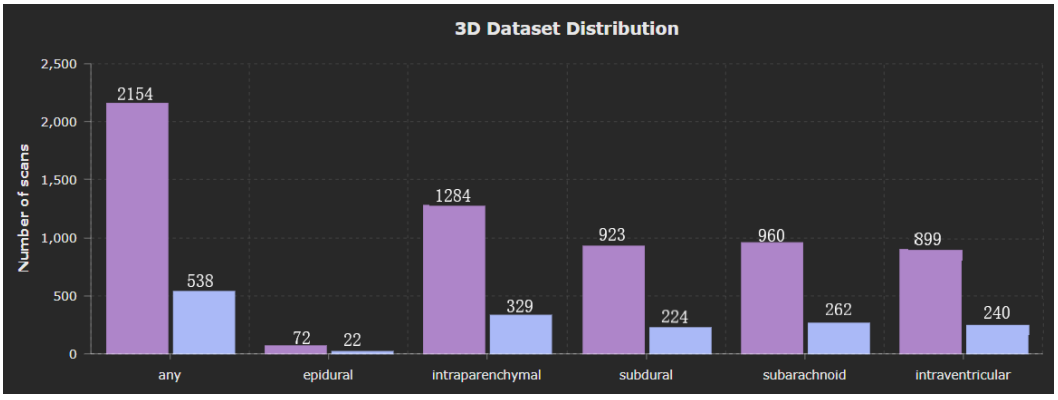


Figure 5.2 Distribution of hemorrhage types in our training set for the 3DCNN

We trained our 3D ResNet18 model for 35 hours on a N*VIDIA GeForce GTX 1060*, for 5 epochs. The learning rate chosen for our Adam optimizer was *3e-4*. The architecture was pre-trained on MedicalNet.

5.2.2 Results of our 3D model

After training, we obtained a **0.4947** validation loss, which was unexpected. The metrics obtained after testing the model on the validation set were far worse than the results we received on the 2D model. We used the same metrics with the same settings to best compare the two approaches.

Table 3. Metrics obtained on the validation set after our 3D model has finished training.

Metric	**Value**
Accuracy	0.6007
Hamming Loss	0.1995
Hamming Score	0.6007
Precision	0.1447
F1	0.1447
Recall/Sensitivity	0.1447
Specificity	0.8034

The value for precision and recall show that the model has given as output very few True Positives, meaning that it did identify only a few hemorrhages, most healthy brains.

This concludes that the 2D Model has outperformed the 3D one. By looking at the data given by the two models we could jump to the conclusion that we were going after in the first place: that the advantages of preprocessing on a similar set of images, computation power, and considering neighboring slices do not compensate to the need of a large number of data that is necessary for the model to give a proper diagnosis. However, we believe that this chosen model could have had two problems. First would be the localization problem, that having a large scan as input can cause the model to not “see” the hemorrhage because is only in a small location. The second would be that many scans have up to five different hemorrhages present, which means that for a single scan the model will have to solve multiple detection tasks. This could have led to difficulties in learning properly.

We believed that breaking a problem down into smaller tasks is easier than attempting to solve it as a whole. So, we experimented with giving five neighboring slices from the same scan stacked together as input for the same 3D architecture. This approach gives as output the prediction labels for the middle slice. We can say that this approach is the middle ground between the first two that we presented: it gives predictions at a slice level while being a 3D Network that takes into consideration neighboring slices, but not the whole scan. We obtain a loss of **0.2031**, after 5 epochs which is significantly less than the one we got for the previous 3D approach, but higher than the one we got for our 2D model.

Table 4. Metrics obtained on the validation set for the 3D model that takes as input five consecutive slices.

Metric	**Value**
Accuracy	0.7812
Hamming Loss	0.0572
Hamming Score	0.8177
Precision	0.6656
F1	0.5678
Recall/Sensitivity	0.4952
Specificity	0.9604

If we look at each model's specificity, we can observe that they all diagnosed healthy brains well, but in the case of the 3D model that considered the whole scan, this was due to the fact that it was diagnosing the majority of the brains as healthy, even though they were not. As we can see in Table 4, the sensitivity/recall value is not so high, being lower than the one we got for our 2D model with more than 20%. We can state that using this approach has made improvements in our 3D model, but it still has not reached the performance that we achieved on the 2D one. However, we got closer to the results that we obtain on the latter, meaning that with further development we could demonstrate that we can compensate the need for large amounts of data with computational power.

Chapter 6 Application Description

6.1 Labeling the data

The labels for our data are presented in the format presented in Figure 6.1. ID is formed by the image id (SOP Instance UID), merged with each subtype of hemorrhage on different rows, followed by the Label column that represents the diagnosis, 1 being present, 0 is not. For easier use, we split the ID field and build a CSV file, presented in Figure x2, that had one column for the Image Id, and 6 columns representing the label for each diagnosis. We then split the new training data in training (80%) and validation (20%). Later, we used this CSV file and structure for both of our networks. As we do not have access to the testing labels, we used our validation set to examine the network's performance.

ID	Label
ID_12cadc6af_epidural	0
ID_12cadc6af_intraparenchymal	0
ID_12cadc6af_intraventricular	0
ID_12cadc6af_subarachnoid	0
ID_12cadc6af_subdural	0

Image	any	epidural	intraparen	intraventri	subarachn	subdural
ID_000012eaf	0	0	0	0	0	0
ID_000039fa0	0	0	0	0	0	0
ID_00005679d	0	0	0	0	0	0
ID_00008ce3c	0	0	0	0	0	0

Figure 6.1 Labels before and after reconstruction

```
columns = ['epidural', 'intraparenchymal', 'intraventricular', 'subarachnoid',
'subdural', 'any']
train[['ID', 'Image', 'Diagnosis']] = train['ID'].str.split('_', expand=True)
train = train[['Image', 'Diagnosis', 'Label']]
train = train.pivot(index='Image', columns='Diagnosis', values='Label').reset_index()
train['Image'] = 'ID_' + train['Image']

validation = train[int(n*80/100):]
training = train[:int(n*80/100)]
```

6.1.1 2DCNN Model Labeling

For the 2D model we used the training and validation labels that we created above.

6.1.2 3DCNN Model Labeling

For our 3D Model labeling, the data was more complex. We first needed to reconstruct the scans, then to build the labels from the slices.

For the reconstruction, we used the metadata found in the DICOM files. Each slice has a field called 'Series Instance UID' that represents the id of the slice. We iterated through the .dcm files and grouped them into a dictionary with the scan id as the key.

Then, using the dictionary, we made the sum of the labels corresponding to each hemorrhage for the slices that belonged to a key scan. Because we are interested in only detecting the presence of a hemorrhage, not the number of slices that were found, we labeled that a subtype with 1 if the sum was greater than 1 for that certain subtype.

Then we created a single CSV file containing the scan id, labels, and images with padding on a single row. Then we split the file into training (80%) and validation (20%) datasets.

```
train = pd.read_csv('scans_with_lables.csv')

with open('building_3d.csv', 'w', newline='') as f:  # Python 3
    w = csv.writer(f)
    first_row = ['scan_id', 'epidural', 'intraparenchymal', 'intraventricular',
'subarachnoid', 'subdural', 'any']
    for i in range(max_slices):
        first_row.append('image_'+str(i))
    w.writerow(first_row)
    for key, items in train_scans.items():
        row = []
        labels = train[train.scan_id.str.match(key)].values[0]
        row.append(labels[0])
        for i in range(1, len(labels)):
            row.append(int(labels[i]))
        for item in items:
            row.append(item)
        w.writerow(row)
```

6.2. Data and Preprocessing

The preprocessing differs between the two types of Neural Networks chosen. For the 2DCNN we will be using a slice approach, while for the 3D model we will need to reconstruct the scans. In the next subchapters, we will be discussing in detail the preprocessing made for each model.

6.2.1 2DCNN Model

For the 2DCNN, is not necessary to reconstruct the scans since we are interested in the diagnosis at the slice level. To get the most use out of the DICOM format, we extracted the different intensity windows that are most useful for diagnosing a hemorrhage. We manually chose three tissue windows: brain, subdural and soft tissue then stacked them as an RGB image. After creating the three-channel images from our CT slices, we reduce their spatial dimension from 512x512 to 256x256 pixels and normalized. Then, we used the Albumentations library to augment the dataset and generate multiple variations of the images. The transformations we selected were horizontal flipping, vertical flipping, rotation, scaling, and brightness and contrast adjustment.

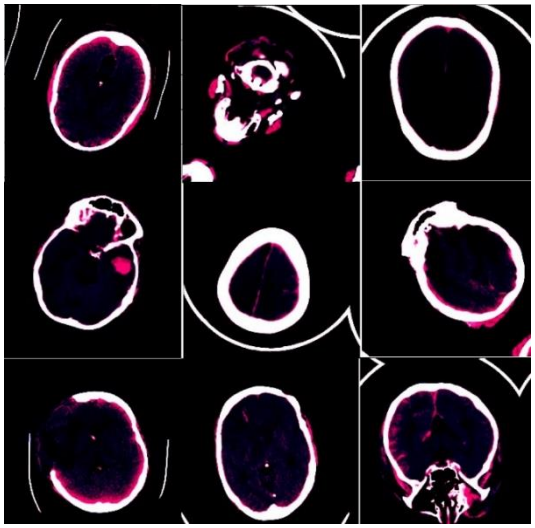


Figure 6.2 Slices after preprocessing

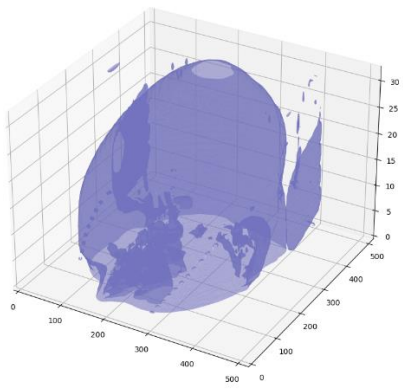
6.2.2 3DCNN Model

For our data preprocessing we were inspired by the tutorial [61] provided by Guido Zuidhof for the Data Science Bowl 2017 Competition [62] on Kaggle. In this tutorial, he goes through the steps to take in preprocessing the 3D CT Lung Scans from the competition's dataset: loading the DICOM files, resampling, 3D plotting, lung segmentation, normalization, and zero centering. For our dataset, we found useful all the steps besides the segmentation and resampling. Next, we will be detailing the steps that we followed.

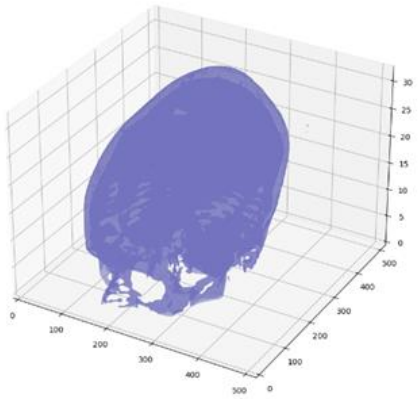
For loading the scans, we save the slices in a python list, sorted by Image Position Patient, that is found in each slice's metadata. In addition to the tutorial, we needed the scans to have the same dimensions, so each scan was padded with slices containing only HU units of value 0 to match the length of the scan with the maximum number of slices. We saved a copy of one of the DICOM slices in which we altered its pixel_array, then for every scan that we load we replace the scan id in the metadata (SeriesInstanceUID). Figure 6.3 shows the 3d plot of the first scan without the padding.

```
zeros = pydicom.dcmread(data_path + 'ID_000012eaf' + '.dcm')
x = np.zeros(zeros.pixel_array.shape)

zeros.PixelData = x.tobytes()
print(zeros.pixel_array)
filepath = os.path.join(data_path, 'Pad'+'.DCM')
zeros.save_as(filepath)

def load_scan(scan):
    slices =[]
    id = pydicom.dcmread(data_path + scan[0] + '.dcm').SeriesInstanceUID
    for sclice in scan:
        if sclice !='Pad':
            slices.append(pydicom.dcmread(data_path + sclice + '.dcm'))
        else:
            pad = pydicom.dcmread(data_path+sclice+'.dcm')
            pad.SeriesInstanceUID = id
            slices.append(pad)


    slices.sort(key=lambda x: float(x.ImagePositionPatient[2]))

    thickness = np.abs(slices[0].ImagePositionPatient[2] -slices[1].ImagePositionPatient[2])

    for sclice in slices:
        sclice.SliceThickness = thickness

    return slices
```

By looking at Figure 6.3, we see that need to convert to HU pixels. The corrected scan is presented in Figure 6.4.



Fiugure 6.3 Scan before converting to HU



Fiugure 6.4 Scan after converting to HU

After that we reduced the dimension from 512x512 to 256x256 for each slice and normalized them.

6.3 Thesis Implementation Details

6.3.1 2DCNN Implementation Details

To feed the data to our networks we used the Dataset and DataLoader classes provided by PyTorch in torch.utils.data.

For the **2DCNN** we have created the 'TwoDCNN_Dataset' class, that inherits Dataset. We implemented the three functions necessary for a custom dataset: *__init__*, *__len__*, and *__getitem__*.

```
class TwoDCCN_Dataset(Dataset):

    def __init__(self, path_dicom, labels_file, transform_functions=None):
        self. path_dicom = path_dicom
        self.data = pd.read_csv(labels_file)
        self.transform = transform_functions
```

Path_dicom represents the path to the folder that contains the DICOM files, labels_file is the training/validation file, we will use the file that we created in chapter 6.1 and turn it into a DataFrame using Pandas and transform_functions will receive the albumentation methods used

to transform the data as discussed in *Chapter 6.2.1*. Intuitively, we use the size of our training table to get the length.

```
def __getitem__(self, id):
    try:
        image = self.data.loc[id, 'Image']
        dicom = pydicom.dcmread(self.path + image + '.dcm')
        slice = rgb_window(dicom)
        if self.transform:
            transformed = self.transform(image=slice)
            slice = transformed['image']
    except:
        slice = np.zeros((256, 256, 3))

    labels = torch.tensor(self.data.loc[
    id, ['epidural', 'intraparenchymal', 'intraventricular', 'subarachnoid',
'subdural', 'any']])

    return {'image': slice, 'labels': labels}
```

For the __getitem__ method we want to return the slice, transformed as we described in Chapter x, and its labels. Using our pandas DataFrame, we read the DICOM file and then transform it into an RBG-like image. Then we apply the transformations and return them with their labels for every hemorrhage. We used the same method for the validation set, only the csv_file was different.

```
train_dataset = TwoDCCN_Dataset(csv_file='train.csv',
                path='../rsna-intracranial-hemorrhage- detection/stage_2_train/',
                transform=albumentations.Compose(
                [albumentations.Resize(256, 256),
                 albumentations.Normalize(mean=[0.1738, 0.1433, 0.1970],
                                          std=[0.3161, 0.2850, 0.3111],
                                           max_pixel_value=1.),
                  albumentations.HorizontalFlip(),
                  albumentations.VerticalFlip(),
                  albumentations.ShiftScaleRotate(),
                  albumentations.RandomBrightnessContrast(),
                  ToTensor()]))
```

For loading the data, we used torch.utils.data.DataLoader, with batches of 32 slices, which was the maximum number our computer allowed.

Then we defined the model that we chose: the ResNet18 pretrained model from PyTorch and put it on our GPU for faster learning.

```
model_2d = models.resnet18(pretrained=True)
```

Fine-tuning the pretrained model comes next. The *FC* is our model's final layer, whose structure can be seen below:

```
(fc): Linear(in_features=512, out_features=1000, bias=True)
```

We need the out_features to be the number of our classes. Thus, we need to reinitialize the *FC* layer to have the same number of input features (512) and six output features. Because we needed to predict multiple labels from multiple classes, we added a Sigmoid activation to the end, paired later with BCE as our loss. We also added a Dropout of 0.5 to reduce over-fitting by regularizing the network. For our optimizer we chose Adam, from torch.optim package, with a learning rate of 1e-5.

```
num_ftrs = model.fc.in_features
model.fc = nn.Sequential(
    nn.Dropout(0.5),
    nn.Linear(num_ftrs, 6),
    nn.Sigmoid()
)

criterion = torch.nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=1e-5)
```

Next is the training part. We trained our model for 6 epochs. After each epoch we switched the model to evaluation mode and calculated the loss and accuracy, saving the model's state when the validation loss was smaller than the ones before it. At every 10 batches in training mode, we printed the loss so we can observe if the model was learning during training.

```
Epoch [1/10], Step [0/18821], Loss: 0.6628    Epoch [2/10], Step [0/18821], Loss: 0.0620
Epoch [1/10], Step [10/18821], Loss: 0.6294   Epoch [2/10], Step [10/18821], Loss: 0.1861
Epoch [1/10], Step [20/18821], Loss: 0.6482   Epoch [2/10], Step [20/18821], Loss: 0.1505
Epoch [3/10], Step [0/18821], Loss: 0.0452    Epoch [5/10], Step [0/18821], Loss: 0.0286
Epoch [3/10], Step [10/18821], Loss: 0.1470   Epoch [5/10], Step [10/18821], Loss: 0.1475
Epoch [3/10], Step [20/18821], Loss: 0.1386   Epoch [5/10], Step [20/18821], Loss: 0.1240
```

Figure 6.5 Printed losses for some epochs

Being a significantly larger number of slices with no hemorrhages, we noticed that the batches with the loss smaller than 0.06 were the ones containing only healthy CT slices, while the ones with the bigger losses, 0,1 and above were the one containing mostly hemorrhages.

6.3.2 3DCNN Implementation Details

For the 3DCNN we have created the 'TreeDCCN_Dataset' class, similar to the one for the 2DCNN that we described, only it has one more argument 'max_scans'. The *__init__* and *__len__* methods are the same as the ones in 'TwoDCNN_Dataset'. For the *__getitem__* method, we got the needed slices for a scan, then we used the loading, correction to HU, normalization, zero centering, and downsizing that we talked about in *Chapter 6.2.2*.

Each CT slice is comparable to a grayscale image in that it contains just one-color channel, so each scan has the dimensions : (max_scan, 256, 256). Since there are no channel details present, the structure of the input does not conform to what our model expects. To make room for the channel dimension we used unsqueeze. Now the shape of our output tensor is [(1, max_scans,256,256)].

```
def __getitem__(self, idx):
    try:
        slices = []
        for i in range(max_scans):
            slices.append(self.data.loc[idx, 'image_'+str(i)])
        img = load_scan(scan=slices)
        img_hu = get_pixels_hu(img)
        img = normalize(img_hu)
        img = zero_center(img)

    except:
        img = np.zeros(max_scans,256,256)
    if self.transform:
        for i in img:
            augmented = self.transform(image=i)
            i = augmented['image']

    vol = torch.from_numpy(img).float()
    vol = torch.unsqueeze(vol, 0)


    labels = torch.tensor(self.data.loc[
    idx, ['epidural', 'intraparenchymal', 'intraventricular', 'subarachnoid', 'subdural',
'any']])
    out_lables =labels.float()
    return {'image': vol, 'labels': out_lables}
```

Then, we defined the chosen: the ResNet18 pretrained model from MedicalNet.

Before defining the model's parameters, we needed to modify the last layer to fit our specific task. This pretrained model is used for Segmentation, and its last layer, *self.conv_seg* was implemented like this:

```
self.layer4 = self._make_layer(
    block, 512, layers[3], shortcut_type, stride=1, dilation=4)

"""self.conv_seg = nn.Sequential(
                               nn.ConvTranspose3d(512 * block.expansion, 32, 2, stride=2),
                               nn.BatchNorm3d(32),nn.ReLU(inplace=True),
                               nn.Conv3d(32, 32 ,kernel_size=3, stride=(1, 1, 1), padding=(1, 1, 1), bias=False),
                               nn.BatchNorm3d(32),nn.ReLU(inplace=True),
                               nn.Conv3d(32, num_seg_classes, kernel_size=1,stride=(1, 1, 1),bias=False))"""
```

Figure 6.6 MedicalNet last layer implementation

We replaced this last layer with a fully connected layer with six out features, with a Sigmoid activation function, as well as a dropout, just like we did for our 2DCNN Model.

```
self.fc = nn.Sequential(
        nn.Dropout(0,3),
        nn.Linear(512*block.expansion, 6, bias= True),
        nn.Sigmoid()
)
```

For model's parameters definition they provide with a .py file named *'setting.py'* in which we find a method called *'parse_opts()'*. Usind this method we parse the arguments that we want for our model.

```
sets = parse_opts()
sets.gpu_id =  [torch.cuda.device(i) for i in range(torch.cuda.device_count())]


sets.n_epochs = 5
sets.no_cuda = False #use cuda or not
sets.data_root = '../rsna-intracranial-hemorrhage-detection/3d_scans/'
sets.pretrain_path = 'MedicalNet/MedicalNet_pytorch_files2/pretrain/resnet_18_23dataset.pth'
sets.num_workers=  0
sets.model_depth = 18
sets.resnet_shortcut = 'A'
sets.input_D = 60
sets.input_H = 256
sets.batch_size = 1
sets.input_W = 256
sets.fold_index =1

sets.save_folder
=r'models_{}_{}_{}_fold_{}'.format('resnet',sets.model_depth,sets.resnet_shortcut,sets.fold_in
dex)
```

Here we defined that we want our model to run for five epochs, using cuda. Data_root is the path where the model will get his images from. Pretrain_path represents the path of the pre-trained model provided by MedicalNet. The resnet_shortcut for the ResNet18 model is 'A'. Then we modeled our input, 60 being the maximum number of slices from our dataset, which will be our depth.

Now, the last step we have to make is to generate the model based on these settings. We can simply do so by using the generate_model method found in MedicalNet.model.

```
torch.manual_seed(sets.manual_seed)
model, parameters = generate_model(sets)
```

Now comes the training part. For each batch, we printed the loss and the average time the model spent learning, as well as the outputs of the model and target labels, for noticing the predictions. At some points, we saved the model and printed its state. After each epoch, we switched the model to evaluation mode and calculated the loss and accuracy for the validation data.

```
tensor([[0.4373, 0.5029, 0.5397, 0.5628, 0.5114, 0.4917]], device='cuda:0',
       grad_fn=<SigmoidBackward>)
tensor([[0., 0., 0., 0., 0., 0.]], device='cuda:0')
2021-06-04 12:57:33 INFO     [new_3d.py:73] Batch: 0-0 (0), loss = 0.712, avg_batch_time = 2.758
```

Figure 6.7 Printed batch output

For the 3D model that takes as input five neighboring slices, we created 'FiveSlices_Dataset' for our dataset class similar to the ones we have already talked about. We created a data frame containing all the DICOM files that were from the same scan as our slice. To make it easier to get the data that we want, we reset its indexes. The variable *'new_idx* corresponding to our slice's index from this data frame. To extract five needed slices, we simply get the ones that are located on the positions *new _idx-2* to *new_idx+2* in our ordered scan. For the first and the last two slices from a scan, we use arrays of zeros as the missing neighbors. Then we used the correction to HU, normalization, zero centering, and downsizing that we talked about in *Chapter 6.2.2*.

```
def __getitem__(self, idx):
    try:
        predict = []
        slice = self.data.loc[idx, 'Image']
        scan = self.data.loc[idx, 'SeriesInstanceUID']
        slices = self.data[self.data['SeriesInstanceUID'] ==
scan].sort_values(by=['ImagePositionPatient2'])
        slices = slices.reset_index(drop=True)
        idx_list = slices.index[slices['Image'] == slice].to_list()
```

```
        new_idx = idx_list[0]
        for i in range(-2,3):
            if new_idx - i >= 0 and new_idx + i < len(self.data):
                s = slices.loc[new_idx - i, 'Image']
                dcm = pydicom.dcmread(self.path +s+'.dcm')
                if (dcm.BitsStored == 12) and (dcm.PixelRepresentation == 0) and
(int(dcm.RescaleIntercept) > -100):
                    correct_dcm(dcm)
                predict.append(dcm.pixel_array)
            else:
                imgs = np.zeros((512, 512))
                predict.append(imgs)


    except:
        predict = np.zeros((5, 512, 512), dtype=np.int16)
    augmented = []
    if self.transform:
        for i in range(len(predict)):
            augment = self.transform(image=predict[i])
            augmented.append(augment['image'])
        image = np.stack([s for s in augmented])
        vol = torch.from_numpy(image).float()
    else:
        image = np.stack([s for s in predict])
        vol = torch.from_numpy(image).float()
```

Besides this, we modeled our network to have a depth of five instead of 60 and take as input batches of 16.

```
sets.input_D = 5
sets.input_H = 256
sets.batch_size = 16
sets.input_W = 256
```

Conclusions

In this thesis we presented the comparison between two different approaches when it comes to brain hemorrhage detection: a slice-by-slice one where our shallow 2D model was fed a large amount of data, and a scan approach where our 3D model received four times fewer data but was pretrained on medical imaging. For both of our models, we followed different preprocessing steps suitable for each type of data, then trained them separately. They both detect hemorrhages and classify them into subtypes jointly.

This thesis was designed to compare both of these shallow network's performances without a lot of complication when it comes to feature selection. Our goal was solely to compare the two, not so much to give an accurate diagnosis by complicating the architecture chosen. However, we can say that the 2D Model has surpassed our expectations with an 89.9% strict accuracy.

Also, to our surprise, the difference between the two models was significant, the 2D model outperformed by far the 3D one. Even though it had four times fewer data for fine-tuning, the fact that it was pretrained on medical images and that it takes into consideration neighboring slices should have been an advantage. However, just as humans can hardly see five different hemorrhage types by looking at a hole 3D Scan, we might say that this task was difficult for our model as well. We considered that splitting a problem into multiple little tasks is easier than trying to solve it as a whole. Having that in mind, we fed our 3D model with only five consecutive slices, instead of the whole scan, which has significantly improved the performance, but it has not reached the level of the 2D model. With how our models have performed, it could seem that the need for large amounts of data for this task cannot be compensated by computing power. However, this remains inconclusive, as further development must be made to the 3D Model to give that statement.

We conclude this thesis with the fact that future additional work should be performed towards improving the 3D network’s performance. The approach where we used five consecutive slices has managed to get us closer to the 2D’s results. We will try to implement the same CNN 3D architecture pre-trained on the Kinetics dataset, to which we will feed all available data in the dataset to see if its performance changes in a positive way. In addition, we want to develop an approach where we look at every slice level to split our task (the hole scan) into multiple problems, then come back in the 3D form to consider neighboring slices as well for a better diagnosis.

Tencent also announced that they are going to update MedicalNet and there will be 2D ResNet models, pretrained on medical imaging, available. We are curious how they would perform at our task as well.

References

[1] A. Géron, Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems, O'Reilly Media, 2019.

[2] D. H. Hubel, "Single unit activity in striate cortex of unrestrained cats," *The Journal of physiology 147.2* , pp. 226-238, 1959.

[3] D. H. Hubel and T. N. Wiesel., "Receptive fields of single neurones in the cat's striate cortex.," *The Journal of physiology 148.3,* pp. 574-591, 1959.

[4] J. O. Neill and D. Bollegala, "Transfer Reward Learning for Policy Gradient-Based Text Generation," *arXiv preprint arXiv:1909.03622,* 2019.

[5] S. Saha, "Towards Data Science," 15 12 2018. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. [Accessed 08 03 2021].

[6] S. Univerity, "CS231n: Convolutional Neural Networks for Visual Recognition," 2021. [Online]. Available: https://cs231n.github.io/convolutional-networks/. [Accessed 08 03 2021].

[7] S. Indolia, "Conceptual understanding of convolutional neural network-a deep learning approach.," *Procedia computer science 132,* pp. 679-688, 2018.

[8] Y. LeCun, "Gradient-based learning applied to document recognition.," *Proceedings of the IEEE 86.11,* pp. 2278-2324, 1998.

[9] W. contributors, "Wikipedia, The Free Encyclopedia," [Online]. Available: https://en.wikipedia.org/w/index.php?title=LeNet&oldid=1021579098. [Accessed 29 05 2021].

[10] A. Krizhevsky, I. Sutskever and . E. Geoffrey, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems 25,* pp. 1097-1105, 2012.

[11] O. Russakovsky, "Imagenet large scale visual recognition challenge," *International journal of computer vision 115.3,* pp. 211-252, 2015.

[12] S. SaxenaShipra, "Analytics Vidhya," 30 03 2021. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/03/introduction-to-the-architecture-of-alexnet. [Accessed 29 05 2021].

[13] C. Szegedy, "Going deeper with convolutions," *Proceedings of the IEEE conference on computer vision and pattern recognition.*, 2015.

[14] S. Tsang, "Medium," 24 08 2018. [Online]. Available: https://medium.com/coinmonks/paper-review-of-googlenet-inception-v1-winner-of-ilsvlc-2014-image-classification-c2b3565a64e7. [Accessed 29 05 2021].

[15] V. Feng, "Towards Data Science," 15 07 2017. [Online]. Available: https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035. [Accessed 30 05 2021].

[16] K. He, X. Zhang, S. Ren and J. Sun, "Deep residual learning for image recognition," *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770-778, 2016.

[17] C. Sunway, "Programmer Sought," 03 04 2020. [Online]. Available: https://www.programmersought.com/article/63014851723/. [Accessed 06 06 2021].

[18] Y. Zheng, C. Quanqi and Z. Yujin, "Deep learning and its new progress in object and behavior recognition," *Journal of image and graphics*, vol. 19.2, pp. 175-184, 2012.

[19] N. S. Singh, "Facial recognition using deep learning.," *Advances in Data Sciences, Security and Applications*, pp. 375-382, 2020.

[20] Y. Jiang, "DeepProduct: Mobile product search with portable deep features," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 14.2, pp. 1-18, 2018.

[21] D. Yu and . L. Deng, "Deep learning and its applications to signal and information processing [exploratory dsp]," *IEEE Signal Processing Magazine*, vol. 28.1, pp. 145-154, 2010.

[22] Z. Wang, "The applications of deep learning on traffic identification," *BlackHat USA*, vol. 24.11, pp. 1-10, 2015.

[23] A. Luckow, "Deep learning in the automotive industry: Applications and tools," in *2016 IEEE International Conference on Big Data (Big Data)*, IEEE, 2016.

[24] D. Shen, G. Wu and H. Suk, "Deep learning in medical image analysis.," *Annual review of biomedical engineering,* vol. 19, pp. 221-248, 2017.

[25] M. Burduja, R. T. Ionescu and V. Nicolae, "Accurate and Efficient Intracranial Hemorrhage Detection and Subtype Classification in 3D CT Scans with Convolutional and Long Short-Term Memory Neural Networks," *Sensors 20.19,* p. 5611, 2020.

[26] P. D. Chang, "Hybrid 3D/2D convolutional neural network for hemorrhage evaluation on head CT," *American Journal of Neuroradiology 39.9* , pp. 1609-1616, 2018.

[27] M. Grewal, "Radnet: Radiologist level accuracy using deep learning for hemorrhage detection in ct scans.," in *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018).*, IEEE 2018, 2018.

[28] W. Kuo, "Expert-level detection of acute intracranial hemorrhage on head computed tomography using deep learning," *Proceedings of the National Academy of Sciences 116.45*, pp. 22737-22745, 2019.

[29] S. P. Singh, "Shallow 3D CNN for detecting acute brain hemorrhage from medical imaging sensors," *IEEE Sensors Journal* , 2020.

[30] L. Hyunkwang, M. Kim and S. Do, "Practical window setting optimization for medical image deep learning," *arXiv preprint arXiv:1812.00572,* 2018.

[31] N. T. Nguyen, "A CNN-LSTM Architecture for Detection of Intracranial Hemorrhage on CT scan," *rXiv preprint arXiv:2005.10992,* 2020.

[32] K. Saab, "Doubly Weak Supervision of Deep Learning Models for Head CT," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, Springer, Cham, 2019.

[33] T. Phong, H. Duong, H. Nguyen, N. Trong, V. Nguyen, T. Van Hoa and V. Snasel, "Brain hemorrhage diagnosis by using deep learning," in *2017 International Conference on Machine Learning and Soft Computing*, Ho Chi Minh City, Vietnam, 13–16 January 2017.

[34] O. Russakovsky, "Imagenet large scale visual recognition challenge," *International journal of computer vision, 115(3)*, pp. 211-252, 2015.

[35] H. Salehinejad, "A Real-World Demonstration of Machine Learning Generalizability: Intracranial Hemorrhage Detection on Head CT," *arXiv preprint arXiv:2102.04869*, 2021.

[36] A. Majumdar, "Detecting intracranial hemorrhage with deep learning.," in *2018 40th annual international conference of the IEEE engineering in medicine and biology society (EMBC)*., IEEE, 2020.

[37] [Online]. Available: https://steelkiwi.com/blog/python-for-ai-and-machine-learning/. [Accessed 30 05 2021].

[38] W. contributors, "Wikipedia, The Free Encyclopedia," 16 05 2021. [Online]. Available: https://en.wikipedia.org/wiki/PyCharm. [Accessed 31 05 2021].

[39] "ComponentSource," [Online]. Available: https://www.componentsource.com/product/pycharm/about. [Accessed 31 05 2021].

[40] W. contributors, "Wikipedia, The Free Encyclopedia," 2021 04 2021. [Online]. Available: https://en.wikipedia.org/wiki/DICOM. [Accessed 30 05 2021].

[41] "DICOM Standard," [Online]. Available: https://www.dicomstandard.org/about-home. [Accessed 30 05 2021].

[42] D. Mason, "pydicom: An open source DICOM library," [Online]. Available: https://github.com/pydicom/pydicom. [Accessed 30 04 2021].

[43] D. Mason, "PyPi," [Online]. Available: https://pypi.org/project/pydicom/#data. [Accessed 31 05 2021].

[44] [Online]. Available: https://en.wikipedia.org/wiki/PyTorch. [Accessed 31 05 2021].

[45] E. Stevens, L. Antiga and T. Viehmann, "Introducing deep learning and the PyTorch Library," in *Deep Learning with PyTorch: Build, train, and tune neural networks using Python tools*, Manning, pp. 3-14.

[46] E. Stevens, L. Antiga and T. Viehmann, "Tensors: Multidimensional arrays," in *Deep Learning with Pytorch*, Manning, pp. 42-26.

[47] "PyTorch," [Online]. Available: https://pytorch.org/docs/stable/data.html. [Accessed 31 05 2021].

[48] Kaggle, "Kaggle," [Online]. Available: https://www.kaggle.com/c/rsna-intracranial-hemorrhage-detection/overview. [Accessed 22 05 2021].

[49] A. E. Flanders, "Construction of a machine learning dataset through collaboration: the RSNA 2019 brain CT hemorrhage Challenge," *Radiology: Artificial Intelligence 2.3,* vol. e190211, 2020.

[50] M. Mustra, D. Kresimir and M. Grgic, "Overview of the DICOM standard," *50th International Symposium ELMAR,* vol. 1, 2008.

[51] W. contributors, "Hounsfield scale. Wikipedia, The Free Encyclopedia.," 22 03 2021. [Online]. Available: https://en.wikipedia.org/wiki/Hounsfield_scale. [Accessed 07 06 2021].

[52] "TorchVision.Models," [Online]. Available: https://pytorch.org/vision/stable/models.html. [Accessed 02 06 2021].

[53] "PyTorch ResNet," [Online]. Available: https://pytorch.org/hub/pytorch_vision_resnet/. [Accessed 02 06 2021].

[54] Tencent. [Online]. Available: https://www.tencent.com/en-us/about.html. [Accessed 04 06 2021].

[55] S. Chen, K. Ma and Y. Zheng, "Med3d: Transfer learning for 3d medical image analysis.," *arXiv preprint arXiv:1904.00625,* 2019.

[56] MRBrainS18. [Online]. Available: https://mrbrains18.isi.uu.nl/data/. [Accessed 04 06 2021].

[57] R. Gomez, "Raul Gomez blog," 23 05 2018. [Online]. Available: https://gombru.github.io/2018/05/23/cross_entropy_loss/. [Accessed 06 06 2021].

[58] D. Godoy, "Towards Data Science," 21 11 2018. [Online]. Available: https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a. [Accessed 05 06 2021].

[59] P. contributors, "Peltarion," [Online]. Available: https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/binary-crossentropy. [Accessed 06 06 2021].

[60] M. M. Arat, "Mustafa Murat Arat," 25 01 2020. [Online]. Available: https://mmuratarat.github.io/2020-01-25/multilabel_classification_metrics. [Accessed 04 06 2021].

[61] "Full Preprocessing Tutorial," [Online]. Available: https://www.kaggle.com/gzuidhof/full-preprocessing-tutorial. [Accessed 03 06 2021].

[62] "Data Science Bowl 2017," [Online]. Available: https://www.kaggle.com/c/data-science-bowl-2017. [Accessed 03 06 2021].