Universitat Oberta de Catalunya

Open University of Catalonia (UOC)

University Master's Degree in Data Science

MASTER'S THESIS

Area: Medical

Study of brain magnetic resonance images reconstruction through convolutional autoencoders

Author: Andrés Pardo Ginés

Supervisor: Dr Baris Kanber

Professor: Dr Ferran Prados Carrasco

Tokyo, July 9, 2020

Copyright

FICHA DEL TRABAJO FINAL

Título del trabajo:	Study of brain magnetic resonance images reconstruction through convolutional autoencoders
Nombre del autor:	Andrés Pardo Ginés
Nombre del colaborador/a docente:	Baris Kanber
Nombre del PRA:	Ferran Prados Carrasco
Fecha de entrega (mm/aaaa):	06/2020
Titulación o programa:	Máster en Ciencia de Datos
Área del Trabajo Final:	Medical
Idioma del trabajo:	Inglés
Palabras clave	Deep Learning, Autoencoder, MRI

Abstract

We studied how a variety of 2D convolutional autoencoders of different sizes performed the task of reconstructing healthy brain magnetic resonance images. The images used were the subset of T1-weighted MRI available in the IXI dataset. Two smaller ResNet-like models were created and tested to see how the number of parameters in the model affected the reconstruction prowess of the autoencoders. The popular ResNet50 model was also tested under two configurations: no pre-trained weights and pre-trained weights trained on the ImageNet dataset, to see if the transfer learning process from an unrelated computer vision task help improve the reconstruction results. All models were trained using mean square error and difference of structural similarity as loss function to explore if the later would assist the models in improving their performances as some publications have pointed out for a range of different computer vision tasks. Both the peak signal-to-noise ratio and structural similarity tests showed that the use of difference of structural similarity as loss function did provide the best results in the test set for models using no pre-trained weights, globally the results of the ResNet-like models were effectively indistinguishable from the original images.

Keywords: Deep Learning, Autoencoder, MRI

Resum

Hem estudiat com diversos autoencoders convolucionals 2D de mides diferents duen a terme la tasca de reconstruir imatges de cervells obtingudes via ressonància magnètica. Les imatges fetes servir provenen del subconjunt d'IRM T1-weighted del conjunt de dades IXI. Van ser creats i testats dos models de mida reduïda basats en ResNet per veure com el nombre de paràmetres a un model afecta a la qualitat de les reconstruccions. El conegut model ResNet50 també va ser testat fent servir dues configuracions: sense pesos prèviament entrenats i fent servir pesos entrenats en el conjunt de dades ImageNet per a veure si el procés de transferència de l'aprenentatge des d'una tasca de visió per computador sense relació amb l'actual podia ajudar a millorar la qualitat de les reconstruccions. Tots els models van ser entrenats fent servir error quadràtic mig i diferència de semblances estructurals com a funcions de pèrdua per a explorar si aquesta última podia ajudar als models a millorar el seu rendiment com algunes publicacions han indicat per a tasques de visió per computador diferents. Les proves fent servir tant proporció màxima senyal-soroll com semblança estructural mostren com l'ús de la diferència de semblances estructurals com a funció de pèrdua proporciona els millors resultats en el conjunt de prova pels models que no fan servir pesos prèviament entrenats, globalment els models de mida reduïda basats en ResNet van aconseguir uns resultats pràcticament indistinguibles de les imatges originals.

Paraules clau: Aprenentatge Profund, Autoencoder, IRM

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

Pathology and artifact-free reconstruction of brain magnetic resonance images is the first step towards achieving unsupervised anomaly detection. Unsupervised anomaly detection would be helpful as there is no need to obtain and annotate images of brains with pathologies in order to detect them in an unsupervised approach.

Image reconstruction becomes the first step of the process as unsupervised detection of anomalies would be carried out in the following manner: a model trained using healthy brain images learns how to reconstruct structures of normal, healthy brains. Then, when that model gets as input an image containing an anomaly it would not be able to reproduce it correctly, so a subtraction between the two images would highlight the areas affected by the anomaly.

1.2 Goals

The primary goal of this work is to achieve a good quality image reconstruction of a healthy individual's brain magnetic resonance images. Fulfilling a good quality reconstruction reduces the noise and acquisition artifacts in the images.

In order to achieve our main goal a variety of convolutional autoencoders were trained and tested on the IXI dataset [21], as the dataset is a collection of almost 600 brain magnetic resonance images from healthy individuals.

Three different backbones were used: a small strided convolutional model based on ResNet, a deeper version of the previous model and the standard ResNet50 available in *keras.applications*.

Two different loss functions were tested to train the three models, mean squared error and structural similarity. For the ResNet50 model two different configurations were used: no pretrained weights or weights pretrained on the ImageNet dataset. Fine-tuning over pretrained ImageNet weights is a standard technique in both the academia and industry that is why we explored its effects in this study.

1.3 Methodology

As this research project is a study comparing different convolutional autoencoders image reconstruction prowess a simple iterative methodology was used to develop it. After an initial data exploration phase were the needed tools to read and pre-process the data were obtained, a minimum working version of the training code was created. From that point it was just a process of simply add options to the training code: different backbones, loss functions, optimizers, etc. When the first results were available a similar process was done to visualize and test the trained models. Once both basic training and test code were mature enough the rest of the time as spend training and expanding the optional methods featured in the final code.

1.4 Research planning

The Research planning based on the original deadlines was designed to last two months, from March 23^{rd} 2020 to May 23^{rd} 2020. The proposed research plan was:

- Develop an initial convolutional autoencoder and train it from scratch on the IXI dataset.
- Test structural similarity as a loss, train a bigger convolutional autoencoder from scratch.
- Add ResNet50 to the project, train it from scratch on the IXI dataset, also train it using the ImageNet weights and compare the results.
- Finish the previous steps and test as many of the secondary objectives as possible.

Due to the pandemic caused by SARS-CoV-2 the deadline for the implementation phase was extended by two weeks. We employed that extra time to re-train some of the models, add some extra optional parameters to the training and test code and expand and re-write parts of the README file.

Chapter 2

State of the Art

The introduction of AlexNet [18] back in September of 2012 accelerated the adoption of convolutional neural networks and, more broadly, deep learning in computer vision research and applications. The AlexNet model won both the classification and localization task of the Large Scale Visual Recognition Challenge 2012 [28], commonly known as ImageNet, reducing the top-5 error rate to 15.3% compared to the second-best entry that achieved a 26.2% error rate.

The architecture of the model is described in detail in Table 2.1. It consists of 5 convolutional layers, some of those followed by max pooling layers to down-sample the representations and, finally, three dense layers ending on a softmax function that outputs the probability distribution of the inputted classes. AlexNet uses rectified linear unit (ReLU) as activation function for all layers but the last one where a softmax is used. ReLU helps train deep neural networks more effectively than prior activation functions as demonstrated by Glorot et al. [10] in 2011.

In this section we will review the evolution of deep learning computer vision models, the commoditization of deep learning and the inception of transfer learning as a useful pretraining technique before talking about how deep learning is being used to solve brain magnetic resonance image reconstruction and anomaly detection.

2.1 Beyond AlexNet

The described basic architecture stacking convolutional layers using ReLU as activation function, pooling and dense layers ending on a softmax activation function became the de facto standard for computer vision classification problems, but subsequent models refined it over time. Some popular models include winners of following ImageNet competitions like ZFNet [40], VGG [32], Inception [34] or ResNet [12]. These newer models introduced a series of im-

Layer	Tensor size	Kernel Size	Stride	Parameters
Input	227x227x3	0	0	0
Convolution 1	55x55x96	11x11	4	34,944
Max pooling	27x27x96	3x3	2	0
Convolution 2	27x27x256	5x5	1	614,656
Max pooling	13x13x256	3x3	2	0
Convolution 3	13x13x384	3x3	1	885,120
Convolution 4	13x13x384	3x3	1	1,327,488
Convolution 5	13x13x256	3x3	1	884,992
Max pooling	6x6x256	3x3	2	0
Dense 1	4096x1	-	-	37,752,832
Dense 2	4096x1	-	-	16,781,312
Dense 3	4096x1	-	-	4,097,000

Table 2.1: *The original AlexNet architecture.*

provements to the described AlexNet architecture. ZFNet reduced the filter size from 11x11 to 7x7 and the stride from 4 to 2. VGG pushed the number of layers to 16-19 while reducing the filter size of the convolutions further to 3x3 using a stride of 1. VGG also included 1x1 filters to add extra nonlinearity. The original Inception model increased the number of layers again to 22 while introducing the inception module. The inception module performs convolutions on the same input but using different filter sizes (5x5, 3x3 and 1x1) and, additionally, a 3x3 max pooling operation. The results of all those operations are concatenated and passed to the next inception module.

The introduction of residual learning in late 2015 [12], made training architectures larger than ever possible. Soon different architectures started to adopt ResNet-like features like Inception did with the Inception-ResNet model [33] in 2016. After that residual learning became a staple in many other deep learning architectures. The 2017 winner of the ImageNet classification competition SeNet [13], for Squeeze-and-Excitation network , used residual learning on top of their own contribution, the SE block, to reduce the top-5 error of their submission to just 2.251%.

The increasing number of layers and parameters included in those models translates into neural networks that can represent more complicated functions. As the ImageNet dataset contains a large amount of annotated data, detailed in in Table 2.2, the error rates for the different competition tasks have been steadily improving thanks to the architectures discussed.

Training images	Validation images	Test images	Number of classes
1,281,167	50,000	100,000	1000

Table 2.2: *ImageNet dataset of 2012.*

2.1.1 ResNet

As introduced in the prior section, the concept of Residual Networks (ResNet) made training of very deep neural networks possible. Before its introduction, networks could not be further extended from a couple of tens of layers.

When ResNets were introduced two different problems, vanishing and exploding gradients, that made very deep neural networks not trainable were fixed through normalization. Solving the vanishing and exploding gradients problem made deeper neural networks able to converge when training. That exposed a new problem of training accuracy degradation, as networks got deeper the accuracy saturated and then decreased quickly. Shallower models had better accuracy than deeper ones as seen in Figure 2.1.

The authors introduced residual learning to overcome this problem, in their own words 'Instead of hoping each few stacked layers directly fit a desired underlying mapping, we explicitly let these layers fit a residual mapping.'. This was formally defined as: $\mathcal{H}(x)$ is a mapping to be fit by a group of layers, where x are the inputs of the first layer of the group. As the authors hypothesized that multiple nonlinear layers can asymptotically approximate complicated functions, that is equivalent to hypothesize that they can asymptotically approximate the residual of those functions, defined as $\mathcal{H}(x) - x$. Then they set the group of layers to explicitly approximate a residual function $\mathcal{F}(x) := \mathcal{H}(x) - x$, making the original function $\mathcal{F}(x) + x$. The authors explained that function can be implemented in a neural network as a shortcut connection that

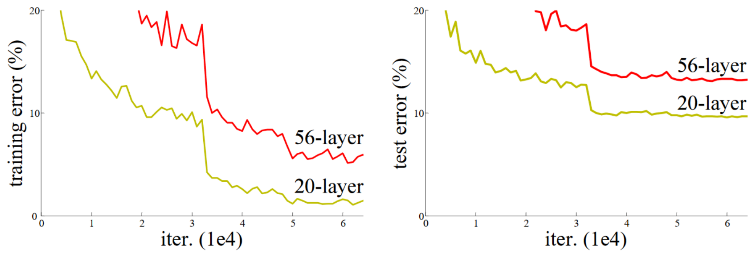

Figure 2.1: Train and test error on the CIFAR-10 dataset of two plain networks with 20 and 54 layers respectively. Included in the original publication [12]

links the inputs of the group of layers $\mathcal{F}$ with its output. That change made deeper neural networks trainable using the standard combination of stochastic gradient descent (SGD), or a similar optimization algorithm, with back-propagation. A detailed image of a building block using shortcut connections implemented in ResNet34 can be seen in the left side of the Figure 2.2. A complete comparison between VGG-19 and the ResNet34 model can be seen in Figure 2.3.

The authors increased ResNet beyond 34 layers, showcasing models using 50, 101 and 152 layers respectively. For those deeper models the authors modified the building block made of two convolutional layers and a shortcut connection used in ResNet34 to a three layer "bottleneck" building block as seen in the right side of the Figure 2.2. The layers in the new building block are a 1x1, 3x3 and 1x1 convolutions respectively. The first 1x1 convolution reduces the dimensions of the input, thus creating a bottleneck, before being restored by the later 1x1 convolution. Via an ensemble of two ResNet152 models, built using the "bottleneck" building block, the top-5 error rate for the classification competition in ImageNet was reduced to 3.57%. The authors won not only the ImageNet classification task but also the ImageNet detection, ImageNet localization and both the segmentation and detection competitions at the Common Objects in Context (COCO) Challenge [20].

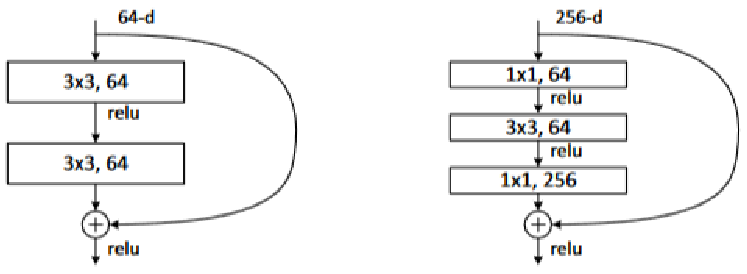

Figure 2.2: Comparison between a residual building block for a network like ResNet34 shown in Figure 2.3 and a bottleneck building block for larger networks like ResNet50, 101 or 152 as shown in the original publication [12]

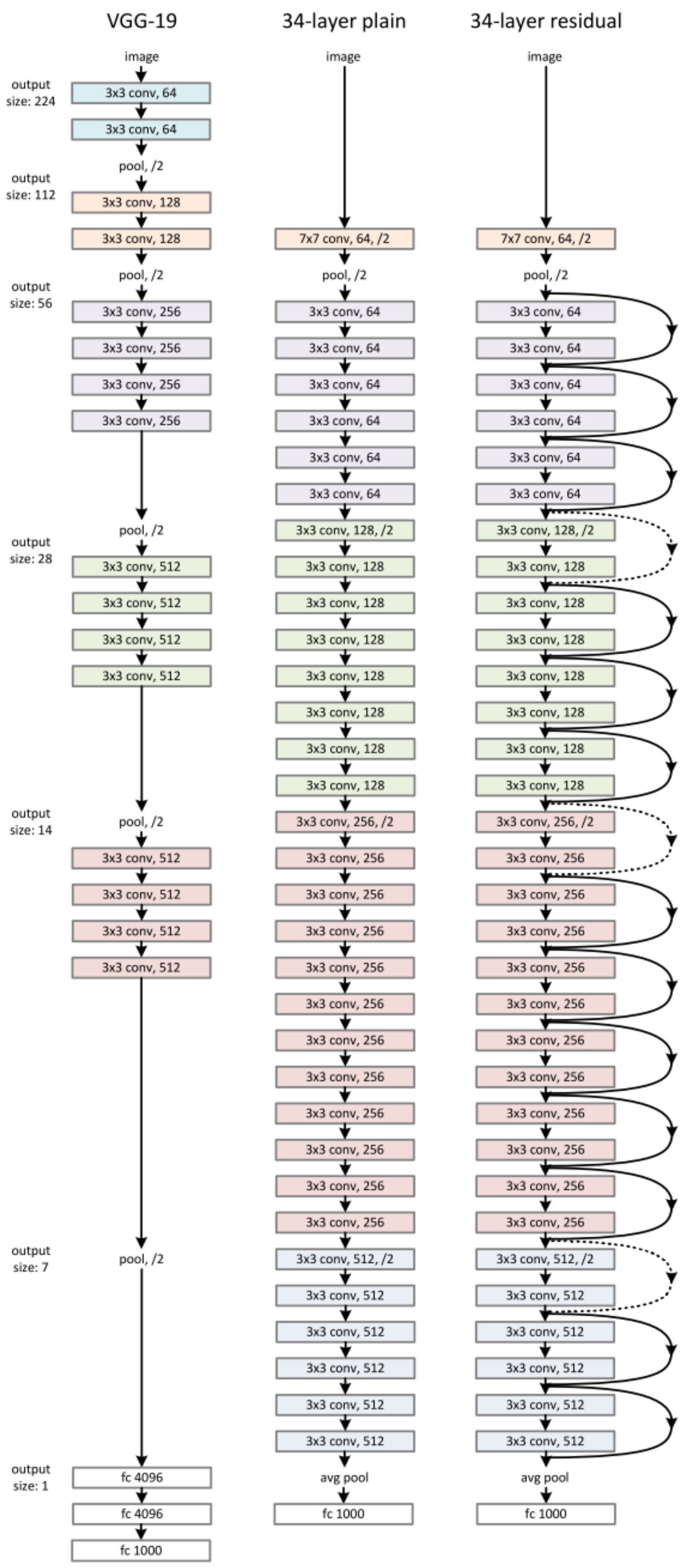

Figure 2.3: Comparison between VGG-19, a 34-layer plain network and ResNet34 as shown in the original publication [12]

2.1.2 The Commoditization of Deep Learning

At the same time that AlexNet was presented its authors made public cuda-convnet [17], their C++/CUDA implementation of convolutional neural networks able to run in graphics processing units (GPUs). Different research teams used cuda-convnet to implement their own networks, and by the summer of 2013 the Berkeley Artificial Intelligence Research group released Decaf [8]. A project able to extract features for generic visual recognition tasks that relied on cuda-convnet to train the network on a GPU. By December 2013 the same group released the Caffe [14] framework providing a full toolkit for deep learning development in C++ with a Python interface. Caffe let developers define, train and deploy neural networks on GPUs, and, later on added support for CPUs, multiple parallel GPU training, etc. The open-source framework was quickly adopted by the community and in September of 2014 the Caffe Model Zoo was created to enable sharing of pretrained models between different academic institutions and the industry. Those pretrained models can be used by other users out-of-box. By 2015 Google released Tensorflow [1], Microsoft did the same with their own framework called CNTK [30], and other smaller industrial players like Preferred Networks published their own deep learning development kits, Chainer [36].

Before this move to the industry most deep learning framework and libraries originated in the academia, like Theano [35], developed mainly by the Montreal Institute for Learning Algorithms of the University of Montreal, or Torch which is a BSD license machine learning library for the Lua language that was updated to support training of deep learning models in both CPUs and GPUS.

François Chollet released Keras [6] in 2015 to enable fast experimentation with deep neural networks in a standard interface that can run on top of either Tensorflow, Theano or, later on, CNTK.

Recently the deep learning framework landscape has consolidated. In 2017, Theano stopped being actively developed. By 2019, Caffe2, which was released by Facebook in 2017, was merged into PyTorch [24] and both Microsoft's CNTK and PFN's Chainer stopped receiving major updates. Tensorflow 2.0 release made the library easier to use and also improved its integration of Keras in the library, furthering its position in the community.

To summarize, deep learning development moved to Python-interfaced libraries with constant updates supported by big industry players. Nowadays is mostly based around Tensorflow, as it is or through its Keras interface, or PyTorch. Pretrained models using standard architectures are shared by researchers and official models zoos are supported by both the academia and the industry. Those factors made transfer learning not just possible but a common way of tackling tasks where few labeled data is available.

2.1.3 Transfer Learning

Transfer learning is a technique that consists of two steps: pretrain the network on a large dataset and then fine-tune it for the target task it has to fulfill by training using a specific dataset for that task. Transfer learning helps the network to increase its accuracy.

Both the adoption of standard deep learning models like VGG or ResNet and the need to apply them to specialized areas where datasets contain far less labeled data than ImageNet made transfer learning a widely used technique to achieve good results in a broader range of fields. Another benefit of transfer learning is that speeds-up the training process as the network initial convolutional layers already know how to detect simple features like lines and don't have to learn them from scratch. Faster training and improved accuracy made using pretrained weights using ImageNet a common technique in deep learning.

Currently used deep learning frameworks like PyTorch or Tensorflow let developers easily import a predefined standard model and then select if the pretrained weights in ImageNet should be used or not. The simplification of the sharing process makes training big models on a small data set feasible, thus we will explore how much it helps to use ImageNet pretrained weights on ResNet50 when training on the IXI Dataset.

2.2 Brain Magnetic Resonance Imaging

There is a competition in the same vein that ImageNet or COCO Challenge using labeled brain magnetic resonance imaging, the Brain Tumor Segmentation Challenge or BRATS. [23]. The first edition of the challenge included a dataset of just 65 patients, providing 4 different scans of each of them took using MR scanners from different vendors and different field strengths. The scan formats included were T1-weighted, T1 with gadolinium enhancing contrast, T2-weighted and FLAIR. In Brain Tumor Segmentation Using Convolutional Neural Networks in MRI Images, Pereira et al. used the BRATS 2013 dataset and the expanded 2015 version to show how a simple CNN-based solution using 3x3 kernels and less than 12 layers was able to best all other solutions until that point for the 2013 Challenge and making it second in the 2015 one by focusing in data pre-processing and data augmentation techniques.

Models described up until this point were created to solve supervised learning tasks, where in order to train a model there is a labeled dataset that contains examples of the desired output, hence becoming a supervised tasks. The end goal of the research introduced here is to achieve unsupervised anomaly detection in brain magnetic resonance images. In order to do that a three steps process is needed, train a model that can reconstruct healthy brain magnetic resonance images, insert a brain image with an anomaly in the prior model and be able to detect the anomalies by subtracting the pixels of the reconstructed imaged minus the original input image. There are two popular methods to build models that can reconstruct input images in deep learning, autoencoders and generative adversarial networks(GAN). Both methods will be introduced and some examples of their use in the field will be showcased in the subsequent subsections.

2.2.1 Autoencoders

Autoencoders are a subcategory of unsupervised neural networks where the objective of the model is to learn to copy the received input to its output. Autoencoders are used to learn a representation of the inputted data, and in order to achieve that they consist on two parts: an encoder and a decoder. The encoder's role is to reduce the data until it achieves a compact representation of it. The decoder's role is to reconstruct the input data from the compact representation built by the encoder. Using the output of the decoder as a supervisor for the loss it is possible through back-propagation to train both parts of the autoencoder at the same time. A schematic representation of an autoencoder can be seen in Figure 2.4. Autoencoders were designed for dimensionality reduction [16] and are used for data representation [37]. Deep convolutional autoencoders have been used in many different tasks related to image reconstruction, from enhancing of low-light and noisy images [22] to super-resolution [19]. Variational autoencoders follow the same general pattern than general autoencoders but can be used as generative models.

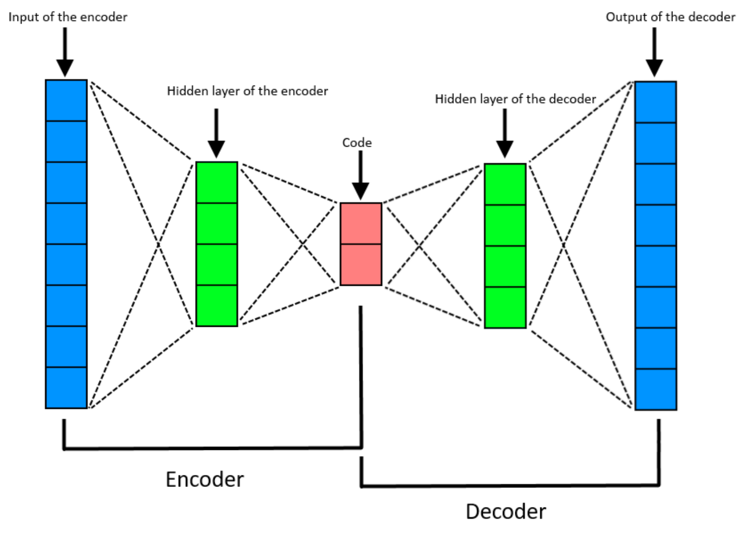

Figure 2.4: Autoencoder basic architecture showing the encoder and decoder parts

In Unsupervised Detection of Lesions in Brain MRI using constrained adversarial autoencoders [5], Chen et al. used T2-weighted structural images from the Human Connectome Project as training data. The set contained data from 35 healthy individuals. As test data they used T2-weighted images from the Multimodal Brain Tumor Image Segmentation (BRATS) Challenge 2015 from 42 subjects. All the images were reduced to 32x32 pixels prior to being used. Then the authors compared the results of both variational and adversarial autoencoders and saw both types were able to achieve unsupervised anomaly detection. Additionally, using a latent constraint as the one proposed in the publication helped improve the results.

2.2.2 Generative adversarial networks

Goodfellow et al. introduced GANs in the article Generative Adversarial Nets in 2014 [11]. Originally introduced as unsupervised learning generative models GANs can be modified to follow a supervised learning, semi-supervised learning or, even, a reinforcement learning approach. In a similar vein to autoencoders, GANs are split into two parts but this time they are complete neural network models. One of the two models, called generative network, is in charge of generating results that increase the error rate of the other network, the discriminative network. That network's objective is to evaluate the generative network's output. This complex setting makes GANs difficult to train effectively.

In 2017 a novel approach to unsupervised anomaly detection using GANs was presented, AnoGAN [29]. For this work the authors extracted 1 million 64x64 pixels 2D images of healthy human retina extracted from high resolution SD-OCT volumes. The generative network encodes anatomical variability of healthy images of retina and the discriminative network uses two loss functions to achieve anomaly detection. One of the loss functions is called residual loss and measures the dissimilarity from the generated image to the input one, the second loss is called discrimination loss and makes the generated images to fill into the latent space of anatomically healthy images. Since its publication AnoGAN have been applied to different medical imaging cases, like brain magnetic resonance images.

In Deep Autoencoding Models for Unsupervised Anomaly Segmentation in Brain MR Images [2], Baur et al. compared different deep learning approaches to modelling brain normality: a dense autoencoder, a spatial autoencoder, a dense variational autoencoder, a spatial variational autoencoder and diverse GAN-based models like: sAE-GAN [25], AnoGAN and their own AnoVAEGAN. Their objective was to capture normal anatomical variability of 2D brain images in order to achieve unsupervised anomaly detection. The resolution used for the images was 256x256 pixels. One of conclusions from the publication is that both dense and spatial autoencoders are not just not able to reconstruct brain lesions but lack the capability of reconstructing fine details altogether. Regretfully, the authors did not describe the architectures used for the autoencoders, and tested all the different models using an in-house dataset made of 83 healthy patients and 49 patients with lesions.

2.3 Summary of the study

As the purpose of this work is to achieve a good quality image reconstruction of healthy individual's brain magnetic resonance images we focused initially on testing ResNet-like convolutional autoencoders to attain image reconstruction, as smaller ResNet-like models have achieved comparable image quality reconstruction for cardiac magnetic resonance images [9] against a 10x bigger and more complex U-Net model [27]. The next step was to test if using a deeper standard backbone model helps improve the results substantially. The selected model for this task was ResNet50 as it delivers a good balance between model complexity and accuracy plus is widely popular on the Medical Imaging area [38], [41]. Further on, we trained using pre-trained weights from ImageNet and compared those results with the previous ones trained on the IXI Dataset from scratch.

Along with the comparison between different models and the effects of transfer learning we studied how different loss functions modify the image reconstruction results. Structural similarity (SSIM) was introduced as a perceived quality index for images [39] in 2004. Since then the use of SSIM as loss function has shown promising outcomes in different tasks similar to ours. Humans preferred SSIM-optimized images to other obtained through pixel-wise loss functions like mean square error(MSE) and mean absolute error in image reconstruction [26] and SSIM also improved the results of pixel-wise loss functions in unsupervised defect segmentation using autoencoders recently [3]. For that reason, both the standard MSE loss function and SSIM were used in this study.

Chapter 3

Dataset

The IXI Dataset [21] collects magnetic resonance images of healthy subjects. The data was made available under a Create Commons CC BY-SA 3.0 license. The images were collected in three hospitals in London using different scanners: Hammersmith Hospital through a Philips 3T system, Guy's Hospital using a Philips 1.5T system and in the Institute of Psychiatry using a GE 1.5T system. The dataset contains T1, T2 and PD-weighted images, MRA images and diffusion-weighted images, but for this study only the subset of T1-weighted images were used. T1-weighted images were selected as they are the most commonly used magnetic resonance image modality in structural imaging tasks. The subset contains 581 images in the NIFTI format.

NIFTI stands for Neuroimaging Informatics Technology Initiative and is an open file format introduced in 2003 to store brain data obtained through magnetic resonance imaging techniques [7]. The NIFTI format stores one big 3D volume that represents the whole brain, scanned using magnetic resonance imaging. In order to achieve that the information is stored in voxels. As the goal of this project is to study how different 2D convolutional autoencoders reconstruct magnetic resonance images, the 3D data in voxels has to be transformed into standard pixel-based 2D images.

The dataset also contains demographic information from the subjects in a spreadsheet file. That information and the T1-weighted files were jointly used in the data exploration process described next.

3.1 Data exploration

The NIBabel library [4] grants easy access to the information stored in the NIFTI format. The scanned brain information stored in voxels can be accessed as an array of 2D slices. In the sub-

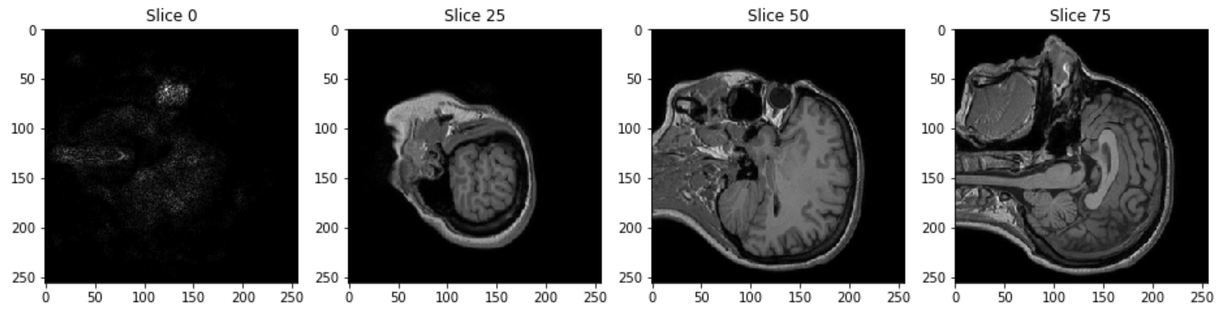

Figure 3.1: Example of four slices, between 0 and 75, from the scan with the IXI ID 002

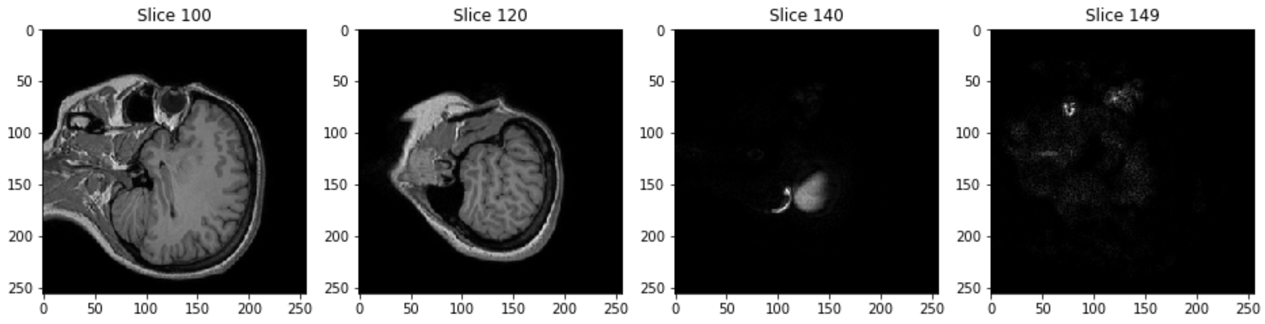

Figure 3.2: Example of four slices, between 100 and 149, from the scan with the IXI ID 002

set of T1-weighted images from the IXI dataset the most common dimensions are 256x256x150, 150 2D slices containing 256x256 pixels each. However, an initial data exploration showed that not all images contained the same amount of slices. Out of the 581 images in the subset: 503 contained 150 slices, 74 of the remaining images 146 slices each, two had 140 slices and the last two 130 slices each. Verifying how many of those slices can be used in this study was the next step in the data exploration process.

Taking the subject with the IXI ID 002 as example, a number of the aforementioned slices can be seen in this page. Figure 3.1 shows slices 0, 25, 50 and 75 from the subject; Figure 3.2 the subsequent slices 100, 120, 140 and 149.

Examining the example slices makes clear that not all slices hold usable information. Training on slices in the extremes that barely contain any information at all would not help the models reconstruct the ones that do. Accordingly, only the 60 slices between the numbers 50 and 110 were used in the development of this project. Using 60 2D slices for image translates to a set of 34860 2D images. An example of four of those slices can be seen in the Figure 3.3.

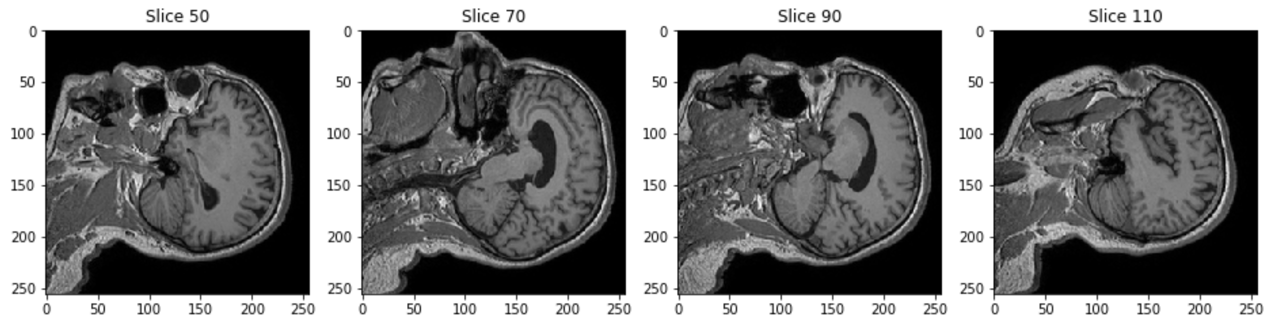

Figure 3.3: Example of four slices in the selected range from the scan with the IXI ID 016

The following step was to check the demographic data information. The spreadsheet identifies patients by the use of the already mentioned IXI ID. Examining the identifiers two issues were found. There are duplicated entries in the spreadsheet for some records, IXI ID 512, 513 or 523 as an example, and not all identifiers present in the subset of data being used had corresponding demographic information. The rows that had duplicated IXI identifiers were removed as they did not contain any new information and the lack of corresponding demographic data was considered when splitting the data.

3.2 Data split

The data available in the spreadsheet is organized in the following columns: sex, height, weight, ethnicity, marital status, occupation, qualification(educational), date of birth, study date and age. Studying the demographic information effect on the reconstruction was not part of this project; still guaranteeing that the training, validation and test set retained the same distribution of patients regarding their sex and ethnicity was deemed particularly necessary as the T1-weighted subsets skews male, with 55% of the patients, and white, 77% of the patients.

The data was then divided into four categories: training, validation, test and other. The last category holds the 15 images that had no demographic data available to consult. As that number represents 2.5% of all subsets, the data was stratified using the data from the sex and ethnicity columns and the images in the category other were added to the training set later.

The 581 magnetic resonance images available in the subset were divided as shown in the Table 3.1. As this project is a study and not an entry in a challenge or competition the data split used was closer to the standard 80/20 or 70/30 split normally employed in machine learning tasks. 77% of the data was used in the training set and the rest for validation and testing. That is a more balanced distribution of the data than the one used, for example, in the

	Subjects	Images
Training set	447	26820
Validation set	86	5160
Test set	48	2880

Table 3.1: *Result of the pre-processing and split of the T1-weighted images from the IXI dataset.*

ImageNet dataset of 2012, Figure 2.2, where 90% of the available data was used for training. The amount of 2D data available relative to the task and the use of real time data augmentation also contributed to the decision.

Chapter 4

Implementation

In the first place, the hardware, programming language and libraries used to develop the code are going to be introduced. The operating system used for the implementation of this the project was Ubuntu 18.04. The specifications of the machine used were: a 6 cores 12 threads AMD Ryzen 2600 CPU, 2 sticks of 8GB DDR4-3200 RAM, a Nvidia GTX 1660 SUPER GPU with 6GB of VRAM and a NVMe solid-state drive.

The code was developed in the Python language, version 3.7.4, using Keras 2.3.1 and Tensorflow 2.1.0 to implement the deep learning models. The CUDA and cuDNN versions used were 10.2 and 7.6.5 respectively. The models were created using Keras' Functional API as the flexibility it provides was needed to implement the shortcut connections used in the residual block and it also improves code's readability.

In order to use structural similarity (SSIM) as loss function the Keras-contrib library version 0.0.2 was also added to the project. NIBabel 3.0.2 was used to read the files in the NIFTI format and imageio 2.8.0 to handle the 2D images [31]. Finally, imgaug 0.4.0 [15] was the data augmentation library selected to generate the augmentations in real time.

The rest of the chapter is devoted to the implementation details of two original models and the use of a standard backbone model and its pretrained weights to achieve transfer learning.

4.1 Encoder

The purpose of the encoder in an autoencoder is to reduce the input signal to a lower dimensionality representation of it. In a convolutional autoencoder the encoder uses a combination of convolutional layers and pooling to fulfill that mission. In this study the encoder receives

2D images of 256x256 pixels and has to create a compact representation of them.

As ResNet-based models have shown better image reconstruction results for cardiac magnetic resonance images than more complex models [9], the goal was to build a simple autoencoder around a building block first described in the original ResNet publication [12] as the residual block. As this block was the building block used for the smaller models introduced in the original publication and can be used to create compact convolutional neural networks.

4.1.1 The Residual block

The residual block implemented is the original version described in the 2015 publication. Residual learning in a neural network can be implemented through a shortcut connection that links the input tensor of a block of convolutional layers with its output, as it was introduced in the section 1.1 of the Chapter 2 State of the Art. The implementation takes an input tensor x that goes through a $Conv2D - BatchNormalization - Relu$ triad and then after an extra $Conv2D - BatchNormalization$ before being added to the input tensor x of the block in a tensor named y that goes through an extra ReLU activation before being returned. If the block has a stride, the dimensions of the input tensor x and the ones of the tensor coming from the first $Conv2D$ layer are not the same, that is why if the block has stride the input gets pooled by a $Conv2D - BatchNormalization$ mini-block before being added to the tensor y. The implementation used is as described in the Algorithm 1.

Algorithm 1 Residual block

```
 Input: tensor x, filters, stride
 Output: tensor y
$y \leftarrow Conv2D(filters, (3,3), stride)(x)$
$y \leftarrow BatchNormalization()(y)$
$y \leftarrow ReLU()(y)$

$y \leftarrow Conv2D(filters, (3,3))(y)$
$y \leftarrow BatchNormalization()(y)$

if $stride = 2$ then
   $x \leftarrow Conv2D(filters, (1,1), stride)(x)$
   $x \leftarrow BatchNormalization()(x)$
end if

$y \leftarrow Add()(x, y)$
$y \leftarrow ReLU()(y)$
```

Layer or Block	Tensor size	Filter size	Stride	Parameters
Input	256, 256, 1	-	-	0
Single convolution 1	128, 128, 32	32	2	288
Residual block 1	64, 64, 64	64	2	58, 112
Residual block 2	64, 64, 64	64	1	74, 240
Residual block 3	32, 32, 128	128	2	230, 912
Residual block 4	16, 16, 256	256	2	920, 576
Up-sampling block 1	32, 32, 256	256	2	590, 848
Up-sampling block 2	64, 64, 128	128	2	295, 424
Up-sampling block 3	128, 128, 64	64	2	73, 984
Up-sampling block 4	256, 256, 32	32	2	18, 560
Single convolution 2	256, 256, 1	1	1	288

Table 4.1: *The Small ResNet-like autoencoder model.*

It is important to note that connecting building blocks that do not use stride have to match the number of filters of the previous block or the dimensions in the add layer would not match either. This was taken into account when designing the two original models used in this study.

4.1.2 Small and Big ResNet-like models

Using the already introduced residual block as the building block for it, a small residual learning model was created to be used as the encoder in the smallest autoencoder to be tested in this study. We named it Small ResNet-like model, its design, that can be seen in the Table 4.1, includes just 4 residual building blocks accounting for around 1.3 million parameters out of a total 2.3 million used by the complete autoencoder. That translates into a lightweight model that can be easily train and deployed in a low specifications machine and that uses 9.2MB to store its weights.

As one of the goals of this study is to see how the number of parameters in the model affect the reconstruction power of an autoencoder, a bigger model was created. That model, named Big ResNet-like, is an extension of the Small ResNet-like model and follows the same design philosophy of not using a number of filters bigger than 256, as that would increase the number of parameters too much, and pooling the input images just four times. It also uses the same decoder of about 1 million parameters, but its expanded encoder consisting of 8 residual blocks is now using around 5 million parameters. That makes the bigger version still easy to train and quite portable as it takes just under 25MB of storage for its weights.

4.1.3 ResNet50

The ResNet50 implementation employed for this study is the one available as part of the standard Keras API under the Keras Applications module. This version was selected as it is included in the API, providing an easy interface, and the ImageNet weights available to download for it are available under the MIT license.

In order to use the model for a task different than classifying the 1000 classes in the ImageNet dataset some modifications have to be done to its layers. Nowadays, most of those modifications can be achieved through the Keras API parameters themselves. In the $ResNet50$ method call, if the parameter *include_top* is set to false, the ResNet50 model would not include the last two layers, a $GlobalAveragePooling$ layer and $Softmax$ layer for the 1000 Imagenet classes, as they are not useful when the model is tackling tasks different than classifying the ImageNet images. After setting it to false, the $weights$ parameter was set to "none" when transfer learning was not selected as an option.

Another important point to mention is that the ResNet50 model uses the more complex identity block, or bottleneck block, instead of the residual block as its building block. The identity block 1x1 initial convolution helps reduce the number of dimensions before the main 3x3 convolution has to compute its part, thus decreasing the overall number of parameters used in a block and helping create a deeper network. In fact, exchanging the original building block in ResNet34 for the 3 convolutions identity block is the origin of ResNet50.

4.1.4 Transfer Learning using ImageNet weights

The transfer learning process described in the Chapter 2, State of the Art, can be implemented in two variants, one that fully retrains all layers in the model pretrained using a different dataset, and another that just fine-tunes some of the layers, effectively freezing the first layers of the model and, thus, blocking them for learning again. The second variant helps improving the results when the images characteristics of the target dataset are similar to the bigger dataset used for pre-training or the number of images to train the model on is very small. This is not the case in this study, as color images featuring objects in naturalistic situations are not similar to greyscale slices of a scanned brain using magnetic resonance imaging and the number of images available for training ranks in the tens of thousands. For this project then, fully retraining the imported model from *keras.applications* using its ImageNet weights is the goal.

The first step in order to achieve transfer learning is to download the weights to be used a knowledge base for the transferring. This process is handled automatically by the Keras API if in the *ResNet*50 method the *weights* parameter is set to "imagenet". This downloads a version of the ImageNet weights in the h5 format that does not include the last two layers of the model designed for classifying the 1000 classes part of the ImageNet datasets, consequently making the file weight around 95MB. Then the resulting ResNet50 backbone can be connected to the decoder in the same manner as for the weight-less version. The only two modifications needed to make the model trainable are related to the number of expected channels as input by the model, as the ImageNet dataset contains RGB images the model requires inputs with three channels and the output has to match that number accordingly. To solve this, the *color_mode* parameter used in the *flow_from_directory* method used to read the data from the solid-state drive has to be set to "rgb" and the last convolutional layer in the decoder has to have the number of filters set to 3 instead of 1. Those two changes are enough to be able to fully retrain the model.

4.2 Decoder

The objective of a decoder in an autoencoder is to reconstruct the original input signal from a representation of the input created by the encoder. That representation contains less information compared to the initial data. In this study the decoder has to take the output of the last layer of the different backbone models used as encoders and up-sample it back to its original size, retaining as much detail as possible. To achieve that we designed a simple up-sampling block which, takes an input tensor, applies to it a *Conv2DTranspose* layer with stride 2 to expand its dimensions in an operation opposite of pooling, and then adds batch normalization and an activation function layer. Its design can be seen in the Algorithm 2.

Algorithm 2 Up-sampling block

```
  Input: tensor x, filters
  Output: tensor y
$y \leftarrow Con2DTranspose(filters, (3,3), strides = (2,2)(x)$
$y \leftarrow BatchNormalization()(y)$
$y \leftarrow ReLU()(y)$
```

The decoder is thus a concatenation of a number of those blocks and, lastly, a simple convolutional layer added to force the output to exactly match the input image size creates of the encoder. The number of up-sampling blocks used is determined by the number of times the

Encoder backbone	Input size	Encoder output size	Up-sampling blocks
Small-ResNet-like	256, 256, 1	16x16x256	4
Big-ResNet-like	256, 256, 1	16x16x256	4
ResNet50	256, 256, 1	8x8x2048	5
ResNet50-ImageNet	256, 256, 3	8x8x2048	5

Table 4.2: *Comparison between the encoder input dimensions, the output dimensions and the number of up-sampling blocks used in the decoder.*

Model	Total number of parameters
Small-ResNet-like	2,263,232
Big-ResNet-like	6,104,256
ResNet50	34,589,600
ResNet50-ImageNet	34,596,448

Table 4.3: *Comparing the total number of parameters used by the models of this study.*

input is shrunk in the encoder. The table 5 compares the input size of the encoders and its output with the number of up-sampling blocks used in the decoder. Lastly, the table 4.3 shows the total number of parameters for the four autoencoders used in this study.

4.3 Data generators and training optimizer

In order to train the different outlined autoencoder models two remaining topics need to be discussed. How to properly train the model in batches and under which optimizer.

As the number of images in the dataset exceeds the number of images that can be easily stored in VRAM the use of Keras' *ImageDataGenerator* becomes a necessity, as that class streams data from the stored dataset in batches that can be fitted in the GPU's VRAM while being able to apply pre-processing and some data augmentation techniques in real time. One call to *ImageDataGenerator* can be used to create both the training and validation generators from different subfolders in the dataset using the *flow_from_directory* method for each of them. As this project is only concerned about training autoencoders the *class_mode* parameter has to be set to "input" to guarantee the loss of the model is calculated using the input image as both input and correct label.

The default optimizer used to train the models was root mean square propagation, from now on RMSprop. It was selected as the optimizer as it normally converges faster than standard stochastic gradient descent helping fast iteration during the implementation process. It is worth noting that after a number of experiments conduct using RMSprop for all models the results for the ResNet50 based autoencoders were below our original expectations and the ResNet50 models were trained again using the adaptive moment estimation, Adam, optimizer instead. More details can be read in the Chapter 5 where experiments are discussed.

4.3.1 Data augmentation

The use of real time augmentation helps in the form of both reducing the space used in storage as pre-augmented images do not need to be stored before being used and also helps training as all batches of images are going to feature slightly different versions of the same input, thus making the model less prone to overfitting. There is the caveat that real time data augmentation does indeed increase the amount of computations needed to train a model, but it is a positive trade-off that became the standard when training deep learning models.

The *ImageDataGenerator* class in Keras only provides a subset of the standard data augmentation techniques usually employed, like random brightness augmentation, zoom, vertical flip, horizontal flip, etc. More advanced techniques are not supported. For this project, due to the nature of the data and the task, we wanted to create models robust to noise introduced by the scanning process or errors in storing leading to areas in the image that do not feature proper pixel information. As the data from the slices used in the dataset do not feature any of those problems, in order improve the robustness against them some of the images used to train the models have to be altered. For example, the autoencoder can be fed an altered image featuring noise and the original one that does not present it. The autoencoder would then try to clean the noise of the image to reduce the loss value and get as close as possible to the original image. As the Keras library cannot generate images with those characteristics an external library was needed. imgaug was selected as it provides a broad range of augmentations, including a diverse set of methods to introduce noise and drop patches of pixels, that can be easily added to an already generated batch of images, becoming an efficient tool to implement our target augmentations.

Specifically, the *data_augmentation_generator* method that can be seen in the training code takes the batches of images generated by a *ImageDataGenerator* for training or validation and adds to the input images the selected augmentations, preserving an original copy of them to be the objective of the reconstruction and help the model become resilient to the adverse conditions created by the augmentations. An augmentation imgaug sequence was set. The sequential imgaug configuration used was: add noise through the *GaussianBlur* method to 50% of the images and apply from 0 to 2 of the following methods: *CutOut* and *CoarseDropout*. *CutOut*

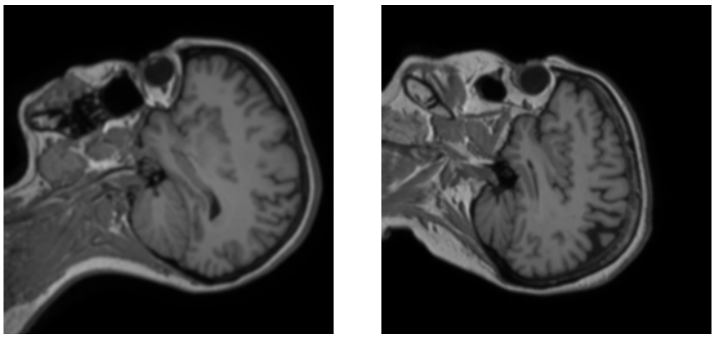

Figure 4.1: Examples of augmented images using the imgaug method GaussianBlur

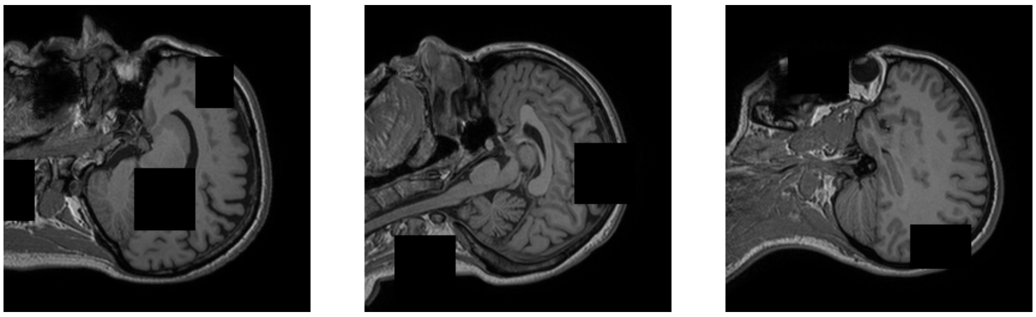

Figure 4.2: Examples of augmentations using the imgaug methods CutOut and CoarseDropout

introduces patches of mono-color pixels that mask the original information, *CoarseDropout* drops a percentage of all pixels in a image. In the configuration used, *CutOut* generates two black patches of pixels in the images and *CoarseDropout* drops between 0 and 5% of all pixels in an image. As not all images are augmented, the models are able to learn how to reconstruct images robustly. Example of the augmentation results can be see in Figures 4.1 and 4.2 and the augmentation sequence itself can be seen in the training code.

Chapter 5

Experiments

As the idea behind the study is to compare the results obtained by different encoders using a variable number of parameters it was decided that the training conditions should be as similar as possible for all models, although always ensuring the models were able to be properly trained under those conditions. That is the reason behind some of the rules set for the training phase: early stopping was not implemented as the objective was to train all models for the same number of epochs each, 100, using the RMSprop optimizer and the same rules for the learning rate reducer method.

As training all models for a 100 epochs was a long process some initial experiments were done to leverage if the selected combination of learning rate, optimizer and loss function would provide usable results. Those partial results showed promising outcomes for both ResNet-like models, teasing to a better quality reconstruction when employing the dssim loss, but the showings for the ResNet50 autoencoders were poor. The loss did not decrease as much as for the ResNet-like models and the results showed images with a clear loss of detail, producing an outlined brain structured easy to recognize yet resulting in a globally blurred imaged. Figure 5.1 shows two examples of blurred reconstruction from an early ResNet50 model trained with mse.

The initial reconstruction results for ResNet50 lead to a discussion of its cause. The model was designed for an image recognition task not reconstruction and the smaller models were learning more and generating good results just pooling the input image 4 times instead of 5 as ResNet50 was doing. Table shows the output dimensions of the encoder for ResNet50 versus the ResNet-like models. So a first parameter to control for would be the number of times the input is pooled, thus it was decided to train a shorter ResNet50 model using the same identity block but that just reduces the input images 4 times each dimension. By training a model like that it would be possible to dismiss that the differences in the outputs were caused by the

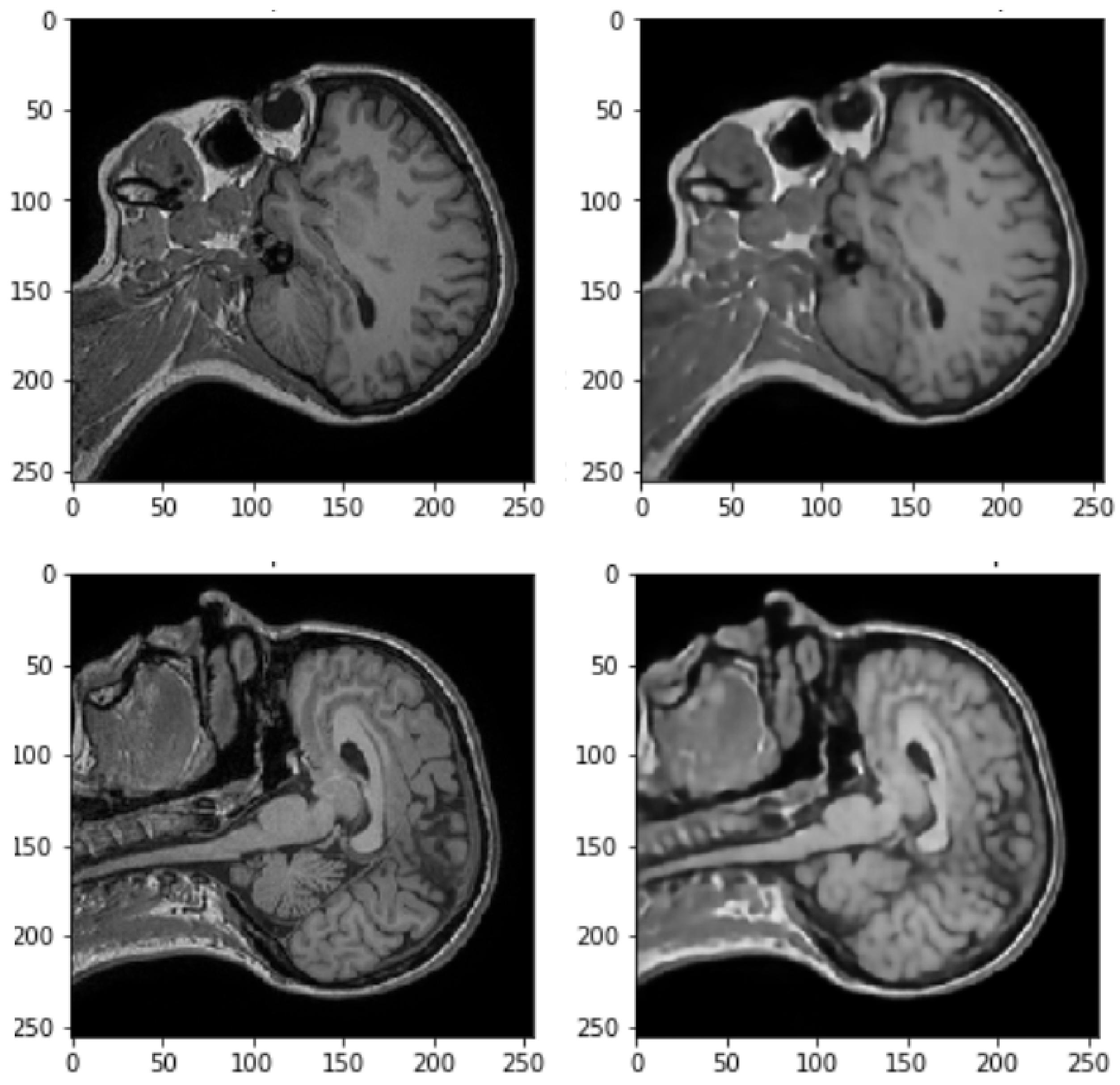

Figure 5.1: Image showing two examples of blurred reconstruction by an early ResNet50 mse model. Left side of the image shows the original slices, slice 52 from IXI ID 021 and slice 74 from IXI ID 045, and the right side their image reconstructions.

Model	From Layer	Loss function	Total parameters	Best loss	Epoch
ResNet50	add_13	mse	11,331,488	0.00038	99/100
ResNet50	add_13	dssim	11,331,488	0.02678	100/100
ResNet50	act_40	dssim	11,331,488	0.02698	93/100
ResNet50-ImageNet	add_13	dssim	11,338,336	0.02182	81/100

Table 5.1: *Training results for the modified ResNet50 models. The batch size used was 16 in all cases.*

inherent differences in the building blocks used by the models: the residual building block and the identity building block.

In order to be able to train such a model the following method was used: import ResNet50 in the same manner as before but force it to connect the output of an intermediate layer to the decoder, making thus the model shallower, and reduce the number of up-sampling layers by one to go in line with the new encoder. The objective was to test a reduced encoder in size that pooled the input images just 4 times, the two last layers of the last identity block before the model was reduced again were tested. Those layers were: add_13 and act_40. Their models outputting from those layers have an identical number of parameters, as the act_40 activation does not increase the number of parameters of the model over add_13.

The Table 5.1 shows the shallow models number of parameters, from which layer of the original ResNet50 were obtained, the best validation loss obtained and in which of the 100 epochs the models reached their lowest validation loss value.

The reconstruction results of those shallower ResNet50 models were more in-line with the partial results seen on the ResNet-like models, but still below them even when the loss for the ResNet50 encoder using the output of add_13 was identical to the best ones obtained by the ResNet-like models using mse as seen in Figure 5.2. Seeing those results it became clear that the problem was not in the use of the identity block, so we decided to further explore training under different hyper-parameter configurations for the standard ResNet50 models to see if there was a combination of parameters able to improve their results.

Figure 5.3 displays the validation loss evolution for the ResNet-like and short ResNet50 models using dssim for a 100 epochs. The short ResNet50 model using transfer learning from the ImageNet dataset obtained the best dssim loss. In this case the shorted models validation losses were in-between the ones obtained by the ResNet-like models and ResNet50.

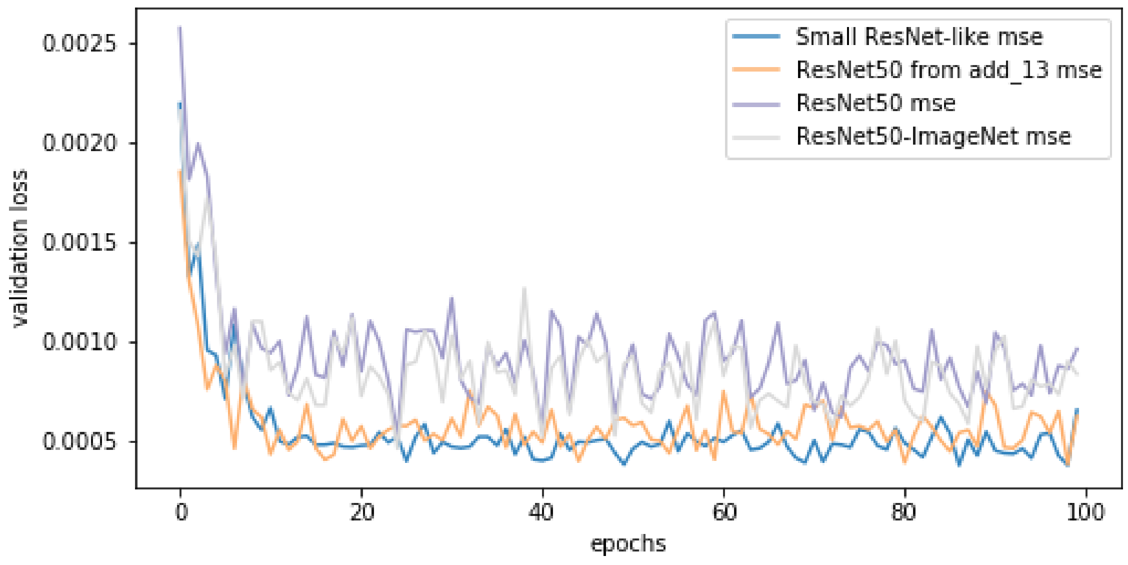

Figure 5.2: Comparison between the short ResNet50 validation loss evolution during 100 epochs to the previous models using mse

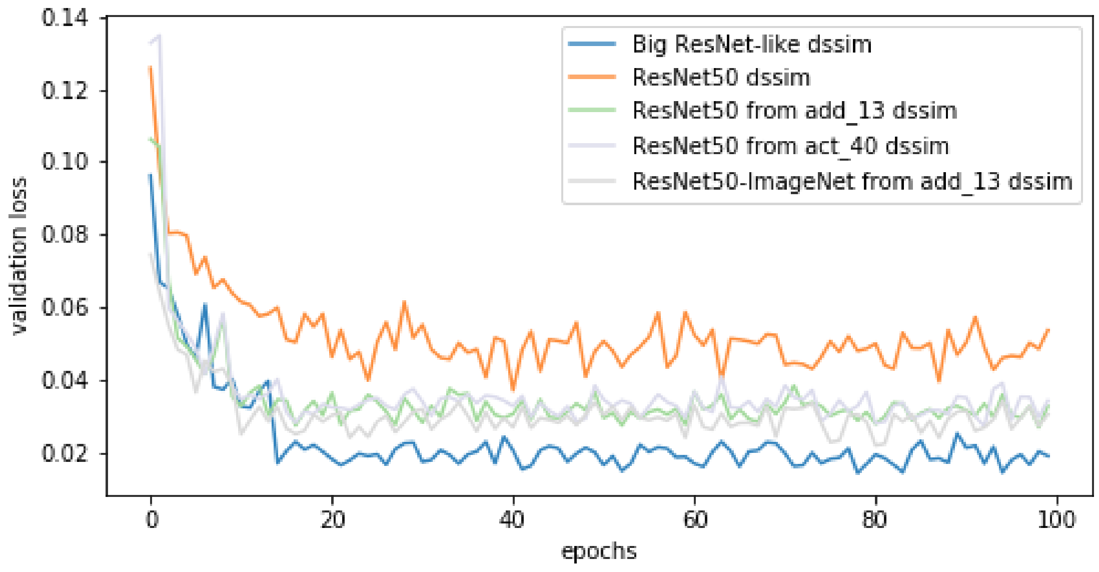

Figure 5.3: Comparison between the short ResNet50 validation loss evolution during 100 epochs to the previous models using dssim

Model	Loss function	Best loss	Epoch	Batch size
Small ResNet-like	mse	0.00038	87/100	32
Small ResNet-like	dssim	0.01756	51/100	32
Big ResNet-like	mse	0.00037	82/100	32
Big ResNet-like	dssim	0.01419	79/100	24

Table 5.2: *Training results for the ResNet-like models.*

The decision of training the ResNet50 models under different parameters was taken and as the implementation phase of the project was only two months long we selected an optimizer known to converge fast. That is the reason why the ResNet50 models were trained using Adam instead of RMSprop for a 100 epochs.

5.1 Quantitative results

In order to test the quality of the image reconstruction two standard metrics were used, peak signal-to-noise ratio, PSNR from here on, and structural similarity or SSIM. It is important to note that the SSIM implementation used in the test code is different from the one used as loss function even if their goals are the same. The difference of structural similarity, DSSIM, loss function from the keras-contrib library was coded in plain Keras while the SSIM method used to test the quality of the reconstruction is part of the core Tensorflow library.

The quantitative results shared in this section are the best results obtained for each of the models, all of them started with a learning rate of 0.001 and through the Keras method $ReduceLROnPlateau$ were set with an aggressive combination of 2 for *patience* and 0 for *cooldown*. Meaning that after two consecutive epochs of no improvement in the loss the learning rate is set to be reduced without a cooldown epoch. The minimum loss was set at $0.5e-6$ and the reduction factor at $\sqrt{0.1}$ to partially compensate for the values used for *patience* and *cooldown*, creating more intermediate learning rate steps. Table 5.2 showcases the best validation loss obtained for the ResNet-like models and in which epoch were reached. The equivalent results for ResNet50 can be seen in Table 5.3.

Model	Loss function	Best loss	Epoch
ResNet50	mse	0.00052	26/100
ResNet50	dssim	0.03698	41/100
ResNet50-ImageNet	mse	0.00046	25/100
ResNet50-ImageNet	dssim	0.04897	41/100

Table 5.3: *Training results for the ResNet50 models. The batch size used was 8 in all cases.*

The final loss values obtained for the ResNet-like models are the lowest, they bested the previous attempts at their training and generate reconstruction results that are clearly satisfactory for all four combinations of model and loss function. For the ResNet50 models the used of the Adam optimizer seemed to be helpful but regretfully the early tests models trained using RMSprop were trained for around 30 to 50 epochs each so the results are not directly comparable as the best losses for the ResNet50 combination were reached in epochs between the number 25 to 41. Nevertheless the original blur in the reconstruction was reduced even if the results are still far from the ones obtained by the ResNet-like models.

Figure 5.4 illustrates the evolution of the validation loss for all models using mse as loss function when trained for a 100 epochs. The Figure 5.5 is the dssim counterpart of the previous image. In those two cases there are no signs of any positive effect provided by the transfer learning technique as there was for the shorter ResNet50 models.

The results for the test set of 2880 2D images can be seen in the Table 5.4 for PSNR method and in Table 5.5 for SSIM method. The scores obtained in both methods are consistent and show a clear preference for the ResNet-like models over the ResNet50 ones and for dssim as loss function. The best overall model is the Big ResNet-like model using dssim, obtaining the best results for all 8 metrics used. The Small ResNet-like model using dssim is pretty close but scoring second best place in all metrics. Both models are better than their mse counterparts, obtaining a mean PSNR score 8% better in the small versions case and 12% better for the big one. The model using mse that achieved the best scores was the Small ResNet-like model even when the validation loss was basically the same for both of them. The PSNR scores for the ResNet50 models are between 21% to 30% below those of the best model, consistent with the lack of detail in their reconstructions. In general the results for the SSIM tests are closer than those of the PSNR tests but consonant with them.

The results obtained show that the ResNet-like models score consistently better, generate images more visually pleasing, as discussed in the next section, and use a smaller number of parameters and operations. On top of that, using dssim provides a boost in performance over

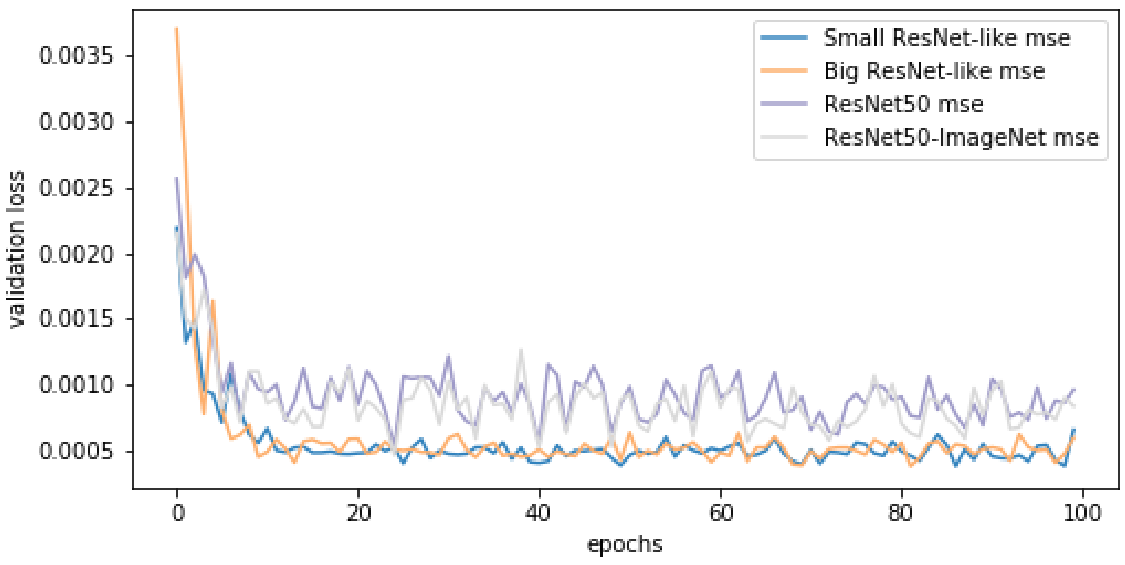

Figure 5.4: Evolution of the validation loss during 100 epochs for the models using mse as loss function

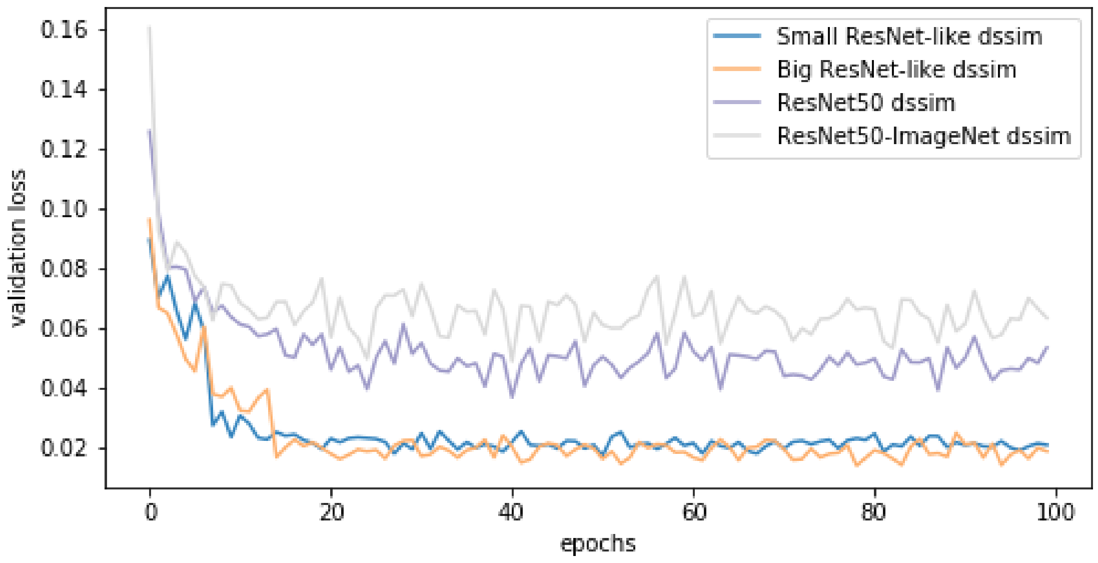

Figure 5.5: Evolution of the validation loss during 100 epochs for the models using dssim as loss function

Encoder	Loss function	mean	std	min	max
Small ResNet-like	mse	37.098689	2.076884	33.467362	42.469414
Small ResNet-like	dssim	40.057103	2.114364	35.955231	46.016052
Big ResNet-like	mse	36.139288	2.163328	32.417344	41.653457
Big ResNet-like	**dssim**	**40.761520**	**1.856601**	**37.010069**	**47.003049**
ResNet50	mse	32.121366	2.257936	28.422910	37.743369
ResNet50	dssim	33.458456	2.372063	29.549638	39.737773
ResNet50-ImageNet	mse	32.513734	2.295906	28.708848	38.310702
ResNet50-ImageNet	dssim	31.248317	2.246222	27.370514	37.470976

Table 5.4: *PSNR results for the test set. The model with the best results is in bold.*

Encoder	Loss function	mean	std	min	max
Small ResNet-like	mse	0.995426	0.000802	0.992202	0.997225
Small ResNet-like	dssim	0.998301	0.000322	0.996286	0.999107
Big ResNet-like	mse	0.994263	0.001094	0.991537	0.996677
Big ResNet-like	**dssim**	**0.998579**	**0.000254**	**0.997920**	**0.999234**
ResNet50	mse	0.978893	0.004586	0.967548	0.990173
ResNet50	dssim	0.986320	0.003356	0.977293	0.994523
ResNet50-ImageNet	mse	0.981160	0.004202	0.970986	0.991268
ResNet50-ImageNet	dssim	0.974766	0.005757	0.959964	0.988441

Table 5.5: *SSIM results for the test set. The model with the best results is in bold.*

mse at no computational cost when using the models.

For completeness the results of the short ResNet50 models are listed in the following tables: the PSNR results are in Table 5.6 and the SSIM scores in Table 5.7.

The scores of the short ResNet50 models are always below the best models but an early inspection showed that their quality was enough to provide reconstructions visually pleasing.

Encoder	Loss function	mean	std	min	max
ResNet50 from add_13	mse	34.958357	2.201821	31.296535	40.600480
ResNet50 from add_13	dssim	37.136453	2.209275	33.522393	43.163153
ResNet50 from act_40	dssim	36.775637	2.259349	33.047043	42.712570
ResNet50-ImageNet from add_13	dssim	37.716954	2.143963	34.072346	43.634058

Table 5.6: *PSNR results in the test set for the short ResNet50 models.*

Encoder	Loss function	mean	std	min	max
ResNet50 from add_13	mse	0.990759	0.001999	0.986000	0.995427
ResNet50 from add_13	dssim	0.994716	0.001285	0.991834	0.997592
ResNet50 from act_40	dssim	0.994342	0.001353	0.991118	0.997415
ResNet50-ImageNet from add_13	dssim	0.995552	0.001051	0.993046	0.997945

Table 5.7: *SSIM results in the test set for the short ResNet50 models.*

5.2 Qualitative results

As described previously, the average results for ResNet-like models show a satisfactory level of reconstruction that makes difficult to tell the original images and the reconstructed ones a part. Figure 5.6 shows a reconstruction example from the Small ResNet-like model using mse and the Big ResNet-like model trained using dssim. A zoomed version of the upper-left part of the images can be seen in Figure 5.7. Looking at the reconstructed examples is not easy to distinguish them from the original slice or between themselves. After further inspection the original image and the dssim based reconstruction may look to be on a slightly different tone of grey compared to the mse image, a minor difference probably easier to spot on the lighter textures. However, looking at the zoomed images it becomes easier to see why the Big ResNet-like dssim model scores better in all tests, it does capture fine details that the mse model cannot.

In order to improve the robustness of the models real time data augmentation was introduced in the project. To test its effects we selected a small number of slices, two from each original MRI in the test set, and applied the same augmentations we used for training to those images one at a time. For each subject one of the slices was augmented with noise and the other one zeroed using the aforementioned methods available in the imgaug library. Figure 5.8 shows the results for 6 of our models trying to reconstruct areas of a slice not seen before. The results for those areas can be roughly ranked in reverse of the discussed order until now. ResNet50 models do a better job at restoring the inherent structures of the scanned brains but still fail to reconstruct the rest of the images with the amount of detail obtained by the ResNet-like models. The worst results were generated by the Small ResNet-like model using dssim, but the two Big ResNet-like models showcase the same kind of behaviour. They lack the ability to fill the zeroed areas realistically, leaving blurry patches in places were the ResNet50 models do a better job at capturing the global structure of the images. Another example can be seen in Figure 5.9 where the ResNet50 model using ImageNet weights and mse is able to rough out the orbit of the eye and the Big ResNet-like model using dssim does not. For the remaining augmentation, introducing blur to the images, the ResNet-like models come back at top. Figure 5.10 shows how the Big ResNet-like model is able to restore some of the details lost

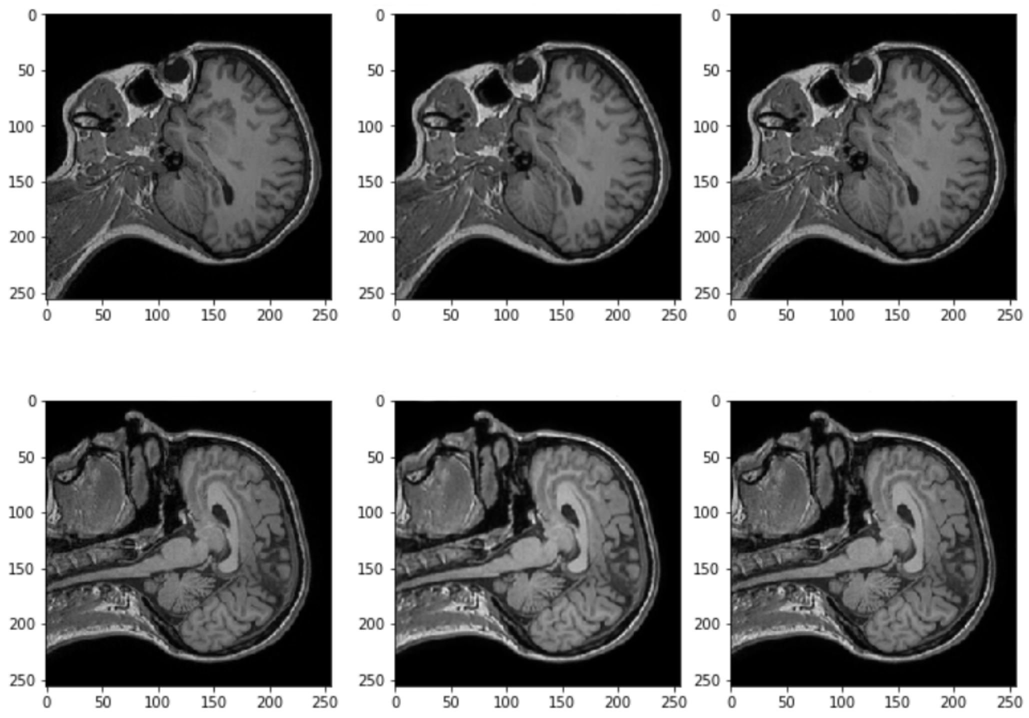

Figure 5.6: Left images are the original slices: slice 52 from IXI ID 021 and slice 74 from IXI ID 045. Middle images are the reconstructions by the Small ResNet-like mse model and the images in the right are the reconstructions done by the Big ResNet-like dssim.

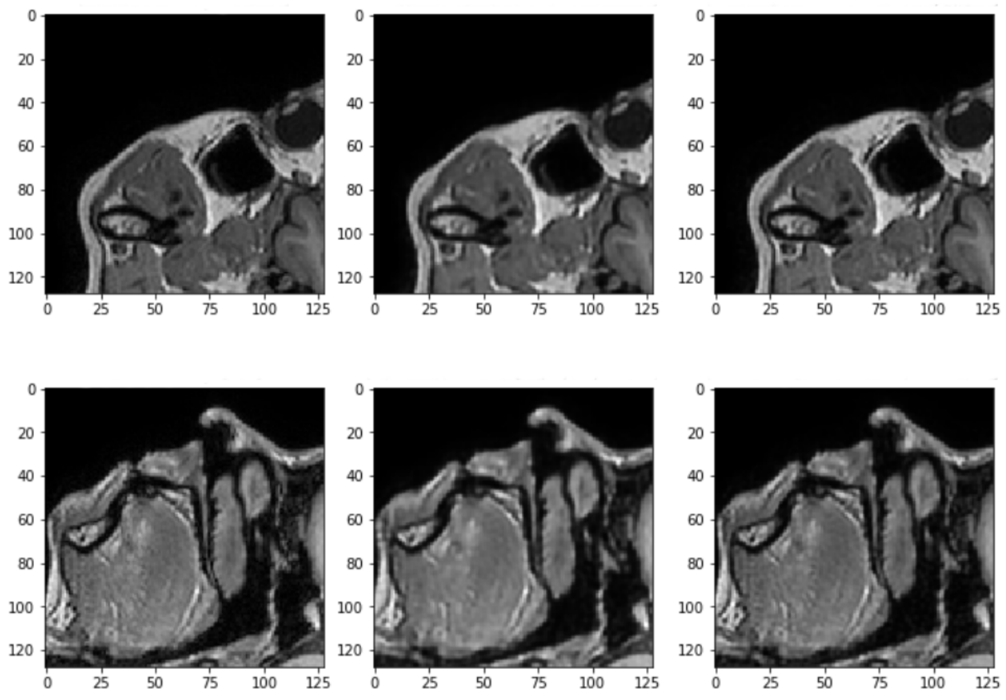

Figure 5.7: Zoomed version of the Figure 5.6

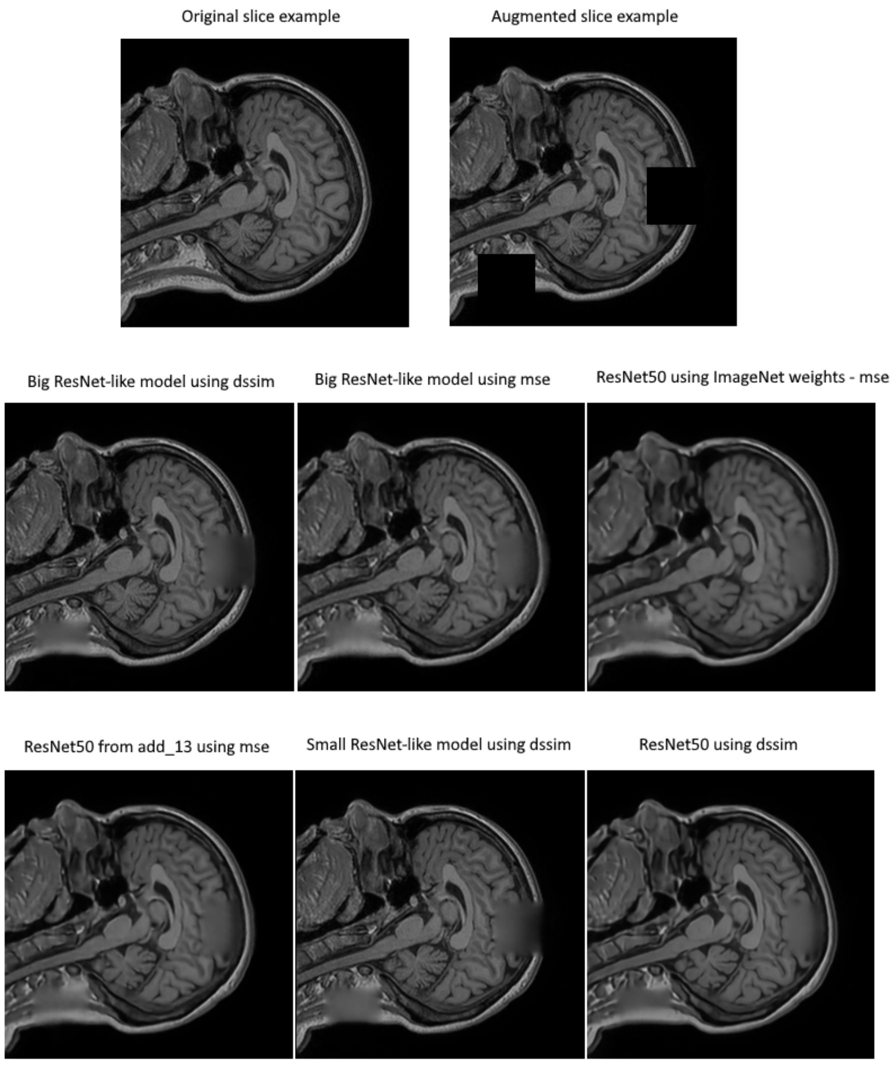
Original slice example
Augmented slice example
Big ResNet-like model using dssim
Big ResNet-like model using mse
ResNet50 using ImageNet weights - mse
ResNet50 from add_13 using mse
Small ResNet-like model using dssim
ResNet50 using dssim

Figure 5.8: Collage comparing how 6 different models reconstruct a zeroed area of a slice from the test set.

due to the introduced Gaussian blur and the ResNet50 based model is not able to even retain the remaining details on the blurred image.

Additionally, some preliminary testing of images with introduced noise seemed to indicate better results for ResNet50 models and mse as a loss function. Figure 5.11 shows a comparison between ResNet50 and the Big ResNet-like using both mse and dssim. The images were created using the *DropOut* method from the imgaug library which differs from the *CoarseDropOut* used for training and from standard pepper noise. *CoarseDropOut* reduces the resolution of the images to 50% before dropping pixels, creating blocks of 2x2 black pixels in consequence, meanwhile pepper noise introduces pixels that are not uniformly black. The results are poor,

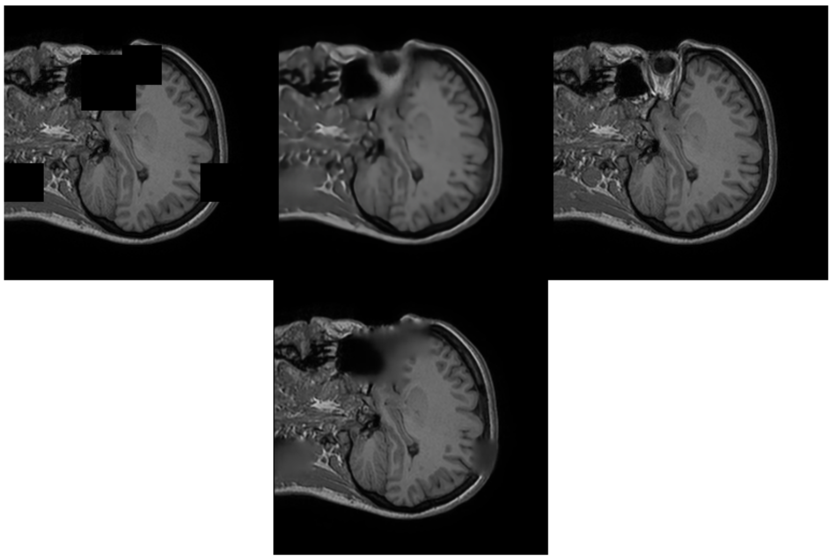

Figure 5.9: Zeroed image can be seen on the left, original one in the right and the reconstructed image by the ResNet50 based model on the middle, on top of the Big ResNet-like one.

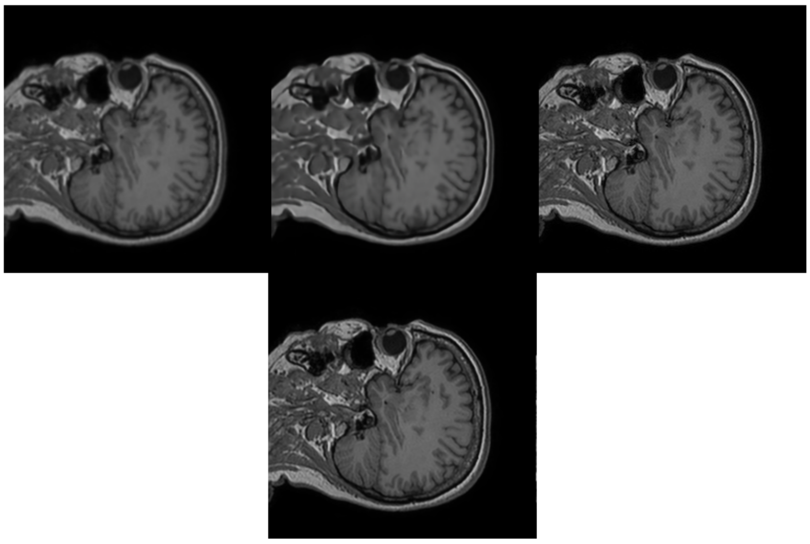

Figure 5.10: Blurred image can be seen on the left, original one in the right and the reconstructed image by the ResNet50 based model on the middle, on top of the Big ResNet-like one.

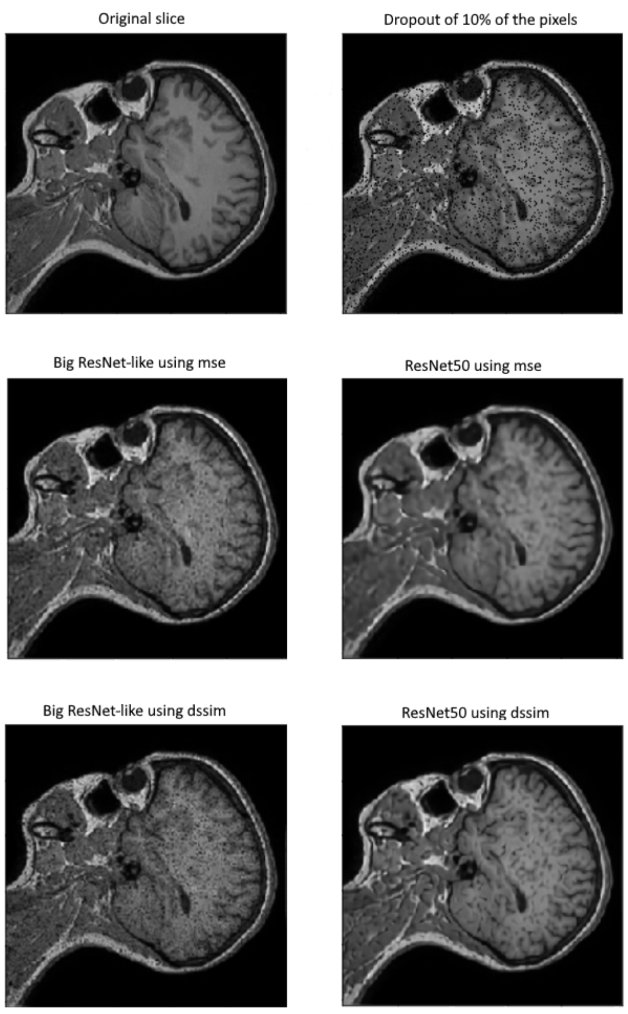
Original slice
Dropout of 10% of the pixels
Big ResNet-like using mse
ResNet50 using mse
Big ResNet-like using dssim
ResNet50 using dssim

Figure 5.11: Collage showing a comparison between the reconstruction results of different models for the slice 52 from IXI ID 021 when 10% of its pixels are dropped.

as ResNet50 using mse shows better resilience to the noise but fails to reproduce fine details as we know from previous experiments and the Big ResNet-like models output grainy images that fall short of their previous results. Adding noise augmentation and building a proper noise augmented test set would be necessary steps to take in order to prepare for and evaluate the models concerning the quality of their reconstruction of noisy images.

Chapter 6

Conclusions

After exploring the experiments realized in this short study it is clear that the handcrafted ResNet-like models provide the best quantitative results, showcasing high quality reconstruction scores in all metrics of the test set, using a fraction of the parameters and operations of the ResNet50 based models.

The ResNet50 models show better resilience to aggressive data augmentation in the form of zeroed pixels but are unable to retain pixel detail in general. As their best validation losses were reached in early stages of the training process we cannot assert the models when trained properly can not obtain results closer to the ones obtained by our handcrafted models. Although, after testing the shorter ResNet50 models it does seems unlikely to achieve levels of performance better than the ones obtained by the ResNet-like models, it remains as an open question for further study.

Regarding the loss function, the models using dssim obtained higher scores than the ones using mse in both the PSNR and the SSIM tests for all tested pairs of models but the ResNet50 model using ImageNet weights. For that dssim model the loss remained high, 34% higher, than for the ResNet50 model that did not use pre-trained weights, creating the only outlier in the series. As it is the only model where the mse version obtains better results than the dssim one and the only case where the ImageNet weights did not provide a boost to the results. Proper training should be able to address this difference and help clarify if transfer learning from the ImageNet dataset can improve the results of training from scratch in the IXI dataset.

Concerning the differences in performance between the two ResNet-like models we created, we only tested one method of increasing the number of parameters: adding extra residual blocks to the Small ResNet-like model. The increment from 2.26 million to 6.1 million parameters led to a rise in performance for the dssim version, making the Big ResNet-like model the best scoring model of the study, but not for the mse model where the Small ResNet-like obtained better scores even when the validation loss reached for the Big ResNet-like model was lower. It is important to point out that the differences in performance between the Small ResNet-like model and the Big ResNet-like are smaller than the ones created by the use of the two loss functions, making the increase in parameters a factor less important than the choice of the loss function in this study.

In summary, ResNet-like models when paired with dssim as loss function yielded the best scores in the tests, showing a larger gap in the mean PSNR metric where the dssim version is 8% better than its mse counterpart for the small model and 12% better for the big one. Noting that the results from all ResNet-like models are close to be indistinguishable from the original images to the human eye. Therefore, using any of the ResNet-like models to achieve unsupervised anomaly detection in brain magnetic resonance images may be an attainable future goal.

6.1 Future work

- Retrain the ResNet50 family of models using a different combination of hyperparameters and learning rate policies.
- Retrain all the models that used the RMSprop optimizer with Adam to compare its results.
- Expand the data augmentation to see if it can help better the results of all models and make the smaller models retain more structural information.
- Check if adding data from different datasets containing MRI of healthy individuals, like the IXI dataset does, would help generalize the models and improve their quality.
- Study if training all models using a higher batch size in a GPU with more VRAM available reduces the differences in their outputs.
- Explore if creating a different model that just increases the number of filters, exceeding the number of 256 filters per convolution, or both the maximum number of filters and the number of residual blocks would lead to better results compared to the Small ResNet-like than the Big ResNet-like model does.

- Lastly, the slight improvement seen on the reconstruction of zeroed images from the Big ResNet-like model compared to the smaller one may point out to another direction but it may be worthwhile to check if an even shorter ResNet50 model trained with a good combination of hyperparameters can indeed retain the structural knowledge showcased by the ResNet50 models while properly reconstructing the general details of the images using a smaller number of parameters as the ResNet-like models we build did.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Christoph Baur, Benedikt Wiestler, Shadi Albarqouni, and Nassir Navab. Deep autoencoding models for unsupervised anomaly segmentation in brain mr images. In Alessandro Crimi, Spyridon Bakas, Hugo Kuijf, Farahani Keyvan, Mauricio Reyes, and Theo van Walsum, editors, *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries*, pages 161–169, Cham, 2019. Springer International Publishing.

[3] Paul Bergmann, Sindy Löwe, Michael Fauser, David Sattlegger, and Carsten Steger. Improving unsupervised defect segmentation by applying structural similarity to autoencoders. *CoRR*, abs/1807.02011, 2018.

[4] Matthew Brett, Christopher J. Markiewicz, Michael Hanke, Marc-Alexandre Côté, Ben Cipollini, Paul McCarthy, Christopher P. Cheng, Yaroslav O. Halchenko, Michiel Cottaar, Satrajit Ghosh, Eric Larson, Demian Wassermann, Stephan Gerhard, Gregory R. Lee, Hao-Ting Wang, Erik Kastman, Ariel Rokem, Cindee Madison, Félix C. Morency, Brendan Moloney, Mathias Goncalves, Cameron Riddell, Christopher Burns, Jarrod Millman, Alexandre Gramfort, Jaakko Leppäkangas, Ross Markello, Jasper J.F. van den Bosch, Robert D. Vincent, Henry Braun, Krish Subramaniam, Dorota Jarecka, Krzysztof J. Gorgolewski, Pradeep Reddy Raamana, B. Nolan Nichols, Eric M. Baker, Soichi Hayashi, Basile Pinsard, Christian Haselgrove, Mark Hymers, Oscar Esteban, Serge Koudoro, Nikolaas N. Oosterhof, Bago Amirbekian, Ian Nimmo-Smith, Ly Nguyen, Samir Reddigari,

Samuel St-Jean, Egor Panfilov, Eleftherios Garyfallidis, Gael Varoquaux, Jakub Kaczmarzyk, Jon Haitz Legarreta, Kevin S. Hahn, Oliver P. Hinds, Bennet Fauber, Jean-Baptiste Poline, Jon Stutters, Kesshi Jordan, Matthew Cieslak, Miguel Estevan Moreno, Valentin Haenel, Yannick Schwartz, Benjamin C Darwin, Bertrand Thirion, Dimitri Papadopoulos Orfanos, Fernando Pérez-García, Igor Solovey, Ivan Gonzalez, Jath Palasubramaniam, Justin Lecher, Katrin Leinweber, Konstantinos Raktivan, Peter Fischer, Philippe Gervais, Syam Gadde, Thomas Ballinger, Thomas Roos, Venkateswara Reddy Reddam, Zvi Baratz, and freec84. nipy/nibabel: 3.0.2, March 2020.

[5] Xiaoran Chen and Ender Konukoglu. Unsupervised detection of lesions in brain MRI using constrained adversarial auto-encoders. *CoRR*, abs/1806.04972, 2018.

[6] François Chollet et al. Keras. https://keras.io, 2015.

[7] Robert W Cox, John Ashburner, Hester Breman, Kate Fissell, Christian Haselgrove, Colin J Holmes, Jack L Lancaster, David E Rex, Stephen M Smith, Jeffrey B Woodward, and Stephen C Strother. Neuroimaging Informatics Technology Initiative. https://nifti.nimh.nih.gov/nifti-1, 2004. [Accessed: 2020-06-13].

[8] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. *CoRR*, abs/1310.1531, 2013.

[9] Vahid Ghodrati, Jiaxin Shao, Mark Bydder, Ziwu Zhou, Wotao Yin, Kim-Lien Nguyen, Yingli Yang, and Peng Hu. Mr image reconstruction using deep learning: evaluation of network structure and loss functions. *Quantitative Imaging in Medicine and Surgery*, 9(9), 2019.

[10] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

[11] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[13] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *CoRR*, abs/1709.01507, 2017.

[14] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[15] Alexander B. Jung. imgaug. https://github.com/aleju/imgaug, 2018. [Accessed: 2020-04-28].

[16] Mark A. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, 37(2):233–243, 1991.

[17] Alex Krizhevsky. cuda-convnet project webpage. https://code.google.com/archive/p/cuda-convnet/. Accessed: 2020-03-15.

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[19] Zeng Kun, Jun Yu, Ruxin Wang, Cuihua Li, and Dacheng Tao. Coupled deep autoencoder for single image super-resolution. *IEEE Transactions on Cybernetics*, 47:1–11, 12 2015.

[20] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[21] Biomedical Image Analysis Group Imperial College London. Information eXtraction from Images. https://brain-development.org/ixi-dataset/, 2015. [Accessed: 2020-06-17].

[22] Kin Gwn Lore, Adedotun Akintayo, and Soumik Sarkar. Llnet: A deep autoencoder approach to natural low-light image enhancement. *CoRR*, abs/1511.03995, 2015.

[23] Bjoern Menze, Andras Jakab, Stefan Bauer, Jayashree Kalpathy-Cramer, Keyvan Farahani, Justin Kirby, Yuliya Burren, Nicole Porz, Johannes Slotboom, Roland Wiest, Levente Lanczi, Elisabeth Gerstner, Marc-Andre Weber, Tal Arbel, Brian Avants, Nicholas Ayache, Patricia Buendia, Louis Collins, Nicolas Cordier, Jason Corso, Antonio Criminisi, Tilak

Das, Hervé Delingette, Cagatay Demiralp, Christopher Durst, Michel Dojat, Senan Doyle, Joana Festa, Florence Forbes, Ezequiel Geremia, Ben Glocker, Polina Golland, Xiaotao Guo, Andac Hamamci, Khan Iftekharuddin, Raj Jena, Nigel John, Ender Konukoglu, Danial Lashkari, Jose Antonio Mariz, Raphael Meier, Sergio Pereira, Doina Precup, S. J. Price, Tammy Riklin-Raviv, Syed Reza, Michael Ryan, Lawrence Schwartz, Hoo-Chang Shin, Jamie Shotton, Carlos Silva, Nuno Sousa, Nagesh Subbanna, Gabor Szekely, Thomas Taylor, Owen Thomas, Nicholas Tustison, Gozde Unal, Flor Vasseur, Max Wintermark, Dong Hye Ye, Liang Zhao, Binsheng Zhao, Darko Zikic, Marcel Prastawa, Mauricio Reyes, and Koen Van Leemput. The Multimodal Brain Tumor Image Segmentation Benchmark (BRATS). *IEEE Transactions on Medical Imaging*, 34(10):1993–2024, October 2014.

[24] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[25] Deepak Pathak, Philipp Krähenbühl, Jeff Donahue, Trevor Darrell, and Alexei A. Efros. Context encoders: Feature learning by inpainting. *CoRR*, abs/1604.07379, 2016.

[26] Karl Ridgeway, Jake Snell, Brett Roads, Richard S. Zemel, and Michael C. Mozer. Learning to generate images with perceptual similarity metrics. *CoRR*, abs/1511.06409, 2015.

[27] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[28] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575, 2014.

[29] Thomas Schlegl, Philipp Seeböck, Sebastian M. Waldstein, Ursula Schmidt-Erfurth, and Georg Langs. Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. *CoRR*, abs/1703.05921, 2017.

[30] Frank Seide and Amit Agarwal. Cntk: Microsoft's open-source deep-learning toolkit. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discov-*

ery and Data Mining, KDD '16, page 2135, New York, NY, USA, 2016. Association for Computing Machinery.

[31] Steven Silvester, Anthony Tanbakuchi, Paul Müller, Juan Nunez-Iglesias, Mark Harfouche, Matt McCormick, Ariel Ladegaard, Arash Rai, OrganicIrradiation, Tim D. Smith, Marcin Konowalczyk, Antony Lee, Almar Klein, rreilink, Joel Nises, jackwalker64, Ghislain Antony Vaillant, Chris Barnes, Zulko, Po-Chuan Hsieh, NiklasRosenstein, Miloš Komarčević, Ph.D. Michael Hirsch, Maximilian Schambach, Joe Singleton, Hugo van Kemenade, Graham Inggs, Ge Yang, Felix Kohlgrüber, and Chris Dusold. imageio/imageio v2.8.0, February 2020.

[32] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[33] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR*, abs/1602.07261, 2016.

[34] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[35] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[36] Seiya Tokui, Ryosuke Okuta, Takuya Akiba, Yusuke Niitani, Toru Ogawa, Shunta Saito, Shuji Suzuki, Kota Uenishi, Brian Vogel, and Hiroyuki Yamazaki Vincent. Chainer: A deep learning framework for accelerating the research cycle. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2002–2011. ACM, 2019.

[37] Michael Tschannen, Olivier Bachem, and Mario Lucic. Recent advances in autoencoder-based representation learning. *CoRR*, abs/1812.05069, 2018.

[38] Xiaosong Wang, Yifan Peng, Le Lu, Zhiyong Lu, Mohammadhadi Bagheri, and Ronald M. Summers. Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases. *CoRR*, abs/1705.02315, 2017.

[39] Zhou Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *Trans. Img. Proc.*, 13(4):600–612, April 2004.

[40] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013.

[41] Jianpeng Zhang, Yong Xia, Qi Wu, and Yutong Xie. Classification of medical images and illustrations in the biomedical literature using synergic deep learning. *CoRR*, abs/1706.09092, 2017.