

De-identification of medical images using object-detection models, generative adversarial networks and perceptual loss

Malik Aasen
Fredrik Fidjestøl Mathisen

Master's thesis in Software Engineering at
Department of Computing, Mathematics and Physics,
Western Norway University of Applied Sciences

Department of Informatics,
University of Bergen

June 1, 2021



Western Norway
University of
Applied Sciences



Acknowledgements

First off, we would like to thank our supervisor Dr. Alexander Selvikvåg Lundervold for his expertise and advice during this project. He was also the one who introduced us to the field of machine learning through the DAT158 course at HVL, and we cannot thank him enough for that. We would also like to thank Mohn Medical Imaging and Visualization Centre at Haukeland University Hospital where the experiments were performed, and especially Hauke Bartsch for providing tools and data, as well as expertise throughout this project. We would also like to thank PhD candidate Sathiesh Kaliyugarasan for guidance and interesting discussions around the field of deep learning and making our time here more enjoyable.

Abstract

Medical images play an essential role in the process of diagnostics and detection of a variety of diseases. Whether it being anatomical features or molecular cells, medical imaging help visualize and gain insight into the human body. These images are a crucial aid in the process of diagnosing patients. While these images are informative, they can also be quite difficult to interpret, necessitating highly trained medical professionals to read the images. The amount of medical images produced is enormous compared to the amount of professionals whose task it is to interpret them. The diagnosis can also vary based on the medical professional who inspects the image.

The recent rise of a new generation of Computer Aided Detection (CAD) systems based on machine learning has become more and more important to battle this problem. These systems aids the medical professional in the diagnostic process. This can lead to a more consistent and accurate interpretations of medical images by removing some human bias. In addition, such systems can be used to decrease the workload by either filtering out images deemed as belonging to healthy subjects, to be otherwise not of interest, or marking images as indicating a risk.

When creating CAD systems utilizing machine learning you are very dependent on data. Since the systems will typically be placed in very delicate, high-risk situations, the quality of the data is always a priority. A common problem in medical imaging research is not getting sufficient data. Not that there is a shortage of images, but to be used in research, they typically have to be de-identified or anonymized. This process has to be verified manually and is therefore time-consuming. With the impressive advancement of machine learning in recent decades, it seems natural to attempt de-identification using machine learning, especially because several powerful models are being applied to similar tasks in other fields. One key reason for the success of machine learning is its ability to detect and generate patterns. Currently, there are several applications that perform de-identification by placing black-boxes on top of information detected as being sensitive [1, 2]. However, the black boxes can end up hiding also other parts of the image, but ideally all non-sensitive features in the image should be preserved. In this thesis we investigate the effect of using image-to-image deep learning to automate 2D medical image de-identification by detecting the sensitive information, and removing it without the use of black boxes. Our results indicate that de-identification models based on machine learning can result in viable and powerful solutions. The deep learning models manage to accurately detect and remove text, without large negative impact on the original image. Fig. 1 illustrates the results of this thesis.

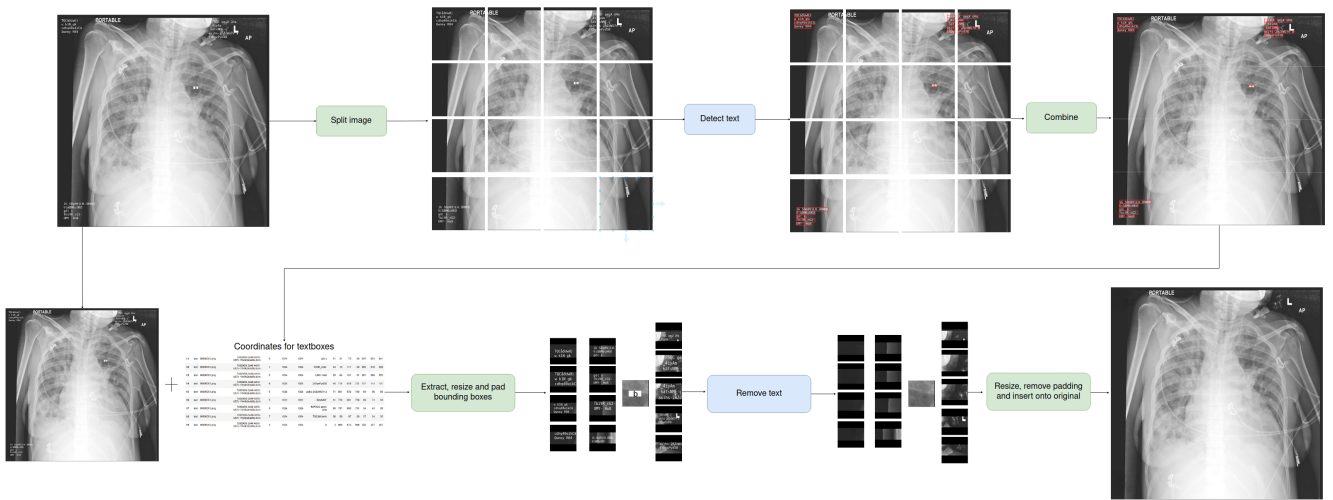


Figure 1: Complete system overview of our work. A medical image is passed through two stages based on two different deep learning models, indicated with the blue boxes. The first model (top row) detects the text and the second model (bottom row) removes the text.

Table of contents

1	Introduction	1
I	Background	3
2	Theoretical background	4
2.1	A quick overview of machine learning	4
2.1.1	What is machine learning?	4
2.1.2	Datasets and features: The models' inputs	6
2.1.3	Evaluation: How to tell if your model is good	7
2.1.4	Errors: Overfitting, underfitting and how it's combated	7
2.2	Computer vision	8
2.2.1	Deep learning for computer vision	8
2.2.2	Introduction to Convolutional Neural Networks	9
2.2.3	Some illustrative examples of CNN architectures	10
2.2.4	Deep learning for computer vision	11
2.2.5	Object detection	11
2.2.6	Image-to-image deep learning models	14
2.3	Medical imaging and imaging diagnostic	16
2.3.1	Medical imaging diagnostic workflow	18
2.3.2	Research PACS systems	18
2.3.3	Workflow-integrated machine learning in imaging diagnostics	19
2.3.4	Privacy in medical image analysis	20
2.3.5	De-identification of medical images	20
2.3.6	Related work: Some current de-identification solutions	21
3	Research questions and hypotheses	23
II	Experiments	25
4	Introduction to the experiments	26
4.1	Introduction	26
4.2	Methods and Materials	26
4.2.1	Data	26

4.2.2	Deep Learning Frameworks and Libraries	28
4.2.3	Google Cloud resources	29
4.3	Objective	29
5	Identifying and extracting sensitive information from images	30
5.1	Introduction	30
5.2	Methods and materials	30
5.3	Experimental results	33
5.3.1	Proof of concept	33
5.3.2	Final model	36
5.3.3	Results	38
5.4	Discussion	39
6	Generating new images without sensitive information: a first attempt	41
6.1	Introduction	41
6.2	Methods and materials	41
6.3	Experimental results	43
6.3.1	Proof of concept	43
6.3.2	GAN	43
6.3.3	Feature loss	45
6.4	Discussion	46
7	Generating new images without sensitive information: an improved approach	48
7.1	Introduction	48
7.2	Methods and materials	48
7.3	Extracting image parts containing text	49
7.4	Discussion	52
8	De-identification application	54
8.1	Introduction	54
8.2	Methods and materials	54
8.3	Results	54
8.4	Discussion	56
III	Discussion and further work	58
9	Evaluation and conclusion	59
9.1	Discussion of research questions	60
9.1.1	Challenges and known limitations	62
9.2	Conclusion and future work	64
	References	67

List of Figures

1	Complete system overview of our work. A medical image is passed through two stages based on two different deep learning models, indicated with the blue boxes. The first model (top row) detects the text and the second model (bottom row) removes the text.	iii
2.1	Synthetically generated faces from NVIDIA’s StyleGAN2 [3]. The figure is released under Nvidia Source Code License-NC. See https://github.com/NVlabs/stylegan2/blob/master/LICENSE.txt for more information	6
2.2	An underfitting, overfitting and a more correct model. How well the line fits the dotted datapoints dictates how well the model generalizes.	8
2.3	Max pooling with 2×2 filter and stride 2	10
2.4	VGG16 architecture as proposed in [4]. The figure is from [5] and is released under Creative Commons Attributions 4.0 International license (see https://creativecommons.org/licenses/by/4.0/ for more information).	10
2.5	ResNet block proposed by He et. al. Figure inspired by [6].	11
2.6	Two-stage detector architecture , more specifically R-CNN	12
2.7	a) Showing all of the possible anchor boxes for the objects in the picture. b) Showing objects with no matching anchor boxes, in this case none	13
2.8	(a) An image from det MS COCO dataset. (b) An image from the PASCAL VOC dataset	14
2.9	Example of image segmentation. The left image is the input and the right image is the output where each pixel has been given a class.	15
2.10	UNet architecture. Figure from [7]. This figured is released under a Creative Commons Licence (see https://creativecommons.org/licenses/by/4.0/ for more info).	15
2.11	An illustration of a simple GAN architecture consisting of a <i>generator</i> and a <i>discriminator</i> each trying to win over the other.	16
2.12	An illustration of the difference between a) a CT image and b) a X-ray image of the lungs.	17
2.13	A simplified model of a PACS system. In reality the PACS archive is part of the PACS system node, but is split apart for explainability in this figure.	18

2.14	A simple overview over some of the areas where machine learning can fit in the medical imaging workflow. Note that this is a rough sketch, and there are several other processes where machine learning can be used.	20
2.15	Example from the current solution running on the research PACS in Helse Vest. The image on the left is the result of the current de-identification process. As we can see, the text boxes down in the left-hand corner have not been detected. There are also parts of the image that have erroneously been marked as containing text. The image on the right is the input image before de-identification.	22
4.1	An example of our first "crappification" solution, where text is inserted into an existing image to provide training data for our text-removal models. Note that in this case the position and font size of the text does not vary. Later, we will introduce a more powerful and flexible text insertion method.	27
4.2	The pipeline of the render text approach. The input is a DICOM image and the output is two PNG images, one with text and one without text. The text is inserted at different locations with different font sizes that can be selected by the user.	28
5.1	Formula for intersection over union. Simply put, the area of overlap between the two bounding boxes divided by the total area of both bounding boxes.	31
5.2	Overview of the RetinaNet architecture. Fig. from [8] Copyright © 2020, IEEE. Here we see all of the components of RetinaNet linked together. a) The resnet backbone which creates rich feature maps. b) The feature pyramid network with lateral skip connections. On each level of the FPN we attach c) a classification subnet and d) a bounding box regression subnet.	32
5.3	Visualization of cross entropy loss vs. Focal Loss with different γ values.	33
5.4	An illustration of the first data set we used. Here the "crappified" bounding boxes is plotted on top of the corresponding image. . .	34
5.5	Results using the "crappified" training data where the text is placed statically. The model finds some text instances, but the areas located are too large compared to the ground truth labels.	35
5.6	New and improved anchor boxes. As we can see these anchor boxes matches the shape of a text line.	37
5.7	Metrics computed on the validation data set during training. Note the monotonic reduction in the bounding box loss (BBloss) during the training epochs.	38
5.8	A selection of predictions from the test set. This is the raw output of the RetinaNet model	38
5.9	The Final result of the text detection pipeline. The image is stitched back together and the predicted bounding boxes are converted to fit the original resolution of the input image.	39
5.10	Example of where the model has failed to detect text, similar to Fig 5.9.	40

5.11	An example of an instance where the model predicts text where there is none.	40
6.1	A simplified illustration of the feature loss system in [9]. The input image is transformed into output, and then a loss network pre-trained for image classification (here VGG-16) defines the features, and measures the differences in content. The original feature loss function included target style as well, however since we do not perform style-transfer, this is omitted. The figure was inspired by Fig.2 in [9].	42
6.2	The original image, the prediction, and the actual image. The GAN has generated an image that is close to the real image visually, however, this may not be the case when looking at the subtraction or histogram normalization.	44
6.3	Subtraction of GAN prediction and the histogram normalization of the subtraction. This is done to highlight differences due to the dark pixel-values in the image, making it a lot easier to detect differences visually. Looking at the normalization, it is easier to see that the model changed the details of the image as well, rather than just the outline.	44
6.4	The original image, prediction, and actual image using Unet with feature loss instead of GAN. The results are similar to the ones in Fig.6.2, with a bit more detail in the left armpit. Looking at the difference and histogram normalization will tell us more about the results.	45
6.5	Subtraction of prediction with feature loss as the loss function, as well as the histogram normalization. We see now that the model has changed pixel-values everywhere, something that would be hard to tell by only looking at the subtraction	46
6.6	Results of using full images as data. The left image is the input, the middle is the prediction and the left is the actual image the model is trying to generate. Overall the GAN has struggled to create an image good enough for the critic to accept, and thus most likely struggled to find out what it needed to do. The Unet with feature loss produced similar results.	46
6.7	Difference in training data. Our own "crappified" images had larger text with a fixed position and color. The render-text program generated more complex data with differentiating position, text-color and background depending on where it was placed. . .	47
7.1	The extract function is passed a whole image and gets the coordinates of the bounding boxes from the DataFrame. Then, using PIL, the boxes are cropped out, resulting in a new dataset of bounding boxes, rather than whole images.	49
7.2	Experimental settings for our RetinaNet model tasked with text detection.	50
7.3	Predictions of bounding-boxes without text. The model is passed the cropped bounding-boxes, and predicts the images without the text. The predictions gives us an indication that the model has performed well.	50

7.4	On the left we have the original image, and the right image is essentially the original image, but with the predictions placed on top. With the images stitched back together, they can now be saved to create subtractions and histogram normalizations. . . .	51
7.5	Subtraction image and histogram normalization of the image. The subtraction contains clear text, and the normalization shows that some changes has been made to surrounding pixels.	51
7.6	Subtraction image and original image. The text in the left image looks almost identical to the ones in the original, meaning that there are almost no residual pixels in our prediction.	52
7.7	Errors highlighted with a red circle. Light-blue background due to colors in the image itself. In some cases the models error is related to color, and others it fails to generate details.	53
8.1	De-identification applications. The application illustrated on the left detects the text and returns a DataFrame containing the coordinates. The one on the right uses the coordinates to remove text present in the area.	56
8.2	An overview of a simple de-identification pipeline. The user uploads a dataset of medical images to a data storage, triggering the de-identification process. The final images are then stored on the research PACS.	57
9.1	Complete system overview. A medical image is split into 16 parts before the text-detection model detects the bounding boxes. The image is then combined again in case the image is needed for other research purposes. Then, by using the coordinates of the bounding boxes, we can crop the original image to extract the relevant areas. Those images are then passed to the text-removal model. After the text has been removed, we place the predictions from this model on top of the original image using the original coordinates, resulting in a de-identified image.	59
9.2	Prediction on the left and original image on the right. We notice the areas where the model has done changes, however it is not possible to read characters.	60
9.3	A comparison of the "rewritepixel" solution used at MMIV on the left and our text detection model on the right. By looking at the two pictures we can observe that our model has a higher detection rate in this example.	61
9.4	Histogram normalization of the "crappified" images in our first, proof-of-concept text removal experiment. We clearly see that the model has changed pixels in irrelevant areas, potentially compromising the integrity of the features in the image.	62
9.5	CUDA out of memory error. Appeared when trying to use 1024x1024 images as training data in the GAN	62

9.6	Lack of computational resources allocated to the MMIV organization by Google repeatedly led us to not being able to run our virtual machines on the cloud. Due to the work being performed during the COVID-19 pandemic, we had to work on machines hosted in the cloud to be able to work consistently, and at random times this error would force us to wait for resources to be available.	63
9.7	Original image, prediction and actual image. We quickly made a function that black-boxes the text (text box in the middle is the one to be removed, rest is context). The model is worse at generating detail compared to the one that did not use black-boxes, but it could be a viable, more privacy-preserving alternative.	65
9.8	An overview of a proposed, more comprehensive pipeline for de-identification. The de-identification trigger sends a request for images to the clinical PACS. The PACS then passes the images to the text detection model. There we can decide whether we want black-boxed images, or use the text-removal model to de-identify them. After de-identification, the images are evaluated. This could be done by running the PNG-enhancer program, and if the image passes set thresholds, we can move on to a deep learning solution, trying to recreate the text. If the image passes evaluation, it is sent to the research PACS for storage, and if not we send a message to a controller. This controller would then request images from the research PACS to train the models again using new data. To verify if the model's performance improves after training, we added a model evaluation step, where the model would use a fixed validation set and see if it improves. One could also daily add the de-identified images to the new training set, so that the models follow the data, and keeps it's good performance.	66

List of Tables

5.1	Snippet of the data used in the proof of concept experiment. We decided to give all of the text an empty string as category since it is not important what class the text is given as long as it is not background.	34
5.2	Experimental settings for our RetinaNet model tasked with text detection.	36
6.1	Experimental settings for our GAN model	43
6.2	Experimental settings for our model using feature loss	45

Chapter 1

Introduction

Machine Learning is becoming more and more important in all sectors of our world, whether it be in research, or for grocery shopping [10], the possibilities are endless as long as the amount of data is sufficient. This especially applies in medicine, a field where machine learning combined with expertise in medicine can create solutions that outperform what either can do alone [11] and the amount of generated data is substantial. The data volume is also rapidly growing, with it being estimated that 153 exabytes of health data were produced worldwide in 2013, compared to the projection of 2.314 exabytes in 2020 [12]. Data engineering is already being used more and more in medicine, with examples being drug discovery [13], cognitive science, as well as many other related practices, and will only continue to become more important in the future. One of the issues with data within the field of medicine is that large parts of it contain sensitive information. If this data is going to be used for research it has to be anonymized or de-identified. This is a very time-consuming task and is often done manually, increasing the already large workload of healthcare workers. There have been several studies on how the intensity of healthcare workers' workload is negatively impacting their performance and mental health [14]. Creating good solutions and tools that lessen the burden of healthcare workers' workload is, therefore, very important and relevant. De-identifying medical records and images is a costly process as well, where professionals often charge hourly rates. If the dataset is large enough the cost of such a process would be substantial. When done manually you also have the issue of human bias and what that person considers to be sensitive information. Neamatullah et. al [15] reported that the recall score of human annotators ranged from 63 to 94 % when de-identifying medical records.

Data anonymization has become more and more important, whether it is within medicine or data gathered by private corporations. This is especially true after the introduction of the General Data Protection Regulation (GDPR) in July 2018 [16]. Our aim for this thesis is to create a system of deep learning models that will be able to detect and remove burnt in text in medical images at a level that complies with GDPR. An ideal solution would work well enough to replace the existing de-identification software at MMIV. This will help ease the workflow of the de-identification process if done successfully. We hope that such a system will contribute to generating richer datasets so the images can be used in powerful models and further medical research without breaking any laws

surrounding privacy and patient confidentiality. We also hope that by creating this system, we can reduce the amount of manual work by healthcare workers, as the results of the current solution have to be verified manually.

In this work, we have explored different image-to-image and object detection models. By taking this approach we have been through different theories and technologies within the field of machine and deep learning. Another aspect of this thesis will be the generation of realistic training data for such models. Since our models can not use real medical data, we have to generate synthetic data, which mirrors that of real life data.

This thesis will be split into three parts. Part I, *Background*, will provide the reader with the theoretical background in regards to machine learning, deep learning, and medical imaging. Chapter 2 starts with a quick overview of machine learning, explaining the fundamental theory. Then we move on to looking more specifically at deep learning, computer vision, neural network architectures and medical imaging. The goal of Chapter 2 is to provide the necessary knowledge required to understand the work we present in part 2, *Experiments*. In chapter 3, we state the research questions our work aims to solve, as well as hypotheses for how we will attempt to solve the task of de-identification.

In part II, *Experiments*, we will present our experiments in de-identification. First off, in chapter 4 we will give an introduction to the experiments conducted. Chapter 5 will go over our approach to detect burnt-in text using RetinaNet, discuss the methods and material used, as well as the results obtained from the experiment. In chapter 6, we present a proof-of-concept experiment regarding text-removal using Generative Adversarial Networks(GAN) and Unet with feature loss as the loss function. The following chapter 7 will continue the work in the previous experiment, introducing an improved version of the proof-of-concept work. Chapter 8 goes through how we created a de-identification prototype application, by combining the text-detection and text-removal models.

Lastly, we have part III *Discussion and further work*, where we will discuss and evaluate different aspects of the experiments and thesis as a whole.

Part I

Background

Chapter 2

Theoretical background

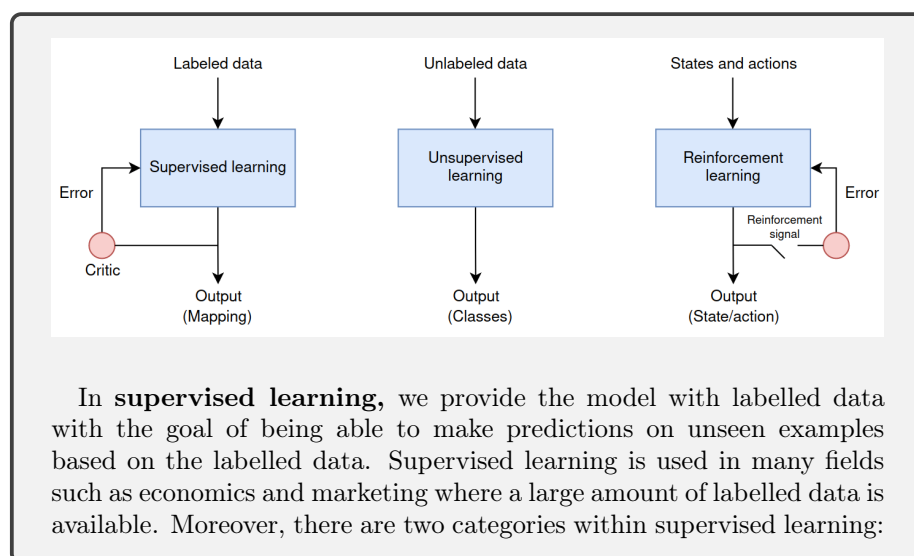
2.1 A quick overview of machine learning

In this chapter, we will go over some basic concepts, techniques and ideas in machine learning (ML), as it is necessary to understand the fundamentals before we cover the more specific theory used in our thesis.

2.1.1 What is machine learning?

Machine learning (ML) is a branch of artificial intelligence (AI), the study of algorithms that automatically improve with experience. This is achieved through a combination of computer science, mathematics and statistics, creating ways for computers to learn. The history of machine learning stretches back to the 1950s, but it is currently experiencing a tremendous amount of interest.

In machine learning, there are three main categories of learning: supervised learning, unsupervised learning and reinforcement learning, distinguished by how the systems consume data.



regression and classification problems. In classification, the goal is to predict the label/category of the input such as whether an animal is a cat or a dog. In regression, the goal is to predict continuous values, examples being house pricing based on size, location, age, etc, as well as predicting stock prices. In supervised learning, common algorithms used include linear regression, decision trees, k-nearest neighbor and neural networks.

As for **unsupervised learning**, instead of learning from labelled data, the algorithms are given unlabeled data with the hopes that the machine recognizes patterns and discovers information not previously detected. Common tasks within unsupervised learning include clustering and anomaly detection [17]. Due to being able to use unlabeled data, unsupervised learning allows for larger datasets with less data engineering, thus saving both time and resources. However, unsupervised learning will not be optimal for every type of task. The reason for this is that the algorithms used are supposed to find their own connections and structures, meaning that there are no right or wrong answers due to the fact that while results might make no sense to humans, it does for the machine. In unsupervised learning, algorithms such as K-means, Principal Component Analysis (PCA) and K-nearest neighbors (K-NN) are often used.

The goal of **reinforcement learning** is to train intelligent agents to take actions in order to maximize reward. Simply put, if the agent performs the right action, it's rewarded. Optionally you could also penalize the agent for taking the wrong action. Reinforcement learning can be used for tasks such as self-driving cars, automated drones and video game AI.

There are also **hybrid learners** such as self-supervised learning and semi-supervised learning where the data contains less or no labelled data.

Strengths and limitations

One of the nice things about machine learning is that it is viable in a wide area of applications, from self-driving cars to recommending movies to watch. Using ML to handle and analyze complex data is very useful as well, a task that could potentially be extremely time-consuming if done manually by humans. One of the examples as previously mentioned is unsupervised learning, often used to notice patterns that are hard for humans to find, giving us a new perspective over what we are trying to solve. While this all sounds great, ML is not without its limitations. First of all, to be able to do anything, you would need a sufficient amount of data. ML is also restricted by the computing power it needs. One of the reasons that ML has become relevant in recent times is due to the breakthroughs and improvements relating to computing power, but depending on how complex you make the model, there might not be sufficient resources to

complete the task. There is also the ethical side that we will briefly touch on. Using machine learning is great, but it is not perfect and can make mistakes. If these mistakes have consequences, who is to blame? It is also important that models are fair, meaning that the results should be independent of sensitive variables that should not impact the outcome(i.e ethnicity, gender, etc.). Human bias is related to that. If the data that the model uses is impacted by our own bias, then the results will be affected by said bias, so reducing bias in general is important in ML. Lastly, there is the "no free lunch theorem". A simplified summary is that no single machine learning algorithm is universally the best-performing algorithm for all problems, meaning that you need to create new models depending on what task you are performing [18].

2.1.2 Datasets and features: The models' inputs

Arguably one of the most important aspects of ML is the data. A model's accuracy and performance are defined by the dataset used, as if the data is poorly put together, the model will likely not produce the expected results. Insufficient high-quality data is a common problem, and according to a 2018 data-science report published by Figure Eight, 55% of the questioned data-scientists cited *"quality/quantity of training data as being their biggest challenge"* [19]. Some of the most common hurdles in data-engineering include missing values, insufficient data, noise, errors. For missing values, there are several methods to fix this. First, if you have a large amount of data, you could drop the instances with missing values. If you don't have this luxury, there are ways of using the existing values to create values that are inserted where there were none before. This can come at the cost of performance.

As for lack of data, in the case where it is impossible or too difficult to gather more, data augmentation comes into play. This is where you use the data you already have acquired to create more. For image-related problems, common techniques include flipping, cropping, rotation and translation, making new ones that are slightly different from the original images. Another, more advanced technique, is to use a Generative Adversarial Network(GAN) [20, 21, 22], where a neural network creates artificial instances of the original dataset. A famous example can be found in Fig. 2.1, where the faces pictured are in fact not of real people, but synthetic faces created by a generative adversarial network.



Figure 2.1: Synthetically generated faces from NVIDIA's StyleGAN2 [3]. The figure is released under Nvidia Source Code License-NC. See <https://github.com/NVlabs/stylegan2/blob/master/LICENSE.txt> for more information

2.1.3 Evaluation: How to tell if your model is good

Now that all the data has been prepared, and the model has been trained, it would be nice to know how well the model is performing. Depending on the model, and the task it is performing, one would usually decide what metrics are appropriate for evaluating the model performance. For classification problems, a simple one is accuracy:

$$Accuracy = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

There are other metrics commonly used in classification, focusing more on model precision and recall.

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

For example, the F1 score, where:

$$F1 = 2 * \frac{1}{\frac{1}{precision} + \frac{1}{recall}}$$

The goal of F1 score is to find the right balance between precision and recall, as high precision but low recall gives you accurate predictions (less false positives) but it struggles to predict difficult instances. For regression tasks there is another set of metrics used to measure the performance of the model. The most common one is the mean squared error (MSE):

$$L(y, \hat{y}) = \frac{\sum_{i=1}^N (y - \hat{y}_i)^2}{N}$$

or root mean squared error (RMSE):

$$L(y, \hat{y}) = \sqrt{MSE}$$

where \hat{y}_i is the predicted value.

2.1.4 Errors: Overfitting, underfitting and how it's combated

When you first get the results from the model, you might wonder why it didn't perform to your expectations. Most likely, the model has either overfit, or underfit your data. What this means is that the model has either become too tuned to specific patterns in the training data, or that other factors such as lack of data, wrong model and parameters might be the issue. Overfitting can occur when a model is over-trained. It can perform really well on your training data, but when shown new never-before-seen examples, its performance drops drastically. In other words, the model fails to generalize to new data.

There are different ways to combat this. First off, if possible you can add more data that your model has to work with. Adding more data usually helps improve results in machine learning, as long as the data is of sufficient quality.

Secondly, a technique called cross-validation where you split your initial training data into multiple train-test splits and train your model on these helps greatly in detecting overfitting, as basically, the model gets new examples multiple times and has to generalize in order to improve. One could also create an ensemble of models, where you train each model separately, and combine them into a more powerful model that is better at generalizing and predicting on never-before-seen examples. Basically, the way to combat overfitting is to train models in a way that leads to models that aren't too simple, but also not overly complex. The graphs in Fig. 2.2 show examples of how a model underfits, overfits, or generalizes well. In the underfitting example, we see that the prediction line does not fit with the data points at all, meaning that the model not only does not generalize well on new examples, but does not even fit the training data well. Some ways to combat underfitting are to add more data, use a more complex model, add more parameters, or increase training time.

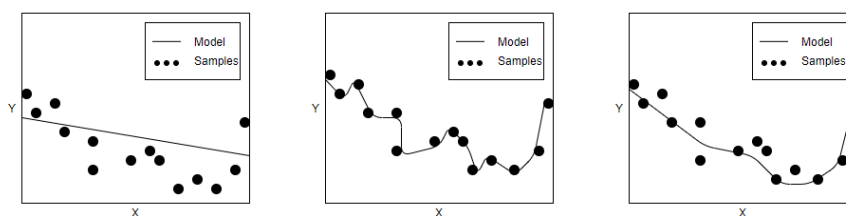


Figure 2.2: An underfitting, overfitting and a more correct model. How well the line fits the dotted datapoints dictates how well the model generalizes.

As for the overfitting example, we see that the predictions hit the datapoints well, but one can observe that if you were to insert a new datapoint at a value around the average of the other data points, the line would miss it completely. Thus, focusing on the generalization of your model is more important than training accuracy.

2.2 Computer vision

Computer vision (CV) is the field of study focusing on giving computers the ability to “see”. According to Prince [23], “*at an abstract level, the goal of computer vision problems is to use the observed image data to infer something about the world*”. While it is considered to be a field of artificial intelligence, there are many other fields involved in computer vision, such as physics, robotics and neurobiology.

2.2.1 Deep learning for computer vision

In the last decade, deep learning models have emerged as a very strong approach to machine learning, outperforming previous state-of-art machine learning techniques. Especially deep convolutional neural networks (CNN) have proven to be effective inside the fields of both computer vision and language processing.

2.2.2 Introduction to Convolutional Neural Networks

Building blocks. When constructing and using a Convolutional Neural Network (CNN) there are some key building blocks that are often used: convolutional layers, activation layers and pooling layers. These building blocks are layers with different “tasks”.

- **Convolutional layer:** Convolutional layers play a vital role in a CNN. They perform an operation called a "convolution". A convolution is a linear operation involving the multiplication of a given set of weights and the input. The set of weights that the input is multiplied with is often called a kernel. The input data comes in the form of an array and the kernel is a multidimensional array, e.g 3×3 . These two arrays are combined using dot product and result in a single value. The kernel slides across all of the input data and the weights are constant. This leads to a reduction in weights needed to be learned. By running the kernel over all of the input locations the convolutional layer produces a feature map.
- **Activation layer:** The feature map from convolutional layers are then fed through an activation function. This is a non-linear function, usually a rectified linear unit, or ReLU. The ReLU function is defined as: $F(x) = \max(0, x)$. It gives an output of x if x is positive, and 0 otherwise. This function allows the neural network to approximate almost any non-linear function. In addition to ReLU you can find information on other activation functions in [24].
- **Pooling layer:** All of the feature maps that have been fed through a convolutional layer often end up in a pooling layer. The main idea of a pooling layer is to down-sample the feature map to reduce the complexity for further layers. In earlier days, max-pooling was frequently used to achieve this, but now strided convolutions are used more and more. Max-pooling divides the feature map into sub-regions and only returns the maximum value inside of the grid. The filter, or sub-region, most commonly used is 2×2 [25]. Another common method used for down-sampling the feature map is to use convolutions with stride lengths greater than 1. These two methods are often combined in the pooling layer shown in Fig. 2.3

Other techniques regularly used in modern CNNs are:

- **Dropout:** Dropout is a regularization technique used to battle the problem of overfitting in CNNs. Deep neural networks tend to have a large number of parameters. Dropout addresses the problem with overfitting by randomly dropping units and their connections during training. This prevents units to adapt too much during training of the network [26].
- **Batch normalization:** Batch normalization is a technique introduced by Ioffe et. al [27] in 2015. This allowed for a drastic decrease in training for CNNs. By subtracting the mean and dividing by the standard deviation you will be able to produce normalized feature maps for each training batch. These layers are often placed after the activation layer, which results in sparser activations. This technique allows for a much higher learning rate and also to not be as careful about parameter initialization, while still achieving satisfactory results [27].

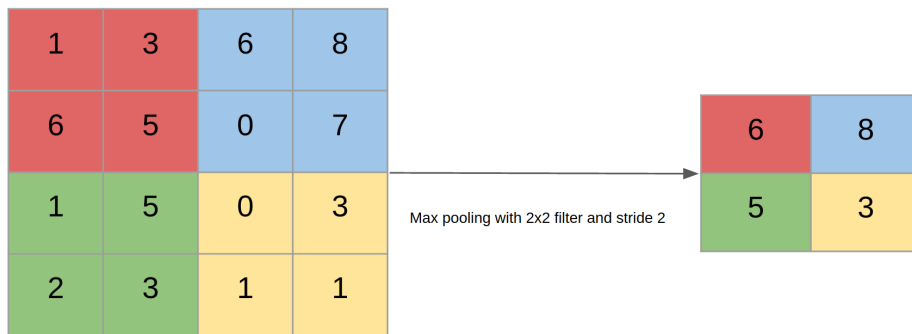


Figure 2.3: Max pooling with 2×2 filter and stride 2

2.2.3 Some illustrative examples of CNN architectures

Alexnet Alexnet is a deep neural network that revolutionized the field of computer vision and the machine learning world. This network won the Imagenet 2012 competition with a large margin by using techniques such as ReLU, max pooling, and convolutional layers [28], when other competitors relied on what turned out to be much less powerful non-deep learning approaches.

VGG VGG16 is a network architecture that was born out of the Oxford Visual Geometry Group, more specifically by Simonyan et. al. This network was created because of the need to reduce the number of parameters in the convolutional layers, and then again reduce the training time. There are some different versions of this network, but the main difference between them is the total number of layers (VGG16, VGG19, etc). This network was an improvement to the Alexnet mentioned above. The main difference between the two is that VGG has a fixed kernel size of 3×3 . Simonyan et. al [4] discovered that by reducing the kernel size to a fixed 3×3 size and increasing the network depth, they achieved a much higher score than the competitors. This helped them to secure four of the top spots in ILSVRC2014 [29].

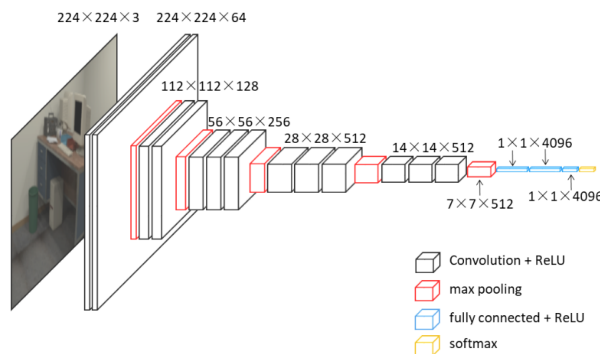


Figure 2.4: VGG16 architecture as proposed in [4]. The figure is from [5] and is released under Creative Commons Attributions 4.0 International license (see <https://creativecommons.org/licenses/by/4.0/> for more information).

ResNet Residual neural network, also known as ResNet, is a deep neural network architecture that won the ImageNet detection and localization competition in 2015 [6]. This architecture was proposed by He et. al [6] to solve the problem where CNNs became deeper and more complex. The solution they proposed is a network architecture containing skip connections, or shortcut connections, in each layer. A skip connection allows the user to skip one or more layers, without adding more parameters or complexity to the network. What this means is that a network with 50 layers should perform at least as well as a 20 layer network since the last 30 layers can be skipped. By using a ResNet-152 He et. al won the Imagenet classification competition in 2015. Variants of ResNets continues to be one of the best architectures for computer vision problems to this day.

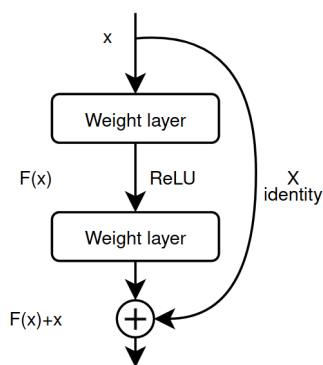


Figure 2.5: ResNet block proposed by He et. al. Figure inspired by [6].

2.2.4 Deep learning for computer vision

Some of the most common computer vision problems are image classification, object detection, and optical character recognition. These problems fall under the category "Object recognition", an area that historically deep learning models have excelled at. In deep learning, images are often represented as a tensor, so that they can be manipulated and read. Loosely speaking, a tensor is a container of data that helps store different dimensions of data in neural networks. Images and videos are 4D and 5D tensors (samples, frames(for video) height, width, channels). Converting them to tensors makes it easier for neural networks to take in the images as input.

2.2.5 Object detection

Object detection deals with the use of computer vision to detect objects of different classes in images and videos. Object detection is one of the fundamental problems of computer vision, with many other computer vision problems deriving from it, e.g. instance segmentation and image captioning. An object detection model's goal is to produce an output of what object is where.

Deep learning for object detection

After Alexnet was introduced in 2012 deep learning models really started to excel in computer vision tasks, and object detection was no exception. The

introduction of deep convolutional networks allowed people to concentrate on developing better algorithms and models instead of focusing on designing new ways to do feature representation. Two different methods of doing object detection quickly emerged: one-stage detectors and two-stage detectors

Two-stage detectors: These types of object detection models are best described as “coarse to fine” models. The models that can be categorized under two-stage detectors tend to be slow, but more precise than one-stage detectors. This is due to the architecture of two-stage detectors. Such an architecture consists of two parts or stages. The first stage is where the models generate proposals of regions where there might be an object of interest. All of the proposed regions are then sent through a CNN where feature extractions are exerted. Then each region is classified based on the feature extraction. Fig. 2.6 presents the architecture of a two-stage detector named Region Based Convolutional Neural Network (R-CNN).

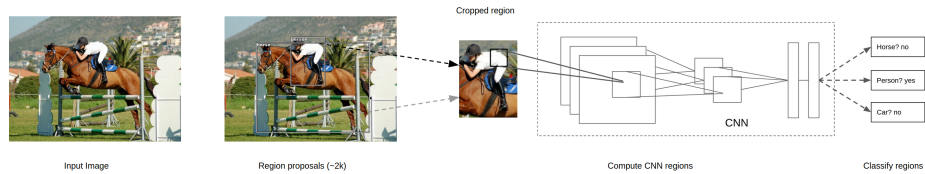


Figure 2.6: Two-stage detector architecture , more specifically R-CNN

One-stage detectors: Both the “Single Shot Detector” (SSD) [30] and “You Only Look Once” (YOLO) [31] presented a new approach to do object detection. These models focus more on speed which means that their accuracy does not quite reach the level of two-stage detectors. This is especially true for YOLO, which uses Darknet [32] to increase the speed of the model. By using this framework YOLO is able to run object detection in real-time, at 78 FPS using Darknet-53 [33]. The way a one-stage detector works can be compared to how a human eye scans a scene and identifies the objects in question. One-stage detectors do this with the help of anchor boxes during training. This is achieved by populating the image with a given quantity of possible matches to the bounding box surrounding the object. As we can see in Fig. 2.7 there is no annotated object in the picture without an anchor box.

RetinaNet with Focal Loss RetinaNet is another model which also had an impact on the field of object detection. The network used a feature pyramid network which allowed for feature extraction on each level. RetinaNet bridged the gap between one-stage and two-stage detectors. The most exciting part of RetinaNet was the loss function, the so-called Focal Loss. This loss function solved the problem of the foreground-background class imbalance which occurs during training [8]. This model will be discussed further in Chapter 5.4.

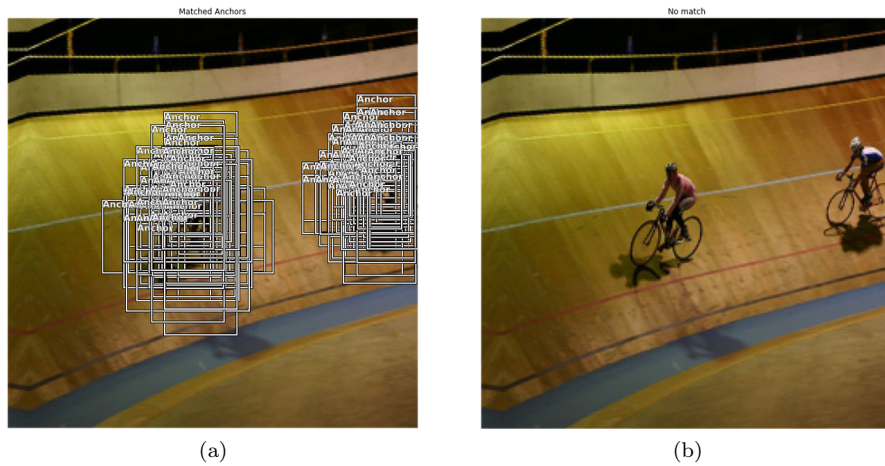


Figure 2.7: a) Showing all of the possible anchor boxes for the objects in the picture. b) Showing objects with no matching anchor boxes, in this case none

ImageNet data set ImageNet is a large-scale hierarchical database introduced in 2009. At its inception, it contained 3.2 million images divided into 5247 categories. The dataset has later grown to contain over 14 million images, aiming to contain upwards of 50 million images by its completion. This database was also used in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). This challenge served as a benchmark for various computer vision problems such as object detection and object recognition [34, 35].

PASCAL VOC In order to drive the field of object detection onward the Pascal Visual Object Classes (VOC) Challenge was created in 2005. In the beginning, this dataset contained only 4 different classes and 1578 pictures. The challenge contained two different competitions: classification and detection. Over the years from 2005 up until 2012 this challenge ran yearly and both the dataset and different tasks expanded. The final year the challenge ran the dataset contained 20 classes, 11530 images and the different competitions now contained classification, detection, segmentation to name some [36]. This dataset became a benchmark to measure the performance of object detection models.



Figure 2.8: (a) An image from det MS COCO dataset. (b) An image from the PASCAL VOC dataset

Microsoft COCO In 2015, Tsung-Yi et al. presented a new and more extensive dataset for object detection called Microsoft Common Object in Context (COCO). The dataset contained 91 different classes, 328k images with 2.5 million labeled instances [37], which was a massive improvement from PASCAL VOC. As with PASCAL, this dataset is connected to a competition that runs yearly and has become a benchmark for object detection. COCO also introduced another aspect to object detection, the context of the object in the image. In older datasets, i.e PASCAL, the objects are usually in focus whereas in COCO the objects might be smaller, bunched together, or seen in a larger context. This is shown in Fig. 2.8 where we can see that there are 9 objects of varying sizes in the image from COCO, but only two relatively large objects in PASCAL. We can see that the top scores are far greater on the PASCAL VOC dataset [38, 39]. The metrics used in the competitions are not quite the same, but they are based on the same principle, mean average precision (MaP). The COCO dataset has become such an important part of object detection and other CV tasks that they have developed a, API that helps to read and parse the annotations. The API works for Python, MATLAB, and Lua [40].

2.2.6 Image-to-image deep learning models

UNet The traditional use for neural networks and imaging has been classification of the whole image. However, in the field of medical imaging, there is another task where neural networks could prove to be useful and that is in image segmentation. This is the task of dividing a picture into different regions and classifying the region to a given label as shown in Fig. 2.9. A network architecture that has proven useful for this task is UNet. UNet can be divided into two different parts, a contracting part (left) and an expansive part (right) as shown in Fig. 2.10. Each layer on the contracting part is connected with a layer on the expansive part with concatenated skip connections. These skip connection works by allowing reuse of features by concatenating them to new layers, therefore allowing more information to be retained from previous layers of the network [41]. These skip connections are similar to the ones used in ResNet. Even though this architecture was introduced in 2015, it still produces state-of-the-art results.

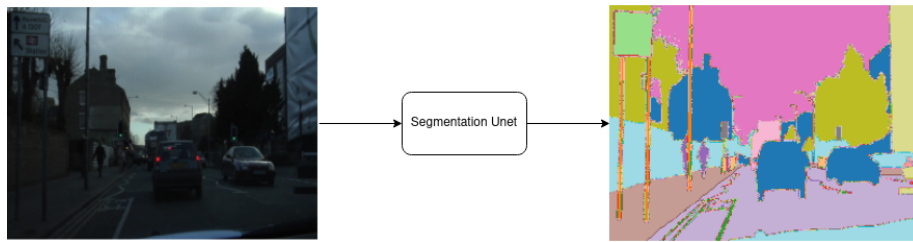


Figure 2.9: Example of image segmentation. The left image is the input and the right image is the output where each pixel has been given a class.

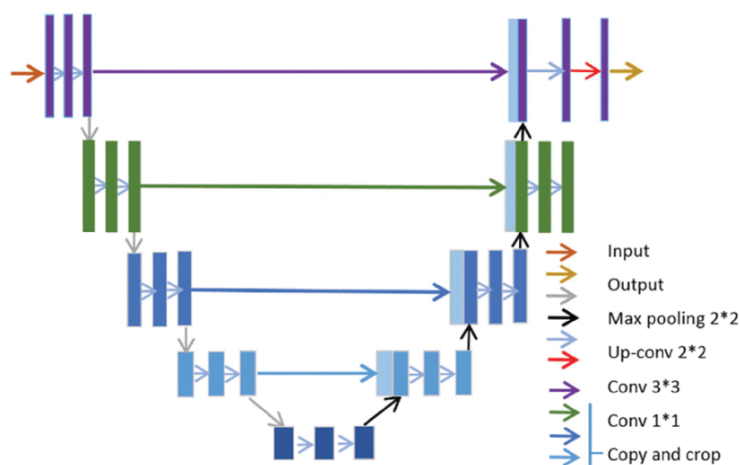


Figure 2.10: UNet architecture. Figure from [7]. This figure is released under a Creative Commons Licence (see <https://creativecommons.org/licenses/by/4.0/> for more info).

GAN General Adversarial Networks, or GANs, is a class of machine learning algorithms, proposed by Goodfellow et. al in 2014 [20]. A GAN model consists of two neural networks, one generator G which will generate fake data based on a data set, and a discriminator D which will try and distinguish the generated from the real data. When training the G network the goal is to maximize the probability of D not detecting the generated image as a fake. This can be seen as a two-player game. Both networks will improve their method of beating their counterpart until the generated data is indistinguishable from the real. Fig. 2.11 shows an overview of a traditional GAN architecture. In later years a variety of GAN models have emerged, i.e cycleGAN [42], Conditional GAN(cGAN) [43] and Wasserstein GAN [44]. It is worth noting that these GAN models would have a different architecture than the simple model shown in Fig. 2.11.

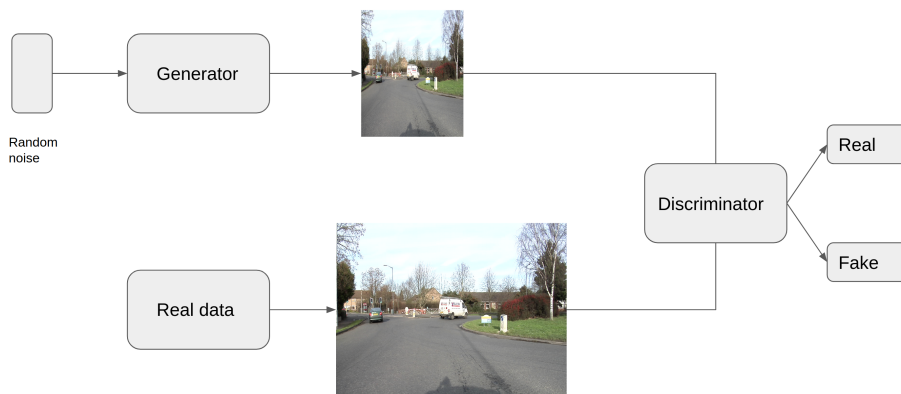


Figure 2.11: An illustration of a simple GAN architecture consisting of a *generator* and a *discriminator* each trying to win over the other.

2.3 Medical imaging and imaging diagnostic

Medical imaging is a set of processes and techniques that can be used to gain insight into the human body. By providing a visual representation of the interior parts of the body, such as bones or soft tissue, techniques from medical imaging aid medical professionals in their diagnosis, prognosis, and treatment decisions for many diseases and injuries. Images can serve as a confirmation or reassurance that the correct diagnosis was given. Radiology is a subfield of medical imaging and imaging diagnostics that is of particular relevance for our work. Patient treatment is however not the only use for medical images: they can also form databases that can be used to increase our understanding of normal and abnormal anatomy and physiology. There are many different ways to capture an image of the inside of the human body. Some common techniques are X-ray, Magnetic Resonance Imaging (MRI), Computed Tomography (CT), and ultrasound. All of the different techniques have different areas of use and produces different types of picture. For example, an X-ray image will produce an image where observing the skeleton or pathological changes in the lung is relatively easy. MRI and CT on the other hand are a better alternative when a medical professional might expect damage to soft tissue. X-ray and CT use ionizing radiation to produce images. MRI on the other hand uses powerful magnets which emit radiofrequency. This radio frequency pulse forces the protons to align with the magnetic field. When the magnetic field is shut off, the sensors in the MRI machine can detect the energy released by the protons [45]. Both CT and MRI produce-cross sectional images which can be stacked on top of each other to create 3D images. X-ray on the other hand produces 2D images. See Fig. 2.12 for a comparison of the two.

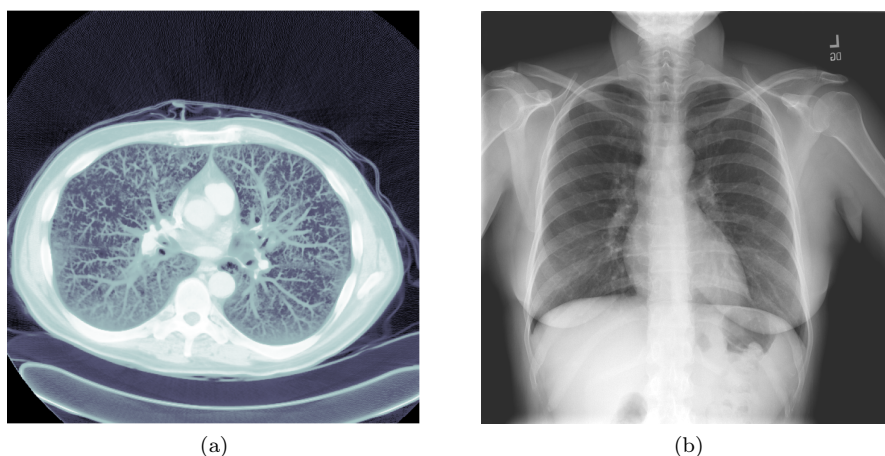


Figure 2.12: An illustration of the difference between a) a CT image and b) a X-ray image of the lungs.

Since there are many different techniques and equipment used to capture biomedical images, there is a need for a common standard for how to store and exchange information between imaging equipment and other systems. The standard used today is *Digital Imaging and Communication in Medicine*, also called DICOM. It dates back to 1983 when the American College of Radiology (ACR) and the National Electrical Manufacturers Association (NEMA) proposed a standardized method of exchanging information between devices from different manufacturers. It would later be known as the DICOM standard [46, 47]. The need for such a standard became particularly acute when more and more computers were integrated into medical equipment. By creating a common set of rules for how the information was stored and sent, the DICOM committee aimed to enable more seamless communication between systems. A DICOM file consists of an image and a DICOM header file. This header file contains all of the additional information needed for both storing the image correctly and analyze it at a later time. The use of the DICOM standard has enabled the use of *Picture Archiving and Communication Systems* (PACS), which will be discussed in Section 2.3.1 below.

Another use for medical imaging is within research, and in later years, especially within computer vision research. There have been many recent striking results in this field, for example, predictive models that can be used to classify whether a patient has breast cancer or not at the time of the screening [48], and much more [49, 50]. Large parts of the research within this field are done on real, anonymized images. By de-identifying or anonymizing medical images, there is potentially a huge amount of data available for researchers, as the number of imaging examinations conducted is immense. In later years there have been several high-profile competitions based on anonymized image data, for example on the Kaggle platform, typically won by teams using deep learning techniques¹. When large, well-characterized data sets are made publicly available, the threshold for creating new and innovative solutions is substantially

¹See <https://grand-challenge.org/> for an overview of such challenges

lowered.

2.3.1 Medical imaging diagnostic workflow

A *Picture Archiving and Communication System*, or PACS, is a technology widely used in the field of medical imaging. This system deals with the storage, presentation, retrieval, and distribution of medical images, using the DICOM standard. By using a PACS system the users eliminate the need for manually handling storage and retrieval of physical images. This allows the hospital to share images and records both internally and externally. Fig. 2.13 shows a simplified model of how a PACS system is configured. There will typically be multiple PACS nodes listening to the modalities on an image server. These nodes store the images in the PACS archive and inform the PACS database about changes. The analytic workstations work as PACS viewing stations. These workstations ask the database about the content, and the database retrieves the images from the storage backend. As shown in Fig. 2.13, all of the arrows are bidirectional. This means that any part of the system can perform operations on each other, i.e., the workstations can tell the PACS system to send operations to the modalities to perform certain operations (store, find and move) [51].

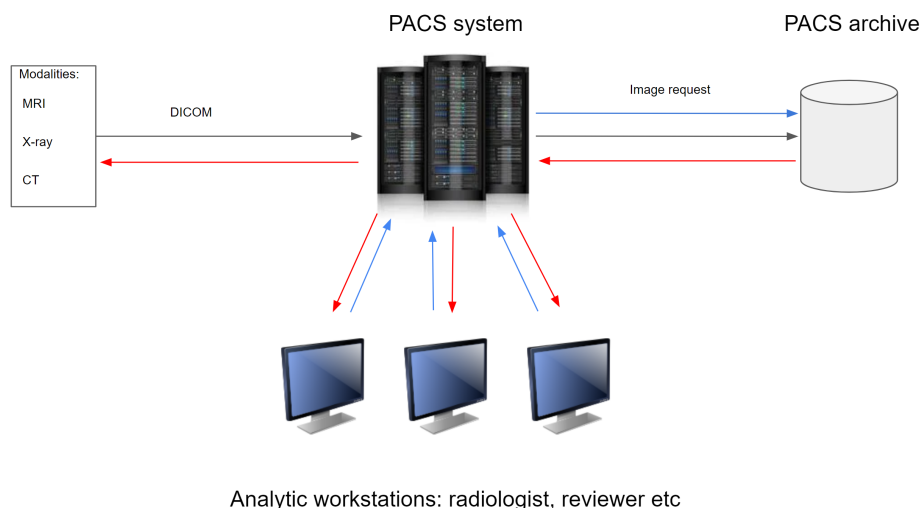


Figure 2.13: A simplified model of a PACS system. In reality the PACS archive is part of the PACS system node, but is split apart for explainability in this figure.

2.3.2 Research PACS systems

Of the different use cases for a PACS system, the one that is most relevant for our task is using PACS systems for research (*research PACS*) in radiology, as opposed to clinical use (*clinical PACS*). A research PACS is a modified version of the one shown in Fig. 2.13. It contains a copy of the PACS archive and the viewing stations. The archive in this system is a pure copy of the one on a regular PACS, which means copies of the DICOM files. As the purpose of the images stored in such a system is research, not clinical, de-identification is

important. The system presented in this thesis is thought to fit between the PACS system node and the PACS archive node so all of the images on the research PACS are de-identified.

2.3.3 Workflow-integrated machine learning in imaging diagnostics

In 2013, McKinsey Global Institute estimated that by applying big data and machine learning strategies to improve decision making, the US health care system could generate upwards of \$100 billion in value annually [52]. Within the field of medical imaging and radiology, the potential of streamlining the diagnostic process using machine learning has a huge potential. Machine learning has the potential to fit in every part of the diagnostic workflow pipeline (see Fig. 2.14). As an example, an application that has proven to work well is a system called *Automated Radiologist Assistant (AURA)*. This system is an extension and modification of a system called Man and Machine Mammography Oracle (MAMMO) [53]. AURA aims to filter out the negative patients in the mammography process, and therefore reduce the negative patient workload for the radiologists [54]. Kyono et al developed a system that uses a machine learning classifier, which has proven to work well on patients showing negative signs of breast cancer, without affecting the diagnostic accuracy. In Fig. 2.14, this process is mentioned in the box between "Medical image is taken" and "Analysis by radiologist". Another really promising machine learning system within the field of imaging diagnostic is Contextflow. Contextflow is a system that can be integrated into the analytical section of image diagnostic, typically between the PACS system node and the PACS archive node in Fig. 2.13. This system will try and match a specific image to other visually similar disease patterns in a database to further improve diagnose. They also have an algorithm to identify and prioritize patients in their database [55]. These are just two examples of many. See e.g. Table 2 in [50] for an extended list of applications.

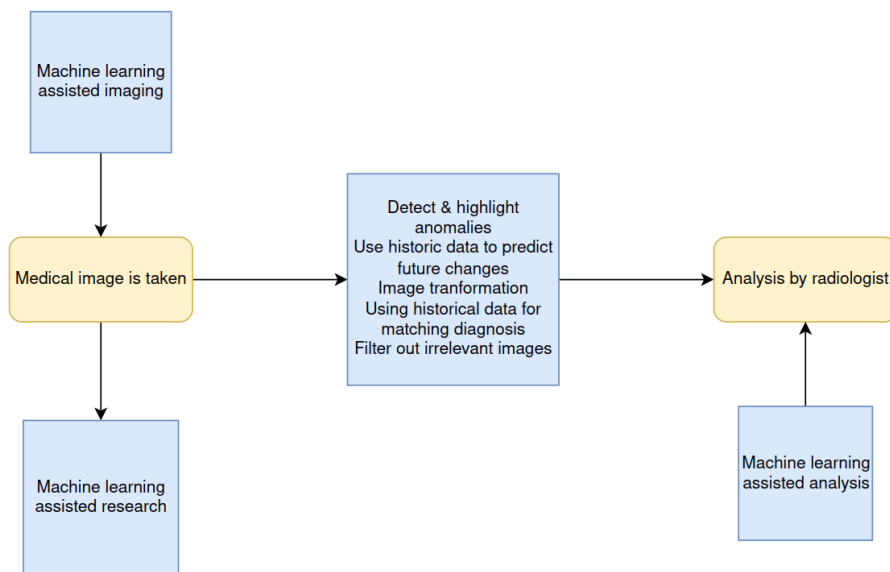


Figure 2.14: A simple overview over some of the areas where machine learning can fit in the medical imaging workflow. Note that this is a rough sketch, and there are several other processes where machine learning can be used.

2.3.4 Privacy in medical image analysis

In medical image analysis, unless the rare cases where synthetic, generated images are used, one tends to use images recorded from real patients or volunteers, whether that be X-rays, CT scans, or regular optical photos of areas of importance. In order to protect the patients' privacy, one would have to make sure that the identity of the patient cannot be extracted from the image- or meta-data. This is where image de-identification comes into play.

2.3.5 De-identification of medical images

The privacy concern is due to identifiers in the image that could be used to identify groups or individuals. The process of removing these identifiers is called de-identification. The U.S. Department of Health and Human Services (HHS) states two methods of de-identification of personal health information (PHI): A formal decision made by a qualified expert, or removal of identifiers, and data that could lead to identifiers being identified [56]. The Norwegian "Regionale komiteer for medisinsk og helsefaglig forskningsetikk" (REK, committee regarding research ethics) have their own guidelines, however, the principle remains the same. One could either go about encrypting the PHI, perform other manipulations in a way that would make it impossible for non-key holders to identify individuals from the data given, or simply just remove all PHI information altogether.

2.3.6 Related work: Some current de-identification solutions

With more and more publicly available medical image dataset emerging, the need for de-identification tools follows. This has led to a sea of different solutions emerging [57, 58]. Even the big tech giants such as Google [2] and Amazon [1] have developed solutions to cope with this demand. We will take a closer look at the existing solution used at Mohn Medical Imaging and Visualization Centre (MMIV) in Helse Vest, since this is what we aim to improve. We will also take a look at the solution created by Amazon to give a perspective on how a big tech company solves this issue.

Current solution in Helse Vest RHF - RewritePixel Hauke Bartsch has created a solution that is currently running on the research PACS at the MMIV. This solution is using the Tesseract 4.0 OCR engine to identify text that is burned into DICOM images. For each text fragment, a black square is written into the DICOM pixel information [59]. This solution has some issues where text might not be detected, see Fig. 2.15. This problem occurs when the contrast between the background and the text is not bright enough i.e yellow text on gray background. This results in the need to manually check the output of the program. This also means that the produced image gets deleted if it does not fulfill the requirements for de-identification. It is worth noting that this solution ignores single characters. This is due to the fact that "L" and "R" are often used to highlight which direction a person is facing in images.

Amazon Comprehend Medical and Amazon Recognition Amazon Web Services offer a solution to identify and de-identify text in medical images. These services are called Medical Comprehend and Recognition. Amazon Recognition offers the user the ability to identify objects and text in images and video, while Amazon Comprehend Medical allows the user to extract the text from the images. By combining these two services the user can integrate the ability to identify, extract and de-identify personal health information in images into applications. The output image has red boxes over the information the services deem sensitive [1].

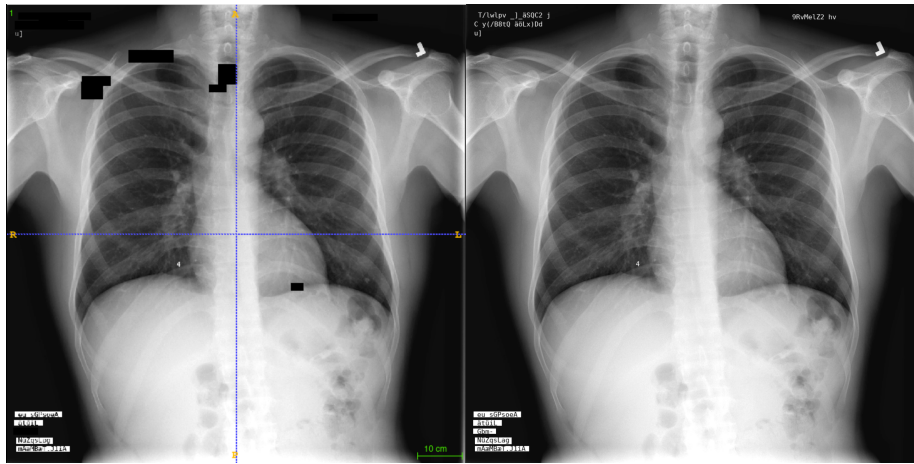


Figure 2.15: Example from the current solution running on the research PACS in Helse Vest. The image on the left is the result of the current de-identification process. As we can see, the text boxes down in the left-hand corner have not been detected. There are also parts of the image that have erroneously been marked as containing text. The image on the right is the input image before de-identification.

Chapter 3

Research questions and hypotheses

To get a clearer picture of how to complete the task of image de-identification, we have come up with a few research questions:

1. *"How can we use machine learning to automate 2D medical image de-identification while preserving image features?"*
2. *"How do we make sure these images are properly de-identified?"* When the pipeline gives an output image, we have to make sure that it's not possible to identify an individual given the image.

From these research questions, we can start to create hypotheses for our solution. We can already conclude that it is natural to construct multiple machine learning models, as two separate tasks that need to be done with machine learning, namely text-detection and image generation. Thus, dividing the first research question into two different parts seems natural:

- 1.1 *"Can we design an object detection model to successfully detect burnt-in text in the images?"*
- 1.2 *"Can we remove the sensitive information using image-to-image deep-learning, while still preserving the image integrity?"*

We can divide the field of text detection into two different subfields: identifying text in scanned printed documents (Optical Character Recognition) and text captured in daily scenes (Scene Text Detection). Since the current solution already uses OCR and the performance is lacking in certain areas we discard this option. Based on Table 1 in [60], we decided to use RetinaNet with Focal Loss. RetinaNet utilizes a Feature Pyramid Network (FPN) with a Resnet Backbone, as this is a very powerful approach that has seen a lot of success. Among other things, this model performs well on the difficult COCO-Text dataset. Since our task is likely less complex than COCO-Text, we expect that this model can produce satisfactory results.

As for the second model, since we are going to essentially remove text, we need to do some form of image generation, aiming to train a model to create

medical images as they would look without the burnt-in text. One option is to create a neural network and give it a dataset that consists of medical images without text, as well as the same images with text. Then, we use the images containing text as input, with the goal of generating the images without text, experimenting with different models and loss functions to see what yields the best results.

Part II

Experiments

Chapter 4

Introduction to the experiments

4.1 Introduction

An important aspect of machine learning is data. Without good and structured data machine learning models will in most cases not be useful. Therefore, it is important to spend a sufficient amount of time gathering and preparing data. For supervised learning, the data instances need labels, i.e. ground truths. A problem one usually has to face in a medical context is that the datasets are imbalanced in terms of their class labels. This is caused by there being a lot more data on healthy patients than sick patients. Another obstacle with medical data is the fact that most of the images and records contain personal information, which for legal and ethical reasons will have to be de-identified if used outside the diagnostic workflow.

For this thesis, the idea is to create a two-part pipeline that can de-identify routinely collected clinical medical images. The pipeline will be split up so that the first part will detect the text in the image using RetinaNet with Focal Loss, and the second part will focus on removing the text using both GANs and feature loss. Since we can not use real medical images, there will also be a focus on the generation of synthetic training data.

4.2 Methods and Materials

4.2.1 Data

Datasets

For these experiments, we used two different datasets. First, we used the "Chest X-Ray Images (Pneumonia)" dataset from Kaggle [61], containing 5.863 x-ray images in JPEG format. Secondly, we used the publicly available "RSNA Pneumonia Detection" dataset, also available from Kaggle [62], containing 29.684 DICOM images. Both of these datasets were originally intended to classify pneumonia, however, the data works fine for our purpose as we only need chest X-rays regardless of whatever illness it might show.

Synthetic Data

As previously mentioned, there is not a huge amount of data that represent realistic training data for our models. This leads us to another part of this thesis, which is that we have to generate our own synthetic training data. For this experiment, we used two different solutions, one that we created by ourselves and one developed by Hauke Bartsch.

A simple way to “crappify” images through text-insertion The first thing we did was to create our own solution for this problem. This solution put text in the top right and top left corner of every image, “crappifying” them by inserting unwanted elements. To calculate the bounding boxes of each text instance, we created a simple function with some helper functions from OpenCV which takes in the desired font and string and calculates the height and width of the string. The coordinates of the text were hard coded which made it relatively easy to construct the bounding boxes for each image. This solution is by no means perfect since the text instances are statically placed in each image. On the other hand, this solution allowed us to test both of our models and gave us a benchmark so we could continue building the full pipeline. Fig. 4.1 displays the result of the process.



Figure 4.1: An example of our first “crappification” solution, where text is inserted into an existing image to provide training data for our text-removal models. Note that in this case the position and font size of the text does not vary. Later, we will introduce a more powerful and flexible text insertion method.

Render text: a more flexible text-insertion solution This solution is far more powerful than the one introduced above. The `render text` program is written in C++ and uses the `freetype` library and `gdcm` to render text onto pixels in a series of DICOM images. The output generates two folders, where one folder is a copy of the input images and the other folder is the same images with randomly placed text in them [63]. The format of both of the new images is changed from DICOM to PNG. The program stores all of the text values in a file called `boundingBoxes.json`, which easily can be converted to a CSV file with the help of a `jq` command. If desired, the program can produce annotations in PASCAL VOC as well, since this format is common within the field of object detection. This solution solved the problem where the text was placed statically in each image. The program controls where the text is placed by using a JSON control file. This file mimics some of the usual positions where text is usually found in medical images. This includes the four corners and in the middle. The frequency of where the text is placed is decided by a random factor. To ensure that the program works on all computers, it can be built with a docker container that downloads all of the necessary libraries and dependencies. It is also possible to convert PNG images to DICOM. By having such an option in

the program, it is possible to create a new and even more diverse dataset by feeding these DICOM images through the render text program.

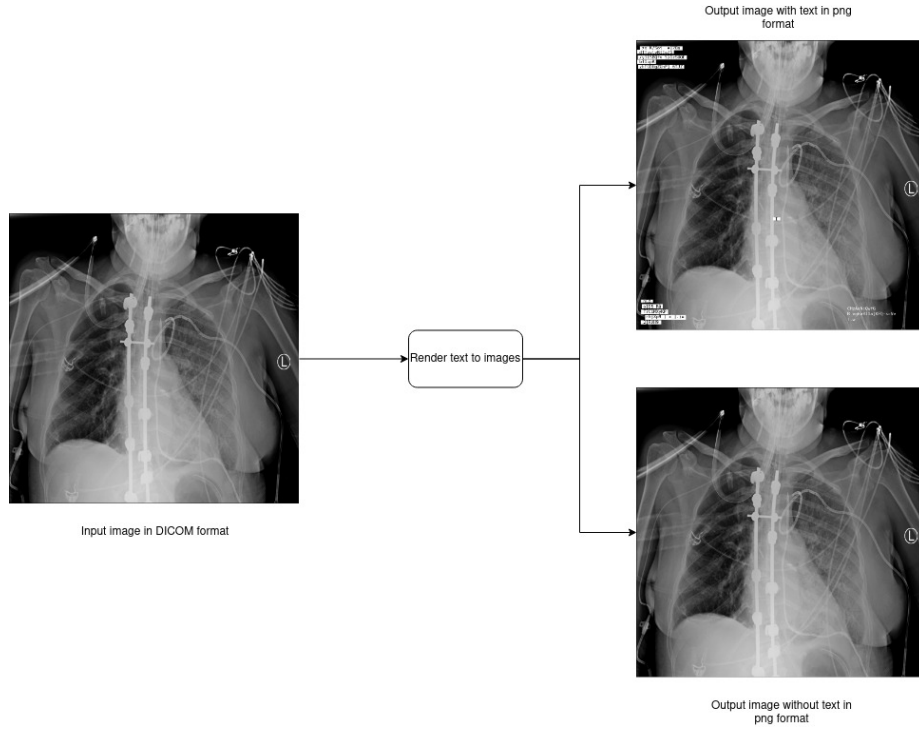


Figure 4.2: The pipeline of the render text approach. The input is a DICOM image and the output is two PNG images, one with text and one without text. The text is inserted at different locations with different font sizes that can be selected by the user.

4.2.2 Deep Learning Frameworks and Libraries

In this thesis, we use several Python based frameworks and libraries. In particular PyTorch and the PyTorch-based library fastai.

PyTorch

PyTorch is an open source framework for Python deep learning, built by Facebook’s AI Research lab (FAIR) [64]. The framework was based on the earlier machine learning library Torch [65] and provides among other things Tensor computing via GPU and CPU. This library is used as a replacement for NumPy to allow for computations to be done on the GPU. PyTorch is built to be deeply integrated into Python [66]. They also state that their framework is fast and flexible, so the user can move relatively fast from theory to prototype to deployment.

Fastai

Fastai is a Python based, open source deep learning library that is built on top of PyTorch, developed by Jeremy Howard et. al. [67]. This framework is built as a layered API such that it is easy to build state-of-the-art deep learning models. Fastai provides an API that allows the user to create state-of-the-art PyTorch machine learning models with minimal lines of code. This, combined with the fact that the library is highly customizable, makes fastai really powerful. The most recent version is version 2, released in August 2020 [67], which was a complete rewrite of fastai v.1. Our work was based on fastai v.1.

4.2.3 Google Cloud resources

For the experiments, virtual machines from Google Cloud Platform (GCP) were used. GCP is a computing service that runs on Google’s infrastructure. They provide a series of services which include compute engines, machine learning frameworks, and data storage to name a few [68]. The scalability and customization options for these machines are almost endless. This allowed us to access the resources we needed at any time.

4.3 Objective

The objective of this thesis is to create a pipeline that automatically detects and removes text. In the first experiment, we explore the possibilities of detecting text using RetinaNet and focal loss. In the second experiment, we explore the different approaches of removing text from images using GAN and Unet with feature loss as the loss function.

Chapter 5

Identifying and extracting sensitive information from images

5.1 Introduction

In this chapter, we investigate the effects of using RetinaNet and focal loss for text detection in images. Our goal for this experiment is to train a model that will identify all text in images. The identified text will either be black-boxed or extracted and sent onward to the text removal model. We will also experiment with different ways to create synthetic data for training as well.

5.2 Methods and materials

Data

For the proof of concept, we used the publicly available Chest X-ray (Pneumonia) dataset [61] from Kaggle. The final model used 10000 images for training and 3000 images for testing from RSNA Pneumonia Detection Challenge [62].

The model: RetinaNet with Focal Loss

RetinaNet is a one-stage detector that Lin et. al developed in 2017 [8]. At this point in time, the models that performed best in the field of object detection were two-stage detectors. Two-stage detectors performed well on this task but were slow. Lin et al proposed a new loss function that would help to bridge the gap between one- and two-stage detectors. With the introduction of the loss function Focal Loss, they were able to match the speed of previous one-stage detectors while surpassing the accuracy of two-stage detectors [8]. This model became part of Facebook AI Research’s (FAIR) object detection framework Detectron [69]. This framework has later been renewed to include state-of-the-art detection and segmentation algorithms and is now called Detectron2 [70].

The RetinaNet architecture can be split into two main parts, the feature pyramid network and two subnetworks for classification and regression. The feature pyramid network used in RetinaNet is adopted from [71]. By using such an architecture you are able to create feature maps for each level as shown in Fig. 5.2. This also gives us the ability to detect objects at each layer. The FPN is built on top of a ResNet architecture. By doing this we are able to use the feature maps already produced by ResNet. For an image with dimensions 256×256 pixels, ResNet produces feature maps of sizes:

- C1 128×128
- C2 64×64
- C3 32×32
- C4 16×16
- C5 8×8

by using stride 2 convolutions. In addition to this, we create two more feature maps, C6 and C7, with sizes 4×4 and 2×2 respectively, using stride 2 convolutions. Then we set $C7 = P7$ and goes down to P2 by upsampling the previous pyramid-layer. In addition, we also add lateral connections. This is done so we are able to run the subnetworks on each layer in the pyramid network. Then each feature map from the pyramid layers goes through two subnetworks, one for classification and one for bounding box regression. These subnetworks consist of four convolutional layers. As previously stated in 2.3.6, one-stage detectors heavily rely on the concept of anchor boxes. Each feature map is then assigned a certain number of anchor boxes. The classifier will end up with *number of anchors* \times *number of classes* channels and the bounding box regressor will have *number of anchors* \times 4 channels. We then define an intersection-over-union (IoU) threshold of 0.5. The formula for IOU is shown in Fig. 5.1. All anchor boxes with an IoU value of 0.5 and above will be assigned to ground truth boxes, and to the background if the value is between 0 and 0.4. IoU values in the range of [0.4,0.5] will be ignored during training. Each anchor box is only assigned to one ground truth object.

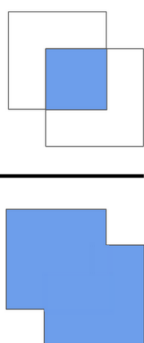
$$\text{IoU} = \frac{\text{Area of overlap}}{\text{Area of union}}$$


Figure 5.1: Formula for intersection over union. Simply put, the area of overlap between the two bounding boxes divided by the total area of both bounding boxes.

Classification Subnetwork: The task of the classification subnetwork is to predict the probability of an object inside each of the anchor boxes. This small fully convolutional network (FCN) is connected to each layer of the FPN and shares parameters across all layers. The loss function used for this network is Focal Loss, which will be explained in detail later. This network is relatively small and contains four 3×3 convolutional layers followed by a ReLU activation. Then another 3×3 convolutional layer and lastly a sigmoid activation [8]. This will produce an output of one object class per anchor box.

Bounding Box Regression Subnetwork: The task of the bounding box regression subnetwork is to regress the offset of each anchor box to the nearest ground truth bounding box. The architecture of this subnetwork is identical to the classification subnetwork, but these two networks do not share parameters. The output of this network is also different. The regressor will produce a linear output of four points which form an anchor box. This output describes the relative offset between the anchor box and the ground truth box [8]. Since this is a regression task, a sigma L1 smooth loss function is used.

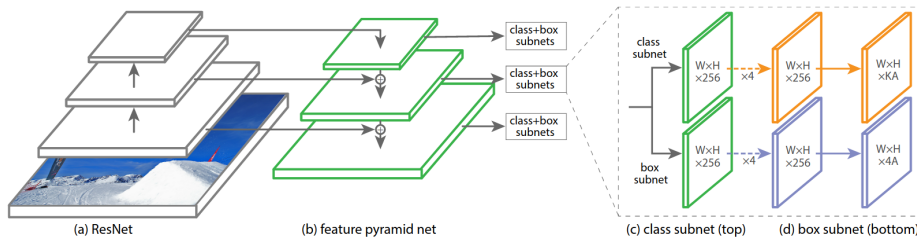


Figure 5.2: Overview of the RetinaNet architecture. Fig. from [8] Copyright © 2020, IEEE. Here we see all of the components of RetinaNet linked together. a) The resnet backbone which creates rich feature maps. b) The feature pyramid network with lateral skip connections. On each level of the FPN we attach c) a classification subnet and d) a bounding box regression subnet.

Focal Loss was designed to handle the class imbalance problem in object detection. This can occur when large parts of the picture contain one class, i. e. one small object in a large picture. In most object detection problems, the object you want to find will be small compared to the background in the picture. By not handling this problem correctly, the model will get a high accuracy by just identifying the background correctly. Here is where Focal Loss differs from other loss functions. Focal Loss addresses class imbalance by down-weighting inliers (easy objects to detect) instead of outliers (hard objects to detect) [8]. This leads to a small contribution in the total loss for easy examples, even though there might be many of them. Focal Loss is inspired by the binary cross-entropy (CE) loss function. CE is a loss function often used in classification problems and can be written as follows:

$$CE(p, y) = \begin{cases} -\log(p), & \text{if } y = 1 \\ -\log(1 - p), & \text{otherwise} \end{cases}$$

The CE formula can be simplified:

$$p_t = \begin{cases} p, & \text{if } y = 1 \\ 1 - p, & \text{otherwise} \end{cases}$$

$$CE(p, y) = CE(p_t) = -\log(p_t)$$

Fig. 5.3 shows this function as the blue curve. The large class imbalance a model encounter during training will most likely be too much for cross-entropy loss. This is caused by too many easily classified negatives. Lin et al decided to reshape the CE loss by adding a parameter $\gamma \geq 0$ and a modulating factor $(1 - p_t)^\gamma$ [8]. We now have Focal Loss defined as:

$$FL(p_t) = -(1 - p_t)^\gamma \log(p_t)$$

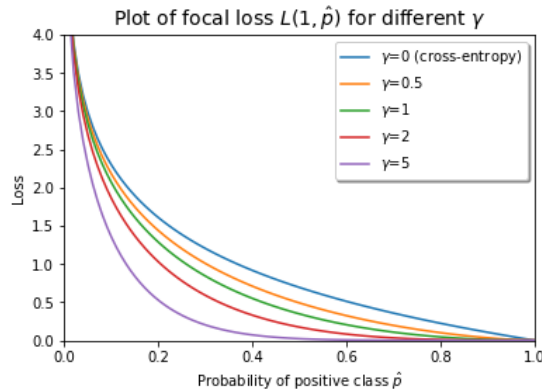


Figure 5.3: Visualization of cross entropy loss vs. Focal Loss with different γ values.

5.3 Experimental results

5.3.1 Proof of concept

We started by running our RetinaNet model on the synthetic data we created using a “crappify” method which placed the text statically in the pictures, shown in Fig. 5.4. this was done on the smaller Chest X-ray (Pneumonia) dataset [61]. Now the data had to be prepared and arranged. We store all of the necessary data for our bounding boxes from the crappify method in a CSV file and read it into a Pandas dataframe, as shown in Fig. 5.1. We then transform the dataframe into a Python dictionary where the filename is the key. This was done because the ObjectItemList from fastai demands that the input labels is a tuple and this was the easiest way to ensure that the data was presented as such. We can then load our data.

	file_name	bbox	cat
0	person5_bacteria_15.jpeg	[[5, 5, 48, 85.0], [65, 5, 88, 74], [95, 5, 11...	["", "", "", "", ""]
1	person1584_bacteria_4148.jpeg	[[5, 5, 48, 68.0], [65, 5, 88, 74], [95, 5, 11...	["", "", "", "", ""]
2	person512_virus_1029.jpeg	[[5, 5, 48, 87.0], [65, 5, 88, 85], [95, 5, 11...	["", "", "", "", ""]
3	person552_bacteria_2315.jpeg	[[5, 5, 56, 86.0], [65, 5, 88, 74], [95, 5, 11...	["", "", "", "", ""]
4	person1449_virus_2476.jpeg	[[5, 5, 56, 86.5], [65, 5, 88, 74], [95, 5, 11...	["", "", "", "", ""]

Table 5.1: Snippet of the data used in the proof of concept experiment. We decided to give all of the text an empty string as category since it is not important what class the text is given as long as it is not background.

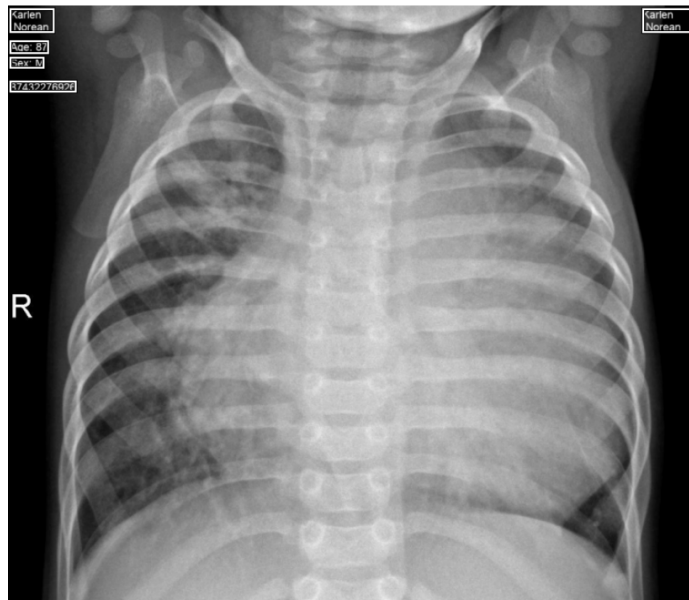


Figure 5.4: An illustration of the first data set we used. Here the "crappified" bounding boxes is plotted on top of the corresponding image.

RetinaNet We then initialize our RetinaNet model. First of all, we need to define our anchor boxes, and corresponding scales and ratios. We decided to test with the scales and ratios for the anchor boxes used in [8]. This gives us a total of 9 anchors for each level. As a backbone for our RetinaNet, we used a pre-trained Resnet-50 and Focal Loss as our loss function. We trained for 5 epochs on the last layers and 10 epochs unfreezed. The results from this experiment are shown in Fig. 5.5. The results were varying in quality as shown. The detected boxes are too large, compared to the area where the text occupies the image. Still, this gave us an indication that it might be possible to perform text detection using this model and approach.

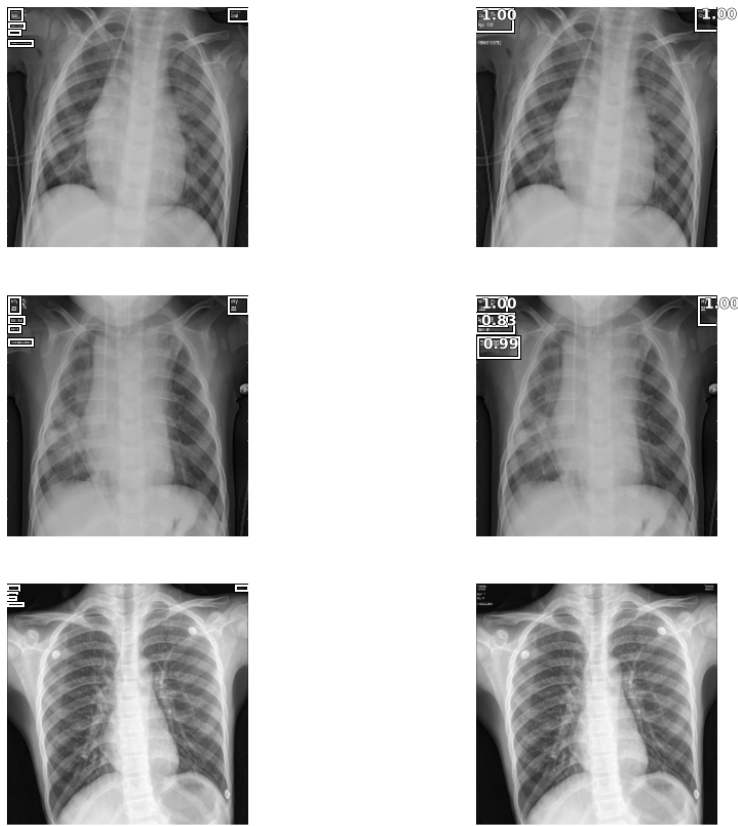


Figure 5.5: Results using the "crappified" training data where the text is placed statically. The model finds some text instances, but the areas located are too large compared to the ground truth labels.

There are two reasons why the result shown in Fig. 5.5 is of varying quality. The first, and perhaps most dominant reason is the sizes of the anchor boxes. The ratios and scales used in [8] to create the anchor boxes will not work very well for a text detection task, since these configurations will create boxes similar to the ones shown in Fig. 2.7. These will always be too large for a line of text. The other reason for the varying result is the size of the image during training. When the image is crappified it is done in the original size of the image, i. e. 1024×1024 pixels. When the data is loaded into a data loader, the image is resized to 256×256 pixels. This leads to two things happening:

1. The resolution of the images is lower, which makes the text in the image harder to locate. This especially applies if there is some texture underneath the text.
2. The already small text instances get even smaller and blurry.

The combination of the two reasons given above and the sizes of the anchor boxes will result in the model struggling with unseen data. One other weakness would be the placement of the text. Since the text instances are in the same

place in every image, the model can just guess that there is text in the same region every time. There is a need for more generalized data.

5.3.2 Final model

Augmenting the input data For the final experiment, we decided to split up the original images into smaller images with dimensions 256×256 . We decided to do this for two reasons. For one, this will most likely increase our detection percentage since all of the text in the images becomes larger relative to the input image and therefore easier to detect. The other reason is that the implementation of RetinaNet we use is built for 256×256 images. By splitting up the images we also increase the amount of training and test data. We take our original 1024×1024 images and divide them into 16 new 256×256 images. By doing so, we also have to change the annotations so that the bounding box coordinates match the new images. The operations required to split the images and annotations are done using a python library called "Image Bbox Slicer" [72].

The results from our first experiment using RetinaNet were promising, but there was room for improvement. First of all, there was a need for more generalized data. This was solved using "render_char_to_28x28". We produce a new dataset for training containing 10.000 images with text instances. We also produce a test set with 3.000 images for evaluating the model on unseen data. We decided to divide the images into 256×256 tiles. By doing so, we increase our training data size by $\times 16$. This gives us a lot more data to both train and test our model on. Fastai has no way of handling empty bounding boxes, which leads us to have to filter out all of the images without a text instance in them. This gives us a total of 56.528 images in our training set which is roughly $\times 5$ the size of the original training set. As mentioned in the previous experiment, the sizes of the anchor boxes were one of the reasons why the model did not perform at an optimal level. For this experiment, we introduce our own scales and ratios shown in Fig. 5.2.

Model: RetinaNet Backbone: Resnet50 pretrained on ImageNet Loss function: Focal Loss Anchor box sizes: [(32, 32), (16, 16), (8, 8), (4, 4)] Anchor box ratios: [0.05, 0.25, 0.4, 0.75] Anchor box scales: [0.35, 0.5, 0.6, 0.8, 1, 1.25, 1.6] Training: 5 epochs on final layers, then 5 epochs unfreezed

Table 5.2: Experimental settings for our RetinaNet model tasked with text detection.

As we see in Fig. 5.6 the sizes and shape of these anchor boxes mimics a text line in a far better way. The number of anchors per level is calculated by $num_ratios \times num_scales$. For our model, this means that we get 28 anchors on each level of the FPN. Since we want the model to work with discriminative learning rates we split the model into a head and a body. Now we train the model for 5 epochs on the final layers, and then 5 more on all layers.



Figure 5.6: New and improved anchor boxes. As we can see these anchor boxes matches the shape of a text line.

By looking at our metrics in Fig. 5.7, this looks very promising. If we just look blindly at the train and validation loss we might suspect that the model starts to overfit due to the gap between them. On the other hand, the metric that is most important for our experiment is the bounding box loss (BBloss in Fig. 5.7). The BBloss still improves, which leads us to believe the model does not overfit. Before we can have a look at our results, we have to process the output of our model. As previously mentioned, RetinaNet produces a huge amount of anchor boxes, so we have to filter out every box except the one which fits the text instances best. This is done by using non-maximum suppression (NMS). NMS works by filtering out proposed anchor boxes that are below a given threshold which we define our self. NMS takes the proposal with the highest confidence score and adds it to a list of final proposals. Now we compare this proposal with every other proposal by calculating the IoU value. If the value is greater than the given threshold, we keep the proposal as a final result. We repeat this process until there are no more proposals left to compare with. To produce the final result we used:

- Detection threshold = 0.4
- NMS threshold = 0.4
- IoU threshold = 0.5

epoch	train_loss	valid_loss	pascal_voc_metric	BBloss	focal_loss	AP-text	time
0	0.366368	0.330426	0.453591	0.127716	0.202710	0.453591	09:47
1	0.177123	0.182189	0.479808	0.076808	0.105382	0.479808	09:35
2	0.123085	0.141660	0.514236	0.059462	0.082199	0.514236	09:39
3	0.082097	0.114975	0.488536	0.051460	0.063515	0.488536	09:31
4	0.063102	0.114939	0.492555	0.050382	0.064557	0.492555	09:32

Figure 5.7: Metrics computed on the validation data set during training. Note the monotonic reduction in the bounding box loss (BBloss) during the training epochs.

5.3.3 Results

Fig. 5.8 shows the results on a subset of the test set. Compared to the results in Section 5.3.1, we can clearly see that the model performs on a much higher level than before. This is mainly due to the changes in the anchor boxes and splitting the image up into smaller tiles. By taking this approach there is also a need to reassemble the image back to its original form. The same issue applies to the bounding box coordinates that the model predicts. Since all of the predicted bounding boxes will be in the range of $[1, \dots, 256]$ we also needed a method to convert them back to coordinates that match the original input size. The model predicts bounding boxes in the format of $[x_min, y_min, width, height]$. We decided to change this format to $[x_min, y_min, x_max, y_max]$ since this is the format fastai expects by default. This will also lead to less work in handling the data in the text removal pipeline. Fig. 5.9 displays the final result after a picture has gone through the RetinaNet. As we see, the model has detected all of the text instances except for one. This is most likely due to the settings used for creating the anchor boxes. More on this in part III.

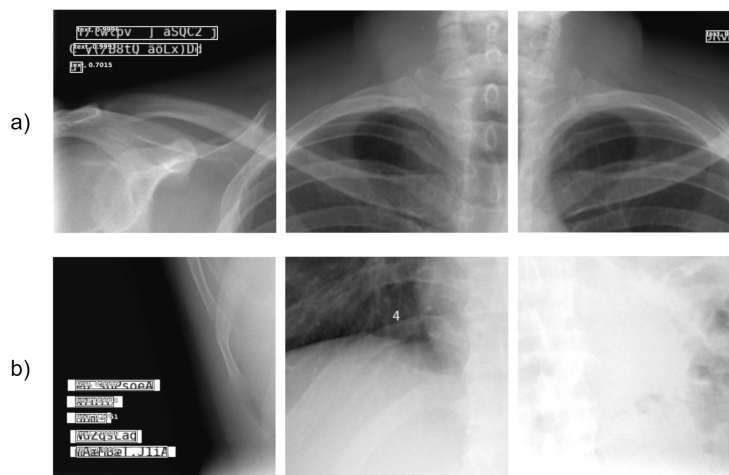


Figure 5.8: A selection of predictions from the test set. This is the raw output of the RetinaNet model

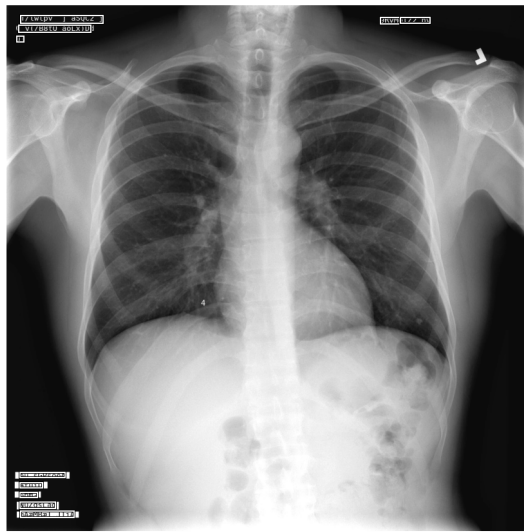


Figure 5.9: The Final result of the text detection pipeline. The image is stitched back together and the predicted bounding boxes are converted to fit the original resolution of the input image.

5.4 Discussion

The combination of reducing the size of the training data and changing the sizes of the anchor boxes turned out to work very well. By tuning the aspect ratios of the anchor boxes to better fit the shape of a text line, the detection rate and accuracy were much more precise. There are still cases where the text instances in the pictures are not detected i.e, Fig. 5.10.

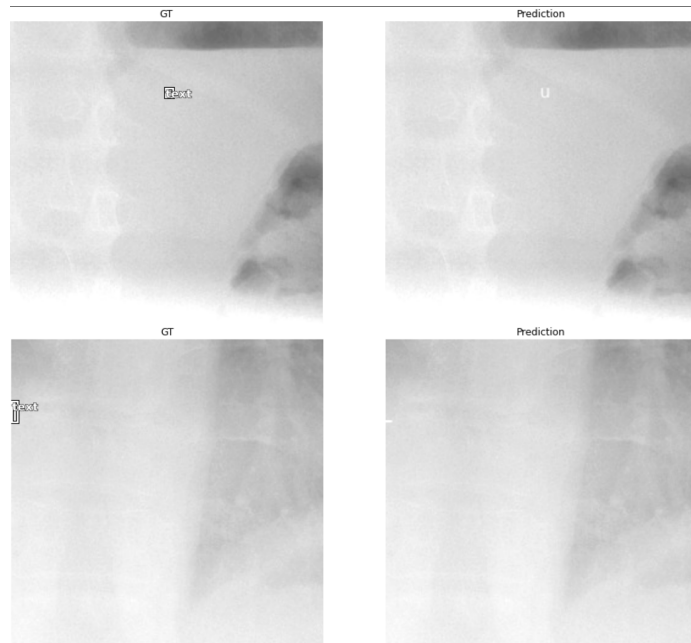


Figure 5.10: Example of where the model has failed to detect text, similar to Fig 5.9.

Another problem that might occur is when the model detects text where there is none. This might be caused by pareidolia where textures in the pictures have a resemblance to text. This is displayed in Fig. 5.11. After inspecting a selection of the validation images there has only been one case of this happening. It is somewhat expected that the model will fail to identify some of these cases. This is due to the fact that most of these cases are single free-standing characters, which leads to an anchor box that is narrow and tall. This is also shown in Fig. 5.9 where we can see that the number “4” is not detected in the middle of the picture. If we compare this to the “rewritepixel” solution, we will see that this program skips single characters. The reason behind this is that some images come from modalities that are still very film-centric. An example of this would be an upright CT scan where the radiologist would add a “L” or “R” to know which way the person was facing during the scan.

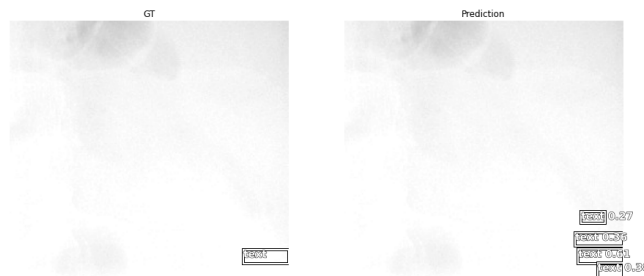


Figure 5.11: An example of an instance where the model predicts text where there is none.

Chapter 6

Generating new images without sensitive information: a first attempt

6.1 Introduction

In this chapter, we will investigate the effects of using both GANs and neural networks using the feature loss function in order to remove text from chest X-Rays. As of now, the de-identification services provided by Amazon and Google, as well as the existing solution at MMIV, redacts the sensitive information, leaving a rectangular box over the text. Our goal is to train a model that performs well enough to not need manual review, as well as allowing for complete removal of the text.

6.2 Methods and materials

Dataset

For the proof-of-concept experiment we used the publicly available Chest X-Ray dataset from Kaggle [61], containing 5.863 X-ray images originally intended to classify pneumonia, however, the data works fine for our purpose as we only need chest X-rays regardless of whatever illness it might show.

Additional methods

Progressive resizing

Progressive resizing is a technique that not only decreases training time, but also improves the model's ability to generalize. Its general idea is to start with decreased resolution on the images and in the early epochs, and then progressively increase the resolution up to the original resolution. This technique was first proposed in the paper "Enhanced Deep Residual Networks for Single Image Super-Resolution" by Lim, et al [73], where they proposed a training

method “which can reconstruct high-resolution images of different upscaling factors in a single model”.

Generative adversarial network

A generative adversarial network (GAN) is a class of machine learning, where two neural networks compete against each other, where one agent’s gain is the other’s loss. The way this works in our project is that we have one network generating cleaned images, while another network tries to predict whether the input is generated or real. This technique was designed by Ian Goodfellow [20] and colleagues in 2014, and is widely used in image related machine learning.

Feature Loss

Feature loss is a loss function first proposed in the paper “Perceptual Losses for Real-Time Style Transfer and Super-Resolution” by Johnson, et al (2016) [9]. The idea was that instead of having a loss function that focuses on per-pixel difference, a loss function that focuses on higher-level differences could potentially be better when generating images. This means that when generating an image of a cat, instead of having the model correct itself based on pixel differences, it should rather focus on getting the high-level features such as ears and eyes right.

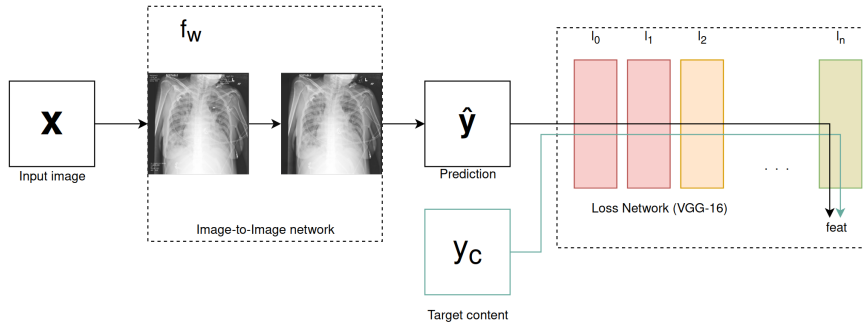


Figure 6.1: A simplified illustration of the feature loss system in [9]. The input image is transformed into output, and then a loss network pre-trained for image classification (here VGG-16) defines the features, and measures the differences in content. The original feature loss function included target style as well, however since we do not perform style-transfer, this is omitted. The figure was inspired by Fig.2 in [9].

This is done by taking a pre-trained network, and using it as the “loss network” essentially defining what a feature is. Then, as previously mentioned instead of looking at per-pixel loss, the average pixel difference over an area(feature) is calculated, so that the model’s loss is equal to the difference in features, and not per-pixel. This is useful in image generation, where the features matter, rather than the image as a whole.

6.3 Experimental results

6.3.1 Proof of concept

To start, we did a proof of concept experiment, where we took the Chest X-ray dataset from Kaggle and used the previously mentioned “crappify” solution, that inserted text to the X-rays. That way we have generated training data to feed the GAN and feature loss model, where the goal is to generate the “crappified” images without text. The images are then resized using the squish method (as they are larger than the size they are being resized to), normalized, and put into a databunch. Now that the data has been prepared, we can move on to the models.

6.3.2 GAN

Creating a GAN primarily requires three steps: a learner, a critic, and a “switch” that combines the learner and critic, creating the generative adversarial network. The process starts with creating a learner. This model learns how to generate images without text. Then, after training, we define a critic. The critic’s goal is to identify whether an image is a real image or one that the learner predicted. Then, after the critic has been trained. We define a switch, that swaps between the learner and the critic when the loss goes below each respective model’s set thresholds. The learner will try to “outsmart” the critic, updating itself after receiving a verdict from the critic. Here is the experiment configuration, highlighting key aspects of the model’s parameters:

<p>Learner Model: Unet Archetype: ResNet34 Loss function: Flattened MSELoss Training: 2 epochs on final layers, 3 epochs unfreezed</p> <p>Critic Model: Fastai’s GAN critic Loss function: AdaptiveLoss using BCEWithLogitsLoss Training: 6 epochs</p> <p>GAN Switcher: AdaptiveGanSwitcher Additional techniques: progressive resizing 256-512-1024 Training: fitted 40 epochs</p>
--

Table 6.1: Experimental settings for our GAN model



Figure 6.2: The original image, the prediction, and the actual image. The GAN has generated an image that is close to the real image visually, however, this may not be the case when looking at the subtraction or histogram normalization.

Looking at the results it seems good. Most of the features are kept intact, and no text is visible. It does seem like the model removed a bit of the letter "R" in the top left corner, so to get a better picture, we represent the difference by subtracting the prediction from the original image, as well as a histogram normalization of the image. Looking at the subtraction, if the removed text is easily readable, we can say that the model has completely removed the text.

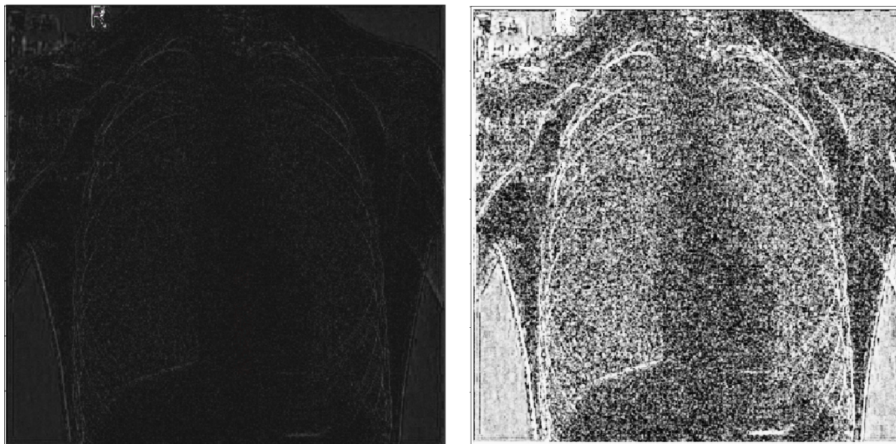


Figure 6.3: Subtraction of GAN prediction and the histogram normalization of the subtraction. This is done to highlight differences due to the dark pixel-values in the image, making it a lot easier to detect differences visually. Looking at the normalization, it is easier to see that the model changed the details of the image as well, rather than just the outline.

We see here that the GAN missed some pixels of text, as well as removing some of the outlines in the image. While the result looked good, this image shows that the results were not as good as initially thought, considering that the existing solution completely blocks out text while keeping the rest of the image intact. The text is also not readable, meaning that the text is still present in the prediction, albeit not readable. We will now perform the same experiment using a regular Unet with feature loss as the loss function.

6.3.3 Feature loss

The first steps are identical to the GAN experiment, where we “crappify” the chest X-Rays, transform and normalize them to the same size, and put them into a databunch. To create the feature loss function, we imported a pre-trained VGG16 [4] network, that is trained on the ImageNet challenge dataset [34] containing more than 1.4 million labeled high-resolution images to train our loss network. Then we define our experiment configuration as follows:

Model: Unet Archetype: ResNet34 Loss function: feature loss Additional techniques: progressive resizing 256-512-1024 training: 10 epochs on final layers, 10 epochs unfreezed

Table 6.2: Experimental settings for our model using feature loss

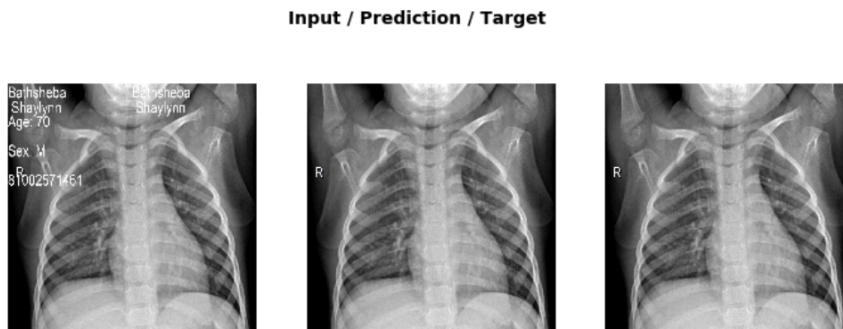


Figure 6.4: The original image, prediction, and actual image using Unet with feature loss instead of GAN. The results are similar to the ones in Fig.6.2, with a bit more detail in the left armpit. Looking at the difference and histogram normalization will tell us more about the results.

While not much difference visually, we can still see that the feature loss model performed better when it comes to preserving the letter ‘R’ in the image, and this could indicate that it also performs better by not removing pixels where it shouldn’t. To check that we again create a subtraction of the original image (Fig. 6.5).

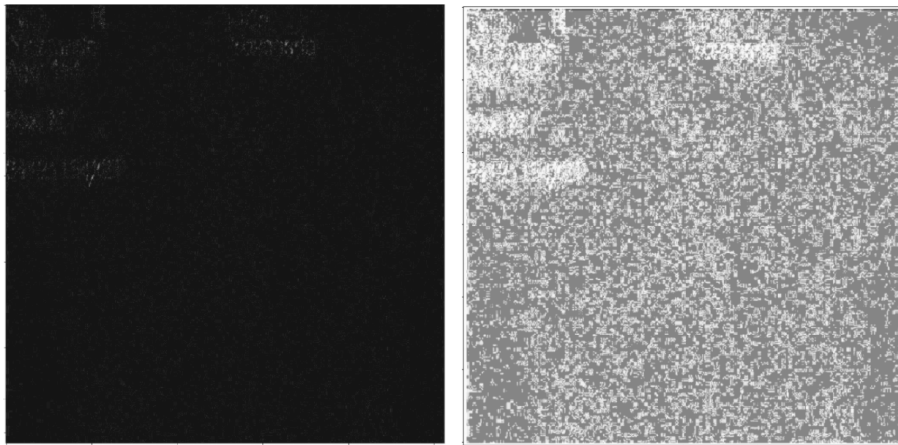


Figure 6.5: Subtraction of prediction with feature loss as the loss function, as well as the histogram normalization. We see now that the model has changed pixel-values everywhere, something that would be hard to tell by only looking at the subtraction

As this image tells us, our initial thoughts were correct. While not a huge upgrade in regards to the text removal, we see that the rest of the image is intact, with no changes to the actual detail, only small changes to the pixel-values

Lastly, to see if this solution is viable, both models are trained on real images from the render-text program. The same steps were applied, with the only difference being the data used. However, the results were not that great, as shown in Fig. 6.6



Figure 6.6: Results of using full images as data. The left image is the input, the middle is the prediction and the left is the actual image the model is trying to generate. Overall the GAN has struggled to create an image good enough for the critic to accept, and thus most likely struggled to find out what it needed to do. The Unet with feature loss produced similar results.

6.4 Discussion

As this was meant as a feasibility study, we see this experiment as successful. The experiment proved that it is possible to generate cleaned images using image-to-image models. Both models managed to produce good results on the

“crappified” data, but struggled with the real-life data. This is most likely due to the complexity of the data, as the images have text with different properties such as color, size, and positioning as shown in Fig. 6.7

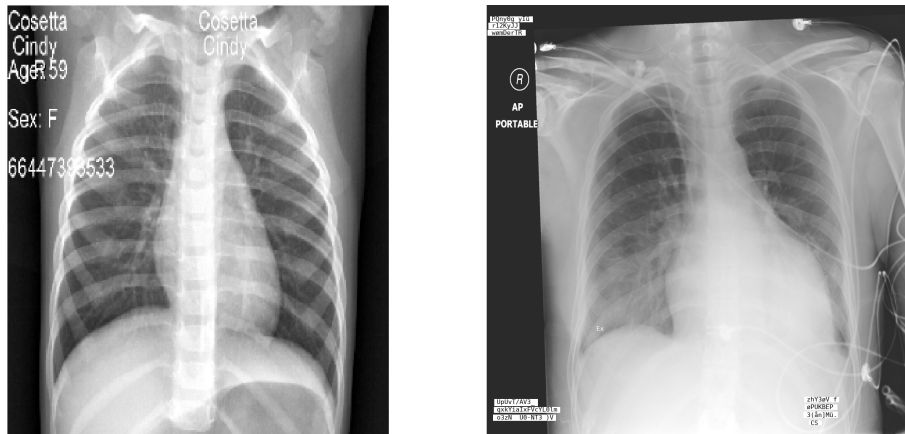


Figure 6.7: Difference in training data. Our own “crappified” images had larger text with a fixed position and color. The render-text program generated more complex data with differentiating position, text-color and background depending on where it was placed.

In the feature loss model’s case, this could be due to the loss network not properly recognizing text as a feature, thus not penalizing the incorrect removal of it. Knowing that good results were possible, we moved on to using generated data from the render-text program.

Chapter 7

Generating new images without sensitive information: an improved approach

7.1 Introduction

In the first experiment, we explored the feasibility of generating medical images from the same images with text. Our initial results were good, however, due to the simplicity of our ‘crappified’ images, the model performed poorly on real-life data. We also noted that the images removed irrelevant pixels, also in parts of the image with no text (Figs. 6.3 and 6.5), which could potentially create problems in the future.

7.2 Methods and materials

Given that this is a continuation and improvement of the experiments in the previous chapter, we will continue using the same libraries and methods, but an improved way of generating training data and a better way to treat irrelevant parts of the images.

Dataset

With the goal of this experiment being a viable solution to de-identifying medical images, we need to use data that represent real-life data used at MMIV. By using the render-text program mentioned in the introduction of the experiment part, we now have images that closely resemble real-life data. Our dataset contains 10,000 DICOM images converted to PNG format with the size of 1024x1024 pixels.

Pillow

Our improved solution will perform image manipulation to prepare the training dataset, and the Python library Pillow offers a wide array of manipulation

functions such as pad, crop, and resize. Pillow is a successor project that forked the python imaging library (PIL), an open source library that adds support for opening and manipulating images given that the file-type is supported[74]. Created in 2011 after the discontinuation of PIL, Pillow is, by many considered a replacement for PIL with the support of Python 3.x versions.

7.3 Extracting image parts containing text

In order to work around the issue with complex training data, we decided to take a different approach by only passing the parts of the image where text is present, rather than using the whole image. Using this technique, we will also minimize irrelevant pixels being removed., as well as a dataset much larger than the original 10,000. This results in a different data preparation step, where we create a function where we pass the full images with text, without text, and a DataFrame containing info about the bounding boxes. That way we can extract images of where the text is, and use that as training data. Fig. 7.1 illustrates the extraction.

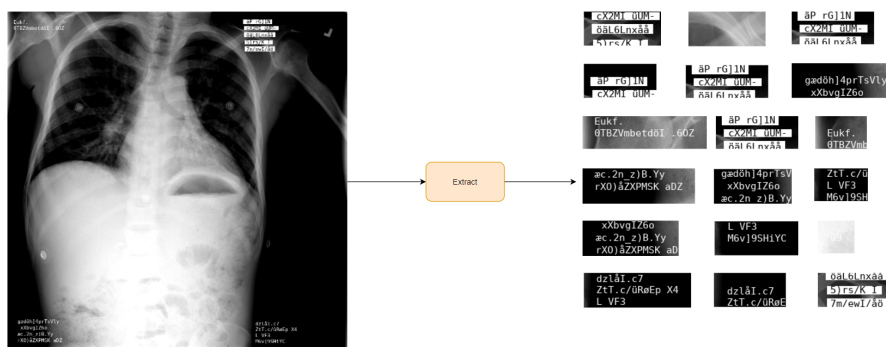


Figure 7.1: The extract function is passed a whole image and gets the coordinates of the bounding boxes from the DataFrame. Then, using PIL, the boxes are cropped out, resulting in a new dataset of bounding boxes, rather than whole images.

You might notice that the cropped images often contain more than one text box. This is due to the fact that we included 20 pixels on each side so that the feature loss function receives more context, rather than just the text-box, and nothing around it. After the image extraction, we pass them to a function that consistently resizes and then pads the image to 128x128. The padding method used by fastai proved hard to reverse, so by using our own padding and resizing functions, we can easily reverse this when the images are being stitched together. With the extracted and transformed images, we prepare the images by creating a databunch, this time omitting the resizing with padding. With the data prepared, we can start training. This is our experiment configuration:

After training, a separate set of images generated by the pixel render program were used for testing, being pre-processed the same way as the training data.

Model: Unet Archetype: ResNet34 Loss function: feature loss(VGG_16bn) training: 10 epochs on final layers, 10 epochs unfreezed

Figure 7.2: Experimental settings for our RetinaNet model tasked with text detection.

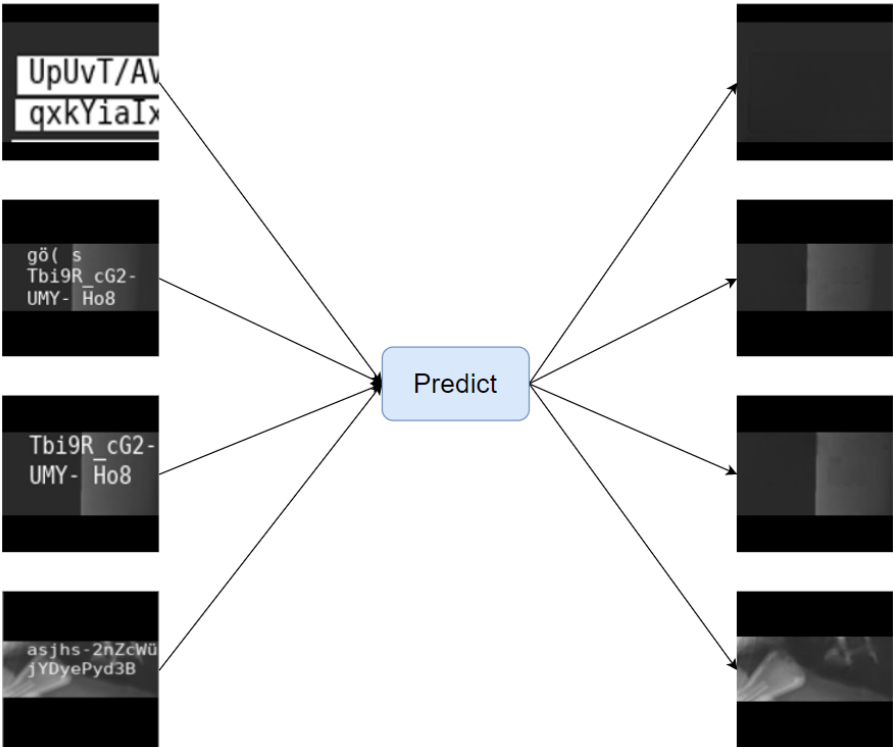


Figure 7.3: Predictions of bounding-boxes without text. The model is passed the cropped bounding-boxes, and predicts the images without the text. The predictions gives us an indication that the model has performed well.

The images in Fig. 7.3 look promising. We see that the text is gone, and the model has managed to generate pixels that make sense in the surrounding context. This is especially apparent in the areas where the text covers detailed parts of the X-ray, such as in the bottom images. Using an identical but reverse version of the extract function, the predictions are placed onto the original image using the same coordinates from the bounding box DataFrame. After the images have been stitched back together, we end up with this result, shown in Fig. 7.4

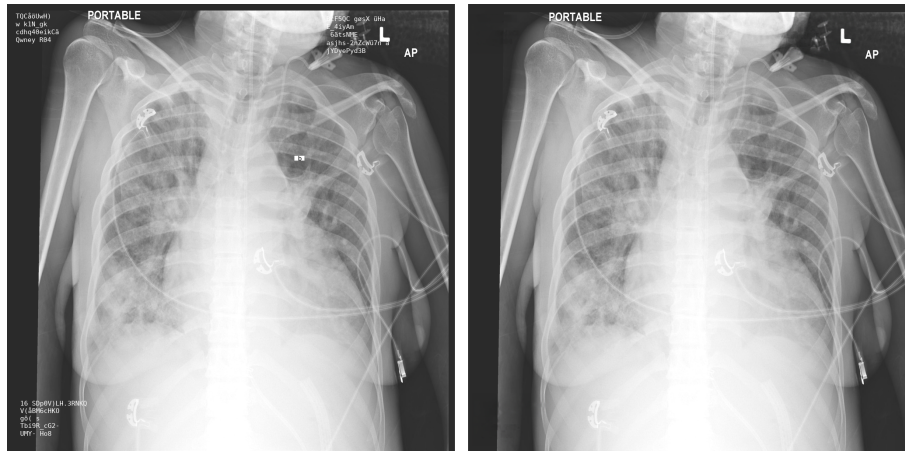


Figure 7.4: On the left we have the original image, and the right image is essentially the original image, but with the predictions placed on top. With the images stitched back together, they can now be saved to create subtractions and histogram normalizations.

Visually the results look great, with the text being completely gone, and the filled-in pixels make sense. This is more noticeable in the areas where the text covers part of the x-ray such as on the chin, bottom left corner, and the small text on the middle-right side. To be sure of our results, we created the subtraction image, as well as performing histogram normalization on the image to highlight the darker areas, shown in Fig. 7.5.

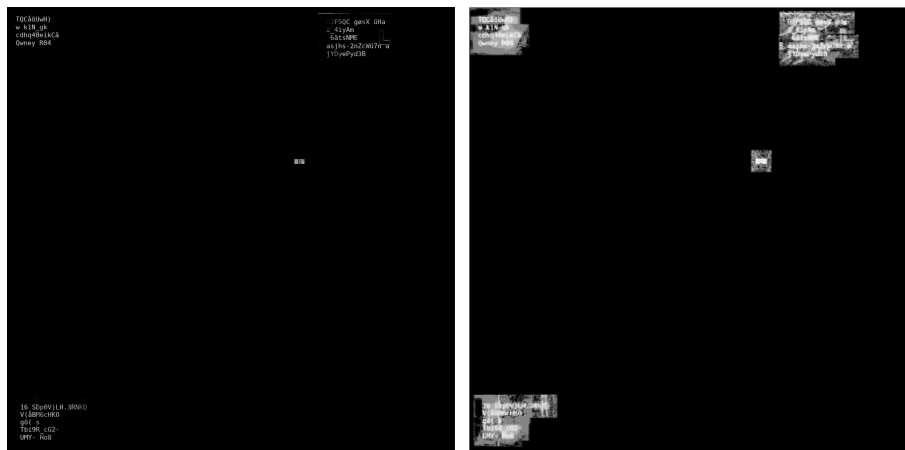


Figure 7.5: Subtraction image and histogram normalization of the image. The subtraction contains clear text, and the normalization shows that some changes has been made to surrounding pixels.

Looking at the difference image, the level of removal is greatly improved. As previously mentioned, clearer text means a greater level of removal, and from what Fig. 7.5 shows, the text is almost completely removed. The normalization shows that there are some changes made to the surrounding pixels, but this was

within expectation, and comparing it to the end-product in Fig. 7.4, tells us that the predictions fit nicely into the original image. It is also useful to compare the subtraction image to the original image, illustrated in Fig. 7.6 where we see the level of text removal.

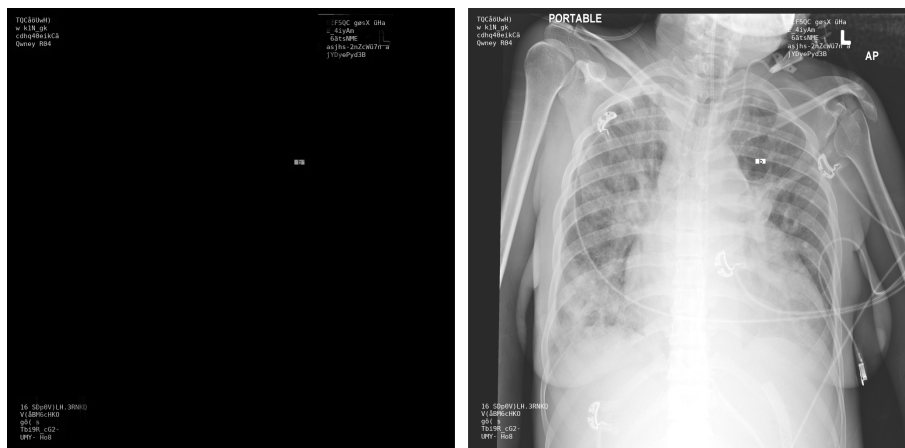


Figure 7.6: Subtraction image and original image. The text in the left image looks almost identical to the ones in the original, meaning that there are almost no residual pixels in our prediction.

7.4 Discussion

With the change in training data, the model produced great results. The experiment configuration used was initially intended as a test to see if cropping the images would improve performance, however, due to the greatly increased performance and good results, we decided on sticking with it to stay on schedule. Progressive resizing was also omitted due to the importance of detail in our data, as well as the consistent sizing of the images. The results shown were good, however, in a few cases, the model made small errors(Fig. 7.7).

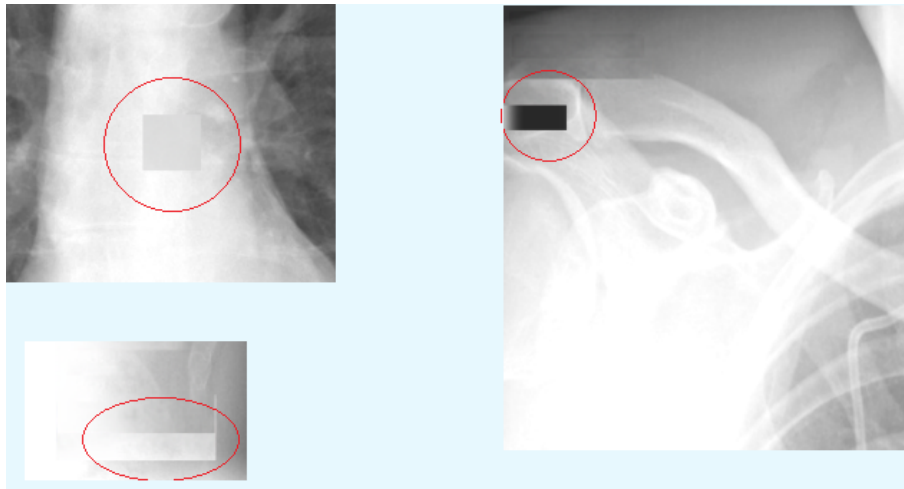


Figure 7.7: Errors highlighted with a red circle. Light-blue background due to colors in the image itself. In some cases the models error is related to color, and others it fails to generate details.

We believe the main reason for the errors lies in the dataset the model trained on. Most of the images the model was trained on had dark borders around the image, so when the model generated the right image in Fig. 7.7, it probably expected the area to be dark. Evening out the amount of different "types" of images with and without dark borders, as well as the average color distribution in the image could potentially fix this issue, although simply increasing the amount of training data seems like a good step in the right direction. Another issue we faced was that in some cases, the bounding boxes were quite large, meaning we would have to resize them down to 128x128 and then resize the prediction back to the original size, leading to decreased image quality. The text generated by the render-text program is usually shorter than 128 pixels so at the time resizing to 128x128 was the ideal solution, however for future training, increasing the resolution could fix that issue.

In conclusion, by looking at the results visually, as well as producing versions of the predictions such as subtraction and histogram normalization, we believe that the experiment was successful. The model does as previously mentioned in a few cases predict poorly, however, we have taken this into account, and will explain our planned solutions in the future work section.

Chapter 8

De-identification application

8.1 Introduction

In this chapter we will combine the two models from our previous experiment, to create an application that will also serve as a prototype for a future pipeline. Our goal is to be able to upload a medical image, detect the bounding boxes, removing the text within the boxes, and then finally produce a de-identified image.

8.2 Methods and materials

Ipywidgets

Ipywidgets is a python library for interactive HTML widgets that can be used in Jupyter notebooks [75]. This sort of turns a notebook into a web page by introducing buttons and other interactive widgets that can be used without manipulating code.

Voilà

Voilà is an application and Jupyter extension that converts a Jupyter notebook into an interactive dashboard that allows users to share their work with others, while giving the user control of what readers experience [76].

8.3 Results

Before creating the application, a function that prepares the data from the object detector is made to comply with the data requirements from the text remover. This is done by generating new coordinates based on what quadrant the text is in, and creating a DataFrame containing the new coordinates.

We then create two notebooks, one for the text-detection and one for the removal. The goal of these notebooks is to be as simple as possible, only importing necessary libraries, and defining important functions relating to the task. In the

case of the text-detector, we import `fastai`, as well as `PIL` for image manipulation, and `ipywidgets` for the buttons. We then simply define two buttons, one for upload, and one for prediction. The user then uploads an image of their choice. The image is then split into 16 parts, and fed one by one to the model. After the prediction is done, we convert the bounding box coordinates so that the text removal model accepts them. With the conversion done, we download the `Dataframe` containing the coordinates, ready for use by the text-remover.

The text-removal application works similarly to the text-detector where we define upload and predict buttons, where the user uploads their image, and press the predict button. The relevant areas of the image are then cropped using the coordinates from the text-detector, and fed one-by-one into the model. After prediction, each crop is placed back on top of the original image, creating a de-identified image.

With the applications done, we use `Voilà` to turn the notebooks into an interactive dashboard, meaning that only the buttons remain, essentially acting as the front-end to our application (Fig. 8.1). Finally, the applications are pushed to `GitHub`, where they can be accessed using `Binder`, a web service where notebooks can be opened in an executable environment [77], hosting the notebook for a short amount of time. As we are aiming for a quick prototype form demo purposes, `Binder` serves us well. However, if the application were to be improved, a more permanent solution for hosting and serving the application, based on e.g. `Google Cloud` or `Heroku` [78], would be recommended.

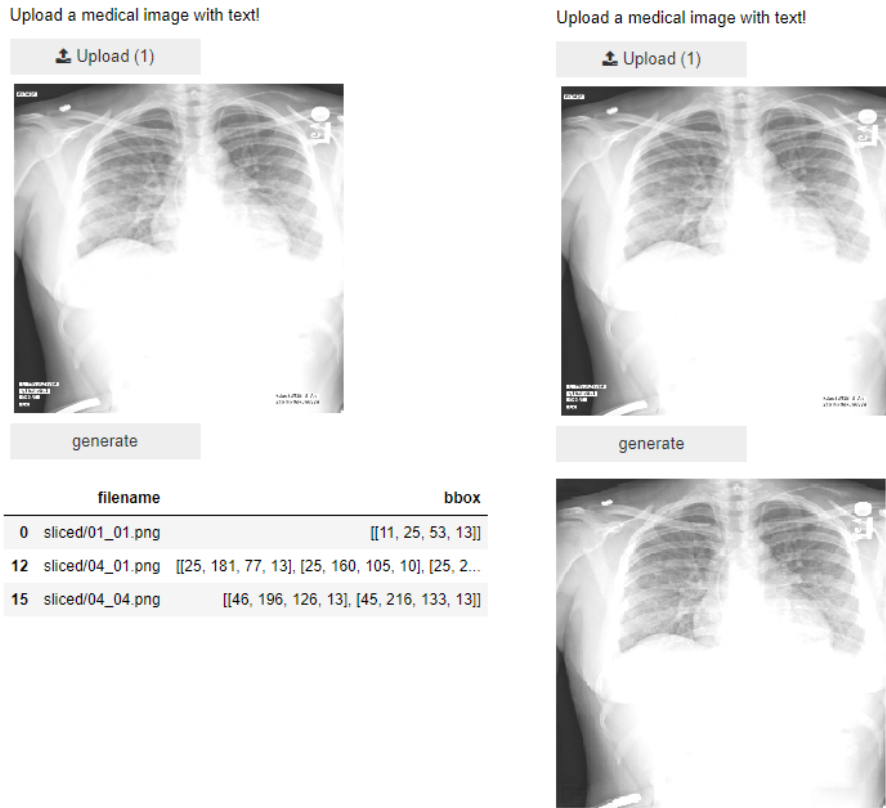


Figure 8.1: De-identification applications. The application illustrated on the left detects the text and returns a DataFrame containing the coordinates. The one on the right uses the coordinates to remove text present in the area.

8.4 Discussion

The reason this application was made was to show a prototype of a potential pipeline. Except for the widgets, the two applications are essentially run-files, containing all necessary functions to perform de-identification. There were some versioning issues in our fastai models, leading us to create two separate applications rather than combine text detection and removal in a single application. A way around the issue would be to rebuild the text-detector using Python version 3.7.10, which was deemed unnecessarily time-consuming for the present purposes. A possible next step for our application would be a system linked to a relevant data storage, see Fig. 8.2. This and other possible future extensions are discussed further in Section III.

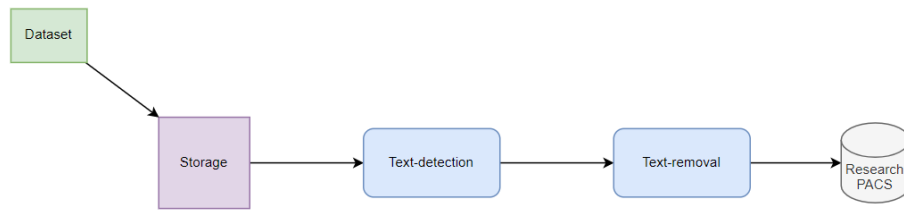


Figure 8.2: An overview of a simple de-identification pipeline. The user uploads a dataset of medical images to a data storage, triggering the de-identification process. The final images are then stored on the research PACS.

Part III

Discussion and further work

Chapter 9

Evaluation and conclusion

In this chapter, we will discuss the results from the three experiments above, and present areas of improvement for future work. Our experiments resulted in two deep learning models, one for text-detection, and one for text-removal. We also made a simple application, tying the two models together as a prototype of how a potential pipeline could function. To recap, we created Fig. 9.1, illustrating a full overview of our work.

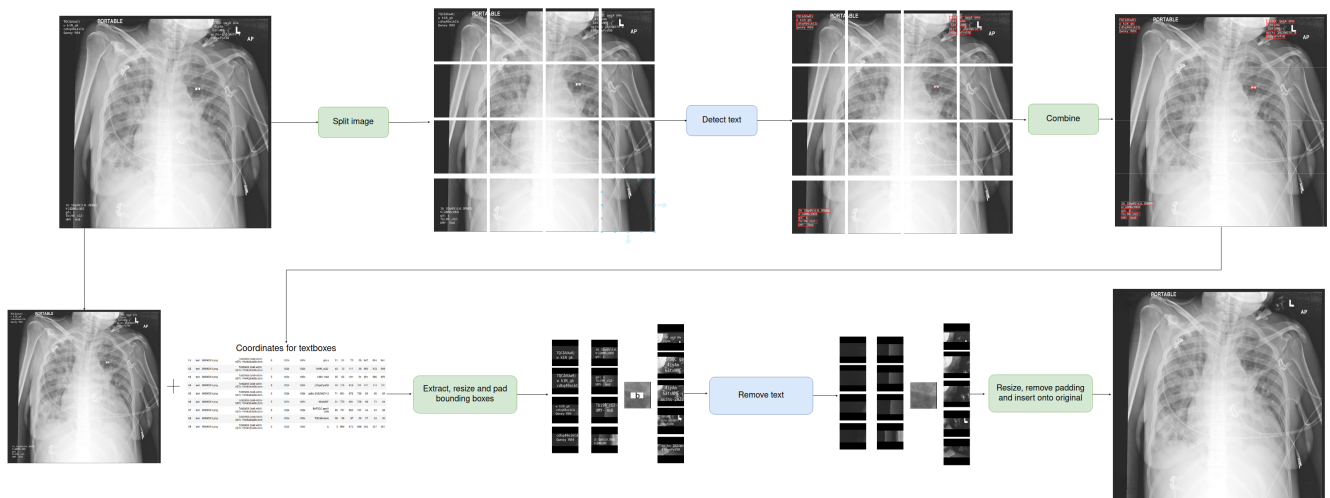


Figure 9.1: Complete system overview. A medical image is split into 16 parts before the text-detection model detects the bounding boxes. The image is then combined again in case the image is needed for other research purposes. Then, by using the coordinates of the bounding boxes, we can crop the original image to extract the relevant areas. Those images are then passed to the text-removal model. After the text has been removed, we place the predictions from this model on top of the original image using the original coordinates, resulting in a de-identified image.

9.1 Discussion of research questions

Early on in the process, we defined research questions to guide us through our work. To properly evaluate our experiments, let us take a look at each one:

1. *"How can we use machine learning to automate 2D medical image de-identification while preserving image features?"*
 - *"How do we make sure these images are properly de-identified?"*

To verify that the images are properly de-identified we created the image subtractions and found that the text was completely removed (Fig. 7.6). Since the text was very readable in the difference image, it was an indication that the model successfully removed the text, but to be sure, we showed the results to an expert on medical imaging. After running it through a PNG-enhancer program meant to highlight pixel-values, we noticed that there are changes made to the image beyond just removing the text. However, it seems impossible to extract any information from it, except for maybe the length of the bounding box (Fig. 9.2)



Figure 9.2: Prediction on the left and original image on the right. We notice the areas where the model has done changes, however it is not possible to read characters.

An idea we recommend pursuing in any future work is to attempt to recreate the text from the de-identified images. By training a "reverse" model that predicts images with text given the de-identified image as input. We did make a small attempt at this when working with the proof-of-concept experiment for the text remover, but the immediate results were not useful in any way and the idea was quickly abandoned. We do still believe that with enough time and resources, implementing such a model could be a valuable step to verify de-identification in a potential pipeline.

- *"Can we design an object detection model to successfully detect burnt-in text in the images?"*

When we worked with the RetinaNet model, we spent a fair bit of time with the generation and preparation of the data as there were no available, existing dataset that would fit our task. We ran a proof-of-concept experiment with the synthetic data we generated using the "crappification" method. The results turned out to be varying (see Fig. 5.5, but it did show us that it could be possible to detect text using RetinaNet with Focal Loss. For the final version of our RetinaNet, we saw the need for better data. This problem was handled with the text render solution developed by Hauke Bartsch [63]. We decided

to divide the images into tiles with dimensions 256×256 pixels to hopefully increase the detection rate. We also tuned our anchor boxes to better mimic the shape of a text line. The combination of the data augmentation and new anchor boxes turned out to work really well, as shown in Fig. 5.9. There are still some instances of text which is not detected, but these are often single letters. As previously mentioned, the existing solution in use in the research PACS system of MMIV/Helse Vest ignores single characters. The reason is that there is next to no chance of any personal information being represented in a single character or number. We were also able to stitch the tiles back together so the output of this pipeline is an image with the original resolution, and a CSV file containing the filename and detected bounding boxes. If we compare the result from the existing solution and our proposed RetinaNet in Fig. 9.3, we can see that we have successfully utilized an object detection model to detect text in images.

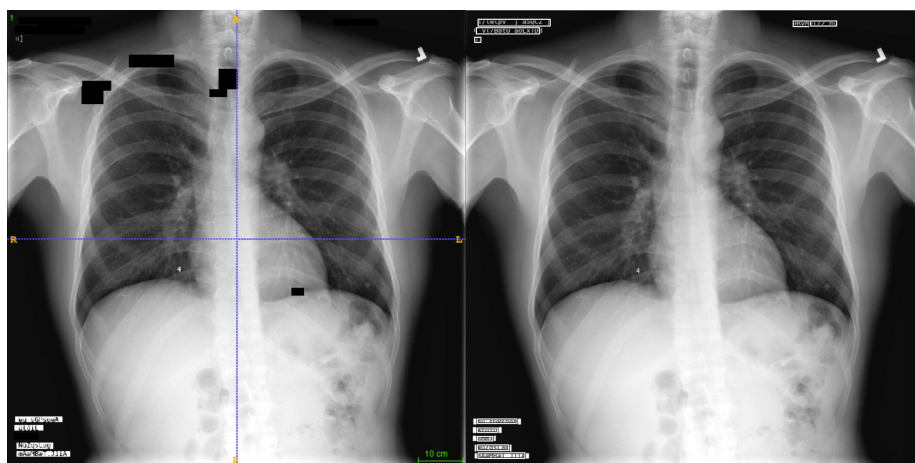


Figure 9.3: A comparison of the "rewritepixel" solution used at MMIV on the left and our text detection model on the right. By looking at the two pictures we can observe that our model has a higher detection rate in this example.

1. *"Can we remove the sensitive information using image-to-image deep-learning, while still preserving the image integrity?"*

When working on the proof-of-concept experiment, we quickly noticed that the image integrity was compromised after looking at the histogram normalization in Fig. 9.4

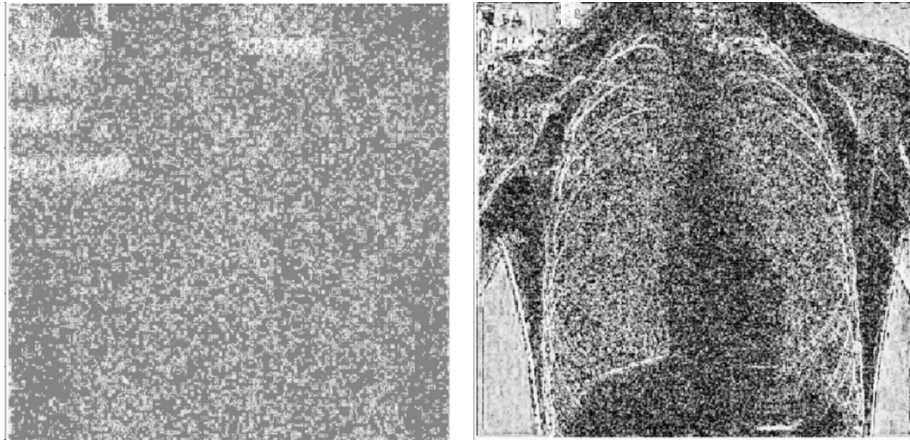


Figure 9.4: Histogram normalization of the “crappified” images in our first, proof-of-concept text removal experiment. We clearly see that the model has changed pixels in irrelevant areas, potentially compromising the integrity of the features in the image.

This problem as well as the issue where our model struggled to remove the text was fixed when moving over to the improved solution where we only applied the removal models in the areas where text was detected. Thereby limiting the areas in which the model could potentially harm the features of the image. After presenting and discussing our solution with a local medical imaging expert (Hauke Bartsch), we conclude that while our solution is likely not a suitable replacement for black-boxing in its present form, it could still be useful for other purposes, e.g. teaching. Its main drawback is that whatever the model has replaced the text with is still not real, but computer generated, and could in the worst case be misinterpreted as something else such as a tumor. And while more pleasing and less distracting to look at than black-boxed images, it would still pose an ethical dilemma to include computer generated pixels in an otherwise real image in diagnostics. One may speculate that a clear indication that a specific region of the image has been changed could somewhat alleviate the issue, as the person reading the image can be instructed to not trust whatever is inside the highlighted regions.

9.1.1 Challenges and known limitations

When working with machine learning in general, often you come up against certain limitations that either blocks progress, or needs to be worked around. Some limitations we faced in our work were due to limited computing power, both locally and in the cloud. When doing computer vision with advanced neural network architectures, sometimes you end up running out of GPU memory, an issue we faced repeatedly throughout our work (Fig. 9.5)

```
RuntimeError: CUDA out of memory. Tried to allocate 32.00 GiB (GPU 0; 14.76 GiB total capacity; 3.21 GiB already allocated; 10.11 GiB free; 3.87 GiB reserved in total by PyTorch)
```

Figure 9.5: CUDA out of memory error. Appeared when trying to use 1024x1024 images as training data in the GAN

Using cloud-based solutions outside of one's control can also lead to various challenges, e.g. Fig. 9.6.

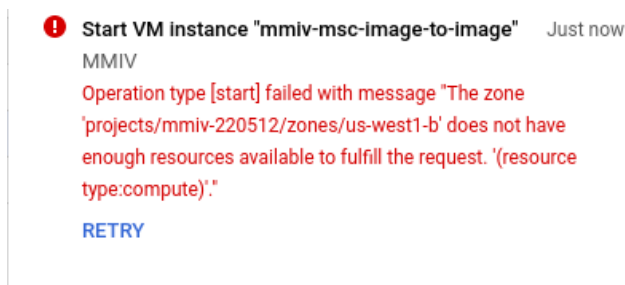


Figure 9.6: Lack of computational resources allocated to the MMIV organization by Google repeatedly led us to not being able to run our virtual machines on the cloud. Due to the work being performed during the COVID-19 pandemic, we had to work on machines hosted in the cloud to be able to work consistently, and at random times this error would force us to wait for resources to be available.

Text detection

A major limitation with this part of the pipeline is the limited input sizes used by our model. As of now, the RetinaNet model will only accept images where the dimensions are divisible by 256, as this is assumed by some hard-coded components of the model. We attempted to design a more flexible model, but quickly realized that it was easier to slice the image rather than changing an already working and relatively complex model. We also tried with larger resolution on the images, both 1024 and 512, but the results were not as good as the final solution. This was mainly due to our failure to find good ratios for the anchor boxes. Another possible limitation, or at least an unnecessary component, is the use of a Focal Loss. There is no real classification task to be done other than foreground-background, which means this loss function, designed to make good use of object classification during object detection, might be a bit overkill.

Text removal

The text-removal model also has a limitation in regards to resolution. As mentioned in the experiment, using 128x128 as the standard size for cropped images, we could potentially lose some image quality when the bounding box is larger than this size, as the images are then both down- and upsampled. However, in future work this can be fixed by increasing the standard size.

On a more fundamental level, there is the problem of explainability. This is a known limitation with using deep learning models. Many deep learning algorithms are considered to be a "black box": you feed the system with data, and a result is produced, with no real way of knowing how the model came to that conclusion. There are many attempts at alleviating this issue, with an entire field called "Explainable AI" [79, 80, 81] dedicated to coming up with new tools and methods. However, this is at the forefront of research in machine learning, and we are not aware of any powerful and flexible techniques that can easily

be used in our case. The need for explainability is especially important with a system like this which handles personal health information. This is a weakness concerning both of the models in this thesis.

9.2 Conclusion and future work

The potential pipeline has the ability to be extended to work with several different medical image formats. The current models are only trained on X-rays of the chest. It is possible to create different datasets with different image modalities, i. e CT, MRI, ultrasound, relatively easy using [63]. We could then create a classifier that predicts what type of modality the image has and then run the correct text detection and text removal model.

One interesting feature which would be possible to implement is a text classifier. Our model already detects text, so implementing text extraction should be possible. One could then train a classifier to decide if the text in question is sensitive or not and then only de-identify the text classified as sensitive. This was an idea we had early on, and actually led us to our choice of using Focal Loss. By implementing such a solution we could make full use of the potential of the Focal Loss function by classifying, as well as detecting text.

In regards to the text removal model, one could try a different approach. When we were planning on how to go about the task of removing text, we decided on using image-to-image models and techniques we were familiar with. This resulted in us focusing fully on using GAN and Unet with feature loss to perform the text-removal. As previously mentioned, the results were good, but not a replacement to black-boxing yet, due to the residual pixels. The first step to fix this would probably be to use more data for training. One could also experiment with transformations so that the residual pixels are not relevant to the identity of the person. One could even go about training on black-boxed images rather than original images. We did experiment with this, however, due to time constraints it was trained with fewer images for a shorter period of time. Some very preliminary results can be found in Fig. 9.7.



Figure 9.7: Original image, prediction and actual image. We quickly made a function that black-boxes the text (text box in the middle is the one to be removed, rest is context). The model is worse at generating detail compared to the one that did not use black-boxes, but it could be a viable, more privacy-preserving alternative.

There are of course many other ways to go about text-removal instead of the methods we focused on, and many choices within the framework we set up. Given enough time and computing power, one could perform an ablation study to identify the components of the system that were most impactful for the performance and spend resources optimizing these. One could also use other image-to-image models such as cycleGAN [42], Conditional GAN (cGAN) [43] or replace Unet with another architecture.

Another area of improvement is using the newest version of the fastai library, v.2. Both the models are using the first version of fastai. This was decided early on in the project because when we started working on this thesis the new version of fastai had just been released. Its newness led to a lack of support for the models we wanted to use. There is still not much support for object detection in fastai 2, but according to [82] it is planned to be supported in the future. At some point, it would be recommended to update the code to use the newer version of fastai as it is a complete rewrite of fastai v.1 with a significantly improved software design.

Lastly regarding future work, one would like to implement a comprehensive pipeline. By creating a pipeline, one could automate certain processes such as training and prediction. This would make the process more future-proof, considering you could train on new data if the model's performance starts to decline. We gave an example of how a potential prototype could look like in

Fig.8.2. However, an improved complete pipeline could perhaps look something like the one in Fig.9.8.

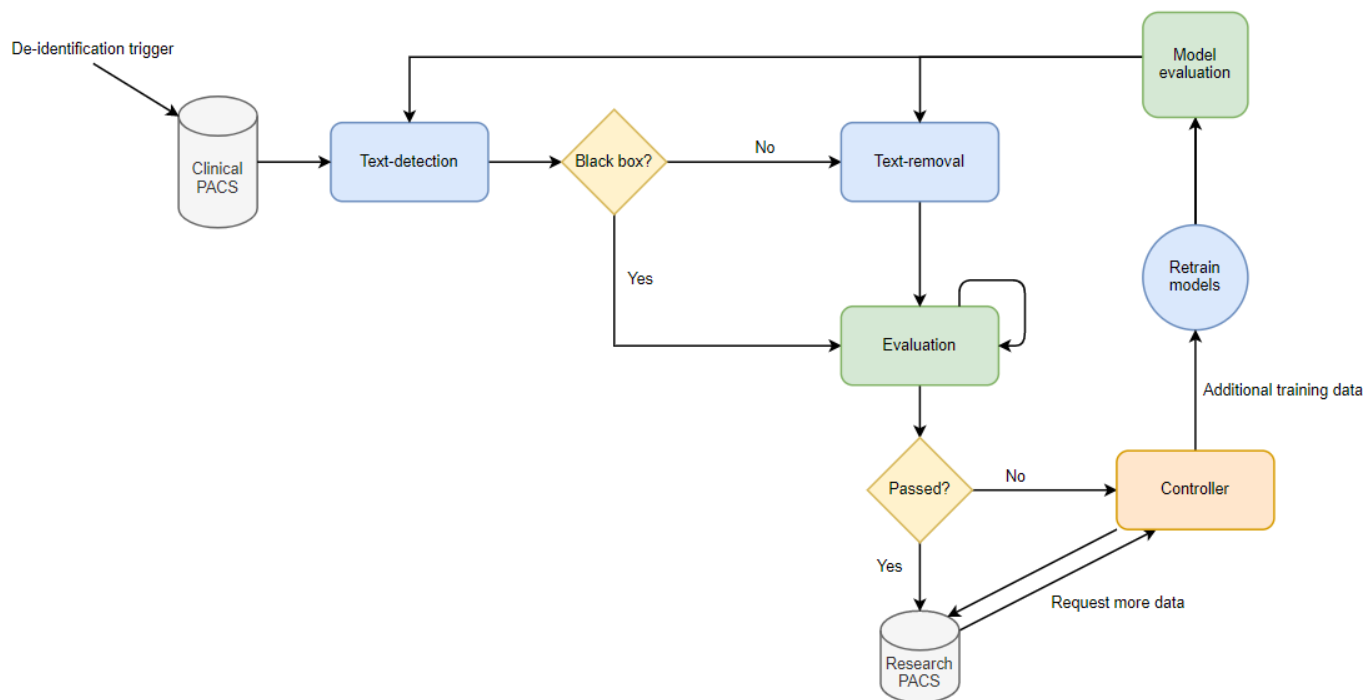


Figure 9.8: An overview of a proposed, more comprehensive pipeline for de-identification. The de-identification trigger sends a request for images to the clinical PACS. The PACS then passes the images to the text detection model. There we can decide whether we want black-boxed images, or use the text-removal model to de-identify them. After de-identification, the images are evaluated. This could be done by running the PNG-enhancer program, and if the image passes set thresholds, we can move on to a deep learning solution, trying to recreate the text. If the image passes evaluation, it is sent to the research PACS for storage, and if not we send a message to a controller. This controller would then request images from the research PACS to train the models again using new data. To verify if the model's performance improves after training, we added a model evaluation step, where the model would use a fixed validation set and see if it improves. One could also daily add the de-identified images to the new training set, so that the models follow the data, and keeps it's good performance.

With the increasing importance of GDPR, developing good tools for de-identification is helpful to protect the patients' privacy, as well as providing researchers with more data. Further developing our de-identification solution into a complete application or pipeline could therefore provide researchers and healthcare workers with a useful tool that could lessen the amount of manual labor needed when dealing with de-identification, leading to more data being available. This could in turn help them achieve better results, as having more data is likely to have a positive effect on their work.

References

- [1] De-identify medical images with the help of Amazon Comprehend Medical and Amazon Rekognition. <https://aws.amazon.com/blogs/machine-learning/de-identify-medical-images-with-the-help-of-amazon-comprehend-medical-and-amazon-rekognition/>, . Accessed: 2020-09-08.
- [2] Classification, redaction, and de-identification. <https://cloud.google.com/dlp/docs/classification-redaction>, . Accessed: 2021-05-24.
- [3] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8110–8119, 2020.
- [4] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- [5] Yujin Chen, Ruizhi Chen, Mengyun Liu, Aoran Xiao, Dewen Wu, and Shuheng Zhao. Indoor visual positioning aided by cnn-based image retrieval: training-free, 3d modeling-free. *Sensors*, 18(8):2692, 2018.
- [6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [7] Yi Ding, Fujuan Chen, Yang Zhao, Zhixing Wu, Chao Zhang, and Dongyuan Wu. A stacked multi-connection simple reducing net for brain tumor segmentation. *IEEE Access*, 7:104011–104024, 2019.
- [8] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.
- [9] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European conference on computer vision*, pages 694–711. Springer, 2016.
- [10] Amazon Go. <https://www.amazon.com/b?ie=UTF8&node=16008589011>, . Accessed: 2020-09-08.
- [11] Asch Chen. Machine Learning and Prediction in Medicine — Beyond the Peak of Inflated Expectations. *N Engl J Med* 2017; 376:2507-2509, 2017. doi: <https://dx.doi.org/10.1056/NEJMp1702071>.

- [12] Marcus A Banks. Sizing up big data. *Nature medicine*, 26(1):5–6, 2020.
- [13] Jeffrey R Wagner, Christopher T Lee, Jacob D Durrant, Robert D Malmstrom, Victoria A Feher, and Rommie E Amaro. Emerging computational methods for the rational discovery of allosteric drugs. *Chemical reviews*, 116(11):6370–6390, 2016.
- [14] Eric S Williams, Kent V Rondeau, Qian Xiao, and Louis H Francescutti. Heavy physician workloads: impact on physician attitudes and outcomes. *Health Services Management Research*, 20(4):261–269, 2007.
- [15] Ishna Neamatullah, Margaret M Douglass, H Lehman Li-wei, Andrew Reisner, Mauricio Villarroel, William J Long, Peter Szolovits, George B Moody, Roger G Mark, and Gari D Clifford. Automated de-identification of free-text medical records. *BMC medical informatics and decision making*, 8(1):1–17, 2008.
- [16] Paul Voigt and Axel Von dem Bussche. The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, 10:3152676, 2017.
- [17] wikipedia. Unsupervised learning. https://en.wikipedia.org/wiki/Unsupervised_learning, . Accessed: 2021-02-07.
- [18] Yu-Chi Ho and David L Pepyne. Simple explanation of the no-free-lunch theorem and its implications. *Journal of optimization theory and applications*, 115(3):549–570, 2002.
- [19] Figure Eight. Data scientist report 2018. <https://www.datasciencetech.institute/wp-content/uploads/2018/08/Data-Scientist-Report.pdf>. Accessed: 2021-05-28.
- [20] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *arXiv preprint arXiv:1406.2661*, 2014.
- [21] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large Scale GAN Training for High Fidelity Natural Image Synthesis. *arXiv preprint arXiv:1809.11096*, 2018.
- [22] Sondre Fossen-Romsaas, Adrian Storm-Johannessen, and Alexander S Lundervold. Synthesizing skin lesion images using CycleGANs—a case study. In *Norsk IKT-konferanse for forskning og utdanning*, number 1, 2020.
- [23] Simon JD Prince. *Computer Vision: Models, Learning, and Inference*. Cambridge University Press, 2012.
- [24] Sagar Sharma and Simone Sharma. Activation functions in neural networks. *Towards Data Science*, 6(12):310–316, 2017.
- [25] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6. Ieee, 2017.

- [26] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.
- [27] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.
- [28] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.
- [29] Stanford Vision Lab. Imagenet Large Scale Visual Recognition Challenge 2014 (ILSVRC2014). <http://www.image-net.org/challenges/LSVRC/2014/results>. Accessed: 2021-03-01.
- [30] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multi-box detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.
- [31] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.
- [32] Joseph Redmond. pjreddie/darknet: Concolutional neural network. <https://github.com/pjreddie/darknet>. Accessed: 2021-04-20.
- [33] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.
- [34] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.
- [35] Olga Russakovsky, Jia Deng, Zhiheng Huang, Alexander C Berg, and Li Fei-Fei. Detecting avocados to zucchinis: what have we done, and where are we going? In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2064–2071, 2013.
- [36] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. The PASCAL Visual Object Classes (VOC) Challenge. <http://host.robots.ox.ac.uk/pascal/VOC/pubs/everingham10.pdf>. Accessed: 2021-03-15.
- [37] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

- [38] PASCAL VOC Challenge performance evaluation server. http://host.robots.ox.ac.uk:8080/leaderboard/displaylb_main.php?cls=mean&challengeid=11&compid=1&submid=29085. Accessed: 2021-04-14.
- [39] COCO - Comon Objects in Context. <https://cocodataset.org/#detection-leaderboard>, . accessed: 2021-04-14.
- [40] cocodataset/cocoapi: COCO API - Dataset @ <http://cocodataset.org/>. <https://github.com/cocodataset/cocoapi>, . Accessed: 2021-04-15.
- [41] Concatenated Skip Connection Explained. <https://ipywidgets.readthedocs.io/en/latest/>, 2021. Accessed: 2021-05-29.
- [42] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.
- [43] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.
- [44] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.
- [45] Magnetic Resonance Imaging (MRI). <https://www.nibib.nih.gov/science-education/science-topics/magnetic-resonance-imaging-mri>. Accessed: 2021-05-19.
- [46] Peter Mildenerger, Marco Eichelberg, and Eric Martin. Introduction to the DICOM standard. *European radiology*, 12(4):920–927, 2002.
- [47] PS3.1. <http://dicom.nema.org/medical/dicom/current/output/html/part01.html>. Accessed: 2021-05-03.
- [48] Nan Wu, Jason Phang, Jungkyu Park, Yiqiu Shen, Zhe Huang, Masha Zorin, Stanisław Jastrzębski, Thibault FÉvry, Joe Katsnelson, Eric Kim, et al. Deep neural networks improve radiologists’ performance in breast cancer screening. *IEEE transactions on medical imaging*, 39(4):1184–1194, 2019.
- [49] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghahfoorian, Jeroen Awm Van Der Laak, Bram Van Ginneken, and Clara I Sánchez. A survey on deep learning in medical image analysis. *Medical image analysis*, 42:60–88, 2017.
- [50] Alexander Selvikvåg Lundervold and Arvid Lundervold. An overview of deep learning in medical imaging focusing on MRI. *Zeitschrift für Medizinische Physik*, 29(2):102–127, 2019.

- [51] Ehsan Samei, J Anthony Seibert, Katherine Andriole, Aldo Badano, Jay Crawford, Bruce Reiner, Michael J Flynn, and Paul Chang. Aapm/rsna tutorial on equipment selection: Pacs equipment overview: general guidelines for purchasing and acceptance testing of pacs equipment. *Radiographics*, 24(1):313–334, 2004.
- [52] Jamie Cattell, Sastry Chilukuri, and Michael Levy. How big data can revolutionize pharmaceutical R&D. *McKinsey & Company*, April, 2013.
- [53] Trent Kyono, Fiona J Gilbert, and Mihaela van der Schaar. MAMMO: A deep learning solution for facilitating radiologist-machine collaboration in breast cancer diagnosis. *arXiv preprint arXiv:1811.02661*, 2018.
- [54] Trent Kyono, Fiona J Gilbert, and Mihaela van der Schaar. Improving workflow efficiency for mammography using machine learning. *Journal of the American College of Radiology*, 17(1):56–63, 2020.
- [55] Solution Archive - contextflow. URL <https://contextflow.com/solution/>. Accessed: 2021-05-10.
- [56] Office for Civil Rights. Guidance Regarding Methods for De-identification of Protected Health Information in Accordance with the Health Insurance Portability and Accountability Act (HIPAA) Privacy Rule. <https://www.hhs.gov/hipaa/for-professionals/privacy/special-topics/de-identification/index.html>, 2015. accessed: 2021-05-28.
- [57] Best Data De-identification and Pseudonymity Software in 2021 | G2. <https://www.g2.com/categories/data-de-identification-and-pseudonymity>. Accessed: 2021-05-24.
- [58] De-identification Tools | NIST. <https://www.nist.gov/itl/applied-cybersecurity/privacy-engineering/collaboration-space/focus-areas/de-id/tools>. Accessed: 2021-05-24.
- [59] Hauke Bartsch. mmiv-center/RewritePixel. <https://github.com/mmiv-center/RewritePixel>, . Accessed:2020-09-08.
- [60] Zobeir Raisi, Mohamed A Naiel, Paul Fieguth, Steven Wardell, and John Zelek. Text Detection and Recognition in the Wild: A Review. *arXiv preprint arXiv:2006.04305*, 2020.
- [61] Kaggle. Chest X-Ray Images (Pneumonia). <https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia>. Accessed: 2020-09-10.
- [62] Radiology Society of North America. RSNA Pneumonia Detection Challenge. <https://www.kaggle.com/c/rsna-pneumonia-detection-challenge/overview>. Accessed:2021-02-15.
- [63] Hauke Bartsch. HaukeBartsch/render_char_to_28x28: Render lines or random text into a series of DICOM files. https://github.com/HaukeBartsch/render_char_to_28x28, . Accessed: 2021-02-10.
- [64] wikipedia. Pytorch. <https://en.wikipedia.org/wiki/PyTorch>, . Accessed: 2021-03-25.

- [65] wikipedia. Torch. [https://en.wikipedia.org/wiki/Torch_\(machine_learning\)](https://en.wikipedia.org/wiki/Torch_(machine_learning)), . Accessed: 2021-03-25.
- [66] pytorch/pytorch: Tensors and Dynamic neural networks in Python with strong GPU acceleration . <https://github.com/pytorch/pytorch>. Accessed: 2021-01-25.
- [67] Sylvian Gugger Jeremy Howard. fastai: A Layered API for Deep Learning. 2020. doi: <https://arxiv.org/abs/2002.04688>.
- [68] Cloud Computing, Hosting Services, and APIs | Google Cloud. <https://cloud.google.com/gcp>, . Accessed: 2021-05-25.
- [69] Ross Girshick, Ilija Radosavovic, Georgia Gkioxari, Piotr Dollár, and Kaiming He. Detectron. <https://github.com/facebookresearch/detectron>, 2018. Accessed: 2021-04-25.
- [70] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. <https://github.com/facebookresearch/detectron2>, 2019. Accessed: 2021-04-25.
- [71] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.
- [72] Akshay L Chandra. `acl/image_bbox_slicer`: This easy-to-use library splits images and its bounding box annotations into tiles, both into specific sizes and into arbitrary number of equal parts. It can also resize them, both by specific sizes and by a resizing/scaling factor. https://github.com/acl21/image_bbox_slicer, 2019. Accessed: 2021-04-05.
- [73] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced deep residual networks for single image super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 136–144, 2017.
- [74] Alex Clark and Contributors. Pillow. <https://pillow.readthedocs.io/en/stable/>. Accessed: 2021-05-23.
- [75] ipywidgets – Jupyter Widgets 8.0.0a4 documentation. <https://paperswithcode.com/method/concatenated-skip-connection#>. Accessed: 2021-03-01.
- [76] Table of contents – voila 0.2.10 documentation. <https://voila.readthedocs.io/en/stable/index.html>. Accessed: 2021-05-29.
- [77] Binder. <https://mybinder.org/>. Accessed: 2021-05-29.
- [78] Cloud Application Platform | Heroku. <https://www.heroku.com/>. Accessed: 2021-05-29.
- [79] David Gunning. Explainable Artificial Intelligence (XAI). *Defense Advanced Research Projects Agency (DARPA), nd Web*, 2(2), 2017.

- [80] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-López, Daniel Molina, Richard Benjamins, et al. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, 58:82–115, 2020.
- [81] Erico Tjoa and Cuntai Guan. A survey on explainable artificial intelligence (XAI): Toward medical XAI. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.
- [82] v2.3. <https://github.com/fastai/fastai/projects/1>. Accessed: 2021-05-25.