Norwegian University
of Life Sciences

Master's Thesis 2021 30 ECTS
School of Economics and Business

Automated Redaction of Historical Documents using Machine Learning

Petter Kolstad Hetland
Entrepreneurship and Innovation

Tomas Sigerstad
Business Administration

Preface

This thesis is written at the School of Economics and Business, Norwegian University of Life Sciences (NMBU) in 2021. The thesis consists of 30 ECTS credits and marks the conclusion of a two-year master's degree in Entrepreneurship and Innovation for Petter Hetland, and Business Administration for Tomas Sigerstad. This thesis has been carried out in collaboration with Arkivverket, The National Archival Services of Norway, and Bouvet.

We would like to thank our supervisors, Kristian Hovde Liland and Oliver Tomic at the Faculty of Science and Technology, NMBU, for their support and guidance. We would also like to thank Mark Alan West and Halvor Grønaas at Bouvet for their mentorship and availability throughout this process. Furthermore, we would like to thank Arkivverket for the opportunity to work with this assignment.

Oslo, 1st June, 2021

Petter Kolstad Hetland
Tomas Sigerstad

Abstract

This thesis aims to assist Arkivverket, The National Archival Services of Norway, in automating the redaction of national identity numbers in historical documents. As historical documents are released to the public at request, it is necessary to prevent personal data misallocation. Today this is handled by manual redaction of national identity numbers performed by employees at Arkivverket. Implementing a workflow where a machine learning model suggests possible national identity numbers (NIDs) to the employee for redaction may save time and increase the overall amount of NIDs identified. Arkivverket has developed a machine learning prototype for automatic document redaction using Optical Character Recognition and other tools. However, the current solution is not sufficiently accurate to be put into production in a suggestion workflow as approximately 11% of the identity numbers are left unredacted (based on the recall score). With a recall score of 89.0%, a precision score of 88.3%, and an F1 score of 88.6%, this model is used as a baseline for the performance of machine learning models developed and trained in this thesis.

The thesis had two main goals. The first was to test whether object detection is a viable choice for automatically identifying NIDs. The documents contain many similar words and numbers, and many documents comprising a combination of hand- and machine-written text. The second goal, assuming that object detection is indeed a viable choice, was to check whether our detection models can reach a performance level that meets the demands of a suggestion workflow where each document is checked for NIDs by the model before being quality-assured by an employee and submitted. This would save time for the employees while preventing the unnecessary release of NIDs due to human error. In the long term, fully automated document redaction is the goal.

Results show that using object detection models based on the Detectron2 framework is a highly viable approach for this problem, perhaps in large part due to the models' ability to recognize difficult, handwritten national identity numbers. The fine-tuned models are capable of reaching scores beyond those of the current prototype developed at Arkivverket. The most accurate model achieved a recall score of 97.9%, a precision score of 94.9%, and an F1 score of 96.4%. Based on our estimations, this model correctly identified *more* NIDs in the dataset than its human counterparts at Arkivverket. A proposal for a deployment architecture is presented to illustrate the potential for combining our model and the existing redaction software to have a lasting economic- and ethical impact on the daily practices of Arkivverket. It is estimated that Arkivverket can initially save 65,417 NOK yearly after maintenance costs by implementing the proposed algorithm. With time and further research, however, the process of redacting national identity numbers may become fully autonomous and the savings potential greater.

Contents

List of Tables

List of Figures

1. Introduction

1.1 Background

Misallocation of personal data can arguably be described as a societal problem as it is often a factor in identity theft [1]. Not only is the sale and abuse of this data a hot topic in today's society, but recent cases imply that large amounts of personal data, such as national identity numbers, are published as a result of lacking identification of these [2, 3, 4]. In Norway, a national identity number is not confidential in and of itself but the mass publication of these on the internet may conflict with privacy laws [5].

In 2016, Aftenposten published a news article pointing out that Arkivverket, The National Archival Services of Norway, had been making historical documents available online which contained unredacted national identification numbers, amongst other person-specific data [6]. A spokesperson from Datatilsynet, The Norwegian Data Protection Authority, commented that since each of these documents was created and made available per request, Arkivverket did not break any laws. However, he commented that the solution was not ideal and that every sensitive data point should ideally be redacted.

Arkivverket wishes to comply with personal protection guidelines and is experimenting with applying Machine Learning (ML) to aid in redacting documents. Implementing a workflow where an algorithm suggests possible national identity numbers to redact and an employee quality-assures the suggestions, may have both economic and ethical/judicial benefits:

1. Significantly decreasing the amount of time spent by employees in the process of redacting national identification numbers, thus lowering costs and increasing employee productivity.

2. Increase the number of correctly redacted national identification numbers by allowing employees to perform quality assurance instead of the tedious and error-prone task of identifying every target in the document manually.

In the long term, an ideal goal would be for the models to be sufficiently accurate to autonomously redact and release historical documents per request.

1.2 Problem Statement

Arkivverket wants to automate document redaction, but the current prototype is not sufficiently accurate at identifying the target national identification numbers. As the current prototype model is based on Optical Character Recognition (OCR), the stagnation in performance may largely be due to difficult-to-read text of various

forms in the documents.

Two sequential goals were set for this project thesis:

1. To test whether or not machine learning models based on object detection can separate relevant national identification numbers from similar words and numbers in historical documents.

2. If the first goal is reached, to test if these models are sufficiently accurate to aid manual redaction by suggesting potential national identification numbers in historical documents.

Arkivverket has not formulated a specific accuracy-threshold for the model to be regarded as *sufficiently accurate*. The accuracy that would be sufficient for implementing a model into the workflow of employees would have to be decided through pilot projects where employees actually test the real-world performance of a prototype. An informal goal of 95 percent recall (i.e. that the model misses no more than 5 percent of the actual national identity numbers) has been set through talks with the project owners at Arkivverket.

In accordance with these goals, the report may have three distinct outcomes: The first potential outcome is that object detection models are not at all able to identify national identity numbers in the documents, failing both goals. The second potential outcome is that a model based on object detection is able to identify national identity numbers, but does not perform better than the current solution. The third potential outcome is that object detection models are sufficiently accurate to be deployed in a suggestion prototype through a trial project.

1.3 Structure of thesis

Chapter 2 will cover the theoretical background for this thesis. Chapter 3 will give an overview of the materials and data used, and Chapter 4 describes the methodology applied. In Chapter 5, the results are presented and discussed. Chapter 6 presents how this project may be implemented at Arkivverket and the possible risks and benefits, both economic and ethical, of implementing such a model. Results are summarized in Chapter 7, in addition to concluding remarks.

2. Theory

This chapter will give a theoretical foundation for this thesis. It covers the basics of data selection and processing and gives an overview of machine learning advancements over the years, with an emphasis on deep learning. An introduction to the inner workings of object detectors is provided and how they are used in the context of this project. Additionally, this chapter will give a brief background on the potential economic consequences of implementing such a model into the daily workflow.

2.1 On the topic of machine learning

Machine learning is a subset of artificial intelligence where a self-learning algorithm uses structured and/or unstructured data to derive knowledge. This technology is used in all spheres of society, affecting people's daily lives in more ways than many are aware of. Machine learning can be used for many purposes, ranging from email spam filters and voice recognition to autonomous cars and the detection of cancer cells [7].

2.1.1 Different types of learning

Machine learning can be broken down into three types of learning: supervised learning, unsupervised learning, and reinforcement learning.

With *supervised learning*, the main goal is to learn based on already labeled data. The model will look for similarities among the observations with the same label and utilize these to predict labels of unseen or future data. As the algorithm learns, it is given the task of predicting the label of observations where the label is known. If the predicted label is identical to the ground-truth label, the model is rewarded. If the predicted label is wrong, the model is updated.

When dealing with unstructured or unlabeled data, one can use *unsupervised learning*. The algorithm will explore the data to find systematic information without the guidance of labels.

The goal of *reinforcement learning* is to develop a system that interacts with its environment and improves its performance through rewards and penalties. A chess engine is an example of this, as the algorithm decides its moves based on its environment, i.e. where the chess pieces are placed. The end goal is to win, and the algorithm is given a positive or negative signal based on whether it achieves victory or not [7].

2.1.2 Machine Learning Project Life Cycle

The objective of a machine learning project is to solve a problem. Typically, the first step is to understand the problem and why it needs to be solved. From this

understanding, the project may be formed, and a defined goal should be set. This goal should include what the model will receive as input, the desired output, and what rate of error is considered to be acceptable.

Based on the goal of the project, relevant data should be collected, selected, and prepared. When the data is ready, a model can be trained and evaluated. This process is repeated until the predictive performance of the model converges. When the model is ready, it can be deployed and begin to serve its purpose. Changes in input data or in the statistical relationship between the features and the label might cause model performance degradation [8]. Therefore, the model should be monitored and maintained to ensure that the performance does not deteriorate over time. Figure 2.1 shows a typical life cycle of a machine learning project.

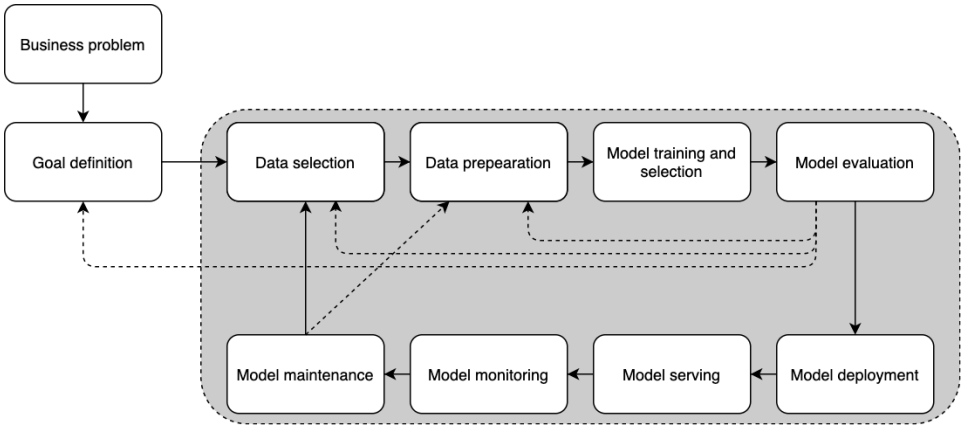


Figure 2.1: *The life cycle of a machine learning project*

The methods proposed for monitoring and maintaining deployed models in this thesis are described in Section 6.1.3.

2.2 Dataset

Data is the basis for all machine learning and the quality of the dataset has great consequences for the output quality of any algorithm. The algorithm is exposed to a set of data that is used for training and then exposed to another set of data used to test how the algorithm performs. To enable this process, the original dataset is divided into a *training dataset* and a *test dataset*. To get the data ready for a machine learning algorithm, it is often necessary to go through the preparatory step of *pre-processing*.

The dataset used for this thesis is described in Chapter 3, and the methods applied in processing the dataset are described in Chapter 4.

2.2.1 Bounding boxes

Before an image is used for training a model, bounding boxes are widely used to describe the target location of objects, such as a personal identity number, within the image [9]. These objects are outlined and classified so that the machine can learn their shape and form, with the end goal of a model to recognize, localize and classify similar objects in other images. The bounding box typically has the shape of a rectangle where the location within an image is determined by x- and y-axis coordinates in both the upper left corner and the lower right corner of the box.

The bounding boxes that were provided by Arkivverket are discussed in Section 3.1.2.

2.2.2 Training-, validation-, and test-sets

To ensure that a machine learning algorithm performs well, not only in a training environment but also when introduced to new data, the dataset is divided into separate "sets". The divided parts are referred to as training data and test data. Oftentimes it is appropriate to add a third division, the validation dataset, [10]. The sizes of these parts may vary, but a typical distribution is 70/15/15. The training dataset consists of 70% of the data and test and validation sets of 15% each. In the process of selecting data for the training-, test and validation-sets, it is important to ensure similar characteristics and features across all sets to avoid non-representable results when testing the algorithm. In the case of this thesis, data similarity may translate to similar styles of text in the documents, image dimensions and other attributes. For this thesis, the dataset splits are *stratified* by a custom value (Φ), that gives each sample a value based on the size of its bounding boxes relative to the size of the image. Stratification of dataset splits means choosing an attribute in the dataset and making sure each of the splits have a similar distribution of values for this attribute. More information on the Φ-attribute and dataset-splits is provided in Section 4.2.1.

The training dataset is the data used to train the model. During this process, the model learns the structure of the data and the similarities of the observations with the same labels. Based on this, the parameters of the model are adjusted to solve the problem in question.

After the algorithm is fit on the training set, one can use a validation set to see how it performs, with the option of making adjustments intact. The resulting error rate provides an estimate of the future error rate. If the error rate is too large, one can adjust the model's parameters to improve prediction performance without compromising the test split.

When a model is fit on the training data, with satisfying results when applied to the validation set, it is introduced to the data it has not seen before, the test dataset. This is done to review its performance and to estimate its ability to generalize,

meaning the model's ability to react to unseen data. As the test dataset is new to the algorithm, it can indicate how the model will perform when introduced to future data outside the dataset.

2.2.3 Preprocessing

Data preprocessing is a preliminary step often executed before further training to improve and ensure the input data quality [11]. This step may include removing outliers, compensate for missing values, and scale observations to make samples more similar and increase the quality of the input data to the model. In the specific case of image analysis, preprocessing may also include adjustment of exposure, sharpening, and resizing to make images more similar.

Preprocessing, data selection and descriptive statistics are discussed in Chapter 4.

2.2.4 National identity numbers

National identity numbers were introduced in Norway in 1964 to simplify the identification of individuals [12]. All Norwegian citizens and those who settle in Norway long term are given a national identity number of 11 digits. The first 6 numbers are defined by the date of birth (dd.mm.yy). The subsequent 5 digits are defined as their personal number [13]. The first three digits of the personal number are individual numbers. This number is allocated depending on the date of birth. The individual number for people born between 1854 and 1899 are allocated from series 500 to 749. From the year 1900 to 1999 are allocated from series 000 to 499. Additionally, the individual number for people born between 1940 to 1999 can be allocated from series 900 to 999. From 2000 to 2039 are allocated from series 500 to 999. The following digit of the individual number indicates gender where odd numbers indicate male and even female. The last two digits are control digits. Figure 2.2 shows how the national identity numbers are structured.

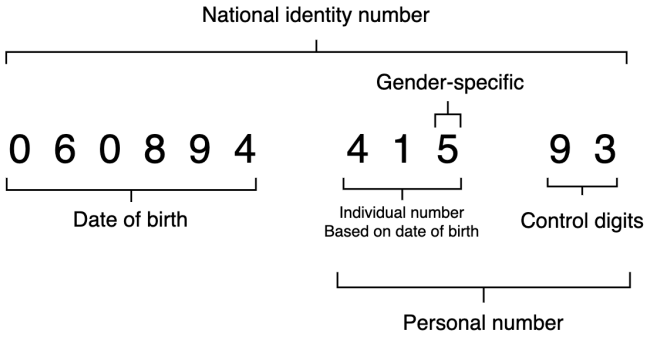


Figure 2.2: *Illustration of how national identity numbers in Norway are structured*

Neither the national identity number nor the personal number is regarded as sensitive national information. However, the use and storage of these numbers are regulated by Norwegian law. The national identity number is only to be used to identify an individual when the cause is justified. Examples of this include tax reporting and credit checks.

Historically an individual's national identity number would grant a person access to confidential information. However, in recent years other means of identification is often required in addition to providing the national identity number.

In accordance with wishes from Arkivverket, this thesis will focus on redacting the last 5 digits of national identity numbers (the personal number) from historic documents. By doing so, the date of birth will remain visible, whilst keeping the individual's privacy protected.

Examples of national identify numbers in the historical documents explored in this thesis are shown in Figure 3.3.

2.3 Deep learning

A machine-learning algorithm uses a set of rules to make predictions from the data [14]. These rules are called the machine learning model. By utilizing these rules, the model processes points of data and returns a prediction. As the model gets feedback on right and wrong predictions, it learns by adjusting the rules. Deep learning is a subset of machine learning used to train networks of artificial neurons and is well suited for analyzing images [7].

2.3.1 Artificial neurons

In 1943, Warren McCulloch and Walter Pitts published the first concept of a simplified brain cell [15]. In their research, they were trying to understand how the human brain works with the purpose of designing artificial intelligence. The simplification is referred to as the McCulloch-Pitts Neuron. They described the neuron as a simple logical gate with binary outputs. Multiple inputs arrive at the neuron and each input is *weighted* according to individual weights ($w_1, w_2 ... w_n$) that are set for each node. If the accumulated weighted input reaches the threshold of the neuron, an output signal is generated.

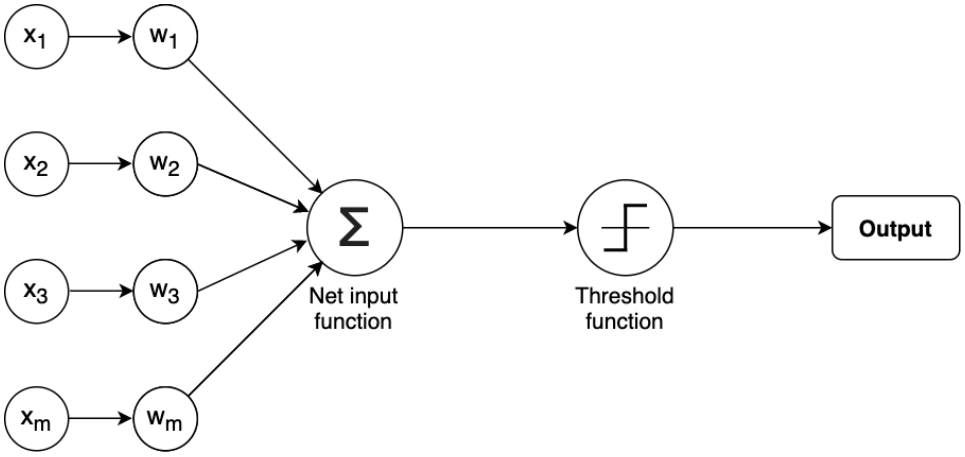


Figure 2.3: *Illustration of the McCulloch-Pitts Neuron*

Perceptron

Based on the McCulloch-Pitts Neuron, Frank Rosenblatt published the first concept of the Perceptron, an algorithm to classify the input received [16]. The model finds the optimal weight of the coefficients and multiplies them with the input features. By doing so, the model can determine if a neuron transmits a signal or not and if they should be included.

$$z = w_1 x_1 + w_2 x_2 + \cdots + w_m x_m \tag{2.1}$$

The net input, z, is the aggregate sum of each weighted input node from the previous layer.

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}, \ \boldsymbol{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix} \tag{2.2}$$

x is a vector of all input values to the layer, and w is a vector of all input weights to a layer.

Adaline

ADAptive LInear NEuron (Adaline) [17] is similar to the Perceptron, a single-layer neural network with a single neuron. The main difference is that the Adaline model uses a linear activation function to measure loss before applying the threshold function. This allows the Adaline neuron to update its weights based on the degree of error, as opposed to the true-or-false approach implemented by the Perceptron.

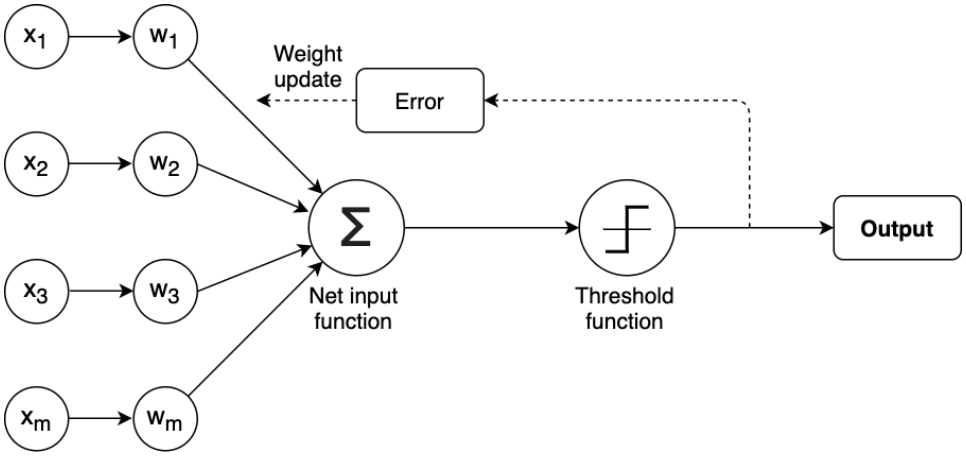


Figure 2.4: *Illustration of a Perceptron*

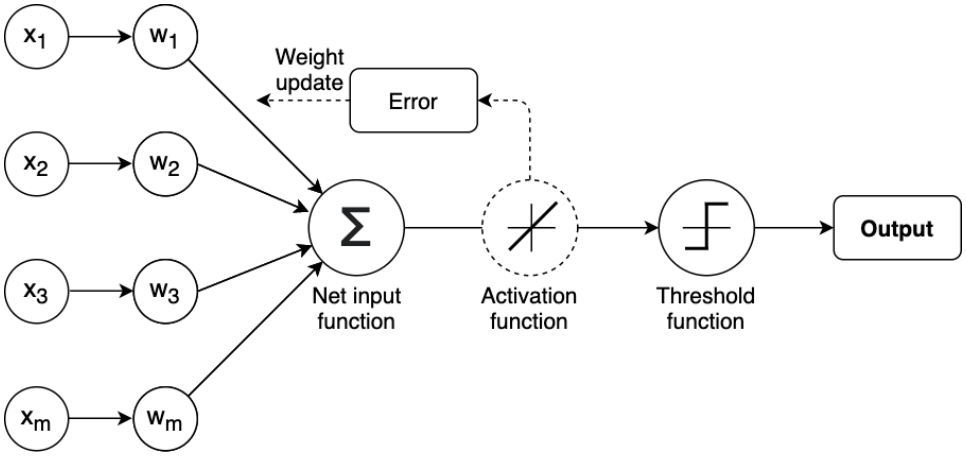


Figure 2.5: *Illustration of the Adaptive Linear Neuron*

2.3.2 Artificial neural networks

As single neurons have limited capacity for learning all relevant patterns in the data, they can be grouped and organized as units in an artificial neural network. The network may consist of multiple layers of parallel units. These units, or neurons, are activated by the same type of activation function (see Section 2.3.2 on activation functions) but each neuron has unique weights associated with the outputs of neurons in the previous layer which are adjusted during training. In the first layer, each input feature is typically distributed to an individual input node before being passed on to a given number of nodes in the next layers.

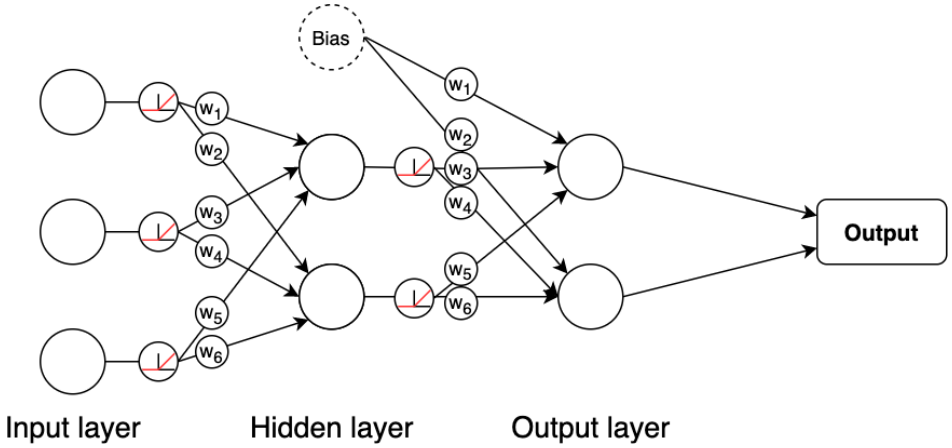


Figure 2.6: *Illustration of layers in an artificial neural network*

Hidden layers and output layer

The layers between the input and output layers in an artificial neural network are referred to as hidden layers. A network can consist of any number of hidden layers, and those containing more than one are referred to as a *Deep Artificial Neural Network*. Each of these layers consists of a bias unit in addition to an arbitrary number of units, or neurons. The hidden layers' neurons receive input from the preceding layers and multiply the inputs by their respective weights before an activation function is applied and the outputs are passed on to the next layer.

The output layer multiplies the weights and inputs from the last hidden layer and may use an activation function on the net input. The number of nodes in the output layer depends on the machine learning problem, with each node representing a class or regression value. If the network's objective is regression with a single output value, the output layer consists of a single neuron representing the regression value, and no activation function is applied. On the other hand, a multi-class problem with n classes requires an output layer with neurons for each class, resulting in n neurons. The output values for each of these neurons represent the probability that a given sample belongs to each of the possible classes after an activation function is applied to each node's net input.

Deep Convolutional Neural Networks

Deep Convolutional Neural Networks, a neural network architecture, is widely used as a foundation for Machine Learning models to analyze and classify objects in images [18]. The method was first developed by Yann LeCun and his colleagues in 1989, as they proposed a method for classifying handwritten digits from images [19]. In 2012, Deep Convolutional Neural Networks attained much attention after outperforming all its competitors in the image classification contest ILSVRC [20]. Since then, it has become a popular field of research leading to major improve-

ments to the method.

For a machine-learning algorithm to perform well, the network must extract relevant features from the data. If done manually, this would require a high degree of domain knowledge. However, Deep Convolutional Neural Networks can automatically learn the features directly from the raw data. The layers of the network can be regarded as feature extractors, as they can recognize and categorize the information in the data to predict a target value or class label.

After the model is fed a numerical representation of an image, it will construct a feature hierarchy. By synthesizing the low-level features, such as textures and edges, it forms high-level features which are more complex shapes, such as the outline of a building or an animal [7].

Activation functions

Activation functions are used to determine the output of each neuron after multiplying the input values by the corresponding weights. Examples of activation functions are the Sigmoid function, which takes any real number as an input and outputs a value between 0 and 1, and the ReLU activation function which returns zero for negative numbers but does not affect positive input values. In order to prevent common neural network challenges such as vanishing or exploding gradients (see Section 2.3.4), activation functions are applied to the output values of hidden layers. Activation functions introduce non-linearity to the outputs, traditionally in order to simulate whether a neuron is "firing" or not based on the input, but research has since found ways of using different activation functions to increase backpropagation speed and allow scaling of models (see section 2.3.3).

The logistic sigmoid function, often referred to as the sigmoid function due to its shape, takes any input value and returns values close to 1 for large positive numbers, and values close to 0 for large negative numbers. It is mainly used in the final layer of binary classifiers (where the model predicts either 0 or 1) to determine the probability that a certain sample belongs to a certain class. Historically, the sigmoid function has been used as an activation function between hidden layers as well, but research has proven that models using the sigmoid function as an activation function do not scale well beyond a few network layers, mainly because of the problem of vanishing gradients (see 2.3.4).

Equation 2.3 shows the sigmoid function [7].

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.3}$$

Rectified Linear Unit (ReLU) is a non-linear activation function that is well suited for learning complex functions with neural networks [7]. One reason for this is that the derivative of the ReLU function, with respect to its input, is always 1 for

positive values, and 0 for negative values. This is what makes it a good choice when trying to prevent the problem of vanishing or explodng gradients (see Section 2.3.4 on scaling neural networks).

Equation 2.4 shows the ReLU function [7].

$$\sigma(z) = max(0, z) = \begin{cases} z & \text{if} \quad z > 0 \\ 0 & \text{if} \quad z \leq 0 \end{cases} \tag{2.4}$$

2.3.3 Optimization

The optimizer determines how the model will learn [21]. By changing the model's attributes such as weights and learning rate, the optimizer may reduce loss and optimize performance by determining the way a model updates weights through backpropagation, as well as introducing regularization of models.

Gradient descent and backpropagation

Gradient descent is one of the most common optimization algorithms used to optimize neural networks. In simple terms, gradient descent is one of the ways a machine learning algorithm "learns" from its mistakes and is able to correctly predict unseen data samples. As illustrated in Figure 2.7, the weights of the model are adjusted taking repeated steps in the opposite direction of the gradient, i.e. the derivative of the cost function, until the cost function is at, or close to, zero. The slope of the cost function is based on how much the predictions made by a model differ from the ground truth, and at each iteration, the set of weights in a model is "nudged" in the direction that minimizes this slope. How much the weights are changed at each iteration is determined by the *learning rate*.

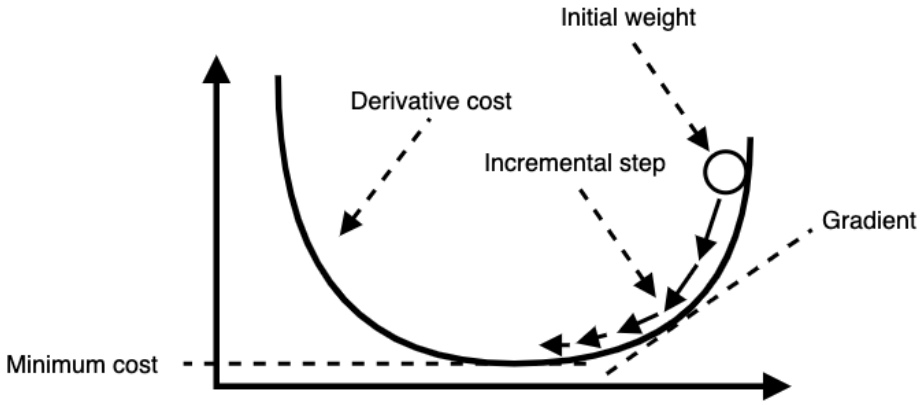


Figure 2.7: *Illustration of how the weight is adjusted to find the minimum cost using Gradient descent*

Backpropagation is an algorithm that computes the partial derivatives of each pa-

rameter using the chain rule for derivatives of complex functions [8]. At each iteration over the sample data, or epoch, gradient descent updates all parameters using partial derivatives. In feed-forward neural networks, this means that after the network has made a prediction based on the input values, the predictions are compared to the wanted output and each set of weights are changed iteratively from hidden layer to hidden layer, from the output layer towards the input layer.

Learning rate and weight decay

The learning rate is a hyperparameter determining the magnitude of change in weights at each iteration as backpropagation attempts to reach the minimum of the loss function. As a model is being trained, one can adjust the learning rate according to a predefined learning rate schedule.

To improve the generalization of the learning algorithm, weight decay can be implemented. Weight decay means gradually reducing the value of the learning rate as the epochs progress. As a result, the parameter updates become finer, reducing the chances of overshooting the cost function minimum [8]. In this thesis, weight decay is implemented through a *learning rate scheduler* which reduces the learning rate as epochs progresses (see Section 4.5.4).

Regularisation

Overfitting a model to the training data is a common challenge in both linear- and non-linear machine learning models, and is usually correlated with the degree of complexity in the model. Overfitting means a model might perform well on training data, but due to the high variance in the predictions for similar samples, the model does not generalize well to the test data. A large number of parameters, i.e. the total number of weights for all neurons across all network layers may be a cause of this problem. The increasing complexity of the model increases as the total number of parameters increases. As more parameters increase the model capacity for learning patterns, the model may eventually start adapting to "noisy" patterns in the data, i.e. patterns that are not relevant for future predictions. At this point the model is too well adapted to the training data, hence the term over-fitting.

Regularization of the model parameters is a method used to prevent overfitting by maintaining the model's ability to generalize well to the data and not adapt to noise in the dataset. Common techniques for regularizing a neural network include adjusting network width and depth, L1- and L2-regularization, dropout, and others [22].

Perhaps the most straightforward method for regularizing a model is to decrease model complexity by reducing the number of total parameters in the network, either by reducing the number of layers or the number of neurons in each layer and thus force the model to learn the more essential patterns in the training data. L1- and L2-regularization are methods for penalizing large weights in the network by adding additional terms to the cost function used in backpropagation.

In the architectures implemented in this thesis, dropout is used frequently as a regularization tool. By dropping random units and their connectors from the neural network during its training, the network can not rely on the activation of a set of hidden units and must therefore learn redundant representations in the network. For every iteration, a different set of units are dropped. The user determines the dropout rate, or the dropout probability, ranging from 0 to 1. When a fraction of units are left out, the weights of the remaining units are re-scaled accordingly. One of the reasons for choosing dropout as a regularization method in these architectures is its simplicity, making it well suited for architectures of as many as 50 or 101 layers.

Batch-normalization

Batch-normalization (or batch-norm) is a method for normalizing the input values of each layer [23]. This is done by calculating the mean and variance of the layer's input in a batch of a given size and utilizing these values to normalize the layer inputs. It then scales and shifts in order to obtain the output of the layer, using parameters that are learned along with the original parameters of the network. During inference, the learned parameters for normalizing batches from training are applied.

2.3.4 Scaling models

Though artificial neural networks tend to perform well on a variety of problems, it is common to run into challenges when *scaling* these networks by increasing model depth and thus model complexity. "Plain" networks tend to scale well to 16-30 layers and thus benefit from a higher amount of features. However, as plain networks are scaled above 30 layers, they often perform worse than the shallower networks [24]. This is largely due to the problem of *vanishing gradients* and can be solved by implementing skip-connections, introduced in ResNet models [24], and increasing model cardinality, introduced in ResNeXt models [25].

Vanishing and exploding gradients

As more layers with certain activation functions (such as the sigmoid function) are added to neural networks, the gradients used to update weights by way of backpropagation get gradually smaller for each layer [26]. Using the sigmoid function as an example, the derivative of very large negative or positive input values is output as very close to 0. As the gradient is multiplied by the gradient of earlier layers, a very small derivative value in the earlier layer will cause the multiplication of two very small numbers, and weights in early layers of large networks are updated little to nothing. This results in reduced learning. There are various ways to reduce the vanishing gradients problem. A common way to do it is to change the neural network's activation functions from sigmoid or tanh to Rectified Linear Unit. Using ReLU as the main activation function prevents the squashing of output values in a value range of [0, 1]. Thus, it helps keep derivatives of values in backpropagation at a non-diminishing level [7].

Using ReLU only partly solves the problem of vanishing gradients when scaling neural networks. Another popular way of preventing the issue is to use *skip connections* when designing the architecture. A more recent method is to introduce the concept of *model cardinality* as an alternative to only increasing model depth.

Skip-connections

Skip-connections, or layer shortcuts, were introduced by He, Zhang, Ren, and Sun in 2015 [24]. In a model architecture they named Residual Neural Networks or ResNets, they reused activations from earlier layers (often skipping 2-3 layers) as inputs to layers. This helped reduce the issue of vanishing gradients and made for more robust, and simpler, neural networks [24].

Cardinality versus layer depth

Model cardinality was introduced in 2017 by Xie et al [25]. In the paper, they further developed the ResNet architecture by implementing a multi-branch architecture that repeated ResNet building blocks "horizontally" for each level of the network, allowing for more complex networks without increasing layer depth. For any given building block (e.g., a set of layers in a network that takes an input of N channels and returns an output of N channels) in a ResNet architecture, the cardinality term implies repeating the building blocks in a homogeneous manner "alongside" the very same building block M times, where M is the cardinality-factor or the size of the set of transformation. The improved model architecture was named ResNeXt and proved to outperform previous ResNet models (at the expense of larger, and thus slower, models).

2.3.5 Transfer Learning

Transfer learning utilizes what a machine-learned to solve one problem and applies that knowledge to solve another problem [27]. In practice, transfer learning means using a model which is previously trained on other, often large, datasets, and then adapting the model to a specific problem. By freezing the weights of the first layers of the model and then nudging the weights in the remaining layers, the pretrained model will keep its former knowledge of low-level features and adapt its ability to combine these features to recognize objects in the new medium - document images in this case.

Machine learning models are traditionally designed to address single tasks, however, the development of algorithms enabling transfer learning is a topic of interest among researchers in the field of machine learning. As the training of deep learning models often requires substantial resources, transfer learning is a technique used to minimize the required sample size and enhance the training of the model, both in terms of speed and performance. Figure 2.8 illustrates the possible benefits of transfer learning. Because the model has the knowledge to draw on before it is trained further, the initial performance surpasses that of a model not using transfer

learning. The learning rate in the training phase is illustrated with the steepness of the curve. As illustrated, the final level of performance achieved when using transfer learning has the potential of exceeding the performance of a model without transfer learning.

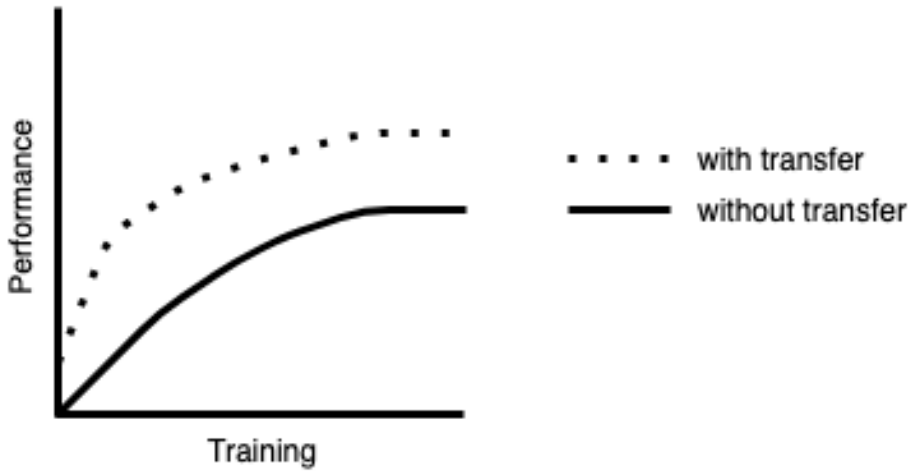


Figure 2.8: *How the use of transfer learning affects performance and training time*

Many research institutions have released models trained on vast and comprehensive datasets, open and free to use. Because of the resources required to train such models, this approach is widely used, especially when working with images or language data. The University of Oxford, Google, Microsoft, and Stanford University are examples of institutions that have released pretrained models for reuse. These can be used in full or parts, depending on the fit between the pretrained model and the problem at hand. For optimal performance, the model will oftentimes need to be fine-tuned according to the given task. A reason fine-tuning pretrained network work so well is that the model already knows how to recognize low-level features such as shapes, edges, and blobs.

2.3.6 Object Detection

Object Detection is a technique that seeks to identify and locate different objects in images and label them with specific class names, such as number plates on cars or a parrot [28]. Utilizing convolutional neural networks, the model analyzes the image, searching for patterns or structures to identify objects. Traditionally, an image classifier would find class labels characterizing the entire or the most dominant contents of its given image. However, an object detector's goal is to find multiple objects within an image and provide the object's location via a bounding box (see Section 2.2.1). An object detection algorithm's output values would typically include a list of bounding boxes in the form of x- and y-coordinates for each ob-

ject. Secondly, it will provide the class labels associated with each bounding box. Finally, the algorithm will rate its predictions, giving a score indicating the match between the object and the label. Oftentimes the probability of a correct match is used as the prediction score.

Regional Convolution Neural Networks (R-CNN)

To overcome the limitations of normal convolutional neural networks for object detection tasks, Ross Girshick et al. proposed a method where a *selective search* is used to extract 2000 regions from an image [29]. In this way, the convolutional neural network can determine an object's presence in each of the 2000 regions, a significantly smaller number of regions than what would be needed if no selective search were applied.

Each of the 2000 regions is warped into a square and fed into a convolutional neural network, yielding a feature vector as an output. To determine whether an object is indeed present in each of the 2000 regions, a support vector machine is used to classify the image based on its feature vector.

Fast R-CNN, Faster R-CNN, and Mask R-CNN

The regional convolution neural network solved a lot of the problems related to object detection using neural networks [30]. However, it suffers from a few glaring drawbacks:

- It is slow, as the convolutional neural net that functions as a feature extractor have to analyze 2000 regions, or images, per image to be predicted.
- The selective search algorithm is fixed, e.g., it does not implement learning to improve its ability to identify the 2000 most relevant regions in the image.

To deal with these drawbacks, the **Fast** Regional Convolution Neural Network was proposed by the same author [31]. This version removes the need to feed every proposed region to the convolutional neural network. Instead, the full input image is fed to the convolutional neural network, which creates a feature map used to identify proposed regions. Again, the proposed regions are warped into squares but are then processed by a pooling layer to enable them to be fed into a fully connected layer.

With the implementation of the Fast R-CNN architecture, the convolutional neural network no longer needs to process 2000 regions per input image but only processes each input image once. However, there is still a large bottleneck in the architecture, the Region Proposal Networks that tries to identify interesting regions from the feature maps that are generated for each input image.

To deal with this bottleneck, the **Faster** Regional Convolution Neural Network was proposed by Shaoqing Ren et al. in 2016 [32]. Contrary to the two algorithms mentioned above, this version does not perform a selective search for potential regions of interest but allows the network to learn the region proposals by itself.

Faster Regional Convolution Neural Networks start processing an image in the same way as the Fast-version by extracting a feature map from the input image. Then, instead of using a selective search algorithm to analyze the feature map, the region proposals are predicted by a separate model. The rest of the process is similar to that of the Fast Regional Neural Network. The Faster Regional Convolution Neural Network is many times faster than the Fast-version, and orders of magnitude faster than the original Regional Convolution Neural Network.

The **Mask** Regional Convolution Neural Network, proposed by Kaiming He et al. in 2017 [33], represents yet another evolution in the line of R-CNNs. All in all, this version is very similar to the Faster R-CNN. It does, however, add another layer of complexity. Instead of only outputting predicted bounding boxes for objects in the image, it also outputs a binary mask for each region of interest. The mask is predicted using a fully connected network, as a mask is predicted on a pixel-to-pixel basis. This fully connected network takes a single region of interest as an input and outputs the region's mask representation.

Backbones and heads

Both Faster R-CNN and Mask R-CNN are two-staged predictors made up of:

- A Region Proposal Network that extracts feature maps from the input image and predicts regions of interest.
- A second stage (in essence Fast R-CNN) that extracts features from each region and performs classification and bounding box-regression.

Common terms for these two stages are the network *backbone* architecture and the network *head*. The object detection algorithms in this thesis use one of two main backbones. 1. A ResNet Feature Pyramid Network of 50 or 101 layers, or 2. A NesNext Feature Pyramid Network of 101 layers.

ResNet, ResNeXt and Feature Pyramid Networks

Deep Residual Networks (ResNets), proposed by He Kaiming et al. in 2015, [24], described a way of dealing with a common problem where deeper networks were becoming increasingly difficult to train due to the vanishing gradient problem. The residual learning framework presented in the ResNet paper used *connection shortcuts* or residual connections between the layers of the network. In short, the residual connection is connecting the output of previous layers to the output of new layers. ResNets are still viable candidates for image recognition and object detection tasks and are used in combination with Feature Pyramid Networks (as explained below) for four of the six architectures tested in this thesis.

The ResNeXt architecture, proposed by Xie et al. [25], is quite similar to the Deep Residual Network architecture. The main difference is the introduction of a new dimension called *cardinality*. This dimension results from "homogenous, multi-branch architecture" [24], The network is constructed by repeating a building block

that aggregates a set of transformations with the same topology. In short, these building blocks allow the model to have an increased capacity when compared to its ResNet counterpart, without going deeper or wider. The ResNeXt architecture is used as a backbone for two of the six models tested in this thesis.

Pyramid representations of images are used in recognition systems for detecting objects at different scales. As Figure 2.9 illustrates, the same image can be re-scaled and analyzed on multiple levels to make predictions. Processing multiple-scale images are both memory and computationally intensive. By creating pyramids based on features, one can minimize the memory and computational costs, but this technique generally has a lower accuracy for object detection.

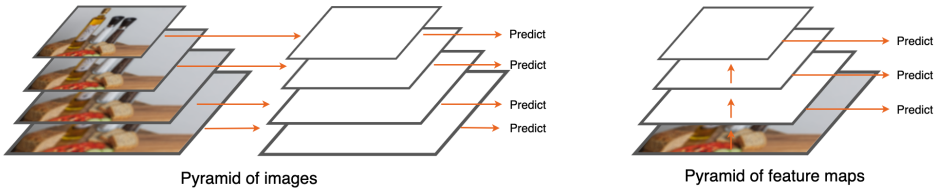


Figure 2.9: *Illustration of the difference between Pyramids of images and Pyramids of feature maps*

The Feature Pyramid Network [34] is a feature extractor designed to be both cost-efficient and accurate. As Figure 2.10 illustrates, the architecture uses a multi-scale pyramidal hierarchy of deep convolutional networks to construct feature pyramids.

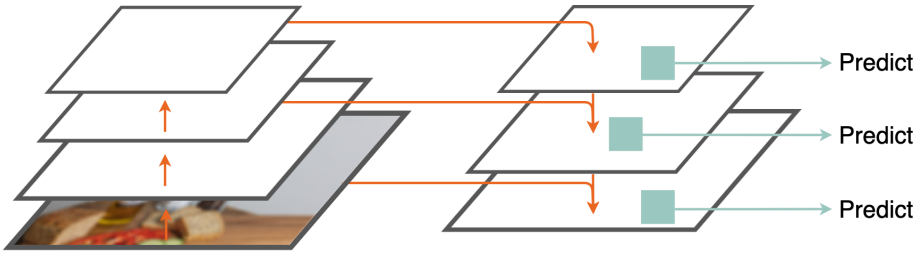


Figure 2.10: *Illustration of how a Feature Pyramid Network constructs pyramids*

In this thesis, all six architectures that are tested use Feature Pyramid Networks in combination with either a ResNet network or a ResNeXt network.

Object Detection Frameworks

As detecting objects in images requires a vast amount of training data to provide accurate predictions, this thesis will focus on utilizing pre-trained models' benefits through transfer learning (see Section 2.3.5). In this thesis, the main focus will be on the well-developed Detectron2 framework for transfer learning in object detection.

Detectron2

Detectron2 is a modular object detection library built on Pytorch [35]. With the popular Detectron open-source project as a foundation, the Detectron2 library offers an array of pretrained models with functionality including:

- Object detection with bounding boxes (used in this thesis)
- Semantic segmentation, where one assigns a class to each pixel in an image to predict its contents [36]
- Panoptic segmentation, as with semantic segmentation, each pixel is classified. However, one also seeks to identify the number of instances of each object
- DensePose, used for mapping of pixels in an image containing humans to make a 3D-model representation of them [37]

This thesis implements bounding box-based models from the Detectron2 Model Zoo [35] that are pretrained on the COCO (Common Objects in Context) dataset.

Table 4.6 provides an overview of the models tested in the thesis.

2.4 Economic background

Artificial intelligence and its subsets, including machine learning, have an increasing impact on workplaces worldwide, and the trend is expected to increase in magnitude. Surveys have found a high level of anxiety regarding job security as automation and new technologies are being introduced. In the popular press and academic circles, warnings have been raised about the loss of jobs as machines do work previously carried out by people [38].

2.4.1 History of automating tasks

Technological advancements have always had an impact on how labor is executed. The invention of the steam engine, electricity, and communication and information technologies are typical examples of advancements that have caused a paradigm shift in the workplace. In recent years, many have pointed to the introduction of artificial intelligence as such an advancement [39].

Artificial intelligence is already an important part of many workplaces [40]. Intelligent chatbots assist in customer support, algorithms calculate scenarios and provide

decision support, and machines can perform medical diagnostics based on images. Artificial Intelligence is not limited to office buildings but has become a part of millions of people's daily lives. Digital assistants implemented in smartphones, robot vacuums, and camera systems with automatic number plate recognition are relevant examples.

The automation of tasks correlates with the development and implementation of new technologies that, in turn, allows for further automation of work [41]. The reinstatement effect describes how labor is introduced to a broader range of tasks due to automation. As the capital, such as machines, overtake tasks previously performed by labor, new tasks arise. Where there is an increase in tasks performed due to the combination of labor and capital, there is a rise in productivity.

2.4.2 Task automation and its impact on employment

Historically, when capital has replaced labor in one place, new jobs have been introduced. How recent development in advanced technologies, such as machine learning, will implicate human labor's future is subject to great debate. More tasks will be subject to automation, but there is no empirical evidence implying major scale effects on the general employment rate.

2.4.3 Estimating profit of a machine learning project

In order to estimate the economic profitability of a machine learning project, the savings potential and project costs are important factors.

Currently, the employees at Arkivverket redact each document manually. The savings potential of implementing a model that assists in this process can be calculated from estimations made on how much time the employees will save. In turn, the savings can be quantified by estimating labor costs saved.

The project costs can be calculated by estimating the time spent on deploying and maintaining such a model. As with the savings potential, the labor costs can be used to quantify the project costs. The project profitability can be calculated by subtracting the expenses from the savings potential.

All comments made about the profitability of implementing the proposed model in this thesis are based on approximations and estimations. If the proposed model were to be implemented, there will always be a level of uncertainty affiliated with the potential profitability of such a project.

2.5 Model deployment

Assuming that a machine learning model performs sufficiently well for a given problem, such as identifying national identity numbers on document pages, a critical next step is making the model accessible for the project's beneficiaries. Chapter 6 describes how each of the concepts in this section may be implemented in the case of this project.

2.5.1 Static versus dynamic deployment

There are several ways of deploying a machine learning model. The four most common patterns [8] are:

- Static deployment, as part of an installable software package
- Dynamic deployment on a user's device
- Dynamic deployment on a server
- Model streaming (disregarded in this report)

As the first three items show, the two main methods for deploying models are *static deployment* and *dynamic deployment*. The difference between the two methods is apparent from the names. Dynamic models (more so than static models) are meant to be improved or maintained continuously after the model's initial release, without active involvement from the end-user. Static deployment methods, however, resemble classical software development where the machine learning code is packaged once and distributed as a resource available at runtime. Static deployment methods have a few advantages over dynamic methods, as they are inherently fully available to the end-user, even offline. The data sent to the developed model does not have to be uploaded to a server, making this method faster for the end-user.

On the other hand, dynamic deployment methods have distinct advantages as opposed to static methods. As mentioned, dynamic deployment methods allow updating the machine learning model or prediction pipeline with little or no effort on the part of the end-user. This is an advantage in applications where the model is used by third-party software such as redaction software by suggesting predictions that may or may not be part of the final redaction product. Another advantage of dynamic deployment methods, especially those that are server-deployed, is the ability to separate hardware-specific requirements from the device or machine of the end-user. In object detection, a high-performing GPU is a requirement (or highly preferred) for performing predictions on high-resolution images.

For a project such as identifying and redaction of document images, dynamic deployment methods seem to be the ideal choice. Furthermore, deploying the model on a web-based server will allow the end-user to passively take redaction suggestions as input to a self-developed or third-party redaction software. One can imagine on-device deployment being useful as well, for instance, by allowing a user to take an image of a physical document, apply redaction suggestions, and then pass it directly on to a recipient. Due to the nature of this thesis, only server-based dynamic deployment is described in detail.

2.5.2 Dynamic deployment on a server

REST API

When a model is deployed on a web-based server, it is most often made available through a *Representational State Transfer application programming interface* or REST-API [42]. Representational State Transfer is, in short, an architectural style for using HTTP coding to receive information from online servers based on JSON- or XML-encoded requests.

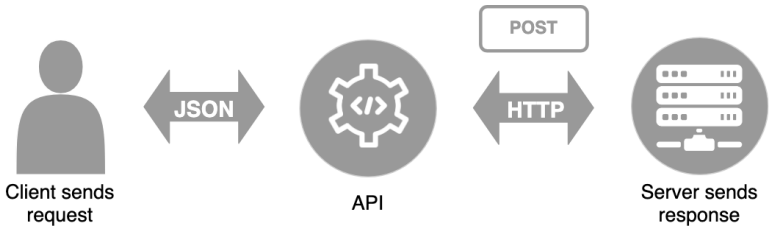


Figure 2.11: *Process overview of a Representational State Transfer application*

In the example of an automatic redaction model made accessible to third-party redaction software, the redaction software may send a request to the web-based server containing the model. The request may be as simple as a JSON dictionary with only a specific ID representing the image in an online database. The REST-API then passes the ID onto the machine learning framework on the server, which uses the ID to extract the image data from the online database. A prediction is performed on the image data and returned to the redaction software as a separate JSON response.

Container-based deployment

For a machine learning model to be available through a REST-API, the model must be hosted on a web-based server. However, different models and frameworks require unique software and hardware combinations to function properly, which can be achieved through *container-based deployment.* A container is similar to a virtual machine, which in turn is a computer file that behaves like a stand-alone computer in an isolated runtime environment, with its own file system, CPU, memory, and process space.

Deploying a machine learning model using a container is achieved by the following steps: The machine learning system and web service (REST-API) are installed inside a container, which has specific characteristics that match the machine learning system's criteria. Then a container-orchestration system is used to run the containers on a cluster of physical or virtual servers. The container-orchestration system allows the server to activate several copies of the container image when the service demand increases, each with its own GPU and other resources. This makes the

system flexible and efficient for production purposes.

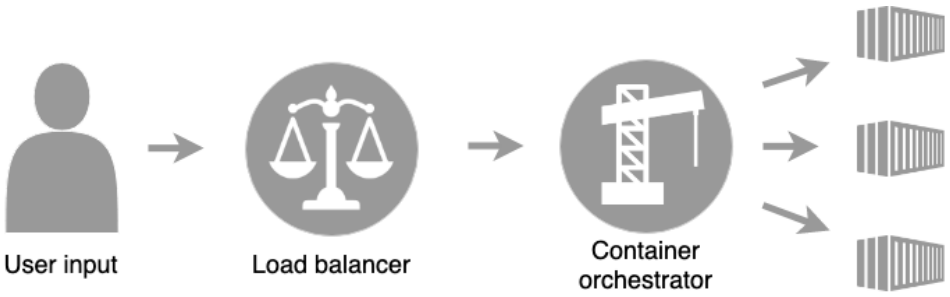


Figure 2.12: *Process overview of how a model can be deployed using containers*

Docker

Docker is an open-source platform used to automate the deployment of applications in containers [43]. As described above, containers reduce the potential friction between the development environment and the environment to which the application is being deployed to. When an application is virtualized using Docker, it can easily be accessed anywhere without making alterations to the software. This process can be described as *dockerizing* an application.

Canary deployment

Canary deployment is the practice of releasing a new version/update of software to a small group of users for initial testing [44]. In the case that one has a functioning machine learning model that is making predictions for users through your API, and you want to release a newly trained version of the model, canary deployment means releasing the new model only to a subgroup of users and comparing results for both models.

This helps identify bugs or errors in the model and eliminates downtime connected with updates. If the newly trained model does not perform better (by the selected performance metrics) in practice than the previous models, it would not be deployed at this stage at all. Comparison of all deployed models is done through a feedback system where the "correct" data is returned to the server via the API (see Section 6.1.3 on real-time performance monitoring).

Load- and performance-testing with Locust.io

Locust is an open-source Python-based tool for evaluating the loading of an application in use [45]. By generating artificial traffic, Locust simulates the usage in the environment that the application is being deployed to. This enables the developer to test how the application performs with many users before it is launched. As a result, the developer can fix errors that might occur with user traffic before the application is deployed.

3. Materials

This chapter describes the materials provided by Arkivverket for the project. There are two main materials, the provided dataset (document images) and the existing solution for identifying national identity numbers (codebase).

3.1 About the data

The corpus of documents, used for testing an approach for redaction using object detection in this thesis, consists of 20 000 compressed images. The images show scanned, historical documents from Arkivverket.

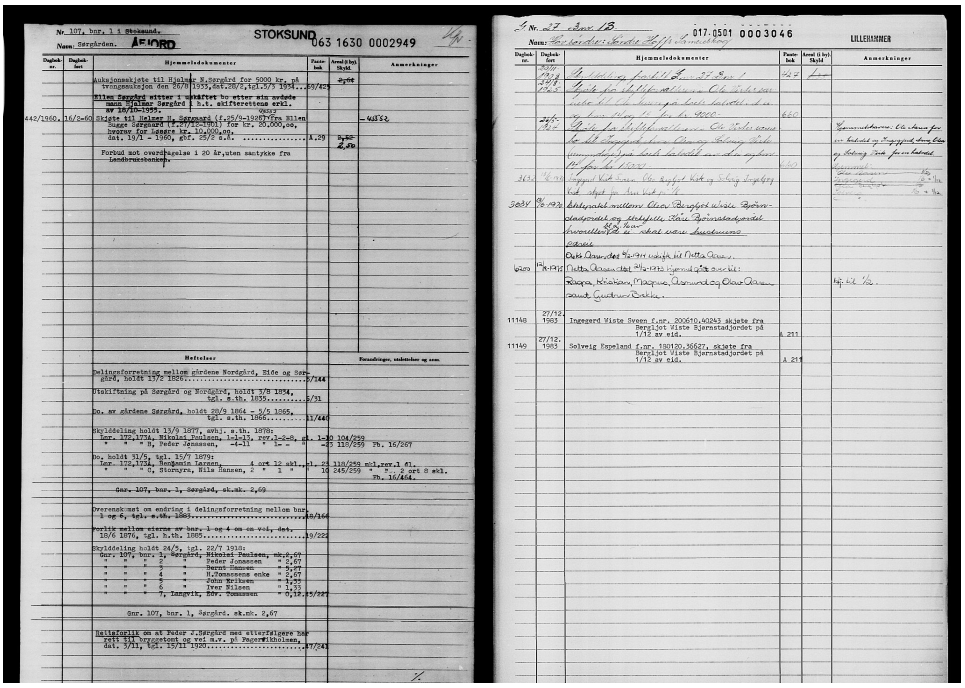


Figure 3.1: *Examples of the historical documents making up the dataset of this thesis.*

The documents are made up of handwritten and machine-written text. The two types of text are often interchanged in the same document, and text is frequently unstructured, i.e. not tabular. The images are, in general, of high perceived quality with a resolution of 72 pixels per inch. Most of the images are computer-scanned documents, with a few exceptions for documents imaged using a camera device.

3.1.1 Size and formatting

Figure 3.2 shows the distribution of image shapes in the dataset. As illustrated, width values are highly continuous between 900 and 2200 pixels, while the height values are grouped in four main values.

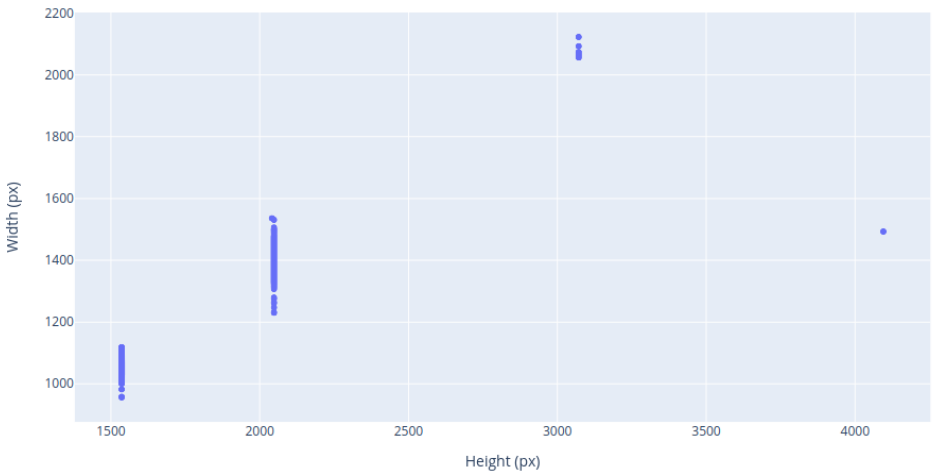


Figure 3.2: *Distribution of image dimensions by width and height in pixels. Width values are continuous, whereas height values are represented by four main values.*

3.1.2 Target data and quality

Table 3.1: *Descriptive statistics of the 35 418 bounding boxes in the dataset.*

Attribute	Min	25%	75%	Max	Mean
Count of bounding boxes	1	1	2	29	1.77
Area covered by bounding boxes	0.000035	0.00059	0.001386	0.0217	0.00116
Bounding box width (px)	13	52.5	67	305	60.24
Bounding box height (px)	1	22	31	397	28.235
Bounding box W/H-ratio	0.159	1.931	2.609	28	2.29

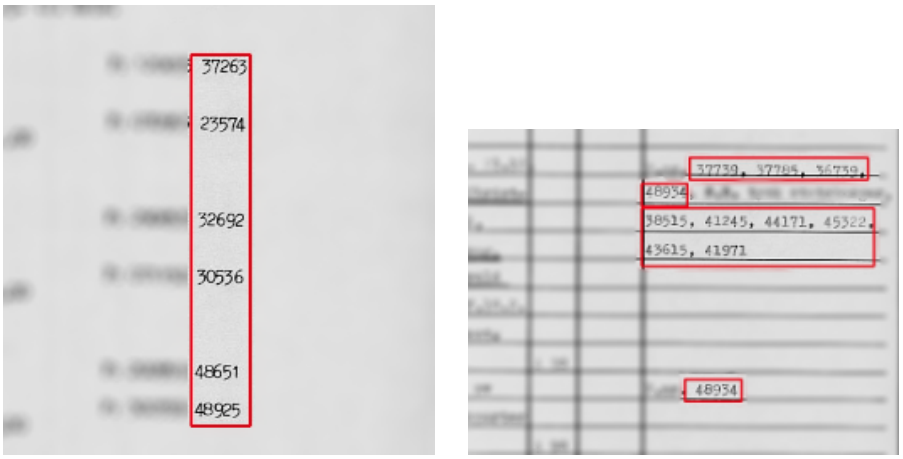
The bounding boxes are rectangles containing one or more national identity number(s). In total, the 20 000 images contain 35 418 ground truth bounding boxes. The mean count per image is 1.77, with a standard deviation of 1.45. The minimum count of bounding boxes per image is 1 and the maximum 29. Table 3.1 shows in-

dications of errors in the dataset, such as a minimum bounding box height of 1 pixel. These outliers, based on human error, are dealt with in Section 4.2.2.

Table 3.2: *The distribution of bounding box orientation*

Orientation	Count
Landscape	34 944
Portrait	474
Total	35 418

Table 3.2 gives an overview of the orientation of the bounding boxes. Out of the total 35 418 bounding boxes, only 1.35% are taller than they are wide. As illustrated in Figure 3.3a and 3.3b, the bounding boxes vary vastly in size, orientation and content. Bounding boxes containing more than one personal identification number may also impact model training and evaluation metrics. This phenomenon is described in Section 3.1.3.



(a) ***(b)***

Figure 3.3: *(a) is an example of a tall (portrait-style) bounding box. (b) displays several wide (landscape-style) bounding boxes. These examples show how bounding boxes vary in size, orientation, and content.*

3.1.3 Ground truth bounding boxes and omitted national identity numbers

Each bounding box in the dataset is hand-labeled. Differences in the shapes and forms of bounding boxes in the ground truth present a couple of challenges for the object detection model as they frequently span over several national identity num-

bers at once. In some cases, they are also drawn very widely around the national identity number. Other pages contain national identity numbers that are not labeled as ground truth.

Figure 3.4: *Example of multiple national identity numbers in one bounding box*

Multiple national identity numbers labeled in the same bounding box may interfere with the metrics at test time. If the model predicts a box for each number in the lot, these may appear as false positives in the model evaluation.

Documents, where the bounding boxes are drawn widely around the national identity number, might harm the model as the algorithm uses them to reference a national identity number's structure and shape.

Pages containing unlabeled national identity numbers will impact the evaluation metrics as each positive prediction of an unlabeled national identity number in the ground truth will be counted as a wrong prediction.

Bounding boxes provided by Arkivverket are hand-labeled by employees manually inspecting the document images through labeling software. As a consequence, the ground truth is prone to a few weaknesses and quirks:

- Employees performing manual redaction frequently encapsulate multiple national identity numbers in a single bounding box.
- A fraction of the national identity numbers remain unmarked due to oversight.

The provided dataset shows multiple examples of documents where national identity numbers are missing in the ground truth, likely due to human error labeling the dataset. The fraction of missing bounding boxes is estimated to be approximately 2.5%, see Section 5.5. Based on this assumption, the count of missing bounding boxes for this dataset (35 418 bounding boxes) is estimated at 885 *.

*Taking into account that many of the bounding boxes cover more than one national identity number, the actual number of missed national identity numbers is probably higher.

Missing bounding boxes in the ground truth causes a few challenges. The first is the ethical issue of missing national identity numbers, defeating the purpose of redacting documents in the first place. Another challenge is the effect missing national identity numbers have on the performance metrics of models, namely the precision score. Assuming a model is quite proficient at identifying the correct targets in documents, missing national identity numbers will erroneously return a lower precision score to the model. This is a result of correctly predicted samples being classified as false positives.

Even though artificially low precision scores are confined to model training and comparison and do not matter when the model is in production, missing national identity numbers still affect the model's ability to learn the dataset's optimal features. This is discussed further in Section 5.6.1.

3.1.4 A note on relabelling the dataset as part of this thesis project

Although relabeling the dataset would probably lead to better model results, both in terms of metrics and real-world performance, this task is outside the scope of this thesis due to time constraints. The least consequential of the flaws in the results are to some degree due to missing identity numbers in the ground truth, meaning that the model might make correct predictions that are counted as a false positive in the metrics and thus return an overly pessimistic precision score. As the identification and redaction of identity numbers in real-world cases are more important than the metrics presented here, time is rather spent on the development of the recommendations in Chapter 6. Following these, the dataset may be updated continuously as the application is used.

3.2 Current solution

3.2.1 Overview

Arkivverket has already developed a prototype machine learning model for automating redaction of national identification numbers from documents. Their approach involves basic preprocessing of the images, Optical Character Recognition to extract the text, and a prediction pipeline[†] applied to the extracted text. However, the model is not deployed to production due to not reaching satisfactory performance.

[†]A prediction pipeline in this context refers to a series of separate steps that include manipulating the data and results in predicting whether a word is national identity number or not.

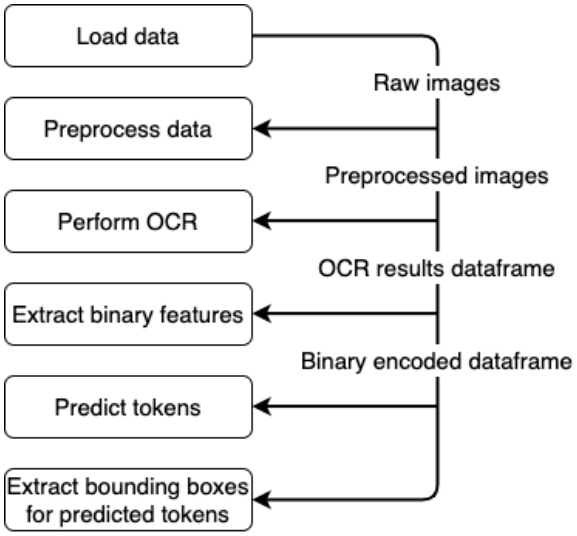


Figure 3.5: *Process overview of the prediction method for the existing solution.*

The following chapters will describe each part of the current solution in detail.

3.2.2 Preprocessing

Arkivverket has implemented 7 main pre-processing steps, inserted into the pre-processing pipeline at training time:

- **Align**: Correcting skew in the image
- **Line removal**: Identifying and removing vertical and horizontal lines in the image
- **Resize**: Increasing or decreasing the proportional dimensions of the image
- **Equalize histogram**: Enhance contrast and the distribution of black and white
- **Denoise**: Remove particles and other noise in the image
- **Sharpen**: Increase contrast between objects in the image and the background
- **Threshold**: Increasing contrast in the image by setting pixel values to 0 or 255 (white or black) based on a given threshold

3.2.3 Extracting text (Optical Character Recognition)

Tesseract is an open source Optical Character Recognition engine sponsored by Google since 2006 [46]. The Python wrapper for Tesseract, Pytesseract, is the package used in the current pipeline for text extraction in historical documents. The Pytesseract package will iterate over any identified text lines in the given document

and output a dataframe containing each identified word and its location, amongst other details.

3.2.4 Feature engineering

Once Pytesseract has created a dataframe containing the identified tokens (words) in the document, the process starts to engineer relevant features. The current implementation offers the following options:

- Analyse the formatting of each word (Boolean): Whether or not the format matches the common format for national identification numbers, e.g., XXXXXX.XXXXX or XXXXX (where X is an arbitrary numerical digit)
- Analyse the prefixes to each word (Boolean): Whether or not the n words preceding the current contains known prefixes to national identification numbers, e.g. 'pnr', 'personnr', 'fnr', 'fødselsnummer' etc.
- Check if a word is a valid birth date (Boolean): Whether or not the formatting of a word matches the valid number format of a birth date, e.g. [dd.mm.yy]
- Check if a word matches common first or last names (Boolean): Match word against a registry of common Norwegian names.

Once the selected Boolean features are generated from the OCR results, the data frame containing the features is binarized to contain 0 or 1.

3.2.5 Prediction pipeline

Bounding boxes for suggested (predicted) national identification numbers in a document are generated by creating an ensemble of classification models and training the ensemble on a portion of the data generated through feature engineering in the previous step. The model's tokens identified as national identification numbers are then checked against the original OCR results to extract the word's location (bounding box).

Classification algorithms that have been tested in the current pipeline include:

- Stochastic Gradient Descent Classifiers
- Logistic Regression Classier
- Gaussian Process Classifier
- K-Neighbours Classifier
- Multilayer Perceptron Classifier
- Support Vector Classifier
- Decision Tree Classifier
- Random Forest Classifier

- AdaBoost Classifier
- Gradient Boosting Classifier
- XGBoost

Documentation for each classifier is available on the Scikit-learn website, except for XGBoost (found at xgboost.ai).

The best results to date are achieved using a combination of three XGBoost models and a *shallow meta-prediction model*. A shallow meta-prediction model is a simple machine learning model that takes the output from each predictive model in the model ensemble and learns to combine these into a single prediction by weighting the inputs [47].

3.2.6 Baseline performance

The current solution is performing quite well on the constructed dataset of binary features, achieving a recall-score of **89.0 %**, the fraction of relevant instances retrieved), and a precision of 88.3%, the fraction of relevant instances among retrieved instances. Also, it achieved an F1-score of 88.6%, a combination of recall and precision describing the extent to which false positives are preferred over false negatives (see Section 4.5.5 on metrics explanation). However, the model's apparent limitation is that the model depends on the OCR engine's ability to correctly extract tokens from the corpus, limiting its ability to increase the total recall score.

4. Methods

The methodology used in this study is made up of four phases: Data selection, data preparation, model training and selection, and evaluation. The data selection phase involves splitting the dataset into training- and test sets used for measuring performance and comparing models. The data preparation (or preprocessing) phase includes identifying and removing outlier values and formatting the target data to the format required by the individual object detection frameworks. Other parts of data preparation are image augmentations and other methods that are applied to images before training. Finally, the training process involves model comparison, selection, and optimization. A few test-time prediction techniques were tested in the model selection phase, namely model-stacking and test-time-augmentation of images, see Section 4.6.

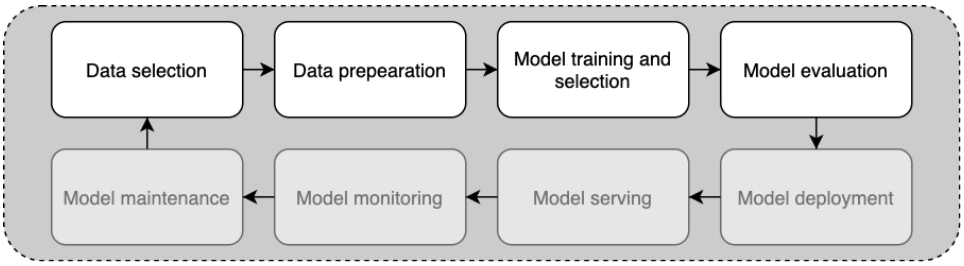


Figure 4.1: *The model training and selection phase of a machine learning project life cycle (highlighted).*

4.1 Software & hardware

4.1.1 Main Python packages

The software used in this study is Python 3.8.5, with the following main packages:

- **Detectron2 (v0.3):** An object detection framework containing pretrained object detection and segmentation models implemented in Pytorch [35]. The Detectron2 package provides the architectures and pretrained models used for all object-detection tasks in this thesis.
- **Pytorch (v1.7):** An open-source machine learning framework based on the Torch library, used for applications such as computer vision and natural language processing [48]. Pytorch is not used directly in this thesis but is the framework on which Detectron2 is built.
- **Torchvision (v0.8.1):** Library built on PyTorch, containing popular datasets,

model architectures, and common image transformations for computer vision [49]. The architectures used in this thesis through the Detectron2 framework are built on the Torchvision extension of PyTorch.

- **scikit-learn (v0.24.0):** Open source machine learning library that contains various popular (non-neural network) features and algorithms for Python [50]. This thesis implements several tools from the scikit-learn library, such as performance metrics and train-test-split functions.
- **OpenCV (v4.4.0.46):** Library of programming functions mainly aimed at computer vision and image processing [51]. In this thesis, OpenCV is used to manipulate and process the document images that are fed to the Detectron2 models.
- **Streamlit (v0.72.0):** Open source app framework, aimed at simplifying prototyping and demonstration of machine learning apps and models [52]. Streamlit has been used in this project to create a minimalistic prototype of a redaction software where one can load images, see predicted suggestions for national identity numbers and correct suggestions by adding or removing bounding boxes.

4.1.2 Hardware specifications

Computations are performed on hardware provided by Arkivverket. Any training times or absolute performance measures provided in the project are based on this hardware with the following specifications:

- **Operating System:** Ubuntu 20.04.1 LTS
- **OS Type**: 64-bit
- **Memory**: 64 GiB
- **Processor**: Intel Xeon (R) W-2123 CPU @ 3.60GHz x 8
- **GPU**: NVIDIA Quadro RTX 4000 (8GiB RAM) with the **following software**:

  NVIDIA driver v455.45.01

  CUDA Toolkit v11.1

4.2 Data selection and validation

As described in Section 2.2.2 on processing datasets, creating training- and test-sets is essential for evaluating a model and ensuring it is well suited to tackle new and unseen problems. In this thesis, a slightly different approach is used as an alternative to the standard train-validation-test split.

For the training and comparison of potential best-model candidates, a single training- and test-split is performed. The images in each split are *siloed* in separate train-,

and test-folders to prevent data leakage.

4.2.1 Training- and test-splits based on custom Phi Φ Value

The dataset is split into train- and test-splits by stratifying images by a custom Phi (Φ) value. The stratification of the splits ensures that the images in both the training- and test sets have similar characteristics regarding the target objects, i.e. bounding boxes drawn around national identity numbers.

Custom Phi Value (Φ)

By identifying the fraction of each image's area covered by ground-truth bounding boxes, it is possible to stratify the train/test split and thus maintain a similar distribution of bounding box size and count for both training- and test images. This, in turn, improves the quality of model comparisons and returns more realistic measurements of model performance. It is also possible to use the Φ-values to identify outliers in the dataset with regards to bounding-box sizes, and thus remove some of the samples where employees have covered multiple targets in a single box, simplifying the training data for the model and potentially improving model performance for real-world samples (see Section 4.2.2).

The dataset's distribution is measured in two ways: Counting the number of bounding boxes in the image, and finding the area covered by bounding boxes in each image as a ratio. The latter is necessary because several ground-truth bounding boxes cover several (sometimes more than ten) individual national identification numbers. By implementing knowledge about these traits during the splitting of the datasets, it is possible to do the splits as similar as possible - both regarding the number of bounding boxes and size of the boxes.

Stratification is performed using a custom ratio value for each document, i.e., the ratio between the area covered by bounding boxes and the number of bounding boxes. This value is referred to as the Φ-value of the document. Mathematical notation for the Φ-value of a document i is as follows:

$$\Phi_i = 1000 \times \frac{A_i}{N_i} \tag{4.1}$$

where

$$A_i = \textit{Fraction of document } i \textit{ area covered in bounding boxes}$$

and

$$N_i = \textit{Number of bounding boxes in document } i$$

The value is multiplied by 1000 to move the Φ-values closer to a more readable range of approximately 0.1 - 10. Figures 4.3 and 4.4 shows examples of documents with the corresponding Φ-values.

Calculating the custom Φ-value seems to be having the desired effect on the dataset by identifying documents containing bounding boxes with unnaturally large or small bounding boxes. Drawing random samples from documents below the lower outlier threshold and above the upper threshold (see Figure 4.7) shows that these samples do indeed contain undesirable targets when training a model to perform the main objective of the thesis. Naturally, even after outlier-removal, there still exists bounding boxes in the dataset tjat are too small for the target national identity number, as well as bounding boxes covering several targets in a single box. However, by eliminating extremes on either side of the normal distribution of values, at least a portion of the "tainted" targets are removed, allowing for more realistic results and a better model. The method for choosing thresholds and the threshold values are described Section 4.2.2.

4.2.2 Outlier Removal

Once the Φ-value of each image is calculated, images with abnormally high or low values should be removed from the dataset.

For many machine-learning projects, outlier-removal is applied only to the training-split of the dataset to ensure *generalization* of the trained model (see Section 2.2.2). However, the test-split is not subject to the method because outliers are a natural part of real data samples, and removing them would result in an unrealistic performance measure. For this particular project, however, outliers are removed from all images in the dataset - even the test split. The reason is that the custom Φ-value is, in essence, a measure of the *size and shape* of target bounding boxes in images. Thus, extreme Φ-values are indicators of human error in the labeling of the dataset. Therefore, removing outliers from the entire dataset is a way of presenting *more* realistic measures of models' performance. 800 samples were removed, representing 4% of the full dataset (see Figure 4.7).

The Φ-value is calculated for every sample in the total dataset. Table 4.1 and Figure 4.2 show the descriptive statistics of Φ-values for all samples in the dataset.

Table 4.1: *Descriptive statistics of Φ-values for all samples in the dataset*

Attribute	Min	25%	75%	Max	Mean	Std
Count of bounding boxes	1	1	2	29	1.771	1.455
Area covered by bounding boxes	0.000035	0.00060	0.001386	0.02168	0.0012	0.0010
Φ-value	0.035	0.504	0.770	15.091	0.684	0.344

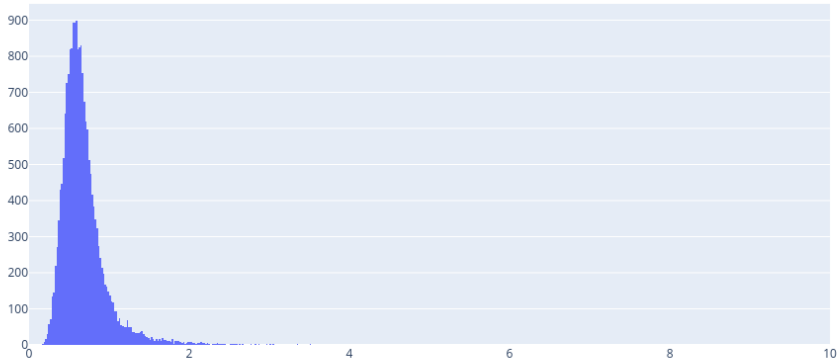
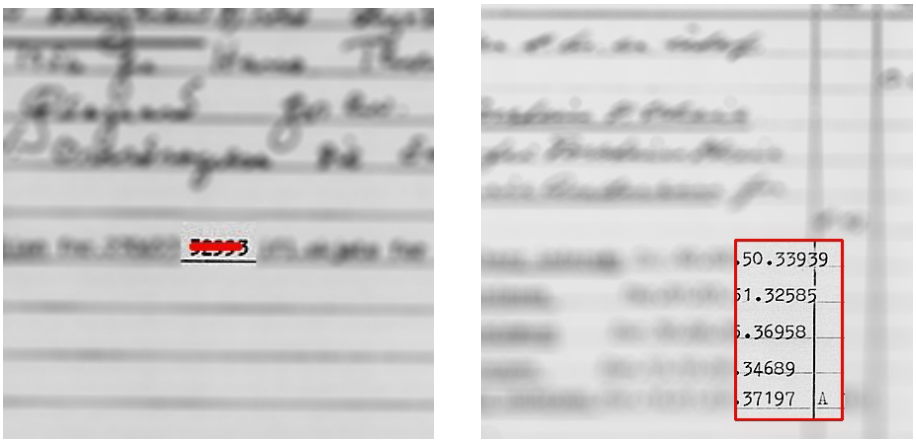


Figure 4.2: *Original distribution of Φ-values. The X-axis is clipped to the [0, 10] range, but the highest Φ-value is 15.091.*

Note that the baseline model results, as described in Section 3.2, are not subject to the outlier-removal performed in this thesis. This may lead to slightly pessimistic results for the baseline, but this is preferred over less realistic performance measures for the models developed in this thesis.

Figures 4.3a and 4.3b illustrate examples of documents with extreme Φ-values that were removed from the dataset. For reference, Figure 4.4 shows an example document with a Φ-value close to the mean of the dataset. Each figure is presented with the corresponding z-score, reflecting how by how many standard deviations the Φ-value of each document deviates from the mean of the dataset.



(a) *Φ-value = 0.035, z-score = -1.879.* ***(b)*** *Φ-value = 6.821, z-score = 17.760.*

Figure 4.3: *Examples of documents with extreme Φ-values that were removed from the dataset, with their corresponding z-scores.*

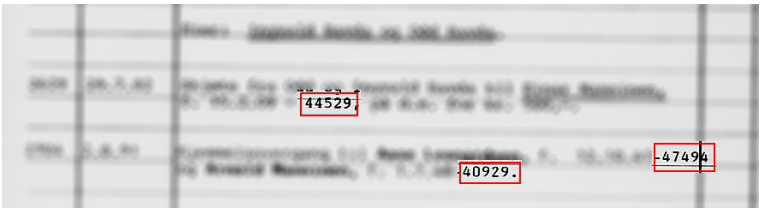


Figure 4.4: *Example of Φ-value close to the mean of the corpus, containing ideal bounding box shapes (Φ = 0.682) with a z-score of -0.00637.*

As illustrated in Figure 4.7, samples with extremely low or high Φ-values are removed from the dataset as they are indicators of human error and, therefore, may negatively impact the real-world performance of the model.

To define what samples are counted as outliers, samples with various Φ-values have been explored manually. As the Φ-value is a custom attribute made for this task specifically, a manual look at the documents helps identify what Φ-values are likely to affect the training and inference on the dataset without removing too many samples.

Document samples from the 1-, 2-, and 3-percent highest and lowest Φ-values were explored. For the lower limit, samples below the 1%-threshold showed several samples with bounding boxes that may have a negative impact on the model. Figure 4.5 shows an example document with a Φ-value of 0.276. This document image, which is just below the lower 1-percent boundary, represents many of the documents below this boundary in that the bounding boxes are cutting into several

of the numbers. This may give the model an incomplete representation of the shape and format of the personal identity numbers. Images between the lower 1-percent and lower 3-percent boundary have quite concise bounding boxes, and thus the 1-percent lower threshold is chosen as an outlier boundary.

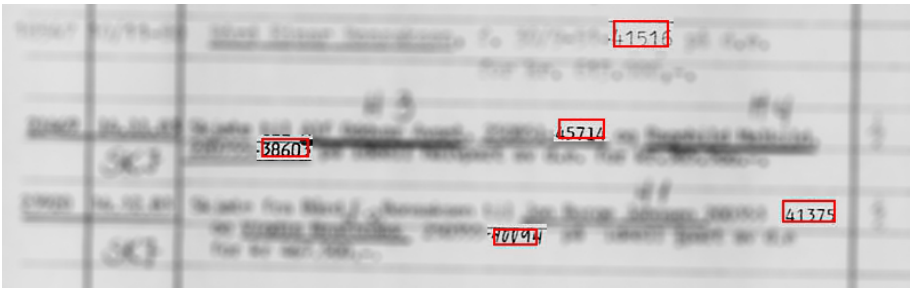


Figure 4.5: *Example document where the Φ-value is just below the 1-percent threshold (Φ = 0.276), and bounding boxes are cutting into personal identity numbers.*

Figure 4.6 shows an example document with a Φ-value of 1.40, just above the upper outlier threshold. Several of the documents at this Φ-threshold and above contain bounding boxes that span more than one personal identity number, or worse (as seen in this example). They cover skewed personal identity numbers and thus include symbols and numbers that are not representative of the target. Thus, the upper outlier threshold is set to 3%. This limit is quite restrictive, and some of the removed images probably contain information that could be valuable to the model. However, due to a large number of images in the dataset compared to the number of images removed, a more restrictive approach is chosen.

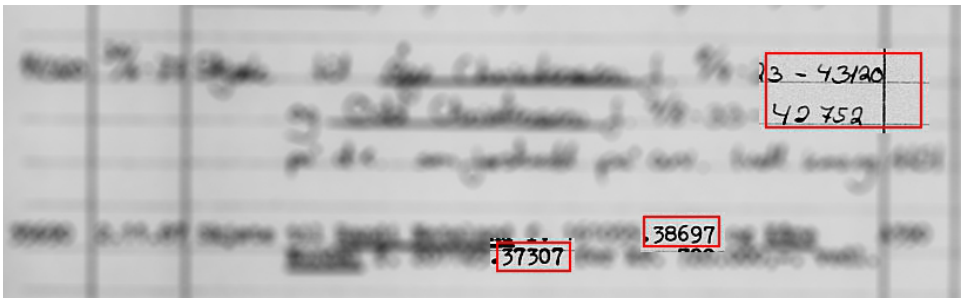


Figure 4.6: *Example document where the Φ-value is just above the 3-percent upper threshold (Φ = 0.276) and the upper right bounding box includes symbols and numbers that do not appear below in the ground truth.*

As a result of this exploration, the outlier-thresholds are set to exclude the **1%** lowest Φ-values and the **3%** highest values. The number of samples outside the thresholds is 800, leaving a total sample size of **19 200** images before performing the stratified train- and test split.

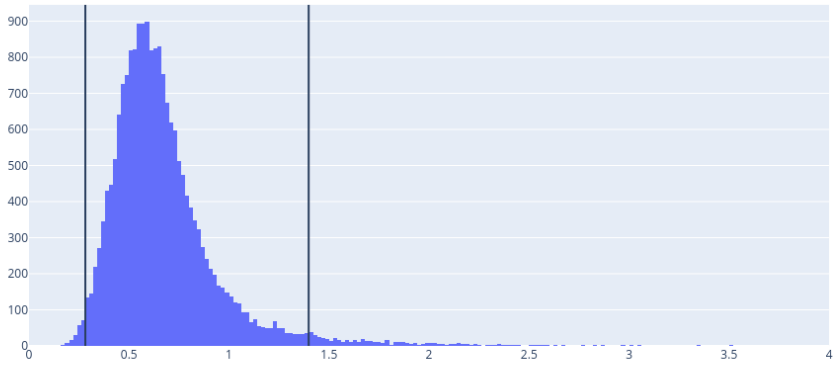


Figure 4.7: *Quantile-thresholds for outlier-removal. The left threshold represents the 1% lowest Φ-values, and the right threshold represents the 3% highest.*

Table 4.2 shows the descriptive statistics of target characteristics after removing outliers by Φ-values.

Table 4.2: *Descriptive statistics of parameters based on bounding boxes across all images in the dataset, after removing outliers based on Φ-values.*

Attribute	**Min**	**75%**	**Max**	**Mean**	**Std**
Count of bounding boxes	1	2	29	1.785	1.466
Area covered by bounding boxes	0.000282	0.001386	0.021683	0.001126	0.000946
Φ-value	0.282	0.753	1.399	0.648	12.087

Note: In Appendix A, all samples that are considered outliers and removed are predicted by a fully trained model. Results show that these samples do indeed have ground truths that are sufficiently dissimilar to most of the data to reduce the model's accuracy.

4.2.3 Quantile binning of samples by Φ-value

To stratify dataset splits, *binning* is performed on images based on the Φ-values. Binning, in this case, *quantile binning*, is a method for splitting up a continuous range of values (in this case, Φ-values) into discrete bins. Using quantile binning specifically, the dataset is split into n bins with an equal number of samples in each bin.

The number of bins chosen in this case is **4**. This number of bins was chosen by incrementally increasing the bin count until an even distribution of Φ-values across the train- and test-sets were achieved, as illustrated by mean and standard deviation values in Table 4.4. With a bin-count of 4, the dataset is split into quantiles of 25% each based on Φ-values.

To divide the data into training- and test-sets that are stratified by Φ-values, the dataset is split into quantiles based on these values. Table 4.3 shows the four quantiles' limits with a bin count of four, each containing 25% of the data samples (4800 samples for each bin).

Table 4.3: *Quantile limits for the 4-bin quantile binning of the dataset.*

Quantile	Lower limit	Upper limit
Q1	0.281	0.504
Q2	0.504	0.616
Q3	0.616	0.753
Q4	0.753	1.398

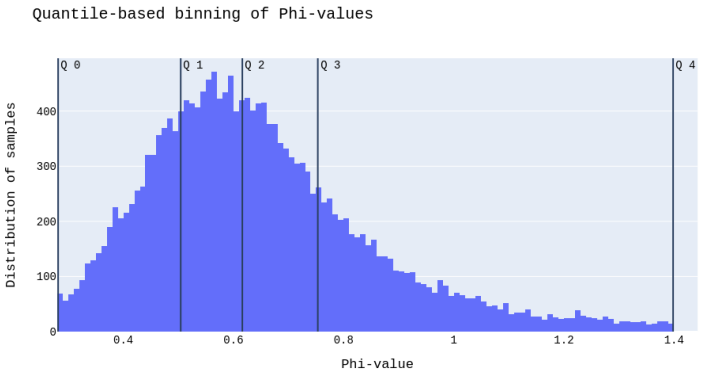


Figure 4.8: *Quantile limits used for binning document-samples by Φ-values, to ensure an equal distribution Φ-values across training- and test-sets.*

4.2.4 Train/test-split

The dataset is split into a training set and a test set to compare competing models on the same samples. Table 4.4 shows the characteristics of the splits.

Table 4.4: *Characteristics of main data-splits, showing the results of stratification based on 4-bin quantile binning of samples based on Φ-values.*

Split	Full dataset	Train split	Test split
Image count	19 200	17 280	1 920
Image fraction	100%	90%	10%
Φ-value mean	**0.648**	**0.648**	**0.645**
Mean number of bounding-boxes per image	1.784	1.778	1.844
Std.dev. number of bounding-boxes per image	1.466	1.458	1.532

The practice of binning samples by Φ-values and performing the stratification (as described in Section 4.2.3) seems to be a sufficiently effective way of maintaining similar characteristics across data splits, allowing for more realistic benchmarking of models. Table 4.4 shows distributions after performing the stratified splits.

4.3 Data preparation

As the dataset in this study consists of mainly machine-scanned documents, they are already of high quality and quite uniform in shape and color. This study's preprocessing steps mainly involved attempting to perform image cropping/warping on documents to remove tilting and unnecessary black edges around documents. However, early results showed that the model proved to perform equally well on the original images, even if the document is tilted a few degrees. In some cases, the performance was actually worse due to the added complexity of warping images using transformation matrices, resulting in errors in bounding-box coordinates. Due to time constraints, it was therefore decided to use the raw document-images without any preprocessing steps.

The content of the images are generally made up of 3 components:

- Text
- Lines and shapes
- Background

Text in the given documents varies from hand-written to machine-written. The text size is generally small to medium, with few documents containing large titles or bits of text.

The number of lines and shapes in the corpus vary from image to image, with some documents containing only text on a clear background, while other documents are made up of a tabular grid of lines and boxes.

The backgrounds of the document images are usually fairly bright in color, and the color varies from yellowish to white.

4.3.1 Data generation and augmentation

To improve the versatility of trained models presented in this report, several random augmentation techniques are applied to images during training. Generating more data that is slightly different from the original dataset may improve the models' ability to generalize features. Models are also trained using no augmentation to compare performance, and results are compared in Section 5.2.

Table 4.5: *Train-time augmentation steps and parameter-values used for improving generalization of trained models.*

Random Brightness	Intensity range: [0.8, 1.8]
Random Contrast	Intensity range: [0.6, 1.3]
Random Saturation	Intensity range: [0.8, 1.4]
Random Lighting	Intensity range: [0.7, 1.3]
Random Crop	Crop size: [0.9, 0.9] relative range
Resize Shortest Edge	Sample style: Choice, Max size: 1 333 px

As no general image enhancement is performed in the preprocessing phase, there are also slight variations inherent in the dataset that can affect the model performance:

- Variations in document size and format.
- Variations in document layout, e.g., presence of lines and grids.
- Variations in document hue and saturation, e.g., white documents vs. yellowish documents et cetera
- Variations in text type and size, e.g., Hand-written vs. machine-written text, and large vs. small letters and numbers
- Variations in text opacity and contrast

4.4 Calculation of cost related to redaction time

In order to recommend a model to Arkivverket based on cost-savings-potential, it is necessary to be able to calculate an estimate of redaction time for a batch of

documents based on a given recall and precision score. Equation 4.2 shows how this can be done:

$$T_{total} = \frac{T_{base} + T_{FN} + T_{FP}}{3600} \tag{4.2}$$

$$\begin{aligned}
&where \\
&\mathrm{T}_{base} = N_{doc} \times t_{doc} \\
&\mathrm{T}_{FN} = N_{doc} \times \overline{N_{bb}} \times (1 - recall) \times t_{FN} \\
&\mathrm{T}_{FP} = N_{doc} \times \overline{N_{bb}} \times (1 - precision) \times t_{FP}
\end{aligned}$$

T_{total} = Total redaction time in hours
T_{base} = Base redaction time for document without errors (seconds)
T_{FN} = Time spent correcting missed targets (seconds)
T_{FP} = Time spent deleting incorrectly predicted targets (seconds)
N_{doc} = Total document count (for calculation period)
$\overline{N_{bb}}$ = Average number of bounding boxes (targets) per document
t_{doc} = Base processing time per document (seconds)
t_{FN} = Average correction time per False Negative (seconds)
t_{FP} = Average correction time per False Positive (seconds)

Based on rough estimates from Akrivverket and through testing, assumptions for the calculation of time-cost have been made. For each *false positive* prediction, the employee must remove the suggested bounding box. This can be done with the click of a button. Rough estimates suggest that the process of evaluating and correcting these takes on average 2 seconds per false positive prediction.

False-negative predictions will not be apparent to the employee redacting. The employee must, therefore, actively search for omitted national identity numbers and mark them. A new bounding box can be placed with two clicks of a button, marking its x- and y-coordinates. The search for potential false negative predictions is time-consuming. Therefore it can be roughly estimated that a false negative has a time cost of 6 seconds per false negative prediction.

In addition, rough estimates suggest that all document pages will require a base time of approximately 12 seconds of processing time from the document is opened, even without the need for adjustments. This time is added to the time required for correcting false negatives and false positives.

These estimations are used for the calculations in Section 6.2.1 and Appendix B. In Section 5.1.5, these values are used to select a beta-value for the Custom Value Metric, a metric for measuring model performance.

4.5 Model training and selection

The Detectron2 framework offers a wide variety of pre-trained models built on different architectures.

4.5.1 Architectures - Capacity versus speed

The architectures used for training and comparing models in the project are described in Table 4.6. Benchmark results presented are from the official Detectron2 Model Zoo [35], and are based on results on the COCO-dataset (see Section 4.5.2).

Table 4.6: *Overview of available object detection architectures from the Detectron2 library. Benchmarks are based on the COCO-dataset. Training speed and inference speed are measured in seconds per iteration.*

Model	Backbone + Head	Layers	Size (MB)	Train. speed (s/it)	Infer. speed (s/it)	COCO Precision Benchmark
Faster R-CNN	ResNet + FPN	50	333	0.209	0.038	40.2
Faster R-CNN	ResNet + FPN	101	485	0.286	0.051	42.0
Faster R-CNN	ResNeXt + FPN	101	838	0.638	0.098	43.0
Mask R-CNN	ResNet + FPN	50	354	0.261	0.043	41.0
Mask R-CNN	ResNet + FPN	101	506	0.340	0.056	42.9
Mask R-CNN	ResNeXt + FPN	101	859	0.690	0.103	44.3

These six models are divided into two main architectures, Mask- and Faster R-CNN, each with two different layer depths (50 and 101 layers) and two different backbone architectures (ResNet and ResNeXt). Although the two main architectures are quite similar (see Section 2.3.6 for information on Mask R-CNN and Faster R-CNN), they perform object detection in slightly different ways. They are therefore compared in the general model selection (see Section 5.2).

The strengths and weaknesses within each main architecture are a matter of balance between model capacity and inference speed (not to mention training speed, but this is less important in a production environment). The "Inference speed" and "Training speed" columns in Table 4.6, measured in seconds per iteration (s/it), can be compared with the "Model size" and "COCO Benchmark [precision]" to measure this balance. Models with a ResNet backbone and 50 layers have the advantage of being more agile, with inference and training speeds at approximately one-third that of the models with ResNeXt backbones and 101 layers. The weakness of this smaller model is its limited capacity for learning more complex patterns in the document images.

The two models with a ResNet backbone (like that of the smaller models), but a capacity of 101 layers (like that of the largest models), serve as the more balanced models. The increased layer count increases the model capacity while maintaining an inference- and training speed at approximately half that of the largest models. This is due to the complexity in the ResNeXt backbones, which substantially

slower than the ResNet backbones.

The largest models, made up of 101 layers and ResNeXt backbones, sacrifice speed for capacity. These models can recognize complex patterns in the dataset but need significantly more epochs to converge on the dataset, combined with a slower training time per epoch.

4.5.2 Transfer learning

Common Objects in Context (COCO) Dataset

All models tested are pre-trained on the Common Objects in Context (COCO) dataset. The COCO dataset represents both a large database of everyday objects and a specific way of formatting the dataset.

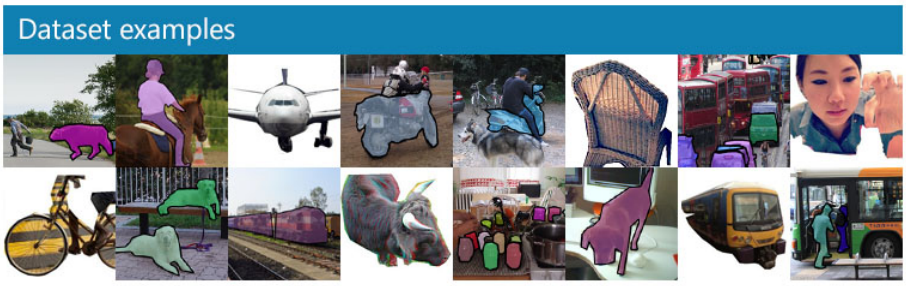


Figure 4.9: *Sample images from the COCO-dataset, with overlaying object masks. The dataset also contains rectangular bounding boxes. [53]*

Even though images in the COCO dataset are all objects depicted in a real-world setting that may seem to differ significantly from the documents predicted in this project, pre-training models on this dataset "teach" the model to recognize useful low-level features in document images. Examples of these features are edges, shapes, areas of interest in the image, and more.

COCO-format

When fine-tuning models pre-trained on the COCO dataset, it is required to format the dataset used for fine-tuning to a specific standard. The format is JSON-based and provides the framework with an overview of training- and test files, their IDs and locations, and coordinates of ground-truth bounding boxes. The COCO-format JSON file is built by creating a list of dictionaries, as illustrated in Figure 4.10. Each dictionary corresponds to a single image in the dataset.

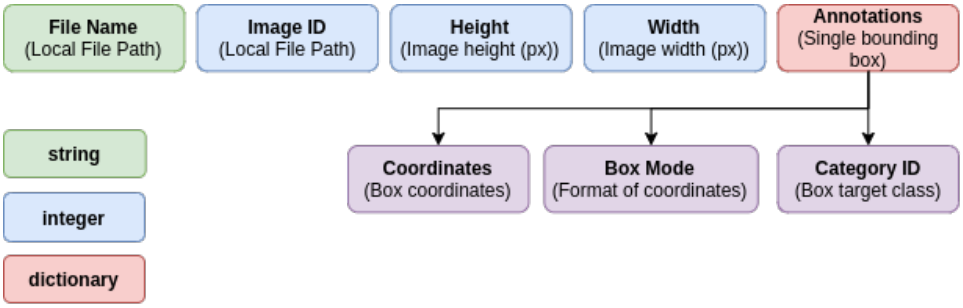


Figure 4.10: *Overview of the COCO-format JSON file.*

4.5.3 Mini-batch learning and epochs

The resolution of the files provided - and the fact that all training has to be performed on a single GPU - sets limitations on the batch sizes used in training models. Batch size is a parameter in the training-dataset generator class of the Detectron2 framework and is passed the following values in this project:

Table 4.7: *Model layers by batch size*

Number of layers in model	Maximum batch size
50 layers	4
101 layers (ResNet)	2
101 layers (ResNeXt)	1

In the Detectron2 framework, *iterations* are used instead of setting a number of epochs. Completing an iteration means iterating over a batch of images, in this case, 1, 2, and 4. To understand how iterations translate to the standard epochs, meaning a full iteration of the training set of images, the following formula can be used:

$$\textbf{Epochs (E)} = \frac{\text{Iterations (I)} \times \text{Batch size (B)}}{\text{Number of training images (N)}} \tag{4.3}$$

In the case of comparing models with the same numbers of epochs (as in model selection), it is needed to calculate the correct number of iterations based on batch size (which is a result of model size). Say you want to train a model for 15 epochs, have 17 280 training samples and a batch size of 2. The number of iterations, i.e. 129 600, can be found by altering the formula:

$$\mathbf{I} = \frac{\mathrm{E} \times \mathrm{N}}{\mathrm{B}} \Rightarrow \mathbf{I} = \frac{15 \times 17\,280}{2} \Rightarrow \mathbf{I} = 129\,600 \tag{4.4}$$

4.5.4 Learning rate scheduler

Two main learning rate schedulers are implemented in the Detectron2 framework, the *Cosine LR Scheduler*, and the *Multi-Step LR Scheduler*. The former sets the learning rate automatically, incrementally decreasing the learning rate in a shape that resembles a cosine curve, as illustrated in Figure 4.11. The latter scheduler reduces the learning rate by a given fraction (a gamma-value) at specified intervals.

The Cosine Scheduler has the advantage that it maintains a high learning rate for a large part of the training period, making it ideal for the rough training of pre-trained models. The Multi-Step Scheduler form resembles a negative exponential curve, depending on how many times the gamma-value reduces the learning rate. This scheduler is ideal for fine-tuning or retraining models, as the learning rate decreases quickly in the training period.

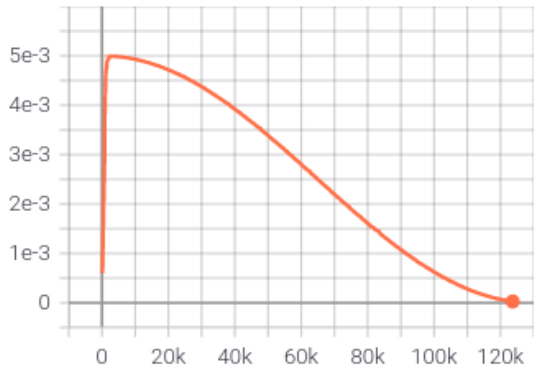


Figure 4.11: *Cosine LR curve from Section 5.1.2*

4.5.5 Metrics

IoU-threshold & Confidence threshold

Intersection over Union (IoU) describes the overlap between the ground truth bounding box and a predicted bounding box with values between 0 and 1 where a high value equals a high percentage of overlap [54]. The IoU-threshold decides to what degree a predicted bounding box and a ground-truth box must overlap to be considered a true positive.

The confidence score [55] indicates how confident the predicting model is that a predicted bounding box contains the targeted object. The confidence score of a given model is not directly comparable to that of other models but is a tool for deciding how confident the user wishes the model to be before making predictions in production.

Evaluation metrics

The performance of a machine learning model can be evaluated in several ways. However, this thesis will focus on four main metrics for the comparison of models:

- **True Positive (TP):** The model correctly identifies a target.
- **False Positive (FP):** The model predicts a target that is not in the ground truth.
- **False Negative (FN):** The model does not predict a target that is in the ground truth.
- **Recall:** The main metric for the project as avoiding false-negative classifications is the main objective. Measures the positives that are correctly identified.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{4.5}$$

- **Precision:** Measures the proportion of positive identification that is actually correct

$$\text{Precision} = \frac{TP}{TP + FP} \tag{4.6}$$

- **F1-score:** Calculated from recall and precision. It is a secondary metric to represent the trade-off between them or to what extent false positives are preferred over false negatives.

$$\text{F1-score} = \frac{2 * p * r}{p + r} = \frac{2 * TP}{2 * TP + FP + FN} \tag{4.7}$$

- **Custom Value Metric / F-beta-score:** The F-beta score takes an input parameter β that changes the relative impact that recall or precision has on the metric score.

$$\text{Custom Value Metric (CVM)} = \frac{((1 + \beta^2) \times r) \times p}{(\beta^2 \times r) + p} \tag{4.8}$$

TP = True positives
FN = False negatives
FP = False positives
r = Recall
p = Precision
β = Recall/precision cost-multiplier

The F1-score is an F-beta metric that has a β of 1, meaning it weighs recall and precision equally. When a Custom Value Metric is referred to, it is an F-beta score with a custom β-value that reflects the impact that recall has on the real cost compared to precision, hence *custom value*. The *F-beta* score is used to measure model performance in a way that is as relevant as possible for cost-saving at Arkivverket.

The chosen β-value for the Custom Value Metric in this project is based on the estimates for the cost related to perform two main actions when redacting documents:

1. Removing a falsely predicted bounding box suggestion, or
2. Searching for and drawing a new bounding box around missed national identity numbers.

A cost-specific metric can be calculated based on the ratios of false positives and false negatives produced by a model by setting a numeric cost for each of these scenarios. It is estimated that identifying and labeling a missed national identity number (false negative) takes an employee *N times longer* than removing a falsely predicted national identity number (false positive). Based on this assumption, the F-beta score is then set to reflect the relative cost of the two actions, N.

By recalculating a model's performance on a set of images in this way, the model with the highest Custom Value Metric should be a more cost-efficient model based on cost-estimations.

As described in Section 4.4 on calculating redaction time, it is assumed correcting a *false negative* takes approximately three times longer than correcting a *false positive*. These assumptions are used to calculate the ideal choice of beta-value once an object detection model is fully trained. See Section 5.1.5 for our choice of beta-value, and Section 4.5.6 for the steps used to make the choice.

4.5.6 Selecting a beta-value for the CVM

After training an object detection model, different β-values for the CVM can be compared to find the best one based on cost assumptions from Section 4.4. This can be done in the following steps:

1. Train an object detection model (see Section 5.1.2) and calculate recall- and precision-scores for a range of confidence thresholds (see Table 5.4),
2. Select a range of β-values for which to calculate total redaction time,
3. For each selected β-value, calculate the Custom Value Metric with the selected β for each combination of recall and precision (e.g., each confidence threshold) in step 1 and find the confidence that leads to the maximal Custom Value Metric score for the selected β-value,

4. Collect the optimal confidence thresholds for each β-value and run the redaction-time function (see Equation 4.2) for the recall/precision-combination corresponding to the confidence thresholds by looking up the recall- and confidence scores in step 1,

5. Finally, find the β that minimizes the redaction-time result from step 4.

Figure 5.2 illustrates this process for the fully trained model.

4.5.7 Choice of IoU-threshold

As described in Section 4.5.5, the IoU-threshold decides to what degree a predicted bounding box and a ground-truth box must overlap to be considered a true positive. It is common practice to use an IoU threshold of 0.5 as a baseline. However, as described in Section 3.1.2, the ground truth bounding boxes vary greatly in size and shape. As a result, a threshold of 0.5 will result in unrealistically pessimistic metric scores.

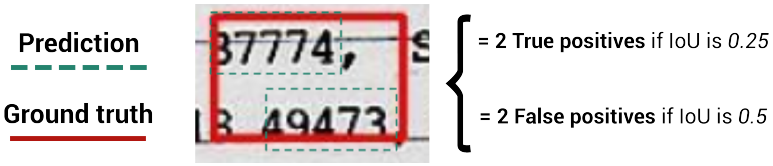


Figure 4.12: *Illustration of how different IoU thresholds impact metric scores.*

As Figure 4.12 illustrates, an IoU-threshold of 0.5 would cause two false positives, while a threshold of 0.25 would result in two true positives. An IoU-threshold of 0.25 was selected to measure performance of all architectures in this thesis, after examining prediction results for 5%-intervals between 0.05 and 0.45. Results showed that thresholds between 5% and 25% yielded similar results, with a rapid deterioration in results with thresholds above 0.25.

Section 6.1.3 about model deployment describes a feedback loop where documents that are processed by the automatic redaction model and quality-assured by an employee are automatically registered as ground truth in a new and improved dataset. As this dataset is formed, and hopefully contains more uniform bounding boxes covering single national identity numbers, it would be natural to raise the IoU-threshold towards 0.5. This would take performance measures closer to industry standards, and yield more robust measures by demanding a higher degree of overlap between the ground truth and predictions.

4.5.8 Choice of training parameters for general and extensive model training

To compare the different architectures and models in Table 4.6, each is trained for an equal number of epochs (4 epochs*) and with the same learning rate scheduler and similar hyperparameters. Models are compared on performance with regard to both prediction accuracy and inference speed. Two of the models are trained using no image augmentations to display the effect that train-time augmentations have on prediction results.

As a result of the initial comparison of models, the fastest and most accurate models are chosen for more extensive training. This phase consists of training both models for 15 epochs and then comparing prediction accuracy and inference speed.

Tables 5.1 and 5.3 list the parameters used for both the initial (general) training and the extensive training of the two best-performing models.

4.6 Ensembling / Stacking models

Ensemble learning aims to combine several object detection models and use the models' combined output to find a single document prediction.

Model stacking is a technique for removing bias in models and achieving better results. Model stacking is performed by combining (stacking) fully trained models from unique architectures into a single predictor. Even though the models are trained on the same data, there is a chance that the different anatomies of the architectures result in them learning to look for slightly different high- and low-level features to identify the targets.

After training the models, each model's predictions are combined and compared against the ground truth.

In this thesis, model stacking is used in two main ways: 1. Where multiple models make independent predictions on the *same* input image (model ensemble), and 2. Where a single model makes multiple predictions on *multiple variations* of an input image (test-time augmentation). For both methods, affirmative voting (hard voting) is used (see Section 4.6.2). Mathematically, the two methods can be described in the following ways:

Model ensembles:

$\text{models} = f_1, f_2 \dots f_n$

*4 epochs was chosen for the initial testing, rather arbitrarily, as a balance between giving the models enough time to converge enough for comparison, while leaving training time low enough to perform many training-cycles without using a large amount of the project time, leaving more time for full training of the most promising models and fine-tuning.

voting function $= \sigma$
input $= x_i$
outputs $= (f_1(x_i), f_2(x_i) \dots f_n(x_i))$

Test-time augmentation (TTA):

model $= f$
voting function $= \sigma$
inputs $= x_{i_1}, x_{i_2} \dots x_i n$
outputs $= (f(x_{i_1}), f(x_{i_2}) \dots f(x_{i_n}))$

For both techniques, the prediction method is the same:

$$\hat{y}(x_i) = \sigma(outputs) \tag{4.9}$$

4.6.1 Test-time augmentation

Test-time augmentation (TTA) is a technique that can improve a model's ability to identify national identity numbers. In the case of this thesis, test-time augmentation means showing that a fully trained model 3-4 variations of a single input image at test-time, allowing the model to return unique predictions for each image variation. The multiple sets of suggestions for each input image are then processed through a voting mechanism and compressed into a single prediction (see Section 4.6.2 and 4.6.3).

4.6.2 Voting function

The voting function applied in this thesis is *affirmative voting* [56]. For affirmative voting, all suggestion made on any of the image variations is included in the prediction. This is beneficial for the recall score, as more predictions are assumed to be made on each input image. However, it may significantly affect the precision, as the model as a whole is less critical towards the required confidence for predictions.

4.6.3 Non-maximum suppression

Non-maximum suppression (NMS) is a technique used in most object detection models but can also be applied to combining models or model predictions into a single prediction. NMS iterates through the proposed bounding boxes for a single image and calculates the IoU-score for each bounding box. The purpose of this is to prevent keeping several predicted bounding boxes for a single target object.

In the case of model ensembling, the probability of predicting almost identical bounding boxes for a single target is substantial. As an example, in the case of using two separate models to make predictions on the same input image, many of the target objects will be identified by both models. NMS is then applied to

combine the most similar predictions by *keeping the bounding box with the highest confidence score.*

Since the threshold for combining predicted bounding boxes (IoU-threshold) is a pre-defined parameter, it needs to be optimized in a model ensemble. Setting a *high* IoU threshold generally leads to a higher recall score (at the expense of precision), as more predictions are kept - even if they are quite similar. However, setting a *low* threshold will increase the precision score by decreasing the frequency of superfluous bounding boxes.

5. Results and discussion

5.1 Model comparison and selection

The purpose of the general selection of models (as listed in Table 4.6) is to compare the most relevant models from the Detectron2 Model Zoo against each other on the dataset to decide which models are to be more extensively trained. Results are shown in Table 5.2.

The parameters chosen for this general comparison of models (listed in Table 5.1) are meant to clarify which models perform better while maintaining a relatively low training time for each training. Thus, more project time is spent on the final training of models.

The two most promising models are Mask R-CNN and Faster R-CNN with 101 layers each. Each is now trained for 4 epochs on the full dataset, both with and without random augmentation at train-time.

5.1.1 Choice of parameters

The model selection parameters for the initial comparison were chosen to allow the model to learn features from a wider range of document samples while still keeping the training time at a manageable level. See Section 4.5.8 for the reasoning behind the choice of 4 parameters for the initial testing.

Table 5.1: *Training parameters for extensive training of the selected small and large models.*

Parameter	Value
Train-images	17 280
Batch size	2 (1 for ResNeXt models)
Iterations	34 560
Epochs	4
Confidence Threshold	0.25
IoU Threshold	0.25
Learning Rate Scheduler	Warmup Cosine LR
Base Learning Rate	0.01
Warmup Iterations	500

Table 5.2: *General model selection across different architectures, network depths, and whether or not train-time augmentation is applied.*

Model	Backbone + Head	Layer depth	Augmentation	F1-score	Recall	Precision
Baseline	–	–	–	*0.880*	*0.890*	*0.883*
Faster R-CNN	ResNet + FPN	101	No	0.863	0.978	0.772
Mask R-CNN	ResNet + FPN	101	No	0.875	0.978	0.791
Faster R-CNN	ResNet + FPN	50	Yes	0.885	0.979	0.807
Faster R-CNN	ResNet + FPN	101	Yes	0.889	0.983	0.811
Faster R-CNN	ResNeXt + FPN	101	Yes	0.907	**0.989**	0.837
Mask R-CNN	ResNet + FPN	50	Yes	0.888	0.980	0.811
Mask R-CNN	ResNet + FPN	101	Yes	0.895	0.982	0.822
Mask R-CNN	ResNeXt + FPN	101	Yes	**0.913**	0.986	**0.850**

In the initial model comparison, the *Mask R-CNN with ResNeXt backbone* model came out on top with a recall-score of **0.986** and an F1-score score of **0.913**. The results were calculated using a prediction confidence threshold of 0.25 and the same value for the IoU-threshold. The best-performing small model (with 50 layers) was the Mask R-CNN model, with a recall score of 0.979 and an F1-score of 0.885.

Note that it can be argued that choosing a number of epochs as low as 4 may lead to an unfair comparison of models due to the fact that some models may be closer to converging than others. However, due to the large size of the dataset and the relative uniformity of the sample images, an epoch count of 4 would mean each model iterates over 80 000 images. Thus, each model should be able to reach a level of convergence that allows for comparison. Results also back this decision, as model performance seems to be highly correlated to model complexity and capacity.

Results in Table 5.2 show that using random image augmentation at training time, as described in Section 4.3.1, yields better results for both Faster R-CNN models and Mask R-CNN models.

The top-performing 101-layer-model ("large" model) and 50-layer-model ("small" model) will be trained for longer in order to reach results that are closer to the full potential of the models for the given dataset.

5.1.2 Full training of selected models

For the full training of the best model from Table 5.2, the purpose is to move closer to the full potential of the Detectron2 framework for this problem by increasing the number of epochs, allowing the model to converge.

The parameters for the full single-model training are very similar to those listed in Table 5.1. However, the epoch count is set to 15, as this allows the model to

converge to a much larger degree while keeping training time between 24 hours and 36 hours.

Table 5.3: *Training parameters for full training of the two selected Mask R-RCNN models.*

Parameter	**Value**
Large model	Mask R-CNN X101-FPN
Batch size (large model)	1
Small model	Mask R-CNN R50-FPN
Batch size (small model)	4
Train-images	17 280
Epochs	15
Learning Rate Scheduler	Warmup Multistep LR
Base Learning Rate	0.01
Warmup Iterations	500
Training augmentations	Yes

In previous model comparisons, model performance has only been measured for a confidence score of 0.25 and an IoU-threshold of 0.25. In this section, the purpose is to explore the potential of a fully trained model more thoroughly. This is achieved by calculating the model's performance over several confidence thresholds (exploring the trade-off between recall- and precision) and illustrating the drop in performance for IoU-thresholds of 0.25 and 0.50.

5.1.3 Recall/precision-tradeoff

By calculating the fully trained model's performance for each 5-percent interval, from 5% up to 95%, one can analyze how the model performs concerning the F1-, recall and precision scores over the different thresholds. This provides insight to what degree accuracy for one metric is sacrificed when tuning another.

Table 5.4: *Recall- and precision results for both fully trained models with IoU = 0.25 for various confidence threshold values.*

(a) Large model *results*

Conf. thr.	Recall	Precision
5 %	**0.993**	0.806
... %		
20 %	0.987	0.881
25 %	0.986	0.891
... %		
60 %	0.979	0.934
65 %	0.978	0.943
70 %	0.974	0.950
75 %	0.972	0.954
80 %	0.969	0.959
... %		
95 %	0.897	**0.970**

(b) Small model *results*

Conf. thr.	Recall	Precision
5 %	**0.992**	0.760
... %		
20 %	0.984	0.853
25 %	0.982	0.867
... %		
60 %	0.967	0.932
65 %	0.964	0.937
70 %	0.960	0.944
75 %	0.955	0.949
80 %	0.950	0.956
... %		
95 %	0.877	**0.970**

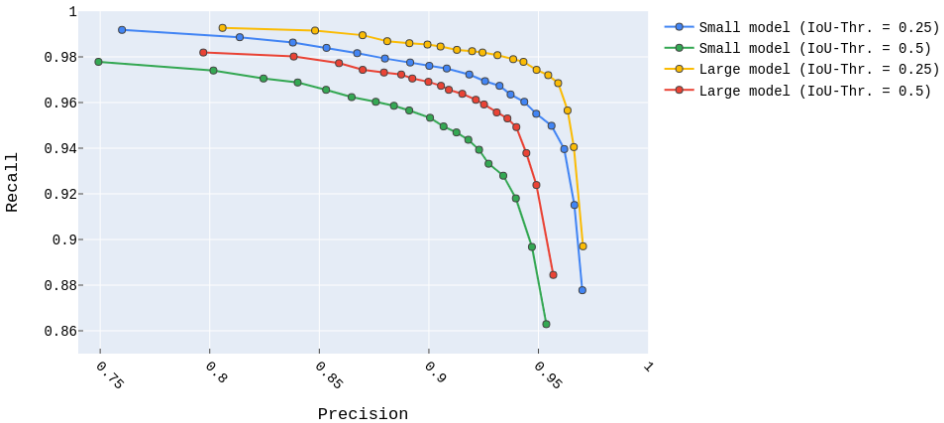


Figure 5.1: *Precision/recall-tradeoff for small and large models for IoU-threshold $\in$ (0.25, 0.50). Created by incrementally adjusting inference confidence threshold for test-set. Each dot marks a 5% confidence interval, with 5% on the far left and 95% on the far right.*

Tables 5.4a and 5.4b shows how the metrics develop over these confidence intervals. One can see that setting a very low confidence threshold, meaning almost all predictions made by the model will be regarded, a maximum recall score of **0.992** is achieved when considering an Intersection over Union-threshold of 0.25. This maximum recall score comes at a price, as precision is at *0.74* due to many false positives in the predictions.

By setting the confidence score very high, e.g., to 0.95, the precision metrics are optimized to **0.970**, with a recall of *0.875*. This indicates that the model can identify 87.5% of the national identity numbers in the dataset with more than 95% confidence.

It is also likely that the precision score for this latter confidence threshold is reaching a *hard limit* due to the number of missing national identification numbers in the ground-truth (Section 3.1.3), marking several predictions above 95% confidence as false positives though they are actually true.

Figure 5.1 shows the performance for both metrics for each confidence interval to get a visual impression of the trade-off between recall and precision. The figure shows that even though recall is the main metric, it is possible to gain a large increase in precision by sacrificing only a small part of the recall performance.

5.1.4 Different IoU-thresholds

As Figure 5.1 shows, the model's performance drops 1-2 percent overall as the IoU-threshold for calculating metrics doubles from 0.25 to 0.50. This result indicates that the model is well adapted to the ground truth, even if some predictions fall outside the threshold at 0.50.

5.1.5 Choice of beta-value for the Custom Value Metric

The purpose of using a Custom Value Metric instead of a standard F1-score is to compare model performance at different confidence intervals with actual redaction-time in mind.

Based on the assumptions presented in 4.4 on calculating redaction cost, it is possible to use the recall/precision tradeoff scores for the large model in Table 5.4 to find a beta-value for the CVM that best represents the time-cost savings for each confidence interval. Figure 5.2 illustrates the total redaction time of 10 000 documents for each beta-value in the range [0.1, 4]. For beta-values below 1 precision is weighted more than recall, and for beta-values above 1 recall is preferred. Although optimizing the beta-value leads to a reduction in redaction time, this reduction is limited to a span of approximately 2 hours. The estimated time for redacting 10 000 images *without* the assistance of machine learning is approximately 63 hours, meaning the optimal beta-value is a 45.1% decrease in redaction time compared to the manual option (See Section 6.2.1 on calculating manual redaction times and economic benefits).

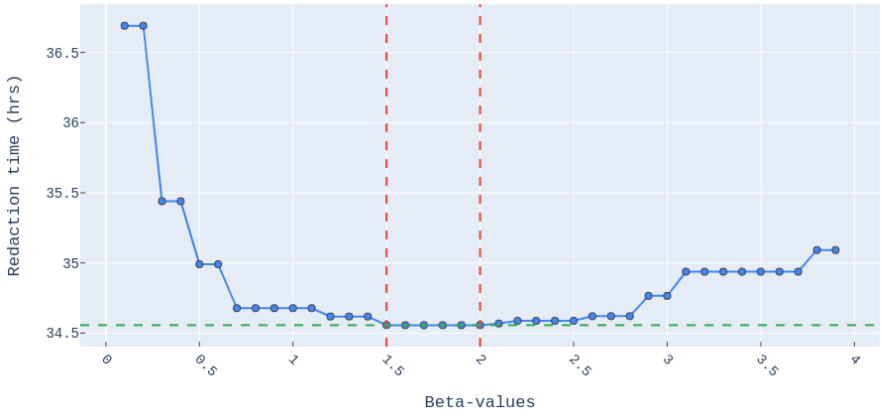
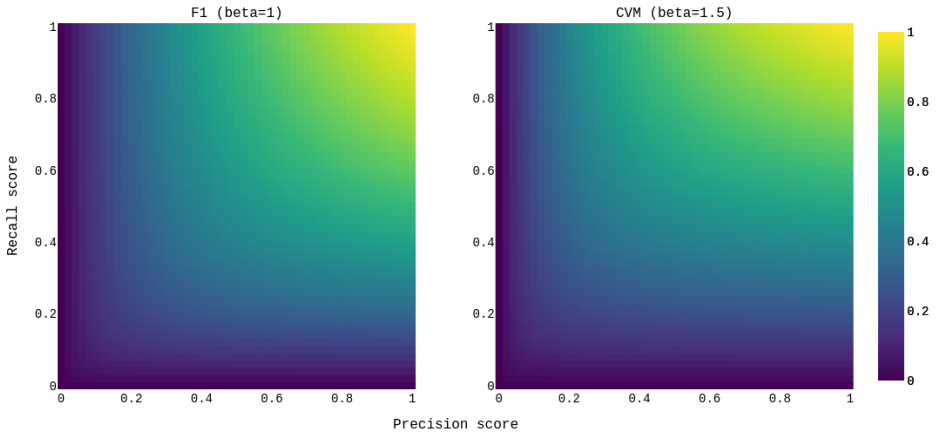


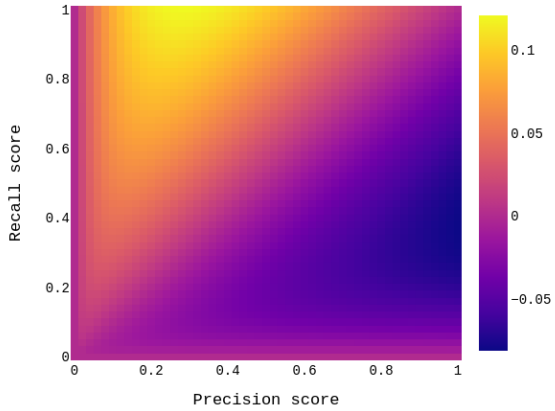
Figure 5.2: *Total estimated redaction time (hrs) for 10 000 documents by CVM-beta-value. Minimum value on the y-axis: 34.56 hrs. For comparison, the estimated time for fully redacting the same number of documents is 63 hours.*

The plot is calculated by following the steps described in Section 4.5.6 on choosing CVM-beta-values and shows that beta-values in the range [1.5, 2] return an optimal confidence threshold for the model that results in a minimum of time spent redacting documents for employees at Arkivverket.

Based on these results, the beta-values for the Custom Value Metric are set to **1.5**. Figure 5.3 illustrates the difference in scoring between the normal F1-score (equivalent to the CVM-value with a beta of 1) and the CVM-value with a beta of 1.5.



(a) *Metric scores in range [0, 1] for F1- score and CVM.*



(b) *Difference between F1- and CVM-scores in range [-0.08, 0.12].*

Figure 5.3: *Illustration of how increasing the beta-value in F-beta score increases the impact recall has on the metric score.*

As Figure 5.3a illustrates, the Custom Value Metric slightly shifts the heatmap so that in areas where the recall score is higher than the precision score, the metric score is increased compared to the normal F1-score. As the shift is quite subtle, it can be hard to tell from only the heatmaps. The third heatmap portrayed in 5.3b shows the difference in scores between the F1- and CVM scores. In the brightest area of the heatmap, the CVM-score is 12 percent higher than the F1-score. In the

darkest spot, however, the CVM score is 8 percent lower.

5.1.6 F1- and CVM-scores of fully trained models

Tables 5.5a and 5.5b shows the CVM-values for each confidence threshold.

Table 5.5: *CVM-values for each confidence threshold*

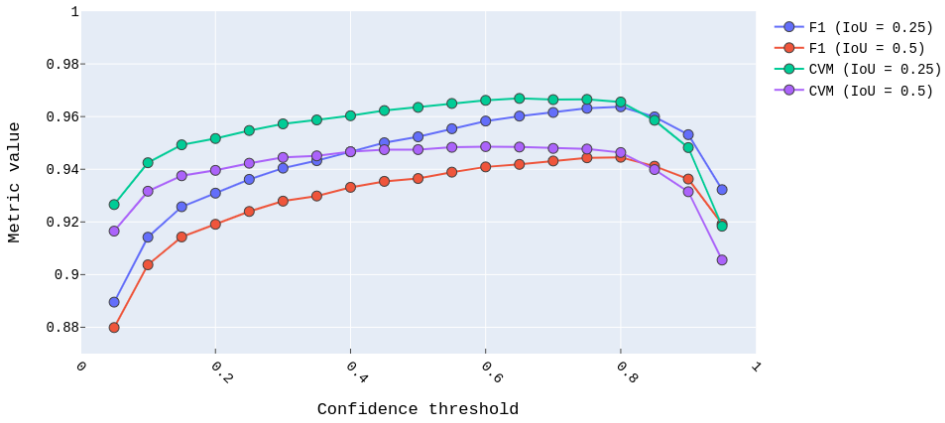
(a) Large model *results*

Conf. thr.	F1	CVM
5 %	0.889	0.923
... %		
20 %	0.931	0.952
25 %	0.936	0.955
... %		
55 %	0.955	0.965
60 %	0.958	0.966
65 %	0.960	**0.967**
70 %	0.961	0.966
75 %	0.963	0.966
80 %	**0.964**	0.965
... %		
95 %	0.932	0.912

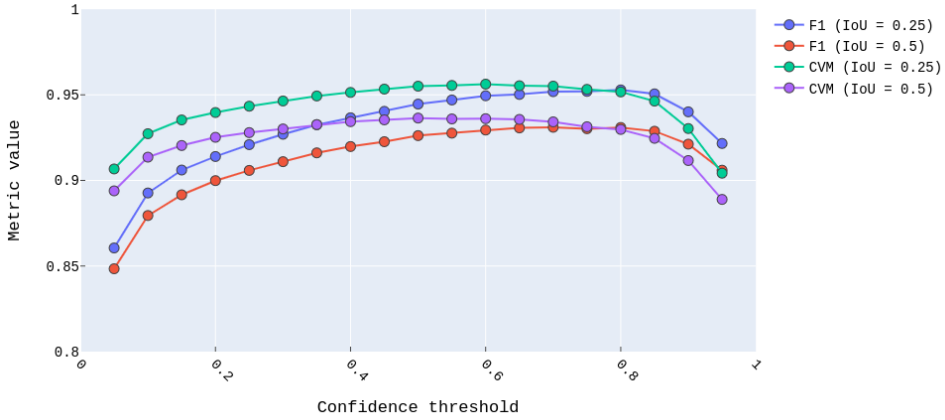
(b) Small model *results*

Conf. thr.	F1	CVM
5 %	0.860	0.907
... %		
20 %	0.914	0.940
25 %	0.921	0.943
... %		
55 %	0.947	0.955
60 %	0.949	**0.956**
65 %	0.950	0.955
70 %	0.952	0.955
75 %	0.952	0.953
80 %	**0.953**	0.952
... %		
95 %	0.923	0.904

As mentioned in Section 5.1.5, the optimal F1-score is found at the confidence threshold where both a relatively high recall- and precision-score is reached. Table 5.5 and Figure 5.4 (which shows the F1-performance by confidence thresholds), the optimal F1-score is found at a confidence threshold of 0.75. At this threshold, an F1-score of **0.953** is reached by combining recall- and precision-scores of 0.957 and 0.949, respectively.



(a) *Large model (Mask R-CNN ResNeXt 101 layers)*



(b) *Small model (Mask R-CNN ResNet 50 layers)*

Figure 5.4: *F1- and CVM-scores for small and large models for IoU-threshold $\in$ (0.25, 0.50).*

As Figure 5.4 shows, the custom value metric (CVM) is also calculated for each of the confidence thresholds, giving insight into what threshold may achieve the best cost-specific results for Arkivverket.

5.1.7 Training- and inference-speed comparison

As Section 5.1.6 shows, the larger model achieved higher recall- and precision scores at any confidence level. However, performance metrics are not the only

Table 5.6: *Comparison of training- and inference speed for the two extensively trained models. The large model spends approximately 50% more time per image in training, and approximately twice as long per image at inference.*

Model	Train time	Train time / img (relative)	Inference time	Inference time (relative)
Small model	2.784 imgs/s	1.000	4.2 it/s	1.000
Large model	1.860 imgs/s	1.497	8.2 it/s	1.952

relevant factors. Table 5.6 compares inference and training speeds for the two fully trained models.

A larger model causes an increase in training- and inference-time. This may have a negative impact on the user experience and will require more computational resources from the server.

5.1.8 A note on retraining the model on difficult samples

In an attempt to increase the model's accuracy when processing difficult samples, a model was retrained on images on which it made mistakes, whether it be false positives or false negatives. However, due to missing bounding boxes in the ground truth, the model adjusted its weights based on flawed data causing a decline in overall performance. As a result of this, retraining only on recall-error images was tried. As a rule, this resulted in a slightly increased recall score, but always at the expense of a large decrease in precision. Due to the sub-par results, retraining was not presented in this report as a viable option.

5.2 Ensembling / Stacking models

In an attempt to beat the best-performing single models trained in Section 5.1.2, two similar models are stacked, and the results are merged into a single prediction using affirmative voting. Test-time augmentation is also attempted, where the single best-performing "large" model makes a prediction on several variations of an image before every prediction is merged in the same way. Presented results for both techniques show results for NMS IoU-thresholds of 0.2 and 0.8. This is done to see whether the results are better when similar predictions are kept apart, or when only the most confident of the overlapping boxes are kept.

5.2.1 Stacking architectures

Two similar models are trained on the same training split to make a more generalized model for national identification numbers. Each prediction of the stacked model performs a vote function and non-maximum suppression on the same input image.

The two stacked models are:

1. Mask R-CNN ResNeXt 101-layers (previously trained "large" model)

2. Faster R-CNN ResNext 101-layers (Trained with same parameters as the above model)

Both models have a confidence threshold of 0.60, as the goal is to maximize Custom Value Metric.

5.2.2 Test-Time Augmentation

The predictor in the test-time augmentation ensembles is also the best single model, the Mask R-CNN ResNeXt 101-layers. Three predictions are made for each image, based on sharpening the image, thresholding color-values, and performing a "remove-lines" method.

5.2.3 Ensembling Results

Table 5.7 compares the performance of stacked- and TTA-ensemble models for two different NMS-thresholds, 0.2 and 0.8, against the single-model baseline.

Table 5.7: *Performance comparison of stacked- and TTA-ensemble models with different NMS-thresholds*

	NMS-threshold	**F1-Score**	**Recall**	**Precision**	**Custom Value Metric**
Best single model @ 0.65 conf.	–	0.960	0.978	0.943	0.967
Stacked model 1	0.2	0.964	0.979	0.949	**0.969**
Stacked model 2	0.8	0.941	0.983	0.902	0.957
Test-time augmentation 1	0.2	0.962	0.978	0.946	0.968
Test-time augmentation 2	0.8	0.937	0.981	0.896	0.953

5.2.4 Inference time

The inference time for models and ensembles is based on the number of predictions, and corresponding calculations, to be made for each inference on a document. For the stacked ensembles in this thesis, two models make predictions, and the inference time is thus approximately twice that of a single model.

For test-time augmentation ensembles, a prediction has to be made for each variation of the input image. Also, the manipulation of the input image at each augmentation involves further calculations. This makes the TTA-approach slightly slower than the multiplicative imposed by the number of predictions.

5.3 Summary of model performance

Table 5.8: *Summary of model performance. Results for the first (small) model is calculated using a confidence threshold of 60%, while the remaining results are calculated using a confidence threshold of 65%. Choice of confidence thresholds are based on results in Table 5.5.*

Top model	Recall	Precision	F1	CVM	Inf.Speed
Best single, small model (Mask R-CNN)	0.967	0.932	0.949	0.956	**8.2 it/s**
Best single, large model (Mask R-CNN)	0.978	0.943	0.960	0.967	4.2 it/s
Stacked architectures (2)	**0.979**	**0.949**	**0.964**	**0.969**	2.1 it/s
Test-time augmentation (3)	0.978	0.946	0.962	0.968	1.2 it/s

Based on these results, we recommend that Arkivverket consider deploying one of two models:

1. **For accuracy:** The most accurate model is the model ensemble containing two fully trained 101-layer-models with ResNeXt backbones. This model has the highest CVM-score, i.e. the best combination of precision and recall. This will help Arkivverket redact as many national identification numbers as possible.

2. **For speed:** The fastest model, both with regards to model training, required training resources, and inference speed, is the 50-layer Mast R-CNN model with a ResNet backbone. Faster inference time may make the redaction process more pleasant for the employee, as suggestions will load faster as a document is loaded into the redaction software.

The most accurate (stacked) model achieves significantly better scores for all metrics but at the expense of being almost four times slower than the most agile (single) model. Our recommendation is that Arkivverket begins by deploying the most accurate model, even if it is slower, and use it in the initial pilot projects to check whether the decrease in speed is experienced as a hindrance for the end-user. If users report that they prefer a model that gives out suggestions faster in the redaction process over the more accurate model, they should implement the fastest model.

5.4 Visualizing model predictions

The purpose of this section is to demonstrate correct and incorrect predictions from a few documents from the test split. The presented examples are based on the best-performing single, large model from Table 5.8. Ground-truth bounding boxes and predicted bounding boxes are marked in red and blue, respectively.

Figure 5.5 shows a predicted document with a standard, tabular layout with all correct predictions. Figures 5.6 and 5.7 show examples where the object detection model performed better than the ground truth by identifying numbers that were omitted by employees, as well as examples where the model predicted difficult-to-read national identity numbers. Figures 5.8 and 5.9 Show errors in the predicted documents, either recall errors where the model missed national identity numbers or precision errors where the model incorrectly predicted a number.

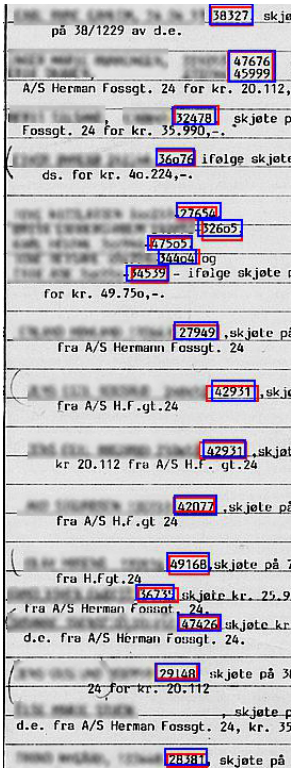


Figure 5.5: *Document containing multiple national identity numbers, where the ground truth (red boxes) and predictions (blue boxes) match.*

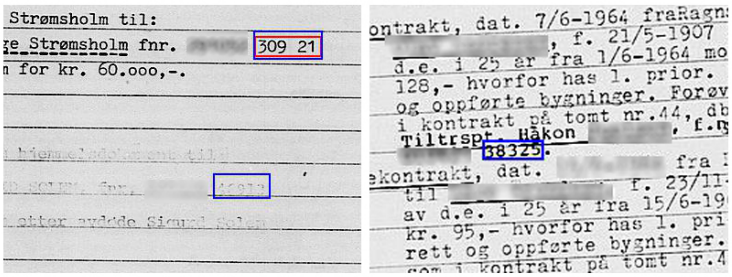


Figure 5.6: ***Left****: The object detection model detects a hard-to-read national identity number (bottom blue box) that is not in the ground-truth (red boxes).* ***Right****: The model detects a national identity number (blue box) that is not in the ground truth. The number is partially distorted by horizontal lines.*

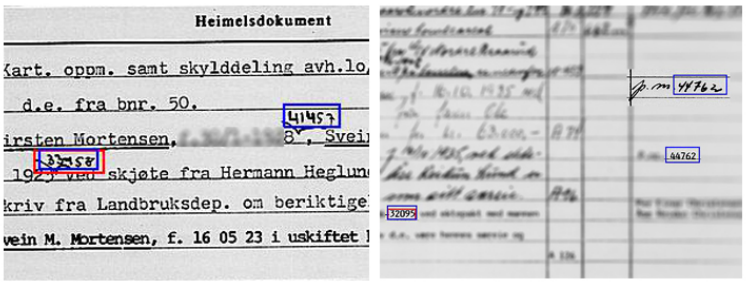


Figure 5.7: *The model detects unstructured, handwritten national identity numbers (blue boxes) in two documents. Some of the predicted numbers are not in the ground truth (red boxes).*

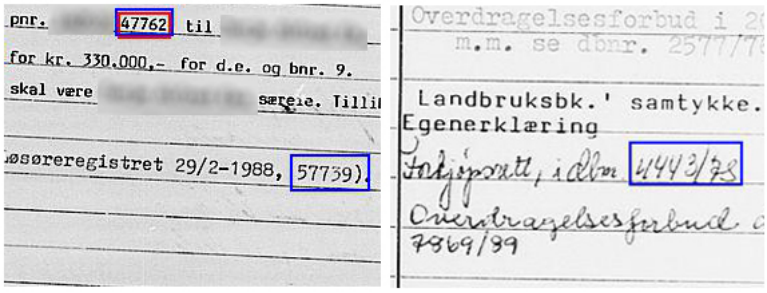


Figure 5.8: *Two example documents where the model made false-positive predictions, i.e. incorrectly predicted a national identity number. Blue boxes illustrate predictions and red boxes illustrate ground truth.*

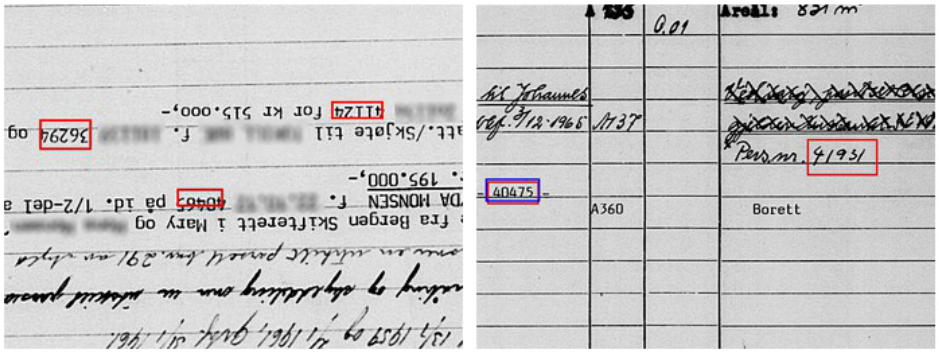


Figure 5.9: *Example documents where the model missed national identity numbers in the ground truth (red boxes).* ***Left****: The document is scanned upside-down, and the model is unable to identify any of the three national identity numbers.* ***Right****: The model is unable to identify the far-right national identity number.*

5.5 Estimating count of omitted national identity numbers

There are a couple of ways to determine *how many* national identification numbers are missed by manual redactors in the dataset provided by Arkivverket. The first and most obvious method is to manually examine each document in the dataset and the ground truth redactions provided, and search for missed national identity numbers. However, with a image count of 20 000 images, this is outside the time constraints of this thesis project.

A second option is to use the best-performing model from Chapter 5 to find an approximation of the count of missed national identification numbers. This can be done by setting the *confidence score parameter* of the model to a very high value, i.e. 90-95%, in order to make it suggest only the bounding boxes in which the model is confident there is one or more national identity numbers. By matching the suggested bounding boxes against the ground truth provided by Arkivverket, and then using the *precision score* (see Section 4.5.5), it is possible to make an estimation of how many national identity numbers are omitted.

In order to make a sound estimation of the count of omitted national identity numbers, the best single, large model from Section 5.8 is used. Our approach for finding the percentage of omitted numbers consists of the following steps:

1. Select a random sample of 1 000 images from the dataset. In this case, it is arbitrary whether the training- or test-split is chosen, since the purpose is to identify national identity numbers that are not in the ground truth.

2. Predict bounding boxes in each of the 1 000 images and compare boxes to the ground truth.

3. Manually inspect every image containing one or more *false-positive* hits, e.g. images where the model predicted (with high confidence) the presence of national identity numbers that were not in the ground truth. Check the predicted bounding boxes and count the number of cases where the prediction was correct and the number of cases where the predictions were wrong.

4. Calculate the fraction of correctly predicted bounding boxes that were *not* in the ground truth but were predicted by the model.

The results of the analysis were the following:

- 1 000 random images were chosen from the training split and the confidence threshold of the model was set to 90%.

- The 1 000 images contained a total of 1 763 bounding boxes in the ground truth, close to the mean of the dataset.

- The model predicted 48 bounding boxes across 44 images that were not in the ground truth. After manual inspection of the 44 images, 4 bounding boxes were found to be actually false positives, leaving 44 omitted bounding boxes.

- The fraction of omitted bounding boxes for the 1 000 sample images were thus $\frac{44}{1\,763} = 0.0249 = \mathbf{2.49\%}$

The confidence threshold of 90% was set to push the model just across the threshold where it starts making mistakes, as seen in the 4 false positives. The estimation of approximately **2.5%** omitted bounding boxes in the dataset is further strengthened by checking the precision score of the model when pushing the confidence threshold all the way up to 95% and making predictions across the full dataset. In this case, the model reaches a precision score of 0.9753, or a precision error of **2.47%**. Although no manual inspection is performed when checking precision across all 20 000 documents, a confidence threshold as high as 95% should not return many false positives, especially considering the fact that a confidence threshold of 90% revealed only 4 actual false positives for 1 763 bounding in the inspected sample-set.

Thus, an estimation of **2.5%** omitted bounding boxes in the dataset is used in economic calculations when estimating the real-world performance of the model, or in other words and experienced precision when the model is implemented in production.

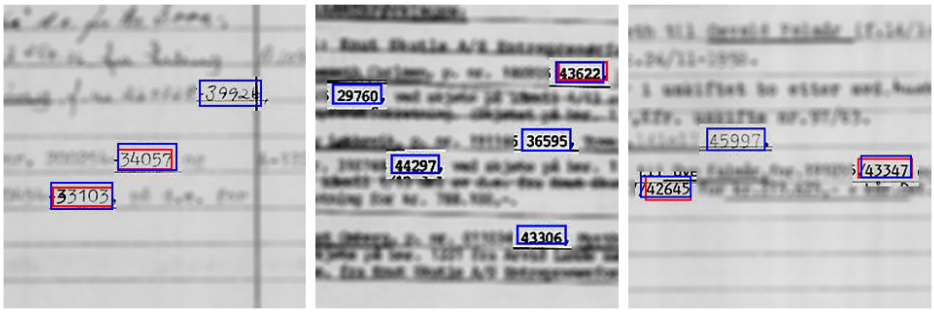


Figure 5.10: *Examples of bounding boxes that were omitted from the ground truth, but correctly predicted by the model (where red and blue rectangles do not overlap). Red rectangles indicate ground truth (manual redaction), and the blue rectangles indicate the model's predictions.*

5.6 Further work

5.6.1 Relabelling the dataset

As described in Section 3.1.3, quirks and errors in the underlying dataset provided by Arkivverket directly affect neural networks' ability to identify national identity numbers in the corpus accurately. Arkivverket could correct two main weaknesses in the dataset by relabelling the dataset:

First, the method for labeling numbers should be uniform across all identified numbers in all documents. The issue of sometimes labeling numbers individually and sometimes in batches disrupt the shapes and sizes a neural network needs to recognize as a target, as numbers are no longer single five-digit words but potentially multiple rows of them. One can assume that this negatively affects the overall performance of the model. Still, it may also increase the neural network's required capacity to learn representations of the targets. This will, in turn, decrease the speed and increase the size of the model.

The second way to improve the dataset is to take greater care to identify all (or at least more) of the dataset's national identity numbers. The consequence on model performance is likely reduced somewhat when the model is shown examples of actual national identity numbers but is mistakenly instructed not to label these as such.

A suggestion is to use the models developed in this project as the basis for relabelling the dataset. Arkivverket may do this by first using the model to suggest each document image and then manually removing false positives or adding missed numbers to the ground truth (see Section 6.1.3).

5.6.2 Creating a custom loss function

It may be possible to increase the model's ability to adapt to the weaknesses in the dataset by creating a custom loss function that takes the shape of the target bounding boxes into account. It could do this by taking the relationship between height and width of each bounding box into account when measuring loss, and then score predictions based on this relationship. A possible solution could be to let the IoU-cutoff between prediction and ground-truth be adaptive based on the width-height relationship, making the scoring more lenient in cases where target bounding boxes are tall.

5.6.3 Analysis of Layer Activation

Analyzing layer activation in the layers of an object detection model is a visual and intuitive way of understanding the high- and low-level features recognized by the model. Though this analysis is outside the scope of this thesis, it could provide insight into how one may improve future models.

5.6.4 Further model development and testing

Preprocessing steps

Although experiments conducted in this project returned better results when using the raw images in the training pipeline, other image manipulation techniques may increase the readability of documents which in turn might yield better results.

Implementing YOLO v4- or v5-models in ensembles

The scope of this project is limited to use models from the Detectron2 framework. However, the family of architectures called YOLO (You Only Look Once) is very popularly used for the same object detection tasks performed in these experiments. The most likely candidates to give Detectron2 real competition are the YOLO v4- and v5-models.

The implementation of YOLO may also perform well when stacked together with Detectron2-models, as the frameworks are vastly different. The architectures used for model stacking in this report have the weakness of being quite similarly built. Implementing a YOLO v4-model may introduce new ways of identifying high- or low-level features in documents.

Alternative Pretraining

All models used in this project were pretrained on the COCO-dataset as described in Section 4.5.2. The purpose of pretraining models on images that at first glance may look nothing like documents is to teach a model to recognize low-level features such as shapes and blobs.

Although pretraining models on the COCO-dataset seem to have worked very well, other pretraining datasets may have more similarities with the documents on which

the models are better suited to identify the specific high-level features in the document images of this specific project. At the time of writing, such a dataset is unknown to the authors.

6. Model deployment and maintenance

Based on the results discussed in Chapter 5, this chapter will describe our suggestions as to how the models developed in this thesis may be deployed to production at Arkivverket.

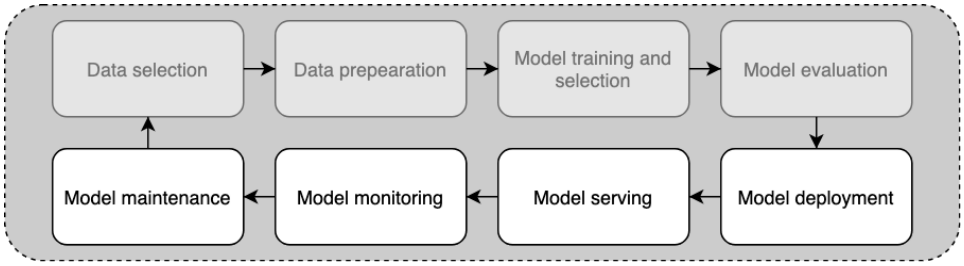


Figure 6.1: *The model deployment and maintenance phase of a machine learning project life cycle (highlighted).*

6.1 Project packaging and architecture

This section describes the Python package that is a product of this thesis (Detectlib), and one of the possible architectures of the model deployment.

6.1.1 Detectlib package

The Detectlib Python package is the code base for experiments performed in this thesis and the tools needed to deploy Detectron2 models through an API. It is handed over to Arkivverket as part of the prototype for model deployment, along with extensive documentation.

The Detectlib package is made up of 7 main modules:

- **model training module**: Containing tools and script for training Detectron2 models, such as custom train-loaders with augmentation.
- **model evaluation module**: Containing tools needed for evaluation models at scale, with custom metrics, plotting tools, and test-set evaluation functions.
- **model library module**: Containing fully trained Detectron2 models to be deployed on their own or part of a model ensemble.
- **prediction module**: Containing tools for performing non-standard predictions, such as stacked predictions or test-time-augmentation.

- **data handling module**: Containing tools for manipulating data, preprocessing images, formatting data to COCO-format, and more.
- **avmllib mod module**: A miniature version of the avmllib package developed and provided by data scientists at Arkivverket, containing tools for interacting with Digitalarkivet API, as well as preprocessing and manipulating images.
- **api tools module**: Containing classes and functions used to validate API requests and responses, as well as instantiating models and performing predictions per request on a server.

6.1.2 API architecture

To deploy the models developed in this thesis, a network of APIs, containers, and databases needs to work together and with external administrative tools to make the model available to the end-user.

Figure 6.2 gives a birds-eye-view of an example of such an architecture, and the following sub-chapters will describe each element in the diagram.

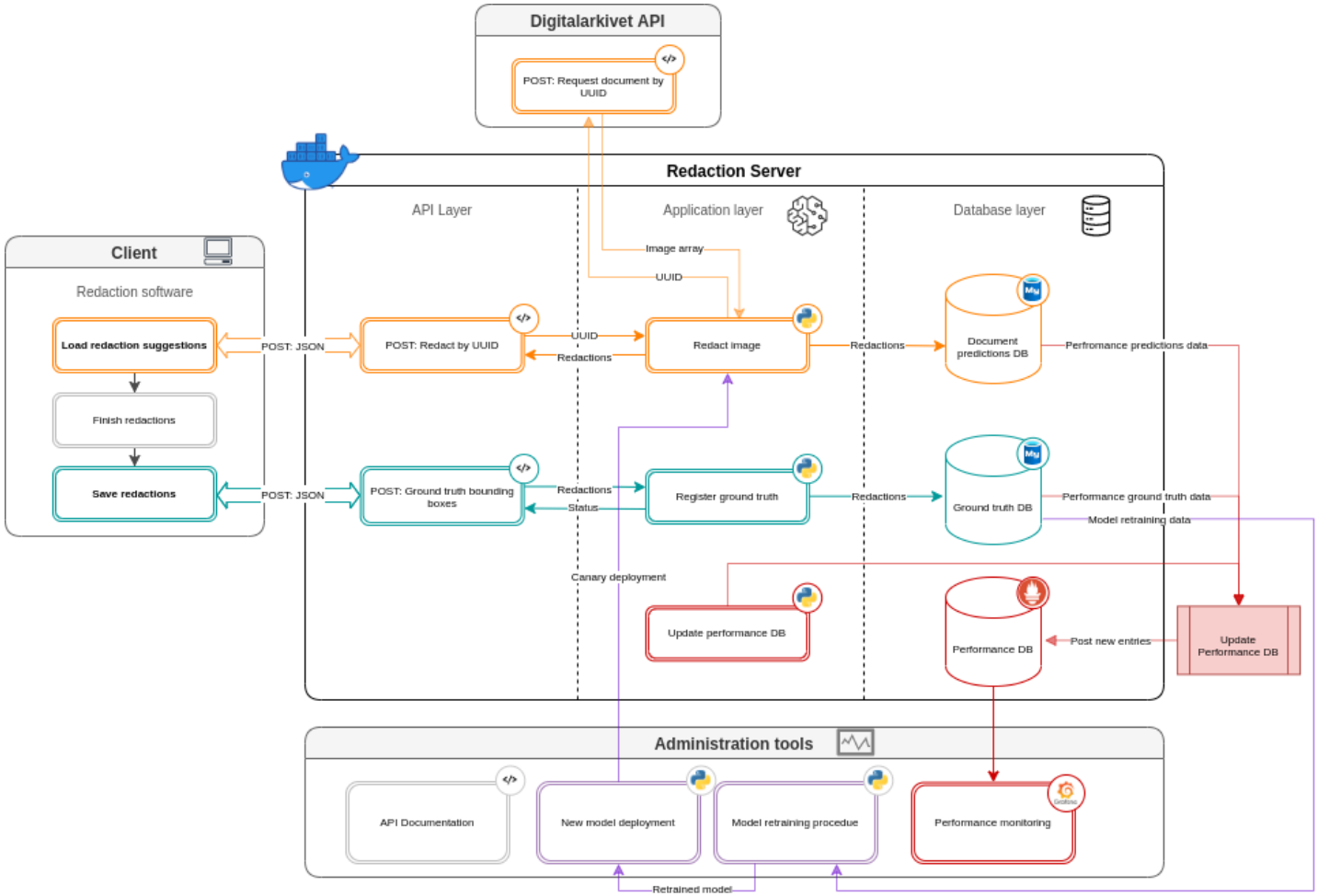


Figure 6.2: *Diagram showing the architecture of deployed model*

Client

The client is the end-user, in this case the employees overseeing and performing redaction of documents. The client software is assumed to be third-party redaction software or redaction software developed and maintained in-house.

The client interacts with the deployed machine learning model through an API-layer, which is part of a "dockerized container" (see Section 2.5.2) on a local server. This is done by sending a request to the server with one of two intentions:

1. Requesting the proposed redactions for a document image in the Digitalarkivet database, or
2. Submitting redactions from manually redacted document images.

Both of these actions are done implicitly or by the click of a button. Suggestions for document redaction areas may be requested automatically by the redaction software as a document is loaded in the interface, and submitting ground-truth redaction areas can be submitted as the document, and final redactions are saved by the user.

Redaction server

The redaction server contains the "brain" of the workflow and the methods for communicating with the machine learning model. The redaction server is made up of three "layers":

- The API layer, which allows communication between the user/client and the application layer.
- The application layer, which houses the scripts used for image prediction, the code base of the API, communication with databases, and other utilities.
- The database layer, which symbolizes the databases that the application layers interact with.

In practice, these layers are part of the same codebases, where:

- The API layer and application layers are part of a Docker container which is hosted on the redaction server, and
- The databases in the database layer may either be hosted on the same server as the other layers (the redaction server) or hosted on another server altogether. In that case, the application layer would contain the necessary code for communication with the databases.

Administration tools

Another part of deploying the model is the suite of administration tools that are part of the project but not a direct part of the application.

The administration tool suite is made up of four parts:

- API documentation used to understand and further develop the code on the server.
- Performance monitoring app (Grafana), used to monitor (in real-time) the model performance against the manually adjusted ground truths.
- Model retraining procedure, for retraining models on the updated and improved ground-truth data collected in the ground-truth database over time.
- Model deployment procedure, for deploying the retrained models safely and productively.

6.1.3 API-flows

The architecture described in Figure 6.2 is divided into three main *flows*; redaction, ground-truth feedback, and performance monitoring and model maintenance.

Redaction flow

The core of the project is the redaction flow. This flow is triggered every time a client requests redaction suggestions using the trained object detection models.

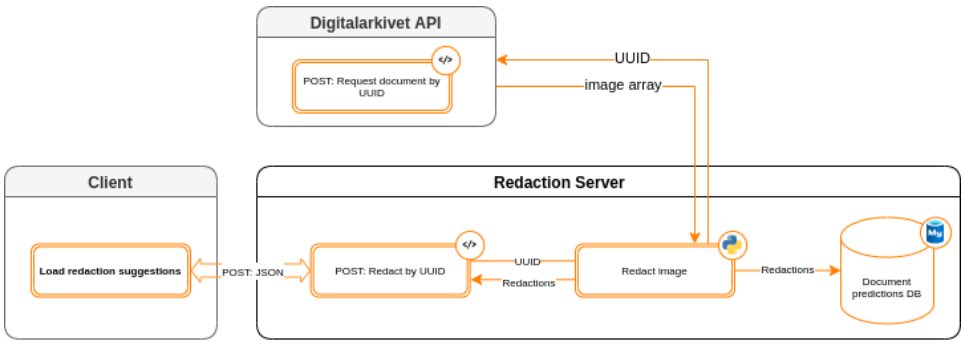


Figure 6.3: *Cut-out from 6.2 with overview of the redaction flow.*

The redaction flow consists of five main steps:

1. The end-user loads a document image, identified by a UUID, into the redaction software, and the software automatically sends a request to the project API.
2. The UUID is transmitted to the "redaction engine" on the Redaction Serer through the API layer.
3. The redaction engine sends a request to the Digitarkivet API, Akriverkets database, using the same UUID, and receives a pixel-array representing the document image.

4. A prediction is made on the image array, returning a series of redaction areas and corresponding confidence scores.

5. The predicted redaction areas are sent to two locations:

    - To a database dedicated to predictions made by the model, and
    - Back to the user (or the redaction software) to be corrected, if necessary, and saved.

Ground-truth feedback flow

To improve the model's predictions by retraining and monitoring the model's performance in real-time, it is necessary to save the ground truth for each image run through the redaction software. This is handled by "capturing" the bounding boxes that the employee saves after reviewing and correcting the redaction software's suggested redactions.

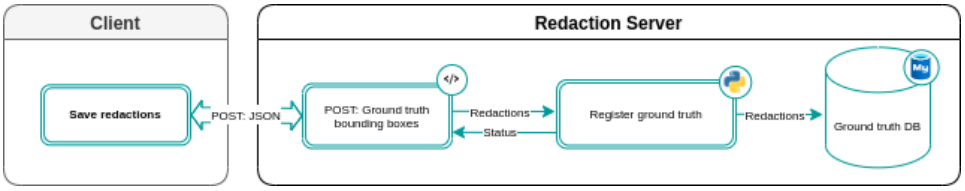


Figure 6.4: *Cut-out from 6.2 with overview of the register-ground-truth flow.*

The feedback flow consists of three main steps:

1. The end-user reviews and saves the redaction areas for an image identified by a UUID.

2. The UUID and corresponding redaction areas are packaged as a JSON and transmitted to the project API-layer and forwarded to the "registering script".

3. This dockerized python script unpacks the JSON and adds the contents as entries to the database, keeping track of ground-truth redactions.

Performance monitoring and model maintenance flow

As the two preceding flows regularly update the two databases containing image predictions and ground-truths, it is possible to monitor the predictions' performance in real-time and maintain and retrain the object detection model at frequent intervals.

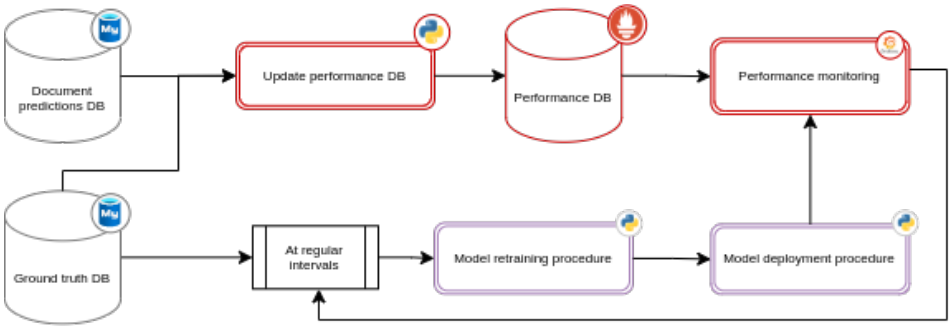


Figure 6.5: *Cut-out from 6.2 with overview of monitoring- and maintenance flow.*

The performance monitoring- and maintenance flow is made up of two separate but tightly linked flows. The first is the monitoring flow:

1. At regular intervals (hourly or daily), a dockerized script is run, which performs the following tasks:
   (a) Loads database entries from both the predicted and ground-truth databases [*],
   (b) Calculates key performance metrics for document UUIDS that are present in both databases, and
   (c) Commits the metrics, combined with the predicted database entries' timestamps, to the *performance database*.
2. A monitoring dashboard continuously queries the performance database for updated logs of timestamped metrics and displays performance over time.

The second flow is the model maintenance and retraining flow:

1. At frequent intervals (weekly or monthly, depending on usage observed performance), entries in the ground-truth database are used to retrain in an attempt to improve the object detection model [†].
2. The retrained model is deployed on the Redaction Server, using a deployment method such as *canary deployment* or similar. If the retrained model

[*]To enable scalability, metrics are only calculated for database entries that are not already present in the performance database.

[†]The model may improve in multiple ways. First, the registered ground-truths may be of higher quality than the dataset on which the model was originally trained (see Section 3.1.3). It may also improve by being trained on document variations or layout style that has recently been introduced and were not part of the original training data.

performs better than the previous model(s) over time, it fully replaces other models.

6.1.4 Performance monitoring

As mentioned above, there are many benefits to monitoring the performance of machine learning models in real-time.

By continuously receiving feedback on key metrics such as recall and precision, it is easy to identify cases where the model starts under-performing. This may be due to bugs in the prediction software, bugs in the client's redaction software, or changes in the document data input by the end-user.

Figure 6.6: *Illustration of performance monitoring of the machine learning models in real-time using Grafana Dashboard. See Appendix C for higher resolution.*

Figure 6.6 shows an example of how such an interface may look. It is developed for this project using the open-source Grafana software and shows potential features, such as:

- Displaying key metrics such as average recall, F1-score, and precision for different time intervals, making it easy to spot when the model falls below expected performance, and
- Histograms displaying the solution's usage by counting requests made to the API over time, making it possible to strike the alarm if employees stop using the solution and work on identifying the cause.

6.1.5 Choice of databases

An important part of making a project scalable is to use databases with optimized infrastructures for the purpose of which they are utilized in the first place. The deployment method suggested in this thesis uses three different database entities, where two or more may be separate tables in the same database. However, as the performance monitoring database serves a different purpose than that of the other two entities, the database architecture should be specialized towards this purpose.

The first two database entities described in Figure 6.5 are meant to serve the logs of predictions and ground-truths only occasionally and also need to filter new entries to the database to prevent duplicate entries. Thus, using a traditional relational database such as MySQL should be adequate. In that case, only a single database is needed, with a separate table of each of the entities.

However, the third database entity will be queried frequently by the monitoring software and should be able to quickly serve high-resolution time-series data for different periods, despite a considerable amount of metrics-entries.

To ensure scalability as the number of entries in the performance monitoring database grows continuously, a time-series database such as Prometheus (prometheus.io) or InfluxDB (influxdata.com) should work sufficiently.

6.1.6 Load-testing before launch

To test how the redaction-API functions under load, we recommend performing load-testing using Locust.io (see Section 2.5.2) **before exposing the API to test-users.**

By using Locust.io to simulate users requesting redactions on various document images, Arkivverket can analyze how many predictions the model can serve at any given time frame and how it performs under increasingly high loads. Locust.io gives Arkivverket the ability to test the new system with many mock-users that far surpasses the load they would be able to stage using real users before the application is deployed, exposing the robustness of the system.

6.2 Economic implications

6.2.1 Savings potential

Figure 6.7 gives an overview of the current workflow of the redaction process before the implementation of the automatic algorithm.

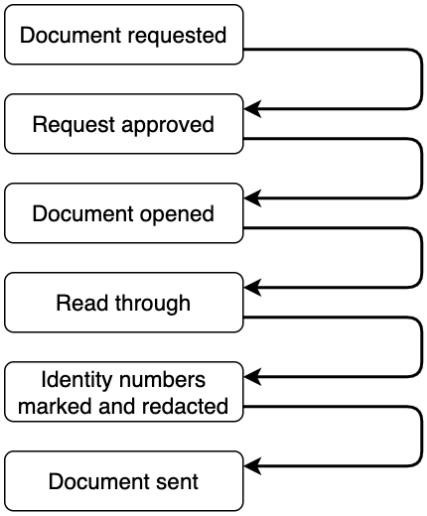


Figure 6.7: *Overview of the redaction process using the current solution at Arkivverket*

The process starts as the requester submits an application for the document, followed by an employee opening and accepting the application. Afterward, the employee will find the document in the file system before it is opened. The employee will then read through the document to find national identity numbers and mark each occurrence with a two-anchored redaction box. When the document is redacted, the employee sends the document to the requester.

The assumptions made in 4.4 suggests that an employee will spend, on average, 12 seconds reviewing and processing any document. Additionally, the employee will use 6 seconds to add a redaction box for each national identity number where there are no suggestions made by a model, and 2 seconds correcting a wrong suggestion. The documents used in this thesis contain, on average, 1.786 bounding boxes per page.

Before the implementation of the proposed model, estimations based on the listed assumptions suggests each document will require 22.7 seconds of processing time from the document is opened to the redaction boxes are in place. For every 10,000 pages, these calculations suggest that Akrivverket spends 63.1 hours marking identity numbers and redacting.

Metrics from the most accurate model, as described in Section 5.3 and adjusted for omitted bounding boxes in Section 5.5, suggest that the model will miss 2.1% of the actual national identity numbers (i.e. not make a suggestion), and incorrectly suggest national identity numbers in 2.6% of the cases. A precision-error of 2.6% is found by adjusting the precision-error of the best-performing model in Table 5.8 (5.1%) by subtracting the estimated 2.5% of national identity numbers that are

missed by human redactors and thus not in the ground truth.

Using the metrics of the most accurate model, for every 10,000 pages, 0.26 hours are spent correcting false positives and 0.62 hours for false negatives. This is added to each document page's base processing time, 33.33 hours per 10,000 pages, totaling at 34.20 hours (see Appendix B).

These calculations suggest that 28.88 hours will be saved per 10,000 pages by implementing the redaction suggestion algorithm. Operating with a yearly wage of 679,120 NOK, including expenses such as employer's tax and benefits, the hourly cost of labor is estimated at 394 NOK. The implementation will, in this case, save Arkivverket 11,370 NOK per 10,000 pages. Based on records provided by Arkivverket, they have the last 12 months redacted and released approximately 70,000 pages. Using the calculations above, Arkivverket can save 79,554 NOK by implementing the algorithm (see Appendix B). Alternatively, keeping the hours worked constant, 59 000 more pages could be redacted per year. If the algorithm reaches the satisfactory performance needed for working autonomously, the savings potential could be far greater.

Implementation cost

As the algorithm is already developed at no additional cost for Arkivverket, the estimated profitability of implementation can be calculated from the estimated savings and the cost of implementation. Even though the algorithm is ready to use, new software for the employees needs to be developed. There is uncertainty connected with whether or not this can be done by Arkivverket's own employees, both in terms of available time and capabilities. However, there will be a cost associated with developing such software, whether it be the cost of labor of own employees or developers from outside the organization.

Project profitability

Rough estimates suggest that software development for implementing the algorithm will require 115 hours of work, provided they can develop it using their own employees. Also, the algorithm will need monthly maintenance, estimated at three working hours per month, totaling 151 hours the year of implementation. Using the same labor cost as above, the implementation cost is calculated at 59,448 NOK in the first year. Based on these estimates, the implementation profitability will total 20,107 NOK in the year of implementation. For the following years, without the cost of implementation, the yearly profitability can be estimated at 65,381 NOK.

6.2.2 Ethical implications

More national identity numbers get censored

Whilst reviewing the bounding boxes set by the employees of Arkivverket, it is evident that many national identity numbers are left unmarked. By implementing

the models described in this thesis, a potential benefit might be an increase in national identity numbers redacted, and fewer released unnoticed.

Automation bias

Automation bias [57] occurs when one depends excessively on automated systems, causing errors as humans tend to disregard or fails to search for contradictory information in light of a computer-generated solution. If the employees responsible for reviewing and correcting the software's suggestions rely excessively on the generated suggestions for redaction, unredacted national identity numbers might be released, and areas without national identity numbers could be wrongfully redacted.

As described in Chapter 5, the algorithm is far from perfect. However, it is successful in finding 97.9% of all national identity numbers. It is plausible that the high rate of success by the algorithm and the automation bias may cause the employees to rely excessively on its suggestions, causing them to ignore errors. In such an occurrence, it will harm the efficiency of this project.

6.2.3 Other consequences

As described in Chapter 2, automation of tasks may have an impact on the labor market. However, it seems unlikely that machine learning-driven task automation will have a major impact on the number of jobs in most workplaces, at least in the near future. A more feasible scenario is that task automation will cause a shift in focus. As repetitive tasks are being automated, employees have more time for activities that are less suited for automation, such as tasks requiring a high level of strategic or creative thinking. In turn, this might cause an increase in work satisfaction [38].

7. Conclusions

Two main goals were defined for this thesis project:

1. To test whether or not object detection models can separate specific sequences of numbers and text, e.g., national identity numbers, from similar tokens in the document. And,
2. To test if this approach is sufficiently accurate at identifying national identity numbers to be implemented as an automated suggestion feature in the redaction workflow.

The results in this report show that object detection models are indeed able to identify national identity numbers in a historical document. Results show that using an abstract objection detection-based approach to finding national identity numbers in historical documents delivers more accurate results than the prototype currently developed at Arkivverket. A likely reason for this is that the current solution is dependent on successfully extracting and interpreting words before making a prediction, making it vulnerable to missing out on handwritten ID numbers. The most accurate model proposed in Table 5.8 achieved metric scores of 97.9% recall, 94.9% precision, an F1-score of 96.4% on the test split containing 1 920 images. Compared to the baseline results achieved by the prototype at Arkivverket of 89.0% recall, 88.3% precision, and an F1-score of 88.6%, our model measures an **8.9% and 6.6% positive difference in recall and precision, respectively, and an improved F1-score of 7.8%**[*]. This increase in performance may largely be due to the object detection models' ability to recognize handwritten national identity numbers in the documents.

Our assessment of the project is that the object detection models listed in Table 5.8 are sufficiently accurate at identifying national identity numbers to be implemented in a trial project at Arkivverket as part of a suggestion feature in the existing redaction software. This would involve employees being presented suggestions for national identification numbers in a given document and manually correct the suggestions. By deploying the selected model as described in Chapter 6, Arkivverket will be able to continuously monitor performance, make adjustments to the models when needed, and improve the quality of the datasets by creating new training data from the quality-assured redactions. At regular intervals, they can then retrain the models on the improved dataset, and compare performance of the new model against the model already in production.

Estimates from Section 6.2.1 show that implementing a machine learning algorithm to suggest bounding boxes may save Arkivverket 65,381 NOK yearly as the

[*]Note that the baseline results reported by Arkivverket were not measured on the same samples as the models in this report, meaning comparisons should be regarded as indicative.

redaction process becomes more efficient. Implementation may also increase work satisfaction by simplifying repetitive and non-rewarding tasks.

With time and further research, the process of redacting national identity numbers may become fully autonomous. In fact, considering our estimation that approximately 2.5% of all national identity numbers are overlooked by employees in the process of manual redaction (see Section 5.5), the developed model, with a recall-error of 2.1%, should be able to autonomously match (or exceed) the number of national identity numbers that are identified by its human counterparts. These results have to be tested through pilot projects at Arkivverket on new and unseen data to be confirmed, but they indicate the possibility that a fully autonomous redaction workflow could be successfully implemented in the near future.

Bibliography

[1] NorSIS, "Trusler og trender 2018-2019," Jan 2019. [Online]. Available: https://norsis.no/trusler-og-trender-2018-1019/

[2] K. Frøjd, "Høgskole varsler studenter om personopplysninger på avveie," Aug 2020. [Online]. Available: https://www.tv2.no/a/11602636/

[3] M. Hofstad, "Tar lekkasje-sak til stortinget," Oct 2020. [Online]. Available: https://www.hblad.no/2020/nyheter/tar-lekkasje-sak-til-stortinget/

[4] A. Budalen, L. F. Hagen, and O.-C. Olsen, "Reagerer på at usladdede rapporter ble delt med pr-byrå: – vanskelig å finne ord," Oct 2020. [Online]. Available: https://www.nrk.no/nordland/usladdede-rapporter-fra-helgelandssykehuset-delt-med-wergeland-apenes-pr-byra-_-helse-nord-bekymret-1.15181973

[5] Datatilsynet, "Fødselsnummer," Jul 2019. [Online]. Available: https://www.datatilsynet.no/rettigheter-og-plikter/personopplysninger/fodselsnummer/

[6] A. Færaas, "Arkivverket har lagt ut svært mange nordmenns personnumre på nettet," Sep 2016. [Online]. Available: https://www.aftenposten.no/norge/i/wLrVo/arkivverket-harlagt-ut-svaert-mange-nordmenns-personnumre-paa-nettet

[7] S. Raschka and V. Mirjalili, *Python machine learning: Machine learning and deep learning with Python, scikit-learn, and TensorFlow 2*. Packt Publishing Ltd, 2019.

[8] A. Burkov, *Machine learning engineering*. True Positive Inc., 2020.

[9] M. Rajchl, M. C. Lee, O. Oktay, K. Kamnitsas, J. Passerat-Palmbach, W. Bai, M. Damodaram, M. A. Rutherford, J. V. Hajnal, B. Kainz *et al.*, "Deepcut: Object segmentation from bounding box annotations using convolutional neural networks," *IEEE transactions on medical imaging*, vol. 36, no. 2, pp. 674–683, 2016.

[10] T. Shah, "About train, validation and test sets in machine learning," Dec 2017. [Online]. Available: https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7

[11] A. Famili, W.-M. Shen, R. Weber, and E. Simoudis, "Data preprocessing and intelligent data analysis," *Intelligent data analysis*, vol. 1, no. 1, pp. 3–23, 1997.

[12] J. Furseth and O. Ljones, "50-årsjubilant med behov for oppgrader-

ing," Apr 2015. [Online]. Available: https://www.ssb.no/befolkning/artikler-og-publikasjoner/50-arsjubilant-med-behov-for-oppgradering

[13] Skatteetaten, "Norwegian national identity number." [Online]. Available: https://www.skatteetaten.no/en/person/national-registry/birth-and-name-selection/children-born-in-norway/national-id-number/

[14] C. Song, T. Ristenpart, and V. Shmatikov, "Machine learning models that remember too much," in *Proceedings of the 2017 ACM SIGSAC Conference on computer and communications security*, 2017, pp. 587–601.

[15] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[16] F. Rosenbaltt, "The perceptron - a perciving and recognizing automation," *Report 85-460-1 Cornell Aeronautical Laboratory, Ithaca, Tech. Rep.*, 1957.

[17] B. Widrow, "An adaptive 'ADALINE' neuron using chemical 'memistors', 1553-1552," 1960.

[18] D.-X. Zhou, "Universality of deep convolutional neural networks," *Applied and computational harmonic analysis*, vol. 48, no. 2, pp. 787–794, 2020.

[19] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network," in *Advances in neural information processing systems*, 1990, pp. 396–404.

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[21] O. Wichrowska, N. Maheswaranathan, M. W. Hoffman, S. G. Colmenarejo, M. Denil, N. Freitas, and J. Sohl-Dickstein, "Learned optimizers that scale and generalize," in *International Conference on Machine Learning*. PMLR, 2017, pp. 3751–3760.

[22] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[23] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*. PMLR, 2015, pp. 448–456.

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

[25] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1492–1500.

[26] C.-F. Wang, "The vanishing gradient problem," Jan 2019. [Online]. Available: https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484

[27] L. Torrey and J. Shavlik, "Transfer learning," in *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*. IGI global, 2010, pp. 242–264.

[28] S. Sun, Y. Yin, X. Wang, D. Xu, W. Wu, and Q. Gu, "Fast object detection based on binary deep convolution neural networks," *CAAI transactions on intelligence technology*, vol. 3, no. 4, pp. 191–197, 2018.

[29] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," 2014.

[30] R. Gandhi, "R-cnn, fast r-cnn, faster r-cnn, yolo - object detection algorithms," Jul 2018. [Online]. Available: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

[31] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[32] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *arXiv preprint arXiv:1506.01497*, 2015.

[33] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[34] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2117–2125.

[35] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick, "Detectron2," https://github.com/facebookresearch/detectron2, 2019.

[36] A. Kirillov, K. He, R. Girshick, C. Rother, and P. Dollár, "Panoptic segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 9404–9413.

[37] R. A. Güler, N. Neverova, and I. Kokkinos, "Densepose: Dense human pose estimation in the wild," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7297–7306.

[38] D. Acemoglu and P. Restrepo, "Artificial intelligence, automation and work," National Bureau of Economic Research, Tech. Rep., 2018.

[39] A. Nuvolari, "Understanding successive industrial revolutions: A "development block" approach," *Environmental Innovation and Societal Transitions*, vol. 32, pp. 33–44, 2019.

[40] G. Wisskirchen, B. T. Biacabe, U. Bormann, A. Muntz, G. Niehaus, G. J. Soler, and B. von Brauchitsch, "Artificial intelligence and robotics and their impact on the workplace," *IBA Global Employment Institute*, vol. 11, no. 5, pp. 49–67, 2017.

[41] D. Acemoglu and P. Restrepo, "Automation and new tasks: how technology displaces and reinstates labor," *Journal of Economic Perspectives*, vol. 33, no. 2, pp. 3–30, 2019.

[42] K. Lane, "Api 101: What is a rest api?" Aug 2020. [Online]. Available: https://blog.postman.com/rest-api-definition/

[43] B. B. Rad, H. J. Bhatti, and M. Ahmadi, "An introduction to docker and analysis of its performance," *International Journal of Computer Science and Network Security (IJCSNS)*, vol. 17, no. 3, p. 228, 2017.

[44] T. Fernandez, "What is canary deployment?" Sep 2020. [Online]. Available: https://semaphoreci.com/blog/what-is-canary-deployment

[45] S. Pradeep and Y. K. Sharma, "A pragmatic evaluation of stress and performance testing technologies for web based applications," in *2019 Amity International Conference on Artificial Intelligence (AICAI)*. IEEE, 2019, pp. 399–403.

[46] C. Patel, A. Patel, and D. Patel, "Optical character recognition by open source ocr tool tesseract: A case study," *International Journal of Computer Applications*, vol. 55, no. 10, pp. 50–56, 2012.

[47] J. Brownlee, "What is meta-learning in machine learning?" Dec 2020. [Online]. Available: https://machinelearningmastery.com/meta-learning-in-machine-learning/

[48] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *arXiv preprint arXiv:1912.01703*, 2019.

[49] Pytorch, "Torchvision." [Online]. Available: https://pytorch.org/vision/0.8/index.html

[50] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[51] I. Culjak, D. Abram, T. Pribanic, H. Dzapo, and M. Cifrek, "A brief introduction to opencv," in *2012 proceedings of the 35th international convention MIPRO*. IEEE, 2012, pp. 1725–1730.

[52] Streamlit, "Welcome to streamlit." [Online]. Available: https://docs.streamlit.io/en/stable/

[53] COCO. [Online]. Available: https://cocodataset.org/#home

[54] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese, "Generalized intersection over union: A metric and a loss for bounding box regression," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 658–666.

[55] D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov, "Scalable object detection using deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 2147–2154.

[56] A. Casado-Garcıa and J. Heras, "Ensemble methods for object detection," 2020.

[57] M. Cummings, "Automation bias in intelligent time critical decision support systems," in *AIAA 1st Intelligent Systems Technical Conference*, 2004, p. 6313.

Appendices

A. Predicting outliers with final model

In order to test characteristics of "outlier" samples that were removed from the dataset, and how they impact model performance, the best performing large model from Chapter 5 is used to predict only the outliers and measure the results.

		Precision	Recall
0.05	**0.25**	0.404540	0.964683
0.10	**0.25**	0.425630	0.960077
0.15	**0.25**	0.439809	0.955086
0.20	**0.25**	0.447805	0.943570
0.25	**0.25**	0.457911	0.935509
0.30	**0.25**	0.473366	0.931286
0.35	**0.25**	0.488620	0.931286
0.40	**0.25**	0.512304	0.927063
0.45	**0.25**	0.541216	0.922457
0.50	**0.25**	0.568978	0.916699
0.55	**0.25**	0.591294	0.912476
0.60	**0.25**	0.611082	0.897505
0.65	**0.25**	0.638950	0.887908
0.70	**0.25**	0.684022	0.877543
0.75	**0.25**	0.726036	0.860653
0.80	**0.25**	0.767058	0.802687
0.85	**0.25**	0.792326	0.768906
0.90	**0.25**	0.810664	0.688676
0.95	**0.25**	0.828305	0.579655

Figure A.1: *Precision- and recall-scores for the predicted outliers show performance is indeed worse at all confidence intervals for IoU-threshold=0.25, as compared to results in Table 5.4.*

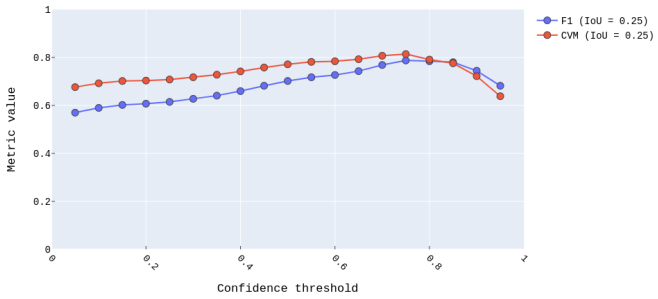


Figure A.2: *Both F1-scores and CVM-scores are significantly decreased for outliers as compared to Figure 5.4.*

B. Economic calculations

Labor costs per year	Factor	NOK
Gross early wage	520000	520000
Holiday pay	12,50 %	65000
Employer's tax	14,10 %	73320
Pension contributions	4 %	20800
Sum		**679120**

Working hours per year	
Working days per year	230
Working hours per day	7,5
Working hours per year	**1725**

Price per working hour in NOK	394

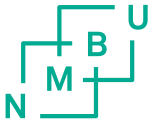
Number of pages	10000	
Number of bounding boxes per	1,785	
Addition for adding bounding boxes, per	6	seconds
Rate of false negatives	2,10 %	
Addition for removing bounding boxes, per	2	seconds
Rate of false positives	2,60 %	

Redaction time without the algorithm	Seconds per page
Baseline	12
Addition for adding bounding boxes	10,71
Sum seconds per page	22,71
Working hours per 10000 pages	63
Costs in NOK	24835

Redaction time with the algorithm	Seconds per page
Baseline	12
Addition for adding bounding boxes	0,22
Addition for removing bounding boxes	0,093
Sum seconds per page	**12,32**
Working hours per 10000 pages	34,22
Costs in NOK	**13471**

	Hours	NOK
70000 pages without the algorithm	442	173848
70000 pages with the algorithm	240	94294
Implementation costs	115	45275
Maintenance	36	14173
Savings first year		**20107**
Savings following years		**65381**

C. Performance monitoring dashboard



Norges miljø- og biovitenskapelige universitet
Noregs miljø- og biovitskapelege universitet
Norwegian University of Life Sciences

Postboks 5003
NO-1432 Ås
Norway