Tilburg School of Economics and Management
Tilburg University

# Deep Active Learning for Automated Damage Assessment after a Natural Hazard

*By:*
Polle Dankers
SNR: 2013741

*Supervised by:*
dr. Marleen Balvert
Robin Swinkels (Pipple)
dr. Jacopo Margutti (510)
dr. Marc van den Homberg (510)

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Business Analytics and Operations Research

June 8, 2023

# Abstract

This thesis aims to improve automated damage assessment by using active learning. Automated damage assessment involves the application of machine learning techniques to rapidly estimate the extent of building damages following a natural hazard. 510, an initiative of the Netherlands Red Cross, employs a model for automated damage assessment using satellite imagery. A challenge is that no labelled images, which can be used to train the model, are available from a newly occurred natural hazard. Labelling a large amount of data is too time-consuming and labour-intensive. Active learning aims to select those datapoints to be labelled by annotators which are the most informative for a model, such that the model can be trained using only a small labelled dataset. In this use-case, it is implemented after having pre-trained a model on available labelled data from previously occurred disasters. The objective is to fine tune the pre-trained model to perform well on a newly unfolding disaster without having to label a large amount of data.

Three different active learning models, which all have different advantages, are implemented: Bayesian Active Learning by Disagreement (BALD), BatchBALD and Wasserstein Adversarial Active Learning (WAAL). WAAL also uses adversarial training, where a part of the network is trained using unlabelled data as well. The methods are compared to randomly selecting datapoints.

Fine-tuning the pre-trained model using a small amount of labelled data from the new disaster resulted in large performance improvements. However, none of the active learning techniques managed to outperform random selection, with WAAL even performing significantly worse. Thus, while 510 should implement the fine-tuning of pre-trained models, it is recommended to use randomly selected data.

## Acknowledgements

Firstly, I would like to thank Pipple and 510 for enabling me to write this thesis, as well as the Zero Hunger Lab which I could join later. The interesting subject and combination of different organizations made writing the thesis much more manageable.

Further, I would like to thank each of my supervisors, Robin Swinkels, Jacopo Margutti, Marc van den Homberg and Marleen Balvert for their help with both the content and structuring of the thesis. Lastly, I want to thank Sanne van den Bogaart for her help when getting started with the thesis and Cascha van Wanrooij for advising on some programming matters.

# Contents

# 1 Introduction

In 2022, over 30 thousand lives were lost due to natural hazards. On top, the lives of 185 million people were effected by these hazards and they resulted in economic losses of over 220 billion Dollars (CRED, 2020). Damages due to natural hazards have been increasing over the years, and are expected to increase further due to climate change and an increased number of people living in areas prone to disasters (Laframboise and Loko, 2012). Since the 1990s, there has been a 35 percent increase in the number of climate and weather-related disasters (IFRC, 2020). While the occurrence of hazards cannot be avoided, well coordinated aid can reduce the impact of them.

Organisations such as the Red Cross provide aid when a disaster strikes. To target this aid towards the locations where it is needed the most and to secure sufficient funding for the aid, it is important to assess where the disaster caused damage and how large the damage is. A common method for this is conducting field surveys. Typically, the assessment of damages has multiple phases. First, a rapid assessment for emergency needs is done within a week, using limited field visits. After this a more detailed assessment is conducted to obtain a more detailed overview of damages. This typically takes up to 4 weeks (ICRC and IFRC, 2008). Even this detailed post disaster needs assessment often has limited field visits (Jeggle and Boggero, 2018). Due to limited resources, as well as limited accessibility of damaged areas, it is generally impossible to visit the whole affected area.

One possibility to conduct damage assessments more efficiently is using remote sensing technologies such as drones and satellites. Remote sensing techniques have previously been used to identify land perturbations such as landslides. Lately however, its usage for damage assessments has been increasingly studied (Lallemant et al., 2017). This research will use satellite images, as they are already collected and cover a large area.

The information extracted from satellite pictures can assist relief work after disasters in various ways. It can help in decisions on deploying international search and rescue teams, identifying unknown damaged areas, coordinating the aid response, decisions on sending international aid and deploying resources and to identify housing requirements as well (Barrington et al., 2011). Conducting damage assessments can be done visually by either experts or crowd-sourcing. Crowd-sourced damage assessments are faster but less accurate as damages are not assessed by experts (Westrope et al., 2014). Even using crowd-sourcing, assessing all buildings in affected areas manually can be time consuming, while a big advantage of using satellite data could be the speed of the initial damage assessment of buildings. On top, a large amount of volunteers needs to be available.

Automated damage assessment (ADA) algorithms can speed up damage assess-

ments by having a computer estimate damages based on the satellite imagery shortly after a disaster occurs, while it also decreases the dependence on the availability of volunteers. The newest ADA algorithms typically use convolutional neural networks (CNNs). 510, an initiative of the Netherlands Red Cross for which this thesis is written, has constructed such a model. As input, the model receives a satellite image of a building from before and and after a disaster and it returns a damage classification. While results differ per disaster, they are promising when the algorithm is trained on labelled data from the examined disaster itself. In a practical situation, such training is not possible since it takes too much time to label sufficient data to train a model. Using models trained on other previously occurred hazards, of which labelled images are available, have less predictive capabilities, though in some cases reasonable results are achieved (Valentijn et al., 2020). Results are likely to depend on the similarity of the type of damage and location between both the trained on disasters which previously occurred, and the newly occurred disaster on which the model is used.

Given training the model on imagery from the newly occurred disaster results in better performance, 510 researches the applicability of active learning (AL) for automated damage assessment. AL is a sub-field in machine learning where an algorithm selects unlabelled data to be labelled by an 'oracle', which can e.g. be a human annotator. The main idea is to make the active learner, which is the model selecting which images it wants to be labelled, ask images in an efficient way such that the model can achieve a high performance with limited labelled instances (Settles, 2009).

Previously a core-set approach focusing on exploring diverse types of images has been implemented (Sener and Savarese (2017) and Van den Bogaart (2021)), but this did not lead to good results. Focusing on a diverse representation of data may result in selecting a large amount of images which the model finds easy to classify already. Another method, focusing on model uncertainty, showed some promise (Van den Bogaart, 2021). Wang (2021) similarly shows promising results when using uncertainty based AL for ADA. Furthermore, she shows using AL with a model pre-trained on labelled data from different disasters improves results compared to solely training a model on new data using AL.

In this thesis, multiple AL methods are therefore implemented which aim improve the previously tested uncertainty methods. Bayesian AL methods are used for more reliable estimates for uncertainty. One of these Bayesian methods takes into account the overlap in information which similar images may have. Lastly, a different type of AL is used which more explicitly combines uncertainty and diversity. On top, it uses unlabelled data when training the model. All methods are explained in more detail later.

In Section 2, a literature overview containing in-depth information on automated damage assessment, convolutional neural networks and transfer learning is given. Furthermore, the basics of active learning are explained. The used data is introduced in Section 3. Next, Section 4 contains the methodology with an elaborate explanation of the used Active Learning methods, as well as explanations on how transfer learning is implemented and the used performance metrics. It also contains explanations on hyperparameters which must be tuned, of which the results can be found in Section 5. The results comparing the different AL methods are given in Section 6, followed by the discussion and conclusion in Sections 7 and 8 respectively.

# 2 Literature overview

## 2.1 Automated damage assessment

Automated damage assessment (ADA) is a research area which focuses on mapping the magnitude of damage after natural hazards. By creating models which conduct such assessment automatically, it has the potential to quickly provide organisations with a overview of damages and needs across an affected area.

In an early application, Naeim et al. (2006) predict the damage to buildings after an earthquake using sensor and structural data with a probabilistic approach. Using this method, damage prediction can be made at a floor level. However, this approach can only be used for earthquakes and needs buildings to be equipped with sensors, which generally is not the case. Hence this type of methods cannot be applied for ADA in most cases.

### 2.1.1 Using aerial imagery for damage assessment

Another option that does not depend on having sensors installed in a building is using aerial imagery, e.g. by using footage produced by news vendors (Ozisik and Kerle, 2004). Given the high quality of images from such sources, researchers were able to conduct ADA based on intensity and color measures to classify buildings in the year 2000 already. The findings in such studies show that ADA does have the potential to be a useful addition to traditional building damage assessment (Hasegawa et al. (2000), Mitomi et al. (2001) and Yamazaki (2001)). However, aerial images of news vendors are not easily accessible for large areas and the automation methods used were not sophisticated.

Due to the improved accessibility and quality of unmanned aerial vehicles (UAVs), often referred to as drones, the usability of aerial imagery for damage assessment has increased lately. Since drones can circle around buildings, three dimensional pointclouds of buildings can be created. Using these pointclouds, detailed assessments of damages can be conducted (Fernandez Galarreta et al., 2015). Damage assessment with these pointclouds can be automated using s-support vector machines and random forests. Using the random forest, 95 percent of the damaged regions could be identified. However, when models were trained on data from one site and tested on other unseen data, the accuracy drops by 15-30 percent (Vetrivel et al., 2015). Performance on unseen data can be improved by combining features from such a 3D pointcloud with features extracted from the original UAV images using a convolutional neural network (CNN, discussed in detail in the next subsection). Using this combination, the transferability of models is increased and an average accuracy of 85 percent is achieved (Vetrivel et al., 2018). Using UAVs, it is even possible to form near real-time damage maps. A UAV

can be sent on a pre-defined flight plan, while automatically creating a map by combining images. ADA can be performed on this map using a combination of algorithms, amongst which a CNN. A damage map is automatically constructed by the algorithm before the UAV has even landed (Nex et al., 2019).

Hence UAVs can be useful in damage assessment after a disaster. This is especially the case for search and rescue teams by using the automated near real-time damage mapping. Whilst being helpful in the disaster response, UAVs have some drawbacks as well. Their usage is often limited by legislation. Furthermore, atmospheric conditions can lead to unforeseen behaviour of a drone. Most importantly, UAVs can only cover a limited area since they have a short battery life and hence small operating range (Fernandez Galarreta et al. (2015) and Kerle et al. (2019)). There is the need for UAVs to be available at the location of the disaster as well.

### 2.1.2 Using satellite imagery for damage assessment

Due to these limitations, other methods are needed for damage assessment in larger areas. For this, satellite imagery can be used. Though satellite imagery does face some drawbacks such as only having images from the top of the building, clouds which may interfere with capturing useful images and a reliance on satellite companies making images available, there are clear advantages of using satellite images. They are collected automatically, removing the need for UAVs at the location. Furthermore, images collected by satellites map the whole impacted areas, providing a better coverage for large disasters.

Satellite images have been used in disaster management for a long time, e.g. for the mapping of areas affected by floods in 1981. They are especially important when the area hit by a disaster is inaccessible (Jayaraman et al., 1997). While previously satellite images could only be used to identify large scale features such as landslides, the introduction of civilian Very High Resolution (VHR) satellites has made it possible to identify damages to smaller structures such as single buildings as well (Van Westen, 2000). Assessing damages to buildings can be done by professional analysts who visually inspect satellite images and classify structures. Since the availability of such professionals is limited, volunteers are also used to map damages (Kerle and Hoffman, 2013). While such damage assessments can provide important information for relief organisations, it currently takes multiple weeks to process the images after a disaster even while using volunteers (Barrington et al., 2011).

Therefore, soon after VHR satellite images became more available, research has focused on automating the damage assessment based on these images. Al-Khudhairy et al. (2005) used eCognition, which groups pixels forming objects and classifies them using fuzzy logic. This assigns a value between 0 and 1 for each possible

class, related to how likely it is that the object belongs to the respective class (Flanders et al., 2003). Huyck et al. (2005) use dissimilarities between pre and post disaster images after some processing steps, amongst which convolutional filters which are explained in the next subsection. Both show that ADA using satellite images is promising, but needs more research to be applicable in practice. Other studies focused on methods such as linear relations, rule-based classification using differences of shadow patters before and after disasters and one-class support vector machines (Dell'Acqua and Polli (2011), Tiede et al. (2011) and Li et al. (2010)).

More recent research into ADA using satellite images has focused on the usage of CNNs. These are able to teach itself to recognize features in pictures, and classifies images based on these extracted features. Vetrivel et al. (2016) found that CNN features significantly outperform handcrafted features. CNNs are also found to outperform other methods such as SVM, classification and regression trees and Random Forests by more that 10 percent (Ma et al. (2020) and Berezina and Liu (2022)).

Using images from both before and after a disaster can improve results as well. Shao et al. (2020) find that combining both images instead of only using the post disaster image results in a close to double F1-score. The F1-score is a metric used for image classification which will be further discussed in Section 4.3. In particular, results improve when images from before and after a disaster are first processed through separate CNNs, after which the output of both is jointly used for classification (Xu et al. (2019) and Yang et al. (2021)). 510 has created such a CNN, which will be discussed in more detail in Section 2.2.7.

While CNNs provide promising results for ADA, most research is conducted by training and testing models on the same disaster event. The practical applicability of such models is limited, since CNNs typically need large amounts of data to train and hence such models would need many images to be labelled by hand. This is not possible within the limited time after a disaster in which the analyses should be conducted. Therefore some research looked into the transferability of models between disasters. This involves the training of a model using labeled images from one or more previous disasters and predicting an unseen disaster using this model. While in some cases reasonable results are found, especially when a model was trained on multiple unseen disasters, such models always perform worse than models trained on the same disaster it is used on (Valentijn et al. (2020) and Xu et al. (2019)). This thesis will focus on active learning for the CNN based ADA model created by 510.

## 2.2 Convolutional Neural Networks

Note that overviews on CNNs by Goodfellow et al. (2016), Albawi et al. (2017), O'Shea and Nash (2015) and Gu et al. (2018) are used throughout this subsection, without explicitly referring to them each time.

CNNs are neural networks that use multiple different types of layers, amongst which convolutional layers, that allow the model to extract features from the data and make predictions based on them. The first well known CNN frameworks, LeNet-5 (LeCun et al., 1998) and AlexNet (Krizhevsky et al., 2017), first use a combination of convolutional and pooling layers. Together, these layers extract features from data while keeping the size manageable. These layers will be explained in detail later in this subsection. After several convolutional and pooling layers, typically the extracted features are flattened and then fed to dense fully connected layers. Figure 1 visualizes an example of a simple CNN with two convolution layers followed by pooling, which is connected to a dense fully connected layer.
As the network is able to extract and recognize certain features, it provides a high performance in image recognition. All aforementioned layers will be explained next. These layers are the basic building blocks of a CNN, upon which more sophisticated CNN models such as the one used by 510 are based.

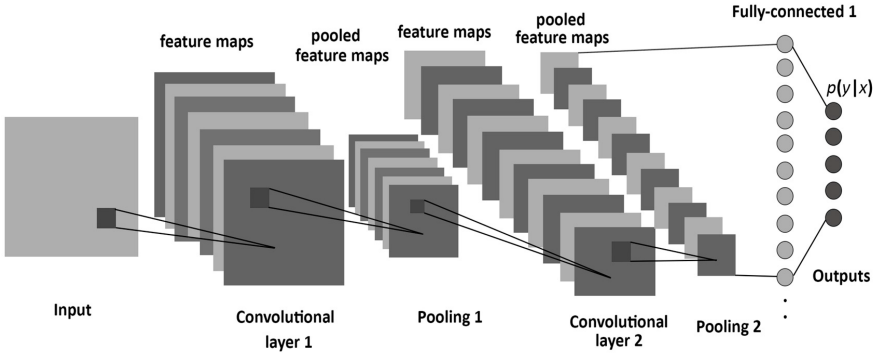



Figure 1: Basic structure of a CNN (source: Albelwi and Mahmood (2017)).

### 2.2.1 Convolutional layer

The input to the first convolutional layer, such as in Figure 1, is generally an image represented by a matrix with pixel intensities or red, green and blue (RGB) values. For convolutional layers following this first one, the output matrices of previous layers, called feature maps, are used as input. A convolutional layer uses a filter, sometimes referred to as kernel, to extract features from the image. The filter is a block of weights, whose dimensions are chosen by the researcher. The filter is moved over the input matrix. At each position, the element-wise product

is computed between the filter and the part of the input matrix considered, which has the same dimensions. The entries of the resulting matrix are summed and this sum is the entry in the relevant position in the feature map, sometimes referred to as the convolved image. In Figure 2 this process is visualized. The number of rows or columns which the filter moves can be adjusted and is referred to as the stride. Sometimes padding is used, which are rows and columns containing only zero values around the original image. Padding can prevent a decrease in the spatial resolution of an image and can be used such that more information on the edge of the image is detected. Without it, less filters would be applied to these edge values compared to those on the interior of the image.

A two-dimensional convolution can be computed as follows (Liquet et al. (2023) and Goodfellow et al. (2016))[1]:

$$S_{i,j} = (K * I)_{i,j} = \sum_m \sum_n I_{a(i,m),b(j,n)} K_{m,n} \ , \tag{1}$$

where $S_{i,j}$ is the entry in row $i$ and column $j$ of the feature map. $I$ is the input image and $K$ is the kernel, which both are matrices. $m$ and $n$ range over the dimensions of the kernel. $a(i,m)$ and $b(j,n)$ are functions that determine the input indexes of the image $I$. Generally this these functions are simply defined as $i+m$ and $j+n$ (Liquet et al., 2023), but this can be altered e.g. to incorporate a stride larger than one. When calculating the full convolution, this function is applied for every combination of $i$ and $j$, with $i$ and $j$ ranging over the number of rows and columns of the feature map respectively.

Convolution has several advantages. By using a filter whose weights are the same for each region it is applied to, the number of weights that need to be trained remains relatively small. Furthermore, by applying the same filter to different regions, features can be recognized anywhere within the image. By applying several filters in parallel, the network can learn to recognize different types of features, such as edges (Albawi et al., 2017). By combining multiple layers with filters, the model can learn complicated structures.

### 2.2.2 Activation Function

Convolution is generally combined with a non-linear activation function. Sometimes this step is considered to be a separate layer: the non-linear layer. It introduces nonlinear capabilities to the network (Hao et al., 2020). Without such

[1]Mathematically, a convolution is defined slightly different. The correct mathematical formulation is less easy to apply and mostly used for proofs. When implementing a convolution filter in practice, the formulation as stated here is generally used, which strictly speaking is a cross-correlation formula. For CNNs, the difference between the cross-correlation and convolution functions is not relevant for its performance (Goodfellow et al., 2016).

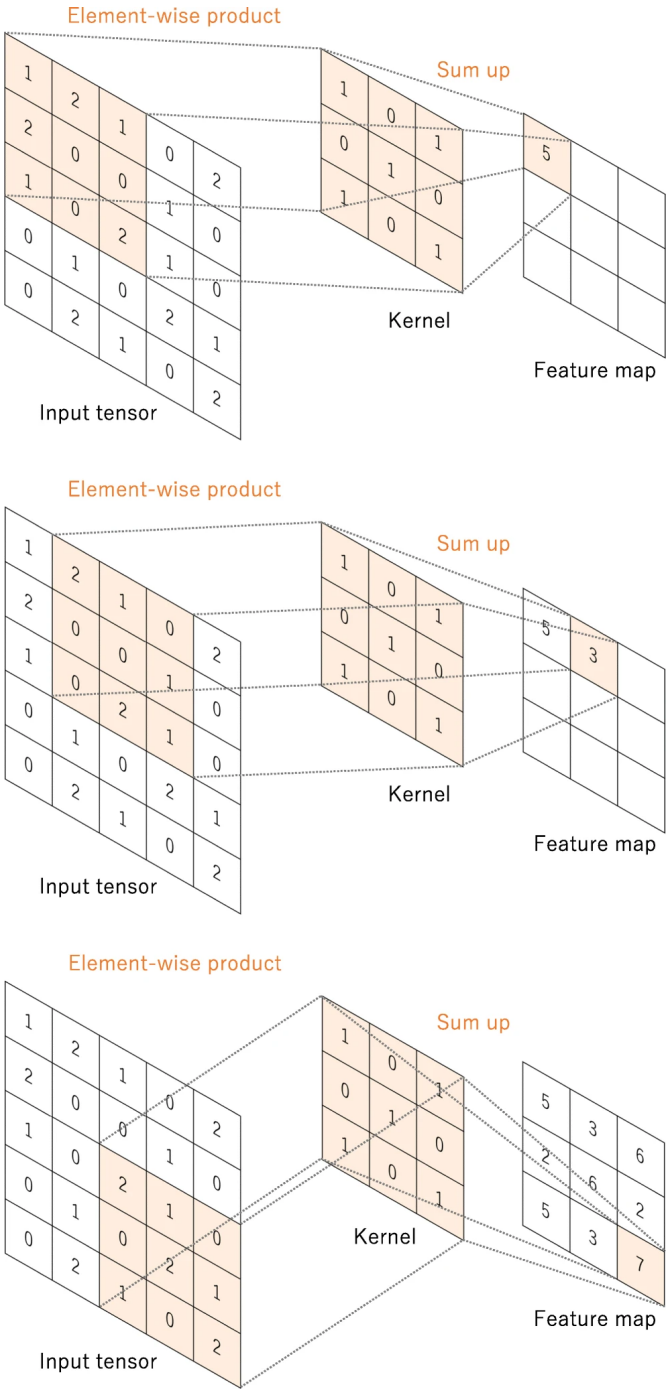



Figure 2: Visualization of a convolution using a filter with stride equal to one and no padding, shifting through the first two and last element-wise products (source: Yamashita et al. (2018)).

a function, the network would not be able to detect non-linear features (Gu et al., 2018). While for a long time the sigmoid function (see equation (2)) has mostly been used for CNNs, recently rectified linear unit (ReLU) has become more popular. ReLU has a simple definition, see equation (3). Furthermore, when a model is trained using backpropagation, which will be explained later in this section, functions such as sigmoid suffer from the vanishing gradient problem. This especially becomes a problem in deeper neural networks. The problem is caused by the sigmoid function having a gradient close to zero when values are not close to zero. ReLU does not suffer from this problem since its gradient is constant for positive input (Albawi et al., 2017). In Figure 3, both functions are shown to clarify this difference. ReLU also has a low computational cost and its usage results in fast learning (Nwankpa et al., 2018). Hao et al. (2020) and Mishkin et al. (2017) both show that ReLU and activation functions closely related to ReLU perform significantly better than sigmoid.

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

$$ReLU(x) = max(0, x) \tag{3}$$

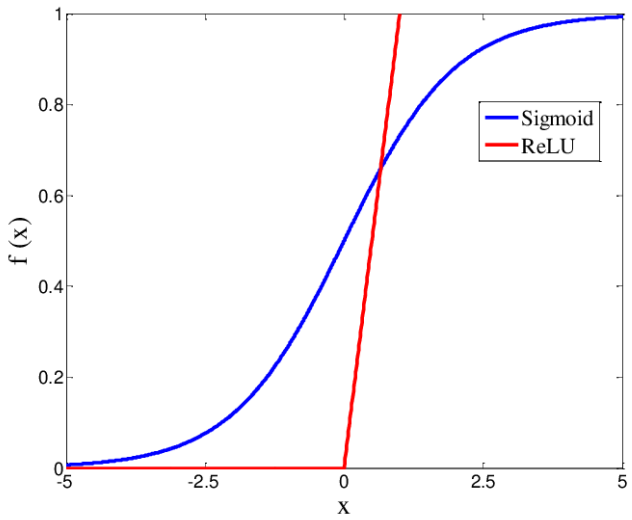



Figure 3: Comparison ReLU and sigmoid function (source: Sun et al. (2019)).

A bias is generally added to the input of the activation function. This bias is a learnable parameter. In regular neural networks, biases are connected to a single neuron, where for each neuron a separate bias is learned by training. When using CNNs, one bias term is used for each kernel. Hence the convolution step does not only decrease the number of weights to be learned but also the number of biases, decreasing the number of parameters to be learned (Nielsen, 2015). The bias can

be interpreted as a term shifting the graphs of the ReLU and sigmoid functions shown in Figure 3 to the left or right.

When the filter is assumed to be a square and an activation function and bias are added, equation (1) can be altered to:

$$S_{i,j} = (I * K)_{i,j} = f(\sum_{m=0}^{p-1} \sum_{n=0}^{p-1} I_{i+m,j+n} K_{m,n} + b) \tag{4}$$

Where $S$, $I$, $K$, $i$ and $j$ are defined as before. Function $f$ is the activation function, often ReLU. $p$ is the dimension of the filter, e.g. when a $3 * 3$ filter is used, $p = 3$. Lastly $b$ is the bias (Liu et al., 2019).

### 2.2.3 Pooling layer

Often, convolutional layers are followed by a pooling layer. Pooling is used to simplify the output from convolutional layers by decreasing the dimensions of the output. It summarizes regions of the feature map into a single unit (Nielsen, 2015). Pooling reduces the number of trainable parameters, which results in faster training. It also reduces overfitting. Multiple ways of pooling are commonly used (Zafar et al., 2022):

- Max-pooling: regions of the original feature map are summarized by the maximum value within the region.
- Average pooling: The region is summarized by taking the average over its entries
- Mixed pooling: combines average and max-pooling by randomly choosing between the two.
- $L_2$ pooling: the square root of the sum of squares of neurons in a $2 * 2$ region is used. This can be extended to $L_p$ pooling by using the $p$ and $\frac{1}{p}$ as exponents instead of squares and square roots.

In Figure 4, max-pooling is visualized. For each $2 * 2$ region without overlap, the largest value is used to insert into the pooled feature map.

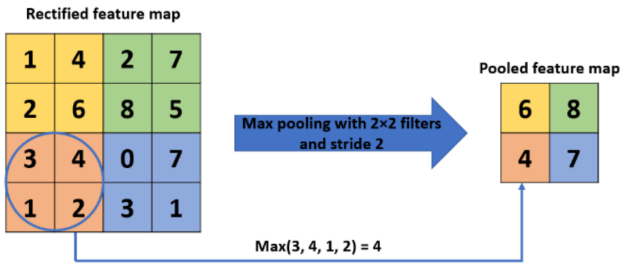



Figure 4: visualization of max-pooling (source: Gholamalinezhad and Khosravi (2020)).

### 2.2.4 Fully connected layer

After the combinations of convolutional and pooling layers, the output of high level features is flattened. Thus, the different resulting feature maps are reshaped to vectors and stacked onto one another, resulting into one large vector. This flattened vector is the input used for the fully connected layers. Fully connected layers are layers where all neurons from the previous layer are connected to all neurons in the next layer (Hijazi et al., 2015). It learns to correctly classify images based on the combination of features extracted by the convolutional layers. These fully connected layers generally contain most of the parameters within a CNN (Basha et al., 2020). Between fully connected layers, activation functions such as ReLU are used.

In the classification layer another type of activation function is used. Often, softmax (see equation (5)) is applied. The function assigns numbers in the range of $[0, 1]$ to each possible class, which together sum up to one. The outcome with the highest number is predicted.

$$Softmax(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}} \tag{5}$$

In the equation, $Softmax(x)_i$ is the score for class $i$. $x_i$ is the sum of the product of weights connected to output node $i$ with the output of the previous layer, with the bias added to it. Hence $x_i = w_i^T z + b_i$ with $z$ a vector containing the output of the previous layer, $w_i$ a vector with weights between this output and class $i$ and $b_i$ the bias for class $i$. $J$ is the total number of classes the model can predict. In Figure 5, the flattening of data and fully connected layers are visualized using one feature map.

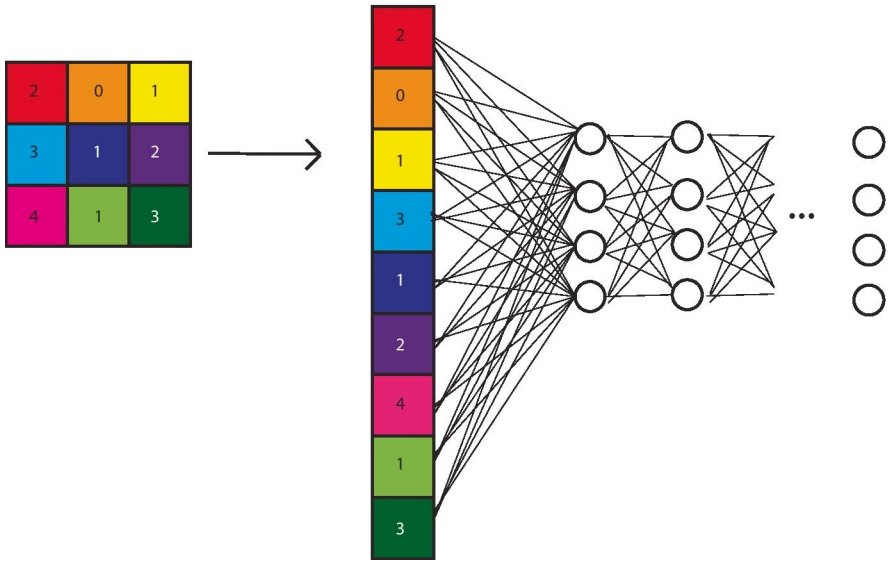



Figure 5: Flattening and fully connected layers (source: Srivastava et al. (2014)).

### 2.2.5 Training of a CNN

Before discussing the training of a CNN, it is important to clearly distinguish between the terms parameters and hyperparameters, which are both important in training. Parameters refers to the weights and biases which the model should learn. The hyperparameters refer to different settings used in training, set by the researcher. For the training of a CNN, labelled data is needed. These are e.g. images for which the class is known. These images are split in a train, validation and test set. The train set is used to learn the parameters. The validation set is used to evaluate the performance of models on unseen data while training by applying the model to the validation data. This is used to select hyperparameters and to assess whether the model is overfitting after a certain amount of epochs. Epochs are a type of training iterations which will be explained later in this section. The test set is only used at the end after a model is constructed and used to give a reliable estimation of the error in the model (Goodfellow et al., 2016).

Given the training set with labelled images, the CNN needs to know what a good prediction is. For this, a loss function is used. For classification, usually cross-entropy loss is used. When the model predicts the correct class with high certainty, the output of the function is small. Predicting a class with high certainty for a given datapoint refers to a large predictive value for the respective class, while the predicted values for the other classes are small[2]. When the model contrarily predicts the wrong class with high certainty, the output is large. The model aims to minimize the cross-entropy loss (Zhang et al. (2021) and Nielsen (2015)):

$$CE(y, \hat{y}) = -\sum_{i=1}^{N}\sum_{j=1}^{K} y_{ij} log(\hat{y}_{ij}) + (1 - y_{ij}) log(1 - \hat{y}_{ij}) \tag{6}$$

Where $i$ denotes the image and $N$ the number of images used. $j$ denotes the class and $K$ the number of possible classes. $y_{ij}$ equals one if image $i$ belongs to class $j$ and $\hat{y}_{ij}$ is the prediction made by the model for image $i$ and class $j$, which is the output of the softmax function in the output layer. Equation (6) can also be written with the images $x_i$ and parameters $\theta$ as input ($CE(x_i, \theta)$), which is useful when describing the learning procedure using gradients.

[2]For example, consider a model predicting four classes. Let there be two datapoints, where the output of the softmax layer of the first datapoint is 0.05/0.1/0.8/0.05, with these numbers being the predictive value of the four classes. The output of the second datapoint is 0.2/0.25/0.3/0.25. Both predict the third class, but the predictive value for that class is much larger for the first datapoint, of which the model is therefore considered to be more certain of it belonging to the thirs class. Thus, the cross-entropy loss will be larger for the second datapoint compared to the first datapoint if class three is indeed correct. If any of the other classes is the true class, the loss will be larger for the first datapoint as its predictive value would then be relatively small for the correct class. Note that this interpretation of predictive values as a measure of uncertainty is not entirely correct, which is discussed further in Section 2.3

The training of a CNN has several steps. Firstly, all parameters are initialized. Next an iterative process of updating weights and biases, together referred to as parameters and denoted by $\theta$, is started. First, feed forward propagation is used. This is simply using the current parameters to obtain prediction $\hat{y}$ for images. This prediction can next be used to calculate the loss. Next, a process called back propagation is started which calculates gradients in order to decrease the loss (Goodfellow et al., 2016).

Backward propagation starts from the output layer and moves backwards through the network, computing gradients in each layer by applying the chain rule (LeCun et al., 2015). Having calculated these gradients, parameters can be updated as follows, using (batch) gradient descent (Bottou et al., 1991):

$$\theta_{t+1} = \theta_t - \eta_t \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta CE(x_i, \theta_t) \tag{7}$$

Where $t$ denotes the iteration. $\theta_t$ are the current parameters, with $\theta_{t+1}$ the newly updated parameters. $\eta_t$ is the learning rate, which is a hyper parameter set by the researcher. It regulates the magnitude by which parameters are updated during a training step and is sometimes referred to as the step size. $\nabla_\theta CE(x_i, \theta_t)$ is the gradient of the parameters for a given image $x_i$. However, one step takes a lot of time since it involves computing the gradients using all images, which can also result in memory issues when using a large dataset. Therefore stochastic gradient descent (SGD) can be used, where one image $z$ is randomly chosen and the parameters are updated based on the gradient of this single image:

$$\theta_{t+1} = \theta_t - \eta_t \nabla_\theta CE(z, \theta_t) \tag{8}$$

Using this method, the model can be trained faster. Being only based on one image, the calculated gradients are not representative for the whole set. Therefore often a combination of gradient descent and stochastic gradient descent is used, where a mini-batch is randomly chosen which is a small sample of the training set. Next the gradient for this batch is calculated and used to update the parameters, see equation (9). This method is often referred to as mini-batch gradient descent, though it is sometimes referred to as stochastic gradient descent as well (Nielsen, 2015).

$$\theta_{t+1} = \theta_t - \eta_t \frac{1}{M} \sum_{i=1}^{M} \nabla_\theta CE(x_i, \theta_t) \tag{9}$$

$M$ is the size of the mini-batch. Training is done in multiple epochs. One epoch consists of randomly choosing mini-batches of images without replacement and updating parameters as described above, until all images have been used in a mini-batch.

Often more sophisticated update rules such as SGD with momentum, AdaGrad, RMSProp and Adam are used which solve different problems that may occur

when using the more simple versions of SGD. One of these problems is caused by an ill-conditioned Hessian[3]. This leads to large oscillation of the gradients, making convergence slow. Momentum is used to solve this issue by using previous gradients in the current updating step. Another problem is caused by sparse features. This can lead to the optimization being too sensitive to features with large values. Adagrad and RMSprop solve this issue by changing the learning rate for each parameter based on past gradients computed for that parameter. When this gradient is large, the learning rate is small and vice versa. Adam combines the advantages of all these methods, hence solving both issues. Using Adam, weights are updated as follows:

$$\begin{aligned} g_{t+1} &= \frac{1}{M}\sum_{i=1}^{M}\nabla_\theta CE(x_i,\theta_t) \\ v_{t+1} &= \frac{1}{1-\beta_1^t}(\beta_1 v_t + (1-\beta_1)g_{t+1}) \\ s_{t+1} &= \frac{1}{1-\beta_2^t}(\beta_2 s_t + (1-\beta_2)g_{t+1}^2) \\ \theta_{t+1} &= \theta_t - \frac{\eta_t v_{t+1}}{\sqrt{s_{t+1}}+\epsilon}g_{t+1} \end{aligned} \tag{10}$$

Here $v_t$ solves the problem with an ill conditioned Hessian, while $s_t$ solves issues related to sparse features. $t$ denotes the iteration and $g_t$ is the gradient computed using mini-batches. $\beta_1$ and $\beta_2$ are hyperparameters controlling the decay rate, set close to 1. $\epsilon$ is a hyperparameter, which is usually small, used such that division by zero is impossible. $\eta_t$ is the learning rate (Kingma and Ba (2014), Ruder (2016) and Balvert (2021)).

### 2.2.6 Dropout

One possible problem that can occur when training a CNN, or neural networks in general, is overfitting. When this happens, the model is adapting too much to the training data used without generalizing well on new data. Overfitting can be identified when the loss of a model is decreasing while training, while the accuracy of the model applied to the validation set does not increase or even decreases (Nielsen, 2015).

Overfitting can be reduced in multiple ways, with the most straightforward way increasing the amount of training data. Collecting additional training data is costly and therefore this is often not feasible. In the model used by 510, dropout is

[3]The Hessian is ill-conditioned when derivatives increase fast in one direction but slow in another direction. This can result in no significant progress in the directions where the derivative only increases slowly (Goodfellow et al., 2016).

therefore implemented[4]. This method randomly drops neurons in a neural network along with all corresponding connections. The dropped neurons are temporarily removed from the network. Figure 6 shows this, with the crossed out neurons representing the randomly selected neurons which are removed. For each mini-batch, dropout is randomly performed and hence the model being trained changes for each training step. When using dropout, the tunable hyperparameter $p$ is introduced. $p$ is the probability of retaining a neuron in the network. A large $p$ means only few neurons are dropped, while a small $p$ results in only few neurons being turned on while training. For neurons in hidden layers, typically $0.5 \leq p \leq 0.8$ is used (Srivastava et al., 2014).

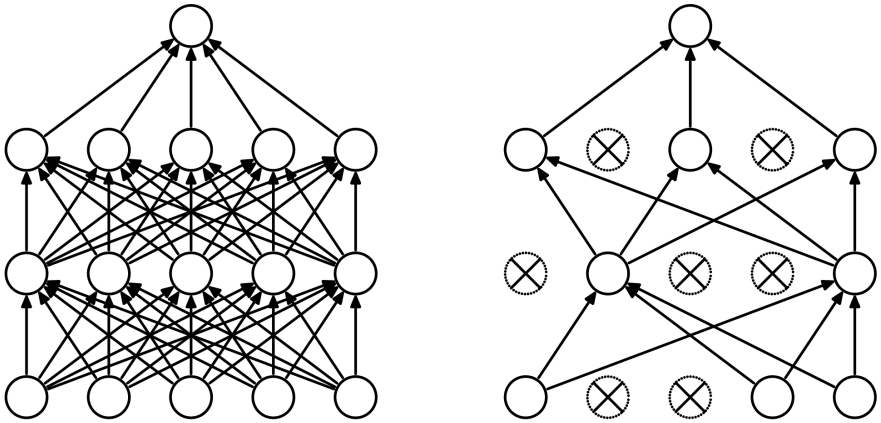


Figure 6: Comparison NN with and without dropout (source: Srivastava et al. (2014)).

### 2.2.7 CNN model 510

To perform automated damage assessment, 510 has created a CNN called Caladrius. The model is described by Valentijn et al. (2020), on which this subsection is largely based.

The architecture of Caladrius is inspired by Siamese NNs. This type of NNs consist of first two distinct networks which both are given an image, or other data, as input. The parameters in both networks are shared. The output of the networks are then compared to make predictions (Bromley et al. (1993) and Koch et al. (2015)). Since satellite imagery is available from before and after disasters, comparing these images is possible in the automated damage assessment application. While usually parameters are shared between the two networks in a Siamese NN, this is not the case in Caladrius as images before and after a disaster have distinct

[4]Other regularization techniques, such as L1 regularization and data augmentation, can be used to reduce overfitting as well. Given that these methods are not implemented, they will not be discussed further.

features which should be extracted. Furthermore, instead of immediately predicting the outcome after both networks, the Caladrius model implements multiple fully connected layers between the two CNNs and classification. These layers can learn which features, or combinations of features, indicate certain damage classes (Valentijn et al., 2020). In Figure 7, the network architecture is visualized with first the two separate CNN blocks, which are explained in more detail after the figure, followed by fully connected layers.

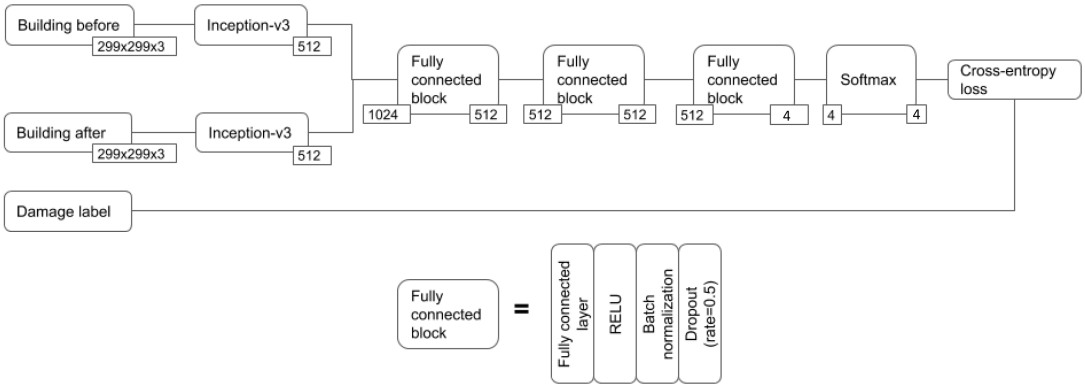



Figure 7: Network design CNN Caladrius (source: Valentijn et al. (2020)).

As shown in Figure 7, the input consists of images of buildings with dimension $299 * 299 * 3$. In Section 3, these will be elaborated upon further. An Inception-v3 CNN is used for both images. It is an advanced CNN mainly consisting of inception modules. These modules consist of parallel combinations of convolution and pooling layers with varying filter sizes, stride values and padding values. Using these different convolutions, the model can learn features of different sizes. The created feature maps within an inception module are concatenated. The network design keeps the computational cost manageable (Szegedy et al. (2015), Szegedy et al. (2016) and Ding et al. (2019)). In Figure 8 the architecture of Inception-v3 is shown with the different types of inception modules used.

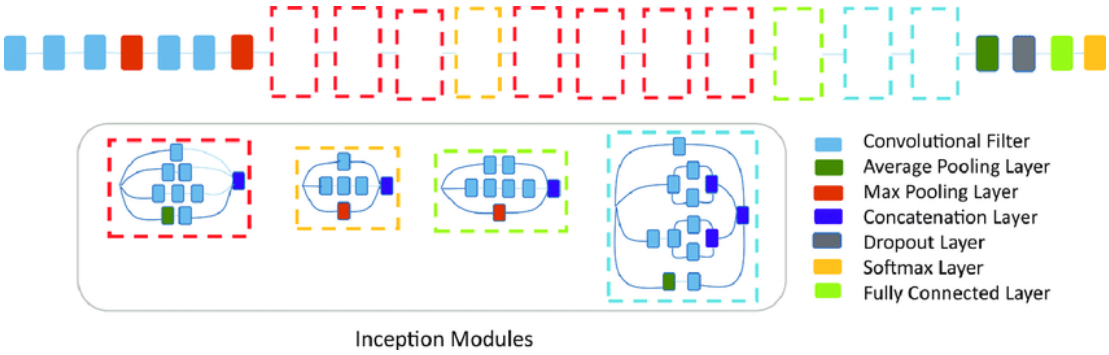



Figure 8: Inception-v3 architecture (source: Ding et al. (2019)).

From each Inception-v3 CNN, a vector with 512 features is extracted. These are concatenated into a vector with the 1024 features from both CNNs. This vector next goes through three fully connected layers, which all use a ReLU activation function, batch normalization to speedup training (Ioffe and Szegedy, 2015), and dropout with $p = 0.5$. Lastly the softmax function is used to create predictions for four different classes, into which the damage label is predicted. Cross-entropy is used for the loss function (Valentijn et al., 2020).

The Inception-v3 CNNs are pre-trained on ImageNet as this improves accuracy. Due to the pre-training, the model already knows how to recognize some features (Marmanis et al., 2015). The full model is trained using the Adam optimizer (Valentijn et al., 2020). The batch size is set to 32, while the number of epochs used for training is elaborated upon further in Section 4.4.1.

## 2.3 Measuring uncertainty in neural networks

In deep learning models such as the CNN by 510, the predictive probabilities do not represent the uncertainty of the model. These predictive probabilities are the output of the softmax layer. Often deep learning models are only used to predict the right class, without placing any importance on the uncertainty of the model. However, when uncertainty is important, such as in Active Learning which is further introduced in Section 2.5, this can become a problem.

Specifically for instances which are more dissimilar from the training data, the predictive probabilities can differ considerably from the uncertainty of the model (Gal and Ghahramani, 2016). To clarify why the predictive probabilities found by deep learning cannot directly be used for measuring uncertainty, an example based on Gal and Ghahramani (2016) is used. In the example, the Gaussian process in Figure 9 is used as this visualization is easily interpretable. The true function to generate data points with is $f(x) = x * sin(x)$, where $x$ is the input. Randomly drawn noisy points from this function are represented with the blue dots. The prediction of the model is shown with the red line, with a 95 percent confidence interval. These predictions are made using a Gaussian process regression, but for this example we assume the red line represents the output for class $c = 0$ of a CNN model with two classes, before the softmax function is used. Now let the model be used to predict for a data point $x^* = 10$. In this case, the predicted function point estimate is very large ($\hat{f}(x^*) \approx 34.1$). If we assume the output of the network for the other class is 8, the predictive probability for class $c = 0$ using the softmax function (see Equation (5)) is $\frac{e^{34.1}}{e^{34.1}+e^8} \approx 1$. Hence the model seems to be close to entirely sure about $x^*$ belonging to $c = 0$ if only the point estimate is used as input for the softmax function. Considering the 95 percent confidence interval from the Gaussian process regression in the figure, the model is actually very uncertain about the prediction it made, but this is not reflected in the point estimate used to make predictions. Thus, the pre-

dictive probabilities of a CNN cannot be interpreted as the certainty of the model.

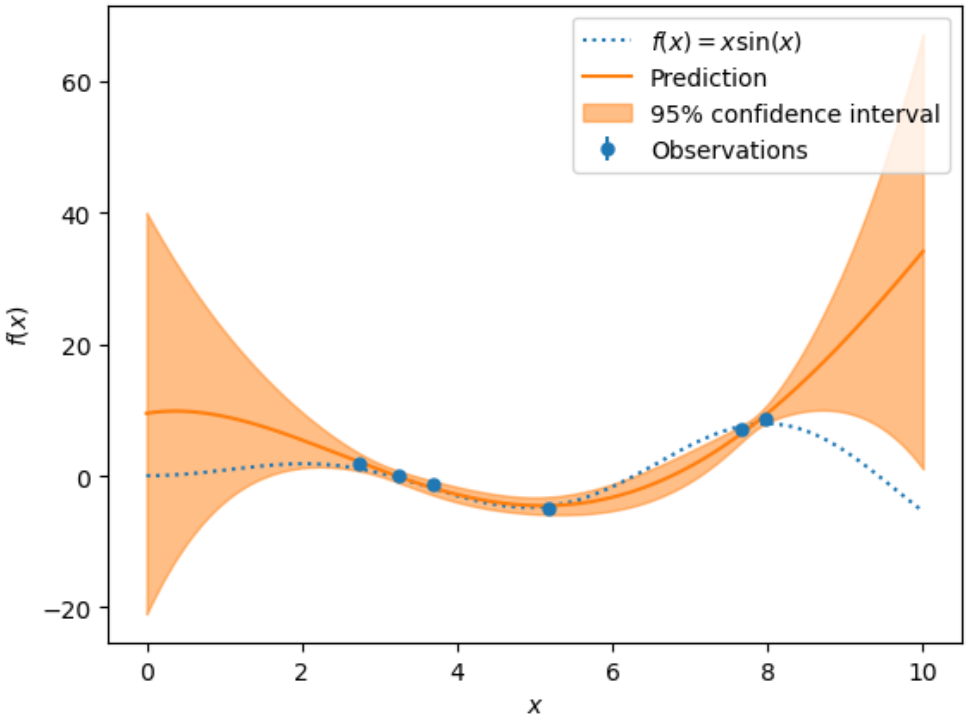



Figure 9: Gaussian process regression with true function $f(x) = x * sin(x)$ to clarify why the predictive probabilities do not equal certainty (made using code from Dubourg et al. (2022)).

To solve this problem, Bayesian Neural Networks can be used. These networks are able to incorporate uncertainty in the model by using a distribution over the estimated parameters. Multiple methods exist to create such BNNs, but these often introduce additional parameters to be trained, increasing training time. This is especially a problem for deep networks, such as Caladrius. Gal and Ghahramani (2016) developed a method for Bayesian approximation in neural networks which does not need any additional parameters. In Section 4.2, this method will be explained in detail.

## 2.4 Transfer Learning

In traditional machine learning, the domain and task of a model is equal when training the model on the training data and when using the model on the test data. The domain is the sample space and distribution of input data and the task refers to the label space and the conditional probability $P(y|X)$ of labels. Transfer learning is used when either the domain or the task differs between the training and test data (Pan and Yang, 2010). Thus for image recognition this is

the case when either the type of images used as input, or the classes in which the model is supposed to classify the images, differs between training and testing the model.

When data from former disasters is used to predict in a new disaster, the task is the same. For different disasters, the aim is still to classify the damaged buildings in the same categories. Furthermore, buildings with a same image from before and after the disaster should be labelled the same irrespective of the exact disaster it was part of. However, the sample space and distribution of data differ between disasters. Thus, in this case transfer learning can be implemented.

### 2.4.1 Transferability models for automated damage assessment

Valentijn et al. (2020) have evaluated the performance of transferring Caladrius CNN models trained using different disasters on the Joplin Tornado. The best performance was found for a model trained on all other wind type disasters (see Section 3). This model got close to the performance of a model which was trained on data from Joplin itself, achieving a macro-F1 score of 0.73 compared to 0.79 for the latter. The macro F1-score will be explained further in Section 4.3.

Hence the model transfers quite well between these wind type disasters. What must be noted is that Valentijn et al. (2020) mostly use data from the same country (the United States of America) as Joplin, which likely results in more similarities between the images compared to using images from other regions. The transferability of models to other regions might be worse.

### 2.4.2 Fine-tuning after transfer learning

Fine-tuning the pre-trained model using data from the new disaster is likely to improve the model. Especially when the new disaster struck in another region than the disaster(s) which the model has previously seen, there will be some differences with the images on which the model is pre-trained. After fine-tuning, the model will have more knowledge of data from the new disaster and hence will be more likely to perform well in this new situation.

Xu et al. (2019) found that such fine-tuning indeed improves the performance of models. They randomly sampled 10 percent of instances available from the new, unseen disaster to fine-tune a pre-trained model. The performance is tested on two different disasters, where a comparison is made with solely using the pre-trained model (Xu et al., 2019). The comparison between different models is shown in Table 1.

Table 1: Comparison of performances of models trained on different data sets, with all considered disasters being earthquakes (source: Xu et al. (2019)).

| Train datasets | Test datasets | AUC | Accuracy |
|---|---|---|---|
| Mexico | Mexico | 0.79 | 0.71 |
| Haiti | Mexico | 0.62 | 0.60 |
| Haiti + Indonesia | Mexico | 0.73 | 0.68 |
| Haiti + Indonesia + 10% of Mexico | 90% of Mexico | 0.76 | 0.72 |
| Indonesia | Indonesia | 0.86 | 0.78 |
| Haiti | Indonesia | 0.63 | 0.60 |
| Haiti + Mexico | Indonesia | 0.73 | 0.67 |
| Haiti + Mexico + 10% of Indonesia | 90% of Indonesia | 0.80 | 0.70 |

Wang (2021) similarly compared using only transfer learning with a combination of transfer learning with 10 percent of the instances from the tested disaster. Besides only testing this with random sampling, she also tests the performance of using uncertainty based active learning to select the 10 percent of images from the test disaster that are used. Active learning is discussed in more detail in the next subsection. Lastly, the results of only using 10 percent of instances selected with active learning are included (Wang, 2021).

In Table 2 the results of Wang can be found. Similarly to Xu et al. (2019), retraining a model using 10 percent of randomly sampled images from the test disaster after having pre-trained on data from other disasters outperformed solely using these other disasters. The difference between both is even more clear in this case study. Sampling these images using active learning improved the model even further, hence showing that using AL for ADA is promising. Lastly, using only the images selected with active learning instead of using them after pre-training on data from other disasters shows significantly worse results (Wang, 2021). Thus, previous exposure to damage assessment seems to improve the performance of the model (Wang, 2021). Given this result, this thesis will focus on the usage of AL in combination with transfer learning, instead of using only AL.

Table 2: Comparison of performances of models trained on different data sets, including active learning (source: Wang (2021)).

| Train Datasets | Test Datasets | AUC | Accuracy |
|---|---|---|---|
| Michael + Matthew | Harvey | 0.472 | 0.473 |
| Michael + Matthew + 10% Harvey via Random | Harvey | 0.756 | 0.756 |
| Michael + Matthew + 10% Harvey via Max-Entropy (330 imgs/query) | Harvey | 0.770 | 0.770 |
| Michael + Matthew + 10% Harvey via Max-Entropy (50 imgs/query) | Harvey | 0.777 | 0.777 |
| Only Active Learning on 10% Harvey via Max-Entropy | Harvey | 0.512 | 0.637 |

## 2.5 Active Learning

Models trained on data from the same disaster event as the one they were tested on were previously found to outperform those trained solely on images from other disaster events, even if these events are the same hazard type. However, labeling a set of images that is sufficiently large for training is time consuming. When time is limited, such as after a disaster, labeling sufficient data is not possible. Hence AL can be used to select those images that can improve models the most, such that only a limited number of images needs to be labelled. As mentioned in Section 2.4, Wang (2021) showed that fine-tuning using active learning indeed outperformed fine-tuning using random sampling. This subsection discusses the general background of research into AL. In Section 4.2 more recent developments that show promising results for image classification, which are implemented in this thesis, are discussed. Note that the overview on active learning by Settles (2009) is used throughout this section without explicitly referring to it each time.

The basic principle of AL is a machine, often referred to as the active learner, that chooses certain unlabelled images which an oracle should label. The oracle is generally a human annotator, which is also the case when AL is implemented for ADA in practice. For this research however, a labelled dataset is used where the labels are not yet known by the active learner. When a label request is made, the label of the image is made available to the active learner. The labelling request by the active learner will from now on be referred to as a query. Before going into more detail on how active learners can decide which images to query, first two common scenarios are discussed.

The first is a stream-based scenario, which assumes that unlabelled instances become available sequentially. The active learner needs to decide whether to query an instance or to discard it, after which the same question arises for the next instance.
The other scenario is pool-based, which is most frequently occurring in literature. This method assumes that a large number of instances can be gathered at the same time. The active learner can decide which instances should be labelled by the oracle based on several criteria, which are applied to a large number of instances. Hence the pool-based scenario enables the active learner to decide which images to query after having analyzed a large number of, or all, instances (Han et al., 2016). In Figure 10, the pool-based AL method is visualized. As satellite images are simultaneously made available, this thesis will focus on active learning in the pool-based scenario.

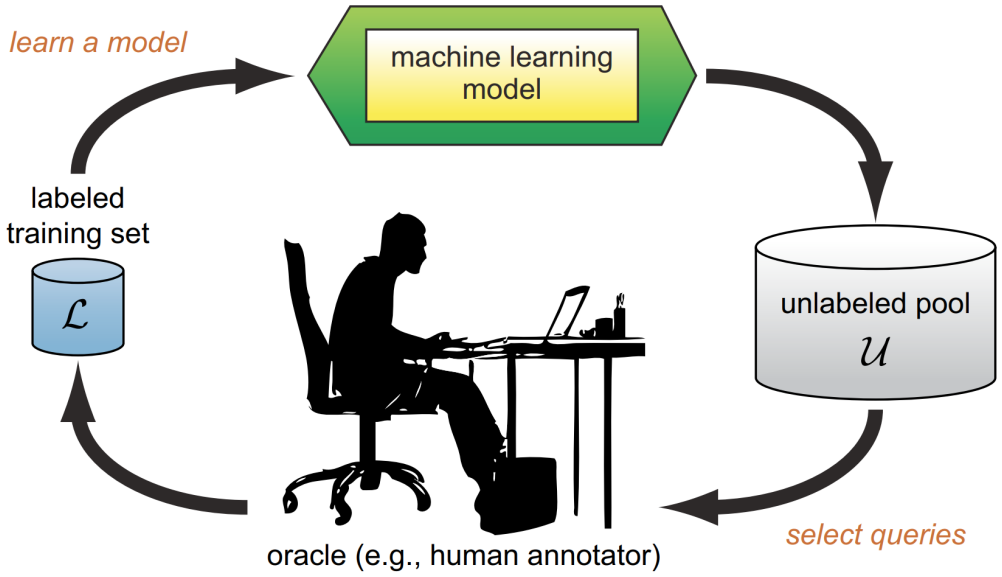



Figure 10: Pool-based active learning (source: Settles (2009)).

In the pool-based scenario, the active learner can decide which queries to make using multiple methods. Next, some commonly used types of methods are discussed, though not focused on deep learning. Most focus on exploitation, meaning they select those instances which are expected to improve the model the fastest. Other methods focus on the diversity of the sample. Methods which are specifically designed for deep learning and / or image data, which are implemented in this thesis, will be discussed in detail in Section 4.2.

### 2.5.1 Exploitation

**Uncertainty Sampling**

Uncertainty sampling is one of the most commonly used AL query strategies. It focuses on exploitation by choosing those images of which the model is the most uncertain. This uncertainty is often measured by entropy, see equation 11. In the equation, $\hat{y}_i$ is the predictive probability[5] of label $i$ given the model for instance $x$. $K$ is the number of classes. This value is large when the predictive probabilities for different classes are close to each other, and small when the probability is large for one class and small for others. Hence it can be interpreted as a measure for the certainty a model has about predicting a class. Other measures can be used as well, such as least confidence in the predicted label, where a low maximum predicted value shows that the model is not very confident about predicting the respective class (Settles, 2009).

[5]The output of the softmax layer is often interpreted as the probability of a class. In Section 2.3, more information on this interpretation is given.

$$H(x) = -\sum_{i}^{K} \hat{y}_i log(\hat{y}_i) \tag{11}$$

Van den Bogaart (2021) and Wang (2021) showed that uncertainty based AL methods are promising when applied to ADA, with Wang (2021) implementing an entropy based AL method similar to the method discussed above. For both, improvements are likely possible due to problems such as using output of a NN directly as discussed in Section 2.3 and no diversity being taken into account, which will be discussed in Section 2.5.3.

**Query-by-committee**

Query-by-committee (QBC) is another commonly used type of active learning. It uses a committee of models, which are trained using the same labelled data, but represent a different hypothesis. A straightforward method to construct this committee is using bagging or boosting (Abe, 1998). When a new instance needs to be labelled, all models in the committee predict the label. The instance about which the committee disagrees the most is chosen, as this is considered the most informative query (Seung et al. (1992) and Settles (2009)). Hence QBC has large similarities to ensemble methods such as random forests. Both use outputs of multiple models, but while random forests try to find the right prediction, QBC uses these models to find the instances that need to be labelled such that a good model can be trained.

**Expected model change**

Other methods, which similarly to uncertainty sampling and QBC aim to find those images the model learns the most of, try to maximise the expected model change more directly. Such methods aim to select an instance which is expected to alter the weights in a model the most. The most straightforward method to calculate the expected model change is using expected gradient length (EGL). Since the true label is not yet known, the EGL is calculated by multiplying the gradient length for all possible classes with their respective estimated probability and summing over these multiplications (Settles (2009) and Settles et al. (2007)). A drawback of such a method is that an instance which belongs to a certain class and has a large gradient but a low predicted value for this respective class is unlikely to be chosen when the gradient lengths for the other classes are small.

**Combining methods**

Given the previously found performance of uncertainty sampling by Wang (2021), a more sophisticated uncertainty based method, specifically designed for deep learning with images, will be implemented. This method, Bayesian Active Learning by Disagreement (BALD), also includes some elements closely related to QBC and expected model change. For a better estimation of uncertainty, multiple

models constructed using dropout are used, as explained further in Section 4.2.1. Using these multiple models to select the most uncertain images has similarities to QBC. On top, BALD aims to query those images which contain the maximum amount of information on the model parameters. These images are expected to result in a large model change, and hence the method has some similarities with expected model change. While QBC and expected model change are not directly implemented, elements from both are found in this method. More information on BALD is given in Section 4.2.2.

### 2.5.2 Diversity

Other methods focus on the diversity of selected datapoints. Instead of finding those instances that the model is most uncertain about, or those that would result in the largest change of the model, it focuses on finding a set of instances which are most representative for the entire sample space. An example of such a method tuned to CNNs is the coreset approach by Sener and Savarese (2017). They choose images such that the maximum distance to the nearest labelled image is minimized for unlabelled images. To measure distances between images, $L2$ distance between activations of the last fully-connected layer is used. In their experiment, the method outperforms methods such as uncertainty sampling (Sener and Savarese, 2017). Van den Bogaart (2021) has already investigated the core-set approach for ADA. Results are mixed, with the method sometimes even performing worse than random selection. Hence, focusing solely on diversity does not seem to be a viable option to train CNNs for ADA. This thesis therefore will not implement stand-alone diversity algorithms.

### 2.5.3 Combining diversity and exploitation

When using methods such as uncertainty sampling, chosen samples may lack diversity. This is especially the case when batches of datapoints are queried together. In such settings, the chosen images can be similar since the model may be struggling the most with a certain type of image. Uncertainty sampling would only select these similar, difficult to classify, images. Since these images have considerable similarities, the amount of information in such batches decreases (Mehrjou et al., 2018).

Retraining a CNN for each chosen instance is not practical since single instances are not likely to have a significant impact on accuracy and full training to reach convergence is time consuming (Sener and Savarese, 2017). Thus, solving this problem by selecting datapoints one by one is not practical, while acquiring batches of images by solely focusing on exploitation may reduce performance. Therefore, methods which combine exploitation with diversity could be preferred. In Section 4.2, such methods which are applicable to CNNs will be discussed in more detail.

# 3 Data

In this section, the data used in this research is discussed. Similar to most other recent research into satellite imagery based damage assessment, the xBD dataset is used.

## 3.1 xBD dataset

Previously, no large labelled dataset containing satellite images of different disaster-struck places was available. Since such dataset is essential to train and test ADA models, Gupta et al. (2019) created the xBD dataset. It uses pre and post disaster satellite images made available by Maxar, a satellite imagery company, through their Open Data program. These satellite images have a high resolution, with a ground sample distance of at most 0.8 meters. The dataset contains the locations of building polygons as well, which can be used to extract images of buildings from the satellite images. In real-life applications, these locations would not be known. For this purpose, 510 has built a building detection model which must be used prior to the damage classification model. However in this research, the building polygons available in the xBD dataset are used.

xBD contains satellite images of more than 800,000 building polygons together with damage labels, based on the Joint Damage Scale. This scale classifies buildings in four different categories: no damage, minor damage, major damage and destroyed. Table 3 contains the characteristics of buildings in each class. The labelling of buildings is performed by human annotators, after which the given labels are reviewed by other annotators. This ensures the consistency of the labels. Lastly a random sample of the annotated images is reviewed by experts, who found that 2-3 percent was mislabelled (Gupta et al., 2019).

Table 3: Joint Damage Scale classes and their respective description (source: Gupta et al. (2019)).

| Disaster Level | Structure Description |
|---|---|
| **0** (No Damage) | Undisturbed. No sign of water, structural or shingle damage, or burn marks. |
| **1** (Minor Damage) | Building partially burnt, water surrounding structure, volcanic flow nearby, roof elements missing, or visible cracks. |
| **2** (Major Damage) | Partial wall or roof collapse, encroaching volcanic flow, or surrounded by water/mud. |
| **3** (Destroyed) | Scorched, completely collapsed, partially/ completely covered with water/mud, or otherwise no longer present. |

**The disasters included in the xBD dataset**

The xBD dataset contains data from 19 different disasters. These disasters include earthquakes, tsunamis, floods, volcanic eruptions, wildfires, hurricanes and tornadoes. These disasters took place in different regions of the world. In Figure 11, the different disasters are shown with the type of damage caused and geographical location.

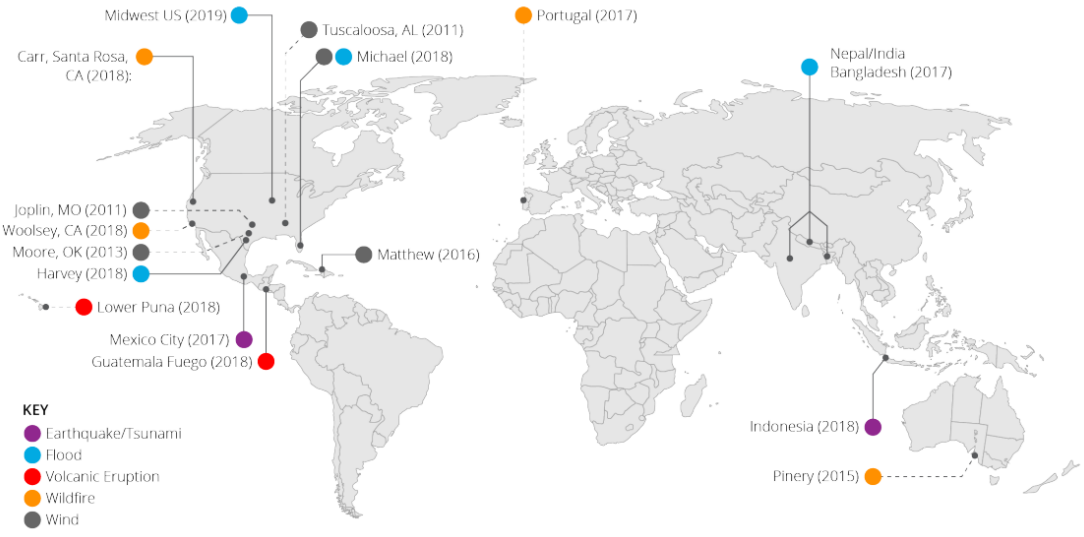



Figure 11: Disasters included in xBD with damage type and geographical location (source: Gupta et al. (2019)).

## 3.2 Data of the disasters used in this research

To align with the priorities of the Red Cross, Valentijn et al. (2020) only focused on tornadoes, hurricanes, floodings, tsunamis and volcanic eruptions. Training CNNs on multiple types of disasters is time consuming. Therefore this thesis will only consider disasters with wind type damages, which are hurricanes and tornadoes. This is chosen since wind type damage is likely to be visible on the roof, and thus the damage is likely to be visible on satellite images. Furthermore, there are multiple disasters available with this type of damage which can be used for transfer learning. Valentijn et al. (2020) showed promising results when using transfer learning for one of the wind type disasters. Lastly, Van den Bogaart (2021) similarly used only wind type disasters, making comparison between results easier. Hence images are used from the following disasters: hurricane Matthew, Moore tornado, Tuscaloosa tornado, Joplin tornado and hurricane Michael. Two more hurricanes, Florence and Harvey, are included in xBD. However, these hurricanes caused flood type instead of wind type damages and hence these are not used in this research. The included disasters mainly took place in the United States of America (USA). Only data on hurricane Matthew is from Haiti instead of the USA.

Table 4 contains an overview of all disasters used in this research, with both the number of buildings available and the distribution over classes. For most disasters, the most common class is no damage. In general, the distribution over classes is very uneven. Hurricane Matthew has a different distribution in damage labels compared to the other disasters, with the most buildings having minor damage.

Table 4: Disasters used in research, with the class distribution being no damage / minor damage / major damage / destroyed, similar to Valentijn et al. (2020).

| Disaster | Location | Number of Buildings | Class Distribution* |
|---|---|---|---|
| Hurricane Matthew | Haiti | 16,457 | 18/58/13/12 |
| Moore Tornado | USA | 18,855 | 87/4/2/6 |
| Tuscaloosa Tornado | USA | 12,577 | 74/15/3/7 |
| Joplin Tornado | USA | 12,163 | 55/16/8/21 |
| Hurricane Michael | USA | 31,332 | 64/25/9/3 |
| Total | | 91,384 | 70/17/6/7 |

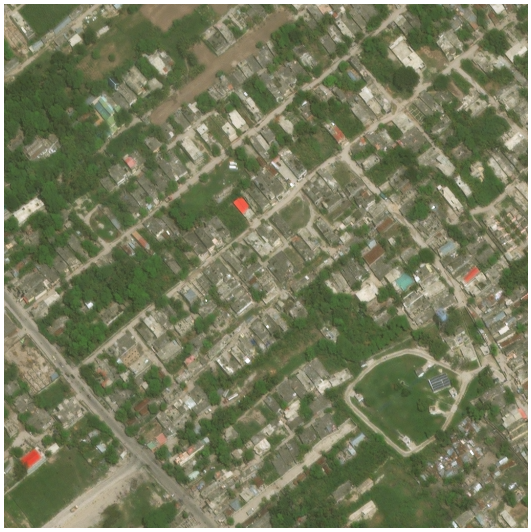
*Note: due to rounding, the class distribution could sum to a different amount than 100. The presented class distribution is based on the labels in the train set.

## 3.3 Data examples

To give some insight into the images used, this subsection shows some examples. In Figure 12, satellite images of a residential area before and after the disaster for

the Matthew and Michael hurricanes are shown. These are the original images from which images of buildings are extracted. While this is only an example, clear differences can be seen between the type of buildings in both regions.

(a) Pre disaster image Matthew



(b) Post disaster image Matthew

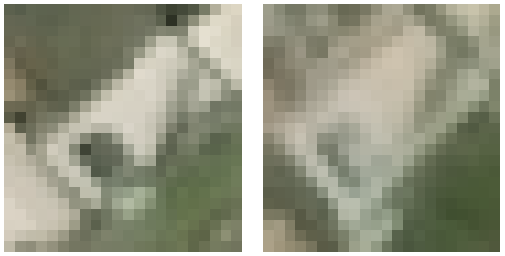
(c) Pre disaster image Michael

(d) Post disaster image Michael

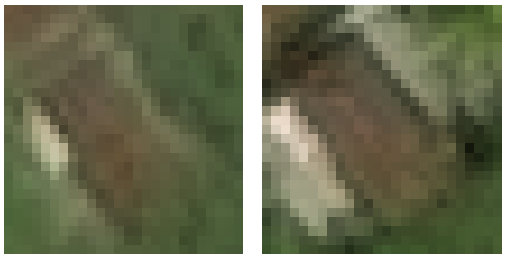
Figure 12: Satellite images from before and after the Matthew and Michael hurricanes (source: Gupta et al. (2019)).

To clarify the input used for the Caladrius model, Figure 13 contains examples of images of buildings extracted from satellite images such as displayed in Figure 12. For each class, one building is selected from imagery of hurricane Matthew. While the used satellite images are high-resolution, still some images are not clear. Even for humans it may be difficult to select the right label when they are not trained

in such classification tasks. Therefore, it can be difficult for models to reach a very good performance.

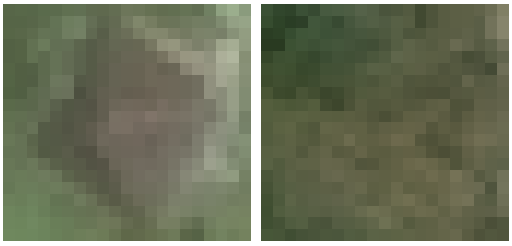


(a) No damage



(b) Minor damage

(c) Major damage



(d) Destroyed

Figure 13: Examples of pre disaster (left) and post disaster (right) images of buildings for all four possible classes. Images are from hurricane Matthew in Haiti (source: Gupta et al. (2019)).

# 4 Methodology

In this section, the transfer learning implemented is discussed first. Different implemented active learning methods are then explained in detail, after which an explanation on the evaluation of the performance of these methods is given. Lastly, the hyperparameters that will be tuned are discussed.

## 4.1 Transfer Learning

As discussed in Section 2.4, transfer learning achieves promising, but mixed, results for ADA. Especially when the type of damage caused by the natural hazard and the region where it occurred are similar between the disasters on which a model is pre-trained and the disaster for which the model is used, good results can be obtained.

In practice, new disasters almost always happen in other places than those of which labelled images from prior disasters is available to train Caladrius on. Using solely transfer learning can be insufficient to achieve good performance in this case. The new region is likely to have different features which need to be learned by a model using data from this new disaster. Fine-tuning pre-trained models is more likely to yield significant improvements in such applications. Additionally, aid in damage assessment by 510 is more likely to be needed in developing countries such as Haiti, where hurricane Matthew struck, than in the USA where all other wind type disasters took place. Therefore, this thesis will use the Moore tornado, Tuscaloosa tornado, Joplin tornado and hurricane Michael to pre-train a model, which will be used on data from Hurricane Matthew.

The class distribution of the data we want to label is not known immediately after a disaster. Therefore, a model should be capable of being tuned to a disaster with a different class distribution. In the data used, hurricane Matthew has a clearly different class distribution compared to the Moore tornado, Tuscaloosa tornado, Joplin tornado and hurricane Michael. When pre-training the model on the other wind-type disasters, the loss is therefore weighted such that each class has an equal contribution to the loss. This way when pre training, the model is not focused more on certain classes which occur more in the disasters pre-trained on. While not being tested due to time constraints, this weighting of the loss when pre-training is expected to make it easier for the model to be tuned for classification in the newly occurring disaster.

As mentioned in Section 2.4, fine-tuning pre-trained models using a small amount of data from the new disaster is likely to improve performance. In the next subsection, different methods to select which images are used for fine-tuning are introduced.

## 4.2 Deep Active Learning

This subsection explains the active learning methods which will be used to select images for fine-tuning pre-trained models. The discussed models all use some measure of uncertainty when querying data, which is combined with diversity for some of them. The outcomes of all models are compared with each other, as well as with random sampling instances. Before selecting images using these methods, an initial batch of 100 images is randomly labelled, with which the model is trained at the start of the AL procedure. This initialization is common practice in active learning (Norouzzadeh et al., 2021), and for some models required. Before introducing the AL methods, Bayesian neural networks are introduced, which are used in some of the AL methods.

### 4.2.1 Bayesian Neural Networks

As explained in Section 2.3, the output of the softmax layer in (convolutional) neural networks does not represent the certainty of predictions. A Bayesian neural network (BNN) can be used to incorporate uncertainty in the model by using a distribution over the estimated parameters. Gal and Ghahramani (2016) developed a method for the approximation of a BNN with relatively low computational costs using dropout. More information on dropout can be found in Section 2.2.6. To approximate the certainty of a model for sample $x^*$, they use the average of the outcomes of the model with different nodes dropped out. This method is called Monte Carlo (MC) dropout. Thus it essentially predicts using multiple different variants of the model. Using the outputs of these different variants of the model, the uncertainty of the model can be quantified better.

More formally, they first define the approximating variational distribution $q(\omega)$ with $\omega = (W_i)_{i=1}^{L}$ as:

$$\begin{aligned} W_i &= M_i * diag([z_{ij}]_{j=1}^{K_i}) \\ z_{ij} &\sim Bernoulli(p_i) \text{ for } i = 1, ..., L,\ j = 1, ..., K_{i-1}, \end{aligned} \tag{12}$$

where $L$ is the number of layers in the model, with the input layer not counting towards the total. $K_i$ is the number of nodes in a given layer. $p_i$ is the probability of retaining a node in a given layer. $M_i$ contains variational parameters to be optimized (Gal and Ghahramani, 2015). This notation is clarified using Figure 14, which is a simple Neural Network with $L = 3$ layers. The values of $K_i$ are given for all layers. $M_i$ is visualized as all the connecting lines between the given layer and the layer before. Usually, the variational parameters are optimized by minimizing the Kullback-Leibler divergence. When a model is trained using dropout between all layers, the optimal weights found while optimizing the model are equal to the optimal variational parameters (Gal et al., 2016). This property is proven in the appendix to Gal and Ghahramani (2016). Hence in such cases, $M_i$ can be substituted with the parameters $\theta$ found when training the network, as introduced in

Section 2.2.5. Given this $\theta$, the approximating variational distribution is denoted as $q_\theta(\omega)$.

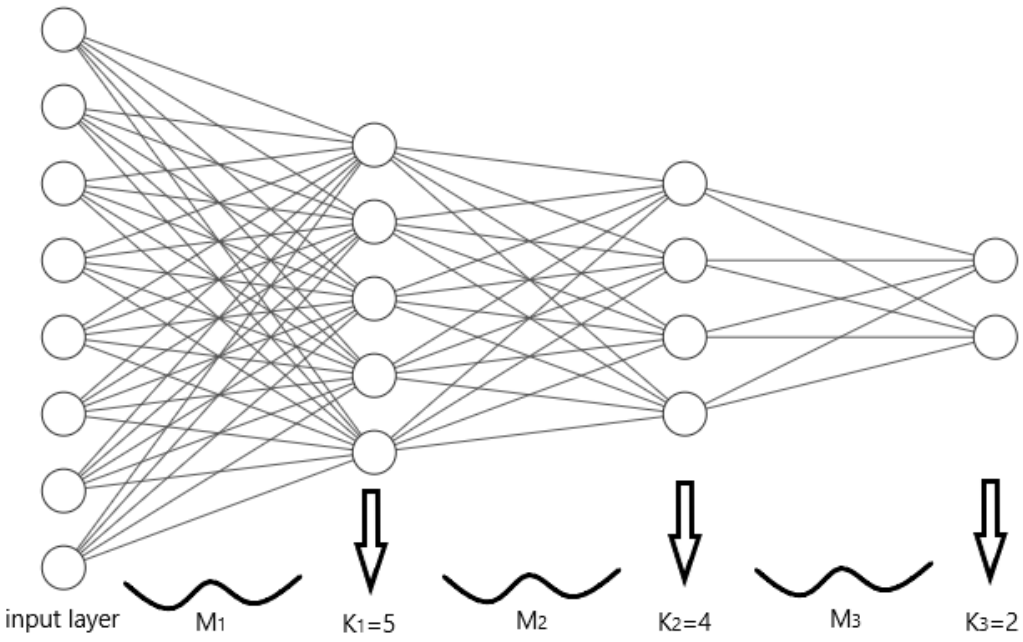



Figure 14: Clarification notation used for $q(\omega)$, with $L = 3$.

Using $q_\theta(\omega)$ the uncertainty of the neural network can be estimated with Monte Carlo integration, using multiple stochastic forward passes through the network, where a stochastic forward pass corresponds to a single prediction using a MC dropout model (Gal et al., 2017):

$$\begin{aligned} p(y = c|x, D_{train}) &= \int p(y = c|x, \omega) p(\omega|D_{train}) d\omega \\ &\approx \int p(y = c|x, \omega) q_\theta(\omega) d\omega \\ &\approx \frac{1}{T} \sum_{t=1}^{T} p(y = c|x, \hat{\omega}_t) \end{aligned} \tag{13}$$

Where $c$ is a possible class, in this case e.g. "destroyed". $D_{train}$ is the training set. $T$ is the number of stochastic forward passes. $p(\omega|D_{train})$ is the true posterior distribution given the training data, which is approximated using dropout distribution $q_\theta(\omega)$. $\hat{\omega}_t$ denotes a random draw from $q_\theta(\omega)$. $p(y = c|x, \hat{\omega}_t)$ is the prediction for class $c$ obtained using a forward pass in the model constructed with $\hat{\omega}_t$. $p(y = c|x, D_{train})$ is thus effectively estimated by averaging over estimates of multiple forward passes with a different set of nodes being dropped (Gal and

Ghahramani, 2016).

Besides having a low computational cost compared to other methods to construct a BNN, this method has another major advantage. Since it averages forward passes in a neural network trained using dropout, with in each forward pass other neurons being dropped out, it can be used on an already existing neural network. Hence, using this method, any (convolutional) neural network which has been trained using dropout can straightforwardly be turned into a BNN, or a Bayesian convolutional neural network (BCNN).

Given that the BCNN is contructed to give a better representation of uncertainty, the most straightforward active learning method is to simply select queries using entropy as discussed in Section 2.5.1. However, Gal et al. (2017) propose more sophisticated methods as well, of which one is introduced next.

### 4.2.2 Bayesian Active Learning by Disagreement

When selecting images using entropy, the model simply chooses those images of which it is the most uncertain. The idea is that a model can learn the most from such images, thus that it contains the most information of which the model can learn. Bayesian Active Learning by Disagreement (BALD) is closely related to entropy, but calculates the information on true parameters contained in images more directly. It aims to choose the image which maximizes the decrease in expected posterior entropy. Houlsby et al. (2011) show that this is equivalent to maximizing the mutual information between the parameters and unknown output conditional on labelled training data and the image which is considered. In Figure 15, this mutual information is visualized by the grey shaded area, which is the intersection of both circles which represent the information contained in the parameters and outcome respectively. Hence, the mutual information can be explained as the overlap of the information in both the parameters and outcomes. It can be calculated as follows (Houlsby et al., 2011):

$$I(\omega, y|x, D) = H[y|x, D] - E_{\omega \sim p(\omega|D)}[H[y|x, \omega]] \tag{14}$$

Here, $x$ is a particular instance for which the BALD value is calculated, with $y$ output that must be predicted. $D$ is the currently labelled train set, with $\omega$ the parameters of the model. $p(\omega|D)$ is the true posterior distribution of the parameters, which can be approximated using the dropout distribution $q_\theta(\omega)$ as defined in the previous section. $H[y|x, D]$ denotes the average entropy of predictions using the posterior parameter distribution $p(\omega|D)$, calculated using Equation (11). $H[y|x, \omega]$ is the estimated entropy for a single model with weights $\omega$ which is drawn from the mentioned posterior distribution. Thus, $E_{\omega \sim p(\omega|D)}[H[y|x, \omega]]$ is estimated by first calculating the entropy over all drawn models from $p(\omega|D)$ and

then taking the average over these entropies.

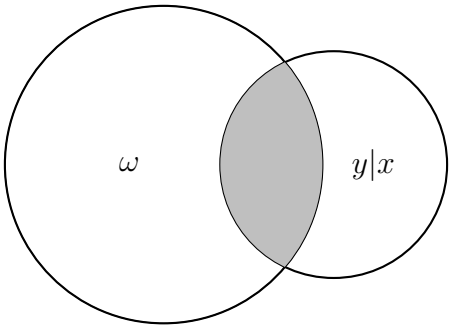



Figure 15: Visualization of mutual information (shaded grey) between parameters $\omega$ and outcome $y$.

Using MC dropout to estimate $p(\omega|D)$ as $q_\theta(\omega)$, Gal et al. (2017) show how BALD can be estimated using a BCNN:

$$\begin{aligned}\hat{I}(\omega, y|x, D) \approx -\sum_c (\frac{1}{T}\sum_t p[y=c|x,\hat{\omega}_t]) log(\frac{1}{T}\sum_t p[y=c|x,\hat{\omega}_t]) \\ + \frac{1}{T}\sum_{t,c} p[y=c|x,\hat{\omega}_t] log(p[y=c|x,\hat{\omega}_t])\end{aligned} \tag{15}$$

Here, $T$ is the number of MC dropout models used, over which $t$ spans. $c$ is the class. $\hat{\omega}_t$ is the set of parameters for dropout model $t$ drawn from $q_\theta(\omega)$ as shown in Section 4.2.1.

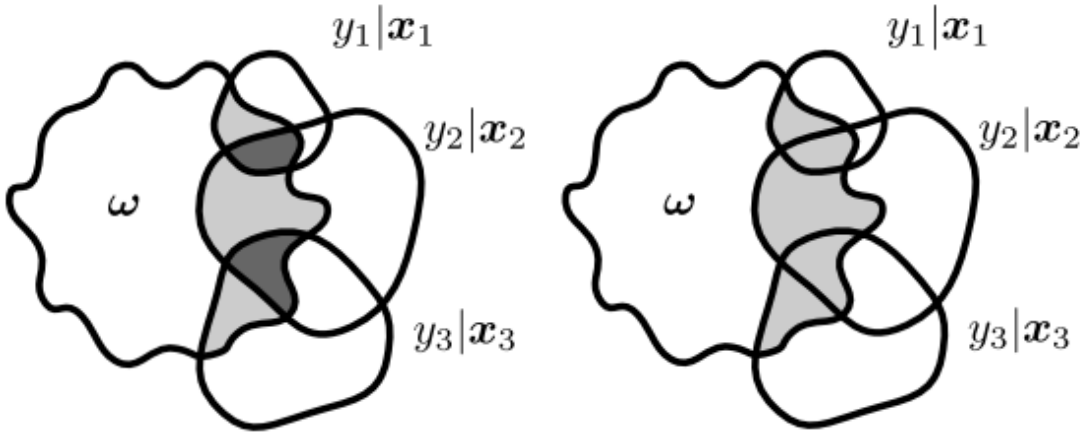
Computing the conditional mutual information between the unknown outcome and parameters can appear complex. The idea behind BALD can be clarified by explaining how the different terms in equation 14 behave. The first term, $H[y|x, D]$, is large when the average of drawn models is uncertain about its prediction. The second term, $E_{\omega\sim p(\omega|D)}[H[y|x, \omega]]$, is small when different models drawn from $p(\omega|D)$ are certain about their predictions. In the MC dropout implementation, this corresponds to the different dropout models being certain about their predictions. As this second term is subtracted from the first term, BALD selects instances of which the model is uncertain on average but different drawn models result in disagreeing predictions (Gal and Ghahramani, 2016).

Using BALD, $T$ is an important hyperparameter to be set. This hyperparameter has a large influence on acquisition time. In Section 4.4.2, the considerations in selecting this hyperparameters are explained further.
Both BALD and BatchBALD, which is discussed in the next subsection, are implemented using code from Atighehchian et al. (2022).

### 4.2.3 BatchBALD

When a batch of instances is selected, the model may be uncertain about similar instances. The active learner may choose to only query these similar instances. Information contained in these selected batches is likely to have overlap. For BALD, this issue is visualized in Figure 16a. Here, the conditional mutual information between the parameters $\omega$ and outcomes $y_1$, $y_2$ and $y_3$ are shown with the shaded area. When BALD were to select a batch of images, it would simply sum the conditional mutual information between the parameters and outcomes without taking into account the overlap in the mutual information for these outcomes. If the same information is contained in different images, this information is counted double when creating the batch, as visualized in Figure 16a by the dark shaded areas (Kirsch et al., 2019).

To maximize the informativeness of a batch of images, the information of a batch should be calculated by counting the overlapping information only once. BatchBALD aims to do this by calculating the union of the mutual information between the outcomes and parameters instead of the sum, as visualized in Figure 16b. Here, the overlapping mutual information is only counted once (Kirsch et al., 2019).




(a) BALD (b) BatchBALD

Figure 16: Visualization of batch information calculated using BALD and BatchBALD (source: Kirsch et al. (2019)).

In order to compute the union of the mutual information, equation (14) must be adjusted. Instead of computing the conditional mutual information for each image separately, the joint conditional mutual information for a batch is calculated (Kirsch et al., 2019):

$$\begin{aligned} I(\omega, y_1, ..., y_b | x_1, ..., x_b, D) = H[y_1, ..., y_b | x_1, ..., x_b, D] \\ - E_{\omega \sim p(\omega|D)}[H[y_1, ..., y_b | x_1, ..., x_b, \omega]] \end{aligned} \tag{16}$$

Where $x_1, ..., x_b$ denotes the $b$ different images selected in a given acquisition batch and $y_1, ..., y_b$ denotes the corresponding predictions of these images. For brevity, these will be referred to as $x_{1:b}$ and $y_{1:b}$ in the remainder of this subsection. Function $H$ denotes the entropy as defined in equation (11) in this whole subsection. Further notation is the same as in equation (14).

Computing the joint conditional mutual information for each possible batch separately using equation (16) would be computationally impossible[6]. Therefore, a greedy heuristic is used which iteratively adds the image resulting in the largest increase in the joint conditional mutual information to the query batch until the required batch size is reached. Kirsch et al. (2019) show that this heuristic is a $(1 - 1/e)$ approximation.

The estimation of equation (16) is explained separately for the left and right term of the equation. Kirsch et al. (2019) note that the right term can be estimated by simplifying the conditional joint entropy into a sum of expected entropies for separate images. This is possible given that the predictions $y_i$ are independent when conditioned on parameters $\omega$. Estimating the expected entropy for these images separately is done using MC dropout (Kirsch et al., 2019). The estimation of this right term is therefore just the sum of conditional entropies for different images similar to the right term of BALD in equations (14 - 15):

$$\begin{aligned} E_{\omega \sim p(\omega|D)}[H[y_{1:b}|x_{1:b}, \omega]] &= \sum_{i=1}^{b} E_{\omega \sim p(\omega|D)}[H[y_i|x_i, \omega]] \\ &\approx \frac{1}{T} \sum_{i=1}^{b} \sum_{t=1}^{T} [H[y_i|x_i, \hat{\omega}_t]] \end{aligned} \tag{17}$$

with all symbols as previously defined in equations (14 - 16).

Computing the left part of equation (16) is more challenging. Given that the joint mutual entropy is not conditioned on the parameters, the predictions $y_i$ are not independent. Therefore, this term cannot be decomposed into a sum of separate mutual information estimates. It is thus approximated using all possible configurations of $\hat{y}_{1:b}$, which are the MC dropout estimations of the configurations of $y_{1:b}$ (Kirsch et al., 2019):

$$H(y_{1:b}) \approx -\sum_{\hat{y}_{1:b}} \left(\frac{1}{T} \sum_{t=1}^{T} p(\hat{y}_{1:b}|\hat{\omega}_t)\right) log\left(\frac{1}{T} \sum_{t=1}^{T} p(\hat{y}_{1:b}|\hat{\omega}_t)\right) \tag{18}$$

While this equation works for small acquisition batch sizes, the calculations explode when more images are selected as the number of possible configurations

[6]When selecting a batch of 100 images out of a pool of 10,000 images, there would be $6.5 * 10^{241}$ possible combinations for which the joint conditional mutual information should be computed.

becomes too large[7]. To reduce the number of calculations needed, $p(y_{1:b}|\omega)$ can be factorized into $p(y_{1:b-1}|\omega)p(y_b|\omega)$. This is possible since the previously chosen images $x_{1:b-1}$ are fixed when using the aforementioned heuristic. In the estimation using MC dropout, this yields (Kirsch et al., 2019):

$$\frac{1}{T}\sum_{t=1}^{T} p(\hat{y}_{1:b}|\hat{\omega}_t) = \frac{1}{T}\sum_{t=1}^{T} p(\hat{y}_{1:b-1}|\hat{\omega}_t)p(\hat{y}_b|\hat{\omega}_t) = \frac{1}{T}(\hat{P}_{1:b-1}\hat{P}'_b)_{\hat{y}_{1:b-1},\hat{y}_b} \tag{19}$$

Where the last part rewrites the equation to a matrix multiplication to speed up calculations. Here $\hat{P}_{1:b-1}$ is a $c^{b-1} * T$ matrix with $T$ MC dropout predictions $p(\hat{y}_{1:b-1}|\hat{\omega}_t)$ for the $c^{b-1}$ configurations of the currently picked $b-1$ images. This only has to be calculated once every iteration. $\hat{P}_b$ is a $c * T$ matrix with the $T$ MC dropout predictions $p(\hat{y}_b|\hat{\omega}_t)$ for the $c$ classes.
Still, when the acquisition batch size grows large, $\hat{P}_{1:b-1}$ becomes too large to compute. Thus Kirsch et al. (2019) reduce the computation time of equation (19) by using $m$ samples of the configurations in $\hat{y}_{1:b-1}$:

$$H(y_{1:b}) \approx -\frac{1}{m}\sum_{i}^{m}\sum_{\hat{y}_b} \frac{(\hat{P}_{1:b-1}\hat{P}'_b)_{\hat{y}^i_{1:b-1},\hat{y}_b}}{(\hat{P}_{1:b-1}1_{T,1})_{\hat{y}^i_{1:b-1}}} log(\frac{1}{T}(\hat{P}_{1:b-1}\hat{P}'_b)_{\hat{y}^i_{1:b-1},\hat{y}_b}) \tag{20}$$

Where now $\hat{P}_{1:b-1}$ is a matrix with dimensions $m * T$ containing $p(\hat{y}^i_{1:b-1}|\hat{\omega}_t)$. Thus, this matrix contains the $T$ MC dropout predictions for each configuration $\hat{y}^1_{1:b-1}, ..., \hat{y}^m_{1:b-1}$, which are sampled from all possible configurations of $\hat{y}_{1:b-1}$. $1_{T,1}$ is a vector of ones with dimension $T * 1$. The rest of the symbols are as defined before. To construct this formula importance sampling is used. For more information on this, please refer to Appendix C of Kirsch et al. (2019).

Plugging the estimations from equation (17) and (20) into equation (16), we get the implemented estimation of the joint conditional mutual information for a batch $x_{1:b}$:

$$\begin{aligned}\hat{I}(\omega, y_{1:b}|x_{1:b}, D) = -\frac{1}{m}\sum_{i}^{m}\sum_{\hat{y}_b} \frac{(\hat{P}_{1:b-1}\hat{P}'_b)_{\hat{y}^i_{1:b-1},\hat{y}_b}}{(\hat{P}_{1:b-1}1_{T,1})_{\hat{y}^i_{1:b-1}}} log(\frac{1}{T}(\hat{P}_{1:b-1}\hat{P}'_b)_{\hat{y}^i_{1:b-1},\hat{y}_b}) \\ - \frac{1}{T}\sum_{i=1}^{b}\sum_{t=1}^{T}(H(y_i|x_i, \hat{\omega}_t))\end{aligned} \tag{21}$$

In the previously mentioned heuristic, this value is calculated for each image that has not yet been selected into the batch. The image with a maximum joint conditional mutual information $\hat{I}$ is added to the batch, which is repeated until the

[7]Considering a situation where we want to acquire a batch of 100 images, with the four possible classes used in this research, the number of possible configurations becomes $4^{100} \approx 1.6 * 10^{60}$. For each configuration, $T$ MC dropout estimates are used. This then needs to be calculated for each possible batch from the unlabelled pool, resulting in a huge number of calculations.

desired batch size is reached.

While for BALD the only additional hyperparameter was the number of MC dropout models used (denoted with $T$), for batchbald $m$ is added as well as well. In the code implemented based on Kirsch et al. (2019), $m$ depends on $T$: $m = T * s$. This $s$ is the additional hyperparameter that must be set in order to increase or decrease the number of draws used from $\hat{y}_{1:b-1}$ given $T$. In Section 4.4.2, considerations in choosing combinations of $T$ and $s$ are discussed.

### 4.2.4 Wasserstein Adversarial Active Learning

Another implemented method, developed by Shui et al. (2020), is Wasserstein Adversarial Active Learning (WAAL). WAAL is a deep active learning method which aims to combine diversity and uncertainty when selecting queries. Note that throughout this section, Shui et al. (2020) is used without always explicitly referring to it. Before discussing how WAAL is used, first the reason to implement this specific method is explained.

Zhan et al. (2022) compared the performance of 18 active learning methods for deep neural networks, amongst which BALD and WAAL, as well as other recently introduced methods combining diversity and uncertainty. BatchBALD is not included in these methods, mainly due to the large amount of memory it uses. They test the performance of these methods on multiple datasets, such as the often used MNIST and CIFAR datasets, as well as medical image datasets. Methods are ranked based on whether they perform better, worse or similar compared to the other tested methods. Using 8 different standard image recognition datasets, WAAL is ranked first. For both the medical image datasets it outperforms all other models by a margin, while even outperforming a model trained on the full dataset (Zhan et al., 2022). An overview of the ranking of AL methods from the paper is included in Appendix Table B.1.

The previous methods used Bayesian models to maximize the mutual information between outcomes and the parameters. WAAL is a different type of algorithm. Whereas BALD and BatchBALD mainly focus on the uncertainty of predictions, with BatchBALD taking overlap of information into account as well, WAAL aims to more directly select images that are both uncertain and result in a diverse sample. It does this by combining metrics measuring uncertainty and diversity, which is explained later in this section. Beyond this, it uses information from the unlabelled images when training the model by implementing adversarial training, similar to the training of Generative Adversarial Networks (GANs). Thus, the aim of WAAL is to both select the right images to label and to improve training by using the unlabelled samples (Shui et al., 2020).

#### Feature extractor, classifier and discriminator networks

To implement WAAL in practice, first the CNN model of 510 (Caladrius) must be adjusted to fit the WAAL framework. Instead of simply using one CNN, WAAL divides the network in different parts as visualized in Figure 17. The first part is the feature extractor, which is used to extract the features from images and should be trained using both the labelled and unlabelled data. When using WAAL with the Caladrius architecture (Figure 7), the feature extractor is made out of the two inception-V3 models for the pre and post disaster images, from which features are concatenated. Features extracted with this network are used as the input for two other networks.
The first of these is a classifier, which only uses the features extracted from labelled data. Its architecture is the same as the fully connected layers from the full Caladrius model, as shown in Figure 17. After the feature extractor and classifier are trained using WAAL, the full Caladrius model can be reconstructed by pasting this classifier behind the feature extractor.
Lastly, a discriminator network is used. This has a similar architecture to the classifier, but instead of 4 output nodes, it only has one output node. Additionally, the batch normalization used in the classifier is replaced with layer normalization, which is further explained in Ba et al. (2016). This is proposed by Gulrajani et al. (2017) for their Wasserstein Generative Adversarial Network, on which the adversarial training in WAAL is based[8]. The output is used to discriminate between labelled and unlabelled data, which can be used to ensure diversity when querying new labels. On top, the discriminator can be used to train the feature extractor on unlabelled data. For each of these networks, different loss functions are used, which are explained later in this section.

Both the discriminator and the classifier can only be used in combination with the feature extractor, since their inputs are the output created by the feature extractor. In Figure 17 this is visualized by both networks being connected to the feature extractor. The feature extractor can be trained both using the classifier and discriminator. Subsequently using the feature extractor and classifier will from now on be referred to with $h(x)$, while subsequently using the feature extractor and discriminator will be referred to with $g(x)$. $x$ is the input data to the feature extractor, which can be either labelled or unlabelled. Unlabelled data is only used in combination with the discriminator. The parameters of the networks are referred to as $\theta^f$, $\theta^c$ and $\theta^d$ for the feature extractor, classifier and discriminator respectively.

[8]Another option would have been to remove the batch normalization without replacing it, but this resulted in bad performance. Further, batch normalization was only replaced with layer normalization after having configured the epochs, optimizer and learning rate hyperparameters.

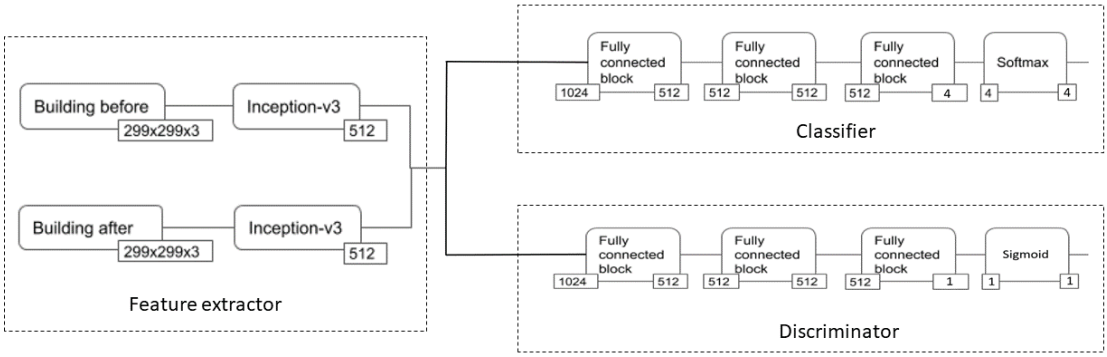



Figure 17: WAAL Caladrius model with a feature extractor, classifier and discriminator. The numbers below the different component of the networks correspond to the input and output dimensions of these components. For more information on the Inception-v3 models, see Section 2.2.7. The fully connected blocks consist of a fully connected layer with the ReLU activation function, batch normalization or layer normalization, and dropout.

**Wasserstein distance**

Before the practical implementation using these three networks is discussed in detail, the Wasserstein distance must be introduced. The Wasserstein distance is a metric used to calculate the minimal transport costs going from one distribution to another distribution. In this case, it is used to measure these costs between the distributions underlying the labelled and unlabelled data. Shui et al. (2020) specifically use the Wasserstein-1 distance, which uses Euclidean distance to measure the distance between data from both distributions.

A small Wasserstein distance means that the distributions are similar. In the AL context, the Wasserstein distance would thus measure how similar the distributions of labelled and unlabelled datapoints are. Selecting query data by minimizing this distance would therefore result in a labelled dataset which closely resembles the unlabelled dataset. Following this intuition, Shui et al. (2020) show that indeed a diverse query results in a small Wasserstein distance.

**AL using Wasserstein distance**

For training and query together, WAAL has two objectives: minimizing the estimation error and minimizing the Wasserstein distance. Let $\hat{L}$, $\hat{U}$, $\hat{B}$ and $\hat{D} = \hat{L} \cup \hat{U}$ be the set of labelled data, unlabelled data, query data and total data respectively. Let $h$ be a hypothesis which essentially is the classification model used[9]. Furthermore, let $\hat{R}_A(h) = \frac{1}{N} \sum_{i=1}^{N} l(h(x_i), y_i)$ estimate the loss of predicting using $h$ for

[9]Shui et al. (2020) define $h$ as an at most $H$-Lipschitz function which transforms input $X$ to output $Y$, but to simplify the explanations it will simply be referred to as a model. More information on the function and assumptions can be found in Shui et al. (2020).

data distribution $A$. Then WAAL aims to minimize (Shui et al., 2020):

$$\min_{\hat{B},h} \hat{R}_{\hat{L}\cup\hat{B}}(h) + \mu W_1(\hat{D}, \hat{L}\cup\hat{B}), \tag{22}$$

where minimizing the first term should result in good predictions on the labelled dataset and minimizing $W_1$, which is the Wasserstein-1 distance, ensures a diverse representation of $\hat{D}$. Both the model $h$ and query batch $\hat{B}$ are chosen by minimizing these functions together. $\mu$ is a hyperparameter which can be used to set how important the two terms are relative to one another.

Estimating the Wasserstein-1 distance is challenging in practice. Shui et al. (2020) therefore rewrite equation (22) as a min-max optimization problem using the discriminator, classifier and feature extractor as defined previously[10]. The equation then becomes:

$$\min_{\theta^f,\theta^c,\hat{B}} \max_{\theta^d} \hat{R}(\theta^f,\theta^c) + \mu\hat{E}(\theta^f,\theta^d) \tag{23}$$

Here, $\hat{R}_{\theta^f,\theta^c} = E_{(x,y)\ \hat{L}\cup\hat{B}} l(h(x,y))$, which is the expected loss of the classification model, implemented using cross-entropy similarly to the other AL models. $\hat{E}(\theta^f,\theta^d) = E_{x\in\hat{D}}(g(x)) - E_{x\in\hat{L}\cup\hat{B}}(g(x))$ is the adversarial loss, calculated as the difference of the discriminator function's predictions on the full dataset and the labelled dataset[11]. Thus, the smaller this expression is, the closer the labelled dataset resembles the full dataset.

Shui et al. (2020) further decompose this equation to enable estimation using the feature extractor, classifier and discriminator. After some steps, the function becomes:

$$\begin{aligned}\min_{\theta^f,\theta^c,\hat{B}} \max_{\theta^d} \frac{1}{L+B} &\sum_{x,y\in\hat{L}} l((h(x,y)) \\ &+ \mu(\frac{1}{L+U}\sum_{x\in\hat{U}} g(x) - (\frac{1}{L+B} - \frac{1}{L+U})\sum_{x\in\hat{L}} g(x) \\ &+ \frac{1}{L+B}\sum_{(x,y^?)\in\hat{B}} l(h(x,y^?)) - \frac{\mu}{L+B}\sum_{x\in\hat{B}} g(x)\end{aligned} \tag{24}$$

Here $L$, $U$ and $B$ are the size of the labelled, unlabelled and queried data in a given active iteration. $y^?$ denotes the unknown label of query samples (Shui et al., 2020), which will be discussed later.
The first two lines in equation (24) are used to train parameters $\theta^f$, $\theta^c$ and $\theta^d$. The last line is used for the query of new datapoints. Using this notation, the

[10]To derive this function, Shui et al. (2020) use Kantorovich-Rubinstein duality. For more information on the derivation, please refer to their paper.

[11]Shui et al. (2020) require $g$ to be 1-Lipschitz, for which an additional term is added in estimation.

training and query step can be executed separately. One active iteration of WAAL consists of subsequently executing the training and query step.

**Training step**
The first two lines of equation (24) are used to train the model. The first line is the predictor loss, used to train the feature extractor and classifier. The second line is the adversarial loss, used for the feature extractor and discriminator. This adversarial loss uses both unlabelled and labelled data, enabling the feature extractor to be trained on both labelled and unlabelled data.

To practically implement the adversarial training, so optimizing the second line of equation (24), one more alteration must be made. When training a CNN, gradient descent related methods are used as discussed in Section 2.2.5. Batches of both the labelled and unlabelled data must be used. Shui et al. (2020) thus rewrite the adversarial training part such that it can be used with equally sized batches for both labelled and unlabelled data. The equal batch size is denoted as $S$.
There is an unbalance between the quantity of labelled and unlabelled data, with generally less labelled data available compared to the amount of unlabelled data. In order to still use the full unlabelled data, they re-sample labelled data which has previously been used in other batches, thus using the same labelled data in multiple batches. Then, the second line of equation (24) is rewritten to[12]:

$$\min_{\theta^f} \max_{\theta^d} \mu\left(\frac{1}{S}\sum_{x\in\hat{U}_S} g(x) - C_0\frac{1}{S}\sum_{x\in\hat{L}_S} g(x)\right) \tag{25}$$

$C_0$ is an additional hyperparameter to reduce the influence of the labelled data in each step. It aims to prevent excessive use of the labelled data, given that each labelled datapoint is used in multiple batches. In Section 4.4.3, this hyperparameter will be discussed in more detail.

Using this new formulation, training on one batch can be defined for each of the three networks. Given that training is done with batches of $S$ datapoints, the first line of equation (24) is reformulated using $S$ as well. For the classifier we obtain:

$$\min_{\theta^c} \frac{1}{S}\sum_{(x,y)\in\hat{L}_S} l((h(x,y)) \tag{26}$$

For the feature extractor we get:

[12]To ensure the 1-Lipschitz property of $g(x)$, in practice a gradient penalty is added this the equation to ensure this property. This will not be discussed in more detail, since it does not add to the understanding of the WAAL framework.

$$\min_{\theta^f} \frac{1}{S} \sum_{(x,y)\in\hat{L}_S} l((h(x,y)) + \mu(\frac{1}{S}\sum_{x\in\hat{U}_S} g(x) - C_0\frac{1}{S}\sum_{x\in\hat{L}_S} g(x)) \tag{27}$$

And lastly for the discriminator:

$$\max_{\theta^d} \mu(\frac{1}{S}\sum_{x\in\hat{U}_S} g(x) - C_0\frac{1}{S}\sum_{x\in\hat{L}_S} g(x)) \tag{28}$$

In the WAAL algorithm, the optimization is executed iteratively for different batches in the same order as above, using the gradient descent methods explained in Section 2.2.5. More information on the configuration of the gradient descent methods, as well as how epochs are defined, can be found in Section 4.4.3.

**Query step**

Having trained the different networks, the query step is executed. In equation (24), this is the last line:

$$\min_{\hat{B}} \frac{1}{L+B} \sum_{(x,y^?)\in\hat{B}} l(h(x,y^?)) - \frac{\mu}{L+B}\sum_{(x)\in\hat{B}} g(x) \tag{29}$$

The left part of this equation represents the uncertainty of the model, with $y^?$ unknown. Similarly to previous methods, uncertainty is thus estimated using the predicted classes. Instead of the commonly used entropy-related methods, WAAL combines two other metrics. Let $h(x, y_i) = \hat{y}_i$ be the prediction made by the classifier for class $i$ for a given unlabelled datapoint $x$. Then the first metric, the highest least prediction confidence score, is defined as:

$$\max_i \ -log(\hat{y}_i) \tag{30}$$

Thus, this metric equals the negative of the logarithmic score of the class with the lowest predicted value. The intuition is that a smaller highest least prediction confidence score means the prediction for the least likely label is relatively large, which is related to the model being uncertain. Hence, the aim is to select samples for which this score is as small as possible.

The second metric measures the uniformity of prediction confidence scores:

$$\sum_i -log(\hat{y}_i) \tag{31}$$

This metric is smaller when predictions are more uniform. Thus, the smaller this metric becomes, the more uncertain the model is. This metric is similar to entropy which is often used to measure uncertainty, see equation (11).

Any convex combination of these metrics can be used. For their paper, Shui et al. (2020) use a combination where both are given an equal weight of 0.5, which will be implemented for this research as well. Thus, the uncertainty metric becomes:

$$U(x) = 0.5 * (\max_i \ -log(\hat{y}_i)) + 0.5 * \sum_i -log(\hat{y}_i) \tag{32}$$

Notable is that this measure does not use MC dropout to estimate uncertainty. Hence, the query of WAAL is susceptible to the issues described in Section 2.3. In case both Bayesian AL methods and WAAL perform well, combining both approaches could be an interesting addition in future research[13]. A combination could be implemented by using Bayesian uncertainty estimation using MC dropout models instead of the uncertainty estimation currently used for WAAL, while keeping the training and diversity part equal.

The right side of equation (29) represents the diversity. Estimation for this expression is straightforward using the discriminator. The discriminator is trained to capture the difference between unlabelled and labelled samples by maximizing the Wasserstein distance. If the output of discriminator $g(x)$ is close to zero, it expects datapoint $x$ to be part of the labelled data while values close to one are related to the model expecting this datapoint to belong to the unlabelled data. Thus, the larger the output of $g(x)$, the more different the image is compared to the labelled data. Thus to ensure diversity, $g(x)$ should be maximized when querying new datapoints.

WAAL then uses equation (33) to calculate a score which represents both uncertainty and diversity. It then ranks all the scores and chooses the $B$ datapoints with the smallest score. The selected images thus have large uncertainty, due to a minimized $U(x)$, and a large diversity as a large diversity score $g(x)$ is preferred. In the equation, $\phi$ is a hyperparameter which is used to place more importance on either uncertainty or diversity. The selection of this parameter is discussed further in Section 4.4.

$$U(x) - \phi g(x) \tag{33}$$

#### 4.2.5 Overview implemented methods

In Table 5, an overview is given of the implemented AL methods, along with their main advantages and disadvantages. All methods use uncertainty when querying new samples. However, BALD and BatchBALD implement Bayesian methods for improved uncertainty estimation, and should therefore be able to better represent model uncertainty. BALD does not use any diversity, while BatchBALD does consider overlapping information within batches, but not with other labelled data.

[13]Due to the limited time available for this research, this will not be implemented even if it is promising.

WAAL explicitly aims to select images which are different to the currently labelled ones, hence this method is more focused on diversity. Lastly, WAAL trains the feature extractor with unlabelled data through adversarial learning, whereas the other methods do not use unlabelled data when training.

Table 5: Overview of the main characteristics of implemented active learning methods.

| | **Uncertainty** | **Diversity** | **Training using unlabelled data** |
|---|---|---|---|
| **BALD** | Yes, using Bayesian methods for better estimation | No | No |
| **BatchBALD** | Yes, using Bayesian methods for better estimation | Overlap of information within acquisition batches accounted for, but not explicitly aiming to find a diverse representation of the data | No |
| **WAAL** | Yes, but does not use Bayesian methods, possibly resulting in worse uncertainty estimation | Yes, uses a discriminator to take diversity into account when querying new datapoints | Yes, feature extractor is trained using unlabelled data by implementing adversarial training with the discriminator |

## 4.3 Performance evaluation

The dataset is divided in a train, validation and test set. As clear from the name, the train set is used to train the model. The validation set is used to compare the performance of different models and choose which model performs the best. The data is not used in training and hence the best performing model is chosen using unseen data. The methods to assess the performance are discussed later in this section. Lastly, the test set can be used to find the performance of the best performing model on again an unseen dataset, which has not been used in model selection.

### 4.3.1 Performance metrics and their interpretation

To evaluate the performance of the trained models, different performance metrics are used. A more detailed explanation of these metrics is given in Appendix B.2.

**Performance metrics**
Firstly, accuracy is used, which is often implemented in other research as well. Thus, accuracy is useful for comparisons with previous research. However, accuracy does not always give a fair representation of the performance of a model, as it does not take into account class imbalance resulting in possibly good performance already when only predicting the majority class. Therefore, it is important

to keep the class distribution in mind when assessing accuracy. To assess if models perform well over multiple classes, the macro F1-score is used as well. This metric assesses the performance in each class separately, after which the score is calculated by taking the average of the scores for these classes. Thus, this metric places equal performance on each class instead of each datapoint, which is the case with accuracy. The macro F1-score is used in some previous work as well, thus making it useful in some comparisons.

Both accuracy and the macro F1-score only measure whether the predicted class is exactly correct, without measuring a degree of wrongness on how far from the true class a datapoint is predicted. There is a clear order in classes for ADA, where predicting further away from the true class is less desirable. Predicting e.g. a destroyed building as majorly damaged is not as bad as predicting it to be undamaged, while both accuracy and macro-F1 score would see both predictions as equally incorrect. To measure the degree of wrongness as well, the Mean Absolute Error (MAE) is used. Similarly to accuracy, the MAE does not account for class unbalance. Therefore, a macro averaged variant of the MAE is used as well, which will be referred to as the Macro Averaged Absolute Error (MAAE). The MAAE is calculated by weighting the absolute error for each datapoint relative to the number of occurrences of the true class within the whole dataset, such that each class equally contributes to the calculated score. While calculated slightly differently, this method is similar to the macro-averaged mean absolute error introduced by Baccianella et al. (2009). As mentioned, more information on the implementation of all discussed metrics can be found in Appendix B.2.

**Interpreting accuracy, macro F1-score, MAE and MAAE**
For both accuracy and macro F1-score, a higher score is related to better performance of the model. For both, scores can be between zero and one. A score of zero means no prediction is correct, while a score of one means the model can predict all datapoints correctly. MAE and MAAE should be interpreted differently. As it uses the error, the MAE and MAAE should be as small as possible. In this case, a perfect model would result in an MAE and MAAE of zero. Thus in this case, the closer the score gets to zero the better the performance of the model.

The unweighted metrics, accuracy and MAE, show how well the model predicts considering all datapoints equally, which is useful to predict all separate buildings as good as possible. The macro averaged metrics are useful to assess if the model predicts well for different classes, or if it only predicts well in the majority classes. Accuracy and macro-F1 are used to assess how well the model performs at predicting the exactly right class, while also being commonly used in previous research thus making comparisons easier. The mean absolute error metrics are used to measure how far off the model is when predicting wrongly, thus taking into account the order in the classes.

### 4.3.2 Confusion Matrices

Additionally, confusion matrices are used for more detailed analyses. More information on confusion matrices and their interpretation, as well as an example, can be found in Appendix B.3.
While being very informative and intuitive to interpret, confusion matrices will not be used as much as the previously discussed metrics. Those metrics are simply one number which makes it easy to plot the performance over different active iterations and compare different models using this. This cannot be done using confusion matrices, and therefore they are only used in some cases where more in-depth performance evaluation is desired.

### 4.3.3 t-Distributed Stochastic Neighbor Embedding

To gain insight into how different methods select images and if this aligns with the expectations from the theory behind them, t-Distributed Stochastic Neighbor Embedding (t-SNE) plots are used. Using t-SNE, data with large dimensions can be plotted in a two-dimensional space. Using the CNN in this research, the features of images from the last fully connected layer are used as input for the t-SNE decomposition. Similarity in these features should indicate that images are similar. For all t-SNE plots, the model constructed in the last active iteration is used to extract features to keep plots consistent over iterations when evaluating them.

t-SNE aims to map datapoints that are close to each other in the high-dimensional space close to each other in the low-dimensional space as well, making it useful to visualize comparable datapoints close to each other. It tries to match the probability of points being close in the low-dimension to the probability of them being close in the high-dimension (Van der Maaten and Hinton, 2008). For more information on the implementation of t-SNE, refer to Van der Maaten and Hinton (2008).

For active learning, this can provide insights on how algorithms work. What would be expected is that random acquisition of images would result in selected images being scattered relatively evenly over the two dimensional space, while BALD would be expected to select more images close to each other. The plots can thus be used to assess if the algorithms work as expected.

While t-SNE can provide some useful insights, it is important to acknowledge some flaws in t-SNE plots. Firstly, different extracted features have a different importance in classification, which is not visualized in t-SNE. Thus, the visualization could visualize similarities while putting disproportional importance on

some irrelevant features. On top, the different t-SNE plots are created using features extracted by the model trained on the selected images, thus comparisons between them are not based on similarities in the same features. It also only aims to measure the closeness in a two dimensional space, hence the position of datapoints itself is irrelevant, only the distance to other datapoints should be assessed. Still given these flaws, it can give an indication on how different AL methods work.

In Figure 18, an example is given of a t-SNE plot, similarly to those that will be used in the results. It shows 300 labelled images plotted in a two-dimensional space using a colour related to their respective class. Images which are not yet labelled are plotted in grey. Important for interpreting the t-SNE plots is that only the relative positions of datapoints to each other is relevant. The numbers on the axes are not interpretable.

Using t-SNE with a lot of active iterations would results in too many plots, which do not add much onto the understanding of these models. Therefore, t-SNE plots for different iterations are only shown for a case where four active iterations of acquiring 500 images are used. In the plot of the first iteration, the 100 images selected randomly for the initialization are shown as well. Since the acquisition size could affect performance of different methods, t-SNE plots from the last iteration when an acquisition batch size of 100 is used are discussed as well. Results can be found in Section 6.1.

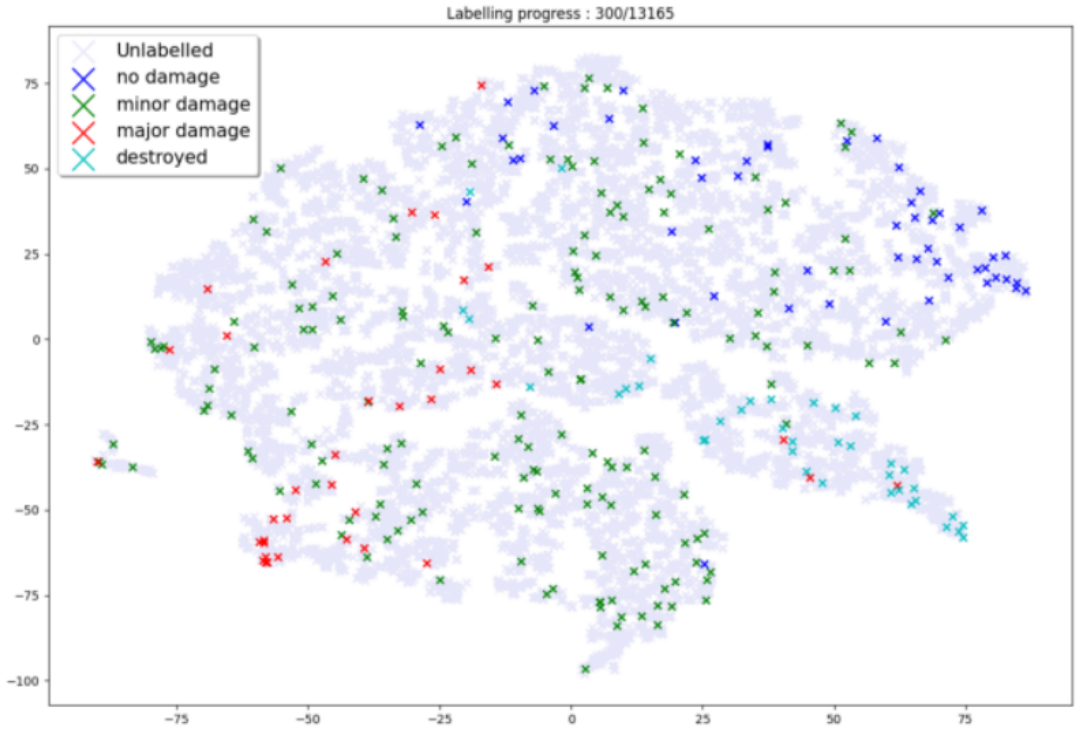



Figure 18: Example of t-SNE plot when using random acquisition.

### 4.3.4 Baseline model

In previous research, models trained on a large proportion of the data from the disaster on which the model is used, performed the best. To establish an 'upper bound' of the performance which could be achieved by the Caladrius model, a pre-trained model is fine-tuned using all training data from hurricane Matthew. This upper bound can be used to evaluate to which extent a model trained using active learning can approach the performance of a model trained on the full dataset. The scores of the baseline model are added in all plots containing the performance of AL methods.

## 4.4 Configuration of hyperparameters

Multiple hyperparameters that can affect performance are tested using different settings. In the comparison, the same seed is used for each setting. Still, randomness is likely to have a large influence on performances. The improvement gained by selected images is partly due to simple luck, as its true class is not known when selecting it, while bayesian AL methods include randomness in the selection algorithms as well. Therefore, multiple test runs should be used for each setting to select the best hyperparameters. However, given a large training time and only limited resources, this is not possible. Therefore, hyperparameters are

selected using only one run with a given random seed, meaning that randomness can have a substantial influence in this selection.
In all comparisons, every hyperparameter except for the one tested is kept constant. If this is not the case, it is explicitly mentioned. The standard values of hyperparameters which are not tested are as shown in Table 6. More information on these different hyperparameters, as well as the configuration of their values, is given in the remainder of this section.

Table 6: Standard hyperparameter settings.

| **Hyperparameter** | **Standard value** |
|---|---|
| Acquisition batch size | 100 |
| Loss type | Unweighted |
| MC dropout models ($T$) | 10 |
| $s$ | 500 |

### 4.4.1 Configuration of general hyperparameters

First the configuration of general hyperparameters, which are used in most methods[14], is discussed. Before discussing the hyperparameters for which different values are tested, the hyperparameters set to a fixed value are given. The batch size is set to 32. The starting learning rate is fixed to 0.001. This learning rate is decreased by a factor 10 when the loss did not improve significantly for 10 epochs[15]. These hyperparameter settings equal the default values used for Caladrius. In the remainder of this section, hyperparameters for which multiple values are tested will be discussed. The results for these hyperparameters can be found in Section 5.1.

**Epochs per active iteration**
The first hyperparameter that can influence performance is the number of epochs used in each active iteration. Higher values could increase performance, or sometimes result in overfitting (Liu et al., 2008). An additional consideration in this application is the increased training time for higher numbers of epochs. Tested values are 5, 20 and 50 epochs.

**Reinitialization of model**
Another choice that must be made is whether or not to use reinitialization of the

[14]For WAAL, most of these hyperparameters are used as selected based on this subsection as well, but the number of epochs and the learning rate are chosen separately due to the different type of training.

[15]These are the standard hyperparameters set for Caladrius, which are therefore used for most AL methods as well. When using WAAL, different values can be chosen since the architecture and training of Caladrius is altered when using that method.

model. Reinitializing the model is resetting it to the pre-trained model after each active iteration. The idea is that without it, the model may focus too much on information from images chosen in early iterations, as those images are used for the training in more iterations. Kirsch et al. (2019) advise using reinitialization when performing active learning using BatchBALD. To test whether this improves the performance, reinitialization is compared using BatchBALD and random acquisition. For the runs using reinitialization, the number of epochs after the last acquisition of a labelled batch is set to 100, since it does not use information on previously trained models.

**Weighting the loss function**
In a neural network, the loss is minimized. The loss can be calculated weighted or unweighted. When unweighted loss is used, this means that each datapoint has the same influence on the total loss. Using weighted loss, the influence of datapoints is weighted by the inverse of the number of datapoints which have the same label, similar to macro-averaging for the performance metrics. The idea is that weighted loss results in a better performance for less frequently occurring classes as these are given equal weight, while unweighted loss results in more focus on learning the classes with more occurrences. Thus it would in general be expected that weighted loss results in better performance using macro-averaged metrics, whereas unweighted loss results in better performance using unweighted metrics.

Important to note is that AL aims to identify those datapoints which are most informative for training the model. Given that the subset of most desirable training images is selected, using unweighted loss would be most logical. With weighted loss, the influence of datapoints selected by the model to be informative would be changed. Still, both types are assessed as weighted loss could still be desirable if it leads to clearly better macro-averaged performance.

**Acquisition batch size**
Another parameter which can be altered is the acquisition size in each active iteration. In general, a larger acquisition batch size would be preferred. The larger the batch size, the more convenient the labelling progress is since the number of times the model has to be retrained is reduced. However, larger acquisition batch sizes are linked to worse performance in active learning as mentioned in Section 4.2 and shown by Kirsch et al. (2019).

Therefore, multiple acquisition batch sizes are tested to investigate their influence on performance. This is tested for each AL method as some methods are more sensitive to the batch size than others. The smallest batch size used is 50. Smaller acquisition batch sizes are not tested, since lower batch sizes are impractical given the large CNN that must be retrained after each acquisition. Furthermore, ac-

quiring 100 and 500 images in each active iteration are tested. The number of active iterations is scaled to the batch size, with 40 active iterations for a batch size of 50, 20 for a batch size of 100 and 4 for a batch size of 500. Thus in total, 2000 images are labelled on top of the 100 random initialization images for each of the acquisition sizes. To keep the total number of training epochs comparable, these are scaled to the number of active iterations: 10 per active iteration for a batch size of 50, 20 for a batch size of 100 and 100 for a batch size of 500.

For the acquisition of 500 images per batch, two versions are compared: only resetting the learning rate in between active iterations such as implemented for the other acquisition batch sizes, and also resetting the learning rate within active iterations after each 20 epochs[16]. The last option is added as it turned out that even for random acquisition, batch sizes of 500 performed poorly. A possible explanation is that the learning rate's decay results in the model getting stuck in a local minimum and only improving marginally after a certain number of epochs. This resetting of learning rates while training the model is similar to stochastic gradient descent with warm restarts introduced in Loshchilov and Hutter (2016). Resets of the learning rate within active iterations will be referred to as 'warm restarts' in the remainder of this research as well.

### 4.4.2 Configuration of hyperparameters for Bayesian AL methods

**BALD**

Using BALD, the number of MC dropout models used ($T$) is an important factor in running time. There is a trade-off that must be made here: more stochastic forward passes results in better estimation of uncertainty of the model for images, but also in an increased time spent to select images. Doubling $T$ is related to twice as much time spent predicting on the unlabelled pool of images, which is the only step in the BALD heuristic that has a significant impact on computation time. The total computation time of an active iteration mainly consists of this prediction step and the training of the model using previously labelled images. Gal and Ghahramani (2016) find that even with only 10 forward passes, reasonable estimates of uncertainty can be found. While 10 MC dropout models should result in reasonable estimations, BALD is performed using 15 MC dropout models as well to see whether this could improve the performance significantly[17]. Furthermore, BALD is also implemented using 5 MC dropout models, to find out whether a lower number of MC dropout models could still yield reasonable results. Lastly, 2 MC dropout models are tested to check the influence of picking a very low value,

[16]Resetting every 20 epochs is chosen to make it comparable to acquiring 100 images in each batch, as using those settings the learning rate is reset after each active iteration of 20 epochs.

[17]Running BALD with 20 MC dropout models has first been tried. However, this led to memory issues when predicting on the unlabelled pool. If using more than 15 MC dropout models would be preferred, these issues can be resolved by either using a GPU with more RAM, or by using less efficient computations.

using which the estimation of uncertainty should be less accurate than for higher values.

**BatchBALD**

In Section 4.2.3, additional parameter $s$ is introduced on top of the number of mc dropout models $T$, which is also used with BALD. $s$ is used jointly with $T$ (as $m = T * s$, with $m$ the total number of configurations used) to determine the number of draws from all possible configurations of $\hat{y}_{1:b-1}$. For BALD the main time-consuming part in active learning was the prediction on the MC dropout models. For BatchBALD, the heuristic itself is time consuming as well. The the total time used in acquiring a new batch of images to label depends on both $T$ and $s$. Kirsch et al. (2019) set $s$ given $T$ such that $s * T = 40000$. However, this yields memory issues[18], while leading to a too large computation time as well. To find a good configuration of these parameters for this application, multiple combinations of $T$ and $s$ are tested. Combinations are tested since they both have similar affects: both should result in better performance at the expense of a higher computation time. In the heuristic, for both $s$ and $T$ a twice as large value also results in a twice as large computation time. $T$ also affects the time spent predicting with MC dropout models and hence its effect is larger on the total computation time.

$T$ equal to 5, 10 and 15 are tested with similar reasoning as previously discussed for BAAL. For $s$, 500 was previously used as a baseline, which combined with $T = 10$ results in a desirable computation time of approximately 30 minutes[19]. Additionally, $s = 250$ is tested to find whether a smaller value clearly yields worse results, with $s = 25$ tested to find out about the effect of an extremely small value. $s = 1000$ and $s = 2000$ are tested to find out whether larger values result in significantly better results.
Not each combination of $T$ and $s$ is tested to reduce the number of runs. For $T = 10$ and $s = 500$, combinations with each value of the other hyperparameter respectively are tested. By keeping either of the two constant, the influence of $T$ and $s$ can be determined separately. Additionally, combinations of $T = 5$ with $s = 1000$ and $s = 2000$ are tested. The computation time of these combinations is manageable, while it can be used to determine if compensating a smaller $T$ value with a larger $s$ value results in good performance. To provide an overview, the combinations used are marked with an X in Table 7:

[18]When using $T = 10$ and $s = 2500$, the memory of the used GPU (Nvidia T4, with 16 GB RAM) is too small to compute the heuristic efficiently.

[19]Together with 510, a computation time of 30 minutes was set as the target for an iteration.

Table 7: Combinations of $s$ and $T$ tested for BatchBALD, marked with an X.

| | | s | | | | |
|---|---|---|---|---|---|---|
| | | 25 | 250 | 500 | 1000 | 2000 |
| T | 5 | | | X | X | X |
| | 10 | X | X | X | X | X |
| | 15 | | | X | | |

### 4.4.3 Configuration hyperparameters WAAL

For WAAL there are more hyperparameters to be tuned, since its training procedure differs from the other methods as well. Therefore, decisions must be made between the hyperparameters set for training the Caladrius model, which are used with the other AL methods, and using the configurations suggested for WAAL. Also, some decisions must be made on the importance of both diversity and uncertainty in the model. Some of these hyperparameters are likely to influence each other. The best method to set the configurations would be to evaluate combinations of these hyperparameters. However, due to time constraints, they will be set separately. The parameters will be tuned in the order they are discussed in below, hence after one hyperparameter is tuned, the next parameter is tuned using this newly determined value. Note that other hyperparamaters, which are not discussed here, are set to the values chosen based on the results of Section 4.4.1, since these were already analysed at the point WAAL was implemented. The remainder of this subsection describes the hyperparameters and settings which are tested subsequently.

**Training epochs**

As training is different, the way epochs work is different as well. For the previously discussed more classic active learning techniques, one epoch simply consists of training the model one time having used all available data in the different batches. For WAAL, both the labelled and unlabelled data is used in training. One WAAL training epoch similarly consists of training on all data. However, there is an unbalance between labelled and unlabelled data quantities. In the WAAL framework, simply the number of batches that can be selected from the largest dataset is chosen, and when the other dataset has used up all images it simply restarts loading datapoints that were previously used already. This can lead to datapoints being used for training a lot of times within one epoch. An example is after the random initialization, there are 100 labelled images. The unlabelled dataset consists of approximately 13000 images. Therefore, labelled images would be used for training $13000/100 = 130$ times. When 20 WAAL epochs were to be used, similarly to the other implementations, this would result in using each image $20 * 130 = 2600$ times in a batch, similar to 2600 epochs for the other AL methods. This could results in overfitting on this small dataset, while also leading to a large computation time. Therefore, multiple ways of train-

ing in WAAL are tested. In the remainder of this thesis when considering the training of WAAL, a WAAL epoch will refer to training using each datapoint in the largest dataset once, while train set epochs will refer to the number of times each labelled image is used. The following settings are tested:

1. The first option combines the train set and WAAL epoch types. To train the classifier similarly to other implemented AL methods, the number of train set epochs is set to the same number of epochs as determined for the other AL methods. This number of epochs is chosen in the next section. Having reached the selected number of train set epochs, the parameters of the classifier are frozen. The current WAAL epoch is still finished for the discriminator and feature extractor. This ensures that the feature extractor and discriminator are trained using each unlabelled datapoint the same number of times. Since the discriminator is used to query the unlabelled data, not training it equally on all datapoints may result in unfair query between datapoints.

2. Updating the parameters of the feature extractor after having frozen the parameters of the classifier can have disadvantages as well. The extracted features, based on which the classifier determines the degree of damage, change. As the classifier is not retrained based on these new features, it may not perform well combined with the updated feature extractor. Therefore, performance is also tested for the same settings as above, but freezing the parameters for the feature extractor as well. This method is also faster, since only the relatively small discriminator network, with a small number of parameters compared to the feature extractor, is trained when finishing the WAAL epoch.

3. Another option is to simply stop training altogether after having reached the number of train set epochs required for the labelled images. This would be the fastest method and most comparable to other AL methods. However, it means the discriminator is only trained on a subset of the unlabelled data when the imbalance between labelled and unlabelled data is large[20], thus query may be less accurate.

4. When training, WAAL combines the cross-entropy loss with a Wasserstein distance loss function. Hence, cross-entropy loss may have a smaller effect when updating weights, which could lead to slower performance improvements compared to the other methods. Therefore, a higher number of epochs compared to the other methods is implemented as well. This number of epochs is defined in Section 5 after having set the number of epochs for the other methods[21].

[20] e.g. consider 20 epochs with a labelled dataset of 200 datapoints. In this case, only $20*200 = 4000$ of the unlabelled images are used.

[21] Note: this is only be implemented when not leading to excessive computation time.

5. Using 5 full WAAL epochs is tested as well in order to see whether their training scheme works better than the self-created training schemes tuned for Caladrius and for better comparison with other AL methods. 5 WAAL epochs are chosen as Zhan et al. (2022) showed that using WAAL, 5 epochs is sufficient to obtain good performance. More epochs did not significantly improve performance anymore (Zhan et al., 2022).

For testing these settings, Adam is used as optimizer[22].

**Optimizer and learning rate**
In Caladrius, the standard optimizer used is Adam with a learning rate of 0.001, whilst WAAL uses SGD with momentum and a learning rate of 0.01 (Shui et al., 2020). More information on these settings can be found in Section 2.2.5. Since a combination of the Caladrius and WAAL training is used, combinations of their respective optimizers and learning rates are tested: both Adam and SGD are tested with a learning rate of 0.01 and 0.001. The same optimizer and learning rate is used for each of the networks.

For the other AL methods, the learning rate is reduced by a factor 10 after no significant improvement of the loss is found for 10 epochs. This is implemented for WAAL as well[23]. For the learning rate reduction, epochs are defined as training set epochs. Thus, after the loss did not improve significantly in 10 times training on the full labelled dataset, the learning rate is reduced. The learning rate of the feature extractor, classifier and discriminator are reduced separately. The learning rate of these networks can thus be different from each other.

**Wasserstein distance balancing parameter ($C_0$)**
Multiple different methods for calculating $C_0$ are implemented:

- $C_0 = \frac{L}{U}$, with $L$ the total amount of labelled data and $U$ the number of unlabelled data. This choice of the hyperparameter is similar to the setting used by Shui et al. (2020)[24]. As $\frac{L}{U}$ is generally smaller than one, which in this application is always the case, this results in giving labelled data a lower weight than unlabelled data. The aim of giving the labelled data a smaller weight is to avoid excessive reusing of the labelled data (Shui et al.,

[22]In some initial runs where tuning hyperparameters systematically was not yet the target, the WAAL type optimizer did not seem to work well. Therefore, adam is used as long as the optimizer is not yet tuned following more thorough analyses.

[23]Note: learning rate reduction was only implemented after the previously mentioned hyperparameters were chosen

[24]In the code with their paper, this is the implemented method. In their paper, the term is slightly different, where the size of the acquisition batch relative to currently labelled dataset size is used as well. In this implementation, this results in giving the labelled train set an even smaller weight (up to half of $\frac{L}{U}$). Given $\frac{L}{U}$ is possibly too small already, which is discussed further in this section, the other weighting method introduced by Shui et al. (2020) is not tested.

2020). When only a small amount of labelled train set epochs is chosen, unlabelled datapoints are not always used in training. In such case, $C_0$ will be set proportionally to the amount of unweighted data used in training[25].

- $C_0 = 1$: When optimizing the discriminator using $C_0 = \frac{L}{U}$ with highly unbalanced data, the unlabelled data has a much larger influence on the discriminator loss and the Wasserstein distance part of the loss used for the feature extractor when considering a single optimization step. When optimizing the loss, close to perfect performance can be obtained by simply predicting each datapoint to be unlabelled. Therefore, other options may give better results. $C_0 = 1$ is therefore tested as well. This is the version implemented in the code supplied to the paper of Zhan et al. (2022).

- $C_0 = 0.5$: Additionally, a combination is implemented. The aim is to use a value where the contribution of the labelled datapoints to the loss is smaller than that of unlabelled datapoints in order to reduce the excessive use of the labelled data, but also not too small such that the model is not likely to just predict each datapoint to be unlabelled. For this purpose, $C_0 = 0.5$ is tested.

Given the explanations above, the decision on which setting of $C_0$ is chosen will mainly be based on the predictions made by the discriminator. The discriminator should not predict all datapoints with the same value, since this would make it useless for including diversity. To assess the performance with different choices of $C_0$, the mean and standard deviation of the estimations of the discriminator is given for both the labelled and unlabelled datasets, computed during each query step. Only if multiple settings of $C_0$ result in usable discriminator estimations, the performance of the classifier given the selected data will be assessed for the different values of $C_0$ to decide between them.

**Classification and Wasserstein distance loss trade-off for training ($\mu$)**
When training, a decision must be made about the importance of both the classification and Wasserstein distance loss components. This is mostly important for training the feature extractor. Shui et al. (2020) use two different values: $\mu = 0.001$ and $\mu = 0.01$, which we will test as well. Furthermore, $\mu = 0.1$ is implemented to assess whether giving the Wasserstein distance a larger importance could increase performance, as the feature extractor learns more from the unlabelled data. Lastly, $\mu = 1$ is tested as well, to assess whether not weighting the loss components yields good results.

**Diversity and uncertainty trade-off for query ($\phi$)**
Similar to training, there is a trade-off between diversity and uncertainty in the

[25]E.g. with 20 train set epochs, a labelled train set of 100 datapoints and 10000 unweighted datapoints, $\frac{L}{U} = 0.01$, while only 4000 of the unlabelled datapoints are used. Therefore in this case, $C_0 = \frac{1}{20} = 0.05$ is used

query as well. Therefore, $\phi$ must be tuned. The larger $\phi$, the larger the influence of the diversity component in query (see equation (33)). In general, $\phi$ is chosen much larger than $\mu$, since the uncertainty component generally has larger values compared to the discriminator values[26]. Shui et al. (2020) use two different values: 5 and 10, which are tested in this application as well. On top, $\phi = 2$ is tested. This value is added due to a difference in the number of classes used in the research by Shui et al. (2020) and this application. In this research, 4 classes are used instead of 10 classes in their research. Due to the smaller amount of classes, the score of the uniformity uncertainty metric is generally smaller (see equation (31)), thus the total uncertainty score is smaller as well. The magnitude of the diversity score does not depend on the number of classes. The smaller value of $\phi$ is tested to compensate for this difference.

**Training data batch size**

A different batch size is used for the labelled train data. Whereas for other methods, the default Caladrius batch size of 32 labelled images is used, this is decreased to 16 for WAAL. The reason for this is simple. Given that the training uses unlabelled images as well, the number of images used in training is twice as large for a given labelled images batch size, resulting in more memory being used. Using 32 labelled images together with 32 unlabelled images therefore results in memory issues on the used GPU when training[27].

[26]The discriminator output is always in $[0, 1]$, while the uncertainty component can grow significantly larger. The smallest possible value of the uniformity uncertainty metric, when each class is given the same predictive value, is $-4 * log(0.25) \approx 2.4$ is already substantial larger. The same holds for the standard deviation.

[27]A Nvidia Tesla T4 GPU with 16 GB memory is used. This only has a shortage of approximately 200 MB of memory. While it is tried to free all memory which was unnecessarily stored, it may be possible to still reduce memory usage to a point where a batch size of 32 can be used. Furthermore, GPU's with a larger memory would certainly enable the usage of this larger batch size.

# 5 Results configuration of hyperparameters

As mentioned, the hyperparameters are configured by testing their performance in different settings. The tested hyperparameters, their settings and explanations on why these are tested are explained in Section 4.4. In this section, the chosen configuration is discussed. In Appendix C, a detailed explanation is given for each of the chosen settings, along with figures and tables containing the results based on which the decisions are made.

## 5.1 Configuration of general hyperparameters

First the configuration of the general hyperparameters is discussed, with more detailed explanations being given in Appendix C.1. Most decisions are based on results when acquiring a total of 2000 images in active iterations, with an acquisition batch size of 100.

While acquiring images with an acquisition batch size of 100 was chosen as a baseline value, this hyperparameter is tuned as well. Given the results presented in the Appendix, two scenarios are chosen to be investigated further when comparing different AL methods over multiple runs[28]:

- Acquiring 500 images with AL using an acquisition batch size of 100
- Acquiring 2000 images with AL using an acquisition batch size of 500 with warm restart

These scenarios simulate different types of settings in which AL could be implemented by 510. The first scenario simulates a relatively small scale disaster, where resources are likely limited. In such a case, data would have to be labelled internally by employees of 510. Consequently, the total number of images that can be labelled is small. Therefore, only 500 images are labelled using active learning in this scenario, with an acquisition batch size of 100.

The second scenario simulates a larger disaster which captures a lot of media attention. In this case, 510 is more likely to have a group of volunteers available to label data. In this situation, it is possible to label a larger total amount of images. A dataset of 2000 images will thus be labelled in this scenario. When using groups of volunteers, labelling larger batches of data each time is more convenient. When using an acquisition batch size of 500 instead of 100, volunteers only have to be asked for help in 4 different iterations with retraining in between. Additionally, labelling 500 images could be done in groups where one expert helps volunteers with images they are unsure of. Therefore an acquisition batch size of 500 is used

[28] On top of the number of images selected using AL, both scenarios use a randomly chosen initial set of 100 labelled images.

to simulate this scenario.

The other hyperparameters are set as given in Table 8. As mentioned, more information on the considerations when setting these hyperparameters is given in Appendix C.1.

Table 8: Configuration of the general hyperparameters.

| | | Scenario | |
|---|---|---|---|
| | | 500 images in total, acquisition bacth size of 100 | 2000 images in total, acquisition bacth size of 500 |
| Hyperparameter | Epochs per active iteration | 20 | 100, with warm restart after every 20 epochs |
| | Reinitialization of the model | No | No |
| | Weighted loss function | No | No |

## 5.2 Configuration of hyperparameters for Bayesian AL methods

Next, the configuration of additional hyperparameters for the Bayesian AL methods, which are BALD and BatchBALD is given. For BALD, the number of MC dropout models used is set to 10 ($T = 10$). For BatchBALD, $T = 10$ is used as well, with $s$ set to 500. For both $T$ and $s$, using different values did not have a clear impact on performance. Therefore, $T$ and $s$ are mainly set based on a trade-off between using values resembling those used in the papers by Gal et al. (2017) and Kirsch et al. (2019), and computation time. More information is given in Appendix C.2.

## 5.3 Configuration hyperparameters WAAL

For WAAL, more different hyperparameters need to be tuned since a different type of training is used. These are tuned using the scenarios described before, so acquiring 500 images using AL with an acquisition batch size of 100 and acquiring 2000 images with AL using an acquisition batch size of 500, after a random initialization step using 100 images. Only the number of epochs are tuned for both scenarios separately, the other hyperparameters are tuned using the first scenario due to limited time. In Table 9, the chosen hyperparameters are given.

An important finding when tuning the hyperparameters is that the discriminator does not always learn the differences between labelled and unlabelled data well. The predicted output is generally larger for the unlabelled data, which is as expected, but differences are small in some cases. More information on this issue, as well as the details of the configuration of the hyperparameters, are given in Appendix C.3.

Table 9: Configuration of the WAAL hyperparameters.

| | | Scenario | |
|---|---|---|---|
| | | 500 images in total, acquisition bacth size of 100 | 2000 images in total, acquisition bacth size of 500 |
| Hyperparameter | Epochs per active iteration* | 50 train set epochs, with warm restart after every 20 train set epochs | 100 train set epochs for feature extractor and classifier, finishing last WAAL epoch for discriminator, with warm restart after every 20 train set epochs |
| | Optimizer and learning rate | Adam with a starting learning rate of 0.001 and learning rate reduction | |
| | $C_0$ | 1 | |
| | $\mu$ | 0.01 | |
| | $\phi$ | 5 | |

*Note: In WAAL, two types of epochs are used: train set epochs and WAAL epochs. See Section 4.4.3 for an explanation on these epoch types.

# 6 Results

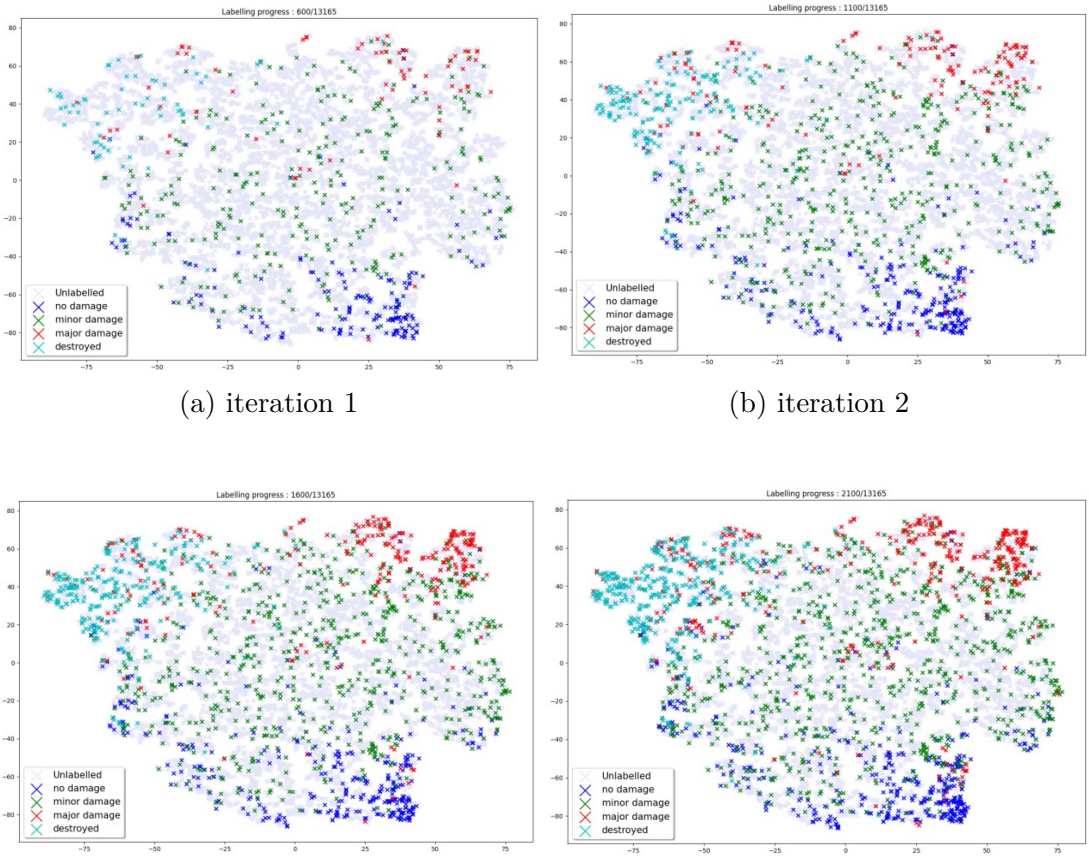
In this section, results comparing the different AL methods and random selection of datapoints are presented. First, t-SNE plots are displayed for each method. Next, results comparing all methods using the mean and standard deviation of the different evaluation metrics over multiple runs are discussed. For a small selection of runs, confusion matrices are discussed as well for more detailed analyses. All models are tested using hurricane Matthew, with models being pre-trained using the Moore tornado, Tuscaloosa tornado, Joplin tornado and hurricane Michael.

## 6.1 Visual results t-SNE

t-SNE plots, created with the different AL methods, are shown to visually assess if these methods behave as expected. Important to note is that these plots are only snapshots of one run, hence results are not necessarily representative of the behaviour of the methods in all cases. Furthermore, it is important to keep the flaws with t-SNE plots mentioned in Section 4.3 in mind. In particular, the reason for certain images to be clustered together can be due to a lot of features, which are not necessarily important for how images are classified and queried. The plots should thus only be used to get some intuition with how methods select images.

**BALD**

First in Figure 19 the plots can be found for BALD. As may be expected, images from some areas are chosen more often than others. In this case, clearly more images are labelled which are plotted in the top left, top right and bottom right of the two-dimensional space of the t-SNE plot. Especially in the middle of the space, fewer images are labelled.



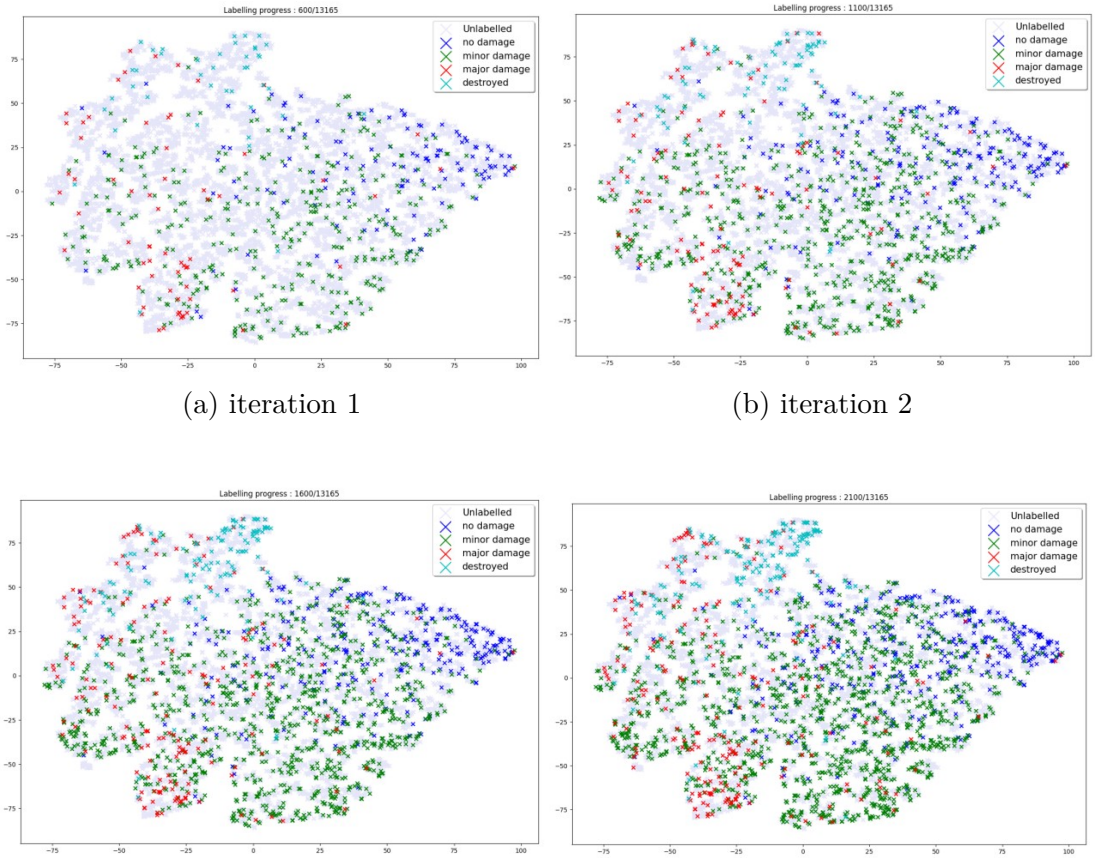

(a) iteration 1 (b) iteration 2

(c) iteration 3 (d) iteration 4

Figure 19: t-SNE plots for BALD in different active iterations, using four active iterations selecting 500 images in each iteration. Training is performed using 100 epochs in each iteration with warm restart each 20 epochs.

**BatchBALD**

In Figure 20, similarly the t-SNE plots of BatchBALD can be found. BatchBALD seems to select more different types of images, as the chosen images are more evenly spread over the space. This is as expected since it accounts for overlap of information between datapoints in batches. Still, images in some areas are more often labelled than in others. On the left, images are less often labelled than on the right of the two-dimensional space.



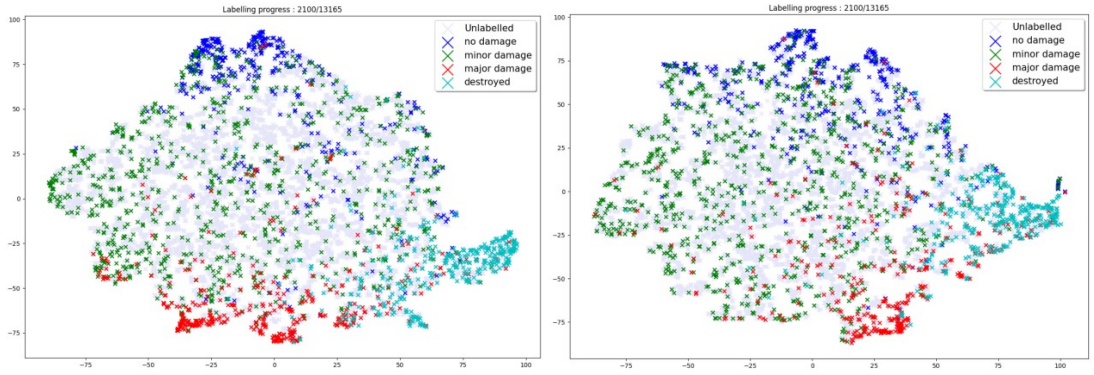

(a) iteration 1 (b) iteration 2

(c) iteration 3 (d) iteration 4

Figure 20: t-SNE plots for BatchBALD in different active iterations, using four active iterations selecting 500 images in each iteration. Training is performed using 100 epochs in each iteration with warm restart each 20 epochs.

Figure 21 contains the t-SNE plot after the last active iteration for BALD and BatchBALD when the same number of images is acquired with an acquisition batch size of 100, hence showing all selected images over the iterations together[29]. This is displayed as well since the acquisition bacth size could affect the performance of the different Bayesian AL methods. With smaller batch sizes, BatchBALD and BALD behave more similar, with both acquiring more images from certain regions in the plot.

[29] All used hyperparameters are set to the default values previously introduces in the introduction to this Section.

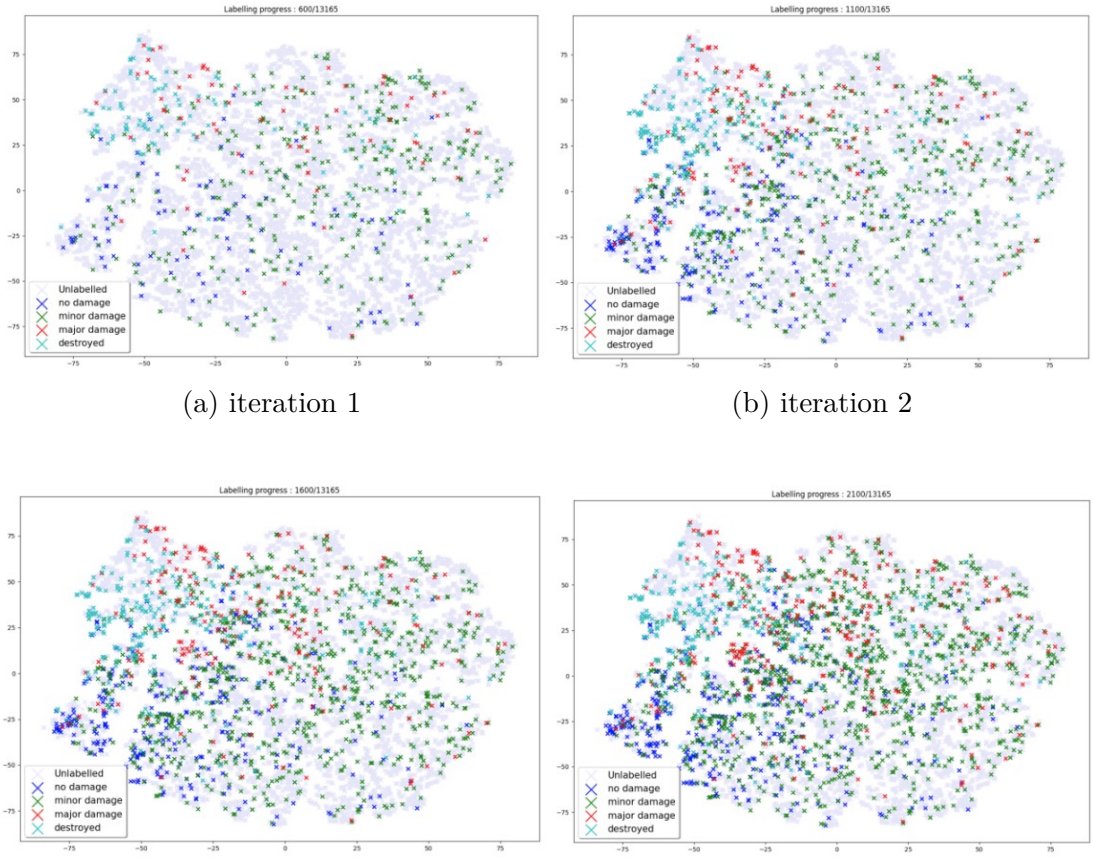



(a) BALD (b) BatchBALD

Figure 21: t-SNE plots after 20 active iterations for BALD and BatchBALD with an acquisition batch size 100.

**WAAL**

Figure 22 contains the t-SNE plots for WAAL. Generally, labelled datapoints are divided fairly even over the two-dimensional area, though some parts are represented somewhat more than others. Datapoints in the middle left of the two dimensional space are labelled more often, while datapoints in the right bottom are labelled less often. Furthermore, the colors representing datapoints from different classes seem more mixed over the space compared to the t-SNE plots for the other methods.




(a) iteration 1

(b) iteration 2

(c) iteration 3

(d) iteration 4

Figure 22: t-SNE plots for WAAL in different active iterations, using four active iterations selecting 500 images in each iteration. Training is performed using 100 epochs in each iteration with warm restart each 20 epochs.

**Random**

In Appendix Figure D.1, t-SNE plots can be found for random selection as well. Selected images are evenly distributed over the space in each iteration, as could be expected when randomly selecting datapoints.

## 6.2 Results comparing different AL methods

In this section, the results comparing AL methods using multiple runs are discussed. For each method, the same seeds are used when randomly selecting the initial dataset of 100 images for the different runs. Still, the performance after training on this random initial dataset may differ slightly between methods. This is due to some stochastic processes within the training of the model, such as dropout.

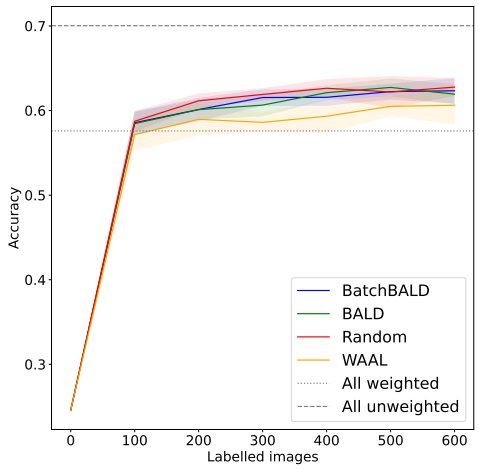
In this section, the mean results for each method are given together with the

standard deviation. Results for an acquisition batch size of 100 are based on 10 runs. For an acquisition batch size of 500, only 5 runs are used due to its larger computation time.

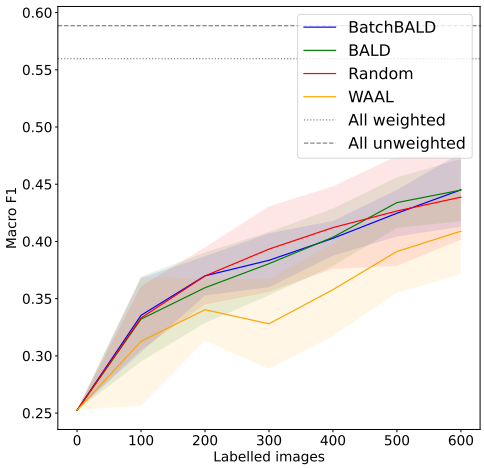
### 6.2.1 Acquisition batch size of 100

First, results using an acquisition batch size of 100 are discussed. As explained in Section C.1, in total 500 images are selected using AL techniques, while 100 images are labelled in the random initialization. The results are displayed in Figure 23. For each of the metrics, fine-tuning the model using data from the newly occurring disaster results in clear improvements in the scores of all performance metrics. For the unweighted metrics, the improvement is mainly found in the random initialization, while the performance using macro-averaged metrics results improves more gradually over different active iterations. For the unweighted performance measures, improvements after the random initialization step are difficult to distinguish. Therefore Figure 24 displays the same plots, but zoomed in to show the performance of AL methods after the random initialization better.

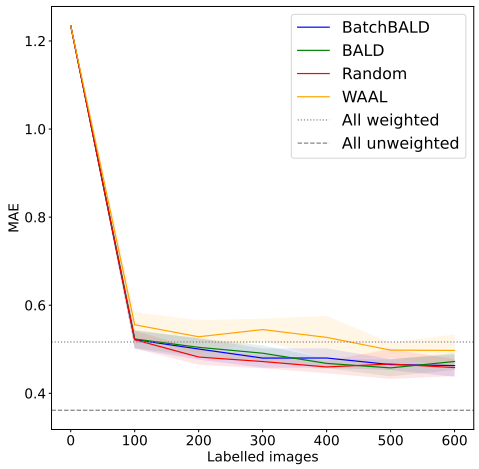
For each of the metrics, BatchBALD (in blue), BALD (in green) and random (in red) show similar performance. Their respective scores fluctuate around each other, while generally being within one standard deviation away from each other. WAAL (in orange) is the only method which clearly performs differently from the other methods. For each of the metrics, the scores for WAAL are worse than those for the other methods. This worse performance is already observed after the random acquisition, when no query step is performed yet.



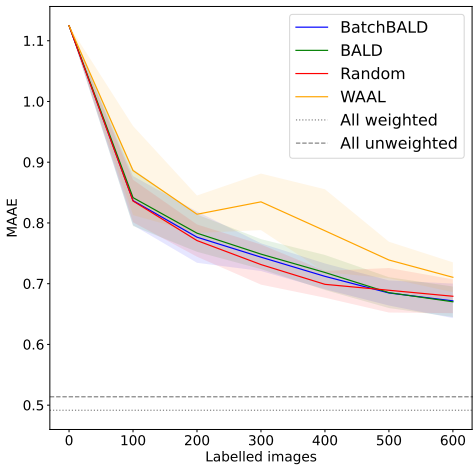

(a) Accuracy




(b) Macro-F1

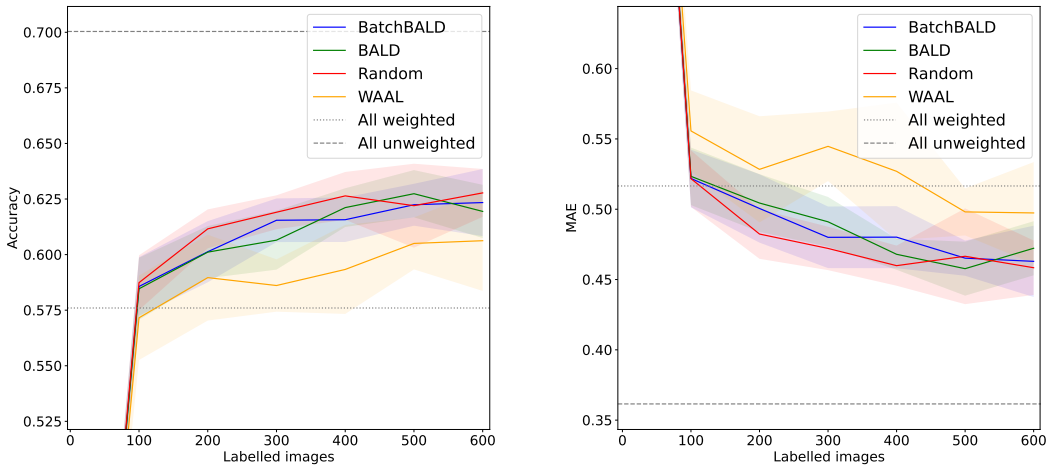



(c) MAE




(d) MAAE

Figure 23: Mean and standard deviation of the performance of different methods over 10 runs for an acquisition size of 100.




(a) Accuracy

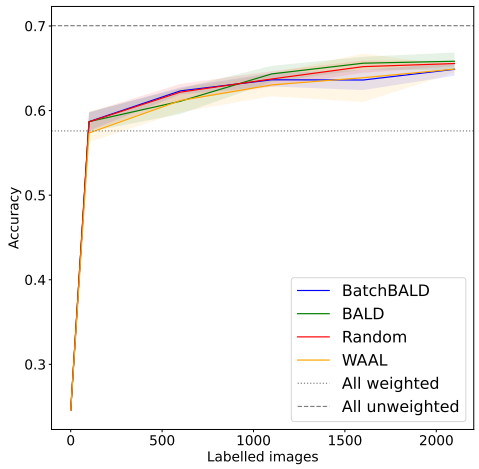
(b) MAE

Figure 24: Zoomed mean and standard deviation of the performance of different methods over 10 runs for an acquisition size of 100, for the unweighted metrics.

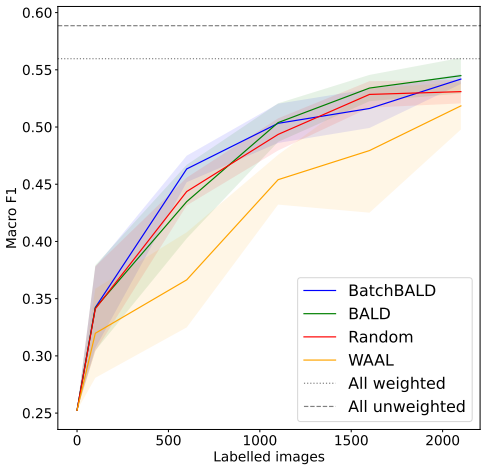
### 6.2.2 Acquisition batch size of 500

Next Figure 25 contains these results similarly for an acquisition batch size of 500, with Figure 26 containing zoomed plots for the unweighted metrics. Similarly to results with an acquisition batch size of 100, all metrics show improving performance when fine-tuning the model. Again, the improvements considering unweighted metrics are mostly found in the random acquisition step, with the improvements considering macro-averaged metrics being found more gradually over different acquisition steps. Additionally, the performance after all active iterations is better compared to the performance previously found in the other scenario where fewer images were labelled.

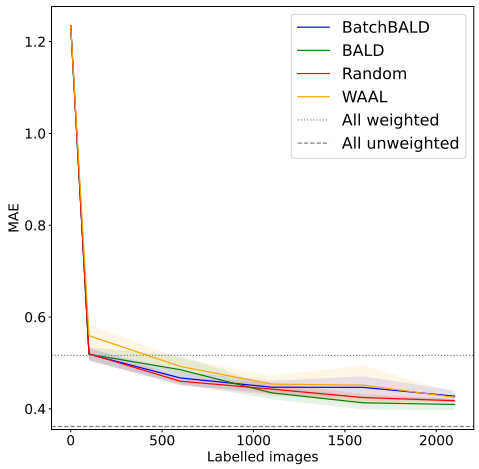
For macro-averaged metrics, differences between the different methods are similar to those found with an acquisition batch size of 100. BALD, BatchBALD and random all show similar performance, while WAAL performs worse than other methods. When the unweighted metrics are considered, WAAL performs more comparably to the other methods, while with the smaller acquisition size it was outperformed by all methods when considering the unweighted metrics as well.



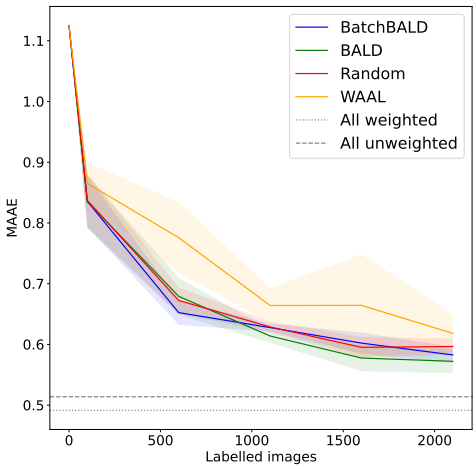

(a) Accuracy




(b) Macro-F1

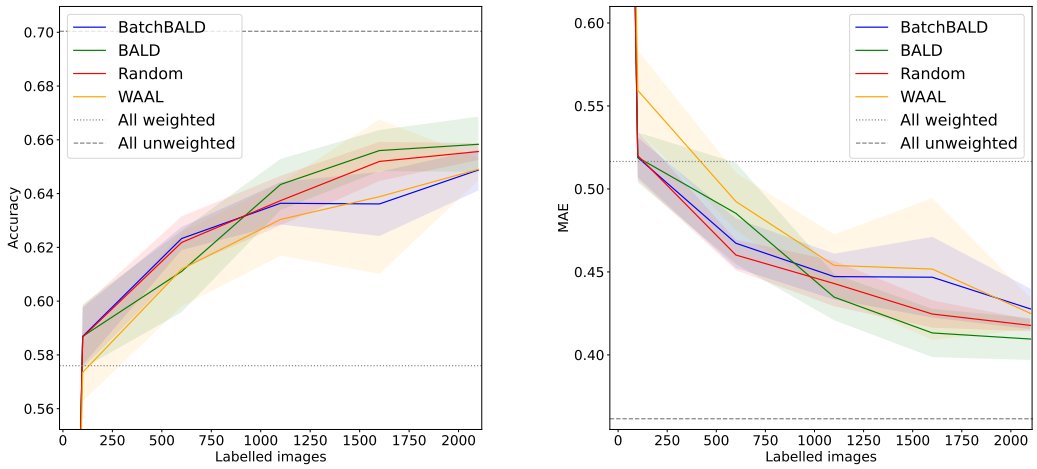



(c) MAE




(d) MAAE

Figure 25: Mean and standard deviation of the performance of different methods over 5 runs for an acquisition size of 500.




(a) Accuracy

(b) MAE

Figure 26: Zoomed mean and standard deviation of the performance of different methods over 5 runs for an acquisition size of 500, for the unweighted metrics.

## 6.3 Comparison pre-trained model, fine-tuning and training on full data with confusion matrices

Next, Figure 27 contains confusion matrices for predictions made by the pre-trained model, fine-tuned models with acquisition batch sizes of 100 and 500, with the total labelled datasets containing 600 and 2100 datapoints respectively and random acquisition used to choose images[30], and lastly a confusion matrix with predictions made using a model which was trained on the full dataset of hurricane Matthew. The full train dataset consists of 13165 datapoints[31]. The figure is used to assess how the pre-trained and fine-tuned models compare with a model trained on the full dataset, as well as to identify how well the predictions are for different classes. Important to note is that these confusion matrices only correspond to the performance on the test data after a single run for each of these options.

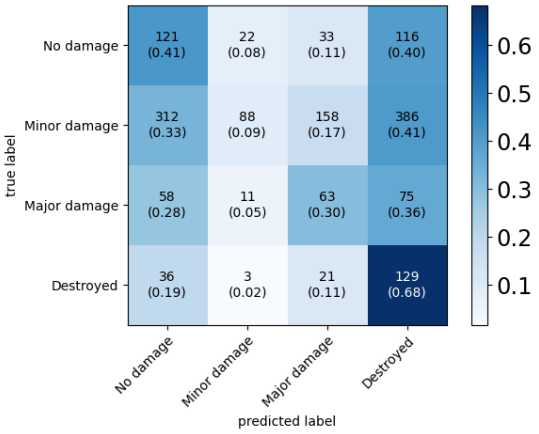
As was noticed using the performance metrics before, the pre-trained model does not perform well. Mostly, it predicts buildings to be either undamaged or destroyed. Only buildings which are destroyed are usually correctly classified, with

[30]Since AL methods do not seem to clearly improve over random acquisition, random acquisition is used in these analyses.

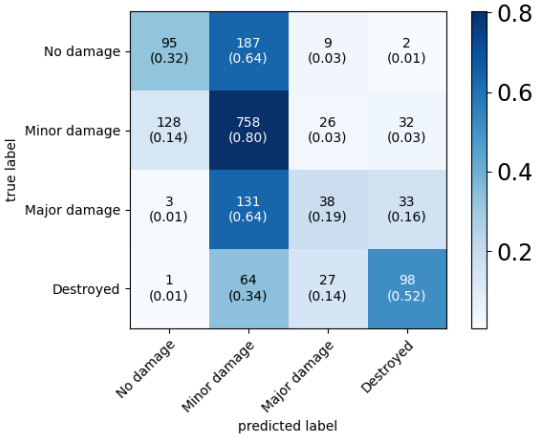
[31]Both the fine-tuning using a small dataset and training on the whole train set are performed using unweighted loss. When training on the full dataset of hurricane Matthew, the pre-trained model was used as well to make it more comparable.

reasonable performance for undamaged buildings as well. When 600 datapoints from Matthew are labelled, it performs better with the predictions generally being closer to the diagonal. Most predictions are at most one class away from the true class. Buildings which have no damage or major damage are often classified wrongly as minorly damaged. When 2100 datapoints are labelled, results are similar. However, the predictions of the model for buildings which have major damage or are destroyed clearly improve, with a minor improvement for minorly damaged buildings as well. When the full train set of hurricane Matthew is used, predictions of destroyed buildings are more often correct, while results are similar to fine-tuning with 2100 images in the other classes.

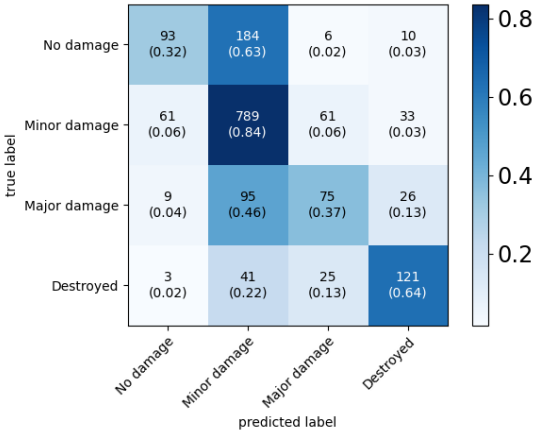
Generally, fine-tuning the model on a small dataset results in similar confusion matrices to the one created using the full dataset of hurricane Matthew. The models mainly predict undamaged and majorly damaged buildings incorrectly as minorly damaged. This is likely due to 57 percent of buildings being minorly damaged, in combination with an unweighted loss function. Probably, this could be improved by using a weighted loss function. Still, predicted classes are close to the diagonal, so the model generally predicts close to correct classes.



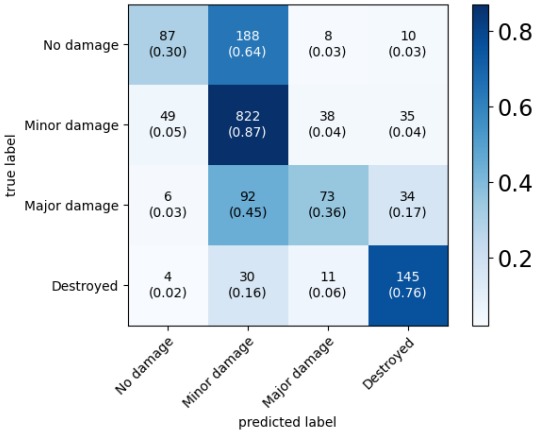

(a) Model pre-trained on other wind-type disasters




(b) 600 images from hurricane Matthew used for fine-tuning




(c) 2100 images from hurricane Matthew used for fine-tuning




(d) Model trained using all images from hurricane Matthew

Figure 27: Confusion matrices for predictions of models trained on different datasets.

# 7 Discussion and Recommendations

This section contains possible explanations for the behaviour of different implemented methods shown in the results. Additionally, recommendations are given based on the results and added explanations.

**Visual interpretation using t-SNE plots**
The visual interpretation of the t-SNE plots is discussed first. BALD clearly focusses more on certain areas, but it still selects datapoints throughout the whole t-SNE area. Thus, it still obtains information on datapoints throughout all locations visualized in the t-SNE plot, while focusing more on some areas. When comparing the behaviour of BatchBALD with BALD using an acquisition batch size of 500, BatchBALD clearly focuses less on certain areas. It queries datapoints more evenly distributed throughout the t-SNE plot. This is in line with expectations, as BatchBALD takes into account the overlap in mutual information between images selected in a query batch.

When assessing plots with an acquisition batch size of 100 images in each iteration, the behaviour of both is more similar. From a theoretical perspective, this behaviour is not unexpected. When only few images were previously selected in a batch, overlap in mutual information is likely to be smaller when acquiring a new image, leading to more images being acquired from certain regions the model finds difficult. When a large amount of images has already been acquired in a batch, a substantial number of images was likely selected from these difficult regions already, leading to a larger overlap in mutual information. The BatchBALD score for new images in these regions is therefore smaller. Therefore, when a large number of images has already been acquired within a batch, BatchBALD is more likely to select images from less researched areas of the data as well. Thus, a larger batch acquisition size could lead to more diverse selection with BatchBALD, which in turn results in larger differences with BALD.

Still, it may have been expected that selecting images in smaller batches would result in a more diverse representation on the data. After having selected images from the more difficult regions, the model has learned from those images and should become better at classifying images from those regions. Having gotten better at predicting in these regions, both BALD and BatchBALD scores should decrease for similar images, and thus the model can focus more other areas. However, both heuristics still focus more on some areas of the data, which indicates that it still finds these regions difficult even though it has already seen similar datapoints. This could indicate that it still learns more from datapoints that have substantial overlap with previously selected images compared to images it finds easier to classify but which do not resemble previously selected data.

For WAAL, the t-SNE plots show expected behaviour as well. Datapoints are

chosen reasonably diverse, while still focussing more on some areas compared to others. This is what would be expected given that it combines uncertainty and diversity. The t-SNE plots for WAAL show similar query behaviour to Batch-BALD when the same acquisition batch size is considered, while showing a more diverse acquisition of labelled datapoints compared to BALD. Given that Batch-BALD incorporates diversity, while BALD does not, this is consistent with our expectations.
Interestingly, the t-SNE plots show that WAAL has more difficulties in clustering together datapoints of similar classes. The labelled datapoints are often close to datapoints from different classes as well. The t-SNE plots were created by using the output of the last fully connected layer of the model. The larger difficulties in clustering datapoints from the same classes together indicates that the WAAL model did not manage to extract as much information to learn the difference between the different types of datapoints from the images compared to other methods. This is also shown by its worse performance using the different performance metrics.

Given the visual representations, the methods seem to largely behave as expected. However, considering the comparisons using different performance metrics, none of the AL methods clearly outperforms random selection, while WAAL even performs worse.

**Possible causes of the bad performance of active learning**
First some possible causes of the bad performance of AL in general are introduced. Later in this section, possible reasons for WAAL performing even worse than random and the other AL methods are discussed

A possible explanation for the bad performance of AL could be that for fine-tuning the model, all images are comparably informative irrespective of how difficult the model finds classifying them. This could be the case when the main difficulties for the model are due to factors such as the lighting of the images before and after the disaster and the angle at which the images are taken. This type of factors are similar over all images from a new disaster, while possibly different from other disasters. Therefore, all images contain similar information on these factors. If such differences are the main difficulty for the pre-trained model, while features it needs to recognize to classify images are similar given these other differences, its classification uncertainty does not tell much about how informative images are. Selecting images based on it therefore would not affect performance compared to randomly selecting images.
Given these possible differences between the pre-trained data and the new data, it could be interesting to perform the pre-training differently. At this moment, pre-training is performed using only the labelled data from previously occurred disasters. This could potentially be improved by altering the pre-training of the

model using a framework inspired by WAAL's adversarial training. Instead of using the discriminator for labelled and unlabelled data, it could be used to discriminate between the labelled data from previous disasters and data from the newly occurred disaster. Given that the feature extractor aims to minimize the Wasserstein distance, it will thus be trained to extract similar features from both datasets. Thus, the pre-trained classifier will have learnt to classify based on features more similar to those extracted from the newly occurred disaster, possibly improving results in the new situation. A drawback of such a method is that it would need pre-training to, at least partly, be performed after the images from the newly occurred disaster are available. This could slow down the ADA process. However, when random acquisition is used for fine-tuning, this WAAL type pre-training could be performed at the same time as the labelling effort. It could thus be interesting to test such a method in the future.
Ge et al. (2023) also mention differences in the imaging environments and data sources, as well as differences in buildings, to result in worse performance of models trained on data from previous disasters. They suggest a different solution than the WAAL inspired framework proposed above. Instead of training the model to extract similar features from images from different disasters, their method augments images from the previously occurred disasters to resemble the style of the new data. This is implemented using a framework based on generative adversarial networks (Ge et al., 2023). 510 could test this method as well.

Furthermore, there are some main differences between the type of data used in this research and in developing and testing the different AL methods. When testing the AL methods, the used data is always nominal. Examples are the MNIST dataset, where images of hand-written digits are used, and the CIFAR dataset, which contains images of objects such as airplanes and cars. In these cases, there is no degree in how wrong classifications are: if not fully correct, the classification is completely wrong. In this case, uncertainty in predictions between categories is therefore closely related to the model not predicting well on this given image.
As xBD data is ordinal, the degree in wrongness of classifications is important. Predicting major damage in case of a destroyed building is less wrong than predicting undamaged. Thus, if the model is uncertain about a datapoint belonging to either of two consecutive classes, this datapoint is probably less informative compared to an image for which it is uncertain between two classes which are further away from each other.
Adding to this, the difference between classes is less strict compared to the other types of datasets. Whereas in CIFAR datasets an image clearly contains e.g. an airplane or a car, this is more nuanced when classifying damage. Some images may be majorly damaged, but very close to destroyed. Other buildings could clearly be majorly damaged, and not that close to destroyed. In this case, the model should be more uncertain about its prediction of the former compared to the latter. However, this uncertainty would not be related to the model not

knowing how to classify the image, but simply to the images truly being more uncertain. Thus this image of which the model is more uncertain may not be more informative.
Therefore, classification uncertainty may not be a correct metric for informativeness of images in Automated Damage Assessment. Instead regression related metrics, which account for distance from the true label, could be more appropriate. An example of such method, based on the MC dropout method by Gal et al. (2017), is given by Tsymbalov et al. (2018). They use a regression based metric of uncertainty. It could be interesting for 510 to implement this method, especially since the code for Bayesian AL using MC dropout is readily implemented. Using this code, this new method could relatively straightforwardly be implemented. Still, some other issues are discussed which would not be solved by simply using methods developed for regression or ordered classification. Therefore, it is doubtful whether this method, with large similarities to the currently implemented methods, would yield desirable results.
Deep active learning for image data regressions is less researched compared to classification. It could be interesting to follow new developments in this area, as these may provide useful insights for ordinal classification as well.

Besides ADA using another type of classification, the data is more complex as well. Instead of images of different objects, all images in damage assessments contain the same object, which is a building. In this application, we aim to classify images based on differences within these same objects, which is a more challenging task compared to classifying different objects. The model should learn detailed differences between images, e.g. it should recognize whether roof elements are missing. Making it even more challenging, buildings themselves differ as well, with e.g. the type of roof and size of buildings differing. The model should thus learn to recognize the damages in before- and after- disaster images of different types of buildings. Given the difficult task, the model may already be more uncertain by default. It could thus be uncertain about a large proportion of images, resulting in less useful selection by uncertainty based AL methods. In particular, there may be fewer differences in uncertainty-based sampling and random sampling when the model is more uncertain about many images.

Additionally, experts found that two to three percent of images was labelled wrongly in the xBD dataset (Gupta et al., 2019). Thus, even human annotators struggle with the right classification sometimes, which underlines the difficulty of the task explained before. Especially for AL using uncertainty, the wrong labels could yield worse results. Likely, the images which are labelled wrongly are those that are relatively difficult to label. The model is likely to struggle more with these difficult images as well, yielding a higher uncertainty and hence larger probability of the wrongly labelled image being chosen.

Whereas the bad performance of all active learning methods was not expected, Saifullah et al. (2023) recently found similar results. Their research focuses on classifying images of documents. For their research they use multiple models and datasets. For two of their tests, a model pre-trained on ImageNet is fine-tuned using a document classification dataset. In both these tests, uncertainty based active learning significantly outperformed random sampling. In another test, they experiment with a model pre-trained on one document classification dataset, which is thereafter fine-tuned using another document classification dataset. This setting is similar to the implementation in this thesis. When fine-tuning the model in such a setting, none of the active learning methods outperforms random selection. Their results indicate that AL does not perform well when fine-tuning models which are pre-trained on similar data (Saifullah et al., 2023), which is consistent with the findings of this thesis. Additionally, Saifullah et al. (2023) find that entropy outperforms BALD, indicating that the more sophisticated Bayesian uncertainty methods do not necessarily improve performance over more straightforward uncertainty methods.

**The bad performance of WAAL**
Especially the bad performance found for WAAL is surprising, given that it outperformed all other deep AL methods tested by Zhan et al. (2022). Likely, this performance is mostly due to the training step. WAAL already performed worse after the random initialization step, when no query step was taken yet. On top, the query step combines an uncertainty metric somewhat similar to the other AL methods with diversity. Given that the other AL methods and random acquisition, which acquires a diverse batch as well, all perform comparably, it is unlikely that the query itself caused such bad performance. Therefore, the discriminator sometimes not being able to predict large differences between the labelled and unlabelled datasets likely did not cause the bad performance either.

There are some differences in this WAAL application and the applications on which it was previously tested, which is possibly linked to its bad performance. In their comparison, Zhan et al. (2022) generally do not implement pre-training. Most tests are done using a model which had not been pre-trained at all, with few tests being performed with a model pre-trained on ImageNet. In none of their tests, the model had been pre-trained on a comparable dataset. The WAAL framework may mostly be useful for such situations. The model would have no knowledge on the specific data used, and hence using adversarial training could help the model to more quickly learn some features present in the data.
In this application, the Caladrius model is used, which combines two inception-v3 models pre-trained on ImageNet. The full model is thereafter pre-trained on data from different disasters. Thus, the model has previously been trained on comparable data and already has knowledge of features which are not specific to the newly occurred disaster. The model mainly needs to fine-tune the previously

learned model to a new situation, which may be a more specific task than what the adversarial training of WAAL is useful for. In this application, the adversarial training procedure could be more interfering with the fine-tuning of the model than it is useful in obtaining some general knowledge about data.
If the adversarial training indeed does mess with the fine-tuning of the feature extractor, we would have expected that smaller values of the adversarial training trade-off parameter $\mu$ result in better performance. When tuning this hyperparameter however, smaller values of $\mu$ did not always improve performance. Thus, this is unlikely to be the only issue causing the bad performance of WAAL.

Another cause could be the configuration of the training procedure using WAAL. While the hyperparameters were tuned to find a well-working variant, possibly other combinations would result in better performance. It is possible that the configuration of the other AL methods is simply chosen better compared to WAAL, leading to the better results. However, given that each of the hyperparameters for WAAL has been tuned by testing its performance with multiple settings, it is unlikely that a configuration can be found which leads to similar performance to other methods, without leading to a large computation time. Finding a configuration which leads to WAAL outperforming random acquisition significantly is even more unlikely.

As previously discussed, Saifullah et al. (2023) found similar results to this thesis when applying active learning to document classification. By comparing the different scenarios tested in their paper, possible causes of WAAL performing poorly can be identified. When a model pre-trained on ImageNet is fine-tuned with balanced data, WAAL performs reasonable. In this scenario, it outperforms both BALD and random sampling, but entropy uncertainty sampling performs slightly better. As WAAL is outperformed by entropy, while having outperformed this method in the research by Zhan et al. (2022), this could indicate that WAAL performs worse when applied to tasks where images are similar, such as classifying different documents instead of finding out the difference between different objects such as a car and an airplane. The same would hold for classifying damages on buildings.
Similarly WAAL outperforms random selection and BALD, while being outperformed by entropy, in a scenario where the same model is fine-tuned with unbalanced data. In this scenario however, its performance is closer to random selection compared to the previous scenario. Thus, unbalanced data seems to negatively affect WAAL.
When the model is pre-trained on another document classification dataset, WAAL even performs worse than all other discussed methods, including random sampling (Saifullah et al., 2023). Thus, both fine-tuning on unbalanced data, as well as using a model pre-trained on similar data seem to hurt the performance of WAAL. Likely, these issues partly caused the bad performance of WAAL in this applica-

tion as well.

**Modelling for ordinal data**

As explained, classification uncertainty may not be the right metric for uncertainty in active learning for Automated Damage Assessment given the ordinal classes. This insight may have implications on the used loss as well. At this moment, the Caladrius model aims to minimize the cross-entropy loss, which does not take into account the degree of wrongness of predictions. However, this degree of wrongness is relevant in practice. On top, this thesis shows that uncertainty in nominal classification likely is not a good metric for the informativeness of samples. This could indicate that minimizing a loss function which does not account for the order of classes, such as cross-entropy, is not the right choice either.

The model may learn more when it is designed to account for the order of data. It is therefore recommended that 510 tests the performance of the Caladrius model using such a method. This could e.g. be implemented by using loss function inspired by mean absolute error or macro-averaged absolute error, or more sophisticated methods designed for ordinal classification in neural networks. For example, Diaz and Marathe (2019) propose using Soft Ordinal vectors (SORD) to model the ordinal character of classes. Using this method, the true labels are transformed into a vector of values, with the highest value assigned to the true class and this value getting smaller based the distance to the true label, e.g. measured using absolute error. This SORD vector then resembles the softmax output for ordinal data more closely, and can be used in combination with conventional loss functions such as cross entropy (Diaz and Marathe, 2019). Predicting a wrong label further away from the true label will then be punished harder than closer wrong predictions.

When redesigning the loss function, another weighting option could be implemented as well. Currently the loss function either gives equal weight to each observation or to each class. However, the end users of predictions may be more interested in ensuring that certain classes are predicted well. End users could e.g. be focused on identifying destroyed buildings, in which case a larger weight for the destroyed class compared to other classes could be desirable.

**Recommendations for fine-tuning models**

Whereas AL methods do not outperform random acquisition, fine-tuning the pre-trained model still results in large performance improvements for each of the metrics. The large performance improvements could be partly due to transferring only American data to Haiti. In regions more comparable to those that the model has been pre-trained on, improvements may be somewhat smaller. Still, when fine-tuning a model which was pre-trained using data from more comparable regions, Wang (2021) found clear improvements as well, with the accuracy of the model increasing from 0.47 to 0.76.

It is thus recommended that 510 implements fine-tuning of their models using randomly selected images of the newly occurred disaster. Before implementing this in practice, some additional research into the fine-tuning should be done. Most importantly, 510 should conduct research on whether the fine-tuning should be performed by simply labelling the desired number of datapoints first and retraining the model using all these datapoints, or to still use an iterative approach as is used for AL. While the latter may not seem logical considering no AL type of query is used, this may still improve results. In particular, Saifullah et al. (2023) find that training the model with random sampling using the iterative AL type of training results in better performance compared to using the full dataset in training. Thus, 510 should investigate whether this is the case in the ADA application as well. Importantly, 510 should also ensure that the model does not overfit on the small amount of labelled data when fine-tuning a pre-trained model.

To enable practical labelling with volunteers, a user-friendly framework could be created for the labelling effort. This framework should present volunteers with one image of a building from before the disaster, and one from this building after the disaster. They should be able to select a label, after which this label is automatically added to the dataset. Possibly, the label could be checked by another volunteer, similarly to the labelling effort used by Gupta et al. (2019), to ensure consistent labelling. The easiest way to implement such a framework is to use readily available software, such as Label Studio (Tkachenko et al., 2022) or the Vertex AI labelling tool. The latter is already used by a similar automated damage assessment product (SKAI, 2022). Their specific implementation could be useful to set up a comparable framework.
The labelling framework can be used when labelling images internally within 510 as well. However, if the expectation is that volunteers will not be used often, simply performing the steps of selecting images to be labelled, adding them to the labelled dataset and retraining the model could also be done manually, saving the hassle of creating user-friendly labelling software.
Additionally, clear labelling instructions should be provided when using volunteers. Such instructions are included in the other ADA framework (SKAI, 2022), while the instructions used for the creation of the xBD dataset could be utilized as well.

# 8 Conclusion

Fine-tuning a pre-trained model for automated damage assessment after a natural hazard results in large improvements for all different metrics which were used. For unweighted metrics, especially the random initialization step using 100 images results in large improvements. For the macro-averaged metrics, iterations thereafter have a large contribution to the increased performance as well. This indicates that the random initialization tunes the model towards mostly predicting the majority class, while overfitting on the small dataset used. When more data is selected, the model learns to generalize better. Whilst fine-tuning clearly improves results, none of the AL methods managed to significantly outperform random selection. As using AL results in a less convenient labelling endeavor given the necessary iterative process with retraining of the model in between, it is not useful to implement any of the discussed active learning methods for ADA after a natural hazard. It is therefore recommended that 510 implements fine-tuning of their models using randomly selected data from the newly occurred disaster.

Additionally, it was discussed that models developed for classifying data with nominal class labels may not be the best choice for this application. Therefore, it is recommended to test whether incorporating methods developed for ordinal classification in deep learning could improve the performance.

# References


Abe, N. (1998). Query learning strategies using boosting and bagging. *Proceedings of the 15th International Conference on Machine Learning (ICML98)*, pages 1–9.

Al-Khudhairy, D., Caravaggi, I., and Giada, S. (2005). Structural damage assessments from ikonos data using change detection, object-oriented segmentation, and classification techniques. *Photogrammetric Engineering & Remote Sensing*, 71(7):825–837.

Albawi, S., Mohammed, T. A., and Al-Zawi, S. (2017). Understanding of a convolutional neural network. In *2017 international conference on engineering and technology (ICET)*, pages 1–6. IEEE.

Albelwi, S. and Mahmood, A. (2017). A framework for designing the architectures of deep convolutional neural networks. *Entropy*, 19(6):242.

Atighehchian, P., Branchaud-Charron, F., Freyberg, J., Pardinas, R., Schell, L., and Pearse, G. (2022). Baal, a bayesian active learning library. https://github.com/baal-org/baal/.

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

Baccianella, S., Esuli, A., and Sebastiani, F. (2009). Evaluation measures for ordinal regression. In *2009 Ninth international conference on intelligent systems design and applications*, pages 283–287. IEEE.

Balvert, M. (2021). Lecture notes operations research and machine learning: Optimization of neural networks.

Barrington, L., Ghosh, S., Greene, M., Har-Noy, S., Berger, J., Gill, S., Lin, A. Y.-M., and Huyck, C. (2011). Crowdsourcing earthquake damage assessment using remote sensing imagery. *Annals of Geophysics*, 54(6).

Basha, S. S., Dubey, S. R., Pulabaigari, V., and Mukherjee, S. (2020). Impact of fully connected layers on performance of convolutional neural networks for image classification. *Neurocomputing*, 378:112–119.

Berezina, P. and Liu, D. (2022). Hurricane damage assessment using coupled convolutional neural networks: a case study of hurricane michael. *Geomatics, Natural Hazards and Risk*, 13(1):414–431.

Bottou, L. et al. (1991). Stochastic gradient learning in neural networks. *Proceedings of Neuro-Nımes*, 91(8):12.

Bromley, J., Guyon, I., LeCun, Y., Säckinger, E., and Shah, R. (1993). Signature verification using a "siamese" time delay neural network. *Advances in neural information processing systems*, 6.

Centre for Research on the Epidemiology of Disasters [CRED] (2023). 2022 disasters in numbers.

Dell'Acqua, F. and Polli, D. A. (2011). Post-event only vhr radar satellite data for automated damage assessment. *Photogrammetric Engineering & Remote Sensing*, 77(10):1037–1043.

Diaz, R. and Marathe, A. (2019). Soft labels for ordinal regression. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4738–4747.

Ding, Y., Sohn, J. H., Kawczynski, M. G., Trivedi, H., Harnish, R., Jenkins, N. W., Lituiev, D., Copeland, T. P., Aboian, M. S., Mari Aparici, C., et al. (2019). A deep learning model to predict a diagnosis of alzheimer disease by using 18f-fdg pet of the brain. *Radiology*, 290(2):456–464.

Dubourg, V., Vanderplas, J., Metzen, J. H., and Lemaitre, G. (2022). Gaussian processes regression: basic introductory example. https://scikit-learn.org/stable/auto_examples/gaussian_process/plot_gpr_noisy_targets.html. Accessed: 2022-01-02.

Fernandez Galarreta, J., Kerle, N., and Gerke, M. (2015). Uav-based urban structural damage assessment using object-based image analysis and semantic reasoning. *Natural hazards and earth system sciences*, 15(6):1087–1101.

Flanders, D., Hall-Beyer, M., and Pereverzoff, J. (2003). Preliminary evaluation of ecognition object-based software for cut block delineation and feature extraction. *Canadian Journal of Remote Sensing*, 29(4):441–452.

Gal, Y. et al. (2016). Uncertainty in deep learning.

Gal, Y. and Ghahramani, Z. (2015). Bayesian convolutional neural networks with bernoulli approximate variational inference. *arXiv preprint arXiv:1506.02158*.

Gal, Y. and Ghahramani, Z. (2016). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR.

Gal, Y., Islam, R., and Ghahramani, Z. (2017). Deep bayesian active learning with image data. In *International Conference on Machine Learning*, pages 1183–1192. PMLR.

Ge, J., Tang, H., Yang, N., and Hu, Y. (2023). Rapid identification of damaged buildings using incremental learning with transferred data from historical natural disaster cases. *ISPRS Journal of Photogrammetry and Remote Sensing*, 195:105–128.

Gholamalinezhad, H. and Khosravi, H. (2020). Pooling methods in deep neural networks, a review. *arXiv preprint arXiv:2009.07485*.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, G., Cai, J., et al. (2018). Recent advances in convolutional neural networks. *Pattern recognition*, 77:354–377.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. C. (2017). Improved training of wasserstein gans. *Advances in neural information processing systems*, 30.

Gupta, R., Hosfelt, R., Sajeev, S., Patel, N., Goodman, B., Doshi, J., Heim, E., Choset, H., and Gaston, M. (2019). xbd: A dataset for assessing building damage from satellite imagery. *arXiv preprint arXiv:1911.09296*.

Han, W., Coutinho, E., Ruan, H., Li, H., Schuller, B., Yu, X., and Zhu, X. (2016). Semi-supervised active learning for sound classification in hybrid learning environments. *PloS one*, 11(9):e0162075.

Hao, W., Yizhou, W., Yaqin, L., and Zhili, S. (2020). The role of activation function in cnn. In *2020 2nd International Conference on Information Technology and Computer Application (ITCA)*, pages 429–432. IEEE.

Hasegawa, H., Aoki, H., Yamazaki, F., Matsuoka, M., and Sekimoto, I. (2000). Automated detection of damaged buildings using aerial hdtv images. In *Proceedings of the IEEE 2000 International Geoscience and Remote Sensing Symposium*, pages 310–312. IEEE.

Hijazi, S., Kumar, R., Rowen, C., et al. (2015). Using convolutional neural networks for image recognition.

Houlsby, N., Huszár, F., Ghahramani, Z., and Lengyel, M. (2011). Bayesian active learning for classification and preference learning. *arXiv preprint arXiv:1112.5745*.

Huyck, C. K., Adams, B. J., Cho, S., Chung, H.-C., and Eguchi, R. T. (2005). Towards rapid citywide damage mapping using neighborhood edge dissimilarities in very high-resolution optical satellite imagery—application to the 2003 bam, iran, earthquake. *Earthquake Spectra*, 21(S1):255–266.

International Committee of the Red Cross [ICRC] and International Federation of Red Cross and Red Crescent Societies [IFRC] (2008). Guidelines for assessment in emergencies.

International Federation of Red Cross and Red Crescent Societies [IFRC] (2020). World disasters report 2020.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr.

Jayaraman, V., Chandrasekhar, M., and Rao, U. (1997). Managing the natural disasters from space technology inputs. *Acta Astronautica*, 40(2-8):291–325.

Jeggle, T. and Boggero, M. (2018). Post-disaster needs assessment : Lessons from a decade of experience.

Kerle, N. and Hoffman, R. R. (2013). Collaborative damage mapping for emergency response: the role of cognitive systems engineering. *Natural hazards and earth system sciences*, 13(1):97–113.

Kerle, N., Nex, F., Gerke, M., Duarte, D., and Vetrivel, A. (2019). Uav-based structural damage mapping: A review. *ISPRS international journal of geo-information*, 9(1):14.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kirsch, A., van Amersfoort, J., and Gal, Y. (2019). Batchbald: Efficient and diverse batch acquisition for deep bayesian active learning.

Koch, G., Zemel, R., Salakhutdinov, R., et al. (2015). Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, volume 2, page 0. Lille.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90.

Laframboise, M. N. and Loko, M. B. (2012). Natural disasters: mitigating impact, managing risks.

Lallemant, D., Soden, R., Rubinyi, S., Loos, S., Barns, K., and Bhattacharjee, G. (2017). Post-disaster damage assessments as catalysts for recovery: A look at assessments conducted in the wake of the 2015 gorkha, nepal, earthquake. *Earthquake Spectra*, 33(1_suppl):435–451.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436–444.

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

Li, P., Xu, H., and Guo, J. (2010). Urban building damage detection from very high resolution imagery using ocsvm and spatial features. *International Journal of Remote Sensing*, 31(13):3393–3409.

Liquet, B., Moka, S., and Nazarathy, Y. (2023). The mathematical engineering of deep learning.

Liu, B., Zou, D., Feng, L., Feng, S., Fu, P., and Li, J. (2019). An fpga-based cnn accelerator integrating depthwise separable convolution. *Electronics*, 8(3):281.

Liu, Y., Starzyk, J. A., and Zhu, Z. (2008). Optimized approximation algorithm in neural networks without overfitting. *IEEE transactions on neural networks*, 19(6):983–995.

Loshchilov, I. and Hutter, F. (2016). Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*.

Ma, H., Liu, Y., Ren, Y., Wang, D., Yu, L., and Yu, J. (2020). Improved cnn classification method for groups of buildings damaged by earthquake, based on high resolution remote sensing images. *Remote Sensing*, 12(2):260.

Marmanis, D., Datcu, M., Esch, T., and Stilla, U. (2015). Deep learning earth observation classification using imagenet pretrained networks. *IEEE Geoscience and Remote Sensing Letters*, 13(1):105–109.

Mehrjou, A., Khodabandeh, M., and Mori, G. (2018). Distribution aware active learning. *arXiv preprint arXiv:1805.08916*.

Mishkin, D., Sergievskiy, N., and Matas, J. (2017). Systematic evaluation of convolution neural network advances on the imagenet. *Computer vision and image understanding*, 161:11–19.

Mitomi, H., Yamazaki, F., and Matsuoka, M. (2001). Development of automated extraction method for building damage area based on maximum likelihood classifier. In *Proceedings of the 8th International Conference on Structural Safety and Reliability*, page 8.

Naeim, F., Hagie, S., Alimoradi, A., and Miranda, E. (2006). Automated post-earthquake damage assessment of instrumented buildings. In *Advances in earthquake engineering for urban risk reduction*, pages 117–134. Springer.

Nex, F., Duarte, D., Steenbeek, A., and Kerle, N. (2019). Towards real-time building damage mapping with low-cost uav solutions. *Remote sensing*, 11(3):287.

Nielsen, M. A. (2015). *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA, USA.

Norouzzadeh, M. S., Morris, D., Beery, S., Joshi, N., Jojic, N., and Clune, J. (2021). A deep active learning system for species identification and counting in camera trap images. *Methods in ecology and evolution*, 12(1):150–161.

Nwankpa, C., Ijomah, W., Gachagan, A., and Marshall, S. (2018). Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*.

O'Shea, K. and Nash, R. (2015). An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*.

Ozisik, D. and Kerle, N. (2004). Post-earthquake damage assessment using satellite and airborne data in the case of the 1999 kocaeli earthquake, turkey. In *Proc. of the XXth ISPRS congress: Geo-imagery bridging continents*, pages 686–691.

Pan, S. J. and Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

Saifullah, S., Agne, S., Dengel, A., and Ahmed, S. (2023). Analyzing the potential of active learning for document image classification. *International Journal on Document Analysis and Recognition (IJDAR)*, pages 1–23.

Sener, O. and Savarese, S. (2017). Active learning for convolutional neural networks: A core-set approach. *arXiv preprint arXiv:1708.00489*.

Settles, B. (2009). Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison.

Settles, B., Craven, M., and Ray, S. (2007). Multiple-instance active learning. *Advances in neural information processing systems*, 20.

Seung, H. S., Opper, M., and Sompolinsky, H. (1992). Query by committee. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, page 287–294, New York, NY, USA. Association for Computing Machinery.

Shao, J., Tang, L., Liu, M., Shao, G., Sun, L., and Qiu, Q. (2020). Bdd-net: A general protocol for mapping buildings damaged by a wide range of disasters based on satellite imagery. *Remote Sensing*, 12(10):1670.

Shui, C., Zhou, F., Gagné, C., and Wang, B. (2020). Deep active learning: Unified and principled method for query and training. In *International Conference on Artificial Intelligence and Statistics*, pages 1308–1318. PMLR.

SKAI (2022). SKAI damage assessment instructions. https://github.com/google-research/skai/blob/main/docs/assessment_instructions.md.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

Sun, Y., Zhang, J., Meng, Y., Yang, J., and Gui, G. (2019). Smart phone-based intelligent invoice classification method using deep learning. *IEEE access*, 7:118046–118054.

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

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826.

Taha, A. A. and Hanbury, A. (2015). Metrics for evaluating 3d medical image segmentation: analysis, selection, and tool. *BMC medical imaging*, 15(1):1–28.

Tiede, D., Lang, S., Füreder, P., Hölbling, D., Hoffmann, C., and Zeil, P. (2011). Automated damage indication for rapid geospatial reporting. *Photogrammetric Engineering & Remote Sensing*, 77(9):933–942.

Tkachenko, M., Malyuk, M., Holmanyuk, A., and Liubimov, N. (2022). Label Studio: Data labeling software. https://github.com/heartexlabs/label-studio.

Tsymbalov, E., Panov, M., and Shapeev, A. (2018). Dropout-based active learning for regression. In *Analysis of Images, Social Networks and Texts: 7th International Conference, AIST 2018, Moscow, Russia, July 5–7, 2018, Revised Selected Papers 7*, pages 247–258. Springer.

Valentijn, T., Margutti, J., van den Homberg, M., and Laaksonen, J. (2020). Multi-hazard and spatial transferability of a cnn for automated building damage assessment. *Remote Sensing*, 12(17):2839.

Van den Bogaart, S. (2021). The use of active learning in automated damage assessment. Master's thesis, Maastricht University.

Van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(11).

Van Westen, C. (2000). Remote sensing for natural disaster management. *International archives of photogrammetry and remote sensing*, 33(B7/4; PART 7):1609–1617.

Vetrivel, A., Gerke, M., Kerle, N., Nex, F., and Vosselman, G. (2018). Disaster damage detection through synergistic use of deep learning and 3d point cloud features derived from very high resolution oblique aerial images, and multiple-kernel-learning. *ISPRS journal of photogrammetry and remote sensing*, 140:45–59.

Vetrivel, A., Gerke, M., Kerle, N., and Vosselman, G. (2015). Identification of damage in buildings based on gaps in 3d point clouds from very high resolution oblique airborne images. *ISPRS journal of photogrammetry and remote sensing*, 105:61–78.

Vetrivel, A., Kerle, N., Gerke, M., Nex, F., and Vosselman, G. (2016). Towards automated satellite image segmentation and classification for assessing disaster damage using data-specific features with incremental learning.

Wang, M. (2021). Active learning for improved damage detection and disaster response. Bachelor's thesis, Harvard College.

Westrope, C., Banick, R., and Levine, M. (2014). Groundtruthing openstreetmap building damage assessment. *Procedia engineering*, 78:29–39.

Xu, J. Z., Lu, W., Li, Z., Khaitan, P., and Zaytseva, V. (2019). Building damage detection in satellite imagery using convolutional neural networks. *arXiv preprint arXiv:1910.06444*.

Yamashita, R., Nishio, M., Do, R. K. G., and Togashi, K. (2018). Convolutional neural networks: an overview and application in radiology. *Insights into imaging*, 9(4):611–629.

Yamazaki, F. (2001). Applications of remote sensing and gis for damage assessment. *Structural Safety and Reliability*, 1:12.

Yang, W., Zhang, X., and Luo, P. (2021). Transferability of convolutional neural network models for identifying damaged buildings due to earthquake. *Remote Sensing*, 13(3):504.

Zafar, A., Aamir, M., Mohd Nawi, N., Arshad, A., Riaz, S., Alruban, A., Dutta, A. K., and Almotairi, S. (2022). A comparison of pooling methods for convolutional neural networks. *Applied Sciences*, 12(17):8643.

Zhan, X., Wang, Q., Huang, K.-h., Xiong, H., Dou, D., and Chan, A. B. (2022). A comparative survey of deep active learning. *arXiv preprint arXiv:2203.13450*.

Zhang, A., Lipton, Z. C., Li, M., and Smola, A. J. (2021). Dive into deep learning. *arXiv preprint arXiv:2106.11342*.

# Appendices

## A List of Acronyms

Below, abbreviations used throughout the thesis are listed. Abbreviations which are only used within one section are not listed.

**ADA** Automated Damage Assessment

**AL** Active Learning

**BALD** Bayesian Active Learning by Disagreement

**BCNN** Bayesian Convoltional Neural Network

**BNN** Bayesian Neural Network

**CNN** Convolutional Neural Network

**MAE** Mean Absolute Error

**MAAE** Macro-Averaged Absolute Error

**MC** Monte Carlo

**ReLU** Rectified Linear Unit

**SGD** Stochastic Gradient Descent

**WAAL** Wasserstein Adversarial Active Learning

# B Appendix to Section 4: Methodology

## B.1 Comparison different AL methods

Table B.1: Ranking of different AL algorithms for multiple datasets based on its accuracy on different acquisition sizes. A win is defined as outperforming another model by at least 0.5 percent, a loss as being outperformed by at least 0.5 percent and a tie in between these. Source: Zhan et al. (2022).

| | *Standard Image Classification (8 datasets)* | | *Medical Image Analysis (2 datasets)* | |
|---|---|---|---|---|
| Rank | Method | win − tie − loss | Method | win − tie − loss |
| 1 | **WAAL** | 103 − 2 − 31 | **WAAL** | 34 − 0 − 0 |
| 2 | **CEAL** | 74 − 35 − 27 | **LPL** | 25 − 6 − 3 |
| 3 | **LeastConfD** | 63 − 59 − 14 | **VarRatio** | 23 − 7 − 4 |
| 4 | **MarginD** | 61 − 55 − 20 | **BADGE** | 24 − 1 − 9 |
| 5 | **Margin** | 60 − 57 − 19 | **Margin** | 19 − 10 − 5 |
| 6 | **BALD** | 56 − 59 − 21 | **Entropy** | 18 − 11 − 5 |
| 7 | **EntropyD** | 54 − 55 − 27 | **LeastConf** | 18 − 9 − 7 |
| 8 | **VarRatio** | 52 − 58 − 26 | **VAAL** | 16 − 9 − 12 |
| 9 | **LeastConf** | 51 − 53 − 32 | **CEAL** | 15 − 7 − 12 |
| 10 | **Badge** | 46 − 49 − 41 | **AdvBIM** | 13 − 11 − 10 |
| 11 | **MeanSTD** | 44 − 50 − 42 | **MarginD** | 12 − 9 − 13 |
| 12 | **Entropy** | 40 − 54 − 42 | **KMeans** | 10 − 9 − 15 |
| 13 | **LPL** | 57 − 4 − 75 | **BALD** | 7 − 8 − 19 |
| 14 | **KCenter** | 41 − 34 − 61 | **LeastConfD** | 7 − 6 − 21 |
| 15 | **Random** | 26 − 23 − 87 | **Random** | 4 − 7 − 23 |
| 16 | **VAAL** | 20 − 15 − 101 | **EntropyD** | 2 − 3 − 29 |
| 17 | **KMeans** | 18 − 9 − 109 | **KCenter** | 2 − 3 − 29 |
| 18 | **AdvBIM** | 10 − 25 − 101 | **MeanSTD** | 0 − 1 − 33 |

## B.2 Performance metrics and their interpretation

### Accuracy

In classification, accuracy simply divides the number of correct predictions by the total number of datapoints. Thus, the accuracy represents the fraction of correctly classified datapoints.

### Macro F1-score

To explain the macro F1-score, first precision and recall must be introduced. For the notation, some additional terms are introduced. For a given class, a prediction is true positive when correctly predicted to be in this class, false positive when incorrectly predicted to be in this class, true negative when correctly predicted not to belong to this class and false negative when incorrectly predicted not to belong to this class. Let the number of true positives, false positives, true negatives and false negatives be denoted with $TP$, $FP$, $TN$ and $FN$ respectively.

The precision for a given class is the fraction of the datapoints predicted to be in

the class that actually belong to that class. Using the $TP/FP/TN/FN$ notation, we get:

$$precision = \frac{TP}{TP + FP} \tag{34}$$

The recall is the fraction of datapoints that belong to the given class which are actually predicted to be in that class:

$$recall = \frac{TP}{TP + FN} \tag{35}$$

The F1-score is the harmonic mean of the precision and recall, hence a high F1-score is obtained when a model performs well in both precision and recall. The F1-score is calculated as follows (Taha and Hanbury, 2015):

$$F1 = \frac{2 * precision * recall}{precision + recall} \tag{36}$$

For each class, an F1-score is calculated separately. To obtain a score representing the full model, an average over these scores for the classes must be taken. There are different methods by which this average can be calculated, e.g. by weighting the score by the number of datapoints belonging to this class respectively. For this application however, macro averaging is used. This means that each class is given equal weight, so it simply takes the average of all computed F1-scores. Compared to accuracy, the macro F1-score therefore represents all classes fairly. When a model would simply predict all datapoints in the majority class, this would result in a small macro F1-score, while the accuracy could still be high.

**Mean Absolute Error**

To take into account the degree of wrongness, the Mean Absolute Error (MAE) is used. MAE is often used as a performance metric for regression. It does not focus on whether or not a prediction is exactly correct, but on the distance between the prediction and the true value, as is usually important for regression. As this research deals with ordinal classes, MAE can provide valuable insights in this application. As given by the name, it simply computes the absolute error for each observation and averages over this:

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{37}$$

Where $y_i$ is the true class of datapoint $i$ and $\hat{y}_i$ is the prediction of the model. This prediction is returned as an integer of 1-4, representing no damage, minor damage, major damage and detroyed repsectively. Thus when a building with minor damage destroyed is predicted to be destroyed, $|y_i - \hat{y}_i| = |2 - 4| = 2$.

**Macro Averaged Absolute Error**

For the MAAE, the frequency of each class in the true labels is calculated first.

Each datapoint is then assigned a weight $w_i = \frac{1}{f_i * C}$, where $f_i$ is the frequency of the true label of datapoint $i$ in the full dataset evaluated on and $C$ is the total number of classes. Thus, when a datapoint is part of a class which occurs more often in the dataset, it is given a lower weight in the score calculation. Using these weights, the MAAE is defined as follows:

$$MAAE = \sum_{i=1}^{n} w_i |y_i - \hat{y}_i| \tag{38}$$

Important to note is that weights $w_i$ sum up to one. Hence, the MAAE still represents an average absolute error, but with different weighting. Whereas the regular MAE formulation gives equal weight to each observation, MAAE gives equal weight to each class. While calculated slightly different, this metric is similar to the macro-averaged mean absolute error introduced by Baccianella et al. (2009).

### B.3 Confusion Matrices

To explain how confusion matrices work, an example is given in Figure B.1. This Figure contains the confusion matrix for validation data using a model created by retraining the pre-trained model using the full training data from hurricane Matthew.

Here, on the x-axis the predicted and on the y-axis the true labels are given. In the horizontal rows, the number of datapoints with the given true label which are predicted into each class are given, with the percentage of datapoints predicted in these classes given between brackets. The diagonal thus contains correctly classified datapoints, and the further away from the diagonal datapoints are predicted, the worse the prediction is. A good model would hence result in a large proportion of datapoints close to the diagonal and only few datapoints being predicted into categories far away from the diagonal.

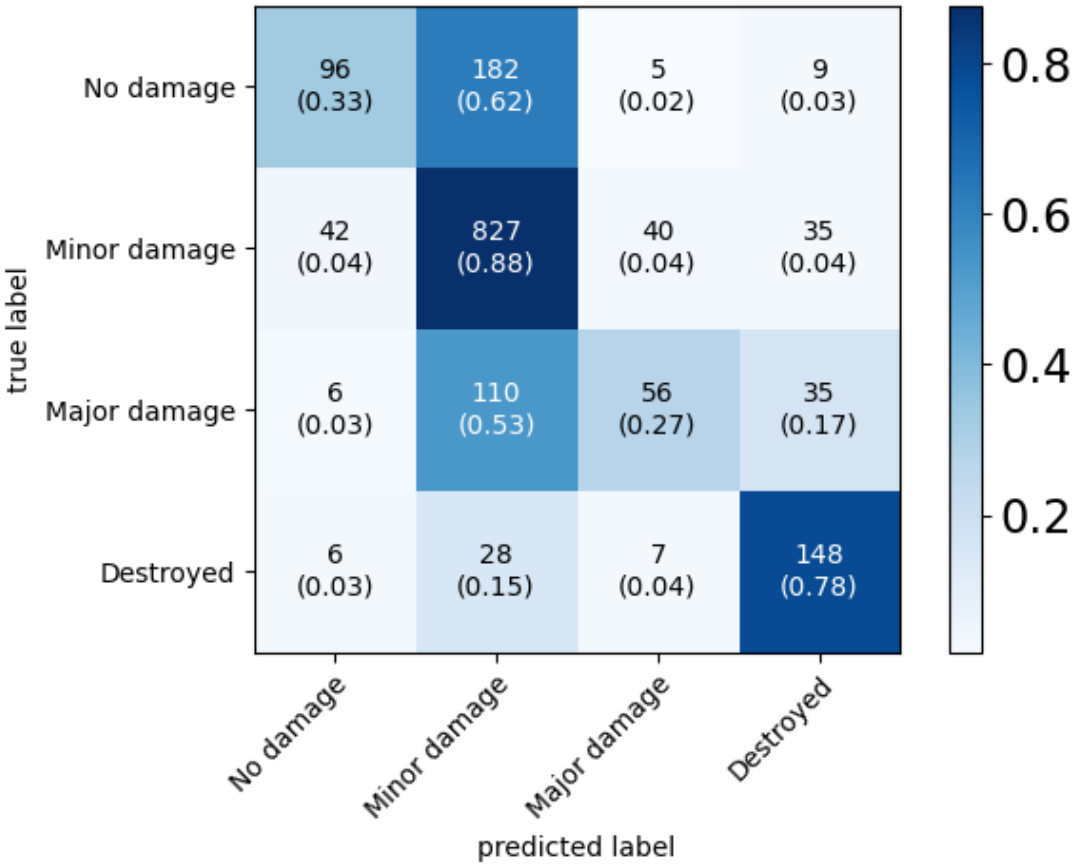



Figure B.1: Example of a confusion matrix, made using predictions on validation data with a model trained on the full dataset of hurricane Matthew.

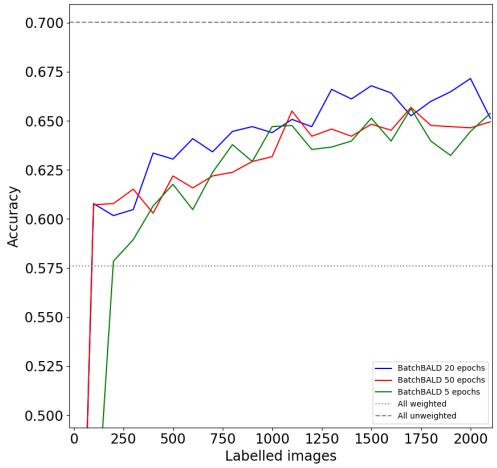
# C Appendix to Section 5: Results configuration of hyperparameters

This Appendix contains all figures and tables used to configure hyperparameters as introduced in Section 5. On top, a detailed explanation on each decision is given.
First, the results for general hyperparameters will be explained, after which they are explained for hyperparameters that are only relevant for certain methods. In all tests, the same seed is used such that the initial random acquisition is the same over all compared runs. All plots are zoomed in on results after the random initial acquisition, as the random acquisitition is the same over all models and hence not informative.

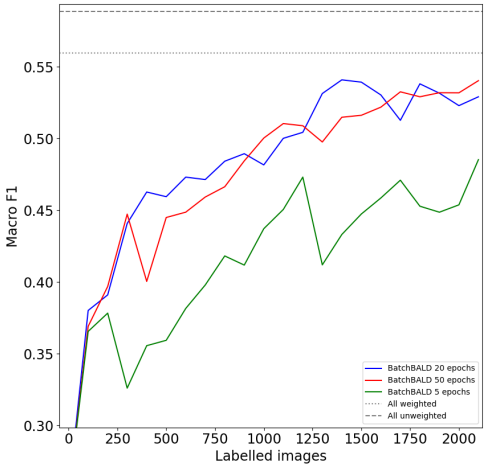
## C.1 Configuration of general hyperparameters

**Epochs per active iteration**
In Figure C.1 the results for different settings of epochs per training iteration are provided. When macro-averaged metrics are used, using only 5 epochs in each active iteration clearly results in worse performance. Hence the models seems to mostly struggle with generalizing the model for all classes when only using few epochs. The performance of using 20 and 50 epochs is comparable. For an acquisition batch size of 100, 20 epochs are used when comparing the different AL methods as using more epochs is time consuming and does not seem to improve performance[32].
For other acquisition batch sizes, which result in a different number of active iterations, the number of epochs in each iteration are rescaled accordingly. E.g. when an acquisition batch size of 500 is used, the number of epochs is set to 100. This way, the total number of epochs used is the same.

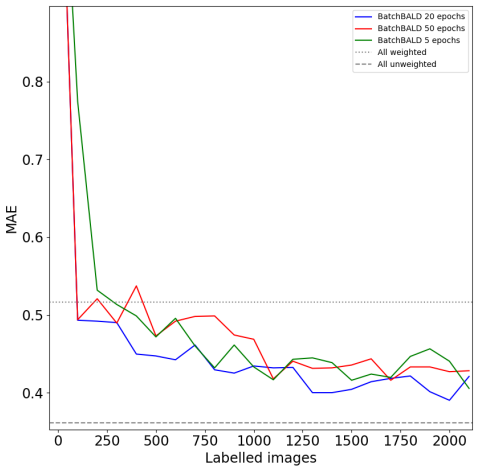
[32]This differs from the number of epochs used by Van den Bogaart (2021) and Valentijn et al. (2020) to train Caladrius, who used 50 and 100 epochs respectively. A main difference is that the model is trained iteratively in different active iterations, whereas both Van den Bogaart (2021) and Valentijn et al. (2020) only train the model once on a dataset. Therefore, their methods need more epochs in a training step for to achieve good performance given the training step is only executed once.



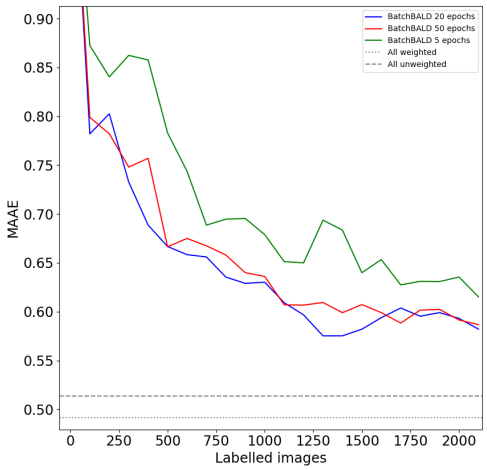

(a) Accuracy




(b) Macro-F1




(c) MAE

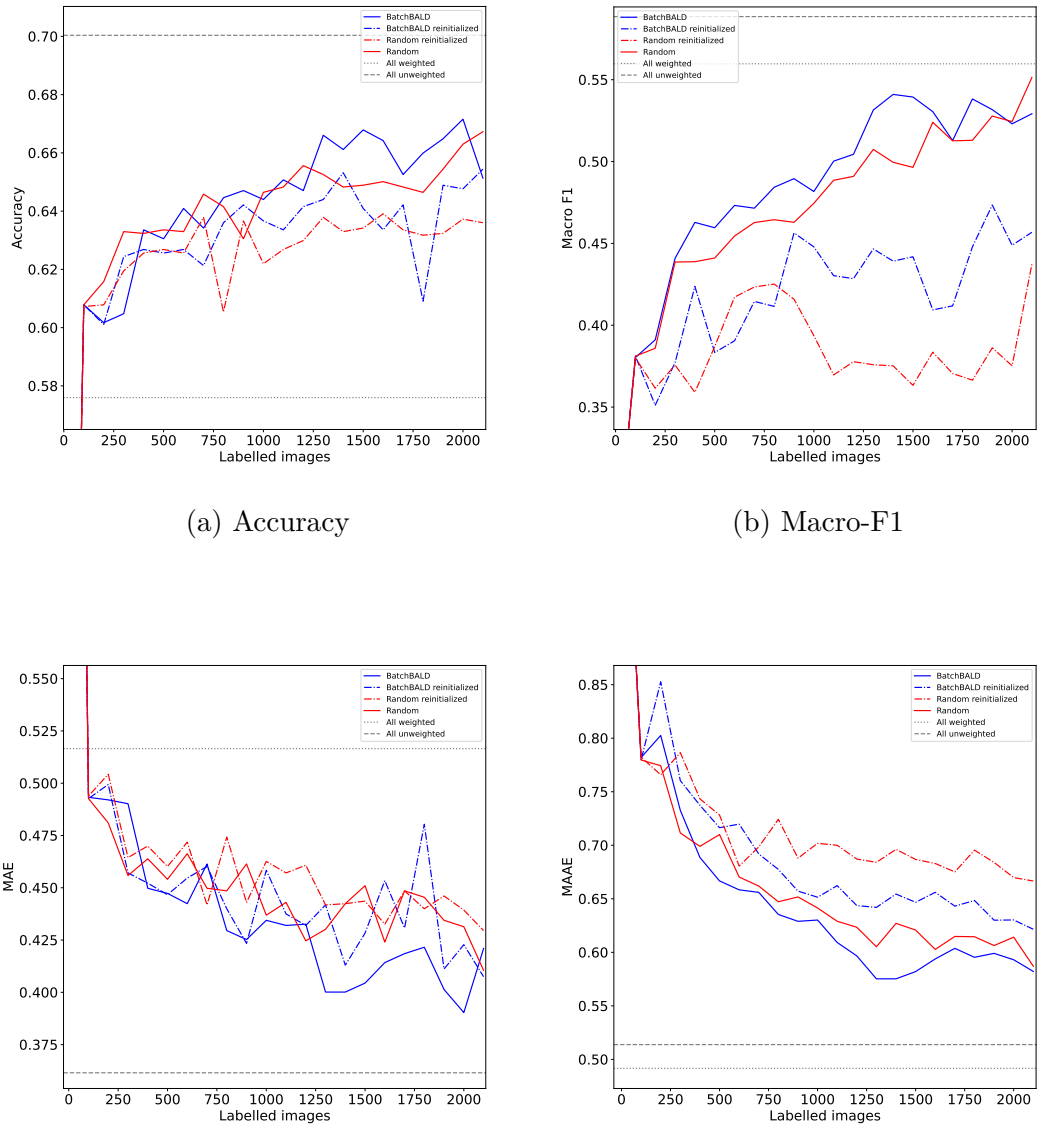



(d) MAAE

Figure C.1: Results of using different number of epochs: 5, 20 and 50.

**Reinitialization of model**

Next reinitializing the model after each active iteration is compared to not reinitializing the model in Figure C.2. The results mainly differ when using macro averaged metrics. For both random and BatchBALD, not using reinitialization results in a better macro F1 and MAAE score. Therefore, no reinitialization will be used in the remainder of this research.




(a) Accuracy

(b) Macro-F1

(c) MAE

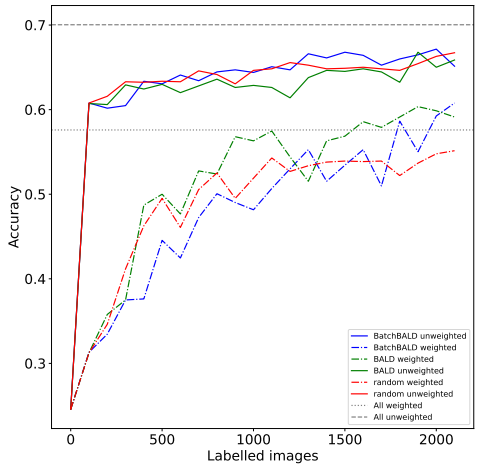
(d) MAAE

Figure C.2: Comparison using reinitialization (dot-dashed lines) with no reinitialization (solid lines) for Random and BatchBALD acquisition.

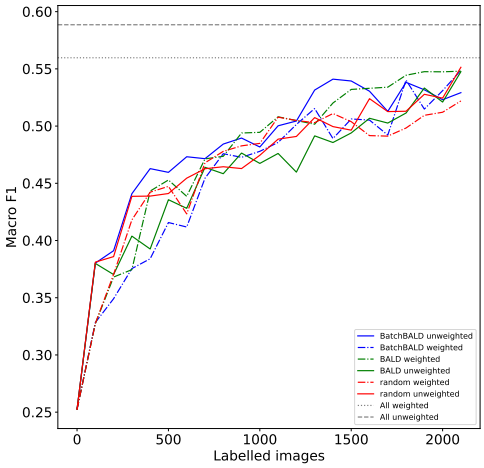
**Weighted loss function**

In Figure C.3, the comparison of using a weighted and unweighted loss function are provided. When accuracy and MAE are used, models trained with unweighted loss clearly show better performance. This is as may be expected, since these metrics do not take into account the number of images in each class. When the macro-averaged metrics are used, performances of both loss types are comparable. Hence, weighted loss does seem to outperform unweighted loss for minority classes,

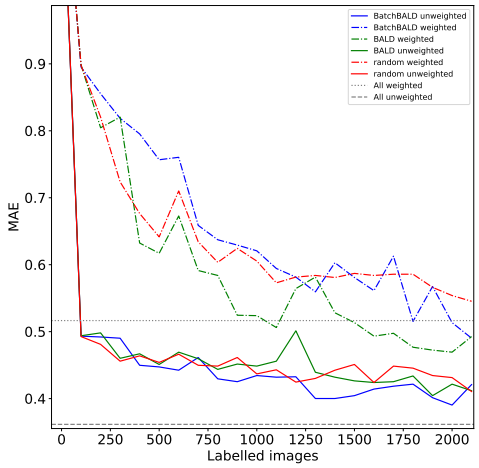
but not by much given that the better performance of unweighted loss in majority classes compensates for this. Additionally, in Section 4.4.1 it was explained that AL aims to select the most useful datapoints, in which case unweighted loss seems appropriate. Combining this with the comparable macro-averaged results and better unweighted results, the unweighted loss function will be used in the remainder of this research. However, in case good performance in the worst class is preferred, weighted loss could be useful as it performs better in minority classes.



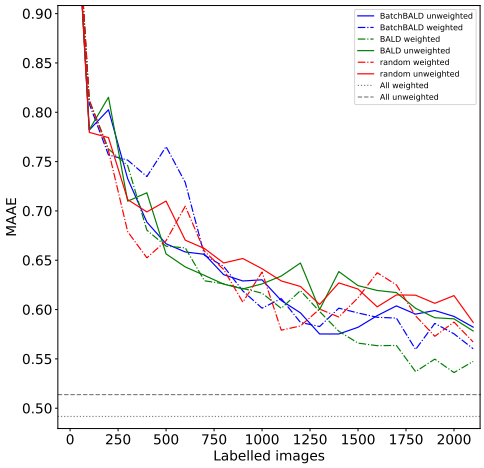

(a) Accuracy




(b) Macro-F1




(c) MAE




(d) MAAE

Figure C.3: Comparison using unweighted loss function (solid lines) with weighted loss function (dot-dashed lines) for different AL methods.

**Acquisition batch size**
Furthermore, the influence of the acquisition batch size on performance is assessed. The results are provided in Figure C.4. The performance of acquiring batches of 50 (dotted lines) and 100 (solid lines) images per iteration are comparable using all metrics. With a small difference, acquiring 50 images in each iteration seems to perform better. However, acquiring the images in twice as large batches makes the training procedure more convenient in practice, while also decreasing training time. Therefore, acquiring 100 images each iteration is preferred.
When 500 images are acquired, two options are implemented: using warm restart (dashed lines) or not (dash-dotted lines). When no warm restart is used, the model only performs well using unweighted metrics. This could be explained by the model quickly moving towards a relatively good solution, which in this case would be predicting most images into the majority class. This solution is next finetuned to a local optimum with a small learning rate. With warm restart used, acquisition batches of 500 images yield good results for macro averaged metrics as well. Hence, using warm restart seems to help the model generalize over all classes, instead of focussing on majority classes.

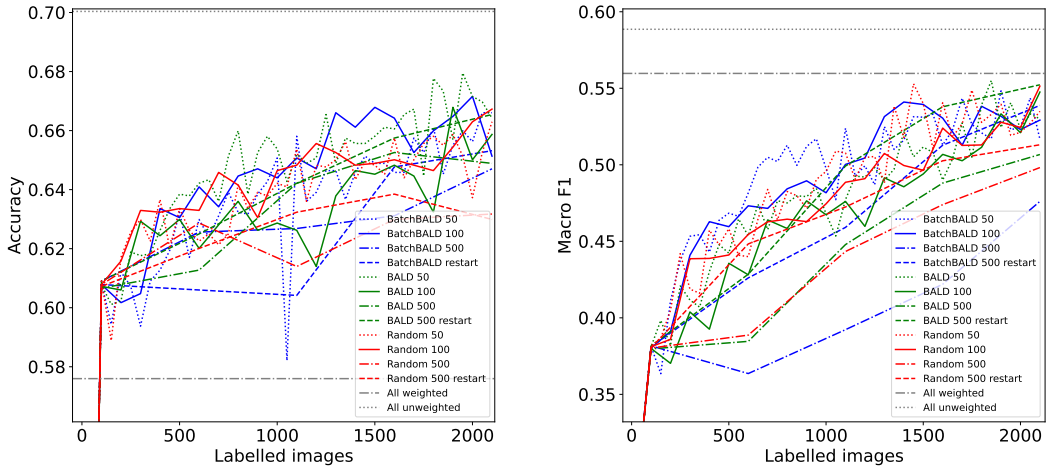
Given these results and the practical considerations explained below, two scenarios will be tested in more detail:

- Acquiring 500 images using an acquisition batch size of 100
- Acquiring 2000 images using an acquisition batch size of 500 with warm restart

These scenarios simulate different types of settings in which AL could be implemented by 510. The first scenario simulates a relatively small scale disaster, where resources are likely limited. In such case, data would have to be labelled internally by employees of 510. Consequently, the total number of images that can be labelled is small. Therefore, only 500 images are labelled in this scenario. An acquisition batch size of 100 is used. This way, models which are trained on data from previous AL iterations are used in the acquisition of new data. When only one iteration of 500 images was used, all data would be acquired by only using the model trained on the random initialized data. This could make the final results more dependent on this initialization.

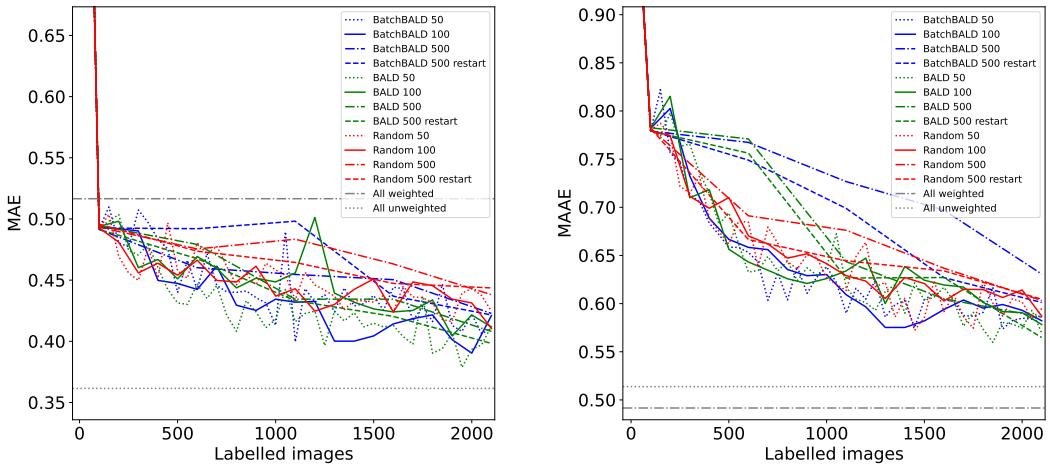
The second scenario simulates a larger disaster which captures a lot of media attention. In this case, 510 is more likely to have a group of volunteers available to label data. In this situation, it is possible to label a larger total amount of images. A dataset of 2000 images will thus be labelled in this scenario. When using groups of volunteers, labelling larger batches of data each time is more convenient. When an acquisition batch size of 100 is used, 510 has to ask volunteers to label a small amount of images 20 times, while having to wait on the volunteers

to be finished labelling before retraining and selecting a new to-be-labelled batch in each of these 20 iterations. Increasing the acquisition batch size could simplify this procedure substantially. When using an acquisition batch size of 500, volunteers would only have to be asked for help in 4 different iterations with retraining in between. Additionally, labelling 500 images could be done in groups where one expert helps volunteers with images they are unsure of. Therefore, an acquisition batch size of 500 with warm restart will be used to simulate this scenario.




(a) Accuracy

(b) Macro-F1




(c) MAE

(d) MAAE

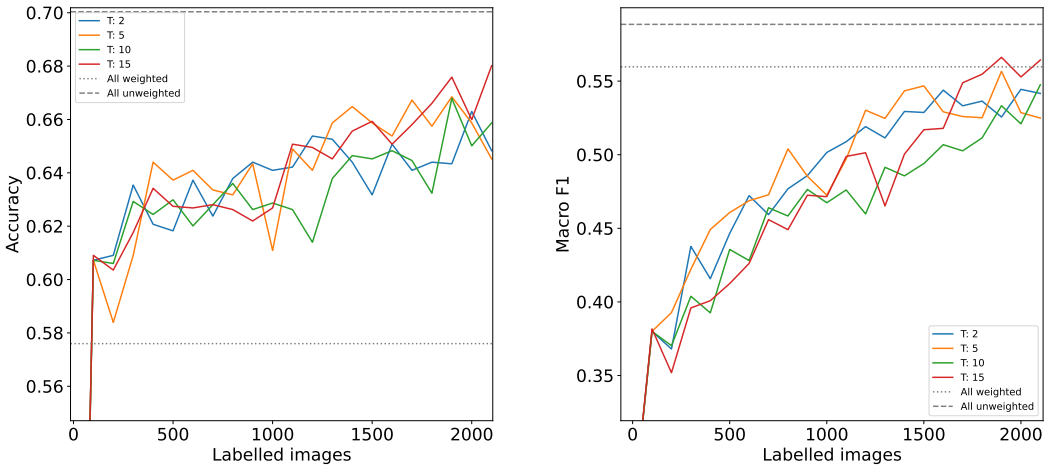
Figure C.4: Comparison using different acquisition sizes for different AL methods: 50 (dotted), 100 (solid), 500 with warm restart (dashed) and 500 without warm restart (dash-dotted).

## C.2 Configuration of hyperparameters for Bayesian AL methods

### BALD

As explained in Section 4.4.2, the number of MC dropout models ($T$) must be set when using BALD. Therefore, BALD is run with $T$ equal to 2, 5, 10 and 15. The results are displayed in Figure C.5. While in theory a higher value for $T$ is linked to a better estimation of uncertainty and hence should result in better

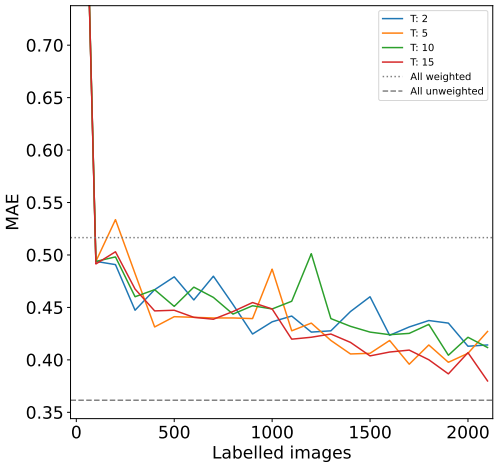
performance, this is not clear from these results. There is no clear link between a higher $T$ and better performance, with all scores being close to each other. This could indicate that $T$ does not influence results for BALD, but it could also be due to randomness, since simple luck plays a part in the performance as well. Thus no conclusive answer can be given from the plots. Therefore the recommended value by Gal and Ghahramani (2016) ($T = 10$) is used to compare BALD with other methods.



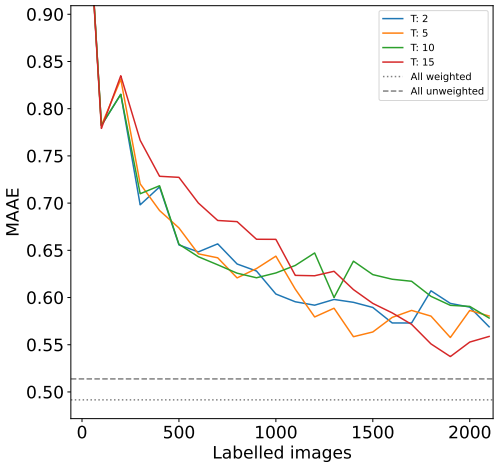

(a) Accuracy




(b) Macro-F1




(c) MAE




(d) MAAE

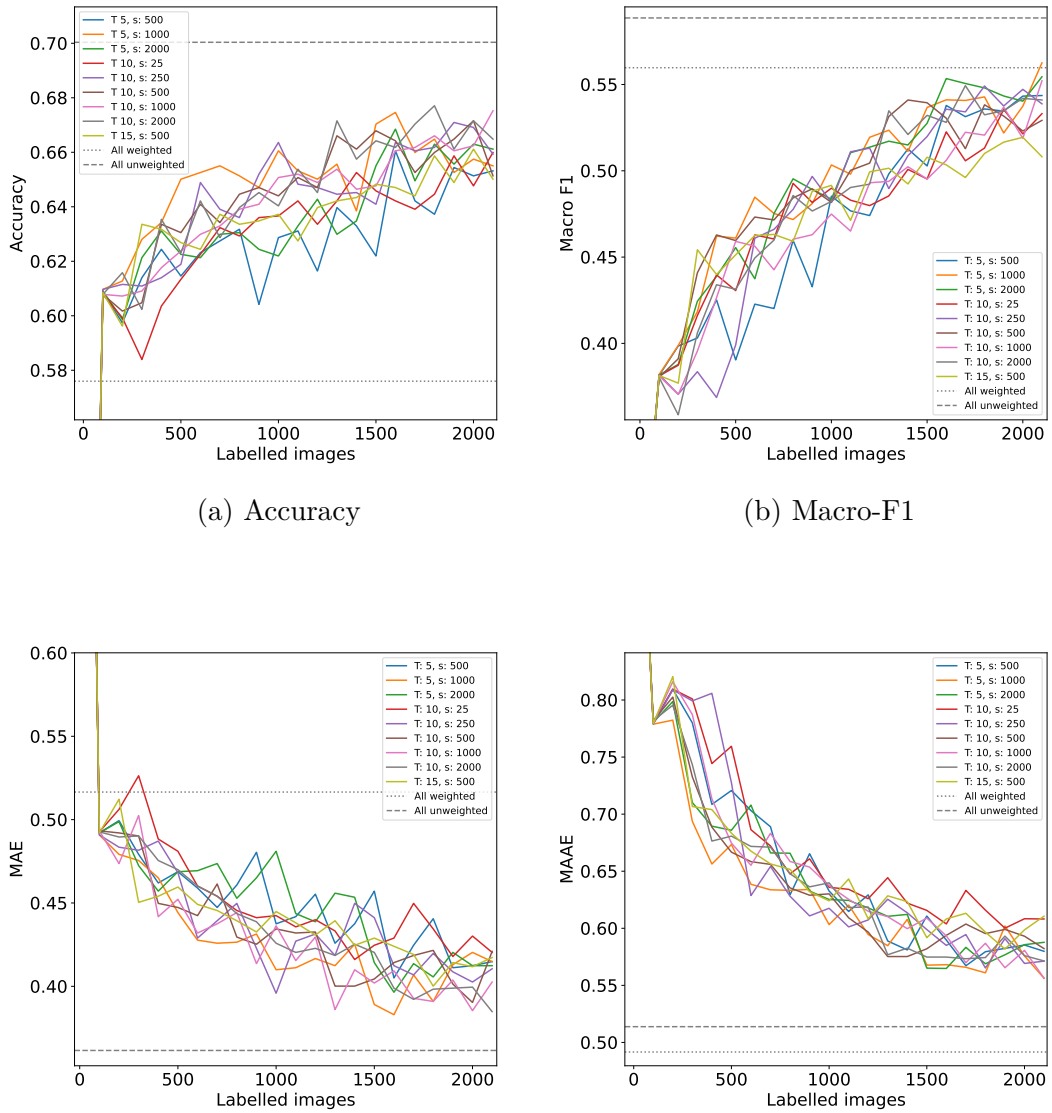
Figure C.5: Comparison of using different numbers of mc dropout models (2, 5, 10 and 15) for BALD.

**BatchBALD**

For BatchBALD different combinations of $T$ and $s$ are tried, as explained in Section 4.4.2. It would be expected that both an increased $T$ and $s$ would lead to better performance. In Figure C.6, the results are shown. The extremely small value $s = 25$, which is plotted in red and used in combination with $T = 10$ to assess the influence of an extremely small $s$, shows generally worse performance than most other combinations. Still, it is comparable with some other combinations which would not be expected. Similarly, the combination of a small $T = 5$ with $s = 500$ plotted in blue performs relatively bad which could be expected given the small $T$. On the other hand, when fixing $T = 5$, $s = 1000$ plotted in orange generally performs better than $s = 2000$ plotted in dark green, which is the reverse of what would be expected. Similarly when fixing $T = 10$, $s = 500$ plotted in brown performs similar to $s = 2000$ in grey. When fixing $s = 500$, $T = 10$ plotted in brown also outperforms $T = 15$ plotted in light green. Both these findings for $T = 10$ with $s = 500$ are the reverse of what would be expected, as a higher $T$ and $s$ should both result in better estimation of the joint mutual information.

Two possible explanations are possible. Firstly it could be the case that the parameters set do not have a large influence on performance, at least when not set extremely small. Another explanation is that the randomness in AL has such a large influence that no clear relationship between these settings and performance can be found using a few runs. As no clear conclusion can be made given the results, other considerations are made similarly to BALD. Firstly, $T$ is fixed to 10 as this makes BatchBALD more comparable with BALD, since $T$ is the same. Therefore it would be clear that differences in performances are due to the differences in query strategies instead of $T$. $s$ is chosen not too small, such that BatchBALD estimations can be made more similar to the paper by Kirsch et al. (2019). This way, results will more clearly reflect whether BatchBALD works well in this application, instead of having the possibility that too small values of $s$ negatively influenced the query strategy too much. On the other hand, $s$ is not chosen too big given the increasing computation time. Given these considerations, as well as its reasonable results in the tests combined with $T = 10$, $s$ is set to 500. Querying 100 images using this strategy takes approximately 20 minutes. With an average training time slightly above 10 minutes per iteration, this means a full iteration takes a bit longer than 30 minutes, which is the waiting time target set with 510[33].

[33]While a waiting time of around 30 minutes is desirable, it is not a necessity for an algorithm to be useful to 510. If longer waiting times would yield significantly better performance, this could still be useful in practice.




(a) Accuracy

(b) Macro-F1

(c) MAE

(d) MAAE

Figure C.6: Comparison of using different values of $T$ and $s$ for BatchBALD.

## C.3 Configuration hyperparameters WAAL

Next, the results of the configuration of WAAL hyperparameters is discussed, as well as general hyperparameters which are altered due to the different type of training used. While previously 20 active iterations of 100 images were used for hyperparameter tuning, the settings discussed in Section C.1 are used here. Therefore, hyperparameters are tuned for two situations: five active iterations of acquiring 100 datapoints and 4 active iterations of selecting 500 datapoints. Only when it could be expected that a desirable hyperparameter setting depends on

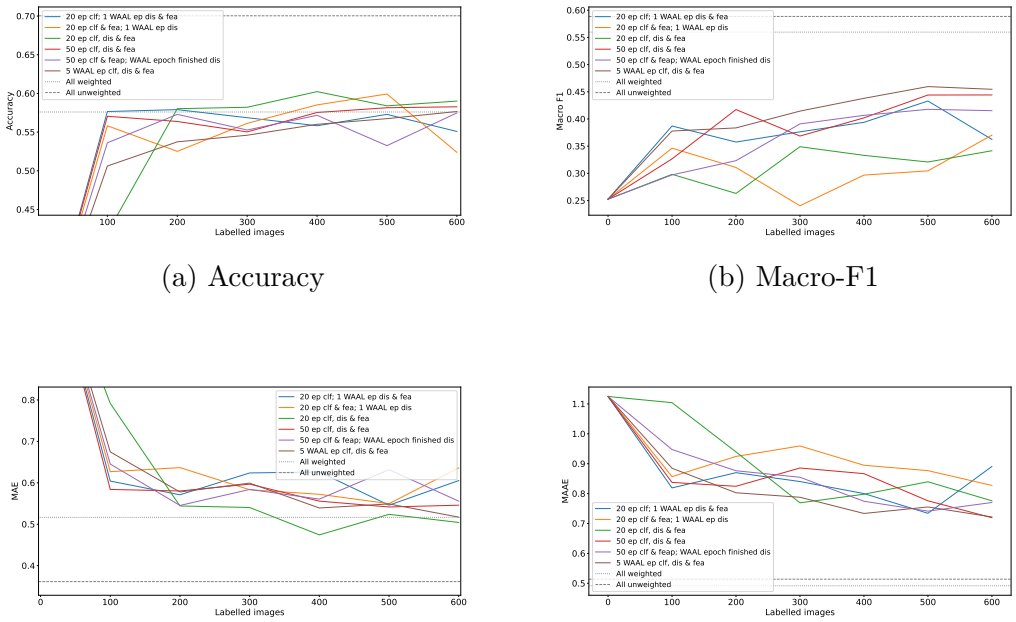
this situation, both are tested. Else, only the situation with an acquisition batch size of 100 is tested to reduce computation times.

**Epochs**
Firstly, the type of epochs are discussed. In Section 4.4.3, five different options were discussed. These options are tested separately for the two scenarios chosen for other AL methods: using an acquisition batch size of 100 and 500. When an acquisition batch size of 100 is used, the training type epochs are set to 20. Option 4 as discussed in Section 4.4.3 uses 50 epochs. For an acquisition batch size of 500, the training type epochs are set to 100, similarly to the selected number for other AL methods. Given that 100 training type epochs for WAAL is time consuming, the fourth option discussed in Section 4.4.3 is not tested.

In Figure C.7 the results are displayed for an acquisition batch size of 100. Using the unweighted metrics, using 20 training set epochs for each network performs the best, while being closely followed by using 50 train set epochs for each of the networks and using 20 train set epochs for the classifier and feature extractor while using a full WAAL epoch for the discriminator. When macro-averaged metrics are used, using 50 train set epochs and using five WAAL epochs clearly outperform the other methods. The smaller numbers of epochs seem to find a good unweighted performance, while using more epochs results in the networks predicting in classes with less labelled data better. Thus, smaller numbers of epochs seem sufficient for the model to tune the model to a new class distribution, while overfitting on majority classes. More epochs result in better performance in other classes as well.

Given that using 50 train set epochs works well with each type of metric, while being significantly faster than using five full WAAL epochs as well[34], 50 train set epochs will be used for an acquisition batch size of 100. The training of the discriminator is stopped after these 50 train set epochs as well. Finishing the full WAAL epoch for the discriminator (in purple) does not outperform simply stopping training after 50 train set epochs (in red), while being more time consuming.

[34]The full iterative training takes approximately 3.5 hours using 50 train set epochs, while five full WAAL epochs take approximately 11.5 hours.




(a) Accuracy

(b) Macro-F1

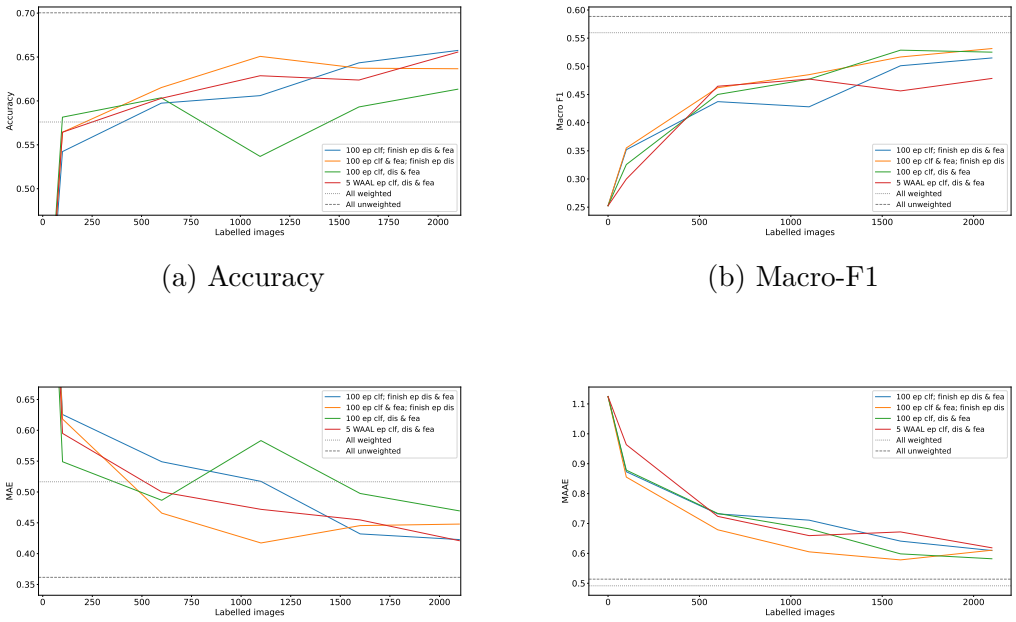
(c) MAE

(d) MAAE

Figure C.7: Comparison of using numbers of epochs for WAAL using an acquisition batch size of 100.

Figure C.8 similarly the results are shown for and acquisition batch size of 500. For the unweighted metrics, using 100 train set epochs (in green) performs the worst. After having selected 2000 datapoints using active learning, 5 WAAL type epochs (in red) and 100 train set epochs for the classifier while finishing the last WAAL epoch for the discriminator and feature extractor (in blue) perform the best. Only finishing the WAAL epoch for the discriminator (in orange) performs well in the first iterations of acquiring 500 datapoints, but less in the last iteration.
When the macro F1-score is considered, using 5 WAAL type epochs performs the worst, with 100 train set epochs and 100 train set epochs while finishing the last WAAL epoch for the discriminator perform the best. Using MAAE, 100 train set epochs performs the best after acquiring 2000 images with ative learning. Using 100 train set epochs while continuing the training of the discriminator for the last WAAL epoch outperforms the other settings when less datapoints were acquired.

Given that 100 train set epochs and continuing the training of the discriminator in the last WAAL epoch performs reasonably well for each metric after having acquired 2000 datapoints with active learning, while generally outperforming other setting when less data is acquired, this is chosen as setting for an acquisition size of 500 datapoints per active iteration.




(a) Accuracy

(b) Macro-F1

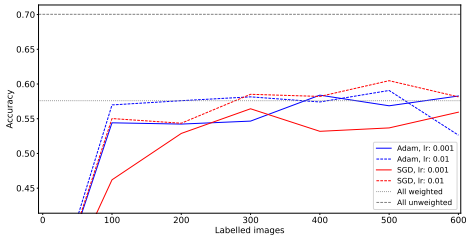
(c) MAE

(d) MAAE

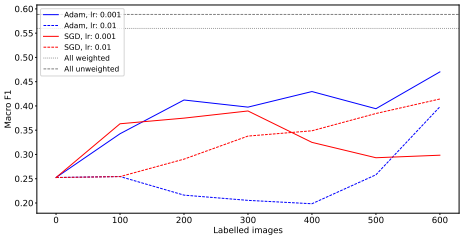
Figure C.8: Comparison of using numbers of epochs for WAAL using an acquisition batch size of 500.

**Optimizer and learning rate**

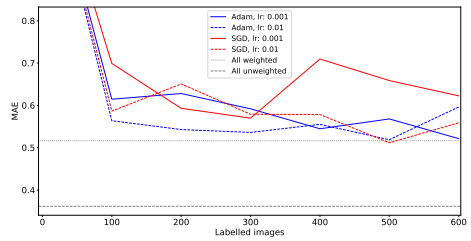
Next the optimizer and learning rate are chosen. This is only tested for an acquisition batch size of 100, given its smaller computation time. Results are displayed in Figure C.9. For each metric, using Adam as optimizer with a learning rate of 0.001 (the solid blue line) performs similar to the best other combination or better. On top, Adam with a learning rate of 0.001 is used for the other AL methods as well. Given its good performance, as well as being more comparable to other methods, Adam with a learning rate of 0.001 is used. Additionally, the learning rate reduction as explained in Section 4.4.3 is used from this moment as well, making it more comparable to the other AL methods. Figure C.10 displays results of using Adam with a constant learning rate of 0.001 and with a reducing learning rate while starting with 0.001. Results are comparable, with the main difference being that the reducing learning rate gives more stable improvements for macro-averaged metrics, while those improvements are more fluctuating when using a constant learning rate.



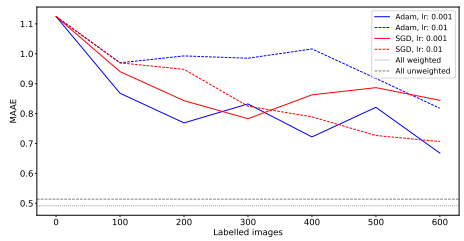

(a) Accuracy




(b) Macro-F1

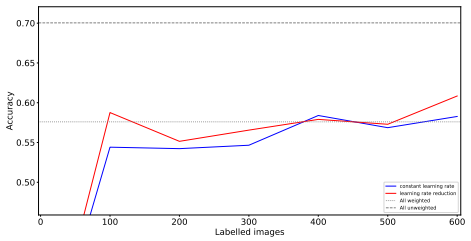



(c) MAE

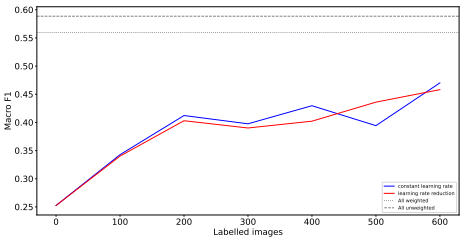



(d) MAAE

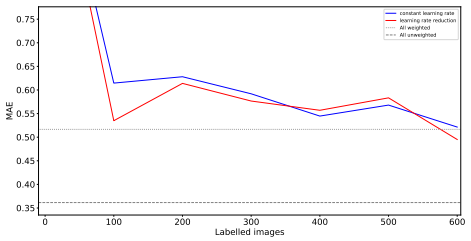
Figure C.9: Comparison of using different optimizers and learning rates for WAAL.



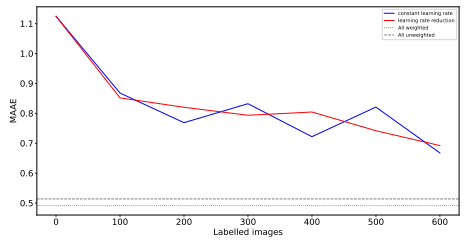

(a) Accuracy




(b) Macro-F1




(c) MAE




(d) MAAE

Figure C.10: Comparison performance of WAAL using constant and reducing learning rates.

**Wasserstein distance balancing parameter ($C_0$)**
As explained in Section 4.4.3, the Wasserstein distance balancing parameter is set by assessing the output of the discriminator. In Table C.1, the results are shown for all implemented settings. The mean output and standard deviation of the output are given for both the labelled and unlabelled data for each query step. Using $C_0 = \frac{L}{U}$ and $C_0 = 0.5$ give similar results, with the results in query steps three up to five being identical. The average output of discriminator is close to one for both the labelled and unlabelled data. Thus the discriminator seems to have difficulties to find differences between labelled and unlabelled data. This is probably partly due to both values of $C_0$ putting more importance on the unlabelled data in training, resulting in the discriminator reaching good performance by simply predicting almost all points to be unlabelled. The standard deviation is small as well, especially for the unlabelled dataset. Given that there is barely a difference in the output of the discriminator for unlabelled data, these values of $C_0$ are not very useful for the query step which uses this output to ensure diversity in the selected datapoints.

$C_0 = 1$ yields more desirable results. Predicted values are not always close to one and the standard deviation for unlabelled data is larger, making the discriminator more useful for the query step. While the average output is larger for unlabelled data compared to labelled data as expected when maximizing the Wasserstein loss given in Equation (28), differences are small in some query steps. To verify its performance using more different runs, Table C.2 contains the results of using $C_0 = 1$ for multiple runs. The results are comparable, with usually the output for the unlabelled data larger than the output of the labelled data. In run 1 however, the output is larger for the labelled dataset compared to the unlabelled datasets in two query steps. Thus, even when $C_0 = 1$, the discriminator does not always perform as desired. This could be due to the small amount of labelled data it can train on. However, this issue would then be expected to get smaller after multiple query steps, which does not happen very clearly. Another possible cause is that the data is difficult in general, which is explained more detailed in the discussion, making discriminating between similar datasets difficult as well.

In further research, $C_0 = 1$ is used given its better performance relative to the other tested values.

Table C.1: Discriminator output for labelled and unlabelled data in different query steps, comparing different settings of $C_0$. L and U refer to the number of labelled and unlabelled datapoints in the query step.

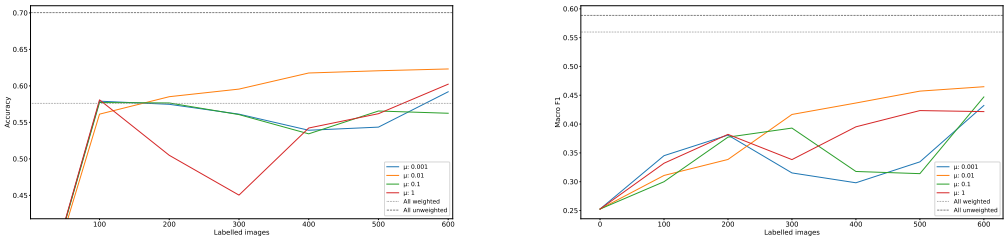
| | | mean (std) | | | | | |
|---|---|---|---|---|---|---|---|
| $C_0$ value | | L/U | | 0.5 | | 1 | |
| Data type | | Labelled | Unlabelled | Labelled | Unlabelled | Labelled | Unlabelled |
| Query step | 1 | 0.9595 (0.1969) | 0.9989 (0.0264) | 0.9157 (0.2715) | 0.9877 (0.0953) | 0.8028 (0.2896) | 0.8681 (0.2019) |
| | 2 | 0.9600 (0.1965) | 0.9996 (0.0196) | 0.9392 (0.2245) | 0.9888 (0.0877) | 0.7032 (0.3921) | 0.7864 (0.3340) |
| | 3 | 0.9600 (0.1963) | 0.9999 (0.0088) | 0.9600 (0.1963) | 0.9999 (0.0088) | 0.4156 (0.3895) | 0.5962 (0.3639) |
| | 4 | 1.0 (0.0) | 0.9990 (0.0319) | 1.0 (0.0) | 0.9990 (0.0319) | 0.4466 (0.4487) | 0.6216 (0.4257) |
| | 5 | 0.9920 (0.0892) | 0.9993 (0.0266) | 0.9920 (0.0892) | 0.9993 (0.0266) | 0.2916 (0.3885) | 0.4262 (0.4211) |

Table C.2: Discriminator output for labelled and unlabelled data in different query steps, comparing different runs with $C_0 = 1$.

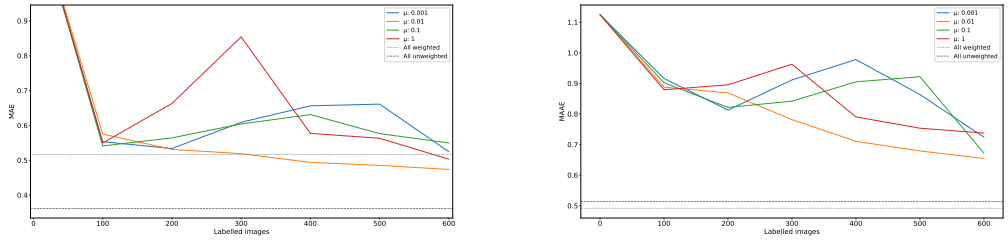
| | | mean (std) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Run 1 | | Run 2 | | Run 3 | | Run 4 | |
| Data type | | Labelled | Unlabelled | Labelled | Unlabelled | Labelled | Unlabelled | Labelled | Unlabelled |
| Query step | 1 | 0.8185 (0.2530) | 0.8966 (0.1342) | 0.4046 (0.1229) | 0.4342 (0.0891) | 0.1621 (0.1285) | 0.1708 (0.1103) | 0.9022 (0.2675) | 0.9436 (0.1750) |
| | 2 | 0.2646 (0.3587) | 0.2757 (0.3720) | 0.1730 (0.1559) | 0.1832 (0.1248) | 0.6502 (0.4556) | 0.9405 (0.2089) | 0.1484 (0.2823) | 0.1766 (0.2955) |
| | 3 | 0.2968 (0.3964) | 0.2874 (0.3915) | 0.7077 (0.3878) | 0.8843 (0.2391) | 0.7190 (0.4452) | 0.9263 (0.2557) | 0.2764 (0.4061) | 0.3103 (0.4220) |
| | 4 | 0.2647 (0.3957) | 0.2121 (0.3585) | 0.8045 (0.3671) | 0.8806 (0.2956) | 0.8312 (0.3705) | 0.9522 (0.2097) | 0.3082 (0.4237) | 0.3746 (0.4508) |
| | 5 | 0.8114 (0.3433) | 0.8269 (0.3103) | 0.7521 (0.4101) | 0.8163 (0.3655) | 0.7574 (0.4247) | 0.9003 (0.2955) | 0.2464 (0.4124) | 0.3417 (0.4580) |

**Classification and Wasserstein distance loss trade-off for training ($\mu$)**
In Figure C.11, the results for different values of $\mu$ are given. $\mu = 0.01$ (in orange) results in the best performance for each of the metrics. Therefore, $\mu = 0.01$ is chosen, which is the setting used previously already[35].

[35]Of course, the best performance of $\mu = 0.01$ could partly be due to this. Other hyperparameters were tuned using this setting, and may hence be chosen to work well in combination with $\mu = 0.01$, while other settings may have worked well with other previously determined hyperparameter settings.



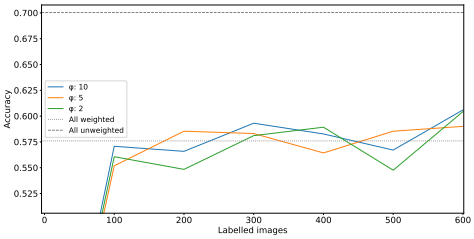

(a) Accuracy (b) Macro-F1



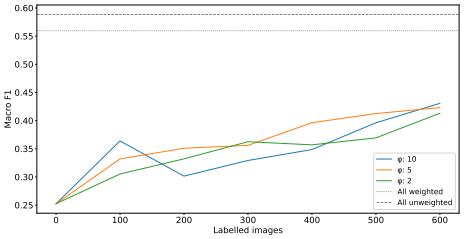

(c) MAE (d) MAAE

Figure C.11: Comparison of using different loss trade-off parameters $\mu$.

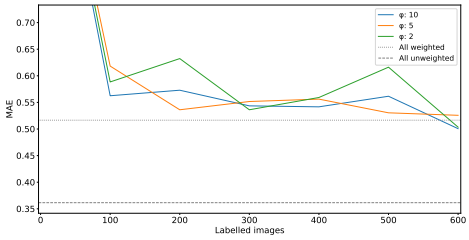
**Diversity and uncertainty trade-off for query ($\phi$)**
In Figure C.12, the comparison is displayed for different settings of $\phi$. Results are generally similar, but $\phi = 2$ (in green) shows more fluctuating performance for MAE, while $\phi = 10$ (in blue) shows worse performance in some active iterations when assessing the macro F1-score. $\phi = 5$ (in orange) most consistently performs well and is therefore chosen.



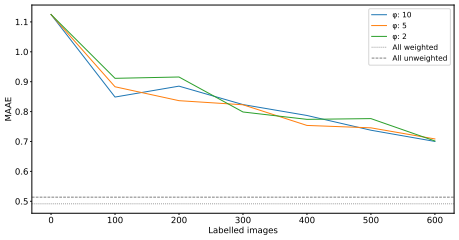

(a) Accuracy




(b) Macro-F1




(c) MAE

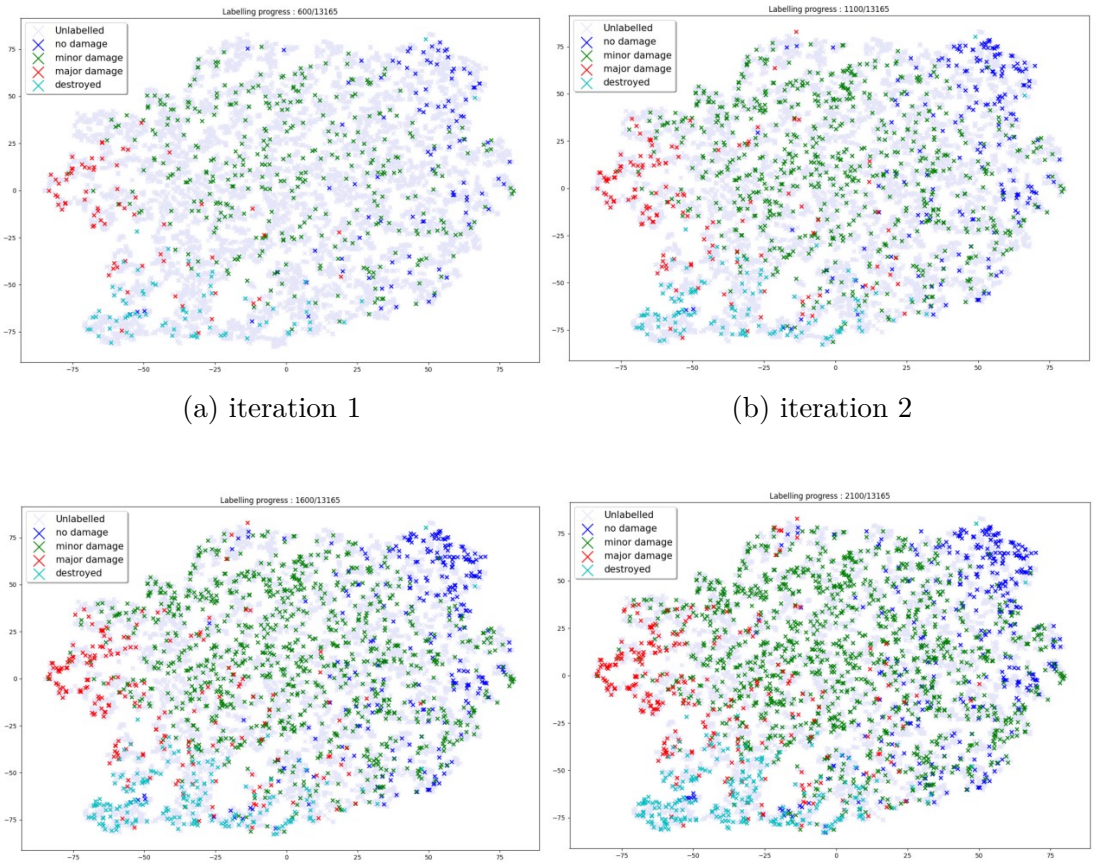



(d) MAAE

Figure C.12: Comparison of using different loss trade-off parameters $\mu$.

# D Appendix to Section 6: Results

## D.1 Appendix t-SNE plots




(a) iteration 1

(b) iteration 2

(c) iteration 3

(d) iteration 4

Figure D.1: t-SNE plots for Random in different active iterations, using four active iterations selecting 500 images in each iteration. Training is performed using 100 epochs in each iteration with warm restart each 20 epochs.