Addressing the class imbalance in aerial images with Generative Adversarial Networks

Itzel Belderbos

February 2021

Contents

Abstract

This research examines whether we can make use of Generative Adversarial Networks as an oversampling technique for aerial solar panel data. Previous work has shown that an imbalance between classes in the training set can harm the performance of a classification model, having a severe bias towards the majority group. As the training data consists of 80% of positives, this bias towards the negatives can be derived from the low recall rate (55.2%) on the test set covering another geographical location. Therefore, we first perform t-SNE and clustering methods on the feature embeddings extracted by the pre-trained network, in order to segregate the training data into 5 clusters, of which four consist of more than 99% positives. We use these clusters as input for our experiments with three types of GANs: vanilla conditional Deep Convolutional GANs, conditional Deep Convolutional Wasserstein GANs with several types of 1-Lipschitz continuity enforcements, and conditional Deep Convolutional Self-Attention GANs with either binary cross-entropy loss or Wasserstein loss. After GAN training, we added the generated positive images to the original training set in order to re-train the VGG11-based classification model. The results show that using the clusters as a GAN input improves the diversity of images and lowers the risk of mode collapse, compared to using the classes as input. Moreover, the optimal GAN model obtains a Fréchet Inception Distance of 181.82 and an improvement in test recall rate of 6.14%, after increasing the percentage of positives from 19.59% to 35.13%.

1 Introduction

In recent years, the improvements in supervised classification of geographical images have shown great success. However, the geographical generalization ability of the models is mostly dependent on how representative the dataset is. In case that the spatial location of the test set is different from the data in the training set, it is possible that the performance on the test set is not as desired. One of these obstacles is the class imbalance problem, where one class is over-represented compared to the other classes. When the class distribution is imbalanced in binary classification, the model might be biased towards the over-represented class. Simply acquiring more labeled data for the training set in order to improve the balance is expensive and time-consuming, which leads to the need for semi-supervised or unsupervised learning techniques [1].

1.1 Problem statement

In 2019, Statistics Netherlands developed a transfer learning model which classifies whether an aerial image contains one or multiple solar panels. The goal of this prediction model is to eventually use this semi-automated model as sustainability indicator for The Netherlands. The model consists of a ImageNet pre-trained VGG16 architecture until the last block (block5_pool) with a classification layer on top. The first two convolutional blocks were frozen during fine-tuning, while the rest of the blocks were re-trained on the Heerlen aerial dataset.

However, testing the model on a test set of a different geographical location shows that the model does not seem to perform as well as on the training set.

Especially the recall seems to decline severely due to the false negatives. As the training set consists of 20% positives, while the test set consists of 7% positives, this could be due to the class imbalance problem. It can be presumed that because it is trained on a majority of negatives, the model is biased towards negatives. Nonetheless, in the field of solar panel recognition, this relatively low percentage of positives in an aerial image dataset is common, as obviously it is easier to obtain an aerial image without a solar panel than an image with a solar panel. In addition, the large proportion of false negatives is not desirable for this model, as this could inject bias in the analysis of sustainability indicators by a lower indication of solar panels than the actual number. Omitting a sample of negatives from the training set in order to create a more balanced dataset would not be an favorable solution, as the training set solely consists of 23,847 samples of which 4,672 positives. This solution will result in a dataset with 9,344 samples, which is too small for a deep learning model to learn. Therefore, we would like to find a way to overcome this lack of data, while simultaneously being able to choose the specific class of data we would like to obtain.

1.2 Research goal and hypotheses

The goal of this research is to analyze if we can find the structure of both datasets as well as determining whether we can find a suitable method to create more data of the minority class. Moreover, we want to idenfity whether the addition of this minority data will lead to better performance on a test set of different location.

The research questions can be formulated as following:

- Can we identify the structure of the two datasets on two different geographical locations?
- Are Generative Adversarial Networks suitable generation methods for aerial images?
- Is the addition of synthetic images generated by Generative Adversarial Networks able to improve the performance on another geographical location?
- Which type of Generative Adversarial Networks is the most suitable for generating aerial images?

1.3 Research approach

This research consists of two parts: an exploratory analysis to obtain a better understanding of the data and its distribution, followed by the generation of synthetic images by means of different types of GANs.

1.3.1 Exploratory analysis

In the exploratory analysis, the trained classification model is used to predict on the test set with aerial images covering Zuid-Limburg (The Netherlands). The misclassified samples are further analyzed to see which types of images the model does not seem to identify well. Subsequently, t-SNE, a dimensionality reduction technique, is used to visualize the distribution of both train and test set. Clustering methods are used to gain more insights into the structure of the datasets, as well as a preprocessing step for the GANs.

1.3.2 Generation of positives with GANs

As stated in the problem statement, we would like to identify whether the addition of positives to the training set will improve the classification of positives in the test set. Several types of conditional Deep Convolutional GANs (vanilla GAN, Wasserstein GAN and Self-Attention GAN) are compared to each other. Their performances are compared both qualitatively and quantitatively; how realistic the generated images are, as well as the performance increase on the test set and the Fréchet Inception Distance metric, explained in Section 3.10.1.

Originally, conditional generation implies being able to condition the output of the generated images on the class, in this case positive or negative. However, in order to control the type of image to be generated, we use the cluster labels obtained in the exploratory analysis as input condition. Thus gives the possibility to choose what type of solar panel image we would like to obtain, while also forcing the GAN model to generate more diverse samples. The cluster analysis has shown that the train data can be divided into clusters where the positives and negatives are finely segregated into different clusters. This makes the clustering labels suitable for conditioning.

2 Related work

Research has shown that an imbalance between classes in the training set can harm the performance of a classification model, having a strong bias towards the majority group [2]. This bias is even more severe in the case of high-dimensional data, such as images [3]. There are various techniques which aim to address this issue, such as oversampling, undersampling and ensemble learning. Oversampling techniques, and in particular Synthetic Minority Oversampling Technique (SMOTE), are most widely used. SMOTE identifies the k nearest neighbors for every minority sample based on the Euclidean distance. Nonetheless, for high-dimensional data such as RGB images, too many dimensions lead to every sample to appear equidistant from the other data points, which is called the curse of dimensionality. As these techniques are more focused on this local neighborhood information, these techniques may not be suitable for synthetic image generation [4].

Recently, Generative Adversarial Networks (GANs) have gained attention as an oversampling and data augmentation technique for high-dimensional data. The vanilla Generative Adversarial Networks (GANs) [5] consist of two neural networks: a generator and a discriminator. Generally, GANs model the real data distribution by simply imitating that distribution. The generator tries to fool the discriminator by creating data as similar to the real data as possible. In contrary, the discriminator classifies whether the given instance, either generated by the generator or sampled from the real data, is real. Mathematically, this is achieved by a minimax loss function (or more recent loss functions), which the discriminator tries to maximize and the generator tries to minimize.

Researchers have shown that synthetic GAN images are suitable as additional training samples in order to improve classification [6]. Douzas et al. [7] successfully compared the performance of GANs with other oversampling techniques for binary class data on 71 datasets and found that GANs outperform other methods. However, generating minority samples with a vanilla GAN might be difficult, as there might not be sufficient minority data to train a GAN. A conditional GAN, which is able to condition the image generation on a specific class, would be more suitable for this oversampling task as it makes use of all data. In this way, the GAN is able to learn features for the majority class to generate samples for the minority class [8]. Moreover, Deep Convolutional GANs (DGGANs) [9] are an extension of the vanilla GAN, but they make use of deep convolutional networks for both the discriminator and generator. Frid-Adar et al. [10] have successfully implemented the conditional DCGAN as a method to generate synthetic data for medical image classification. They found that GANs outperform simple oversampling techniques due to their ability to recover the data distribution. Another improvement in the GAN architecture is the Wasserstein GAN (WGAN) [11], which uses another loss function instead of the binary cross-entropy loss. With the addition of gradient penalty (WGAN-GP) to control learning, it diminishes the vanishing gradient problem the original GAN has [12]. Zheng et al. [13] have succesfully tested the conditional WGAN-GP as an oversampling approach on 17 datasets to empirically show the increased performance in classification.

Various researchers have also tried to investigate combinations of different models. Shamsolmoali et al. [14] have integrated Capsule Neural Networks and GANs to generate the minority class. A more end-to-end framework is proposed by Mullick et al. [15], which came up with a framework consisting of a discriminator, generator and classifier to address class imbalance. The generator learns to generate the minority class which are misclassified by the classified, whereas the classifier learns to classify samples as minority or majority class.

However, these works are highly focused on solving the class imbalance in empirical datasets, such as MNIST and ImageNet. To our knowledge, oversampling with GANs has not been performed before in the field of remote sensing and especially solar panel data.

In the field of GANs for remote sensing, several researchers have addressed the lack of annotated data with unsupervised learning. Lin et al. [16] have created MARTA GAN, a framework which performs unsupervised learning when there is not much labeled data available. Duan et al. [17] have created GAN-NL, which is another unsupervised framework which is suitable when there is a lack of annotated data. Ma et al. [18] have designed SiftingGAN, another framework which generates new samples when there is a lack of labeled data, and outperforms other data augmentation methods. Nonetheless, these remote sensing GAN frameworks are not designed to generate data of a specific class, which makes these models are not suitable for solving the class imbalance problem.

Therefore, we propose several new GAN frameworks which generate aerial solar panel data of a specific class. Additionally, we make sure we can control the generation of the minority class data by conditioning the GAN on generated clusters instead of solely the class to diversify the outputs.

3 Methodology

3.1 Data and dataset

The training dataset covers aerial images around Heerlen area (Limburg, The Netherlands) and has a resolution of 200x200 pixels. The dataset originally consists of 84,693 images, of which 23,847 are labeled. There are 4,672 positives (20%) and 19,175 negatives (80%).

The test set covers aerial images from the South of province Limburg (The Netherlands), consisting of 49,686 images, with 39,506 labeled. There are 36,693 negatives (93%) and 2,813 positives (7%) in the dataset.

For computational efficiency purposes, a sample of 10,000 images is taken from the train set to cluster the images and train the GANs. This new set contains 4,672 positives (47%) and 5,328 (53%) negatives. Hence, all positives are used for this subset, while a sample of negatives are used. Since the aim of this research is not to train a classifier, but rather to explore the dataset and generate new images, this omission of negatives is not significant. If we would train a classifier, we would need these disregarded negatives to train on all types of features, but as we will only generate positives with the GAN, taking such a subset is not necessarily disadvantageous. In contrary, when training the classifier to evaluate the GANs performance, we use the complete training set of 23,847 labeled images.

3.2 Extraction of feature embeddings

Transfer learning makes use of the generic features learned on another domain, and uses these for another domain. In computer vision, this is often performed with pre-trained models (e.g. VGG, MobileNet and Inception), trained on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) dataset [19]. While the layers at the beginning of the network learn low-level features (e.g. edges), layers close to the output learn the feature which are more specific to the domain. This makes a pre-trained model trained on a generic domain also suitable as a feature extraction model, which omits the classification layer, and substitutes this with a prediction on the (new) dataset. In this research, the VGG16 model, which is fine-tuned on the Heerlen training set, is used as a feature extractor. The output of the last flatten layer is used, giving a feature vector of 12,800 dimensions.

3.3 Clustering on extracted feature embeddings

Clustering is an unsupervised learning method, which implies that the ground truth labels of the clusters are not available. It aims to find clusters in which the samples are similar to other samples inside of group (intra-cluster distances), but dissimilar to samples of other groups (inter-cluster distances). The quality of the clustering is determined by a certain similarity measure and the capacity to explore some hidden patterns.

As already explained, in this work clustering is used as a preprocessing step and the created clusters are used as input for the GAN. Moreover, it is used as an exploratory tool to gain insights into the distribution of the training and testing data.

3.3.1 KMeans

KMeans is a partitioning methods which iteratively determines the middle point of a cluster by measuring the distances of the samples to this point. First, the samples are divided into k clusters, where k is the predefined number of clusters desired. Subsequently, the seed points are the centroids (middle points) of the clusters, and every sample is assigned to the nearest centroid, based on the Euclidean distance. These steps are repeated until the assignment of samples does not change.

A downside of KMeans is that is it sensitive to outliers, as an outlier can severely distort the mean of the data. Therefore, KMedoids (e.g. Partitioning Around Medoids) is introduced, which takes the most centrally positioned sample in the cluster as the 'centroid'. However, due to the computational complexity, PAM does not work well for larger datasets. Therefore we choose to work with KMeans.

As the number of clusters has to be defined beforehand, it might be difficult to define the optimal number of clusters. The sum of squared error (SSE) is a method that could aid defining the optimal number of clusters. The SSE is the sum of the squared distances between a sample and the cluster centroid. Equation 1 shows the formula for computing the SSE, with i as the cluster number, k as the total number of clusters, x as a data point, c_i as the centroid of cluster i and C_i as cluster i.

$$SSE = \sum_{i=1}^{k} \sum_{x \in C_i} (\text{distance}(c_i, x))^2 \tag{1}$$

Computing a clustering for several k and plotting the SSE with respect to the number of clusters k could give more insights into the optimal number of clusters.

3.3.2 Hierarchical clustering

Hierarchical clustering is a collection of algorithms that create nested clusters by splitting or merging them subsequently. Agglomerative clustering is a bottom-up form of hierarchical clustering, as it starts with every sample as a cluster, and merges the closest pair of clusters until k clusters are left.

The merging strategy is determined by the type of linkage criteria. Single linkage, average linkage and maximum linkage imply minimization of the distances between respectively the closest samples, all samples and furthest samples between 2 clusters. Ward [20] aims to minimize the sum of squared differences inside of the clusters.

3.3.3 Cluster distances

There are several distance measures for computing the clusterings. KMeans only works with the Euclidean distance, which is the most common distance measure used. The Euclidean distance computes sum of squared differences between a data point and the cluster center.

Euclidean distance/L2 norm

$$\rho(\mathbf{x}, \mathbf{y}) = \left[\sum_{i=1}^{m} (x_i - c)^2 \right]^{1/2} \tag{2}$$

Hierarchical clustering can also work with the Manhattan distance and cosine similarity. The Manhattan distance takes the sum of absolute differences between data point and cluster center. Cosine similarity quantifies the similarity between two vectors. The function divides the dot product of two vectors by the product of the norms of the vectors.

Manhattan distance/L1 norm

$$\rho(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{m} |x_i - c| \tag{3}$$

Cosine similarity

$$\begin{aligned} \cos\theta &= \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \left\|\vec{b}\right\|} \\ \|\vec{a}\| &= \sqrt{a_1^2 + \cdots + a_m^2} \\ \left\|\vec{b}\right\| &= \sqrt{b_1^2 + \cdots + b_m^2} \end{aligned} \tag{4}$$

3.3.4 Cluster validity evaluation

The validity of a clustering can be based on the cohesion (intra-cluster distances), separation (inter-cluster distances) of the formed clusters, or a combination of these. In this case, we do not have access to the ground truth labels of the clusters. Therefore, we make use of metrics which do not require these labels: Silhouette coefficient and Calinski-Harabasz index.

Silhouette coefficient
Silhouette coefficient [21] takes into account a combination of cohesion and separation. The score is bounded between -1 (completely incorrect clustering) and $+1$ (very good clustering). In case the score is around 0, it implies that there are many clusters with overlapping data. This metric can also be used to compare two clusterings or two clusters. The Silhouette Coefficient s for a single data point can be given:

$$s = \frac{sep - coh}{\max(coh, sep)} \tag{5}$$

Where *sep* is the mean distance between a data point and all other samples in the next closest cluster, while *coh* is the mean distance between a data point an all other samples in the same cluster.

Calinski-Harabasz Index
The Calinski-Harabasz Index [22] is another evaluation metric which is suitable when ground truth labels are unknown. It is an index which is the ratio of the sum of squared distances between clusters and inside of clusters. The index can be formulated as:

$$s = \frac{tr(B_k)}{tr(W_k)} \times \frac{n-k}{k-1} \tag{6}$$

Where n_D is the number of data samples and k is the predefined number of clusters. $tr(W_k)$ and $tr(B_k)$ are the trace of the within cluster dispersion matrix respectively the between cluster dispersion matrix. These two matrices are formulated as:

$$W = \sum_{i=1}^{k} \sum_{x \in C_i} (x - c_i)(x - c_i)^T \tag{7}$$

$$B = \sum_{i=1}^{k} n_i (c_i - c_D)(c_i - c_D)^T \tag{8}$$

Where c_i is the center of cluster i, c_D the center of the data D and n_i the number of samples in the cluster. The higher the score, the better the separation of the clusters.

3.4 t-SNE

T-Distributed Stochastic Neighbor Embedding (t-SNE) is a nondeterministic method which aims to visualize high dimensional data into two or three dimensions, without losing too much information. Hence, it is a form of dimensionality reduction which measures pairwise distances between high-dimensional objects. The idea behind t-SNE is that it uses an objective function which measures the discrepancy between the similarities between samples in the (low-dimensional) map and similarities between samples in the original data. Nowadays, Principal Component Analysis (PCA) is a widely-used method for dimensionality reduction. However, this technique is not suitable for high-dimensional data, as PCA aims project the samples into a low-dimensional subspace such that the variance is maximized. Due to this variance maximization, mostly the distances between the dissimilar points are preserved. Nonetheless, the higher-dimensional the data, the less important the distances between dissimilar points, as they become less indicative of the real data structure. An example of this is shown in Figure 1. We can clearly see that in this 3D setting, the points with the same color (similar points) are more indicative for the structure than the points further away from each other (dissimilar points). When we want to map the data to a lower dimensional subspace, we thus want to preserve these smaller distances.

Hence, when the data is high-dimensional, it is desired to mostly maintain the small distances between data points in the mapping, while the large distances between the data points are more negligible.

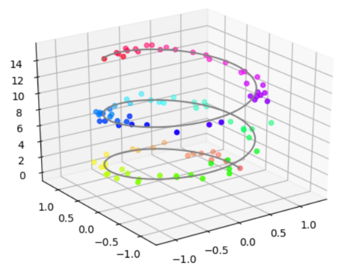

Figure 1: Example of high dimensional data [23]

In Equation 9, the probability p_{ij} of picking a pair of points i and j, proportional to how similar these points are in the original data. In contrary, q_{ij} in Equation 10 computes the probability of picking a certain pair of points proportional to how similar the points are in the (low-dimensional) map.

$$p_{ij} = \frac{\exp(-||x_i - x_j||^2/2\sigma^2)}{\sum_k \sum_{l \neq k} \exp(-||x_k - x_l||^2/2\sigma^2)} \tag{9}$$

$$q_{ij} = \frac{\exp(-||y_i - y_j||^2)}{\sum_k \sum_{l \neq k} \exp(-||y_k - y_l||^2)} \tag{10}$$

The goal of t-SNE is to make the two probability distributions q_{ij} and p_{ij} as similar are possible, in order to not lose too much information contained in the original data. This discrepancy is measured by the Kullback-Leibner (KL) divergence:

$$KL(P||Q) = \sum_i \sum_{j \neq i} p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{11}$$

Hence, the points in the low-dimensional map are moved around in order to minimize the KL-divergence by means of gradient descent.

3.5 The vanilla GAN

A Generative Adversarial Network consists of two parts: a generator and a discriminator, which are both neural networks. The goal of the GAN is to make the real and fake distributions as similar as possible. In the original GAN, the discriminator is a classifier which aims to determine whether a given image is real or fake. It tries to evaluate the conditional probability of the class (real/fake) Y, given the data/features X:

$$P(Y|X) \tag{12}$$

In contrary, the generator creates samples of a specific class. In order to ensure that the generator does not generate only one type of image, it uses a random noise vector as input, which inject randomness into the generation of images. The generator evaluates the probability of the features/data X, and models the real distribution of the features, having a higher probability to generate the more common features and being less likely to sample the more rare types of features.

$$P(X) \tag{13}$$

The binary cross-entropy (BCE) cost function is used to determine the GAN loss. The BCE loss is the average of the discriminator cost due to wrongly classifying fake and real observations. This loss function is defined as:

$$L(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log p(x^{(i)},\theta) + (1-y^{(i)})\log(1-p(x^{(i)},\theta))] \tag{14}$$

In Equation 14, $y^{(i)}$ is the binary true label (fake/real), $x^{(i)}$ are the features of the images, θ are the parameters and $p(x^{(i)})$ is the prediction from the discriminator.

The first part of the function $(y^{(i)}\log p(x^{(i)},\theta))$ is focused on the real images. This is because when the true label $y^{(i)}$ is 0, this term will obviously also be 0, regardless of how well the prediction is. Hence, this first part will always be canceled out for the fake images, and thus only have an influence on the cost for the real images. For the real images, when the prediction is really good and thus close to 1, this term will approximate 0, as $1 \times \log 0.99 \approx 0$. Hence, when the prediction is close to the true label, there will be low cost. In contrary, when the prediction is far from the true label and thus close to 0, the output of this term will be $-\infty$. This minus will be canceled out by the minus at the beginning of the function. In short, the first term of the function will give high cost the prediction for real images is bad, and will give almost zero cost when the prediction for the real images is close.

The same applies for $(1-y^{(i)})\log(1-p(x^{(i)},\theta))$, the second term of the loss function. This term only applies to fake samples, as the $(1-y^{(i)})$ cancels out for the real samples. This term solely plays a role when the prediction is 0, and will give a high cost when the prediction for the fake samples is bad and low cost when the prediction is close to the true label.

The discriminator tries to minimize its cost, while the generator tries to maximize this cost: a minimax game. The GAN trains in an alternating way: the discriminator trains by updating its parameters on the basis of its own classification of the real and fake images, while the generator learns with the feedback from the discriminator, trying to generate images that fool the discriminator more easily. It is important that both discriminator and generator improve, as otherwise the feedback is not helpful.

3.6 Deep Convolutional GANs

In normal GANs, the discriminator and generator are fully connected layers. In Deep Convolutional GANs [9], the model consists of convolutional layers,

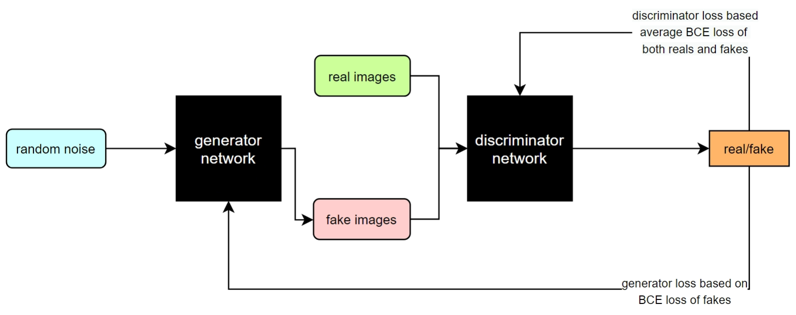
random noise
generator network
real images
fake images
discriminator network
real/fake
discriminator loss based average BCE loss of both reals and fakes
generator loss based on BCE loss of fakes

Figure 2: Generative Adversarial Network

which perform downsampling with the convolutional stride and upsampling with transposed convolutions. Downsampling is used in the discriminator, as the network receives an $C \times H \times W$ image and downsamples it into a scalar prediction. In contrary, the generator upsamples the noise vector into a $C \times H \times W$ image. Moreover, batch normalization is introduced between the convolutional layer and the activation layer, except from the input layer of the discriminator and the last layer of the generator. In addition, ReLU is used in the generator blocks, while tanh is used in its last block. For the discriminator, the activation LeakyReLU is used, instead of sigmoid.

3.7 Conditional GANs

The vanilla GAN does not specify the classes which the generator creates, while the conditional GAN adds an extra condition class to the output of the GAN. This implies that the training dataset needs to be labeled, while in the original case this requirement was not specified. In order to make a GAN conditional, the input of the generator is the noise vector with a one-hot vector of the class appended.

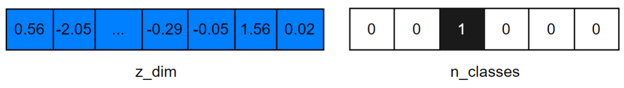

Figure 3: Concatenation of noise and one-hot labels

The discriminator also has this class information added to its input image in the form of one-hot class matrices. As visual representation is shown in Figure 4. As the image shows, the number of matrices appended to the image is dependent on the number of classes. In this example, the number of classes is 6 and the third class is the class which belongs to the image.

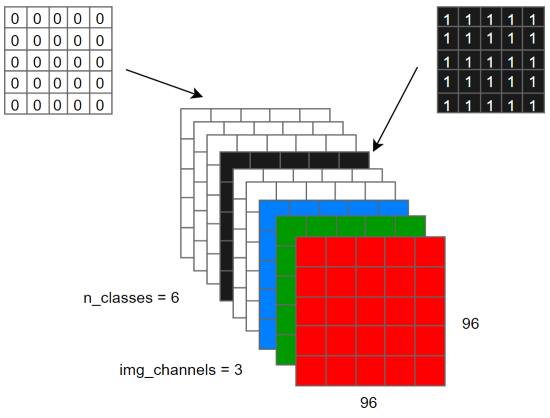

Figure 4: Concatenation of image and matrix one-hot labels

3.8 Wasserstein GANs

3.8.1 The problem with BCE loss

The vanilla GAN is trained with BCE loss, a cost function derived from the binomial distribution and ideal for binary classification tasks. However, it is evident that the generator has to provide a very complex output (an image), while the discriminator only has to output 1 or 0. Hence, the task of the discriminator is much easier than the generator's task, which makes it common for the discriminator to perform better than the generator. In the beginning of training this is not yet the case, as the discriminator has not yet learned to distinguish properly between reals and fakes. Thus, the discriminator still gives valuable feedback to the generator in the form of a nonzero gradient. But as the discriminator improves, it begins giving feedback in the form of numbers close to zeros or ones, which in turn is not valuable for the generator, as it is unable to improve. As the cost function approximated by the BCE loss has flat regions, vanishing gradients (gradients close to 0) can occur, leading to mode collapse. Mode collapse means that the generator tends to only generate one type of sample, as it learned that this sample easily fools the discriminator. This problem is even more severe in unconditional GANs, as the model is not forced to generate different classes of outputs, but it can generate any image.

3.8.2 Earth Mover's Distance

The Earth Mover's Distance (EMD) is a distance which gauges the distance between two distributions. Generally, the EMD computes the effort for making the fake distribution similar to the real distribution. Hence, this effort is dependent on the amount and distance which has to be 'moved' to make them similar. The advantage of EMD compared to BCE loss is that there is not a restriction of the loss being between 0 and 1. The gradient will not approach zero, even when the distributions are very dissimilar. This lack of flat regions in the loss functions makes it less prone to vanishing gradient problems, and thus also to mode collapse.

3.8.3 Wasserstein loss

The BCE loss can be simplified into:

$$\min_{d} \max_{g} -[\mathbb{E}(\log(d(x))) + \mathbb{E}(1 - \log(d(g(z)))))] \tag{15}$$

The generator wants to maximize the cost, while the discriminator wants to minimize the cost. The sum of the samples i to m divided by the number of samples m is actually an expected value, where $y^{(i)}$ equals 1 in the first term, and in the second term $y^{(i)}$ equals 0, which makes the $(1 - y^{(i)})$-term 1.

In contrary, the Wasserstein loss approximates the Earth Mover's Distance:

$$\min_{g} \max_{c} \mathbb{E}(c(x)) - \mathbb{E}(c(g(z))) \tag{16}$$

Hence, the main difference with the simplified BCE loss function in Equation 15 is that the function does not contain any logarithms. In this case, the discriminator is called the 'critic', as it does not classify anymore, but evaluates the distance. The critic tries to maximize the distance between the real and fake distribution, while the generator aims to minimize the distance. The loss is not bounded between 0 and 1, because it outputs a real number representing the distance. This W-loss can be used as the new cost function for the Wasserstein GAN, introduced by Arjovsky, Chintala and Bottou [11].

3.8.4 1-Lipschitz continuity

The Wasserstein GAN has the requirement that the gradient of the critic is stable. This is controlled by 1-Lipschitz continuity, implying that the norm of the critic's gradients should be at most 1 at every point of the function. In this way, the gradient cannot grow more than linearly. This makes sure that the W-loss is not only differentiable and continuous, but also stable during training. This is shown in the Equation below, with c as the critic and x as the image.

$$\|\nabla c(x)\|_2 \leq 1 \tag{17}$$

However, there are multiple ways to enforce 1-L continuity on the Wasserstein GAN. The first way is weight clipping [11]: imposing the critic's weights to be between bounds, meaning that the weight values which are lower or higher than these bounds are 'clipped' to these bounds. However, the main risk of this method is that it inhibits the learning of the critic, which could lead to poor performance.

Another way to ensure 1-L contintuity is gradient penalty, which is a form of regularization of the critic's gradient, introduced by Gulrajani et al. [12]. This term penalizes the critic when its gradient norm is larger than 1. However, it is impossible to check whether the gradient norm is smaller than 1 at every point in the feature space. Therefore, points are sampled by means of interpolation between reals and fakes. The reals get a weight of ϵ and the fakes get a weight of

$1-\epsilon$, which leads to a random interpolation $\hat{x}$. The gradient norm of $\hat{x}$ should be less than or equal to 1.

$$\hat{x} = \epsilon x + (1-\epsilon)g(z) \tag{18}$$

In the function above, x is the real image, $\hat{x}$ is the interpolated image, z is the noise vector and $g(z)$ is the generated image. This leads to the following regularization term:

$$(\|\nabla c(\hat{x})\|_2 - 1)^2 \tag{19}$$

As the gradient norm of the critic is requested to be (lower than) 1, the -1 between the brackets will penalize any value which is not 1. Thus, instead of penalizing any value larger than 1, it penalizes any value which is not 1. The square of this term rather than the absolute value makes sure that values very different from 1 are penalized more. Eventually, the new Wasserstein loss function with gradient penalty and gradient penalty weight λ is defined as:

$$\min_g \max_c \mathbb{E}(c(x)) - \mathbb{E}(c(g(z))) + \lambda \mathbb{E}(\|\nabla c(\hat{x})\|_2 - 1)^2 \tag{20}$$

A visual representation of the process of computing the gradient penalty is shown in Figure 5. The figure shows how first an interpolation of the real and fake image is computed, and with the concatenation of the interpolation and the one-hot labels the critic predicts the distance. The interpolated image and the critic prediction are used to compute the gradients of the interpolated image, after which the norm is computed. In the image, this is shown for one single image, while in practice the norms are computed for one batch simultaneously. The average of the norms of a batch is computed and used to calculate the gradient penalty.

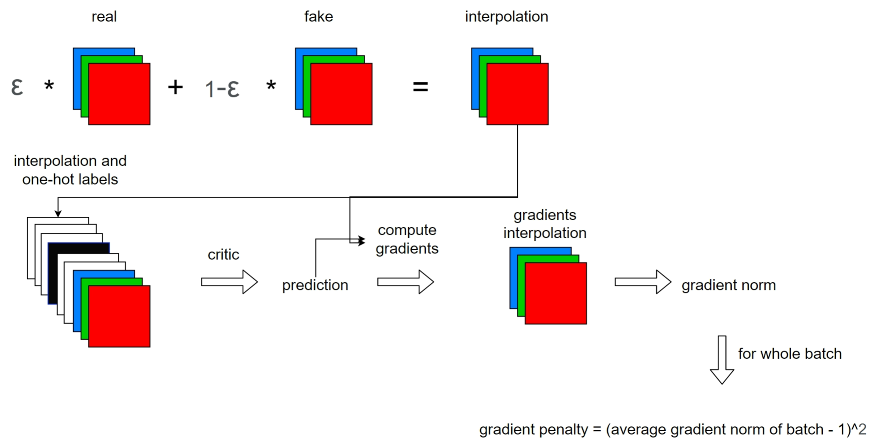

Figure 5: Computing the gradient penalty

A lighter Lipschitz regularization technique is Spectral Normalization (SN), introduced for GANs by Miyato et al [24]. SN is a weight normalization method which stabilizes the training of the discriminator by normalizing the spectral norm of the weight matrices. In this way, it stabilizes the training of the discriminator and enforces 1-L contintuity, lowering the risk of mode collapse. The advantage of this method is that it does not require much hyperparameter tuning and it is very easy to implement, while also not making to model more computationally expensive.

3.9 Self-Attention GAN

Although Deep Convolutional GANs are good in generating images with geometric structures (e.g. landscapes), DCGANs sometimes fail to generate the complete object accurately, e.g. two separate legs of an animal. This is due to the convolutions in the DCGAN, which are local operations. The convolutional filters are excellent in discovering the local structure of the image, but the receptive fields might not have a sufficiently large size to discover bigger structures. Increasing the kernel size or making the network deeper is at the expense of the computational efficiency, while GANs are already really expensive to train. Therefore, the concept of self-attention applied to GANs is introduced by Zhang et al [25].

The addition of a self-attention module to the GAN could improve the discovery of these non-local dependencies in images. Intuitively, the self-attention module computes what part of the feature map should receive more attention by means of interactions between the different parts.

The concept of self-attention uses three representation matrices: query (Q), key (K) and value (V) matrices. Conceptually, the query matrix entails the representation of every position related to itself, the key matrix contains the representation of every position with respect to other positions. The value gives the weight of importance of the attention at every position. The most important positions should obtain more weight than insignificant ones. In Equation 21, W_i is the representation matrix and F the flattened image input.

$$\begin{aligned} Q &= W_q^\top F \\ K &= W_k^\top F \\ V &= W_v^\top F \end{aligned} \tag{21}$$

The importance of two specific positions relative to each other is computed by the dot product of the query (Q) and key (K) matrices, which is called dot product attention. This dot product is converted to a probability distribution by means of a softmax. Both the discriminator and generator make use of the self-attention. The computation which is used for the Self-Attention GAN [25] is shown in Equation 22, which is slightly adapted from the official self-attention paper [26]. Moreover, spectral normalization is used for the weights of the representation matrices.

$$\begin{aligned} \text{attention} &= \text{softmax}(QK^\top) \\ \text{scaled attention} &= (\text{attention})V \end{aligned} \tag{22}$$

The proposed module is visualized in Figure 6. In the figure, $f(x)$, $g(x)$ and $h(x)$ are the query, key and value matrices.

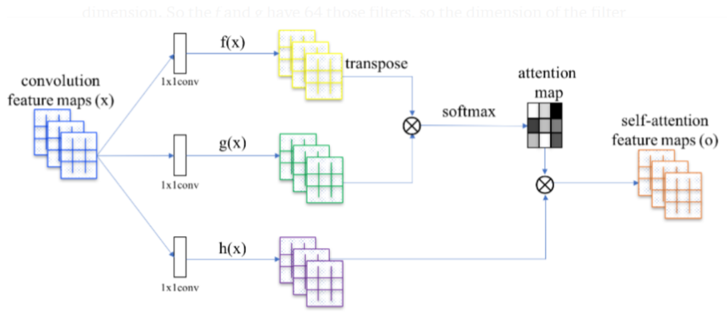

Figure 6: The Self-Attention mechanism [25]

In the case of solar panels in aerial images, self-attention might also improve the GANs ability to identify the importance of the solar panels in the images, as in some images the solar panel covers a small fraction of the image. In this way, we would like to decrease the risk of generating positives without any solar panel in the image.

3.10 Evaluation

The evaluation of GANs is more difficult than regular classifiers. For regular classifiers it is relatively simple to evaluate whether the model has the right classification compared to the true classification. In contrast, as GANs are unsupervised and generate samples which are fairly complex, it is harder to define an evaluation metric. For the evaluation of GANs, two characteristics of the images are important. First the fidelity, defined as the quality (e.g. blurriness or realism) of the images. Moreover, the diversity implies the variety of the generated images.

An example of an evaluation metric could be the pixel distance, which computes the absolute difference between the pixels of a real and fake image. However, this metric is not reliable at all, as a small shift in a feature in one image has a high impact on the absolute difference between the images, although the images might be similar. Therefore, a metric which assesses the difference between images in a more high-level way is required.

3.10.1 Fréchet Inception Distance

The Fréchet Inception Distance is an evaluation metric which computes a high-level feature distance between fakes and reals by means of extracted feature vectors and the statistics of the images. The method extracts feature vectors for the fake and real images with a pre-trained neural network and cuts off the fully connected layer. The output of the model is defined by the last pooling layer, which gives is an encoded representation of the features in the image of 2,048 dimensions. Generally, the Inception-v3 network is widely used as the state-of-the-art pre-trained network for extracting features for FID evaluation [27].

The Fréchet distance [28] computes the difference between two distributions. In case of Fréchet Inception Distance (FID), the fakes and reals are both represented as two multivariate normal distributions, with a means μ_{fake} and μ_{real} and covariance matrices Σ_{real} and Σ_{fake}. The multivariate normal Fréchet Distance is defined as:

$$\|\mu_{real} - \mu_{fake}\|^2 + \mathrm{Tr}(\Sigma_{real} + \Sigma_{fake} - 2\sqrt{\Sigma_{real}\Sigma_{fake}}) \tag{23}$$

The FID is computed by extracting n feature vectors of both the reals and the fakes, fit a multivariate normal distribution on them and compare their statistics (means and covariances) with the Fréchet distance of Equation 23. The lower the FID, the more similar the distributions are, having a smaller distance between the reals and the fakes.

3.10.2 Performance increase on classification model

A second quantitative metric to evaluate the performance of a trained GAN is assessing whether adding synthetic positives to the training set leads to a classification performance increase in the test set. This is performed by first training a classification model on the original Heerlen dataset and report its accuracy, precision and recall for the train, validation and test set. The architecture of this model is based on VGG11 by Simonyan et al. [29] and the last 4 convolutional blocks are retrained, freezing the first 4 convolutional blocks. This performance is considered the baseline performance. Afterwards, a predetermined number of generated positives (generally 5,000) is added to the training set in order to create a better balance in the dataset. The classification model with the same hyperparameter settings is trained on this new dataset, and its performance metrics are compared with the performance of the baseline model.

3.10.3 Influencing the generation of images

Clusters as GAN input
After training the GAN, the generator is able to generate synthetic images. However, we would like to have more control over the features which the trained model generates. As we make use of conditional GANs, we can control whether we want to generate an image of a certain class. However, there might also be a lot of diversity within the classes. As we are only interested in the generation of positives, we would like to have more influence on what kind of positives our GAN generates. Therefore, we test whether conditioning the GAN on the clusters instead of the classes increases its performance and controllability. Furthermore, conditioning the GAN on multiple classes could increase diversity of generation, as well as decrease the risk of mode collapse, since the generator is forced to create multiple types of samples. In order to do this, it is important that the clustering methods are able to make a division between within and between the positives and negatives. In the ideal situation, clusters have the positives separated into different groups. In this way, we can evaluate whether only generating certain types of positives leads to a higher FID and classification performance increase.

Truncation trick
As previously explained, the generator uses a noise vector as input for its image generation. The values within this vector are sampled from a Gaussian distribution, which implies that values closer to the mean of this distribution occur more frequently in the generator's input during training. In contrary, the tail values of this distribution will have a low occurrence in the generator input.

This means that after training of the GAN, we can influence the generation of images with the noise vector we impute. Using a noise vector with values close to 0 (the Gaussian mean) will give higher quality images, as the GAN has seen these values more regularly during training, and thus the generator has received more feedback about these images and has learned to generate better images of this kind. The downside is that these images generated by these values will be less diverse.

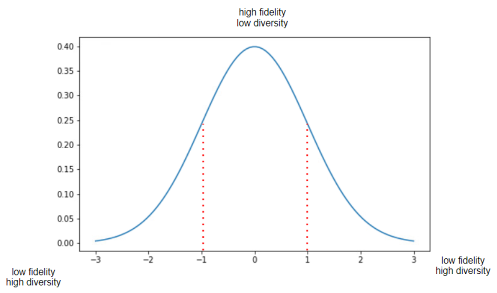

Figure 7: Truncation trick

The truncation trick makes use of this increase of quality by using the cut off values of the Gaussian distribution as an input for the trained generator. In this way, low-quality images are not generated. Previous work has proven that truncating the noise vector could lead to improved quality of generated results [30]. A possible downside could be that the FID decreases due to the reduction in diversity. However, it is a possibility that the increase in quality still leads to higher performance increase in the classification model, as the conditioning on the clusters already improves diversity of generation.

4 Results

4.1 Exploratory analysis

4.1.1 Identification of misclassified samples

In order to obtain better understanding of the model, we first validate the trained model on a subset of the test, which consists of 10,000 samples. As can be viewed in the confusion matrix in Figure 8, this subset consists of 7,202 negatives (72%) and 2,783 positives (28%), which is a more balanced ratio between positives and negatives than in the original test set (93:7). The confusion matrix uncovers that the model has difficulty classifying the positives correctly, as 1,248 out of 2,783 positives are misclassified. Nonetheless, the model seems to identify the negatives almost flawlessly, as only 14 samples of 7,202 negatives are misclassified as a positive. Therefore, the precision is really high (99.1%), while the recall is low (55.2%) This arouses the impression that the model could be biased towards the negatives, due to the class imbalance in the training set (80% negatives).

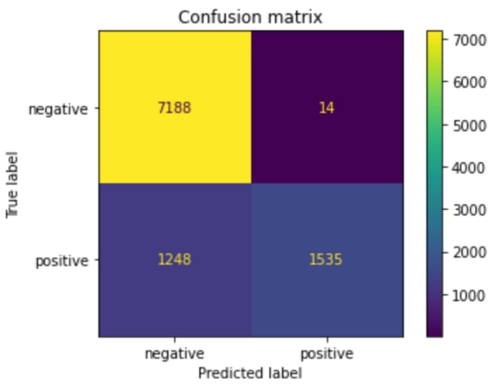

Figure 8: Confusion matrix test set (Zuid-Limburg)

The histograms of the output probabilities are shown in Figure 9 and 10. Figure 9 shows the probabilities of some samples which are correctly classified as positive. On the other hand, Figure 10 shows the probabilities of a sample of positives which are wrongly classified. It is remarkable that the probabilities are not very different between the true positives and the false negatives. It would be desirable that the probabilities of the true positives are close to 1, while the probabilities of the misclassified positives (false negatives) are close to 0.5. However, this is not the case for both of the histograms. It is suspected that these images contain features which also occur frequently in images classified as negative. Moreover, it is also possible that the model is not calibrated well, implying that the output probabilities of the network are not representative of the true likelihood of correctness [31]. Some example images of the train and test set are shown in Figures 11 and 12.

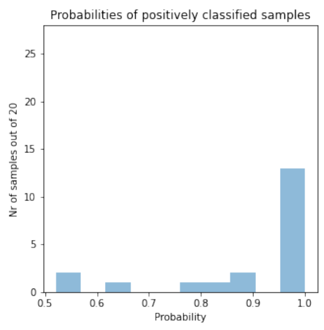

Figure 9: Histogram true positives

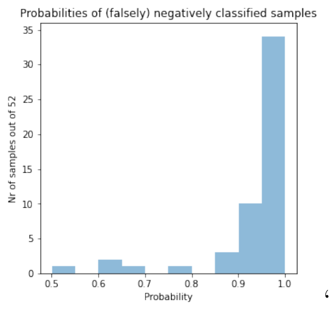

Figure 10: Histogram false negatives

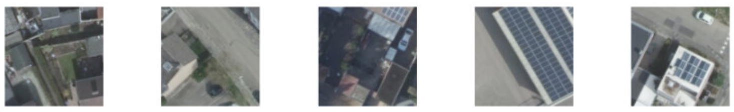
Figure 11: Sample of train set

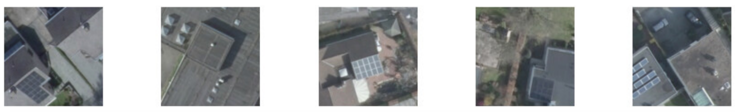
Figure 12: Sample of test set

4.1.2 t-SNE

As explained in Section 3.4, t-SNE can be used as a dimensionality reduction technique, in order to visualize the distribution of a dataset in fewer dimensions. As t-SNE is computationally expensive, a common method is to first perform Principal Component Analysis (PCA) on the dataset as a first dimensionality reduction method. Afterwards, t-SNE is performed on the obtained dimensions (mostly 4 or 5). Nonetheless, as explained in Secion 3.4, PCA is not suitable for very high-dimensional data. Therefore, instead of PCA, the feature vectors obtained with the pre-trained deep learning model are used as input for the t-SNE. The feature vectors encoded the features within 12,800 dimensions, instead of the original $200 \times 200 \times 3 = 120{,}000$ dimensions.

The output of t-SNE is highly dependent on the perplexity, which is generally a value between 5 and 50. Intuitively, the perplexity is a guess about the number of near neighbors a data point has. There is no rule of thumb of defining this parameter, thus experimenting with different perplexity values is desired. Therefore, perplexity values 10, 20, 30, 40, 50 and 60 are compared for both train and test set. The results for perplexity value 30 for a subset of 10,000 images are shown in Figure 13 and 14. The training plot shows that there is a division between the positive and negative samples based on the feature vectors, although there is some overlap between some samples. The same applies for the test set. Intuitively this makes sense, as the solar panel only covers a small part of the image. This could imply that some positives and negatives have very similar backgrounds and thus similar features, making it more difficult for the model to identify positives.

4.1.3 Clustering on extracted feature embeddings

Similar to t-SNE, the clustering is performed on the feature vectors, which are obtained by using the pre-trained model. In the ideal situation, the clustering methods are able to segregate the images not only on the basis of their class, but also find different categories of images within the class. For example, finding different sizes of solar panels within the positive class or different types of backgrounds. Two types of clustering methods are used: KMeans and hierarchical clustering.

KMeans
As mentioned in Section 3.3.1, KMeans is a clustering method which can only be

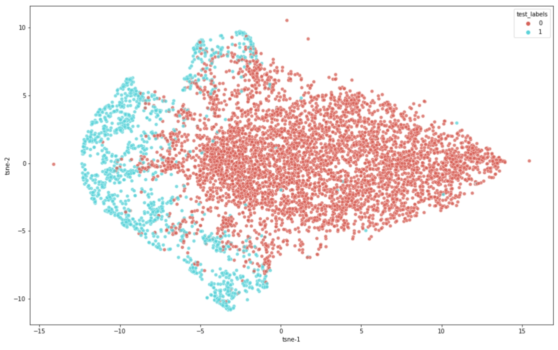

Figure 13: t-SNE train set

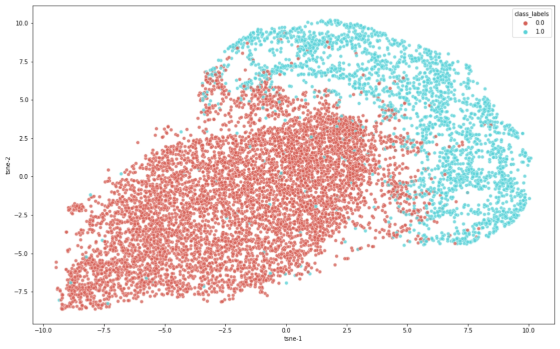

Figure 14: t-SNE test set

computed with the Euclidean distance. The main drawback of KMeans is that the number of clusters k has to be defined beforehand. Nonetheless, computing multiple KMeans clusterings with a different k can give more understanding in the influence of the number of clusters. For every clustering, the sum of squared error can be computed and shown in a plot, which is shown in Figure 15. Usually, the bending point in the plot defines the optimal k. Nevertheless, there is no clear bending point, instead the plot indicates that there is a (close to) negative linear relation between k and performance. The lower the k, the lower the error, which intuitively makes sense. The more clusters we can group the data into, the smaller the error, as the distance between the points within a cluster will become smaller. Additionally, the cluster analysis for 80 clusters shows that there are many clusters with one sample. Only 4 clusters contain more than 1 sample, whereas the other 76 clusters encompass one single data point. If there are many cluster with one sample, for these clusters the single cluster point is also the centroid, making the distance zero. Therefore, the sum of squares of the clustering will obviously decrease. This problem also occurs with a k between 10 and 80. Therefore, further experimentation shows that a k around 4-8 would be a suitable number of clusters.

The KMeans cluster analysis is performed for a subsample of 10,000 samples for both train and test set. This size of subset is chosen because the clustering labels are also used as input for the GAN. As training GANs is computationally expensive, the generation of images is performed on a subset instead of the

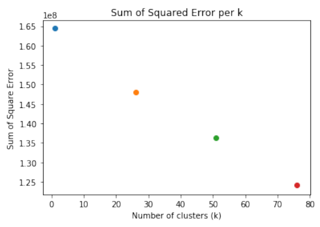

Figure 15: Error per number of clusters

complete dataset. All positives are included in both subsets, having a training set with 4,672 (46.7%) positives and 5,328 (53.3%) negatives and a test set with 2,813 positives (28.1%) and 7,187 negatives (71.9%). As explained in Section 3.3.4, the Silhouette score is bounded between -1 and $+1$, and a score of -1 implies a very bad clustering, while a score of $+1$ is optimal. The Calinski-Harabasz (CH) score is not bounded; the higher the score, the better. In Table 1 the Silhouette score and CH score for KMeans with $k = 5$ are shown. We can see that for the CH score, the train set has a slightly better score than the test set. However, the opposite applies for the Silhouette score.

dataset	silhouette score	calinski-harabasz score
train set	0.51	1675.7
test set	0.67	1667.4

Table 1: Evaluation metrics KMeans (5 clusters)

In Table 2 and 3, some extra information about the clustering with regard to the classes is shown. We can see that for the train set, there are four clusters (0, 2, 3, and 4) which consist of more than 99% positives. Cluster 1 contains the most samples (71% of the samples) and consists of mostly negatives, although it still has 25% positives. Furthermore, the proportion of samples out of all samples in the cluster which are wrongly classified by the classification model are shown in the table. We can see that cluster 1 has the higher percentage of samples wrongly classified by the trained classification model. The results for the test set show a similar pattern: in four clusters, the positives dominate, while in the remaining cluster almost all negatives are gathered. In this case, this cluster consists of 85% of the samples and has relatively the lowest proportion of falsely classified samples compared to the other clusters. This is different from the train set. However, the proportion of positives in this cluster of the test set is 10% lower than in the training set. As we have seen in the test set confusion matrix in Figure 8, 98% of the negatively classified samples are positive. Therefore, it makes sense that the percentage of misclassified samples for this cluster is higher in the test set than in the train set.

Hierarchical clustering
As hierarchical clustering has several parameters to tune (e.g. number of clusters, affinity, linkage type), multiple experiments are performed, comparing its performance metrics. The most important results are shown in Table 4. The table shows that for the training set, Ward linkage and the Euclidean distance

cluster	number of samples	number of positives	number of falsely classified samples	cluster label
0	739	736 (99.6%)	41 (5.5%)	positive
1	7098	1775 (25.0%)	870 (12.3%)	negative
2	678	677 (99.9%)	66 (9.7%)	positive
3	781	781 (100.0%)	61 (7.8%)	positive
4	704	703 (99.9%)	69 (9.8%)	positive

Table 2: Samples of KMeans train set

cluster	number of samples	number of positives	number of falsely classified samples	cluster label
0	8524	1331 (15.6%)	978 (11.5%)	negative
1	403	401 (99.5%)	66 (16.4%)	positive
2	344	343 (99.7%)	80 (23.3%)	positive
3	344	340 (98.8%)	78 (22.7%)	positive
4	370	368 (99.5%)	60 (16.2%)	positive

Table 3: Samples of KMeans test set

has the best Silhouette score (closest to +1) and best CH score.
Hierarchical clustering with Single link and Cosine similarity seemed to perform very poorly. Moreover, there did not exist clear clusters, as there emerged one cluster with more than 1 sample and 4 clusters with a single sample.

dataset	linkage	distance measure	silhouette score	calinski-harabasz score
train set	ward	euclidean	0.43	1207.40
	single-link	cosine	-0.47	0.14
	complete-link	cosine	0.33	942.73
test set	ward	euclidean	0.57	1267.4
	single-link	cosine	-0.21	0.81
	complete-link	cosine	0.35	1074.80

Table 4: Evaluation metrics hierarchical clustering

Table 5 and 6 show more information about the clustering with Ward linkage and Euclidean distance. A similar pattern as for KMeans is identified, with multiple 'positive' clusters and one large cluster with most of the negatives.

4.1.4 Analysis of clustering

We can derive from Table 1 that KMeans has the best clustering performance on the train and test set. In Appendix A a sample of images per cluster is shown.

cluster	number of samples	number of positives	number of falsely classified samples	cluster classification
0	956	953 (99.7%)	86 (9.0%)	positive
1	6854	1551 (22.6%)	709 (10.3%)	negative
2	743	741 (99.7%)	82 (11.0%)	positive
3	766	764 (99.7%)	95 (12.4%)	positive
4	681	663 (97.4%)	135 (20.0%)	positive

Table 5: Samples of Hierarchical training set

cluster	number of samples	number of positives	number of falsely classified samples	cluster classification
0	8544	1363 (16.0%)	953 (11.2%)	negative
1	423	415 (98.1%)	82 (19.4%)	positive
2	481	469 (97.5%)	130 (27.0%)	negative
3	245	244 (99.6%)	33 (13.5%)	negative
4	292	292 (100%)	64 (21.9%)	negative

Table 6: Samples of Hierarchical test set

Inspecting the clusters shows there are negligible differences between the positive clusters, as the distinction is hard to capture with the eye. Cluster 0 seems to have smaller solar panels and more monotone colors, while cluster 2 seems to have clearer solar panels and more vivid colors. Cluster 3 and 4 are more alike and consist of a mix of large solar panels and colorful backgrounds.

Since we would like to use the clusters as input for the GAN, it might be interesting to focus on the 15-20% of positives within the cluster with mostly negatives. It could be that these positives are the samples which the classification model tends to misclassify as negative. A t-SNE visualized with the KMeans cluster labels and class labels is shown in Figure 16 and 17. The visualization of the train set shows that the clustering is able to segregate between the positives and the negatives, but that the cluster which would be expected to mainly consist of negatives (the green colour), also catches a lot of positives which are expected to be in the other clusters. This can be derived from the many green samples scattered between the four other clusters. The same applies for the test set, where this scattering is even more severe. A lot of red samples tend to be dispersed between the other colours. It becomes clear that for the test set, the clustering has more difficulty catching the difference in features between the positives and negatives.

We could zoom in on the division of samples within the large 'negative' cluster. The t-SNEs for this specific cluster in the train and test set are shown in Figure 18 and 19. For both clusters, the t-SNE shows that there is a division between the positives and negatives, although there are a few positive samples scattered within the negatives.

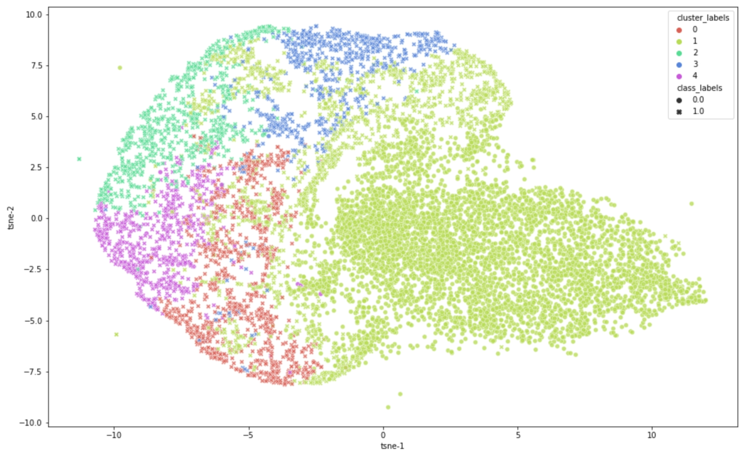

Figure 16: t-SNE train set

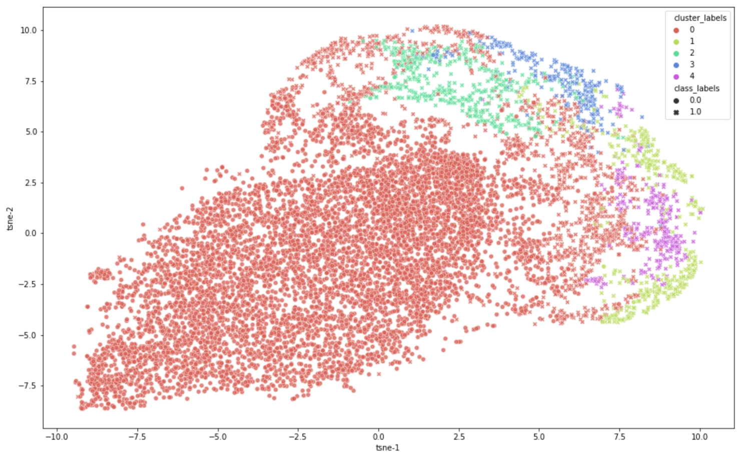

Figure 17: t-SNE test set

Having these positives in this negative cluster implies that when we train a GAN with the train set on these 5 clusters, the positives in this 'negative' cluster will not be generated, as we only generate samples from the 4 'positive' clusters. Therefore, it could be a solution to split the negatives from this cluster into cluster 1 (the split positives) and a new cluster 5 (the negatives). We can now analyze the percentage of positives per cluster. As we would eventually like to increase the performance on the positives of the classification model with the generated images, the percentage of positives within the cluster which are misclassified by the classification model is also shown. This can give an indication which cluster contains the positives which are the hardest to classify.

As we can see in this table, cluster 0 until 4 are the 'positive' clusters, with cluster 1 the positives of the 'negative' cluster. The newly added cluster 5 is the 'negative' cluster. It shows that the positives in this originally 'negative' cluster are the positives that are mostly misclassified by the classification model (almost 50%). Therefore, it might be interesting to also train the GAN on these positives

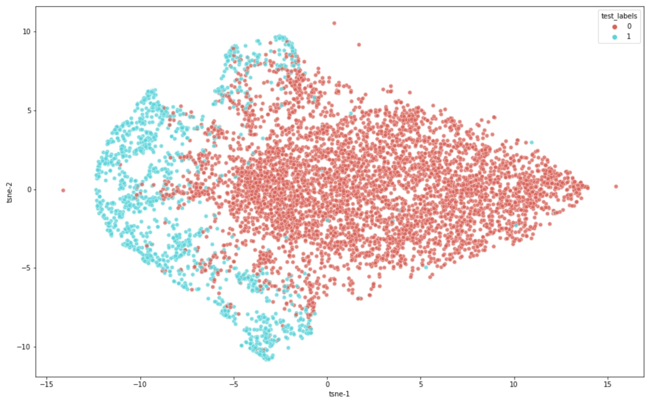

Figure 18: t-SNE 'negative' cluster in train set

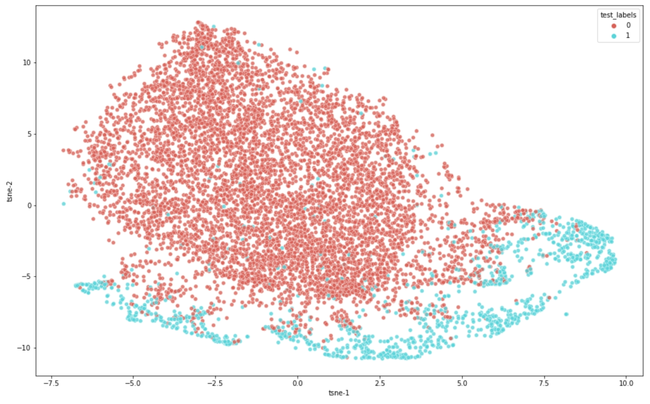

Figure 19: t-SNE 'negative' cluster in test set

cluster	% of positives in cluster	% of positives that is misclassified
0	99.6%	5.5%
1	100%	47.8%
2	99.9%	9.7%
3	100%	7.8%
4	99.9%	9.8%
5	0%	-

Table 7: Samples of KMeans train set

clustered in the negative group, since adding more of this kind of positives to the training set to train the classification model on might thus possibly increase the performance. However, the t-SNE of these 6 clusters in Figure 20 shows that simply splitting these positives into a new cluster might also lead to more bias in the GAN, as the figure shows that these positives are scattered across the other positive clusters and might actually not belong in one cluster.

Therefore, we will compare the GAN performances of four analyses with each

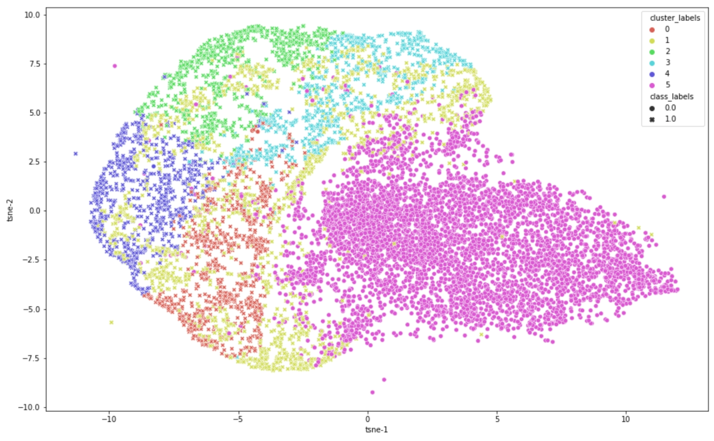

Figure 20: t-SNE with positives split from 'negative' cluster

other: training the GAN with the 2 original classes as input, the original 5 clusters, the 6 clusters with the positives split and 5 clusters with the positives omitted from the whole input dataset.

4.2 Generation of images

We will test three different types of GANs: the conditional vanilla GAN, the conditional Wasserstein GAN with gradient penalty and the Self-Attention GAN. We will also condition the GANs on two types of classes: the original classes and the classes generated with clustering. For all training, a GPU (Tesla-V100-SXM2) of 32 GB and CUDA version 11.1 is used.

Performance evaluation
As mentioned in Section 3.10, we evaluate the performance of the GANs both quantitatively and qualitatively. Qualitatively, we observe the diversity and fidelity/quality of the generated image, which is a subjective evaluation metric. Quantitatively, we make use of the Fréchet Inception Distance and the classification performance on the test sets after adding synthetic samples.

For this classification performance metric, we split the original dataset in a train, validation and test set in the ratio (70:20:10). It is made sure that in all datasets, the percentage of positives is 20%. The train set consists of 16,692 samples, the validation set of 4,769 samples and the test set of 2,386 samples. We avoided randomly splitting the the datasets into train and test set during the training runs, as this would inject randomness into the comparison. Hence, we have two test sets: a test set taken from the Heerlen training set and the Zuid-Limburg test set. The Zuid-Limburg test set in the test set which eventually defines the performance metric.

We first trained a VGG11 model on the whole training set, freezing the weights of the first 4 convolutional blocks and re-training the rest of the layers. We used the Heerlen test set to evaluate the baseline model. We report the accuracy, precision and recall for the train, validation and 2 test sets and use this as our baseline. In our case, the accuracy is less relevant, as this metric is already high for the baseline model. This is due to the model almost flawlessly predicting the negatives. As the test set consists of 93% negatives, this leads to a high accuracy, even though half of the positives are misclassified. Additionally, the precision is already high for the baseline model as well. Therefore, we will merely focus on the recall on the Zuid-Limburg test set as our performance metric.

When evaluating a GAN model, we add the generated samples to the train and validation set, and train the classification model with the same hyperparameters. A number of 5,000 generated samples is chosen, adding 4,000 samples to the train set and 1,000 samples to the validation set. In this way, the percentage of positives in the train set increases from 19.59% out of 16,692 train samples to 35.13% out of 20,692 train samples. This implies the new train set consists of 3,270 real positives and 4,000 fake positives. Adding more synthetic samples to the train set would mean that there are proportionally a lot more synthetic positives than real positives in the train, which could make the classification model focus more on the fake positives. Furthermore, although most of the generated images contain solar panels, there is not 100% guarantee that every generated positive image contains a (clear) solar panel. This could eventually inject bias in the classification. Thus, in the experiments below, we will show the performance metrics consistently for 5,000 added samples.

We run all classification models three times and denote the average performance. We compare the classification evaluation metrics with the baseline model, in order to perceive which GAN model is better and whether the classification performance increases.

After training a GAN, we can control the generation of images by experimenting with the truncation trick or only generating a subset of clusters. In this way, we can choose to e.g. only generate high-quality images (but with a low diversity) or only generating one type of solar images. We can compare these different images with the above mentioned classification performance metric.

4.2.1 Vanilla conditional GAN

The best vanilla GAN architecture
The simplest model is the vanilla conditional Deep Convolutional GAN (cDC-GAN), which makes use of the binary cross-entropy loss. The original image size of 200x200 showed not to be able to generate realistic images. Experimenting with resizing the input image size showed that 96x96 images gave valuable results without decreasing the quality of the image too much, and simultaneously drastically decreasing the training time.

As it takes at least 5 hours to train a GAN which generates valuable output, while the discriminator and generator training losses are not always an indication of good image generation, it is difficult to perform a grid search or early stopping on the hyperparameter tuning. Therefore, we started with experimenting with several parameter settings and evaluating the output images on the basis of the fidelity and diversity. A variety in generator and discriminator architectures, batch size, learning rate, optimizer, noise dimension and weight initialization were explored. During this experimentation phase, the 2 original classes were used as input, instead of the clusters.

As mentioned before, training an unconditional GAN may not be possible in our case, as we might not have sufficient positives (4,672) to train an unconditional model. We tested this by training an unconditional vanilla GAN on the positives, instead of training a conditional GAN on both positives and negatives. Training a GAN on only the positives showed that it was not able to generate realistic output. Experimenting with the same unconditional model on all labeled data (23,847 samples) showed that the model was able to generate relatively realistic output. This confirmed the suspicion that unconditional generation on positives-only does not work due to the lack of samples. Hence, we decided to focus on the conditional models.

The best model has a generator with 6 convolutional blocks and a generator with 4 convolutional blocks. The optimal batch size is 256, noise dimension 64, learning rate 0.0002 and optimizer Adam. The best weight initialization is normal initialization for the convolutional and batch normalization layers. The model was trained for 600-2000 epochs. This model was optimal for both the 2 classes and the 5 clusters as conditioning input. A visual representation of the generator and discriminator architectures and blocks are shown in Figure 21, 22, 23 and 24. For the generator, the input and output channels, kernel size, stride and padding are chosen in such a way that the (64 + n_classes, 1, 1) noise vector is upsampled to a 3x96x96 image. As we want to upsample the noise vector, 2D transposed convolution layers are used. For the discriminator, we want to downsample from an image to a scalar prediction. Therefore, we want a spatial convolution over the images, using 2D convolution layers.

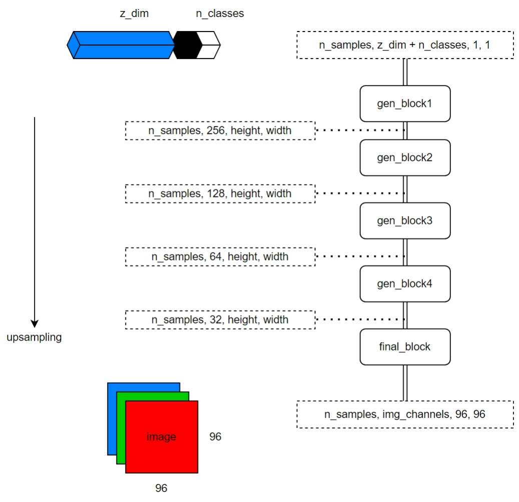
z_dim
n_classes
n_samples, z_dim + n_classes, 1, 1
gen_block1
n_samples, 256, height, width
gen_block2
n_samples, 128, height, width
gen_block3
n_samples, 64, height, width
gen_block4
n_samples, 32, height, width
upsampling
final_block
n_samples, img_channels, 96, 96
image
96
96

Figure 21: Architecture of generator

The best GAN input
As mentioned, we want to make use of the generated clusters of Section 4.1.3 as conditioning of the GAN instead of the class. The results in Section 4.1.3 have shown that KMeans has the best Silhouette score and Calinski-Harabasz score, thus we will make use of these generated clusters for the GAN. However, as we have seen in Section 4.1.4, we could choose to split the positives in the

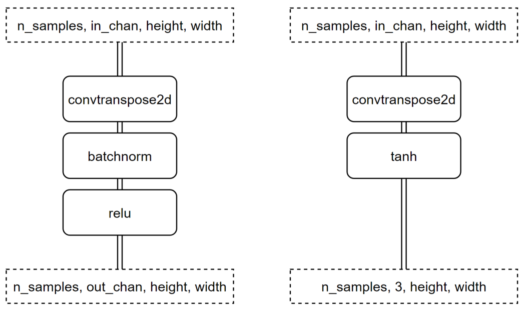
n_samples, in_chan, height, width
convtranspose2d
batchnorm
relu
n_samples, out_chan, height, width
n_samples, in_chan, height, width
convtranspose2d
tanh
n_samples, 3, height, width

Figure 22: Architecture of generator blocks

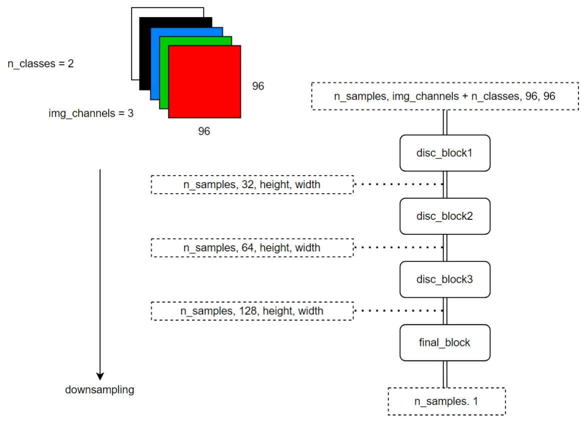
n_classes = 2
img_channels = 3
96
96
n_samples, img_channels + n_classes, 96, 96
disc_block1
n_samples, 32, height, width
disc_block2
n_samples, 64, height, width
disc_block3
n_samples, 128, height, width
final_block
downsampling
n_samples. 1

Figure 23: Architecture of discriminator

'negative' cluster, although this might lead to more bias in the GAN. Therefore, we test four different GAN inputs: the original 2 classes, the 5 original clusters, 5 clusters with the positives in the 'negative' cluster completely omitted from the dataset, and 6 clusters with the positives in the 'negative' cluster separated. The results are shown in Table 8. The results show that the 6 clusters with the split positives gives the highest FID, while also generating images with comparably high fidelity and diversity. The value of fidelity and diversity are subjective and evaluated on the basis of the generated images. Also, Table 8 shows that the model with 6 clusters leads to the highest increase in recall compared to the baseline, although decreasing the precision. We see that in all models, the increase in recall is at the expense of the precision. We also see that training the GAN on the classes decreases the diversity, which could be due to the generator only being forced to generate 2 types of images/classes, while the other models are forced to generate multiple types/classes. Completely omitting the 25% of positives in the 'negative' cluster decreases the FID even more, which could be due to not having enough data.

Varying the discriminator output

The discriminator is expected to give a scalar output for every image it receives,

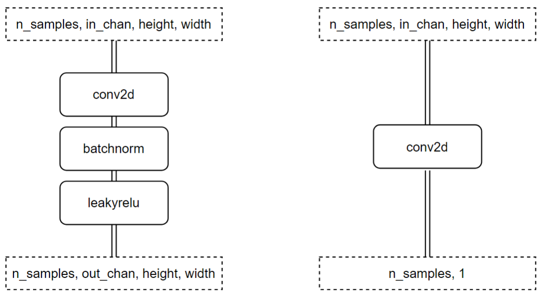
n_samples, in_chan, height, width
conv2d
batchnorm
leakyrelu
n_samples, out_chan, height, width
n_samples, in_chan, height, width
conv2d
n_samples, 1

Figure 24: Architecture of discriminator blocks

input	FID	fidelity	diversity
classes	251.6	high	low
5 original clusters	220.6	high	high
5 clusters and positives omitted	292.6	low	low
6 clusters with positives separate	**198.85**	high	high

Table 8: Performance comparison of GAN input

input	test accuracy	test recall	test precision
baseline	0.8719	0.5710	0.9481
5 clusters and positives omitted	0.8736	0.5972	0.9227
5 original clusters	0.8731	0.5971	0.9182
classes	0.8728	0.5918	0.9237
6 clusters with positives separate	0.8747	**0.6001**	0.9208

Table 9: Classification performance comparison of different inputs

evaluating the probability that the image is real or fake. Nonetheless, it could be interesting to experiment with changing the convolutional layers in the discriminator such that the output per image is not a scalar, but a multi-dimensional vector. The reasoning behind this is that this could impose an additional challenge to the discriminator, which has a relatively easy task compared to the generator. For a x-dimensional output, the discriminator now has to compare x values to the x ones (real) or zeros (fake), instead of one scalar value per image. The results for a discriminator architecture which gives 8-dimensional and 16-dimensional outputs for an image prediction, compared to a scalar dimension output are shown in Table 10 and 11. Table 10 shows the FID, diversity and fidelity of the generated images, while Table 11 compares the accuracy, recall and precision on the Zuid-Limburg test set after re-training the classification model with the added generated images.

output	FID	diversity	fidelity
scalar	198.85	high	high
8-dimensional	**185.58**	high	high
16-dimensional	227.86	high	high

Table 10: Image comparison of discriminator output

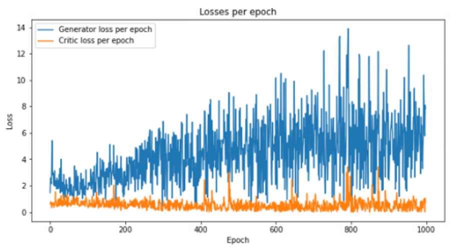

Figure 25: Losses (8-dimensional)

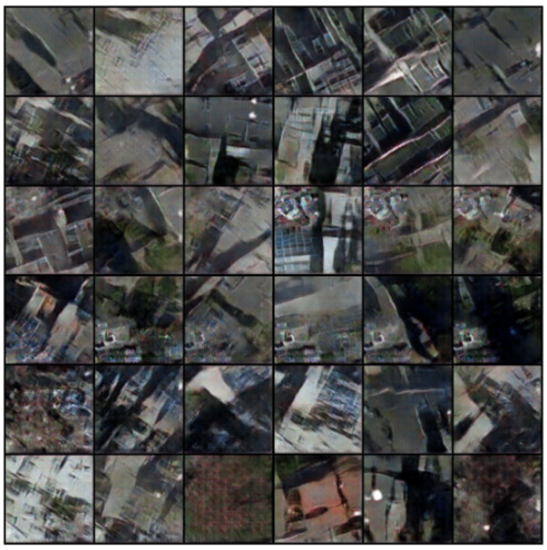

Figure 26: Generated images with 8-dimensional discriminator output

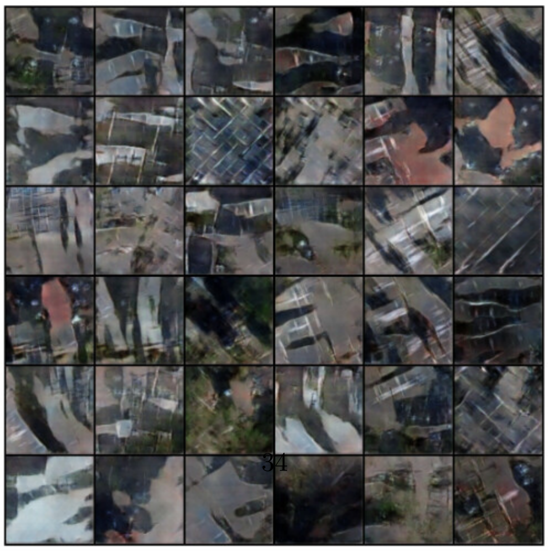

Figure 27: Generated images with 1-dimensional discriminator output

Table 10 shows that the model with 8-dimensional output has a better FID (185.58) than the scalar output. The losses of the generator and discriminator are shown in Figure 25. We can see that the generator loss is relatively unstable compared to the discriminator loss. Figure 26 and 27 show a sample of generated images by the model with 8-dimensional and scalar output. The images in the first row of the grid represent cluster 0, the second row represents cluster 1, and so on. We can see that both models with 1-dimensional and 8-dimensional discriminator outputs are able to generate outputs which resemble aerial images. In Table 11 the classification performance on the test set is shown for both generated models, compared to the baseline classification without generated images. The table shows that the 8-dimensional output for the discriminator shows the highest recall increase. Nonetheless, this again is at the expense of the precision, which decreases with 2.73%.

output	test accuracy	test recall	test precision
baseline	0.8719	0.5710	0.9481
scalar output	0.8747	0.6001	0.9208
8-dimensional output	0.8747	**0.6037**	0.9180
16-dimensional output	0.8736	0.5972	0.9208

Table 11: Classification performance comparison of discriminator output

Controlling the generation
After training the GAN, we could have more control over the image generation by influencing the input of the trained generator. The first way to do this is the truncation trick, where we sample the noise vector from a truncated Gaussian distribution. A truncation of 0 means that the distribution of the noise is completely truncated except for the mean, which leads to a very high fidelity but a low diversity. A truncation of 1 implies that there is little truncation and thus we will have diverse outputs. Furthermore, after training the GAN on the clusters, we could choose a subset which the GAN generates. In Table 12 we have shown some of the experiments conducted with both control methods, performed with the GAN trained on the 6 clusters. In the first line, the test accuracy, recall and precision of the training set without any synthetic images is shown. The rest of the table represents some of the best performance results when the synthetic images are added to the training set, based on controlled input. We can see that especially a truncation of 0.7 improves the recall compared to original training set, although the precision decreases.

truncation	cluster input	test accuracy	test recall	test precision
baseline	-	0.8719	0.5710	0.9481
0.7	0, 2, 3, 4	0.8753	**0.6096**	0.9224
0.4	0, 2, 3, 4	0.8746	0.5986	0.9240
none	0, 1, 2	0.8736	0.5972	0.9208
none	0, 3	0.8737	0.5954	0.9236

Table 12: Classification performance comparison of different inputs

4.2.2 Wasserstein conditional GAN

The second type of GAN we will evaluate is the conditional Deep Convolutional WGAN. The WGAN is supposed to decrease the risk of mode collapse and improve stability during training. However, in order to ensure 1-Lipschitz continuity of the critic (previously called discriminator), we can impose gradient penalty, spectral normalization or weight clipping. Therefore, we will evaluate several variations of the WGAN: WGAN without any 1-L enforcement, WGAN with gradient penalty, WGAN with spectral normalization and WGAN with gradient clipping. We will also visualize the gradient norm of the critic, in order to check whether the average gradient norm is smaller than 1.

For the comparison of the different 1-L enforcements, we will make use of the 6 clusters (with the split positives) as GAN input. The generator and critic architecture both consist of 5 blocks, which is slightly different from the vanilla GAN, which had a discriminator with 4 blocks. The models are run for a varying number of epochs between 600-2000, with noise dimension 64, batch size 256 and learning rate 0.0002. Experimenting with different hyperparameters has shown to worsen the results.

Wasserstein GAN with gradient penalty
In Table 13, the results for some of the WGAN models with gradient penalty are shown. The column *penalty weight* states whether and what weight value for the gradient penalty is used. *Critic repeats* implies how many times the critic is updated before updating the generator. This is common for a WGAN with gradient penalty, as we want to avoid that the generator overpowers the critic due to not having a gradient penalty. We also experiment with both optimizers Adam and RMSprop, as the authors of the vanilla WGAN paper [11] state that a momentum-based optimizer such as Adam may destabilize training. However, the authors of the WGAN-GP paper [12] found that for the WGAN-GP optimizer Adam outperforms RMSprop, therefore we experiment with both.

P1 is the WGAN without any penalty or 1-L enforcement. It shows good results until the 250th epoch, after which the model destabilizes and fails to generate good results. Hence, for its FID calculation, the model at the 250th epoch is chosen, which shows a relatively high diversity and medium realistic results. The rest of the models make use of gradient penalty. The WGAN with penalty weight 10, 2 updates for the critic before updating the generator and optimizer RMSprop shows the best FID (model P2). The results also show that updating the critic too many times (P5) worsens the performance drastically, as the critic might overpower the generator.

model name	penalty weight	critic repeats	optimizer	FID	fidelity	diversity
P1	-	1	adam	232.6	medium	high
P2	10	2	RMSprop	**193.51**	high	high
P3	15	1	RMSprop	200.88	medium	high
P4	10	1	adam	251.3	medium	high
P5	10	5	adam	332.1	low	low

Table 13: WGAN comparison with gradient penalty

In Figures 28 and 29 the losses and average gradient norm in a batch are visualized for the best gradient penalty model (P2). The loss shows that the critic loss keeps fluctuating around zero and is very unstable. The generator loss gradually increases. Also, Figure 29 shows that the gradient penalty does not ensure that the gradient norm is below 1.

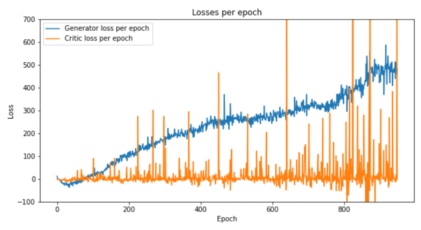

Figure 28: Losses (P2)

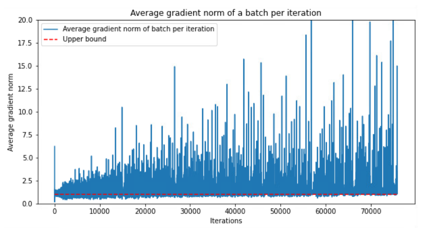

Figure 29: Gradient norm (P2)

Figure 30 shows a sample of generated images by model P2 for the 6 clusters. We can see that especially the 4th clusters (the 4th row in the grid) shows realistic solar panels.

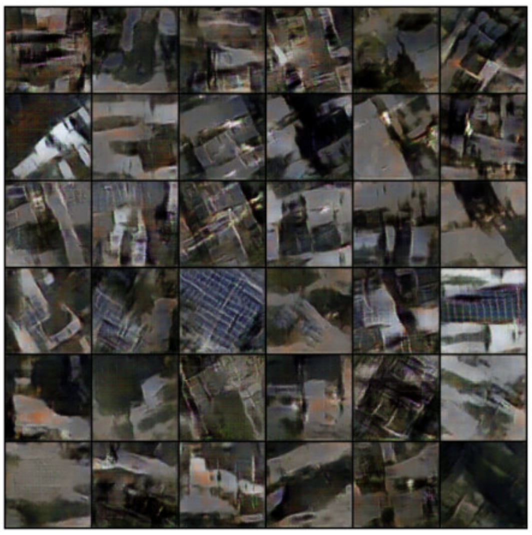

Figure 30: Generated images (P2)

Table 14 shows the performance results for the test set for the two best models from Table 13. It shows that although model P2 has the lowest FID and the most realistic images, the model without any 1-L enforcement (P1) has slightly better performance on the test set. Nonetheless, both models outperform the baseline model with roughly 3.52% and 2.90% in recall.

Wasserstein GAN with spectral normalization
When spectral normalization is used, it is only imposed on the convolutional layers of the critic, as we would only want the gradient norm of the critic to be below 1. Also, no batch normalization is used for the critic blocks, as stated in the paper which introduced spectral normalization in GANs [24]. However,

model name	test accuracy	test recall	test precision
baseline	0.8719	0.5710	0.9481
P1	0.8761	**0.6062**	0.9219
P2	0.8741	0.6000	0.9201
P3	0.8735	0.5995	0.9175

Table 14: Test performance increase comparison gradient penalty

this model (S1 in Table 15) showed no realistic outputs, while the generator loss was unstable compared to the critic loss. Additionally, the 1-L continuity requirement was not met, as the average gradient was mostly larger than 1. Model S2 imposes spectral normalization on the generator as well (as performed in the BigGAN paper [32]), while S3 updates the generator multiple times before updating the critic. Both models did not improve the outputs, meaning that in our case spectral normalization is not a stable method for the 1-Lipschitz contuinity requirement.

model name	conv generator	conv critic	batchnorm	FID	fidelity	diversity
S1	-	spectral	only generator	304.1	low	low
S2	spectral	spectral	none	295.4	low	low
S3	-	spectral	both generator and critic	289.5	low	low

Table 15: WGAN comparison with spectral normalization

Wasserstein GAN with weight clipping

For the WGAN with weight clipping, various clip values were chosen. The results of some models with various parameters are shown in Table 16. Model C1 makes use of the default optimal parameter settings as used in previous models. For model C2, the dimensionality of the critic was set to 8, as this has shown good results for the vanilla GAN. For model C3, a larger batch size of 1024 and a smaller learning rate of 0.0001 was chosen. These models are shown to demonstrate that a small change in the parameter settings can have a large influence on the quality of the results, as both models perform worse. The rest of the models experiment with the default parameter settings of the best model, but with a larger clip value than 0.01. The model with a clip value of 0.01 showed the best FID value. Inspecting the progress of the loss in Figure 31 shows that both losses are really stable around zero and gradually converge. Moreover, the plot of the gradient norm in Figure 32 shows that the 1-L requirement is met. A sample of generated images for C1 and C5 is shown in Figure 33 and 34.

model name	clip value	adaptation	FID	fidelity	diversity
C1	0.01	-	**181.82**	high	high
C2	0.01	critic output	254.56	medium	medium
C3	0.01	batch size	293.84	low	low
C4	0.02	-	207.04	high	high
C5	0.03	-	210.06	high	high

Table 16: WGAN comparison with weight clipping

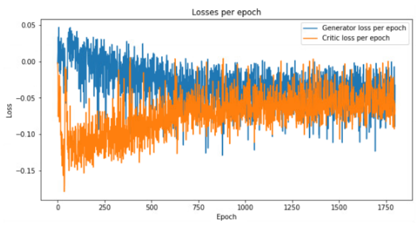

Figure 31: Losses (C1)

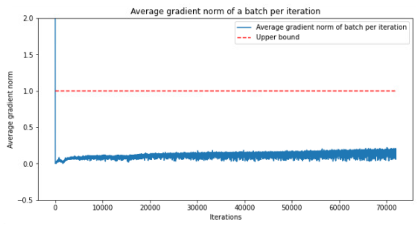

Figure 32: Gradient norm (C1)

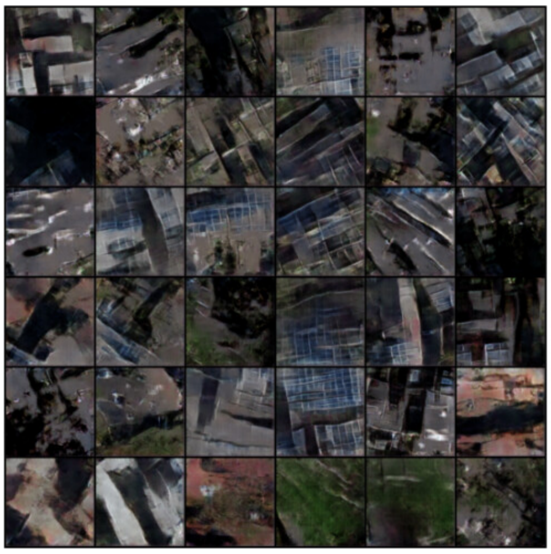

Figure 33: Generated images (C1)

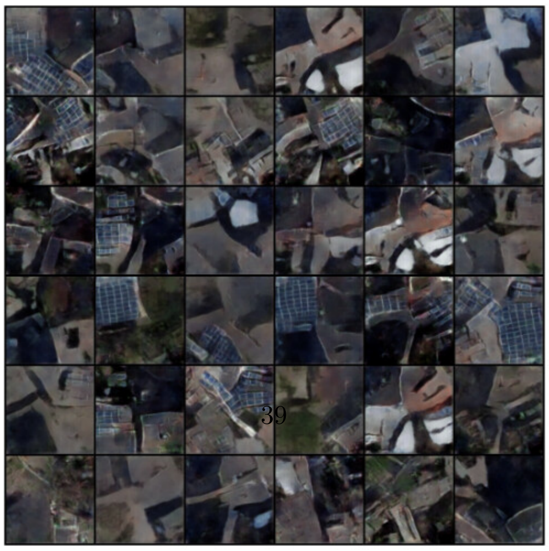

Figure 34: Generated images (C5)

The results for the best models C1, C4 and C5 are shown in Table 17. We can see that model C1 actually increases the test recall with 6.14%, while the other models improve the recall with roughly 3.10% and 4.34%.

model name	test accuracy	test recall	test precision
baseline	0.8719	0.5710	0.9481
C1	0.8864	**0.6324**	0.9269
C4	0.8735	0.6020	0.9240
C5	0.8792	0.6144	0.9211

Table 17: Test performance increase comparison weight clipping

Controlling the generation

With the best WGAN model with weight clipping (C1), we experimented with re-training the classification model on a subset of positives or images generated with a truncated noise vector. The results are shown in Table 18.

truncation	cluster input	test accuracy	test recall	test precision
baseline		0.8719	0.5710	0.9481
-	2, 3	0.8802	**0.6329**	0.9273
-	0, 4	0.8798	0.6307	0.9252
0.7	-	0.8753	0.6184	0.9269
1.2	-	0.8728	0.6172	0.9250

Table 18: Classification performance comparison of different inputs

The table shows that the truncation trick does not improve the test set performance compared to untruncated inputs. This can already be derived from the visualizing the truncated samples, which shows that the samples decrease in diversity excessively and also become less realistic. However, only using cluster 2 and 3 instead of all 4 positive clusters improves the recall, although the increase compared to the complete set of positives is negligible.

4.2.3 Self-Attention GAN

The goal of the Self-Attention GAN is to improve discovering the relationship between different parts of the image. Especially in the case of aerial images, it is useful if the model identifies the importance of the solar panel in the image, since it covers a small part of the image.

The Self-Attention (SA) GAN paper states that spectral normalization in both the generator and discriminator improves the performance of the SA-GAN [25]. Moreover, the authors of the BigGAN [32], which also makes use of Self-Attention, also make use of spectral normalization in both networks. However, analogous to the results for the WGAN, spectral normalization completely destabilized the GAN. Moreover, both WGAN with weight clipping and WGAN with gradient penalty did not show stable results. However, the SA-GAN with BCE loss did showed relatively stable results.

model name	FID	fidelity	diversity
A1	**209.68**	medium	high
A2	240.97	low	high

Table 19: SA-GAN comparison

In Table 19, the results for the two best models are shown. Both models use the BCE loss. In model A1, both the generator and discriminator consist of 5 convolutional blocks, while the attention block is placed after the second block. In model A2, an extra attention block is placed after the second convolutional block. Table 19 shows that increasing the number of attention blocks in the network decreases the FID. Putting an attention block after all of the 5 convolutional blocks did even further distort learning.

model name	test accuracy	test recall	test precision
baseline	0.8719	0.5710	0.9481
A1	0.8750	**0.6022**	0.9213
A2	0.8739	0.5964	0.9232

Table 20: SA-GAN classification performance

Table 20 shows the classification results after adding the generated images. The results show that although not all images are completely realistic, they still improve the accuracy and recall rate with 3.12%. As can be shown in Figure 35, although some images are not completely realistic, some images contain very clear solar panels.

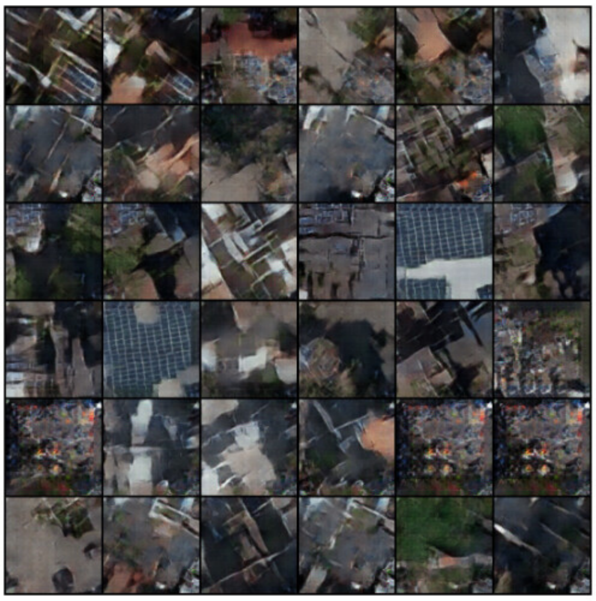

Figure 35: Generated images (A1)

5 Discussion

This research was focused on gaining more insights into the structure of the aerial image train and test set as well as adding synthetic aerial images to improve the class balance. In the exploratory analysis, we first identified that with the former classification model, the recall on the test set was 55.2%, while the precision was relatively high (99.1%). This implies that the model misclassifies almost half of the positives, but when the model classifies a sample as positive, it is mostly correct. Intuitively, we could say the model is too cautious with classifying a sample as positive.

We used t-SNE to visualize the class distribution in both datasets. KMeans and Hierarchical clustering were performed on the extracted feature embeddings to segregate the data into five clusters. Eventually, KMeans had the best Silhouette score (0.51) and Calinski-Harabasz score (1675.7), and was thus used as training input for the GANs. The t-SNE of the obtained clusters showed that the clusters are able to segregate between the classes as well as within the positive class. However, the negative cluster consisted 25% of positive samples, while from these positives 47.5% were misclassified by the classification model. Splitting these from the negative cluster into a new cluster showed more diverse GAN results, which could be due to the GAN having to generate an extra type of cluster. We also saw that omitting the positives from the negative cluster decreased the FID, fidelity and diversity. Moreover, the GAN trained on the 2 classes also had a low diversity, which could be due to not being forced to generate multiple clusters as outputs, but only 2 types of outputs. Therefore, using the clusters showed that the GAN outputs are more diverse and less prone to mode collapse.

The results of the vanilla GAN showed that a multi-dimensional output of the discriminator rather than a scalar output probability could improve the FID as well as the recall performance on the test set. This could be due to the additional challenge for the discriminator, which has a relatively easy task compared to the generator. Also, applying the truncation trick to the noise vector could further improve the recall on the test set. The best vanilla model had an FID of 198.85 and a recall rate of 0.6037, compared to a baseline recall of 0.5710.

For the Wasserstein GAN, three 1-Lipschitz continuity enforcements were compared to each other. Experimenting with spectral normalization showed no realistic results. For the WGAN with gradient penalty, the best results were obtained with optimizer RMSprop, penalty weight 10 and 2 critic updates before updating the generator. The FID was 193.51 and the test recall 0.6062. The Wasserstein with weight clipping is the best GAN model, having an FID of 181.82 and test recall of 0.6324. A clip value of 0.01 was the optimal value, while values 0.02 and 0.03 also worked, but the models were less stable. Using a subset of clusters to generate images as input for the classification model improved the recall a bit, although the increase is negligible.

The Self-Attention (SA) GAN did not give realistic results with the Wasserstein loss. Nonetheless, the SA-GAN gave good results with the BCE loss if one or two attention blocks were added after the first and second convolutional layer. The best model had an FID of 209.68 and test recall of 0.6022. Although the SA-GAN gave reasonable results, inspecting the images showed that the SA-GAN was not able to improve the image quality obtained with the models without Self-Attention mechanism.

However, in all optimal models the precision also decreased with approximately 2-3%. As the model decreases its bias towards the negatives, the consequence is that more negatives will accidentally be classified as positive. However, as the positive is already relatively high (>92%), we do not perceive this as an issue.

One of the main limitations of this research was the computational complexity. Training a GAN could easily take one to two days, while the evaluation process (FID and re-training of the classification model) also takes time. Moreover, in the field of GANs there is not yet a standard architecture which is guaranteed to work on every type of dataset. Most of the validated architectures in literature are trained on empirical datasets such as ImageNet or MNIST, which have a very different distribution and dataset size than our dataset. As GANs are heavily sensitive to small parameter or architecture changes, finding the optimal GAN is an expensive process. Another limitation is that solar panels cover a small part of the image, making it difficult for the generator to identify that the solar panel is an important part of the image. Therefore, it is not guaranteed that every generated positive image contains a solar panel. This is different from GANs trained on e.g. animal images, as it is clear that the object in the middle has to be generated. Although the recall (and accuracy) performance on the classification model increased, the performance could even improve more when this problem is addressed. Due to this issue we did not add more than 5,000 samples to the train and validation set, as this would imply that the proportion of fake positives compared to real positives would become too large.

Therefore, for future work we propose investigating whether this problem could be addressed, e.g. by testing the Heerlen-only trained classification model on the generated images and filtering out the images classified as negative. Another experiment could involve generating also negative synthetic samples and re-training the classification network on both classes, which could test whether the lack of data also plays a role in the recall rate. Moreover, the GAN architure can be further improved with state-of-the-art techniques, such as injecting more random noise with Adaptive Instance Normalization [33], substituting the noise vector by a Noise Mapping Network [33] or adding skip connections [32]. Also, progressively growing the image to be generated could speed up the training time [32].

6 Conclusion

The first research question denoted *'Can we identify the structure of the two datasets on two different geographical locations?'*. We have shown that we can use t-SNE and the clustering methods to segregate between different types of positive and negative samples. Moreover, as GANs are able to mimic the distribution of the training set, the generated images give us more insights into the train set structure.

Another question stated *'Are Generative Adversarial Networks suitable generation methods for aerial images?'*. The results demonstrated that both vanilla conditional GANs and conditional WGANs are able to generate images that resemble aerial solar panel images.

The third research question is *'Is the addition of synthetic images generated by Generative Adversarial Networks able to improve the performance on another geographical location?'*. Increasing the percentage of positives from 19.59% to 35.13% is able to improve the recall with 6.14%, while increasing the accuracy with 1.45%. This is at the expense of the precision, which decreased with 2.12%.

The last research question denotes *'Which type of Generative Adversarial Networks is the most suitable for generating aerial images?'*. The results have shown that the conditional Deep Convolutional WGAN with weight clipping of 0.01 has shown the most realistic and diverse results, as well as improving the recall rate the most.

In short, we showed that GANs are a suitable method to generate more samples of the same distribution, as well as using clustering as a pre-processing step to improve the image diversity. However, there are still some improvements possible to ensure that every generated image actually contains a solar panel.

References

[1] D. Tuia, C. Persello, and L. Bruzzone, "Domain adaptation for the classification of remote sensing data: An overview of recent advances," *IEEE geoscience and remote sensing magazine*, vol. 4, no. 2, pp. 41–57, 2016.

[2] M. Buda, A. Maki, and M. A. Mazurowski, "A systematic study of the class imbalance problem in convolutional neural networks," *Neural Networks*, vol. 106, pp. 249–259, 2018.

[3] L. Lusa and R. Blagus, "Smote for high-dimensional class-imbalanced data," *BMC Bioinformatics*, vol. 106, p. 14, 2013.

[4] L. Lusa *et al.*, "Evaluation of smote for high-dimensional class-imbalanced microarray data," in *2012 11th International Conference on Machine Learning and Applications*, IEEE, vol. 2, 2012, pp. 89–94.

[5] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[6] X. Zhu, Y. Liu, Z. Qin, and J. Li, "Data augmentation in emotion classification using generative adversarial networks," *arXiv preprint arXiv:1711.00648*, 2017.

[7] G. Douzas and F. Bacao, "Effective data generation for imbalanced learning using conditional generative adversarial networks," *Expert Systems with applications*, vol. 91, pp. 464–471, 2018.

[8] G. Mariani, F. Scheidegger, R. Istrate, C. Bekas, and C. Malossi, "Bagan: Data augmentation with balancing gan," *arXiv preprint arXiv:1803.09655*, 2018.

[9] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[10] M. Frid-Adar, I. Diamant, E. Klang, M. Amitai, J. Goldberger, and H. Greenspan, “Gan-based synthetic medical image augmentation for increased cnn performance in liver lesion classification,” *Neurocomputing*, vol. 321, pp. 321–331, 2018.

[11] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein gan,” *arXiv preprint arXiv:1701.07875*, 2017.

[12] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, “Improved training of wasserstein gans,” in *Advances in neural information processing systems*, 2017, pp. 5767–5777.

[13] M. Zheng, T. Li, R. Zhu, Y. Tang, M. Tang, L. Lin, and Z. Ma, “Conditional wasserstein generative adversarial network-gradient penalty-based approach to alleviating imbalanced data classification,” *Information Sciences*, vol. 512, pp. 1009–1023, 2020.

[14] P. Shamsolmoali, M. Zareapoor, L. Shen, A. H. Sadka, and J. Yang, “Imbalanced data learning by minority class augmentation using capsule adversarial networks,” *Neurocomputing*, 2020.

[15] S. S. Mullick, S. Datta, and S. Das, “Generative adversarial minority oversampling,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1695–1704.

[16] D. Lin, K. Fu, Y. Wang, G. Xu, and X. Sun, “Marta gans: Unsupervised representation learning for remote sensing image classification,” *IEEE Geoscience and Remote Sensing Letters*, vol. 14, no. 11, pp. 2092–2096, 2017.

[17] Y. Duan, X. Tao, M. Xu, C. Han, and J. Lu, “Gan-nl: Unsupervised representation learning for remote sensing image classification,” in *2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, IEEE, 2018, pp. 375–379.

[18] D. Ma, P. Tang, and L. Zhao, “Siftinggan: Generating and sifting labeled samples to improve the remote sensing image scene classification baseline in vitro,” *IEEE Geoscience and Remote Sensing Letters*, vol. 16, no. 7, pp. 1046–1050, 2019.

[19] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, “Imagenet large scale visual recognition challenge,” *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[20] J. H. Ward Jr and M. E. Hook, “Application of an hierarchical grouping procedure to a problem of grouping profiles,” *Educational and Psychological Measurement*, vol. 23, no. 1, pp. 69–81, 1963.

[21] P. J. Rousseeuw, “Silhouettes: A graphical aid to the interpretation and validation of cluster analysis,” *Journal of computational and applied mathematics*, vol. 20, pp. 53–65, 1987.

[22] T. Caliński and J. Harabasz, “A dendrite method for cluster analysis,” *Communications in Statistics-theory and Methods*, vol. 3, no. 1, pp. 1–27, 1974.

[23] G. Seig. (2019). “An easy introduction to 3d plotting with matplotlib,” [Online]. Available: `https://towardsdatascience.com/an-easy-introduction-to-3d-plotting-with-matplotlib-801561999725` (visited on 02/08/2021).

[24] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, “Spectral normalization for generative adversarial networks,” *arXiv preprint arXiv:1802.05957*, 2018.

[25] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, “Self-attention generative adversarial networks,” in *International conference on machine learning*, PMLR, 2019, pp. 7354–7363.

[26] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” *arXiv preprint arXiv:1706.03762*, 2017.

[27] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, G. Klambauer, and S. Hochreiter, “Gans trained by a two time-scale update rule converge to a nash equilibrium.,” 2017.

[28] D. Dowson and B. Landau, “The fréchet distance between multivariate normal distributions,” *Journal of multivariate analysis*, vol. 12, no. 3, pp. 450–455, 1982.

[29] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.

[30] M. Marchesi, “Megapixel size image creation using generative adversarial networks,” *arXiv preprint arXiv:1706.00082*, 2017.

[31] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, “On calibration of modern neural networks,” in *International Conference on Machine Learning*, PMLR, 2017, pp. 1321–1330.

[32] A. Brock, J. Donahue, and K. Simonyan, “Large scale gan training for high fidelity natural image synthesis,” *arXiv preprint arXiv:1809.11096*, 2018.

[33] T. Karras, S. Laine, and T. Aila, “A style-based generator architecture for generative adversarial networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2019, pp. 4401–4410.

7 Appendix A

7.1 KMeans clustering on train set

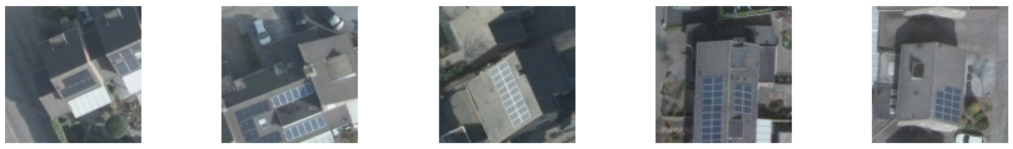

Figure 36: Images in cluster 0 (KMeans, train set)

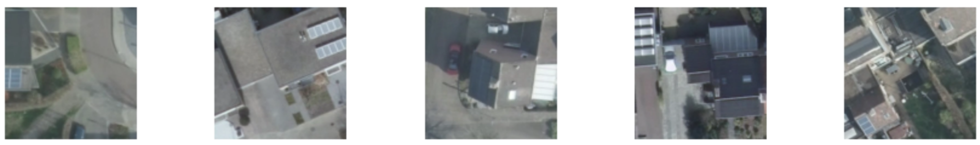

Figure 37: Images in cluster 1, positives (KMeans, train set)

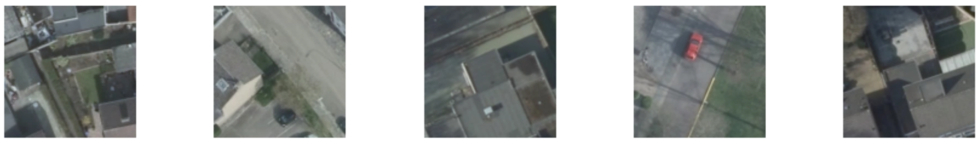

Figure 38: Images in cluster 1, negatives (KMeans, train set)

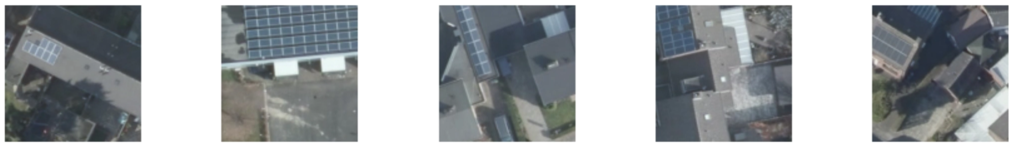

Figure 39: Images in cluster 2 (KMeans, train set)

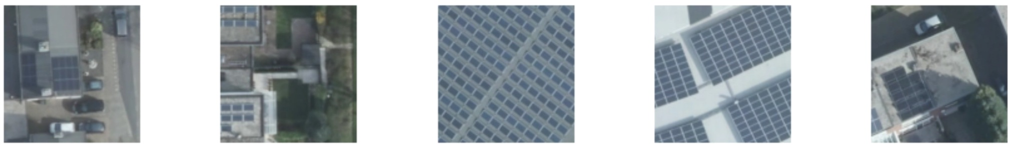

Figure 40: Images in cluster 3 (KMeans, train set)

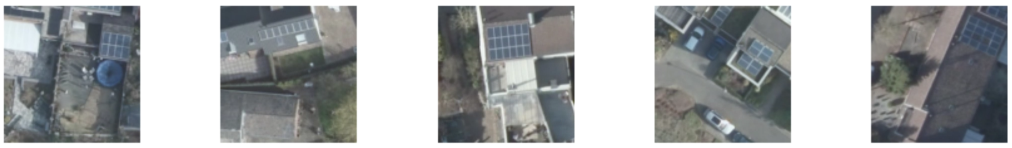

Figure 41: Images in cluster 4 (KMeans, train set)