Change Detection of Land Use: A Deep Learning Case-Study

Britt Schmitz[1] and Marc Ponsen[2]

[1] Maastricht University
[2] Statistics Netherlands

Abstract. *Bestand Bodemgebruik* is a file wherein the Netherlands is split up into many polygons, each polygon labelled with the most common type of land use for that specific area. This file is created by Statistics Netherlands only every few years. Data from many different sources must be manually studied and combined into a single new file and this takes considerable time. In this paper we apply a Siamese Convolutional Neural Network to the task of detecting changes of land use from one year to another using aerial imagery for those two years. The network's predictions point the creators of the *Bestand Bodemgebruik* towards areas that are highly likely to have changed, which allows them to work more efficiently. Our experimental results show that changes can be detected with accuracies varying from 68% to 90% for three different (and often mutating) land use classes.

Keywords: Siamese- and Convolutional Neural Networks · Earth Observation Data · Change Detection of Land Use

1 Introduction

Bestand Bodemgebruik (BBG) is a file containing digital geometries with information about the land use of the Netherlands for a given year. The country is split up into many polygons, each polygon labelled with the most common type of land use for that specific area. This file is created by Statistics Netherlands only every few years. Data from many different sources must be manually studied and combined into a single new file and this takes considerable time. Some examples of sources are the previous BBG, aerial images and the official land registry.

The creation of the BBG could be accelerated by the application of Deep Learning to aerial imagery. We recognize two different approaches, namely (1) a *direct classification* of the land use or (2) do a binary classification if land use for a particular area has changed or not. The current BBG has 38 different land use classifications, with a main focus being on change detection. Therefore, the second approach seems more intuitive. Direct classification could however definitely still be considered at a later stage.

Siamese networks haven been successfully applied to the task of *change detection*, whereas Convolutional neural networks have achieved state of the art

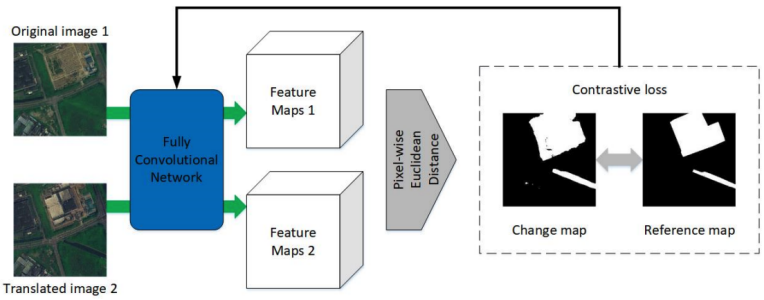

Fig. 1. A visual representation of a Siamese network. Image taken from https://towardsdatascience.com/change-detection-using-siamese-networks-fc2935fff82

results on image classification. Therefore, we apply a Siamese Convolutional Neural Network (SCNN) to the task of detecting changes of land use from one year to another. The input for the network are aerial imagery for those two years labeled with either *changed* or *unchanged*. Our experimental results with SCNNs show that changes can be detected with reasonable to high accuracy for three different (and often mutating) land use classes. The network's predictions point the creators of the BBG towards areas that are highly likely to have changed, which allows them to better prioritize their workload and work more efficiently.

In the remainder of this paper, we will first discuss related work on *change detection*. We will next describe our methodology in more detail in Section 3. In Section 4 the experimental setup and results are discussed. We finish with conclusions and ideas for future work.

2 Related Work

Change detection is a technique of recognizing temporal changes based on images acquired at distinct times. This is extensively used in many real-world applications [5]. In particular Deep Learning (DL) algorithms based on convolutional neural networks have been proven to be the state of the art when working with image data [4, 6]. DL is a specific domain of Machine Learning (ML) and is inspired by the information processing patterns found in the human brain. It requires a large amount of data to map the input to a desired output [1]. It has the ability to automate the learning of feature sets for several tasks and it enables learning and classification to be achieved in a single shot [1].Their ability to automatically derive complicated, hierarchical, and non-linear features from raw data allows them to overcome several limitations present in traditional change detection methods [5]. Their performance, however, is highly dependent on both the size and quality of the data set. Small sets of training data are unlikely to perform well [5, 4]. Nonetheless, preprocessing of the images before inputting them into the network may have significant benefits on the outcomes

[5]. Transfer learning is another commonly used method in change detection and its pre-trained structure allows for good results even with a small sets of training data [6].

Convolutional Neural Networks (CNNs) are a specific approach of DL that are widely used for solving complex problems [2]. They are most commonly applied to analyze visual imagery as they require relatively little preprocessing and they automatically detect the significant features without any human intervention [1]. In this work, we use pre-trained convolutional models.

CNN can be used to directly classify images (e.g, predict the type of land use), but as mentioned before, in this paper we aim to detect changes in images made at distinct times. As images from both years are needed in order to infer change from one year to another, both images should be simultaneously put into the network. The vanilla CNN architecture does not support this and therefore a Siamese network architecture is used. A pure-Siamese structure consists of two convolutional sub-networks that share weights and convert both inputs into a single output by identifying common features in the data at various levels [4, 6] (see Figure 1). This structure will be discussed in more detail in Section 3.2.

In the next Section we will explain in more detail the learning task addressed in this research, the data preparation steps involved and finally discuss the relevant parts of the chosen network architecture.

3 Methodology: change detection of land use

The encompassing learning task is to detect changes in land use based on aerial imagery for two different years. We may therefore choose to train the SCNN to detect any change (potentially 1369 distinct transitions between land usages). The goal of CNNs in particular is to derive (high level) patterns in images (e.g., forest turning into buildings may produce a distinguishable pattern for CNNs to discover). Intuitively, training a network to find patterns for *any transition of land use* seems as a daunting task. Therefore it was decided to simplify the learning task to ensure optimal performance. We train a separate model for each class of land use and only input images that initially contain this class. As for example, we train a model for detecting changes in "forest". The input then are images of two distinct years, where the image in the first year includes a polygon classified as "forest". If the polygon in the second year is still classified as "forest", the resulting label (i.e., output) for the network is "unchanged" and "changed" otherwise.

Figure 2 shows the mutations between (most frequently changing) land uses in the Netherlands based on the BBG in 2015 and 2017. We are looking for the classes that show the biggest number of changes. This is to ensure that we have enough data to run experiments. Additionally, predicting change in classes that are likely to change should be more useful for the creators of BBG at Statistics Netherlands compared to predicting change in classes that barely mutate. Both *building* sites and *other agricultural usage* classes are shown to change quite

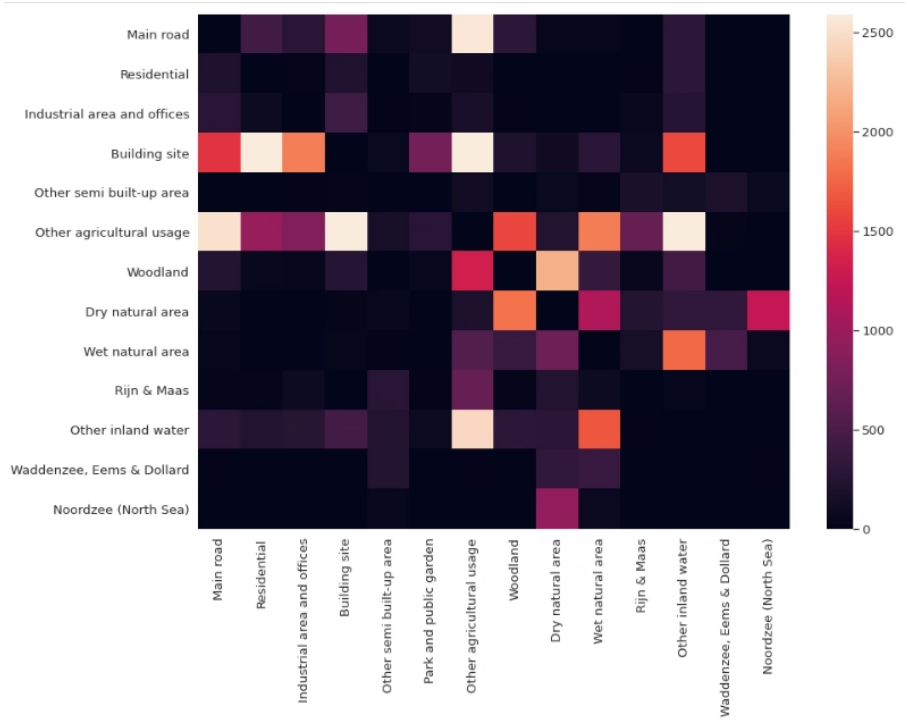

Fig. 2. A matrix of changes between the different classes of land use for the years 2015 and 2017. For simplicity, only the classes with significant amounts of changes are included. The y-axis denotes the class in BBG-2015, the x-axis the class of the same polygon in BBG-2017

.

often and are therefore used in the experiments. Additionally, we use the *forest* class as this is deemed the easiest example to visually detect changes for.

3.1 Data preparation

The aerial images of the Netherlands were first stored as squares each covering 500 by 500 metres of the Netherlands with 25 centimeters per pixel. Deep Learning networks require fixed size inputs, so this step is mandatory. We then created a separate dataset for each class we are currently considering. In order to achieve this, we have selected all the 500 by 500 metres squares which contain the desired class in the BBG-2015. A 500 by 500 metre square may contain several polygons of different land-use classes. For that reason we masked our images such that only the polygons belonging to the class are visible in the images of both years, this means that all other pixels are turned black. So far, we are not yet dealing with changing seasons and weather, as all images are obtained at once for each

year. Although taking into account these challenges might be something to work on in the future.

In order to train the network, the data must be labelled. Since we have the BBG of both 2015 and 2017, we can easily check each polygon to see whether the land use class is the same for both years. In the case where there is no change in class between both years, the image pair is given the label 1. This is common practise for change detection with Siamese networks, as both classes are the same and both images therefore belong to the same class. For the same reasoning, image pairs for which the classes differ are assigned the label 0. These labels seem counter intuitive at first, but are consistent with the contrastive loss function which will be explained in detail in Section 3.2.

A common problem is that in many real-world applications, the class distribution within the data is highly imbalanced, increasing the difficulty as there will often occur a bias towards the majority class resulting in recurring miss classification of the minority class [3]. In most use-cases, the main interest lies in the positive class which occurs with reduced frequency [3]. However, our focus revolves around the negative samples that make up our minority class, which will later be discussed in more detail. There exist many different methods to compensate for this bias. In this research we use straightforward methods such as adding class-weights and under sampling of the majority class data.

Lastly, we visually inspected our data after all data preparation steps by making montages for all classes we have worked on (see Figure 3 for the *forest* dataset). Montages are not necessary for the training of the network, or for any further predictions made by the model. However, they are useful for us to see what data is fed to the network and to get a clear impression of whether this learning task is feasible for each class. We can visually see that polygons that changed tend to have fewer green pixels, this is a pattern that a CNN potentially may detect.

3.2 Algorithm

Once we have our training data, we can start training the network. As previously mentioned, we use a SCNN architecture. We apply transfer learning by using pre trained models, such as ResNet and Inception, as underlying CNNs. The CNN processes both images before combining their results by computing the euclidean distance between the two feature maps. These values are finally processed by an output layer with a sigmoid activation function to ensure that the final results are contained between 0 and 1. The lower the output, the more likely it is that a change in land use has occurred. This occurs as it is common to implement Siamese networks with a contrastive loss function which is formulated as:

$$Y * D^2 + (1 - Y) * max(m - D, 0)^2 \tag{1}$$

where Y is the label (i.e. 0 or 1), D is the Euclidian distance between the sister networks, and m is the margin. It calculates the distance between an output of the network and an example from the same class, then contrasts it with the

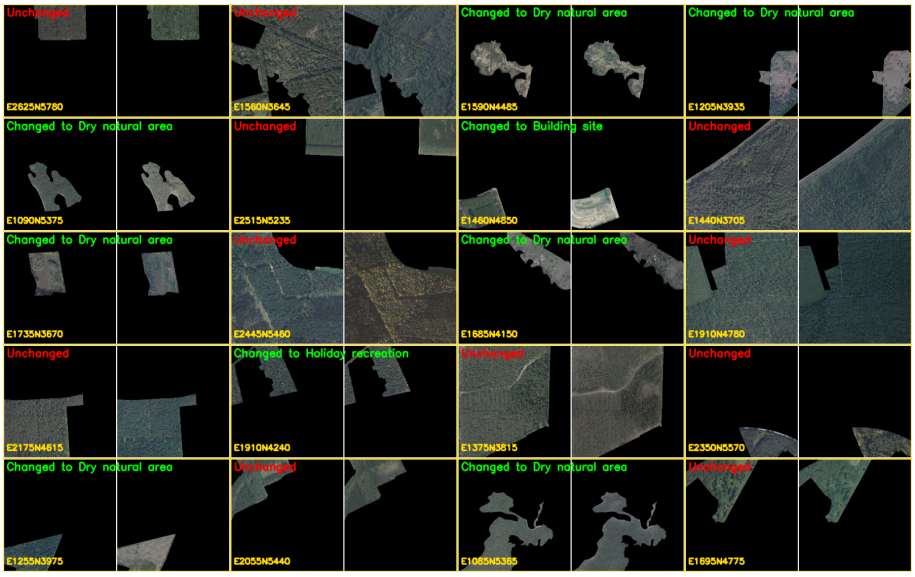

Fig. 3. An illustration of the training data for the forest class. We see four by five image pairs. The left image in the pair is the aerial image of 2015, whereas the right is the same polygon in 2017. The label 'unchanged' means that the polygon was classified as 'forest' in both years.

distance to an example from the opposite class. In other words, the loss is low if positive examples are similarly represented whereas negative examples are further in distance. It's goal is to evaluate how well the network can distinguish between the image pairs.

In the next Section we will discuss our results with SCNNs on the three selected land use classes, using the above described learning task, data and network architecture.

4 Experimental study

We did preliminary experiments with inception and restnet. Inception outperformed restnet consistently for our learning task for all selected land use classes. Therefore, in the remainder of this paper for further experiments we report only on the inception model. We first discuss our experiments for finding the optimal algorithm configuration wherein we train models with various different training datasets and hyper-parameter settings. Next, we train the optimally configured network for a longer period and test it on independent, unbalanced test set.

4.1 Finding the optimal algorithm configuration

The goal of these experiments is to find the best configuration for the SCNN. With respect to our training data, as a first step we balance the dataset us-

Model	F1/Accuracy	Precision	Recall
Balanced dataset			
inception-16-binary crossentropy-0.0001	0.69	0.70	0.61
inception-16-binary crossentropy-0.001	0.53	0.66	0.10
inception-16-binary crossentropy-1e-05	0.62	0.61	0.58
inception-16-contrastive loss-0.0001	0.68	0.67	0.72
inception-16-contrastive loss-0.001	0.52	0.74	0.02
inception-16-contrastive loss-1e-05	0.62	0.61	0.59
inception-32-binary crossentropy-0.0001	0.69	0.66	0.70
inception-32-binary crossentropy-0.001	0.54	0.53	0.26
inception-32-binary crossentropy-1e-05	0.62	0.59	0.64
inception-32-contrastive loss-0.0001	0.67	0.67	0.60
inception-32-contrastive loss-0.001	0.58	0.57	0.49
inception-32-contrastive loss-1e-05	0.61	0.58	0.61
Balanced and filtered dataset			
inception-16-binary crossentropy-0.0001	0.67	0.65	0.84
inception-16-binary crossentropy-0.001	0.53	0.54	0.86
inception-16-binary crossentropy-1e-05	0.57	0.59	0.61
inception-16-contrastive loss-0.0001	0.7	0.75	0.64
inception-16-contrastive loss-0.001	0.54	0.54	1
inception-16-contrastive loss-1e-05	0.54	0.58	0.51
inception-32-binary crossentropy-0.0001	0.67	0.7	0.65
inception-32-binary crossentropy-0.001	0.59	0.64	0.55
inception-32-binary crossentropy-1e-05	0.54	0.58	0.5
inception-32-contrastive loss-0.0001	0.67	0.7	0.65
inception-32-contrastive loss-0.001	0.58	0.61	0.58
inception-32-contrastive loss-1e-05	0.55	0.59	0.54

Table 1. Experimental results on data of building sites.The name of the model is based on hyper parameter settings, first the batch size, then the loss function and finally the learning rate.

ing under-sampling of the majority class. The balanced data contains an equal amount of changed and unchanged examples. We chose to balance the data due to the time complexity of training on the unbalanced data. Next, we can decide to either use the entire dataset, or we can filter out polygons that are too small. We have decided to work with a threshold of 10%, meaning that at least 10% of the total number of pixels in the image must be unmasked. Arguably, images with hardly any unmasked pixels do not really contribute to the learning process, but slows down convergence of the model.

Furthermore, we do a grid search for several hyper-parameters. For the batch size we compare performances for both 16 and 32. For learning rates we considered 0.001, 0.0001, and 0.00001. The last hyper-parameter is the loss function. We compare the previously described contrastive loss function with the more standard binary cross entropy loss function. In total we therefore ran (2 x 3 x 2) twelve different settings for the hyper-parameters on the two different training datasets (i.e. filtered or unfiltered). Training epochs were constrained (20) and

Model	F1/Accuracy	Precision	Recall
Balanced dataset			
inception-16-binary crossentropy-0.0001	0.86	0.88	0.84
inception-16-binary crossentropy-0.001	0.62	0.65	0.52
inception-16-binary crossentropy-1e-05	0.83	0.83	0.84
inception-16-contrastive loss-0.0001	0.86	0.89	0.81
inception-16-contrastive loss-0.001	0.72	0.90	0.50
inception-16-contrastive loss-1e-05	0.84	0.86	0.81
inception-32-binary crossentropy-0.0001	0.87	0.87	0.86
inception-32-binary crossentropy-0.001	0.83	0.85	0.81
inception-32-binary crossentropy-1e-05	0.81	0.80	0.82
inception-32-contrastive loss-0.0001	0.86	0.87	0.84
inception-32-contrastive loss-0.001	0.78	0.88	0.65
inception-32-contrastive loss-1e-05	0.83	0.83	0.81
Balanced and filtered dataset			
inception-16-binary crossentropy-0.0001	0.90	0.93	0.87
inception-16-binary crossentropy-0.001	0.80	0.77	0.87
inception-16-binary crossentropy-1e-05	0.77	0.78	0.78
inception-16-contrastive loss-0.0001	0.86	0.82	0.95
inception-16-contrastive loss-0.001	0.52	0.52	1.00
inception-16-contrastive loss-1e-05	0.72	0.74	0.71
inception-32-binary crossentropy-0.0001	0.90	0.93	0.87
inception-32-binary crossentropy-0.001	0.77	0.96	0.59
inception-32-binary crossentropy-1e-05	0.76	0.79	0.75
inception-32-contrastive loss-0.0001	0.91	0.92	0.91
inception-32-contrastive loss-0.001	0.42	0.47	0.76
inception-32-contrastive loss-1e-05	0.68	0.67	0.74

Table 2. Experimental results on *agricultural usage sites* dataset.

training was stopped if no considerable progress was made (via early stopping). Again, the goal was to quickly find an optimal algorithm configuration.

See all results on standard test set (twenty percent of whole dataset) for the "building site" model in Table 1. The best performance is decent with an accuracy of 70%. The best accuracy's for "other agricultural usage" and "forests" classes are considerably higher (see Table 2 and 3), respectively 91% and 88% of instances in test set are correctly classified.

Generally, we find that filtering the dataset seems to improve model performance. Also, the common denominators for the best algorithm configurations are contrastive loss and a learning rate of 0.0001. We choose these hyper-parameter settings, along with a batch size of 32, for the next batch of experiments.

4.2 Evaluating the best algorithm classification

The goal of these experiments is to evaluate the effectiveness of a longer trained model (using 'optimal' hyper-parameters). To recap, the experiments in the previous subsection were on balanced data. We now test on unbalanced data, be-

Model	F1/Accuracy	Precision	Recall
Balanced dataset			
inception-16-binary crossentropy-0.0001	0.82	0.77	0.91
inception-16-binary crossentropy-0.001	0.63	0.98	0.25
inception-16-binary crossentropy-1e-05	0.76	0.72	0.83
inception-16-contrastive loss-0.0001	0.82	0.90	0.72
inception-16-contrastive loss-0.001	0.64	0.96	0.27
inception-16-contrastive loss-1e-05	0.75	0.72	0.80
inception-32-binary crossentropy-0.0001	0.83	0.86	0.77
inception-32-binary crossentropy-0.001	0.76	0.93	0.54
inception-32-binary crossentropy-1e-05	0.72	0.68	0.81
inception-32-contrastive loss-0.0001	0.86	0.84	0.88
inception-32-contrastive loss-0.001	0.56	0.74	0.15
inception-32-contrastive loss-1e-05	0.72	0.69	0.78
Balanced and filtered dataset			
inception-16-binary crossentropy-0.0001	0.83	0.89	0.79
inception-16-binary crossentropy-0.001	0.71	0.66	0.97
inception-16-binary crossentropy-1e-05	0.63	0.65	0.7
inception-16-contrastive loss-0.0001	0.85	0.91	0.82
inception-16-contrastive loss-0.001	0.54	0.54	1
inception-16-contrastive loss-1e-05	0.64	0.69	0.62
inception-32-binary crossentropy-0.0001	0.84	0.84	0.87
inception-32-binary crossentropy-0.001	0.84	0.98	0.72
inception-32-binary crossentropy-1e-05	0.65	0.67	0.69
inception-32-contrastive loss-0.0001	0.88	0.9	0.87
inception-32-contrastive loss-0.001	0.4	0.38	0.17
inception-32-contrastive loss-1e-05	0.66	0.67	0.73

Table 3. Experimental results on *forest* dataset.

cause this matches our intended use-case, i.e., detecting changes of land-use for new years (such as 2020). Obviously, this new data is then unbalanced. We filter this dataset, as was described for previous experiments, because this increases model performance (see Tables 1, 2, 3). We then create a training set by picking out 80% of the unbalanced dataset and equally splitting the remaining data in a validation and a test set. Only the training set is then balanced, as this is important to prevent a biased model. However, both the validation and test sets remain unbalanced as this simulates the input once we start working on new data. We took the best hyper-parameter configuration found in the previous section and after training evaluated the model on the unbalanced test data. The results for selected classes are shown in Figure 4. The accuracy can be computed by adding the correct classifications divided by all instances in the test set. The accuracy for "building sites", "other agricultural usage" and "forests" classes respectively are 71% (655 instances), 91% (30791 instances) and 91% (4760 instances). These accuracy's reflect those in previous experiments.

The pitch for this research was potential efficiency gain for creators of BBG at SN which arguably can be best measured with the *recall* of the *changed* class.

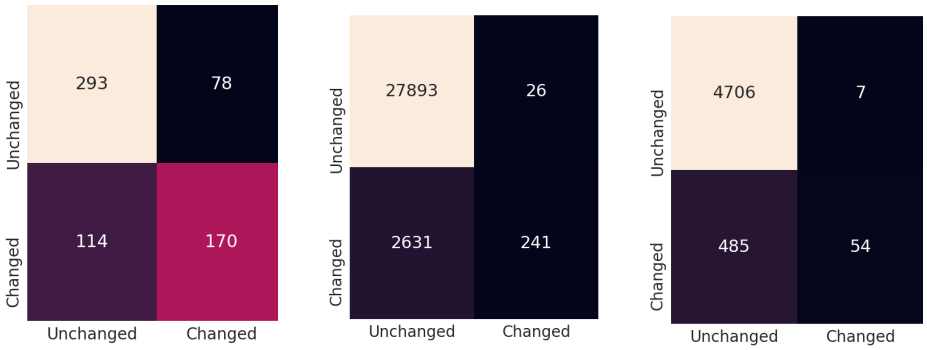

Fig. 4. Confusion matrix of *optimal models* tested on unbalanced test set respectively from left to right on "building sites", "other agricultural usages" and "forest". The real label is displayed on the x-axis, whereas the y-axis shows the predicted label. For the 'changed' class, the ratio of values horizontally reflects the precision, the ratio of values vertically reflects the recall.

We find that the *recall*, which can be computed by dividing the number of true positives by the total amount of positive samples, of the *changed* class is high for all classes, notably 68% for *building sites*, 90% for *other agricultural usage* and 88% for *forests*. In other words, the model detects the vast majority of the changes. For SN' use-case, this is an important metric. However, the current model often falsely predicts changes, i.e., the *precision* of the *changed* class is low, respectively 60%, 8% and 10%. Even though precision is low, the model still correctly filters out the vast majority of unchanged items (e.g., the precision of unchanged class is high) which enables people at SN to focus and reduce their workload.

5 Conclusions and future work

In this paper we applied a Siamese Convolutional Neural Network (SCNN) to the task of change detection of land use. The input data are aerial imaginary of the Netherlands and the so-called Bestand Bodemgebruik (BBG), a file assigning one out of 38 different land-use classes to areas in the Netherlands. Statistics Netherlands creates the BBG, a this takes considerable time. The use of SCNNs should accelerate the workflow for the creators, by pointing them to areas in the Netherlands that are likely yo have changed.

Experiments show that filtering images with hardly any unmasked pixels boosts performance. Smaller instances are disregarded as we consider them to contain too little information to be significant for the training process. Models were tuned with a grid search on several relevant hyper-parameters and then applied to the full data. The trained models seem to perform well on all three classes and most importantly, the amount of false positives is low. This means that the algorithm barely miss classifies any changed instances, therefore the

annotators are guided towards most changes. Also, notable efficiency gains can be expected as the model filters out the vast majority of instances that remain unchanged (and as such are not interesting for the creators of the BBG to inspect).

The results are promising so far, and it seems very feasible to at least semi-automate the production of the BBG using these networks. In order to confirm this, the next step is to transform the obtained predictions from the model into a format that can easily be interpreted by the creators of the BBG, such that they can implement these models in practise. As further future work we can continue rerunning our experiments on unbalanced datasets (using class-weights) to see whether we can improve our results even further. Additionally, we can extend our current work to more land use classes as well as look into the possibilities with direct classification. Finally, we can experiment with the input data by trying to implement *data augmentation* and *data fusion* by adding auxiliary data, such as SNs registry data, into the network to improve our results.

References

1. Alzubaidi, L., Zhang, J., Humaidi, A.J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Santamaría, J., Fadhel, M.A., Al-Amidie, M., Farhan, L.: Review of deep learning: concepts, CNN architectures, challenges, applications, future directions, vol. 8. Springer International Publishing (2021). https://doi.org/10.1186/s40537-021-00444-8, https://doi.org/10.1186/s40537-021-00444-8
2. Indolia, S., Goswami, A.K., Mishra, S.P., Asopa, P.: Conceptual Understanding of Convolutional Neural Network- A Deep Learning Approach. Procedia Computer Science **132**, 679–688 (2018). https://doi.org/10.1016/j.procs.2018.05.069
3. Johnson, J.M., Khoshgoftaar, T.M.: Survey on deep learning with class imbalance. Journal of Big Data **6**(1) (2019). https://doi.org/10.1186/s40537-019-0192-5, https://doi.org/10.1186/s40537-019-0192-5
4. Lee, S.H., Lee, M.J.: Comparisons of Multi Resolution Based AI Training Data and Algorithms Using Remote Sensing Focus on Landcover. Frontiers in Remote Sensing **3**(May), 1–12 (2022). https://doi.org/10.3389/frsen.2022.832753
5. Shafique, A., Cao, G., Khan, Z., Asad, M., Aslam, M.: Deep Learning-Based Change Detection in Remote Sensing Images: A Review. Remote Sensing **14**(4), 1–40 (2022). https://doi.org/10.3390/rs14040871
6. Shi, W., Zhang, M., Zhang, R., Chen, S., Zhan, Z.: Change detection based on artificial intelligence: State-of-the-art and challenges. Remote Sensing **12**(10), 1–35 (2020). https://doi.org/10.3390/rs12101688