Deep Learning for Archaeological Object Detection in Airborne Laser Scanning Data

Bashir Kazimi[1]
Leibniz University of Hannover, Institute of Cartography and Geoinformatics
kazimi@ikg.uni-hannover.de

Frank Thiemann
Leibniz University of Hannover, Institute of Cartography and Geoinformatics
thiemann@ikg.uni-hannover.de

Katharina Malek
Lower Saxony State Service for Cultural Heritage, Mining Archaeology
katharina.malek@nld.niedersachsen.de

Monika Sester
Leibniz University of Hannover, Institute of Cartography and Geoinformatics
sester@ikg.uni-hannover.de

Kourosh Khoshelham
University of Melbourne, Department of Infrastructure Engineering
k.khoshelham@unimelb.edu.au

Abstract

It is important to preserve archaeological monuments as they play a key role in helping us understand human history and their accomplishments for times with no or little written sources. The first step for this purpose is an efficient method for collecting and documenting information about objects of interest for archaeologists. Airborne laser scanning (ALS) is of great use in collecting and documenting detailed measurements from an area of interest. However, it is time consuming for scientists to manually analyze the collected ALS data. One possible way to automate this process is using deep neural networks. In this work, we propose a hierarchical Convolutional Neural Network (CNN) model to classify archaeological objects in ALS data. The data is acquired from the Harz mining Region in Lower Saxony, where a high density of different archaeological monuments including the UNESCO world heritage site Historic Town of Goslar, Mines of Rammelsberg, and the Upper Harz Water Management System can be found. To compare and validate our method, we run experiments on the same data set using two existing deep learning models. The first model is VGG-16; an image classification network pretrained on ImageNet[2] data. The second model is a stacked autoencoders model. The results of the classification as analyzed in this paper show that our model is suitably tuned for this task as it achieves the best classification accuracy of around 91 percent, compared to 88 percent and 82 percent accuracy by the pretrained and stacked autoencoders models, respectively.

2012 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Archaeology, Deep Learning, Object Detection

Digital Object Identifier 10.4230/LIPIcs.COARCH.2018.

[1] Some parts of this work has been carried out during a research visit at the University of Melbourne, Department of Infrastructure Engineering.
[2] http://image-net.org/about-overview

1 Introduction

Archaeology studies physical remains of objects in order to understand human history and culture for times with no or little written sources. It helps us have a glance at the lives of people in the past and how things have changed through time. While history uses written records to analyze events and explain human life, archaeology investigates what humans made and left behind and makes sense of how, where, and when they lived. Places with physical remains of human activities in the past are called archaeological sites. Archaeological sites can be of different types like settlements, cemeteries or depositions. They can be visible or not visible above ground. Studying these materials informs us of unrecorded human history in the past, and thus it is important to protect and conserve archaeological sites. To do so, it is necessary to first identify and document such sites. Airborne laser scanning (ALS) is an efficient remote sensing technique for collecting 3D data of large areas by measuring the range and reflectance of objects on their surface. It can be used to capture data from archaeological sites, but it needs to be manually analyzed by archaeologists for correct identification and localization of archaeological objects. This could be very time-consuming, and needs an automated process.

Deep learning has shown great potential in automating processes in many applications and outperformed classical machine learning methods. It has performed well in image classification, machine translation, speech recognition, image captioning, healthcare domain applications, prediction of events, and many more [1]. It can work with 2D image data [2], 3D point cloud data [3], hyperspectral image data [4], and interpret sequence to sequence data [5]. Among deep learning models, convolutional neural networks (CNN) have been proved to work well on image data [2, 6], while stacked autoencoders are more suitable for learning features from the input data and encoding them to a compressed vector, from which a decoder learns to generate the original input data [7]. Deep learning models usually require a lot of training data to be able to generalize well. In many applications where the amount of training data is not sufficient, transfer learning can be applied. Transfer learning refers to training a model on a large set of training data first, and then using the pretrained model as a feature extractor for a new application with small dataset [8]. There are many models pretrained on ImageNet data; Xception [9], VGG Net [10], ResNet [11], Inception [12], etc, which are shown to perform well as feature extractors for other image-based domains.

In this work, we build a hierarchical CNN model to detect objects in archaeological sites using digital terrain models (DTMs) generated from ALS data. The data is collected from the Harz mining region in Lower Saxony, Germany. Objects to be detected are archaeological objects such as hollow ways, streams, pathways, lakes, streets, ditches, heaps, mining shafts, and more, but for this study, our model is fit to detect 4 classes of objects: natural streams, lakes, tracks, and an 'others' class which represents the rest of the objects for which enough labeled data is not available yet. We also implement and use a stacked autoencoders model, and the pretrained VGG16 model to compare the performance of our model. The rest of the paper is arranged as follows. Section 2 explains previous work on deep learning for object detection. Section 3 gives details of our model, explains the stacked autoencoder model, and describes the pretrained VGG16 model used for comparison. Section 4 includes the experiments performed and the results obtained, comparing our model with the other models used in this study. Finally, Section 5 summarizes this work and points out open future directions in this line of research.

2 Related Work

Deep learning has been successfully used in many applications. Among the deep learning methods, recurrent neural networks (RNNs) are good at dealing with sequential data as they take into account temporal information. RNNs have been applied in speech recognition to map acoustic sequences to phonetic sequences [13]. RNNs have also been used in natural language processing to translate text from one language to another [14, 15, 16]. Another famous method in deep learning is convolutional neural networks (CNNs). CNNs take into account spatial correlation among data points and hence perform well in image-based data. CNNs have been used in image classification [2] video classification [17], face recognition [18], scene labelling [19], action recognition [20], pose estimation [21], and more. Autoencoders are neural networks that take an input, map it to a latent space using a hidden layer, and reconstructs the original input by the final output or decoding layer. Stacked autoencoders are deep networks made of multiple autoencoders stacked together. Stacked autoencoders have also been applied in many applications: feature extraction and classification of hyperspectral data [22, 4], internet traffic prediction [23], classification and diagnosis of the parkinson disease [24], and more. Even though deep learning models perform well in most applications, they are highly dependent on a lot of data, and usually labeled data. In domains where sufficient data is not available, transfer learning can be applied. First, a deep neural model is trained on a domain with a high amount of labeled data, and the learned parameters are saved. The learned parameters can then be transferred and used in a domain with a smaller amount of data for feature extraction. The extracted features are used to train a classifier directly, or in some cases, the transferred weights are further tuned for that specific application [8]. There are many other deep learning methods with various fields of applications, but that is out of the scope of this research.

Similar to other domains, deep learning methods have gained popularity and are used in the field of remote sensing as well, such as scene classification in satellite imagery [25], object detection in optical remote sensing images [26], 3D shape reconstruction [27], and more. Some example applications of deep learning methods specifically on laser scanning data include tree classification [28], road manhole covers detection [29], and road markings detection [30]. Hu and Yuan [31] used CNNs to extract DTMs and filter out non-ground points from ALS data, and claim that their method performs better than previous filtering methods.

Generally, deep learning models are fit to work with image data or 3D point clouds where the example objects have a complete 3D shape. In some cases, deep learning models need to work with height information or DTMs extracted from ALS data as in the work of Politz et al. [32]. In this study, we also work with ALS data. In order to leverage deep learning, specifically convolutional neural networks, we build a hierarchical CNN model inspired by the work of Palafox et al. [33], and preprocess our ALS point cloud data to generate a DTM, and image-like inputs, 2.5D height maps, to train our model and detect objects in archaeological sites.

3 Approach

To detect objects in archaeological sites, we propose a two-step approach. The first step is to train classifier models that take height map data of size n x n as input and assign a label to it. The second step is the detection process in which the input is height map data of size h_1 x h_2 where $h_1, h_2 \gg n$, and the output is a heat map of size h_1 x h_2 for each class showing the probability of the objects at each location. Since the objects to be detected

Algorithm 1 Selecting optimal values of n for the input size n x n

```
1: $nList$: list of possible values of $n$, in this research {8,10,...,98,100}
2: $num\_optimal$: number of optimal values to select (= 5 in this research)
3: $trainData$: DTM showing height maps from the training region.
4: $testData$: DTM showing height maps from the test region.
5: procedure TopNselection($nList$, $num\_optimal$, $trainData$, $testData$)
6:     $models \leftarrow \emptyset$
7:     $accuracies \leftarrow \emptyset$
8:     $losses \leftarrow \emptyset$
9:     $topN \leftarrow \emptyset$
10:    for each item $n$ in $nList$ do
11:        $models_n \leftarrow$ model trained with input size $n$ x $n$ clipped from $trainData$
12:        $accuracies_n \leftarrow$ accuracy of the $models_n$ on $testData$
13:        $losses_n \leftarrow$ loss of the $models_n$ on $testData$
14:    $sortedList1 \leftarrow nList$ sorted based on $accuracies$ in decreasing order
15:    $sortedList2 \leftarrow nList$ sorted based on $losses$ in increasing order
16:    $i \leftarrow 1$
17:    while true do
18:        $first\_i\_acc \leftarrow$ first i elements in $sortedList1$
19:        $first\_i\_loss \leftarrow$ first i elements in $sortedList2$
20:        $common\_Ns \leftarrow$ list of common numbers in $first\_i\_acc$ and $first\_i\_loss$
21:        for each item $c$ in $common\_Ns$ do
22:            if $c$ is not already in $topN$ then
23:                append $c$ to $topN$
24:        if $i >$ length of $nList$ then
25:            break
26:        $i \leftarrow i + 1$
27:    return $topN$[: $num\_optimal$] # first $num\_optimal$ elements
```

Algorithm 2 The detection process

```
1: $classifier_n$: classifier model trained with input size $n$ by $n$
2: $H$: Input matrix of size $h_1$ x $h_2$, showing height maps from the region of interest.
3: $M$: Number of classes the the classifier model is trained to detect.
4: procedure DetectObjects($classifier_n$, $H$, $M$)
5:     $O \leftarrow 0$ # a matrix of zeros of shape ($M$, $h_1$, $h_2$)
6:     for each item $i$ in range(0, $h_1$ - $n$ + 1) do
7:         for each item $j$ in range(0, $h_2$ - $n$ + 1) do
8:             $currentPatch \leftarrow H[i : i + n, j : j + n]$
9:             $m \leftarrow$ class predicted by $classifier_n$
10:            $O[m, i : i + n, j : j + n]$ += ones(n, n) # increment by 1
11:    return $O$ # a heat map of size $h_1$ x $h_2$ for each class
```

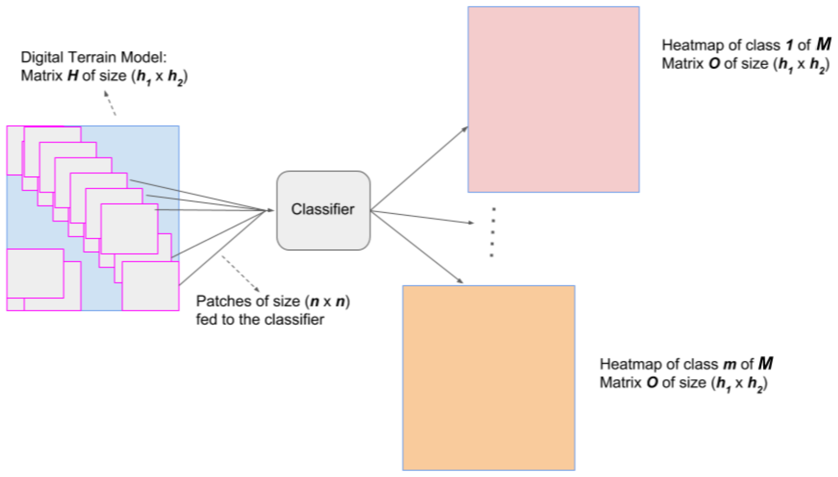

Digital Terrain Model:
Matrix **H** of size (**h_1** x **h_2**)
Classifier
Patches of size (**n** x **n**)
fed to the classifier
Heatmap of class **1** of **M**
Matrix **O** of size (**h_1** x **h_2**)
Heatmap of class **m** of **M**
Matrix **O** of size (**h_1** x **h_2**)

Figure 1 *Detection process. Heatmap matrices O are initialized to 0. Sliding one pixel at a time, patches of size n x n are extracted from the DTM H, and fed to the classifier. Based on the class label predicted by the classifier, the pixels on the same location on matrix O of the predicted class are incremented by 1. At the end of the process, each object class 1 to M, will have their corresponding heat maps O_1 to O_M*

are of different sizes and granularities - sometimes even within a single object class, we first investigate and select 5 optimal size for the input to the classifier networks. The 5 optimal values of n for the input size are selected using Algorithm 1.

After selecting 5 optimal values of n for the input size to the classifier models, the 5 corresponding trained classifiers are used in the second step, the detection process. The detection process takes a big matrix of size h_1 x h_2, height map data from the region of interest, as input and returns one heat map of size h_1 x h_2 for each class, indicating the probability of objects at each location of the heat map. The detection method is performed using Algorithm 2, and illustrated in Figure 1.

To this end, we build a hierarchical CNN model and compare it with two of the existing deep learning methods: the pretrained VGG16 model available in the Keras API [34], and stacked autoencoders model implemented by ourselves. The following subsections describe the three models.

3.1 Our CNN Model

Our CNN models are multi-class classifiers consisting of 4 Convolutional layers each of which is followed by a max pooling function and a ReLU operation which is defined in the following equation.

$$ReLU(x) = max(x, 0) \tag{1}$$

In the end, there is one fully connected layer with a softmax function outputting the probabilities for each class label. The structure used for the CNN models is depicted in Figure 2. For this study, we train multiple CNN models that take as input matrices of size; n x n, with n ranging between, and including 8 and 100, taking only even numbers. When the models are trained, we check their performance based on the *accuracy* and *loss* obtained from the test set and choose the best 5 for the second step, using Algorithm 1.

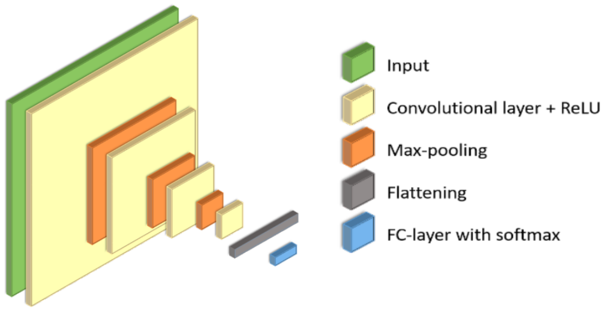


Figure 2 *Structure of our CNN Model. The model takes an input matrix of shape n by n and outputs a class label for it.*

3.2 Pretrained VGG16 Model

VGG16 is a deep convolutional neural network for object recognition developed by Visual Geometry Group at Oxford [10]. The model has been trained on ImageNet data, a database of images where each image is labeled as one of the thousand classes consisting of concepts included in the WordNet hierarchy. The learned training weights have been made available online, making it possible for various deep learning frameworks, such as Keras, TensorFlow, and Caffe to include the implementation of the model as an application, along with the pre-trained weights. We use the Keras implementation of this model. The Keras implementation allows for input images with height and width greater than 48 up to 224 and input channels of 3. Since the input data in our study is height map data with a single channel, we triple the number of channels by copying the same values for our single-channeled input data to create 3-channeled inputs. The final classification layer of the model is replaced with our own layer, a fully connected layer followed by a softmax function classifying 4 object classes. The structure of the modified VGG16 model is depicted in Figure 3. The model is fine-tuned for our task, keeping the weights in the original VGG16 model fixed, and only training the weights in the newly added fully connected layer. The same approach is taken for this model.

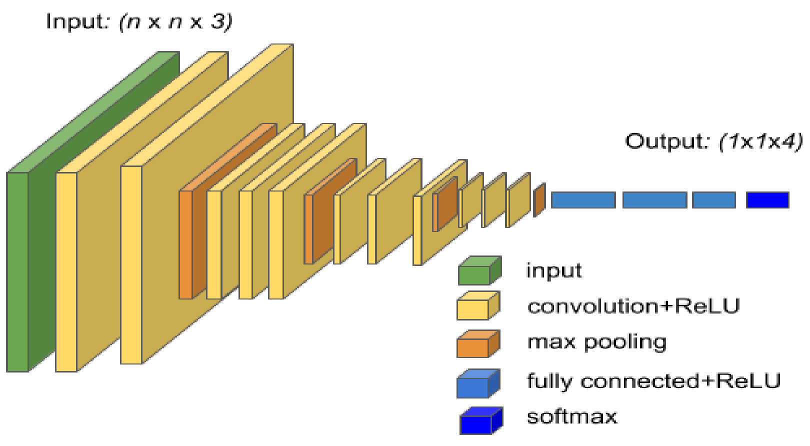


Figure 3 *Structure of the modified pretrained VGG16 Model, instead of 1000 classes as the original model, we classify 4 objects, hence the modification in the final layer.*

We train the model with input sizes n x n, with n ranging between, and including 48 and 100, taking only even numbers. The range of values for n starts from 48, rather than 8 since 48 is the smallest possible value for the input size to the pretrained model. After selecting top 5 models based on *accuracy* and *loss* on the test data using Algorithm 1, similar to our own model, we scan the area of interest and generate heat maps of each object class for each model using Algorithm 2.

3.3 Stacked Autoencoders (SAE) Model

The stacked autoencoders model takes as input a height map data of size n x n, learns to *encode* it to a compressed, smaller matrix and is trained to *decode* and generate the original input, a height map of size n x n. The encoder consists of 3 convolutional layers, each followed by a ReLU operation, and a max pooling operation. Then follows a fully connected layer whose output is the *code*, used by the following decoder. The decoder consists of 4 convolutional layers, each with a ReLU operation, and an upsampling function. The output of the decoder has the same size as the original height map data that is given as input to the encoder. The structure of the SAE model is illustrated in Figure 4. After training, only the encoder model is used to extract features from the data. The training data is given to the encoder as input, the resulting compressed, smaller vectors – referred to as *code* – are then used as features to train a simple neural network to classify into 4 classes. The simple classifier neural network that we have implemented consists of a fully connected layer with a softmax function. Similar to the previous two methods, we try multiple values for n ranging between, and including 48 and 100. The final top 5 models are then used for object detection in the same manner as explained before.

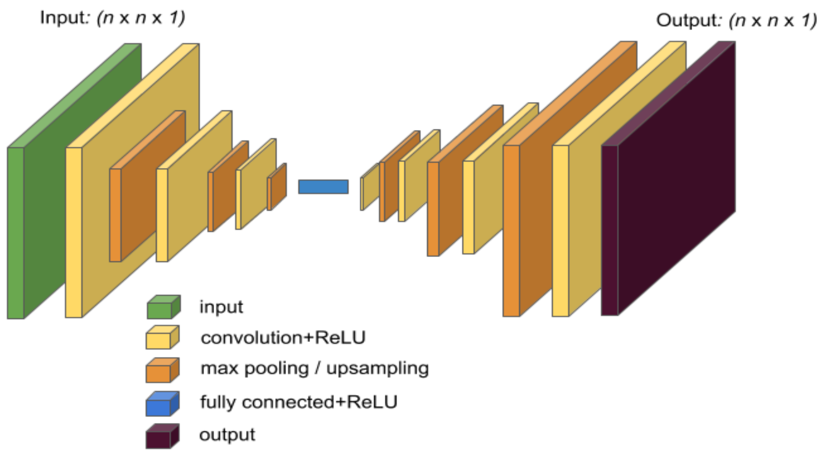
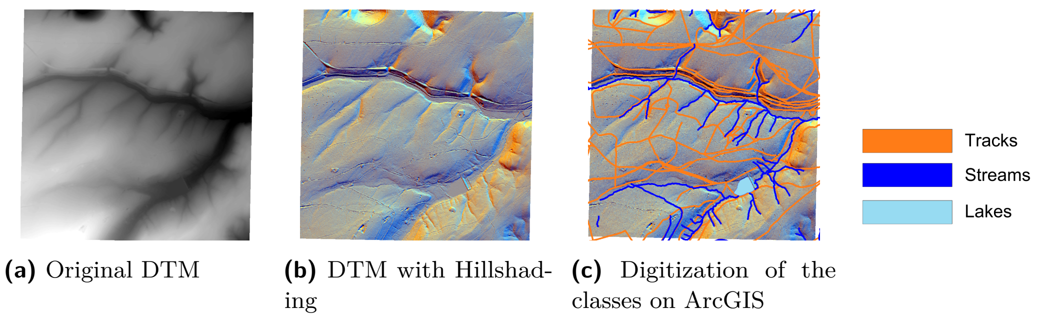


Figure 4 *Structure of the SAE model trained to take an input of size (n x n), and regenerate it.*

4 Experiments and Results

Our region of study is the western part of the Harz Mountains in the south of Lower Saxony, Germany. The upper Harz was once one of the most important mining regions in Germany. The major products of its mines were silver, copper, lead, iron, and zinc. The area is home for the Upper Harz Water Management System, which together with the Mines of Rammelsberg and the Historic Town of Goslar was declared as UNESCO World Heritage Site. The Upper

Harz Water Management System is one of the largest and most important historic mining water management systems in the world where dammed ponds, ditches, tunnels and drainage adits were built for collecting, diverting, and storing surface runoff water during the 16th-19th centuries. The area is also covered with plenty of mining relics such as mine shafts, hollow ways, medieval heaps, and more. ALS data is acquired from the region of the interest, and known objects are digitized using the ArcGIS software. Since our model and the other two previous methods use 2D data as input, we first create a DTM with a 1-meter resolution for the area of interest using the ALS point cloud data. Afterwards, the digitized labels are used to buffer known objects and generate matrices of size 100 x 100 with the corresponding label for the object, in order to create training and test datasets for our models. In this study, our labeled data contains four object classes including streams, tracks, lakes, and an 'others' label representing everything else in the dataset. Streams and tracks are labeled using continuous, linear-shaped buffers while lakes are labeled using closed areal buffers. Areas excluding the three known object classes are all labeled as 'others'. The generated matrices indicate height values and are used as input to our model and the SAE model. For the pretrained VGG16 model, the matrices are tripled to create 3-channeled input data of the same width and height from the 1-channel height map data. The resulting 3-channeled matrices are then used for training and testing the pretrained models. To create training and test data for models with input size n smaller than 100, matrices of size n x n are cropped from the center of the original matrices generated earlier, and used by the corresponding model. A small part of the region of study with an overlay of the classes used in this study is shown in Figure 5, and the statistics for the generated training and test data used in this study are shown in Table 1.



(a) Original DTM **(b)** DTM with Hillshading **(c)** Digitization of the classes on ArcGIS

Figure 5 Test area from the region of study, 2x2 kilometers squared area in the Harz Mountains Region in Lower Saxony.

	Train	**Test**
Number of examples	20.000 (80 %)	5.000 (20 %)

Table 1 Dataset statistics

To have a fair comparison among them, extreme care has been taken to use as much similar configuration for the 3 models as possible. Some of the hyperparameters used while training the models are listed in Table 2. As explained in Section 3, the first method was explored using input sizes n ranging from 8 to 100, but since 48 is the smallest possible n value for the pretrained VGG16 model, the range of values used for n for this model and the SAE model is 48 to 100. The 5 optimal filter sizes for each model is different. Therefore, to compare the models, we extend our previous method of choosing optimal filter sizes,

	Epochs	Batchsize	Range of values for n	Optimizer
Our CNN	10	32	8-100	Adam (lr=0.001)
SAE	10	32	48-100	Adam (lr=0.001)
VGG16	10	32	48-100	Adam (lr=0.001)

Table 2 Settings used for the three models during training. *lr* stands for **l**earning **r**ate

Algorithm 1, to select 5 common filter sizes for all the three models. The new method for selecting optimal filter sizes, n, for the three models in common is shown in Algorithm 3.

Algorithm 3 Selecting optimal values of n for the three models in common

```
1:  $nList$: list of possible values of $n$, in this section {48,50,...,98,100}
2:  $num\_common$: number of common optimal values to select (= 5 in this research)
3:  $trainData$: DTM showing height maps from the training region.
4:  $testData$: DTM showing height maps from the test region.
5:  procedure CommonNSelection($nList$, $num\_common$, $trainData$, $testData$)
6:      $len\_nList \leftarrow$ length of $nList$ # Passed to Algorithm 1 as $num\_optimal$ instead of 5
7:      $optimalsCNN \leftarrow TopNselection(nList, len\_nList, trainData, testData)$
8:      $optimalsVGG16 \leftarrow TopNselection(nList, len\_nList, trainData, testData)$
9:      $optimalsSAE \leftarrow TopNselection(nList, len\_nList, trainData, testData)$
10:     $topN \leftarrow \emptyset$
11:     $i \leftarrow 1$
12:     while true do
13:         $i\_cnn \leftarrow$ first i elements in $optimalsCNN$
14:         $i\_vgg \leftarrow$ first i elements in $optimalsVGG16$
15:         $i\_sae \leftarrow$ first i elements in $optimalsSAE$
16:         $common\_Ns \leftarrow$ common numbers in $i\_cnn$, $i\_vgg$, and $i\_sae$
17:         for each item $c$ in $common\_Ns$ do
18:             if $c$ is not already in $topN$ then
19:                 append $c$ to $topN$
20:         if $i >$ length of $nList$ then
21:             break
22:         $i \leftarrow i + 1$
23:     return $topN[: num\_common]$ # first $num\_common$ elements
```

In this experiment, the 5 common optimal filter sizes and the results obtained by the corresponding models are listed in Table 3. As the filter size grows, VGG16 generalizes better while our model and the SAE decreases generalization ability. This notion is visible in the loss values obtained by the models on the test set. Confusion matrices for the smallest and biggest optimal filters for each model are shown in Table 4. The evaluation metrics in Table 3 and the confusion matrices in Table 4 are based on the test data set, which similar to our training data set, is located, labeled, and verified by archaeology experts. In general, as the filter size grows, there is a decrease in prediction accuracy of the three models for linear structures like tracks and streams, but an increase in prediction accuracy for areal structures like lakes. It is also observed from the confusion matrices that the models confuse tracks with streams and vice versa more than they confuse them with lakes. This is expected since tracks and streams are linear and are somewhat more similar in structure. Additionally, a higher number of lakes are confused by the three models with streams rather than tracks,

Filter	Our CNN	VGG16	SAE
48	90.6	85.5	82.1
64	90.2	85.9	80.7
76	89.8	87.1	76.2
82	89.7	86.7	78.3
98	90.7	88.0	80.9

(a) Accuracy in % obtained from the test set.

Filter	Our CNN	VGG16	SAE
48	0.31	0.40	0.49
64	0.31	0.37	0.54
76	0.31	0.36	0.63
82	0.32	0.38	0.58
98	0.33	0.35	0.57

(b) Loss obtained from the test set.

Table 3 Quantitative evaluation results for the three models using 5 common optimal filter sizes.

Actual \ Predicted	Tracks	Streams	Others	Lakes
Tracks	91.9	2.1	4.8	1.3
Streams	4.1	89.3	4.9	1.7
Others	8.1	6.6	83.0	2.2
Lakes	0.8	2.0	0.8	96.4

(a) Our Model; Filter size $n = 48$

Actual \ Predicted	Tracks	Streams	Others	Lakes
Tracks	89.7	3.9	5.8	0.5
Streams	3.3	89.8	5.5	1.4
Others	8.1	8.1	82.6	1.1
Lakes	0.7	1.0	0.6	97.6

(b) Our Model; Filter size $n = 98$

Actual \ Predicted	Tracks	Streams	Others	Lakes
Tracks	73.5	12.1	12.4	1.9
Streams	4.2	86.7	6.5	2.6
Others	10.4	9.5	77.8	2.3
Lakes	0.7	1.4	0.9	97.0

(c) VGG 16; Filter size $n = 48$

Actual \ Predicted	Tracks	Streams	Others	Lakes
Tracks	72.1	14.7	13.1	0.1
Streams	2.1	91.5	5.8	0.5
Others	5.4	12.4	82.0	0.2
Lakes	0.4	1.1	0.4	98.1

(d) VGG16; Filter size $n = 98$

Actual \ Predicted	Tracks	Streams	Others	Lakes
Tracks	70.0	19.9	8.2	1.8
Streams	1.8	91.9	3.4	2.8
Others	8.5	24.3	66.0	1.1
Lakes	1.8	7.2	1.4	89.7

(e) SAE model; Filter size $n = 48$

Actual \ Predicted	Tracks	Streams	Others	Lakes
Tracks	77.9	12.0	8.4	1.6
Streams	6.0	87.2	4.2	2.6
Others	22.1	12.0	64.6	1.3
Lakes	2.9	8.2	2.3	86.5

(f) SAE model; Filter size $n = 98$

Table 4 Confusion matrices for the three models and the two filter sizes. Values are shown in percent.

which is also intuitive since some of the lakes with a smaller width could be confused with larger streams.

In a subsequent step, the detection results are analyzed visually by inspecting the generated heat maps. The final heat maps generated by the models for filter sizes $n = 48$ and $n = 98$ are illustrated in Tables 5 and 6. The heat maps are generated for the four object classes from the area shown in Figure 5. The colors from low to high show the certainty of the model in detecting the object at that location. In the following, the visual analysis takes into account, that potential features are represented with high confidence, and as connected, continuous components with similar confidence.

Taking into account the resolution of the DTM, 1 meter resolution, using smaller values for n while creating training data, the n x n patches include a good representation of tracks and streams or other objects of smaller width. However, they yield poor results for lakes or other objects of larger width. This is due to the fact that the patch represents just a small part of the whole object, e.g. it would be a flat height map cropped from the middle of a lake. On the other hand, using bigger values for n results in a better representation for objects of larger width, such as lakes, since the training patches contain the full structure of lakes and parts of their surroundings. However, it results in a poor representation of objects of smaller width, since only a small section in the middle of the patch represents the object and the majority of the patch includes other objects. The effect of this choice of values of n is clearly seen in the heat maps shown in Tables 5 and 6. For models with smaller values of n, the generated heat maps of narrow tracks and streams are more accurate, but those of lakes are less accurate. Streams with larger widths appear to be labeled as lakes. For models with larger values of n, the heat maps are more accurate for lakes, which are of a larger width, but those of streams and tracks are less accurate. A larger portion in the surroundings of tracks and streams are also labeled as streams and tracks.

While the variation in the filter size has the same effect for all the three models, there are significant differences among the models using the same filter sizes. As illustrated in the heat maps for filter size 48 in Table 5, the VGG16 model creates the best heat map for *tracks* while the one generated by the stacked AE model is less accurate than our model. In generating heat maps for the *streams*, our model performs better than the other two, and the Stacked AE model is the worst among them again. While there are some similarities between our model and the VGG16, after taking a closer look, one can realize that the *streams* heat map by our model is more continuous and flowing, thus more accurate, compared to discontinuities on the *streams* heat map by the VGG16 model. For example, the continuous stream on the lower left corner is more apparent in our model than the VGG16 model. Finally, VGG16 performs better in generating heat maps for *lakes*, and our model generates a rather similar heat map to that of stacked AE model. Comparing the results to the ground truth, it seems that the flat areas in the upper part and the middle of the test area are detected as lakes by our model and the stacked AE model.

Using filter size of 98 (see Table 6), the generated heat maps of our model and the stacked AE model indicate a clear pattern of *tracks* on the upper section of the area, while the VGG16 model is not as confident. In the lower part of the area, however, there is a better pattern of tracks by the VGG16 model, even though the confidence values are still not high. In the heat maps for the *streams*, our model is better than the other two. In the heat map generated by our model, the streams are continuous and more similar to the actual streams, while the VGG16 model produces discontinuous streams in the middle and upper section of the area, and shows multiple tiny streams in the lower section of the area as one big stream, rather than the detailed information on the heat map by our model. The stacked AE model

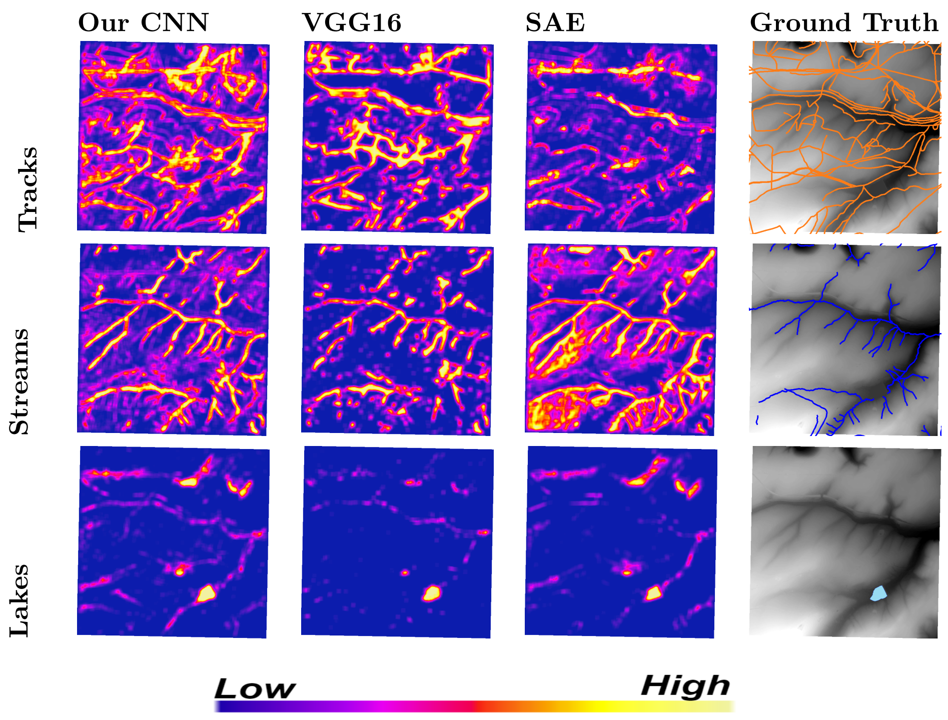


Table 5 Heat maps using filter size 48 x 48. Colors show the confidence of the models in detecting objects at that location.

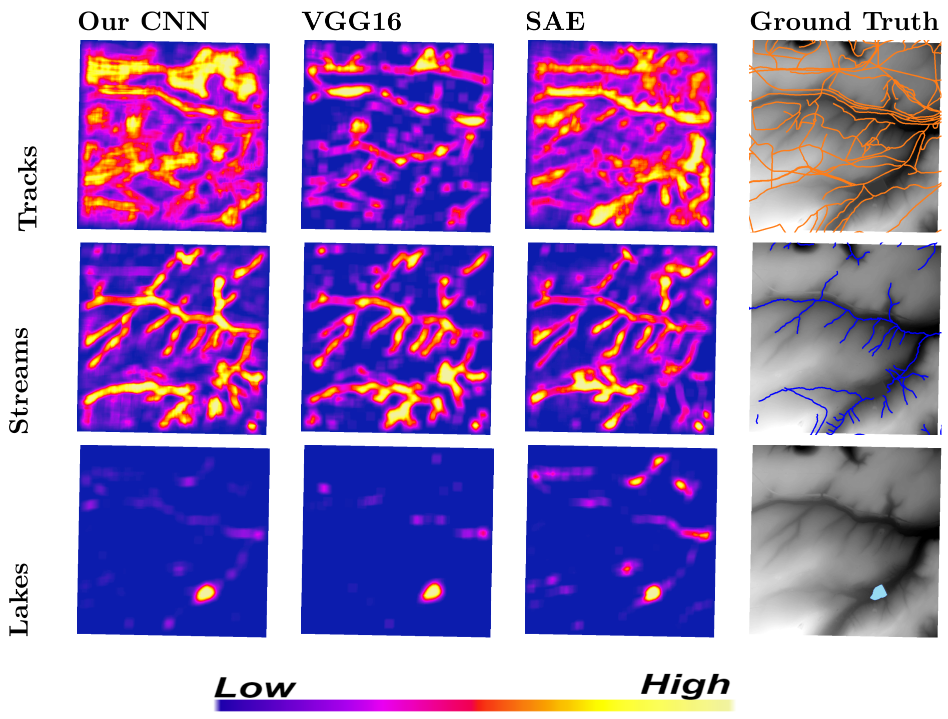


Table 6 Heat maps using filter size 98 x 98. Colors show the confidence of the models in detecting objects at that location.

produces a heat map with even more discontinuity, and with lower confidence values. The heat maps for *lakes* are produced in a similar manner by our model and the VGG16 model. The stacked AE model falsely labels some flat areas in the upper right and middle section of the area as lakes instead of labeling them as 'others'.

5 Conclusion and Outlook

In this study, we present a two-step approach to detect objects in archaeological sites, leveraging deep learning, specifically CNNS and using ALS data. In the first step, we develop a multiclass classifier CNN model and train multiple instances of the model with various sizes as the input matrix. Based on their accuracy, the best models are then in the second step applied to the DTM of the area of interest, to classify objects and generate a heat map for each class. As a result of the experiments performed, it is concluded that smaller matrices as training data result in better detection of objects with a smaller width, while bigger matrices are good for objects of larger width.

Our model is compared with two existing methods, the pretrained VGG16 model and a stacked AE model which are also trained with various sizes as the input matrix. The evaluation matrices in Table 3 indicate that our model performs better than the two others. The ultimate goal of the research is to provide a tool for archaeology experts that they could load raw point cloud data acquired from an area of their choice, automatically see the results of the detection algorithm and find out locations of each object.

The result of the current approach is given in terms of a heat map. However, ultimately, we are interested in geometric delineations and description of the objects. In order to obtain them, additional analysis steps are needed, e.g. region growing or line following methods. We also plan to use parametric models in order to geometrically reconstruct objects such as charcoal stack locations. Such approaches have successfully been used in the domain of building reconstruction (e.g. [35]). Currently, we are able to detect only three classes: tracks, streams, and lakes, but everything else is labeled as 'others'. We incrementally label objects in the 'others' class and plan to include them in the training when sufficient number of instances are labeled for an object class.

The current approach uses a classification scheme where each pixel is classified with a filter mask and the classification is assigned to the pixels under that mask. This approach guarantees that appropriate environments of the individual pixels are taken into account for the classification. Future research will exploit other networks, especially semantic segmentation approaches, such as UNET (e.g. [36]). Such approaches have the advantage, that each pixel is labeled in the classification process (e.g. [37]).

Acknowledgements The project is funded by the Ministry of Science in Lower Saxony. The joint work with Kourosh Khoshelham has been supported by the DAAD. We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan X Pascal GPU used for this research.

References

1 Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

2 Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

3 Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 1(2):4, 2017.

4 Yushi Chen, Zhouhan Lin, Xing Zhao, Gang Wang, and Yanfeng Gu. Deep learning-based classification of hyperspectral data. *IEEE Journal of Selected topics in applied earth observations and remote sensing*, 7(6):2094–2107, 2014.

5 Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

6 Pim Moeskops, Max A Viergever, Adriënne M Mendrik, Linda S de Vries, Manon JNL Benders, and Ivana Išgum. Automatic segmentation of mr brain images with a convolutional neural network. *IEEE transactions on medical imaging*, 35(5):1252–1261, 2016.

7 Geoffrey E Hinton and Richard S Zemel. Autoencoders, minimum description length and helmholtz free energy. In *Advances in neural information processing systems*, pages 3–10, 1994.

8 Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

9 François Chollet. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint*, 2016.

10 Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

11 Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

12 Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

13 Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013.

14 Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

15 Junyoung Chung, Kyunghyun Cho, and Yoshua Bengio. A character-level decoder without explicit segmentation for neural machine translation. *arXiv preprint arXiv:1603.06147*, 2016.

16 Bashir Kazimi and Marta Ruiz Costa-Jussà. Coverage for character based neural machine translation. *Procesamiento del lenguaje natural (SEPLN)*, 59:99–106, 2017.

17 Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

18 Steve Lawrence, C Lee Giles, Ah Chung Tsoi, and Andrew D Back. Face recognition: A convolutional neural-network approach. *IEEE transactions on neural networks*, 8(1):98–113, 1997.

19 Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

20 Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 3d convolutional neural networks for human action recognition. *IEEE transactions on pattern analysis and machine intelligence*, 35(1):221–231, 2013.

21 Sijin Li and Antoni B Chan. 3d human pose estimation from monocular images with deep convolutional neural network. In *Asian Conference on Computer Vision*, pages 332–347. Springer, 2014.

22 Jaime Zabalza, Jinchang Ren, Jiangbin Zheng, Huimin Zhao, Chunmei Qing, Zhijing Yang, Peijun Du, and Stephen Marshall. Novel segmented stacked autoencoder for effective dimensionality reduction and feature extraction in hyperspectral imaging. *Neurocomputing*, 185:1 – 10, 2016.

23 Tiago Prado Oliveira, Jamil Salem Barbar, and Alexsandro Santos Soares. Multilayer perceptron and stacked autoencoder for internet traffic prediction. In Ching-Hsien Hsu, Xuanhua Shi, and Valentina Salapura, editors, *Network and Parallel Computing*, pages 61–71, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

24 Hasan Badem, Abdullah Caliskan, Alper Basturk, and Mehmet Emin Yuksel. Classification and diagnosis of the parkinson disease by stacked autoencoder. In *Electrical, Electronics and Biomedical Engineering (ELECO), 2016 National Conference on*, pages 499–502. IEEE, 2016.

25 Qin Zou, Lihao Ni, Tong Zhang, and Qian Wang. Deep learning based feature selection for remote sensing scene classification. *IEEE Geoscience and Remote Sensing Letters*, 12(11):2321–2325, 2015.

26 Gong Cheng, Peicheng Zhou, and Junwei Han. Learning rotation-invariant convolutional neural networks for object detection in vhr optical remote sensing images. *IEEE Transactions on Geoscience and Remote Sensing*, 54(12):7405–7415, 2016.

27 Xinchen Yan, Jimei Yang, Ersin Yumer, Yijie Guo, and Honglak Lee. Perspective transformer nets: Learning single-view 3d object reconstruction without 3d supervision. In *Advances in Neural Information Processing Systems*, pages 1696–1704, 2016.

28 Haiyan Guan, Yongtao Yu, Zheng Ji, Jonathan Li, and Qi Zhang. Deep learning-based tree classification using mobile lidar data. *Remote Sensing Letters*, 6(11):864–873, 2015.

29 Yongtao Yu, Haiyan Guan, and Zheng Ji. Automated detection of urban road manhole covers using mobile laser scanning data. *IEEE Transactions on Intelligent Transportation Systems*, 16(6):3258–3269, 2015.

30 Yongtao Yu, Jonathan Li, Haiyan Guan, Fukai Jia, and Cheng Wang. Learning hierarchical features for automated extraction of road markings from 3-d mobile lidar point clouds. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 8(2):709–726, 2015.

31 Xiangyun Hu and Yi Yuan. Deep-learning-based classification for dtm extraction from als point cloud. *Remote sensing*, 8(9):730, 2016.

32 Florian Politz, Bashir Kazimi, and Monika Sester. Classification of laser scanning data using deep learning. *38. Wissenschaftlich-Technische Jahrestagung der DGPF und PFGK18 Tagung*, 2018.

33 Leon F Palafox, Christopher W Hamilton, Stephen P Scheidt, and Alexander M Alvarez. Automated detection of geological landforms on mars using convolutional neural networks. *Computers & geosciences*, 101:48–56, 2017.

34 François Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015.

35 Claus Brenner. Building reconstruction from images and laser scanning. *International Journal of Applied Earth Observation and Geoinformation*, 6(3-4):187–198, 2005.

36 Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

37 Zhishuang Yang, Wanshou Jiang, Bo Xu, Quansheng Zhu, San Jiang, and Wei Huang. A convolutional neural network-based 3d semantic labeling method for als point clouds. *Remote Sensing*, 9(9):936, 2017.