Deep Learning for Classification Tasks on Geospatial Vector Polygons

R.H. van 't Veer* P. Bloem† E.J.A. Folmer‡

June 12, 2018

Abstract

In this paper, we evaluate the accuracy of deep learning approaches on geospatial vector geometry classification tasks. The purpose of this evaluation is to investigate the ability of deep learning models to learn from geometry coordinates directly. Previous machine learning research applied to geospatial polygon data did not use geometries directly, but derived properties thereof. These are produced by way of extracting geometry properties such as Fourier descriptors. Instead, our introduced deep neural net architectures are able to learn on sequences of coordinates mapped directly from polygons. In three classification tasks we show that the deep learning architectures are competitive with common learning algorithms that require extracted features.

1 Introduction

The ability to analyse vector shapes of geospatial objects is useful for many tasks, such as quality assessment or enrichment of map data (Fan et al, 2014) or the classification of topographical objects (Keyes and Winstanley, 1999). An increasingly more common method for shape analysis is through machine learning. For example, machine learning can be applied to assess correct building types (Xu et al, 2017) or classify road sections (Andrášik and Bíl, 2016). The prediction of house prices (Montero et al, 2018) and the estimation of pedestrian side walk widths (Brezina et al, 2017) are tasks that could possibly also benefit from the application of machine learning analysis on geometric shapes.

Current machine learning methods applied to geospatial vector data rely on extracting information from a geometry that characterizes its shape. This preprocessing step is known in machine learning as *feature extraction* (LeCun et al, 2015, 438) or *feature engineering* (Domingos, 2012, 84). The algorithms used by Andrášik and Bíl (2016) for example are trained on geometry properties

*Vrije Universiteit Amsterdam, Kadaster, Geodan: r.h.vant.veer@vu.nl
†Vrije Universiteit Amsterdam
‡University of Twente, Kadaster

based on angles and radii of vertices in road sections, extracted from simplified road geometries. The problem is that finding the geometry properties that are best suited for the machine learning task can be time-consuming, having a plethora of extraction methods at our disposal (Zahn and Roskies, 1972; Kuhl and Giardina, 1982; Zhang et al, 2002; Loncaric, 1998). Ideally, it would not be necessary to know the relevant properties of geospatial data in advance to obtain good predictions. With deep learning, we argue in this article, this is possible: we introduce machine learning methods to train on geospatial vector data without the need for feature extraction, by directly feeding the geometry coordinates to a deep learning model.

The 'deep' aspect of deep learning refers to the possibility of stacking multiple learning layers to form a model that is able to train latent representations at varying levels of data abstraction (LeCun et al, 2015). In this paper, will will use the term *shallow* machine learning to refer to methods that are *not* based on deep learning methods and that require feature extraction to learn a mapping from extracted features to the training labels. The motivation for using deep learning on geospatial vector data goes beyond matching or improving existing methods. Deep learning allows us to explore new methods for working with geospatial data, in complex pipelines involving combinations of raster, numerical and textual data (Ngiam et al, 2011), including geospatial vector data. Deep learning can be used for classification or regression tasks, but also for training generative models, producing new text (Sutskever et al, 2014), image (Goodfellow et al, 2014) and even vector shape (Ha and Eck, 2018) outputs.

One example of a more complex deep generative model involving vector shapes is by Ha and Eck (2018), using a model they named *sketch-rnn*. Sketch-rnn is instrumental in showing how a deep learning architecture can be used not only to classify, but also generate new vector shapes. The data collected for sketch-rnn used a web-based crowd-sourcing tool, inviting users to draw simple vector drawings of cats, t-shirts and a host of other object categories. Given an object category, the generative sketch-rnn model is able to analyse partial shapes drawn by the user and extrapolate these to complete sketches.[1] With sketch-rnn as a starting point, we can speculate on the possibilities of generative deep learning models for geospatial vector data: we can investigate how to generate building geometries directly from aerial photography, create area descriptions from neighbourhood area map data for the tourist industry, or create maps of house structures from archaeological excavation data.

However, from the research by Ha and Eck (2018) we do not yet know what the classification performance is compared to current shallow learning algorithms. Knowing how well deep learning models can learn directly from geometries is a first step in building more complex generative pipelines with confidence that the model is able to correctly interpret the data. The purpose of this article is to assess the accuracy of working with vector geometries in deep neural nets, by comparing them with existing shallow machine learning methods in an experiment with three classification tasks on vector polygons. In order to

[1] `https://magenta.tensorflow.org/assets/sketch_rnn_demo/index.html`

work with geometries in deep neural nets, we need to investigate whether deep learning models perform at accuracy levels at least on par with current shallow methods. Thus, the main question we want to answer is this: how well do deep learning models learn directly from geometries, without extracting data as a proxy?

1.1 Contributions, disambiguation and article structure

The first contribution of this paper is in showing that a deep neural net is competitive in classification tasks involving geometries, without need for feature extraction. We designed a series of experiments to investigate the performance of shallow and deep learning methods on geospatial vector data. The experiments involve three classification tasks restricted to polygons, with performance scores measured in accuracy. For our baseline methods, we evaluate models trained on generic shape description methods using Fourier descriptors (see Section 4) rather than data set specific properties in order to compare performance across a set of tasks. The claim in this study is that the deep learning models introduced here match baseline methods in accuracy at classification tasks on real-world geospatial polygon data.

The second contribution is in providing a new benchmark on geospatial vector shape recognition. We hope to establish this benchmark for use by anyone in the geospatial domain and we encourage others to re-use, improve and publish these methods. We release all preprocessing code, deep learning models, baseline models as open source software[2] and the benchmark data files as open data.[3]

Since the domains of geospatial information systems (GIS) and machine learning (ML) have partially overlapping vocabularies, we provide Table 1 of homonyms and their use in the two fields of GIS and ML. Where used in this article, the terms are clarified by their field or, where possible, avoided.

The further article structure is as follows: we explain the classification tasks in Section 2, discuss the introduced deep learning methods in Section 3, and discuss the experiment and its results in Section 4.

2 Tasks

We created a set of experimental classification tasks to evaluate the performance of several machine learning algorithms in shape recognition on geospatial (multi)polygons. For this purpose, we formulated three tasks, with varying levels of polygon complexity and class distributions. We chose data sets that have enough data to draw conclusions on model generalization; we set a requirement for data sets to provide a 10% test subset of at least 1000 geometries. Also, the data sets were chosen for containing geometry shapes that are likely to provide some class information but not a trivial solution. Further considerations for the

[2]Code available at `https://github.com/SPINlab/geometry-learning`
[3]Data available at `http://hdl.handle.net/10411/GYPPBR`

Term	GIS	ML
Vector	A geometry defined by vertices and edges	A one-dimensional array
Vectorization	Conversion of raster or analog data into geospatial vector geometries	Conversion of data into a tensor interpretable by a machine learning algorithm
Feature	A geospatial object	A data property
Shape	A geospatial object	A tensor size along its dimensions
K-nearest neighbours	The k spatially closest objects	A learning algorithm based on closest resemblance

Table 1: Terms in the fields of GIS and ML

chosen data sets were to test on geodata from different domains, with different use cases and on different spatial scales:

1. Predicting the number of inhabitants in a neighbourhood to be above or below the national median, based on the neighbourhood geometry;
2. Predicting a building class from its geometry;
3. Predicting an archaeological feature type from its geometry.

The classes and frequencies are displayed in Table 2. We will now discuss each of these benchmark tasks in more detail.

2.1 Neighbourhood inhabitants

The first task in the set is to predict the number of inhabitants for a certain neighbourhood to be above or below the national median in the Netherlands. A neighbourhood is a geographical region as defined by Statistics Netherlands.[4] For the sake of simplicity, the task has been shaped into a binary class prediction: to predict a neighbourhood for having equal or more (6,610 neighbourhoods) or less (6,598 neighbourhoods) than the median of inhabitants of the entire set of neighbourhoods for the Netherlands for the year 2017. The median was chosen to create a near-even[5] split of the two classes.

[4] `https://www.cbs.nl/-/media/_pdf/2017/36/2017ep37%20toelichting%20wijk%20en%20buurtkaart%202017.pdf` (only available in Dutch).

[5] The difference between the class frequencies is explained by a slightly uneven frequency distribution of neighbourhoods over the number of inhabitants.

Neighbourhood inhabitants		Buildings		Archaeological features	
Class	frequency	Function	frequency	Class	frequency
$\geq$ median	6,610	Habitation	23,000	Posthole	26,359
$<$ median	6,598	Industrial	23,000	Pit	8,241
Total	60,623	Lodging	23,000	Natural phenomenon	6,284
		Shopping	23,000		
		Gatherings	22,007	Recent disturbance	5,762
		Office	21,014		
		Education	10,717	Ditch	5,232
		Healthcare	7,832	Wooden object	2,547
		Sports	6,916		
		Total	160,486	Layer	2,380
				Wall	1,419
				Posthole with visible post	1,237
				Water well	1,162
				Total	60,623

Table 2: Class frequency for the three tasks of neighbourhood inhabitants (left), building types (middle) and archaeological feature types (right)

2.2 Building types

The second task is to classify a building from the building footprint geometry. The data set consists of buildings in nine functional classes with a less uniform class distribution than the previous, for a total of 160,486 buildings in the combined training and test dataset. Since the complete source data set comprises over five million buildings, each class was trimmed to a maximum of 23,000 instances per class to prevent creating a data set too large to experiment on. With this maximum, the buildings set still has the most data of the three tasks.

2.3 Archaeological features

The third task is to classify an archaeological feature from its observed geometry. Archaeological features are field observations of disturbances in the subsoil as a result of human activities in the past. Archaeological institutions in the Netherlands store the results of archaeological field research in a digital repository (Gilissen, 2017; Hollander, 2014). From the DANS EASY repository,[6] from ten archaeological projects (Roessingh and Lohof, 2010; Gerrets and Jacobs, 2011; Van der Veken and Prangsma, 2011; Dijkstra and Zuidhoff, 2011; Dijkstra et al, 2010; van de Velde et al, 2002; van der Velde, 2011; Dijkstra, 2012; Roessingh and Blom, 2012; Van der Veken and Blom, 2012), a total of

[6] `https://easy.dans.knaw.nl`

60623 geometries was collected in ten classes. The class distribution for this dataset is the most unbalanced of the three tasks: the data shows a clear over-representation of post holes (43,5 %).

3 Models and preprocessing

We introduce two deep learning, end-to-end trained models where vector-serialized geometries are given as input data—the neural net figures out the relevant data properties for itself. Before we explain the deep learning models, we discuss in detail the machine learning vector format to which the geometries are mapped.

3.1 Geometries as machine learning vectors

The classification tasks in this paper operate on data from (multi)polygons. To be precise, we use the term polygon to mean a single connected sequence (i.e. without polygon holes) of three or more coplanar lines. Every line in a polygon is defined by two points in $\mathbb{R}^2$, where each point is shared by exactly two lines to form a closed loop. We impose no validity constraint on polygons, i.e. polygons may be self-intersecting.

For our deep neural net architectures, geometries are expressed as input vector sequences. This method was derived from the method devised by Ha and Eck (2018). Each geometry sample G_i in a data set of size n is encoded as a sequence of geometry vertex vectors: $\langle \mathbf{g_1^i}, \mathbf{g_2^i}, \mathbf{g_3^i}, \ldots, \mathbf{g_m^i} \rangle$, where m is the number of vertices in the geometry. Each vector $\mathbf{g_j^i}$ is a concatenation of:

- a coordinate point vector $\mathbf{p_j^i}$ in $\mathbb{R}^2$ for longitude and latitude. Both longitude and latitude are generally bounded by the interval $[-360, 360]$, but our data is located in the Netherlands, bounded by the interval $[3.2, 7.22]$ for longitude and $[50.75, 53.7]$ for latitude.[7]
- A one-hot vector $\mathbf{r_j^i}$ in $\mathbb{R}^3$ to mark the end of either the point, sub-geometry or a final stop for the vertices in G^i. For each $\mathbf{g_j^i}$ in a polygon geometry, $\mathbf{r_j^i} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$ except for the last vertex, where $\mathbf{r_m^i}$ marks the end of the polygon as the final stop $\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$. In case of a multipolygon, each subpolygon is terminated by a subgeometry stop $\begin{bmatrix} 0 & 1 & 0 \end{bmatrix}$ except for the last, which is marked as a final stop.

Combined, $\mathbf{g_j^i}$ is a vector of length 5, as shown in Figure 1.

Geospatial coordinates are often expressed in degrees of longitude and latitude, where one degree of latitude equals roughly 111 kilometres. However, the tasks in our experiments operate on the level of meters or, in the case of the archaeology task, even centimetres. For the neural net to make sense of the data, the coordinate data is normalized. For every point vector $\mathbf{p_j^i}$ in every

[7] `https://epsg.io/28992`

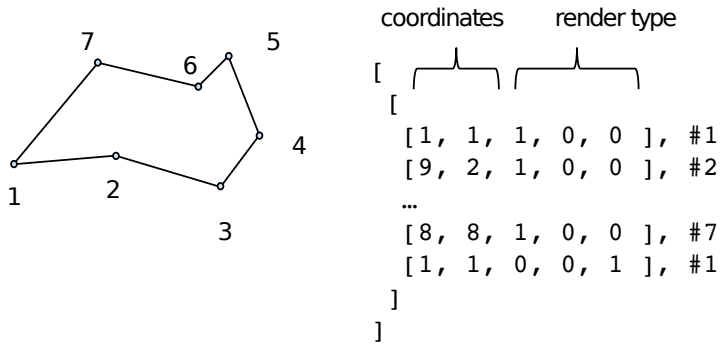

Figure 1: Graphical and vector representations of a polygon.

geometry G^i, $\mathbf{p_j^i}$ is normalized to

$$\mathbf{p_j^{i\prime}} = \frac{\mathbf{p_j^i} - \overline{\mathbf{p^i}}}{s}, \tag{1}$$

where $\overline{\mathbf{p^i}}$ is the geometry centroid of geometry G^i, computed as the mean average of all $\mathbf{p^i}$ in a *single* geometry G^i. Scale factor s is the standard deviation over the bounding values b^i_{min} and b^i_{max} of *all* geometries. b^i_{min} and b^i_{max} for a geometry G^i are defined as

$$b^i_{min} = \min(\mathbf{p^i} - \overline{\mathbf{p^i}}), \tag{2}$$

and

$$b^i_{max} = \max(\mathbf{p^i} - \overline{\mathbf{p^i}}). \tag{3}$$

This is a simpler two-value version of the standard bounding box that would normally list the minimum and maximum values for a geometry in two dimensions. Scale factor s is then computed as the scalar standard deviation over all bounding values B:

$$B = \langle b^1_{min}, b^1_{max}, b^2_{min}, b^2_{max}, \ldots, b^n_{min}, b^n_{max} \rangle \tag{4}$$

Geometries often vary in the number of vertices required to approximate the shape of a real-world object. As a consequence, the geometry vector sequences vary in length. Deep learning models have the benefit of being able to train and predict on variable length sequences (Bahdanau et al, 2014). However, within one batch the sequences need to be of the same length in order to uniformly apply the model weights and biases on the entire batch as a single tensor. To achieve this fixed sequence size within a batch, the geometry vectors are first sorted in reverse order, with the largest geometry first and the smallest last. This sorted set of geometries is subdivided into bins of size n_{bin}, where n_{bin} is at least the training batch size. This is to increase computational efficiency and

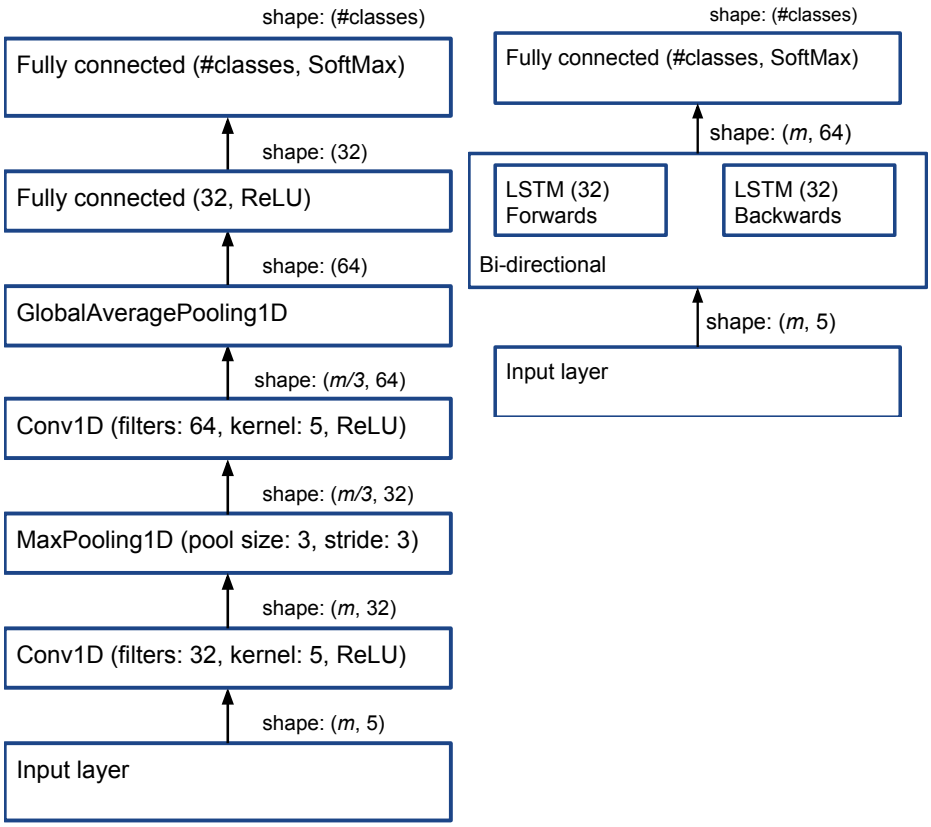

Figure 2: Convolutional (left) and recurrent (right) model layouts.

reduce training time on what otherwise would be a large array of very small batches. If there are insufficient geometries of sequence length m_{bin} to create a set of samples of batch size, smaller geometries are added and padded to sequence length m_{bin}. Thus, a geometry with a sequence length m of 144 points is zero-padded to a size m_{bin} of 148 if the largest sequence length in the batch is 148. This preprocessing of binning and limited padding reduced the training time to one quarter of the time needed for training on fixed size sequences.

Although there is no theoretical upper bound to the sequence length, there is a practical one for the amount of memory on commodity hardware. The data sets contain a small amount of very large geometries. To improve computational efficiency and prevent memory errors, these rare cases are simplified using the Douglas-Peucker algorithm (Douglas and Peucker, 1973), implemented in the python *Shapely* package.[8] In this way, only 0.17 percent of the geometries needed to be simplified.

[8] https://pypi.python.org/pypi/Shapely

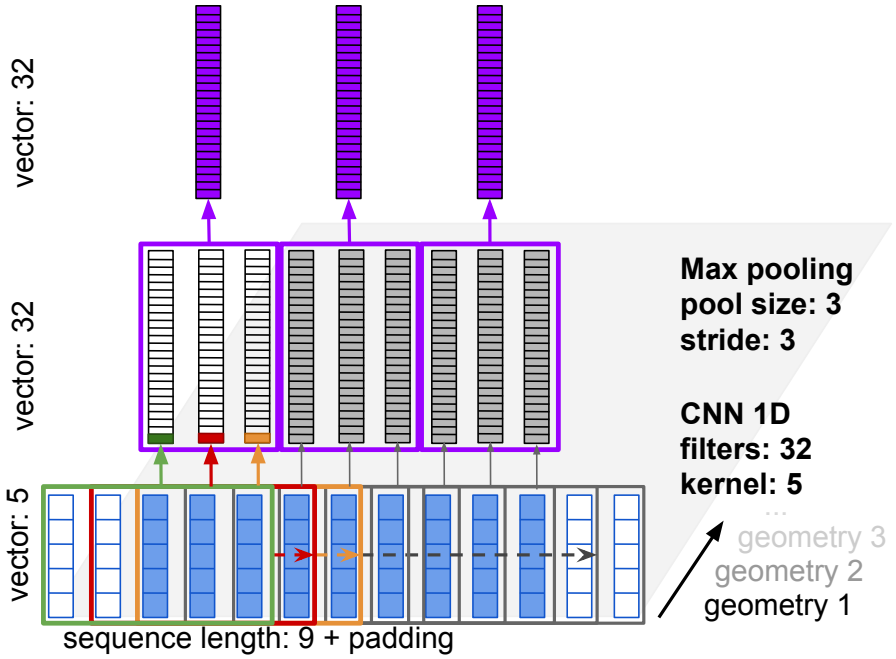

Figure 3: The first two layers of the CNN model. With a kernel size of five, the CNN inspects a sliding window over the first five geometry vectors in geometry G^1, producing the green element in the CNN output vector. The CNN then moves to the five elements to the right, and produces the red vector element, next the orange vector, repeated until the end of the geometry (the next three windows in grey). This process is repeated for each filter and then moves to the next geometry, in the direction of the black arrow. The max pooling operation combines the maximum output element values of the CNN, shown in purple for geometry 1.

3.2 Convolutional neural net

The first introduced deep learning model uses a 1D convolutional neural net (CNN) layout, shown in Figure 2. For an introduction to the workings of the CNN, we refer the reader to Olah (2014). As a first layer, our model uses a ReLU-activated convolution layer with a filter size of 32, a kernel size of five and a stride of one. With this configuration, the CNN starts a sliding window across the first five geometry vectors, i.e. g_1^i through g_5^i, producing a vector of size 32 as specified by the filter hyperparameter.[9] This window of size five is slid along the vectors of the geometry, until the end of the geometry including padding. Padding ensures outputs by the CNN of the same sequence length as the input, to prevent size errors on small geometries where the tensor size

[9]Hyperparameters are the configuration settings of the machine learning model that aren't optimized during training, such as the batch size.

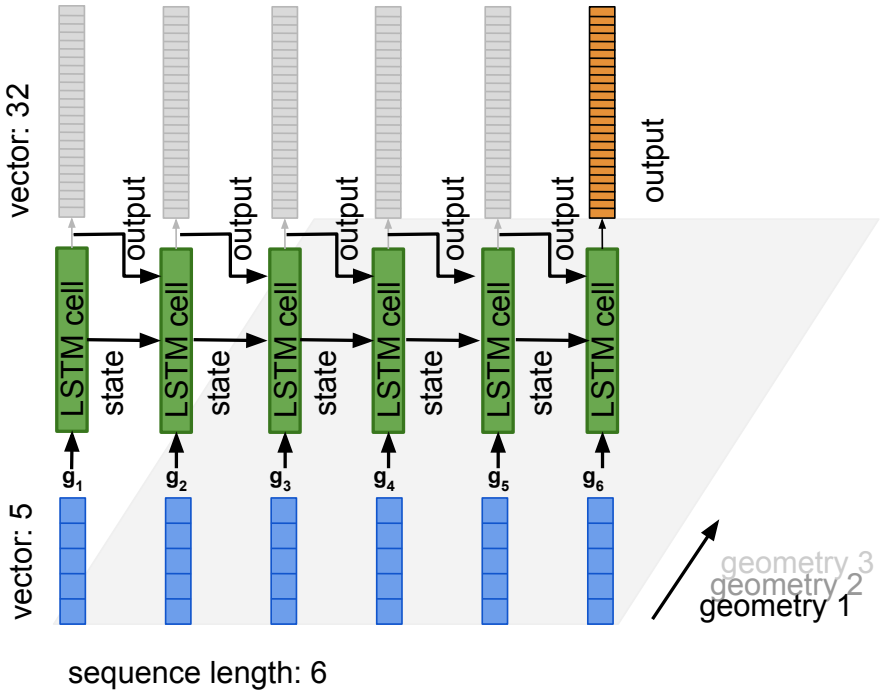

Figure 4: Forward-facing LSTM, as part of the first layer of the proposed LSTM model. Unlike the CNN architecture, complete vectors are fed one by one to the same LSTM cell. The green boxes therefore represent the same cell, with only its state updated: along with each next geometry vector, the output and previous state of the LSTM cell are passed along from one vector to the next. For the purposes of classification as in this article, only the last LSTM output is returned (in orange), the intermediate outputs (in grey) are discarded.

becomes too small to pass through the specified network layers. After g_1^i through g_5^i the CNN continues at the second set of geometry entries g_2^i through g_6^i. After inspecting all values of all the vectors in the first geometry, the CNN continues at the next geometry (see Figure 3).

The first CNN layer is followed by a max pooling layer with a pooling size of three and a stride of three. The max pooling operation with a pool size of three combines the maximum values of three CNN output vectors into a single sequence vector of the same length. The reduction of the CNN output to one-third is specified by the max pooling *stride* hyperparameter: after combining CNN output vectors $\mathbf{c_1^i}$, $\mathbf{c_2^i}$ and $\mathbf{c_3^i}$, the max pooling operation skips forward to combine outputs $\mathbf{c_4^i}$, $\mathbf{c_5^i}$ and $\mathbf{c_6^i}$, and so on. After the max pooling layer, a second convolution layer (not shown in Figure 3) interprets the output of the max pooling layer, with hyperparameters identical to the first but with 64 filters instead of 32. This CNN layer is followed by a global average pooling layer that

reduces the tensor rank to two by computing the average over the third tensor axis. The output is subsequently fed to a ReLU activated fully connected layer. The last layer is a softmax-activated fully connected layer to produce probability outputs that sum to one.

3.3 Recurrent neural net

The second introduced deep learning model uses a recurrent neural net (RNN) layout as shown in Figure 2. Core of the model is a single bi-directional LSTM (Hochreiter and Schmidhuber, 1997) layer. The Long-Short Term Memory (LSTM) architecture is a particular type of RNN, designed to process sequences of data with a trainable *forget gate*. This forget gate regulates the information retained from one geometry vertex to the next. During training, the LSTM learns which information in the sequence is of interest to retain, by passing both the vector in the sequence, the output and the cell state from one input vector to the next (see Figure 4). For a detailed discussion of recurrent neural nets and LSTMs in the geospatial domain, we refer the reader to Mou et al (2017), an introduction to LSTMs is given by Olah (2015). LSTMs have been shown to be effective on sequences such as words in a sentence (Sutskever et al, 2014), but also sequential geometry-like data such as handwriting recognition and synthesis (Graves, 2013). The ability of the LSTM to learn long-term dependencies (Goodfellow et al, 2016, 400) in sequences of input data renders it a suitable architecture to test its abilities on geospatial vector geometries.

The bi-directional architecture feeds the sequence of geometry vertices forwards as well as backwards through LSTM cells (Schuster and Paliwal, 1997), where the resulting output from these cells is concatenated. This allows the network to learn from the preceding vertices in the geometry as well as the ones that are ahead. In our set-up, we configured both the forwards and backwards LSTM to produce an output of size 32, combined to 64. As with the CNN model, the last layer is a softmax-activated fully connected layer to produce probability outputs that sum to one.

4 Experiment and evaluation

To evaluate the deep learning model performance, we compared the proposed deep learning models with a set of shallow machine learning baseline algorithms. The experiment consists of the tasks and data described in Section 2, comparing the deep learning models with the following baselines:

- Majority class: the fraction of the prevalent class, included as an indication of the most simple method to exceed;
- Support Vector Machine (SVM) with Radial Basis Function (RBF) kernel. Other kernels (linear, polynomial) were tested but RBF always produced better results;
- Logistic regression;

- K-nearest neighbour classifier;
- Decision tree classifier.

Scikit-learn (Pedregosa et al, 2011) provided the baseline shallow learning algorithms. For each task, a brute force grid search with 5-fold cross validation was used to find the best applicable hyperparameters for k (k-nearest neighbours), degree (decision tree), C (SVM, logistic regression) and gamma (SVM). SVM grid searches were restricted to a maximum number of 10M iterations to allow the grid search operation to complete within a day. Grid searches on SVM models and k-nearest neighbours were restricted to a subset of the training data to allow the grid search to finish within a day on commodity hardware. All models were trained, however, using the full training set on the the best hyperparameters obtained from the grid search.

The deep learning models are implemented using Keras (Chollet et al, 2015) version 2 with a TensorFlow (Abadi et al, 2016) version 1.7 backend. All deep learning hyperparameters were tuned on training data only, using validation data split randomly from the training data.

We discuss the preprocessing for the baseline methods, the results and the evaluation in detail below.

4.1 Baseline preprocessing

Contrary to the introduced deep learning models, the baseline methods in the experiment do not operate on geometry coordinates directly, but on Fourier descriptors derived from the geometries. Fourier descriptors are a common choice as a feature engineering method for extracting properties from geometries (Zhang et al, 2002; Keyes and Winstanley, 1999; Zahn and Roskies, 1972; Loncaric, 1998). We restricted this study to methods that were available through open source libraries. The Fourier descriptors were constructed using the *pyefd* package,[10] which implements the algorithms by Kuhl and Giardina (1982). Elliptic Fourier descriptors are created by iterating over the coordinates of the vertices in a geometry, transforming any number of coordinates of the geometry into a vector representing the geometry in an elliptic approximation. This transformation can be reversed, producing an approximation of the original geometry, its reconstructive accuracy depending on the *order* or the number of *harmonics* (Kuhl and Giardina, 1982, 239), the order or number being a positive integer. The higher the order, the better the approximation gets, as shown in Figure 5.

The *pyefd* package produces normalized and non-normalized descriptors; the normalized descriptors are start position, scale, rotation and translation invariant (Kuhl and Giardina, 1982, 236). For the data used in training the baseline models, both normalized and non-normalized Fourier descriptors were included. For the grid search, we did not assume that including an arbitrary high number of descriptors would produce the best accuracy score. Instead, the number of

[10] https://pypi.python.org/pypi/pyefd

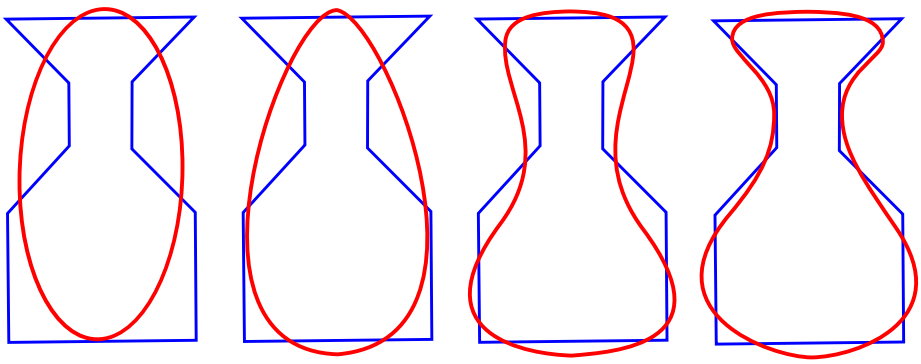

Figure 5: Order 1, 2, 3 and 4 elliptic Fourier reconstruction approximations (red) of a vase-shaped polygon (blue). Each order level adds to the approximation. Adapted from Kuhl and Giardina (1982, 237)

extracted Fourier descriptors used during training was included as a hyperparameter in the grid search for each baseline model, to produce the descriptor order at which the grid search obtained the best results. The best parameters found in the grid searches are listed in Table 4 in Appendix A.

Added to the descriptors are three other easily obtained geometry properties: the polygon surface area, number of vertices and geometry boundary length. These three properties were included in each grid search for all baseline models.

4.2 Results and evaluation

The accuracy scores from the experiment allow us to compare the performance of our introduced deep learning models against the baseline shallow learning models. Table 3 shows the results for each of the three benchmark tasks. The figures were produced from model predictions on the test set, consisting of 10% of the geometries in the data set that were unseen by the models during training. The deep learning model experiments were repeated ten-fold: randomized network initialisation and batch sampling produce slight variations in accuracy scores between training sessions. The accuracy figures for the deep neural models therefore represent mean and standard deviation from the test predictions on the independently repeated training sessions.

The evaluation accuracy figures in Table 3 allow for several conclusions to be drawn:

1. The introduced deep neural nets are at least competitive with the best baseline models, for each of the three tasks. In five out of six cases (the LSTM on the neighbourhoods task excepted), the deep models perform on par with or slightly better than the best baseline models, but in the broad sense they do not significantly outperform the shallow models by a wide margin.

Method	Task (no. of classes)		
	Neighbourhood inhabitants (2)	Building types (9)	Archaeological feature types (10)
Majority class	0.500	0.143	0.435
k-NN	0.671	0.377	0.596
Logistic regression	0.659	0.328	0.555
SVM RBF	**0.683**	0.365	0.601
Decision tree	0.682	0.389	0.615
CNN	0.664 ± 0.005	$\mathbf{0.408 \pm 0.003}$	$\mathbf{0.624 \pm 0.002}$
RNN	0.608 ± 0.016	0.389 ± 0.008	0.614 ± 0.004

Table 3: Table of results with accuracy scores for the introduced deep learning models (bottom two rows) and the baseline models (top five rows), with the best scores per task in bold. The number of classes per task is listed between brackets in the column headers. The standard deviations on the deep learning models on the bottom two rows were obtained from test set predictions on ten-fold repeated, independent training sessions.

2. On two of the three classification tasks, the CNN architecture is able to outperform the baseline models by a few percentage points. If top performance in a certain geometry classification task is required, the CNN is likely to be a good choice.

3. The best overall performing baseline model is the decision tree. An advantage of using decision trees is that they are very fast to train.

As mentioned in Section 3.1, the number of elliptic Fourier descriptors used for training the baseline models was included as a hyperparameter in the grid search. A closer inspection of these optimal hyperparameters in Appendix A for the baseline models is of interest:

1. Nearly all baseline models benefit from adding Fourier descriptors. A notable exception is the k-nearest neighbours algorithm, which scores the highest accuracy in two of the three tasks only when no Fourier descriptors are added to the training data. As it appears from the three tasks, the k-NN algorithm is less able to extract meaningful information from the Fourier descriptors.

2. The baseline models have a clear preference for lower orders of Fourier descriptors. Even though many higher orders (up to order 24) were tested, no baseline algorithm was able to perform better on descriptor orders higher than four. Order four descriptors only provide a very rough approximation of the original geometry, as is well visualised in Kuhl and Giardina's paper Kuhl and Giardina (1982, 243) and Figure 5. Still, the descriptors evidently contain enough important shape information for most baseline algorithms to improve the accuracy score.

3. Support vector machines come with a misclassification tolerance hyperparameter C. SVMs with a preference for high C settings (low tolerance), such as the ones for the archaeology classification task, were exceedingly time-consuming to train on our data. Where low C-values tended to converge in seconds, high values could literally take days or even weeks to converge. To prevent having to wait for extended periods of time—there is no indication in what time frame a training session on a set of hyperparameters will converge—we needed to constrain the amount of training data and the maximum of iterations, especially on hyperparameter grid searches. It is quite possible that as a consequence of these constraints, the grid search fails to produce the optimal hyperparameter settings, but this is an unfortunate side effect of using SVMs on Fourier descriptors of geometries.

5 Conclusion and future research

In this paper, we compared the accuracy of deep learning models against baselines of shallow learning methods on three new classification tasks involving only geometries. This article shows that deep neural nets are able to perform classification tasks on geospatial vector data directly with accuracy results that are competitive with established baseline methods. For all tasks tested, the deep learning models show performance that is competitive with the shallow models, in some cases outperforming them by a small margin.

None of the chosen recognition tasks appear to be trivial. Classifying objects from geometries alone is a tough assignment for any algorithm. The advantage of having a set of tough tasks is that these can serve as a benchmark: in future experiments, different learning algorithms or model layouts it may be possible to obtain higher accuracy scores. However, there is a possibility that the accuracy figures presented in the evaluation represent the maximum that can be learned from geometries alone. From these experiments alone it cannot be deduced whether these figures can actually be improved on. If there is a hard ceiling at the best performing models, perhaps the benchmarks can be improved by including more data than just the geometries alone, for example information gathered from the direct spatial surroundings or other properties of the spatial objects. The benchmark presented here can be considered a first attempt.

An area that might see improvement is the performance of LSTMs. In an earlier development stage, the LSTMs were trained on fixed length rather than on the variable length sequences. During this stage, the LSTMs performed significantly better (on validation data, no final tests were performed) on fixed length sequences, outperforming the CNNs. Training on fixed length sequences was abandoned because it requires simplifying geometries to a fixed maximum of points per geometry. Simplification causes loss of information which was detrimental to our intention to train on complete geometries. Creating fixed length sequences also required adding a large amount of zero-padding to increase sequence length on all geometries shorter than the fixed size. After switching

to variable length sequences, the performance of the CNN models increased and the LSTM performance dropped considerably. We hypothesize that there is room to improve the LSTM model configuration to CNN model performance or perhaps even better. To test this in future research, the fixed length sequences were included in the benchmark data.

There are several roads to further explore the use of deep learning models for geometries. It would be helpful to verify the accuracy on other types of geometries, such as multi-lines, multi-points or even heterogeneous geometry collections. Also, the deep neural net's comprehension of holes in polygons could be beneficial, this was outside the scope of this paper. Another interesting road to explore is to combine different information sources into machine learning tasks. This paper is a step in that direction, by showing that is possible to have a deep neural net learn from geometries directly. Deep learning poses a viable route to explore more complex pipelines involving geometries and other multi-modal input, to produce sequences (Sutskever et al, 2014), images (He et al, 2017) or other generative data (Ha and Eck, 2018) as output.

Acknowledgements

This work was supported by the Dutch National Cadastre (Kadaster) and the Amsterdam Academic Alliance Data Science (AAA-DS) Program Award to the UvA and VU Universities. We would also like to thank the following organisations. The source data for the neighbourhoods task is published by Statistics Netherlands (CBS) and distributed by the Publieke Dienstverlening op de Kaart organization (PDOK) under a Creative Commons (CC) Attribution license. The data for the buildings task was published by the Dutch National Cadastre under a CC Zero license. The archaeological data in raw form is hosted by Data Archiving and Networked Services, and re-licensed by kind permission of copyright holder ADC ArcheoProjecten under CC-BY-4.0. A small subset of the data cited in paragraph 3.3 was used, only geometries and type labels were redistributed. We thank Henk Scholten, Frank van Harmelen, Xander Wilcke, Maurice de Kleijn, Jaap Boter, Chris Lucas, Eduardo Dias, Brian de Vogel and anonymous reviewers for their helpful comments.

Appendices

A Hyperparameter grid search results for baseline models

The grid searches discussed in Section 4 resulted in a set of best hyperparameter settings for the baseline models. These best settings are listed in Table 4 and include the ranges that were searched. The range for the elliptic fourier descriptor order o is always the same: each grid search was executed on the orders

Method	Task		
	Neighbourhood inhabitants (2)	Building types (9)	Archaeological feature types (10)
Decision tree	o=2 d=6 in $[4, 9]$	o=3 d=10 in $[6, 12]$	o=3 d=9 in $[5, 10]$
k-NN	o=1 k=26 in $[21, 30]$	o=0 k=29 in $[21, 30]$	o=0 k=29 in $[21, 30]$
SVM RBF	o=1 C=1 in 1e$[-2, 3]$ γ=1 in 1e$[-3, 3]$	o=0 C=1000 in 1e$[-2, 3]$ γ=10 in 1e$[-2, 3]$	o=2 C=100 in 1e$[-1, 3]$ γ=0.01 in 1e$[-4, 4]$
Logistic regression	o=1 C=0.01 in 1e$[-3, 1]$	o=4 C=1 in 1e$[-2, 3]$	o=8 C=1000 in 1e$[-2, 3]$

Table 4: Hyperparameters for baseline methods. Interval values for decision tree and k-nearest neighbours $\in \mathbb{N}$, for the SVM in log scale, with the exponent interval $\in \mathbb{N}$.

$\langle 0, 1, 2, 3, 4, 6, 8, 12, 16, 20, 24 \rangle$. The search intervals for the other hyperparameters are listed in Table 4. The k-hyperparameter of the k-nearest neighbour models and the maximum depth d hyperparameter for the decision tree have intervals with values $\in \mathbb{N}$. For the other hyperparameters, the listed interval values are powers of ten, as indicated by the scientific notation.

References

Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, Corrado GS, Davis A, Dean J, Devin M, Ghemawat S, Goodfellow IJ, Harp A, Irving G, Isard M, Jia Y, Józefowicz R, Kaiser L, Kudlur M, Levenberg J, Mané D, Monga R, Moore S, Murray DG, Olah C, Schuster M, Shlens J, Steiner B, Sutskever I, Talwar K, Tucker PA, Vanhoucke V, Vasudevan V, Viégas FB, Vinyals O, Warden P, Wattenberg M, Wicke M, Yu Y, Zheng X (2016) Tensorflow: Large-scale machine learning on heterogeneous distributed systems. CoRR abs/1603.04467, URL `http://arxiv.org/abs/1603.04467`

Andrášik R, Bíl M (2016) Efficient road geometry identification from digital vector data. Journal of Geographical Systems 18(3):249–264, DOI 10.1007/s10109-016-0230-1, URL `https://doi.org/10.1007/s10109-016-0230-1`

Bahdanau D, Cho K, Bengio Y (2014) Neural machine translation by jointly learning to align and translate. CoRR abs/1409.0473, URL `http://arxiv.org/abs/1409.0473`, `1409.0473`

Brezina T, Graser A, Leth U (2017) Geometric methods for estimating representative sidewalk widths applied to vienna's streetscape surfaces database. Journal of Geographical Systems 19(2):157–174, DOI 10.1007/s10109-017-0245-2, URL https://doi.org/10.1007/s10109-017-0245-2

Chollet F, et al (2015) Keras. https://github.com/fchollet/keras

Dijkstra J (2012) Wijk bij Duurstede veilingterrein do opgraving. https://doi.org/10.17026/dans-x8d-qmae, DOI 10.17026/dans-x8d-qmae

Dijkstra J, Zuidhoff F (2011) Veere rijksweg N57 proefsleuven begeleiding opgraving. https://doi.org/10.17026/dans-xyc-re2w, DOI 10.17026/dans-xyc-re2w

Dijkstra J, Houkes M, Ostkamp S (2010) Gouda Bolwerk opgraving en begeleiding. https://doi.org/10.17026/dans-xzm-x29h, DOI 10.17026/dans-xzm-x29h

Domingos P (2012) A few useful things to know about machine learning. Communications of the ACM 55(10):78–87, URL https://dl.acm.org/citation.cfm?id=2347755

Douglas DH, Peucker TK (1973) Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. Cartographica: The International Journal for Geographic Information and Geovisualization 10(2):112–122, DOI 10.3138/FM57-6770-U75U-7727, URL https://doi.org/10.3138/FM57-6770-U75U-7727

Fan H, Zipf A, Fu Q, Neis P (2014) Quality assessment for building footprints data on openstreetmap. International Journal of Geographical Information Science 28(4):700–719

Gerrets D, Jacobs E (2011) Venlo TPN deelgebied 1 en 2 opgraving. https://doi.org/10.17026/dans-26f-55zu, DOI 10.17026/dans-26f-55zu

Gilissen V (2017) Archiving the past while keeping up with the times. Studies in Digital Heritage 1(2):194–205, DOI 10.14434/sdh.v1i2.23238, URL https://doi.org/10.14434/sdh.v1i2.23238

Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A, Bengio Y (2014) Generative adversarial nets. In: Ghahramani Z, Welling M, Cortes C, Lawrence ND, Weinberger KQ (eds) Advances in Neural Information Processing Systems 27, Curran Associates, Inc., pp 2672–2680, URL http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

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

Graves A (2013) Generating sequences with recurrent neural networks. CoRR abs/1308.0850, URL http://arxiv.org/abs/1308.0850, 1308.0850

Ha D, Eck D (2018) A neural representation of sketch drawings. In: International Conference on Learning Representations, URL https://openreview.net/forum?id=Hy6GHpkCW

He K, Gkioxari G, Dollár P, Girshick RB (2017) Mask R-CNN. CoRR abs/1703.06870, URL http://arxiv.org/abs/1703.06870, 1703.06870

Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural computation 9(8):1735–1780, DOI 10.1162/neco.1997.9.8.1735, URL https://doi.org/10.1162/neco.1997.9.8.1735

Hollander H (2014) The e-depot for dutch archaeology. archiving and publication of archaeological data. In: Conference on Cultural Heritage and New Technologies (CHNT), Vienna, Stadt Archäologie Wien

Keyes L, Winstanley AC (1999) Fourier descriptors as a general classification tool for topographic shapes. IPRCS, pp 193–203, URL http://eprints.maynoothuniversity.ie/66/

Kuhl FP, Giardina CR (1982) Elliptic fourier features of a closed contour. Computer graphics and image processing 18(3):236–258, DOI 10.1016/0146-664X(82)90034-X, URL http://dx.doi.org/10.1016/0146-664X(82)90034-X

LeCun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521(7553):436–444, DOI doi:10.1038/nature14539, URL http://dx.doi.org/10.1038/nature14539

Loncaric S (1998) A survey of shape analysis techniques. Pattern Recognition 31(8):983 – 1001, DOI https://doi.org/10.1016/S0031-2023(97)00122-2, URL http://www.sciencedirect.com/science/article/pii/S0031202397001222

Montero JM, Mínguez R, Fernández-Avilés G (2018) Housing price prediction: parametric versus semi-parametric spatial hedonic models. Journal of Geographical Systems 20(1):27–55, DOI 10.1007/s10109-017-0257-y, URL https://doi.org/10.1007/s10109-017-0257-y

Mou L, Ghamisi P, Zhu XX (2017) Deep recurrent neural networks for hyperspectral image classification. IEEE Transactions on Geoscience and Remote Sensing 55(7):3639–3655, DOI 10.1109/TGRS.2016.2636241, URL http://dx.doi.org/10.1109/TGRS.2016.2636241

Ngiam J, Khosla A, Kim M, Nam J, Lee H, Ng AY (2011) Multimodal deep learning. In: Proceedings of the 28th international conference on machine learning (ICML-11), pp 689–696, URL http://ai.stanford.edu/~ang/papers/icml11-MultimodalDeepLearning.pdf

Olah C (2014) Conv nets: A modular perspective. URL https://colah.github.io/posts/2014-07-Conv-Nets-Modular/

Olah C (2015) Understanding lstm networks. URL https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E (2011) Scikit-learn: Machine learning in Python. Journal of Machine Learning Research 12:2825–2830, URL http://www.jmlr.org/papers/v12/pedregosa11a.html

Roessingh W, Blom E (2012) Oosterhout Vrachelen de Contreie Vrachelen 4 opgraving. https://doi.org/10.17026/dans-25d-fpe5, DOI 10.17026/dans-25d-fpe5

Roessingh W, Lohof E (2010) Enkhuizen Kadijken 5a en 5b opgraving. https://doi.org/10.17026/dans-27r-e5f8, DOI 10.17026/dans-27r-e5f8

Schuster M, Paliwal KK (1997) Bidirectional recurrent neural networks. IEEE Transactions on Signal Processing 45(11):2673–2681, DOI 10.1109/78.650093, URL https://www.researchgate.net/profile/Mike_Schuster/publication/3316656_Bidirectional_recurrent_neural_networks/links/56861d4008ae19758395f85c.pdf

Sutskever I, Vinyals O, Le QV (2014) Sequence to sequence learning with neural networks. In: Advances in neural information processing systems, pp 3104–3112, URL http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf

Van der Veken B, Blom E (2012) Veghel Scheiffelaar ii opgraving. https://doi.org/10.17026/dans-z93-7zbe, DOI 10.17026/dans-z93-7zbe

Van der Veken B, Prangsma N (2011) Montferland Didam westelijke randweg Kerkwijk opgraving. https://doi.org/10.17026/dans-zmk-35vy, DOI 10.17026/dans-zmk-35vy

van der Velde H (2011) Katwijk Zanderij Westerbaan opgraving. https://doi.org/10.17026/dans-znz-r2ba, DOI 10.17026/dans-znz-r2ba

van de Velde H, Ostkamp S, Veldman H, Wyns S (2002) Venlo Maasboulevard. https://doi.org/10.17026/dans-x84-msac, DOI 10.17026/dans-x84-msac

Xu Y, Chen Z, Xie Z, Wu L (2017) Quality assessment of building footprint data using a deep autoencoder network. International Journal of Geographical Information Science 31(10):1929–1951, URL http://www.tandfonline.com/doi/abs/10.1080/13658816.2017.1341632

Zahn CT, Roskies RZ (1972) Fourier descriptors for plane closed curves. IEEE Transactions on computers C-21(3):269–281, DOI 10.1109/TC.1972.5008949, URL http://dx.doi.org/10.1109/TC.1972.5008949

Zhang D, Lu G, et al (2002) A comparative study of fourier descriptors for shape representation and retrieval. In: Proc. of 5th Asian Conference on Computer Vision (ACCV), Citeseer, pp 646–651, URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.73.5993