# Deep Contrastive Learning for Feature Representation in Historical Maps

Antonio León Villares



4th Year Project Report Computer Science and Mathematics School of Informatics University of Edinburgh

2023

## Abstract

Physical sources of knowledge, such as historical maps, have been increasingly digitised, making them more available to the wider public. However, investigating certain features within these maps remains challenging, particularly if there are a lot of maps. In this project, I investigate how contrastive learning can be used to not only find common features across maps from different time periods, but to numerically score the degree of similarity between these features. Through contrastive learning, I convert patches from maps into vectorised representations in some latent space. These representations can then be used to compute a similarity score between patches, with which the most similar patches can be found, thus allowing a quick and efficient search of specific features across many maps. I develop a novel way of evaluating the capacity of contrastive models to identify positive pairs correctly. Moreover, I employ proxy tasks to better understand how the representations learnt by 2 different contrastive models encode these features. I found that using SimCLR leads to more discriminative representations, which are better at finding patch pairs containing the same set of features. Meanwhile, BYOL learns representations which are more semantic in nature, and thus can be useful to more broadly explore feature combinations across maps. Finally, I propose a new contrastive framework, GeoSimCLR, aimed at encoding both visual and contextual similarity. I compare this with SimCLR and BYOL by using clustering, which helps reveal how the representations are structured in latent space.

## **Research Ethics Approval**

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

## **Declaration**

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(Antonio León Villares)

## Acknowledgements

Thank you to Amos Storkey, the supervisor of this project, for his invaluable guidance at all stages of this project, from guiding the research question, to helping me navigate the technical intricacies of this project. I am grateful for your dedication, support and patience.

Thank you to Chris Fleet, map curator at the National Library of Scotland, who so kindly provided us with the maps used for this project, and whose passionate explanations have helped me appreciate the subtleties required to understand historical maps.

Thank you to Shuo Chen and Hao Zhou, not only for helping develop the alignment algorithm, but for the many fruitful discussions we were able to have throughout the duration of this project.

Thank you to Eugenio López and Jennifer Turner, for fostering my curiosity, and for pushing me to never stop questioning and seeking answers.

Thank you to my family, who have incessantly supported me throughout my academic life, and in particular, during the completion of this project.

Thank you Anaïs, you are the G to my LG.

# Contents

g and Maps earning earning for ( s  t	Geog		· · · · · · · · · · · ·	· · · · · · al 7 · · · · · · · · ·		    			- · · · · · · · · · · · · · · · · · · ·		· · · · · · · · · · · · · · · · · · ·	· · · · ·	· · · · · · · · · · · · · · · · · · ·	· · · · · · · · · · · · · · · · · · ·	1 2 2 3 4 4 5 5 5 6 7 8 <b>9</b> 9
g and Maps earning earning for ( s  t	Geog		 hic:    	· · · al 7 · · · · · · · · ·	Tas	 ks  		· · · · · · · · ·	· · · · · · · · · · · · · · · · · · ·		· · · · · · · · · · · · · · · · · · ·	· · · · ·	· · · · · · · · · · · · · · · · · · ·	· · · · · · · · · · · · · · · · · · ·	2 2 3 4 4 5 5 5 6 7 8 <b>9</b> 9
g and Maps earning earning for ( as  t t	S		 hica    	· · · al 7 · · · · · · · · ·	Tas	 ks  		· · · · · · · · ·	- · · · · · · · · · · · · · · · · · · ·		· · · · · · · · · · · · · · · · · · ·	· · · ·	· · · · · · · · · · · · · · · · · · ·	· · · · · · · · · · · · · · · · · · ·	2 2 3 4 4 5 5 5 6 7 8 <b>9</b> 9
earning earning for ( as  t	Geog	;rapl	 hica     	 al 7     	Tas	 ks  		· · · · · · · ·	- · · · · · · · · · · · · · · · · · · ·		· · · ·	· · · ·	· · · · · · · · · · · · · · · · · · ·	· · · · · · · · · · · · · · · · · · ·	2 3 4 5 5 5 6 7 8 <b>9</b> 9
earning for ( as	Geog		hica     	al T • • • • • • • • • • •	Tas	ks    		· · · · · · · · · · · · · · · · · · ·	- · · · · · · · · · · · · · · · · · · ·		· · · ·	· · · · · · · · · · · · · · · · · · ·	· · · · · · · · ·	· · · · · · · · · · · · · · · · · · ·	3 4 5 5 5 6 7 8 <b>9</b> 9
IS	· · · · · · · · · · · · · · · · · · ·	· · · ·	· · · · · · · · ·	· · · · · · · · · · · · · · · · · · ·	· · · · · · · · · · · · · · · · · · ·	· · ·		· · · · · · ·	- · ·	• • • • • • • •	· · ·	· · ·	· · · · · · · · · · · ·	· · · · · · · · · · · · · · · · · · ·	4 4 5 5 5 6 7 8 <b>9</b> 9
t	· · · · · · · · · · · · · · · · · · ·		· · ·	· · · · · · · · · · · · · · · · · · ·	· · ·	· · ·	•	· · · · ·	· · ·	· •	· · ·	· · ·	· · ·	•	4 5 5 6 7 8 <b>9</b> 9
t	· · · · · · · · · · · · · · · · · · ·	· · · ·	· · ·	· · · · · · · · ·	· · ·	· · ·	•	· · · · ·	· · ·	· •			· · · · · · ·	•	<b>5</b> 5 6 7 8 <b>9</b> 9
t	· · · · · · · · · · · · · · · · · · ·		· · ·	· · · · · · · · · · · · · · · · · · ·	· · ·	· · ·	•		· · ·	· · ·			· · ·	•	5 5 6 7 8 <b>9</b> 9
t	· · · · · · · · · · · · · · · · · · ·	· · · ·	· · ·	· · · · · · · · · · · · · · · · · · ·		· ·			• •	· ·				•	5 6 7 8 <b>9</b> 9
Aaps	· · · · ·	· · · ·	· · ·	· · ·	• • •	· ·	•	• • •	• •	· ·			• • •	•	6 7 8 <b>9</b> 9
Aaps	· · · · ·	· · · ·	· · ·	· · ·	• • •	· ·	•		• •	•••		•		•	7 8 <b>9</b> 9
s Maps raining the Maps of the Same	· · · · ·	· · ·	· ·	· ·	•		•	•	• •	•••	•	•	•	•	8 9 9
Iaps    raining    the Maps    of the Same	· · · · ·	· · ·		 	•		•	•			•	•	•		<b>9</b> 9
Maps    raining    the Maps    of the Same	· · · · ·	••••	 	 	•	• •	•	•	• •	•	•	•	•	•	9
raining the Maps of the Same	· · · ·	•••													
the Maps of the Same	· · ·					• •	•	•	• •	•	•	•		•	11
of the Same	. D						•		•						11
	ie Keg	gion	ι.				•		•						13
Patches .							•		•						13
tasets from	n Map	os.					•		•						14
Dataset							•		•						16
Patches	••••	•••			•	• •	•		• •	•	•			•	17
															18
							•		•						18
ers							•		•						18
edure							•		•						19
tion Tasks							•		•						19
)							•		•						19
red							•		•						20
															20
		• •	• •	• •	•		•	•	• •						20
of Results	· · · ·	· · ·	· ·	· ·	•	•••	•	•	•	•					20 21
ers edure ution Tasks o red	· · · · ·	· · · ·	· ·	· · · · · ·		· · · ·	•	•		· · · ·	· · · · ·	· · · · · ·	· · · · · · · ·	· · · · · · · · · · ·	

	4.3	Exploring Encoded Features	25 25
		4.3.2 Invariance Under Transformations	28
	4.4	Geographical Awareness	29
		4.4.1 Geographically-Aware Representations	29
		4.4.2 Clustering	33
5	Disc	ussion and Conclusions	38
	5.1	Main Conclusions	38
	5.2	Future Work	39
A	Add	itional Work	45
	A.1	Aligning Maps of the Same Region from Georeference Information .	45
	A.2	Aligning Maps at the Patch Level	46
	A.3	Aligning OS Maps with Bartholomew Maps	48
	A.4	Modifying Representations Through Learning Task	49
B	Map	os and Metadata	51
	B.1	GeoJSONs	51
		B.1.1 GeoJSON Containing Full Map Region	51
		B.1.2 GeoJSON of Overlapping Bounding Boxes	52
	B.2	Code for Rotating and Cropping the Badly Scanned Maps	54
	B.3	Metadata Table Columns	56
	B.4	Code for Removing Uninformative Patches	56
С	Exp	erimental Results	58
-	C.1	Estimate for Runtime	58
	C.2	All Experiments Run	58
	C.3	Losses for Experiments	59
	0.0	C 3 1 BYOL	59
		C 3 2 SimCLR	60
	C 4	Results for Experiments Run Longer	60
	0.1	C 4 1 Training Results	60
		C.4.2 PPIT Results	60
	C 5	Deriving Automatic Thresholds for Information Awareness	60
	C.5	Transformations Applied	61
	C.0	$C 6 1  \text{Potetion} (30^{\circ})$	61
		C.0.1 Rotation $(50)$	62
		C.0.2 Holizonial Fup $\ldots$	62
		C.0.5 Random Rotation (90, 180, 270) $\ldots \ldots \ldots \ldots$	02
		C.0.4 Random Crop $(64 \times 64)$	02 (2
		C.6.5 Centre Crop $(64 \times 64)$	62
		C.0.0 Gaussian Blur	63
		C.6./ Colour Jittering	63
	a -	C.6.8 Grayscaling	63
	<b>C</b> .7	Clustering Results	63
		C.7.1 Settings for UMAP	63
		C.7.2 Settings for HDBSCAN	64

		C.7.3	Clustering on Validation Data	65
		C.7.4	Cluster Similarities Using 100 Features	65
		C.7.5	Cluster Examples	66
		C.7.6	Score Distributions for All Clusters	71
		C.7.7	Clusters for Regions from GeoSimCLR Representations	73
		C.7.8	Similarity Score Matrices for Region Patches from GeoSimCLR	
			Representations	76
		C.7.9	Map Segmentation Through Clustering	77
D	Unde	erstand	ing Similarity Distributions	80
D	<b>Unde</b> D.1	e <b>rstand</b> Simila	ing Similarity Distributions	<b>80</b> 80
D	<b>Unde</b> D.1	e <b>rstand</b> Simila D.1.1	ing Similarity Distributions rity Score Distributions	<b>80</b> 80 80
D	Unde D.1	e <b>rstand</b> Similar D.1.1 D.1.2	Similarity Distributions   rity Score Distributions   BYOL   SimCLR	<b>80</b> 80 80 83
D	Unde D.1	erstand Similar D.1.1 D.1.2 D.1.3	Iing Similarity Distributions   rity Score Distributions   BYOL   SimCLR   Weighted GeoSimCLR	<b>80</b> 80 80 83 86
D	Unde D.1 D.2	erstand Similar D.1.1 D.1.2 D.1.3 Most S	Img Similarity Distributions   rity Score Distributions   BYOL   SimCLR   Weighted GeoSimCLR   Similar Patches	<b>80</b> 80 83 86 87
D	Unde D.1 D.2	erstand Similar D.1.1 D.1.2 D.1.3 Most S D.2.1	Iing Similarity Distributions   rity Score Distributions   BYOL   SimCLR   Weighted GeoSimCLR   SimCLR and BYOL	<b>80</b> 80 83 86 87 87

# **Chapter 1**

# Background

## 1.1 Motivation

With the rise of technology, we have been able to digitise physical sources of knowledge, both textually (books, manuscripts, plays) and visually (images, maps, sketches). Being able to efficiently understand, explore and analyse these digitised sources is particularly critical, should we want to gain important historical knowledge. This is especially true for historical maps, as they contain a plethora of information showcasing how human settlements have varied over time. However, for highly detailed maps, manually searching for changing features is arduous, especially if we want to comprehensively study these changes not only over different time periods, but also using information present from different map styles. Nonetheless, doing so automatically is also highly non-trivial: a system would need to be capable of understanding the difference between permanent features (such as buildings, roads and rivers) and dynamic features (such as text labels, shadings and legend style).

To this end, I believe that deep learning is a sensible way of tackling such a problem, given its demonstrated capacity over the last few years to automatically learn key attributes in images. In particular, in this project I seek to learn common features across temporally-spaced maps from the NLS (National Library of Scotland) through the use of contrastive learning (CL). This technique allows us to translate the notion of image similarity to similarity in some latent representation space, which can be leveraged to explore shared features across different maps. Whilst simpler methods could be employed to extract and compare these features, CL architectures are trained with the objective of assessing the degree of similarity between different images. Lastly, I have taken this as an opportunity to compare the types of attributes which are learnt by two different contrastive methods, not only by seeing how these methods score similarity between regions, but also by employing the latent representations in downstream tasks.

### 1.2 Related Work

#### 1.2.1 Deep Learning and Maps

Different deep learning techniques have been employed on maps, particularly for text/object detection. The work by Hosseini et al. [1] is of particular relevance, since they used historical maps from the NLS to develop a deep learning, computer vision library. One of their aims was to, for example, identify railways in these maps, for which they used a ResNet [2] (Residual Neural Network), trained on augmented versions of patches taken from the maps, for classification (admittedly with around 30 million, very high resolution patches - this might go beyond the scope and computational capacity of this project). They also deal with false positives by gauging contextual information (if a railway was found in an "isolated" patch, this is probably a misclassification). Post-processing steps such as these, which exploit the rich contextual information of maps, could be a useful technique to improve the learnt representations. Another very interesting technique which they apply is that, alongside the patch, they provide a larger "contextual" patch (containing the patch used for classification). Information from the patch and its context can then be used to train a more powerful model. Maps from the NLS have been used in a variety of other publications and projects. For instance, Li [3] applied a CycleGAN [4] to convert OpenStreetMap [5] images into the style of historical maps. Li et al. [6] worked on using OCR, alongside deep neural networks, to extract text from historical maps (amongst which were some from the NLS), and use this to generate meta-labels for the maps, by using geolocation based on the extracted text. In fact, extracting text has been the focus of many map-related projects. In this dissertation by Kai Williams, they developed deep learning models for extracting symbols (i.e trees) and text from the historical maps provided by the NLS.

#### 1.2.2 Contrastive Learning

CL is a form of self-supervised learning (SSL), whereby a model only has access to unlabelled, training data, which it uses to derive labels/representations that can then be used to tackle a downstream supervised or unsupervised task. In this regard, SSL is a crucial technique, since data labelling is a time-intensive and expensive process [7].

Whilst CL has applications in other fields, it has been employed extensively in computer vision tasks, where learning and representing visual similarity might be critical. CL algorithms generally follow a similar workflow. We start with raw, unlabelled images, which are augmented (using a range of transformations, such as cropping, resizing, noise addition, blurring, colour jittering) to generate what are known as positive and negative training pairs. Positive pairs correspond to two different augmentations of the same image, whilst negative pairs are augmentations of different images. The contrastive algorithm learns to think of positive pairs as similar, whilst also learning to think of negative pairs as dissimilar. Generating a good set of positive-negative pairs is crucial for learning useful, contrastive representations, so special care needs to be taken when generating them. For instance, if the positive and negative pairs are easily identifiable as dissimilar, the algorithm might default to learning extremely simple representations, which don't capture the nuanced features that characterise a particular

input image [8]. After the positive and negative pairs have been generated, they get passed through the contrastive model, and converted into a vector representation in latent space. The loss for contrastive models encompasses both the level of similarity between negative pairs, alongside the level of dissimilarity between negative pairs. This similarity/dissimilarity is typically defined by the cosine distance of the latent representations, and can be incorporated into the loss in a variety of ways, such as max margin contrastive loss [9], triplet loss [10], n-pair loss (which generalises triplet loss) [11], InfoNCE [12] and NT-Xent Loss (normalised temperature-scaled cross-entropy) [13]. The most well-known CL architectures by far are SimCLR [13], MoCo [14] and CPC [12], which were able to attain nearly supervised-level Top-1 accuracy on ImageNet [13]. More recently, BYOL [15] was released, which revolutionised the CL paradigm by removing the need for negative samples, whilst obtaining better results than all of the aforementioned methods.

Due to the versatility of CL, it has been employed in a plethora of practical applications beyond benchmarking on ImageNet [16]. Medicine has greatly benefited from CL, since human annotated medical images (X-rays, MRIs, CT-Scans, etc...) are scarce [17] [18] [19] [20]. For instance, CL can be applied to obtain better data efficiency in downstream classification tasks, which is particularly important when data is scarce [17]. Furthermore, Chaitanya et al. [20] describes a novel contrastive approach, whereby the contrastive loss is derived by using both global and local features of an image. CL can also be applied beyond the world of images, since it provides a general framework to capture underlying features, which can act as signals. As such, contrastive approaches have been used to develop recommender systems [21], or to represent videos [22], by encoding temporal information between frames.

#### 1.2.3 Contrastive Learning for Geographical Tasks

It is first important to address the fact that many of the CL frameworks described above employ a ResNet as an encoder to convert the unlabelled images to latent representations. ResNets have also been used for identifying land usage (e.g. is it residential, industrial, agricultural, are there lakes, etc...) from satellite images [23], which indicates its power as an encoder for geographical tasks. However, as demonstrated in Chiang et al. [24], there are many challenges to address when applying a deep Convolutional Neural Network (CNN) [25] (such as a ResNet) to encode images of historical maps. This is due to the unique nature of maps, which provide a simplified representation of reality. As such, features such as roads, mountains and buildings are represented in a very simplistic manner, as groups of lines. Extracting features from these representations can thus be harder than when faced with the typical images of ImageNet, where for each label there are a variety of well-defined, easily-identifiable features. Moreover, this simplicity means that a model can more easily pick up on incorrect features (for example, if a patch of a map cuts off a section of a river, for a CNN to identify this correctly will be particularly hard, since it could also be a section of a house). This is particularly the case when small, low-resolution patches are used; for larger, highresolution patches, ResNets should be powerful enough to easily identify the presence of certain features, since such patches inherently contain a lot more information which is harder to misinterpret.

4

To the best of our knowledge, CL has never been used to learn map representations. However, there have been some uses of CL in geographical applications, particularly from satellite data. Agastya et al. [26] adapted the SimCLR architecture to learn representations of images in BigEarthNet-S2 [27], to then be used for determining the presence of irrigation in images. Perhaps most relevant to the project at hand is the recent work by Ayush et al. [28], who modified the MoCo architecture to generate what they called a geography-aware model. They trained 2 models on 2 datasets (a subset of ImageNet with geo-tagged metadata, and the remote sensing dataset fMoW (Functional Map of the World) [29])). The fMoW dataset is of interest because it contains temporally spaced images; this is analogous to the case described here, where we have temporally-spaced and stylistically spaced map images. The authors incorporate these temporal differences to generate positive training pairs within their loss. In addition to generating representations, they also train the model to predict the geo-location of the images. In doing this, they are able to significantly reduce the gap between supervised and SSL performance in a downstream classification task, when compared with the standard MoCo architecture. This indicates that using temporal data as "augmentations", alongside training the model to output metadata on the image can be used to bridge the unavoidable gap between supervised and self-supervised performance.

## 1.3 Goal and Contributions

The goal of this project is to learn representations for historical maps, which can be used to numerically score the degree of similarity between regions, and thus provides an effective way to query different maps in the search for regions with a certain combination of features. To this end, I seek to train 2 different contrastive models, namely SimCLR and BYOL. To evaluate these models, I propose a set of tasks, known as Positive Pair Identification Tasks. To the best of our knowledge, CL has never been applied for this purpose. Furthermore, I use this as an opportunity to systematically compare the latent representations learnt by the two models. To do this, I explore the degree to which temporal and content information are encoded within these representations. Furthermore, I analyse the effect that different augmentations have on the contrastive similarity scores. Lastly, I develop a new contrastive framework, aimed at encoding both visual and contextual similarity, whose representations I explore by applying clustering.

## 1.4 Report Structure

I begin this report by explaining the objective of CL, followed by a short explanation on the SimCLR and BYOL architectures, and how I have adapted them for this project. Afterwards, I explain the data processing pipeline: how I obtained the maps, how they were pre-processed, and how the final dataset was generated for training the contrastive models. Finally, I showcase how the models have been employed, by explaining the experiments performed, the results obtained, and the subsequent conclusions that can be drawn.

# **Chapter 2**

## Model Architectures

### 2.1 Contrastive Learning

Contrastive learning is a learning paradigm by which a model learns to gauge the similarity or dissimilarity between pairs of data points, in a completely self-supervised manner. More explicitly, say we have a set X of objects (typically images). In contrastive learning, we learn a function f which maps elements in X to some D-dimensional latent space:

$$f: \mathcal{X} \to \mathbb{R}^D, \quad D \in \mathbb{N}$$

This is done in such a way, so that for some similarity function:

$$\operatorname{sim}: \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$$

if  $x_1, x_2$  are similar in  $\mathcal{X}$  (that is, they form a positive pair), their embeddings  $f(x_1), f(x_2)$  will be similar in latent space (according to sim); if on the other hand  $x_1, x_2$  form a negative pair in  $\mathcal{X}$ , this dissimilarity should also be reflected in latent space.

The most common similarity measure employed in the literature is cosine similarity. Given 2 embeddings in latent space  $\underline{z}_1, \underline{z}_2 \in \mathbb{R}^D$ , their cosine similarity is given by:

$$\sin(\underline{z}_1, \underline{z}_2) = \frac{\underline{z}_1^T \underline{z}_2}{\|\underline{z}_1\|_2 \|\underline{z}_2\|_2} \in [-1, 1]$$

where  $\underline{z}_1^T \underline{z}_2$  is the standard dot product, and  $\|\cdot\|_2$  is the (Euclidean) L2 norm. This similarity is nothing but the cosine of the angle between  $\underline{z}_1$  and  $\underline{z}_2$ , whereby perfectly similar vectors are such that  $\operatorname{sim}(\underline{z}_1, \underline{z}_2) = 1$ , whilst perfectly dissimilar vectors are such that  $\operatorname{sim}(\underline{z}_1, \underline{z}_2) = -1$ .

## 2.2 Models for this Project

For this project I compared the performance of 2 common contrastive architectures, SimCLR (Simple Contrastive Learning of Representations) and BYOL (Bootstrap Your Own Latent). I first provide a general overview, and then explain how I have adapted them to fit the objectives of this project. Lastly, I have deviated slightly from the notation employed by the original SimCLR and BYOL papers, in order to make the model descriptions more homogeneous and comparable.

#### 2.2.1 SimCLR

#### 2.2.1.1 Model Architecture

SimCLR learns a representation for an input x, by minimising the contrastive loss between 2 augmentations of x. More specifically, given a set of transformations  $\mathcal{T}$ , one samples 2 transformations  $t_1 \sim T$  and  $t_2 \sim T$ . These include colour jittering, random cropping with flipping (followed by resizing), and random cropping (however, in the original paper they explored the effect of many other transformations, such as random rotations or sobel filtering).  $t_1$  and  $t_2$  are applied to the input to generate augmented views  $x_1, x_2$ , which are treated as similar. The augmented views are passed through an encoder f, which generates representation vectors in latent space,  $\underline{z}_1 = f(x_1), \underline{z}_2 = f(x_2)$ . These representations are projected by a projection head g into some other space, where the contrastive loss is actually computed. Nonetheless, the output of the model (the embedding) will be the representation in latent space *before* passing it through g, since they found that this provided better representations. In the original SimCLR (and in most papers utilising this model), f is a pretrained ResNet 50 model. However, instead of using the full network, they take the output of the network at its penultimate layer, which is an average pooling layer. Moreover, the projection head g is a simple multilayer perceptron (MLP), with a single hidden layer and ReLU activation.



Figure 2.1: Sketch of the SimCLR architecture, adapted from Figure 2 in the original SimCLR paper.

#### 2.2.1.2 Loss

SimCLR uses NT-XENT as its contrastive loss, as introduced by Sohn [30]. This loss adapts categorical cross-entropy loss, where the contrastive objective is framed as a multi-class classification problem. In particular, this loss seeks representations, such that positive samples have maximised similarity (according to cosine similarity), whilst negative samples have maximised dissimilarity. Given a minibatch of N positive pairs, each positive pair will use the remaining N - 1 pairs as negative pairs. In particular, if I define:

$$sim_{loss}(\underline{a}, \underline{b}) = \exp\left(\frac{sim(\underline{a}, \underline{b})}{\tau}\right)$$

the loss for the *i*th image  $z^i$  in the minibatch will be:

$$\begin{split} \ell_i = &-\log\frac{\operatorname{sim}_{-} \operatorname{loss}(\underline{z}_1^i, \underline{z}_2^i)}{\sum_{k=1}^N \mathbbm{1}_{k \neq i} \left(\operatorname{sim}_{-} \operatorname{loss}(\underline{z}_1^i, \underline{z}_1^k) + \operatorname{sim}_{-} \operatorname{loss}(\underline{z}_1^i, \underline{z}_2^k)\right)} \\ &- \log\frac{\operatorname{sim}_{-} \operatorname{loss}(\underline{z}_1^i, \underline{z}_2^i)}{\sum_{k=1}^N \mathbbm{1}_{k \neq i} \left(\operatorname{sim}_{-} \operatorname{loss}(\underline{z}_2^i, \underline{z}_1^k) + \operatorname{sim}_{-} \operatorname{loss}(\underline{z}_2^i, \underline{z}_2^k)\right)} \end{split}$$

where sim is cosine similarity,  $\mathbb{1}_{[k\neq i]}$  is an indicator function (evaluates to 1 if and only if  $k \neq i$ ) and  $\tau$  is a temperature parameter, which modulates the degree of similarity which I want to assign to positive and negative pairs. A higher  $\tau$  will decrease the similarity terms in the NT-XENT loss, which pushes the model to increase its discriminative power in order to minimise the loss. This means that negative samples will be pushed further apart in representation space, which makes distinguishing between positive and negative pairs easier.

#### 2.2.2 BYOL

#### 2.2.2.1 Model Architecture

Unlike with SimCLR, BYOL doesn't require negative pairs to learn a representation. To do this, it defines 2 networks: an online network with parameters  $\theta$ , and a target network with parameters  $\xi$ . Critically,  $\xi$  is computed as an exponential moving average of  $\theta$ , and is thus not trained. In particular, for some target decay rate  $\tau \in [0, 1]$ 

$$\xi \leftarrow \tau \xi + (1 - \tau) \theta$$

BYOL solely trains  $\theta$ , with the purpose of using the online network to predict the output of the target network. To this end, the online network is defined by an encoder  $f_{\theta}$  (which converts the input image into a vector), a projector  $g_{\theta}$  (which projects the encoded vector into some other space) and a predictor  $h_{\theta}$  (which predicts the output vector of the target network from the projected vector). The target network only has an encoder  $f_{\xi}$  and a projector  $g_{\xi}$ . As above, an input image x is augmented to produce a positive pair; the first augmentation  $x_1$  is passed through the online network, whilst the second augmentation  $x_2$  is passed through the target network. As with SimCLR, once  $\theta$  has been trained, we only keep  $f_{\theta}$  to convert an image into its contrastive embedding. Moreover, the encoders  $f_{\theta}, f_{\xi}$  are also residual networks, whose output will be the final average pooling layer. The projectors  $g_{\theta}, g_{\xi}$  are also MLPs, with a single hidden layer and RELU activation, but unlike with SimCLR, BYOL employs batch normalisation. The predictor  $h_{\theta}$  uses the same architecture as  $g_{\theta}$ .



Figure 2.2: Sketch of the BYOL architecture, adapted from Figure 2 in the original BYOL paper.

#### 2.2.2.2 Loss

Since BYOL requires no negative pairs, and it frames its contrastive problem as a prediction problem, the loss will be the MSE (mean square error) between  $\hat{q}_2$  and  $\underline{q}_2$  (after they have been normalised to unit vectors). Moreover, since the architecture is asymmetric, we do a second pass through the network, but this time with  $x_2$  going through the online network and  $x_1$  going through the target network. Given the *i*th image  $z^i$  in a minibatch of N elements, this is nothing but:

$$\ell_i = 4 - 2\left(\operatorname{sim}(\underline{q}_2^i, \underline{\hat{q}}_2^i) + \operatorname{sim}(\underline{q}_1^i, \underline{\hat{q}}_1^i)\right)$$

where sim is once again cosine similarity.

#### 2.2.3 Project Models

The main difference between my approach and the original SimCLR/BYOL implementations, is that I don't apply any direct data augmentation to generate positive pairs. Instead, much like in the work by Ayush et al. [28], I use temporally-spaced images of historical maps. Due to the time differences, general structure in the images is preserved, but there are certain stylistic changes, such as differences in colour, positioning of legends, legend style, building shading, and so on. I believe that these differences are substantial enough to be considered augmentations, which is why no further processing was performed on these images.

I implemented all the models used in this project (except for the ResNets) from scratch using PyTorch [31]. Whilst there are many implementations of BYOL and SimCLR available online, I wanted to construct a consistent API for the models to make model training and comparison simpler. Beyond this, due to memory constraints I decided to not use ResNet50, and instead opted to consider ResNet18 and ResNet34 (I used the architectures and pretrained weights available from PyTorch: https://pytorch.org/hub/pytorch\_vision\_resnet/). Note that when using ResNets, images have to be resized to have dimension 224 × 224, and then normalised in a per-channel basis using:

$$\mu = [0.485, 0.456, 0.406]$$
  $\sigma = [0.229, 0.224, 0.225]$ 

Furthermore, I wanted to see the effect on learning of using a simple CNN encoder, so somehow following the style of the ResNet encoders, I defined a CNN model which used 5 convolutional layers, with ReLU activation, batch normalisation and max pooling. After the final convolutional layer, average global pooling was applied to generate the encoder output. When using ResNets, I followed the implementation details for SimCLR and BYOL, and used the output of the final global average pooling layer as my latent representation. For both CNN and ResNet models, this results in 512-dimensional embeddings. For the sake of consistency, the MLPs for the SimCLR and BYOL models followed the BYOL structure: a single hidden dimensions, with ReLU activation, and batch normalisation. All the MLPs used 2048 hidden dimensions, and outputted a 256 dimensional vector. Lastly, we used a random seed of 23 for any random process required for this project. In particular, random shuffling was used throughout this whole project when training models, to improve generalisability and convergence.

# **Chapter 3**

## Data

In this section, I explain the origin of the historical maps, and how these were processed to generate the data for the project. This data consists of positive pairs, where each pair is composed of patches (square regions from a map) corresponding to the same region but from different maps. I would like to thank Chris Fleet at the NLS for giving me access to all the maps used for this project.

## 3.1 Obtaining Historical Maps

All the maps used for this project were sourced thanks to the NLS, which contains over 1.5 million sheet maps [32]. From these, approximately 200,000 are available in high resolution at https://maps.nls.uk. This project focused on maps of Edinburgh (both the city and surroundings), since there were a lot of these freely available. The NLS contains a plethora of different historical map styles, ranging from the 16th century up until the 20th century. Out of all of these, I wanted to focus on maps which showcased a more modern Edinburgh, as this would provide more applicable results. I also sought georeferenced maps, which have been aligned to a specific geographic coordinate system, as this allowed me to easily visualise temporal differences.

In particular, this project focuses on 25 Inch Ordnance Survey Maps (which I'll call "OS maps" from here onwards). The Ordnance Survey is the national mapping agency of Great Britain [33]. These maps are known as "25 Inch", since every inch of the map corresponds to 25 inches on the ground. These maps are suitable exemplars, since not only are they georeferenced, but I have access to a large number of them: 212 distinct maps, split into 57 distinct regions which together encompass a total area of 278.39  $km^2$ . For each region, either 3 or 4 full maps were made available to me, each of which corresponds to a different time period of production. In particular, I was given 57 maps for the periods 1894-1896, 1906-1908 and 1913-1914; and 41 maps sheets for the period 1933-1947. All these maps are available as GeoTIFF files, which are standard TIFF (Tag Image File Format) files containing additional metadata, such as bounding boxes and the coordinate system used to georeference the map.



Figure 3.1: An example OS map obtained from the NLS (https://maps.nls.uk/ view/82877409).



Figure 3.2: The rectangular region encompassed by all the maps. The bottom left corner has longitude-latitude (-3.38374788940603, 55.8715983250667), whilst the top right corner has longitude-latitude (-3.07422210721001, 56.0020051346329). This is an area of 278.39  $km^2$ . Image generated using geojson.io (see subsection B.1.1).

The NLS also contains georeferenced maps in different styles, such as "Bartholomew" style maps. However, I chose not to use these, since they had a lower resolution (many OS maps could fit into a single Bartholomew map); they were more blemished than the OS maps (for instance, I found red lines coloured in some Bartholomew maps); and preprocessing them is a lot more challenging, particularly due to their border style (more on this in the next section). Hence, and given that OS maps are more widely used and referenced, I found it unnecessary to produce a dataset containing both OS and Bartholomew maps.



Figure 3.3: An example Bartholomew map. The colourful regions could harm the model's performance. The borders make these maps harder to pre-process. Moreover, this map is oriented vertically, whilst all the OS maps are horizontal.

## 3.2 Generating Data for Training

### 3.2.1 Preprocessing the Maps

Before generating a dataset, I pre-processed the maps in 2 steps. Firstly, I rotated and cropped the maps, to rectify scanning issues which would have made map alignment and positive pair generation impossible. Secondly, I downsampled the maps, as their large memory footprint and high resolution would have made them harder to use for training.

### 3.2.1.1 Rotating and Cropping

The TIFF files contained maps which had been rotated during scanning, and then to create a rectangle, padded with white pixels. Since each TIFF file had a different rotation and dimensions, aligning 2 maps correctly was impossible.



Figure 3.4: The map showcased in Figure 3.1 before any pre-processing. I have added the black frame to show the padding present in the maps before any pre-processing.

However, using OpenCV [34] I could rotate the map to make it horizontal, and then easily crop out the padding. Code for this is available at section B.2. Whilst the resulting maps were still of slightly different widths and heights (since the original maps had a dimension mismatch), this difference was negligible, given how large the maps are, and posed no problem for training down the line. Nonetheless, sometimes the boundary between map and padding couldn't be properly located, so not all padding was removed. Such maps were removed from further pre-processing, since this affected less than 10 of the 212 maps, most of which contained very few informative features anyways.



Figure 3.5: A map whose right padding wasn't removed during preprocessing.

#### 3.2.1.2 Downsampling

The metadata and high resolution of the GeoTIFFs means that each map occupies 400-600MB of memory, which makes them challenging to work with. Moreover, the high resolution would require me to work with relatively large patches to distinguish features like roads or full buildings, which would lengthen the learning process unnecessarily. Therefore, I decided to downsample the maps by considering kernels of width 2, 3 and 4, where each kernel converts square regions of the kernel's width into a single pixel. I considered **standard downsampling** (pick the minimum, average or median pixel value over the region) and **resizing** (use bilinear, bicubic or nearest-neighbour interpolation).



(a) Standard downsampling applied on each map. For the  $2 \times 2$  kernel, all methods seem to behave similarly. For wider kernels, using the minimum pixels results in a loss of pattern detail, alongside a darkening of the image. Similarly, mean downsampling results in a blurring effect. The median, to a lesser degree, also sees loss of detail, particularly with lightening of pixels.



(b) Downsampling via resizing applied on each map. Both bilinear and bicubic interpolation look very similar for all kernel sizes, with good detail preservation. Nearest interpolation also works quite well, but loses some details in the writing for the  $4 \times 4$  kernel.

Figure 3.6: Downsampling strategies applied to  $64 \times 64$  patches.

Given the results from Figure 3.6, I chose to apply bilinear interpolation as my down-

sampling method. I felt that it preserved structure better (letters, shapes, shadings) than the standard downsampling methods; it also looked extremely similar to the bicubic method, and it seems to work better than nearest-neighbour interpolation for bigger kernel sizes. I also felt that using a kernel size of 4 allowed the most flexibility, since then even small patches of size  $32 \times 32$  contained identifiable features. After all these pre-processing steps, each map occupied around 28MB (this suggest that I successfully isolated the maps from the original TIFF files, as before the rotation and padding meant that the TIFF files had variable file sizes of 400-600MB); by converting them to PNG, we further reduced the file sizes to around 14MB.

### 3.2.2 Finding Maps of the Same Region

Once the maps had been pre-processed, I had to determine when 2 maps corresponded to the same region, as these would be used to construct the positive pairs. Doing this manually would have been tedious, inefficient and non-scalable, particularly since the file names I was given were rather cryptic (for example, 82878027.tiff). Whilst this can be done by exploiting the georeference information present in the GeoTIFFs (such as the bounding boxes for the maps), I found that the rotations present within the scanned maps had affected the georeferenced data aswell. This could be dealt with fairly easily (as I detail in section A.1), but the method we developed assumed that the georeferenced bounding boxes for maps of the same region were similar, and we couldn't guarantee this. Because of this, I asked the NLS whether they had more reliable metadata about the maps, as this would allow me both identify maps of the same region, but also access other useful pieces of information, such as year of production. Thankfully, this was made available to us as a table (see Table B.1 for further details), and we were able to group maps by region, using the bounding boxes present in the table.

### 3.2.3 From Maps to Patches

To generate the patches, I used the package patchify [35] to split up each map. I created a custom class to represent these patches, in order to store additional data, such as the map from which the patch was extracted and the coordinate of the patch's left pixel in the original map. This helped organise the patches, particularly when working with them in downstream tasks.

It is important to note that patchify will discard parts of the map, so as to ensure that every patch which is created has the same shape. For example, if the maps were of size  $640 \times 660$ , and I wanted to create  $64 \times 64$  patches, patchify would "ignore" the column which is 20 pixels wide at the end of the map, and effectively just create 100 patches, by considering a region of size  $640 \times 640$ . patchify does include functionality to create overlapping patches, which would mean that we'd be able to potentially use the whole map for patches. This could provide some benefits, in terms of learning more geographically aware embeddings (since neighbouring patches would share features). However, for this project I decided to focus only on non-overlapping patches, as I felt including the overlaps could complicate the analysis and applicability of my results. Once I had the patches, I developed an alignment algorithm, which would shift patches to maximise their degree of visual alignment, and thus generate positive patch pairs. However, after applying it, I noticed that barely no alignment was required in most cases (this is discussed in detail, alongside the algorithm, in section A.2), so I chose to not use the alignment algorithm. Moreover, misalignments of a few pixels, could constitute some degree of data augmentation, which is an integral part of CL.



Figure 3.7: In general, upon converting maps into patches, patches tended to align well with one another visually, so we decided to not apply the alignment algorithm.

### 3.2.4 Generating Datasets from Maps

For training the models, each training sample corresponds to a positive pair of patches (i.e 2 patches representing the same area, but taken from different maps). Since I had 3 or 4 maps for each region, I had 3 or 4 different patches representing the same area. I defined positive pairs by considering all unique combinations of aligned patches. For instance, if I have 4 patches for the same area, this results in 6 positive pairs:

- patch 1 gets associated with patches 2,3 and 4
- patch 2 gets associated with patches 3 and 4 (since a pair already includes patches 1 and 2)
- patch 3 gets associated with patch 4 (it has already been associated with patches 1 and 2)

Similarly, if I had 3 patches, then these would generate 3 positive pairs.

However, I decided to remove some positive pairs from the final datasets, since a lot of patches were blank or contained few discernible features (such as residual segments cut off from buildings/text during the patchifying process). Any patch pair containing at least one of these uninformative patches was removed from the data, as I hoped that a more feature dense dataset would improve the contrastive representations. To achieve this, I once again used OpenCV: I converted patches to grayscale, applied Gaussian blur with a  $3 \times 3$  kernel (and automatic standard deviation), applied the Canny [36] edge detection algorithm, and counted the number of black pixels found in the edges. If these constituted less than 1% of all pixels in the patch, any positive pair containing the patch was removed.



Figure 3.8: Edge detection applied to a patch. This provides a crude measure of information content, and provides a simple heuristic for informative features.

I chose a 1% threshold after visually exploring the patches which were removed, and observing that it generally correlated well with my judgement of whether a patch contains informative features or not:



Figure 3.9: Random selection of 10 patches, 5 of which are uninformative according to the algorithm defined above. Notice how patches which are deemed as uninformative can still contain features (such as pieces of text, legends or parts of a road), but these are minimal, and I don't believe they'll contribute to learning more useful representations. On the other hand, non-empty/informative patches are feature rich. Notice how a patch in which full-text appears will generally be recognised as informative.

Whilst exploring the generated dataset, I observed that certain patches contained black lines at the edges, which I believe are lines corresponding to the borders of the maps. I found these to be problematic, since potentially empty patches which have this border would still remain in the dataset, even though the border doesn't constitute an interesting feature to learn. To filter out patches which contain these border pixels, I looked at whether the patches contained continuous vertical/horizontal bands of pixels across the left/right/top/bottom 10% of the image (with edge detection). If any of these bands was completely composed of black pixels *and* these black pixels constituted at least 50% of all edge pixels found in the patch, I removed any patch pair containing said patch. This ensured that if images do have informative content, they remain in the dataset, even if they do have artifacts derived from the border pixels.



Figure 3.10: Examples of patches which have border pixels. Our filtering algorithm will remove the first 4 of these patches, since most of the information that they contain is in the border pixels. However, the last patch will be preserved in the dataset, since it does hold informative value.

I also considered much harsher thresholds (such as 25% instead of 50%), but observed that this only contributed to removing an additional 1,000 positive pairs. Whilst it is true

that borders themselves don't contribute an informative features, patches containing borders can still hold useful information (such as the types of features which are typically at the borders of maps). Thus, by choosing a higher threshold, I could be more certain that I was removing patches containing mainly border pixels. See the full implementation in section B.4.

Based on training speed and model performance on preliminary runs, I decided to utilise  $128 \times 128$  patches. This resulted in a final dataset containing 104,239 positive patch pairs, after removing 44,531 positive patch pairs via the methods outlined above.



(a) Examples of positive pairs preserved for the final dataset (each column is a positive pair). As can be seen, they contain plenty of information, and the alignment is fairly good.



(b) Examples of positive pairs removed from the final dataset (each column is a positive pair). As can be seen, empty, uninformative patches are removed. However, due to differences in styling, and the unperfect alignment, certain positive pairs are removed, despite one of the patches containing useful features.



### 3.2.5 Splitting the Dataset

In general, self-supervised learning is used to learn pretrained representations, which can be employed in unrelated downstream tasks, where there is little data for training. As such, large datasets (such as ImageNet) are fully used to train the contrastive models. In my case, since I wasn't using any external dataset, and all the downstream tasks which I'd designed employed these historical map patches, I decided to split the 104,239 positive patch pairs into training, validation and testing pairs, using an 80-10-10 split respectively. I used this split both for training the contrastive models, and evaluating them in downstream tasks.

Initially, I thought of applying this split in a straightforward manner, by selecting a corresponding proportion of positive pairs for each of the datasets. However, this has a clear flaw: whilst this prevents the same positive pair from being used in training and evaluation, the nature of the data makes it so that the same patch can appear in training, validation and testing, since every patch appears between 1 and 4 times within different positive pairs in the dataset. This would be problematic, as at evaluation time, our model will have likely already seen some of the patches, so even if it has never directly observed a specific positive pair, this is likely to inflate generalisation performance.

Because of this, I opted for applying the split by using patch indices. Say we have a region *r*, and that there are 4 OS maps representing said region:  $\mathcal{M}^{r,1}, \mathcal{M}^{r,2}, \mathcal{M}^{r,3}, \mathcal{M}^{r,4}$ , and let  $\mathcal{M}_i^{r,j}$  denote the *i*th patch of the *j*th such map, where  $j \in \{1,2,3,4\}$ . Then, let  $P^r$  denote a set of indices, such that if  $i \notin P^r$ , then  $\mathcal{M}_i^{r,j}$  is a patch which was removed from the final dataset in the process outlined in subsection 3.2.4, and otherwise  $\mathcal{M}_i^{r,j}$  is a patch which has remained in the dataset. For each region *r*, I sample 80% of the indices in  $P^r$  for training, 10% for validation, and 10% for testing, to generate  $P_{\text{train}}^r, P_{\text{validation}}^r, P_{\text{test}}^r$  respectively. Our final training, validation and testing sets are then constructed by, for each region, using the patches corresponding to the indices in  $\mathcal{P}_{\text{train}}^r, \mathcal{P}_{\text{validation}}^r, \mathcal{P}_{\text{test}}^r$ . This ensures that if there is a positive pair of the form  $(\mathcal{M}_i^{r,j_1}, \mathcal{M}_i^{r,j_2})$  in one of the datasets, then the patches  $\mathcal{M}_i^{r,j_1}$  and  $\mathcal{M}_i^{r,j_2}$  will **only** ever appear within the same dataset. Due to different regions having different degrees of informativeness, after splitting the dataset, the result wasn't a perfect 80-10-10 split:

- 83,648 positive patch pairs for training (80.20% of the full dataset)
- 10,285 positive patch pairs for validation (9.91% of the full dataset)
- 10,306 positive patch pairs for testing (9.90% of the full dataset)

### 3.2.6 Using Larger Patches

As part of my experiments, I also wanted to see whether increased patch sizes of  $224 \times 224$  would influence the training results (these patches wouldn't have to be reshaped to pass them through the ResNet models, and they should contain more information than the  $128 \times 128$  patches). Since these patches were larger, since I didn't use overlapping patches, and since the maps have different sizes, the different maps for the same region could get split into a different number of patches. In future work, overlapping patches could be used which ensure that all regions get split into the same number of patches (as otherwise we wouldn't be able to align patches). However, this wasn't the main focus of our investigation, so for simplicity I decided to proceed by only using regions in which all the maps split into the same number of patches. Overall, following the procedure outlined in subsection 3.2.4 followed by the same 80-10-10 split, this resulted in:

- 24,093 positive patch pairs for training (80.53% of the full dataset)
- 2,938 positive patch pairs for validation (9.82% of the full dataset)
- 2,886 positive patch pairs for testing (9.65% of the full dataset)

# **Chapter 4**

# **Methodology and Results**

In this section, I outline the different hyperparameters considered for training, and showcase the 3 proxy tasks I have used to evaluate my models, which should help understand the sort of features being captured by the contrastive representations:

- 1. **Positive Pair Identification Tasks**: how good are the representations at identifying true positive pairs? How does the similarity vary between patches?
- 2. **Encoded Features**: what sort of information is encoded within the learnt representations? Does this correspond with what I expect?
- 3. Clustering: is there meaningful structure to the representations in latent space?

This is common when evaluating contrastive models, since then one can see whether the representations capture the underlying structure of the data. For instance, in the SimCLR and BYOL papers, they used the representations to classify images from ImageNet, comparing the performance with a fully supervised model. In this case, evaluation isn't as simple, since I don't have access to any specific labels for each of our patches.

## 4.1 Training the Models

### 4.1.1 Hyperparameters

The choice of encoder is critical for CL, so I opted for investigating ResNet18, ResNet34 and a simple CNN (as described in subsection 2.2.3) as encoders. I also considered the effect of using pretrained ResNet models. Moreover, I varied the learning rate and the batch size (since in the SimCLR and BYOL papers they claim that larger batches lead to better representations). Finally, SimCLR has the temperature parameter in its loss, whilst BYOL has the exponential moving average proportion. I refer to both of these hyperparameters with  $\tau$ ; the reader should understand which hyperparameter is being referred to by context.

### 4.1.2 Training Procedure

In order to select the models for downstream tasks, I performed a total of 20 experiments (see Table C.1), where each hyperparameter was changed at a time, for both a BYOL and a SimCLR model. These changes were relative to a "base model" for each architecture, which used a pretrained ResNet18 encoder, with  $\tau = 0.99$ , a learning rate  $\eta$  of  $1 \times 10^{-3}$ , a patch size of 128 and a batch size of 64. Ideally, I would compare every hyperparameter combination, but this would be unfeasible. Given hardware constraints, I conservatively estimated that training **each** of the models for just 5 epochs would take 64 days (this doesn't include using the CNN encoder, which require less training time than ResNet-based models). See section C.1 for details of this estimate.

All experiments were set to run for 25 epochs on the Informatics Teaching Cluster (given preliminary tests, this was the most that I could afford to do). However, I included early stopping to avoid wasting resources in unnecessary computations. I evaluated each model 150 times per epoch, and stopped training if there was no validation loss improvement in one full epoch, or there was no validation loss improvement in 60 consecutive evaluations after 5 epochs. I wanted to ensure that the models had at least 5 epochs to train, as in preliminary runs models generalised fairly well after 5 epochs. However, if a model got stuck in a local minimum, and its validation loss barely changed within the first 5 epochs, this would have wasted computing resources. All training results can be found in section C.3.

After training, I evaluated each model using the weights which lead to the lowest validation contrastive loss. Even if a lower loss doesn't necessarily correspond with a better model, this might be the fairest way of comparing the models amongst themselves, as I am using the versions which performed best on unseen data.

## 4.2 Positive Pair Identification Tasks

Evaluating each trained model on the subsequent tasks would have been unrealistic (due to memory and time constraints), so I only compared one SimCLR model and one BYOL model on downstream tasks. To decide which models to use, I defined a series of **Positive Pair Identification Tasks** (PPIT), which I executed on the validation set. These classification tasks aimed at showcasing the capacity of the contrastive representations in finding true positive patch pairs, which is the principal goal of this project.

### 4.2.1 Problem Setup

To gauge how well the latent representations comprehend our notion of similarity, we can use them to determine whether 2 patches form a positive pair. Let P,Q be an ordered list of all validation patches (so that  $(P_i, Q_i)$  forms a positive validation pair). We construct 2 matrices  $X, Y \in \mathbb{R}^{N \times 512}$ , where the *i*th rows of X, Y are defined by:

 $X_{i,:} = \text{embed}(P_i)$   $Y_{i,:} = \text{embed}(Q_i)$ 

and embed takes a patch, embeds it using a contrastive model and normalises it to unit length. Then, define a similarity score matrix as  $S = XY^T$ , where  $S_{ij}$  is the similarity

score between  $P_i$  and  $Q_j$  (the cosine similarity between their 2 embeddings). We can find the patch in Q most similar to a  $P_i \in P$ :

$$\hat{Q}_i = \max_{Q_j \in Q} S_{i,j}$$

or find the patches which are most similar to  $P_i \in P$ , based on which  $Q_j \in Q$  are such that  $S_{i,j} \ge \text{sim\_thresh}$ , where  $\text{sim\_thresh} \in [-1, 1]$ .

Due to how I have constructed the positive pairs in the datasets, some patches can appear multiple times in P and Q. For instance, a region present in 4 maps contributes 6 positive patch pairs  $(p_1, p_2), (p_1, p_2), (p_1, p_3), (p_2, p_3), (p_2, p_4), (p_3, p_4)$ , so  $p_2$  appears twice in P and once in Q. Since a patch has a similarity score of 1 with itself, some entries in S will be 1, which is undesirable, as it isn't indicative of how well the model captures similarity. Hence, I masked such entries (by assigning them a large negative value) to ensure that when finding the most similar patch, a trivial patch isn't returned.

#### 4.2.2 Tasks Considered

The first task involves computing **Top-K Accuracy** in identifying the patch  $Q_i \in Q$ which forms a positive pair with patch  $P_i \in P$ . To do this, for each K, I find the Kpatches in Q which attain the largest similarity scores with  $P_i$ . If  $Q_i$  is within these patches, I count this is a correct classification. I considered  $K \in \{1, 5, 10\}$ . Using just Top-1 Accuracy wouldn't be that useful of a metric, due to the fact that for each patch I can have multiple positive pairs; I hoped to mitigate this by also considering Top-5 and Top-10. As a variant to Top-K Accuracy, I instead computed accuracy with respect to identifying **all** true positive pairs, which I call **Positive Pair Accuracy**. To this end, say that a single patch  $P_i \in P$  forms distinct positive pairs with K patches in Q. I then find the K distinct patches in Q with the highest similarity score with  $P_i$ , and compute the proportion of these which form a positive pair with  $P_i$ . Finally, I take the average of these proportions to compute the **Positive Pair Accuracy**.

#### 4.2.3 Results

Table 4.1: Results for the PPIT for each of the BYOL models.

Model	Top-1 Accuracy	Top-5 Accuracy	Top-10 Accuracy	Positive Pair Accuracy
BYOL (Base)	0.22460	0.73962	0.78999	0.72309
BYOL ( $\tau = 0.95$ )	0.22178	0.74108	0.79290	0.72363
BYOL ( $\tau = 0.90$ )	0.22256	0.74759	0.79679	0.73359
BYOL ( $\tau = 0.80$ )	0.22071	0.74118	0.79028	0.72708
BYOL (Not Pretrained)	0.18172	0.55527	0.62722	0.53350
BYOL (ResNet34)	0.21682	0.72611	0.77647	0.71123
BYOL (CNN)	0.17151	0.53972	0.58532	0.52460
BYOL (Patch Size 224)	0.25323	0.87474	0.90572	0.86964
BYOL (Batch Size 32)	0.20924	0.66952	0.72280	0.65270
BYOL $(\eta=1\times 10^{-2})$	0.0048	0.01575	0.02178	0.0146

Model	Top-1 Accuracy	Top-5 Accuracy	Top-10 Accuracy	Positive Pair Accuracy
SimCLR (Base)	0.22168	0.74166	0.79242	0.72557
SimCLR ( $\tau = 0.95$ )	0.21915	0.73320	0.78785	0.71940
SimCLR ( $ au=0.90$ )	0.22538	0.74545	0.79582	0.73155
SimCLR ( $ au=0.80$ )	0.22547	0.75158	0.80010	0.73515
SimCLR (Not Pretrained)	0.22615	0.75790	0.79922	0.74210
SimCLR (ResNet34)	0.22421	0.74263	0.79417	0.72679
SimCLR (CNN)	0.22654	0.76305	0.80272	0.74905
SimCLR (Patch Size 224)	0.25528	0.88904	0.91355	0.88462
SimCLR (Batch Size 32)	0.22217	0.73534	0.78882	0.71740
$\texttt{SimCLR}(\eta=1\times 10^{-2})$	0.22480	0.75012	0.79689	0.73525

Table 4.2: Results for the PPIT for each of the SimCLR models.

The most noticeable fact about the results shown in Table 4.1 and Table 4.2 is that using  $224 \times 224$  patches leads to better performance in all the PPIT. However, comparing these accuracies with those for runs using  $128 \times 128$  patches is unfair: with larger patches, more information is available to distinguish between patches, so better performance doesn't imply better representations. For this project, I focused on the  $128 \times 128$  patches, since they encompassed all the maps, so derived conclusions should be more representative, although use of larger patches should be further explored in future work.

Looking at Table 4.1, BYOL obtains variable results in the PPIT. In some cases, this is because a model suffers from representational collapse: under certain settings, the lack of negative pairs make it so that BYOL learns to map all patches to very similar representation. For example, when using  $\eta = 1 \times 10^{-2}$ , as can be seen in subsubsection D.1.1.9. This harms the model's capacity of identifying when 2 patches are similar. Beyond this, the best BYOL model seems to use  $\tau = 0.90$ , achieving the best results in all metrics except Top-1 Accuracy (where it got the second best result). On the other hand, SimCLR obtains relatively similar results across models, probably due to the stabilising effect of the negative pairs. Surprisingly, the SimCLR model using a CNN encoder obtained the best result in all metrics. Lastly, putting the results of Table 4.1 and Table 4.2 side by side, SimCLR model sattain slightly higher results across all PPIT metrics. Henceforth, I use the BYOL model trained with  $\tau = 0.90$  as the "best BYOL model", and the SimCLR model trained with a CNN encoder as the "best SimCLR model".

I was worried that early stopping hadn't allowed ResNet-based SimCLR models to train for enough time and learn good representations. Thus, I retrained the 2 best ResNet models (ResNet18 without pretraining, and with  $\tau = 0.80$ ) for 15 epochs. Despite a slight decrease in training and validation loss, and a small improvement in PPIT performance, the CNN-based SimCLR model still obtained better results (see section C.4).

#### 4.2.4 Applicability of Results

The metrics from Table 4.1 and Table 4.2 highlight how all the SimCLR models, and a majority of the BYOL models, are adept at identifying similar patches, even if said

patches are unseen by the model. To verify this, I directly inspected the 5 most similar patches found by each model, given some reference patch (more observations available in subsection D.2.1). For the rest of this project, when discussing a patch  $P_i$  (which I'll interchangeably name "reference patch" or  $P_i$ ), I'll refer to a patch  $Q_j$  as a "positive patch" if  $(P_i, Q_j)$  constitute a positive patch pair, and as a "negative patch" otherwise.

I observed that the learnt contrastive representations are capable of understanding how many different features (roads, buildings, trees, streets, etc...) are composed together to generate a patch. Whilst this might seem fairly trivial, given the variability in styles across the maps, the capacity for generalisation of these models is impressive. For instance, they seem to understand differences between roads with trees to the side, and city roads. Moreover, the models seem to find positive pairs, even if there isn't an immediately obvious visual similarity (see Figure D.4).

Furthermore, I noticed that, generally, SimCLR identified all positive patches correctly as the most similar patches (if  $P_i$  had 3 positive patches, these were often deemed as the 3 most similar). Contrarily, BYOL wasn't as discriminative, and sometimes the most similar patch wasn't part of the correct positive patches. This is somewhat perceptible when looking at positive pair accuracy in Table 4.1 and Table 4.2. However, when looking at the most similar negative patches, BYOL was better at finding "the next best patch". For instance, in Figure 4.1 and Figure 4.2 one can observe how BYOL has found patches which are more nuanced, and share more features with the reference patch. This indicates that SimCLR based models can be better at meticulously retrieving positive pairs, whilst BYOL might find more diverse, albeit semantically consistent patches.



Figure 4.1: Both SimCLR and BYOL identify the correct positive pairs, and assign to them a notably higher similarity score. However, the most similar negative patches found by SimCLR aren't that great: whilst they contain railtracks, they are visibly semantically dissimilar. On the other hand, both positive and negative patches found by BYOL contain a high degree of semantic similarity, staying true to the railtrack style and positioning.



Figure 4.2: Both SimCLR and BYOL identify 2 of the 3 correct positive patch pairs (although they can't find the darker shaded one, despite having identical building structure to the other positive pairs). SimCLR assigns much higher similarity score to the true positive patches, whereas BYOL gives fairly consistent scores throughout. The patches found by BYOL seem more semantically correct, as most of them show a diagonal road (with text) in the rightmost 2/3 of the patch, with buildings at its sides. On the other hand, SimCLR finds visually similar buildings, but which convey different information.

Overall this reveals how the contrastive models can be utilised to "lookup" patches which share a set of features across a variety of different maps, which can allow cartographers or historians to efficiently search for similarities and differences in maps across different eras.

Another potential usage for these methods which I wasn't able to explore in this project is unsupervised map alignment. For instance, say we have patches corresponding to an unseen historical map. Then, I can search through a database of all the contrastive representations embeddings, and for each unknown patch, find the K most similar corresponding patches (here K represents the number of distinct hisotrical time periods encompassed by the maps in the database; in our case K = 4). Then, one can take a majority vote between the locations of the found patches, to determine the longitude and latitude to assign to the unknown patches. Measures could be taken to ensure a consistency between the coordinates assigned to adjacent unknown patches (i.e 2 patches adjacent in the map should be assigned adjacent longitudes and latitudes).

### 4.2.5 Visualising Similarity Scores

To understand the similarity score distributions of the models, I visualised the similarity score matrices from subsection 4.2.1 as grayscale images where pixel intensity corresponds to similarity score. I also randomly sampled 100 rows in *S*, and plotted the corresponding  $10,285 \times 100 = 1,028,500$  scores as a histogram using 100 bins (excluding masked entries).



(a) Similarity score distribution for the SimCLR model with CNN encoder.



(b) Similarity score distribution for the BYOL model with  $\tau = 0.90$ .

Figure 4.3: Darker colours in the similarity score matrices represent higher similarity scores (closer to 1). Notice the white pseudo-diagonal, generated by the mask applied to ensure that I don't take the similarity score of identical patches into account. The y-axis of the histograms corresponds to the proportion of scores which are part of a certain bin. Distributions for all other models are available at section D.1.

In general, all SimCLR models had a strongly left-skewed similarity score distribution (Figure 4.3a exemplifies this). The type of encoder seemed to influence the range of the distributions: with a ResNet, similarity scores were in the range (0.2, 1), whilst with a CNN scores were in the range (-0.1, 0.95). Moreover, all ResNet-based SimCLR models had a small peak in the last bin of the distribution (due to assigning scores above 0.99), whilst the CNN-based SimCLR model is monotonic. Lastly, when the ResNet wasn't pretrained, the similarity score distribution became symmetric.

On the other hand, BYOL models displayed a more notable distributional difference, although generally they were slightly right-skewed, as shown in Figure 4.3b. For collapsed representations (like when using  $\eta = 1 \times 10^{-2}$  or a batch size of 32), the scores were extremely right-skewed, since most pairs obtained a similarity score close to 1. Strangely, both ResNet34-based and CNN-based BYOL models had a bimodal distribution. I also noticed that score distribution ranges followed a similar trend as with SimCLR models: a CNN encoder lead to wider score ranges (including negative scores), whilst ResNet encoders assigned positive scores above 0.2.

Since the distributional differences between SimCLR and BYOL generally apply independent of hyperparameter choice, they could be due to architecture choice, particularly the use of negative pairs in training. In SimCLR they push representations to be more discriminative; however, BYOL solely "sees" positive pairs, so representations might move towards pursuing higher similarity, rather than similarity with positive pairs, and dissimilarity with negative pairs. This is noticeable when comparing Figure 4.3a and Figure 4.3b, whereby the BYOL similarity score matrix is darker, indicating it is assigning higher scores to every pair. Moreover, negative pairs could provide a stabilising effect; without them, high learning rates or small batch sizes might lead to local minimum convergence, which for BYOL could be representational collapse.

## 4.3 Exploring Encoded Features

### 4.3.1 Classification Tasks

To explore how information was encoded in the representations, I defined two classification tasks, where I classify patches based on time period and amount of information, based solely on their contrastive representations. I used an MLP classifier (call it MLP\_C), with a 512-dimensional input, followed by 3 fully-connected, hidden layers (1024, 2048 and 512 hidden units respectively), each with bias and ReLU activation. The output uses a linear layer with bias and softmax activation. I trained for 5 and 50 epochs (to see the effect of longer training time), using a batch size of 128, Adam optimiser, cross-entropy loss and learning rate of  $1 \times 10^{-3}$ . After removing duplicate patches, I obtained 61,647 training patches, 7,689 validation patches and 7,697 testing patches.

#### 4.3.1.1 Temporal Awareness

Positive pairs were constructed using temporally-spaced patches, so I expected temporally invariant contrastive representations. Ideally, such embeddings will encode structural patch elements, and compress out stylistic aspects characteristic of maps from a given time period. I used the NLS metadata to define 4 temporal classes for the maps: before 1900, 1900-1910, 1910-1930, and after 1930. This ensures balanced classes: the first 3 classes contain 57 maps each, whilst the final class contains 41 maps (see section 3.1). Then, I trained MLP\_C to predict patch time period from their embedding. Table 4.3: Training, validation and testing data distribution for the time prediction problem.

Class [Year]	0	1	2	3
	[< 1900]	[< 1910]	[< 1930]	[> 1930]
Training Samples	16,426	16,600	16,767	11,854
(Proportion)	(26.65%)	(26.93%)	(27.20%)	(19.23%)
Validation Samples	2,033	2,073	2,115	1,468
(Proportion)	(26.44%)	(26.96%)	(27.51%)	(19.09%)
Testing Samples	2,063	2,080	2,099	1,455
(Proportion)	(26.80%)	(27.02%)	(27.27%)	(18.90%)

If MLP\_C obtains an accuracy close to a random classifier (25%), this could indicate that the embeddings are temporally indifferent. However, perhaps the patches don't contain temporal information. Thus, I define an end-to-end classification problem using CNN\_C, a simpler CNN encoder than those used by the contrastive models. If CNN\_C converts patches into embeddings from which MLP\_C successfully predicts time period, this indicates the presence of temporal information in the patches. CNN\_C takes in a  $128 \times 128$  patch, and is comprised of 2 convolutional layers (learning 32 and 64)

#### Chapter 4. Methodology and Results

filters respectively), each with ReLU activation, batch normalisation and max pooling, followed by global average pooling, and a linear layer with bias, which outputs a 512dimensional embedding. Finally, I defined 3 baselines (uniform random model, random model sampling according to training proportions and a most common class model).

Table 4.4: Results of the time period prediction problem, over 5 and 50 epochs. I include the best training and validation performances, alongside the test performance of the model achieving the best validation accuracy.

Classifier	Training Accuracy		Validation	Accuracy	Testing Accuracy	
Classifier	5 epochs	50 epochs	5 epochs	50 epochs	5 epochs	50 epochs
Uniform Random	0.2500		0.2500		0.2500	
Weighted Random	0.2549		0.2547		0.2553	
Most Common Class	0.2720		0.2751		0.2727	
SimCLR Embedding + MLP_C	0.5313	0.9844	0.4973	0.5252	0.5027	0.5095
BYOL Embedding + MLP_C	0.375 0.4141		0.2965	0.3811	0.3077	0.3857
CNN_C Embedding + MLP_C	0.8672		0.6700		0.6592	

The results from Table 4.4 imply that the BYOL model is good at ignoring temporal differences between patches, as its representations perform poorly on the time prediction task. In fact, after training I observed that MLP\_C primarily predicted classes 0 and 2 throughout all 3 datasets, implying that it struggled to find features in the BYOL representations to predict the patch's time period. Furthermore, MLP\_C doesn't perform well on validation/testing with SimCLR representations, but overfits to the training representations, particularly after 50 epochs. Lastly, when MLP\_C uses CNN\_C it significantly outperforms both contrastive models, even when run for only 5 epochs.

This indicates that the simple CNN\_C finds temporal information in the patches (probably due to the characteristic styles of maps from different time periods). I hypothesise that the contrastive models also somewhat encode temporal information, but this isn't their primary goal. This can be seen from Table 4.4, whereby training for longer allows MLP\_C to more fruitfully extract temporal features from the contrastive representations of both models, leading to a sizeable performance improvement.

Given that MLP\_C overfits with SimCLR embeddings, but generally predicting the same labels with BYOL embeddings, this indicates that BYOL learns more temporally invariant representations than SimCLR. Perhaps SimCLR better encoding the characteristic stylistic features of the different maps, and is thus better at temporal discimination. Alternatively, BYOL might better abstract away from these features, and is thus learning more semantic representations. Beyond architectural differences, this could be due to the encoder used. Complex ResNets should better generalise to these characteristic features, whereas the simpler CNN encoder might struggle more to ignore the stylistic differences.

This could help explain how in Figure 4.3 (and section D.1), the CNN-based SimCLR model had a wider similarity score distribution, with all similarity scores below 0.95, whilst ResNet-based models had a narrower similarity score distribution, but with many similarity scores above 0.99. If CNN-based models better encode temporal information, stylistic differences between patches might prevent patch pairs from attaining extremely

high similarity scores, whereas the higher abstraction capabilities of ResNets makes the models perceive more patches as highly similar.

#### 4.3.1.2 Information Awareness

If we apply edge detection to the patches as in subsection 3.2.4, and count the pixels in detected edges, this roughly describes patch informativeness, as patch features are primarily simple lines (generally, more edge pixels will imply more features). Intuitively, a patch with a letter is less informative than a patch with many trees, as it has less learnable features. I use MLP\_C to classify contrastive representations based on edge-detected pixel count to gauge informativeness encoding in the embeddings.

Initially, I defined the classes using equally spaced ranges for pixel counts (for example, class 2 could be composed of patches with 1000-1500 edge pixels). However, I visualised the edge pixel distribution in training and validation patches, and it was quite left-skewed (since patches with few features are more common). Thence, this approach would have lead to class imbalance, hindering classification. Consequently, I arbitrarily chose to use 5 classes, and found 5 thresholds to split the training data into roughly equally sized subsets (see section C.5 for implementation, and Table 4.5 for the thresholds). Class labels for validation and testing were generated using these thresholds. As in subsubsection 4.3.1.1, I compared the performance with 3 baselines, but didn't use the end-to-end encoder (this task would be trivial even for CNN\_C).

Class [Threshold]	0	1	2	3	4
	[≤453]	[≤969]	[≤1864]	[≤ 2995]	[> 2995]
Training Samples	12,352	12,320	12,319	12,335	12,321
(Proportion)	(20.04%)	(19.98%)	(19.98%)	(20.01%)	(19.99%)
Validation Samples	1,611	1,670	1,533	1,404	1,467
(Proportion)	(20.95%)	(21.72%)	(19.94%)	(18.26%)	(19.08%)
Testing Samples	1,514	1,709	1,573	1,468	1,433
(Proportion)	(19,67%)	(22.2%)	(20.44%)	(19.07%)	(18.62%)

Table 4.5: Training, validation and testing data for the information prediction problem.

Table 4.6: Results of the information prediction problem, over 5 and 50 epochs. I include the best training and validation performances, alongside the test performance of the model achieving the best validation accuracy.

Classifiar	Training Accuracy		Validation	Accuracy	Testing Accuracy		
Classifier	5 epochs	50 epochs	5 epochs 50 epochs		5 epochs	50 epochs	
Uniform Random	0.2000		0.2022		0.2035		
Weighted Random	0.2000		0.2000		0.2000		
Most Common Class	0.2004		0.2095		0.1967		
SimCLR Embedding + MLP_C	0.8672	1.000	0.8486	0.8504	0.8433	0.8424	
BYOL Embedding + MLP_C	0.7656	0.9141	0.7602	0.7930	0.7502	0.7819	

Looking at Table 4.6, running MLP\_C for 5 epochs leads to consistent accuracy for both models across all datasets, indicating that the contrastive models generate representations which generalise well to unseen data (at least when encoding information amount).

Longer training leads to overfitting, implying that both contrastive models are adept at encoding information in a fine-grained manner. Nonetheless, SimCLR representations lead to the best performance across all datasets, so it might encode information more precisely, or dedicate more features to represent the amount of information in a patch.

Architecture choice is probably responsible for this performance gap. SimCLR uses the interplay between positive and negative pairs to generate representations, so effective encoding of information amount is a simple way of assessing rough similarity (for instance, it is easy to differentiate between a patch full of building and a patch with a single letter). This can lead to it learning more discriminative representations. Contrarily, BYOL's loss relies on the similarity between the target network encoding and the online network's predictor encoding. Since the target network's weights are an exponential moving average of the online network's, the online network won't be pushed as hard to learn information amount, as then matching the target encoding becomes harder (as the target network is using "old" weights). This might lead to BYOL learning more semantic representations, which could explain the results in Table 4.6: BYOL prioritises "knowing" if in a patch there is a "road" or a "building", alongside its rough orientation and placement; factors such as shading or size might become secondary, so it becomes less precise when encoding the amount of information in a patch.

### 4.3.2 Invariance Under Transformations

In CL, image augmentations are typically used to generate positive pairs, but I used temporal differences. Thus, I wanted to see how augmentations affect the contrastive representations. I considered augmentations inspired by the SimCLR and BYOL papers: centre/random cropping, fixed/random rotation angle, gaussian blur, colour jitter, horizontal flip and grayscaling (see implementation in section C.6). Then, for positive pairs  $(P_i, Q_i)$ , I applied each transformation to  $Q_i$ , and computed the similarity scores between  $P_i$  and each transformed  $Q_i$  (which I call the *transformed similarity score*).

Table 4.7: Original and standardised MSE between similarity scores before and after transformation. I have ordered the table in descending order of MSE.

Transformation	BYOI	L MSE	SimCLR MSE			
11 ansior mation	Original Standardised		Original	Standardised		
Rotation (30°)	0.06816951	1.000000	0.3952508	1.000000		
Horizontal Flip	0.06319004	0.9267467	0.3585143	0.9069046		
Random Rotation $(90^\circ, 180^\circ, 270^\circ)$	0.04029370	0.5899183	0.1678344	0.4236947		
$\begin{array}{c} \text{Random Crop} \\ (64 \times 64) \end{array}$	0.022805834	0.3326539	0.1540349	0.3887248		
Centre Crop $(64 \times 64)$	0.02073835	0.3022392	0.14532146	0.3666436		
Gaussian Blur	0.005725238	0.08138097	0.048866849	0.1222139		
Colour Jittering	0.001208771	0.01493915	0.007873815	0.01833176		
Grayscaling	0.0001932637	0.000000	0.0006399037	0.000000		

To score the difference between transformed and untransformed similarity scores (simi-

larity scores of  $P_i$  and  $Q_i$  without any transformation), I computed their Mean Square Error (MSE) (lower MSE indicates representations were less affected by the augmentation). The MSE exacerbates large differences, which should clarify which transformations affect the contrastive representations most. I standardised the MSEs (between 0 and 1) to make them more comparable between models.

The results from Table 4.7 show that rotations and random flips are the transformations which most notably alter the representations produced by both contrastive models. These seem to affect BYOL more significantly (higher standardised MSE), particularly with random rotations. This corresponds with what can be observed in Figure 4.1, where orientation is critical for querying the most similar patches (for instance, the most similar patches to a patch containing a road are patches containing roads which are positioned and oriented in the same way; see Figure D.2 and subsection D.2.1).

Conversely, colour transformations (gaussian blur, colour jittering, grayscaling) seem to have a negligible effect on the contrastive representations (particularly for BYOL). This also makes sense: ignoring colour and resolution differences between maps is critical to learn good contrastive representations.

Generally, the results in Table 4.7 showcase desirable properties for the representations. Colour invariance is important for finding similar patches in maps from different time periods, and sensitivity to orientation/location is imperative for map-related tasks (differences in these attributes greatly change the semantics of a patch).

However, cropping seems to substantially affect representations, which could indicate sensitivity to feature size (since patches are resized after cropping), or that the models aren't fully attending to local features. Encoding fine-grained local structure is critical for patch representation, as it provides a discriminatory nuance which can better align human and model judgement. Hence, it could have been useful to include cropping as an augmentation for this project. Alternatively, I could have used the global and local contrastive losses suggested by Chaitanya et al. [20], or do as Agastya et al. [26], and provide a large contextual patch to help directly learn local features.

## 4.4 Geographical Awareness

### 4.4.1 Geographically-Aware Representations

#### 4.4.1.1 Motivation

I thought I'd be able to improve the contrastive representations by imbuing them with both visual and geospatial information. This was inspired by Ayush et al. [28], where they learnt contrastive representations with both a contrastive and classification objectives. They clustered images by longitude and latitude with K-Means, and used the representations to predict the cluster. However, I didn't think this would work for my project: their images ranged over the whole world, whilst our maps encompass a small, neatly subdivided rectangular region. Hence, clustering the patches' longitude and latitude wouldn't be too informative, as it wouldn't preserve local structure (for example, 2 very different areas, like city and grasslands, could be clustered together).
#### 4.4.1.2 GeoSimCLR

I felt that for this project, directly imbuing geographical information within the contrastive representations was the way to go. I devised GeoSimCLR, a new architecture which learns latent representations using 2 separate contrastive objectives. I based myself on the SimCLR architecture, although it could be adapted for BYOL.

GeoSimCLR learns a latent geocontrastive representation  $\underline{z} \in \mathbb{R}^d$  for an image which is composed of 2 distinct parts. The "upper-half" of  $\underline{z}$  is a visual representation  $\underline{z}_v \in \mathbb{R}^{d/2}$ encoding visual similarity (akin representations learnt by standard SimCLR models). The "lower-half" of  $\underline{z}$  is a contextual representation  $\underline{z}_c \in \mathbb{R}^{d/2}$  encoding contextual similarity, and should maximise the similarity of a patch with its directly adjacent neighbours. For instance, given 2 neighbouring patches  $p, r, \underline{z}_c^p$  and  $\underline{z}_c^r$  should be similar in (contextual) latent space. To do this, I use 3 patches: a reference patch  $p_1$ , a visual patch  $p_2$  (a patch of the same region as  $p_1$ , but from a different map) and a contextual patch r (a patch directly adjacent to  $p_1$ , which is randomly sampled). Passing them through an encoder f generates representations  $\underline{z}^{p_1}, \underline{z}^{p_2}, \underline{z}^r \in \mathbb{R}^d$ . The visual  $(\underline{z}_v^*)$  and contextual  $(\underline{z}_c^*)$  representations get passed through different projectors  $g_v, g_c$ , to generate projections  $\underline{q}_v^*, \underline{q}_c^*$ . Then, a visual contrastive loss  $\ell_v$  is computed between  $\underline{z}_v^{p_1}$  and  $\underline{z}_v^{p_2}$ , and a contextual contrastive loss  $\ell_c$  is ocmputed between  $\underline{z}_c^{p_1}$  and  $\underline{z}_c^r$  to generate a contextual contrastive loss. The geocontrastive loss is a weighted sum of these 2 losses:

$$\ell = w_v \cdot \ell_v + w_c \cdot \ell_c, \quad w_v \in [0,1] \quad w_c = 1 - w_v$$

I chose  $w_v = 0.7$ , since I wanted to prioritise learning good visual representations, but optimising  $w_v, w_c$  to suit specific tasks could be further investigated.



Figure 4.4: Sketch of how GeoSimCLR uses 2 separate contrastive losses to learn latent representations: one for visual similarity, and one for contextual similarity.

#### 4.4.1.3 PPIT with GeoSimCLR

To train the GeoSimCLR model, I used a CNN encoder (as it gave the best results in the PPIT from subsection 4.2.2), and followed the procedure from subsection 4.1.2, although without early stopping. Since visual and contextual representations are smaller (256-dimensional), encoding useful information should be harder, so I wanted to allow

more training time. Nonetheless, from epoch 13 onwards the best validation loss barely changed, so I think that training for all 25 epochs wasn't that necessary, particularly since a CNN might not benefit as much as a ResNet from longer training.



(a) Similarity score distribution for the GeoSimCLR model.



(b) Similarity score distribution for the visual representation from the GeoSimCLR model.



(c) Similarity score distribution for the contextual representation from the GeoSimCLR model.

Figure 4.5: The visual similarity score distribution is akin to that showcased in Figure 4.3a. The contextual similarity score distribution remains left-skewed due to the contrastive objective, but since different patches are likely to share a lot more contexts, contextual similarity is likely to be higher in general. Notice how the contextual representation heavily influences the geocontrastive representation.

I then evaluated the model with the best validation loss on the PPIT. Surprisingly, using the full geocontrastive representations lead to extremely poor results in the PPIT. These tasks weren't defined to evaluate contextual similarity, so perhaps including a contextual representation harmed the similarity scores. I was able to confirm this by visualising the similarity score matrix for the GeoSimCLR model, alongside the similarity score matrices for the visual and contextual representations (see Figure 4.5). Therefore, I also evaluated the visual representations on the PPIT.

Moreover, I defined a weighted geographical similarity matrix as a weighted sum of the visual and contextual similarity matrices. From Figure 4.5a, the geocontrastive similarity score matrix seems to be most similar to the contextual similarity score matrix in Figure 4.5c. With this weighted sum I hoped to push similarity scores in the direction of visual similarity, whilst still encoding some contextual information. For the following results, I used 0.9 as a weight for the visual similarity score matrix, and 0.1 as a weight for the contextual similarity score matrix (I also tried 0.7,0.75,0.8,0.85,0.95, but I found that 0.9 resulted in a good balance between visual and contextual similarity). Table 4.8: Results for the PPIT. I compare the 2 best contrastive models with GeoSimCLR.

Model	Top-1 Accuracy	Top-5 Accuracy	Top-10 Accuracy	Positive Pair Accuracy
Best SimCLR	0.22654	0.76305	0.80272	0.74905
Best BYOL ( $\tau = 0.90$ )	0.22256	0.74759	0.79679	0.73359
GeoSimCLR	0.01925	0.07059	0.09334	0.06529
Visual GeoSimCLR	0.21079	0.69965	0.76296	0.68085
Weighted GeoSimCLR	0.17151	0.57608	0.68284	0.54171

#### 4.4.1.4 Future Improvements

Preliminarily, the results in Table 4.8 aren't spectacular: adding contextual information seems to hinder the model's positive pair identification capacity. Moreover, there is a significant accuracy difference when comparing GeoSimCLR's visual representations with the representations from SimCLR. I believe that this is likely due to SimCLR learning larger representations (512-dimensional) than the visual representations learnt by GeoSimCLR (256-dimensional), which allows better encoding of key details. This could indicate that larger representations lead to better contrastive encodings. In retrospect, and keeping this in mind, I can think of a variety of improvements, which would allow GeoSimCLR to incorporate both visual and contextual information more effectively.

Firstly, the contrastive objective could be modified to ensure that visual and contextual representations are learnt more cohesively: since projection and contrastive loss utilise different parts of the representation, these don't work synergistically in representing similarity. Ideally, I'd like visually similar patches appearing in similar contexts to obtain the highest similarity scores, which isn't currently happening. To enforce this, instead of splitting the representation, I could pass the whole representation through 2 distinct projectors, and apply the visual and contextual contrastive losses to these 512-dimensional representations. The main drawback is that visual and contextual representations won't be distinguishable.

Secondly, using a CNN encoder for GeoSimCLR could be too ambitious, particularly when gauging contextual information, so the enhanced feature extraction capabilities of a ResNet could be more appropriate.

Thirdly, it seems that the contextual representation is getting too highly weighted, especially after normalising the geocontrastive represention to compute similarity scores, so perhaps I could reduce the number of features involved in encoding contextual information (i.e only use 10% of the features, instead of 50%).

Lastly, the data could be changed. Instead of randomly sampling contextual patches from different positions for each visual positive pair, I could use the same contextual patch position for visual patch pairs of the same region (for example, for a given region, always use the upper right neighbouring patch). This would reduce the variability of contextual patches to which the model is exposed to, which should hopefully improve the sharpness of the contextual representations. Alternatively, perhaps the data itself isn't too representative of context, so the contrastive representations are encoding other features unrelated to context. To fix this, perhaps a more selective contextual sample procedure could be used (for example, if a road is divided into 2 patches, make sure these patches appear as contextual pairs), or an alternative way of providing context could be used (for instance, use a  $256 \times 256$  patch which contains the reference patch).

#### 4.4.2 Clustering

The contrastive objective gives us representations which should be close in latent space, so it is natural to ask whether there is any structure to them within the space. To investigate this, I clustered the different contrastive representations, using 2 methods. The first one was K-Means, due to its simplicity. However, it requires that I define the number of clusters. Therefore, I also considered HDBSCAN [37] (Hierarchical Densitiy-Based Spatial Clustering of Applications with Noise): not only does it automatically find the number of clusters, it can also detect outliers, and has soft clustering capabilities. For K-Means, I employ the scikit-learn [38] implementation, whilst for HDBSCAN, I employ the implementation developed by McInnes, Healy, and Astels [37].

Since contrastive models use cosine similarity in latent space to define representation similarity, it is the most natural measure with which to cluster these representation. However, neither K-Means nor HDBSCAN have cosine distance as an available metric. Thus, for clustering, I normalised all the representations to unit vectors; then Euclidean distance becomes proportional to cosine distance. Indeed, if  $\underline{x}, \underline{y}$  are unit vectors, the larger the cosine similarity, the smaller the Euclidean distance:

$$\begin{aligned} \|\underline{x} - \underline{y}\|_{2}^{2} &= (\underline{x} - \underline{y})^{T} (\underline{x} - \underline{y}) \\ &= \underline{x}^{T} \underline{x} - 2 \underline{x}^{T} \underline{y} + \underline{y}^{T} \underline{y} \\ &= \|\underline{x}\|_{2}^{2} + \|\underline{y}\|^{2} - 2\cos(\theta) \|\underline{x}\|_{2} \|\underline{y}\|_{2} \\ &= 2(1 - \cos(\theta)) \end{aligned}$$

I also considered an alternative pipeline: first, reduce the dimensionality of the embeddings (I considered 2,3,5,10, 15 and 100 dimensions), and then cluster the resulting lower-dimensional embeddings with Euclidean distance. For dimensionality reduction, I considered 2 methods: Principal Component Analysis (PCA), since it is simple and fast (using the scikit-learn [38] implementation) and UMAP [39], since it is good at preserving local and global structure (using the implementation by McInnes, Healy, and Melville [39]).

#### 4.4.2.1 Clustering Validation Data

I began by clustering the 5,538 unique patch representations from the validation data using HDBSCAN (manually specifying cluster number for K-Means meant that dissimilar patches were often clustered together). Since clustering all these 512-dimensional representations directly took too long, I applied dimensionality reduction. UMAP produced semantically consistent clusters (meaning that patches in a cluster are semantically similar) when using 10-100 features, although with less features the number of clusters and outliers decreased (see Table C.6 for details). PCA generated small clusters and deemed most representations as outliers, regardless of the number of features used. Thus, I used 15 features for dimensionality reduction, alongside HDBSCAN for clustering.

Generally, BYOL-derived clusters tended to be very semantically consistent, whilst SimCLR-based clusters often included anomalous patches (for instance, clusters which predominantly had patches with trees sometimes contained patches with buildings). Surprisingly, GeoSimCLR clusters mainly contained patches from a single map style, without much regard to visual content (I found a cluster containing patches with rail tracks, trees and buildings which seemed to come from the same map). This could indicate that contextual representations learnt by GeoSimCLR are encoding map style instead of context. Cluster examples available at subsection C.7.5.

PCA clusters were extremely semantically consistent for BYOL representations (I could immediately come up with a textual label to describe cluster content) and distinct. Conversely, different clusters generated with UMAP contained similar patches (although clusters remained semantically consistent). Nonetheless, they were impressively nuanced, particularly with BYOL and SimCLR. For instance, a variety of clusters contained rail tracks, differentiated by orientation and positioning (cluster 24 had horizontal rail tracks on the top of the patch, while cluster 57 had rail tracks at the bottom going diagonally from left to right). I also found clusters with natural elements (small or large number of trees, trees alongside roads, mountains, grasslands), city-based (building blocks, roads) and containing text.

Evaluating clustering systematically, particularly with unlabelled data, is a challenge. In this case, I have exploited the canonical similarity score of the contrastive representations to visualise how scores are distributed over clusters. In particular, for each cluster (excluding outliers), I computed the similarity score between each of its patches, grouped all these scores together, and plotted a histogram using 100 bins.



(a) UMAP with 15 features and HDBSCAN (excluding outliers).



(b) PCA with 15 features and HDBSCAN (excluding outliers).

Figure 4.6: Similarity score distribution across all clusters after using UMAP/PCA and HDBSCAN. The y-axis corresponds to the proportion of similarity scores falling in a bin.

The distributions in Figure 4.6 are fairly different to those from Figure 4.3 and Figure 4.5. With UMAP, all distribution become markedly more right-skewed; with PCA this is only the case with BYOL. These distributions correspond with what I observed above, whereby BYOL seems to form highly similar clusters with both PCA and UMAP, whilst SimCLR clusters benefit from the complexity of UMAP. This could indicate that BYOL generates representations in a simpler or more structured manner, such that even a simple method like PCA is capable of reducing the dimensions whilst preserving similarity information.

#### 4.4.2.2 Visualising Contextual Representations

Following from subsubsection 4.4.2.1, I wanted to see whether GeoSimCLR's contextual representations were encoding stylistic features instead of contextual features. If this were the case, then high similarity scores will be biased towards patch pairs from the same map style, which would hinder the PPIT of the geocontrastive representations, as observed in Table 4.8. To this end, I fetched all the patches corresponding to the same region from the training, validation and test datasets (which came from 3 or 4 distinct maps). I converted these into a geocontrastive representations, applied UMAP and PCA to convert the geocontrastive, visual and contextual representations into 2-dimensional vectors, which I finally clustered with HDBSCAN.



(a) Clusters for region 8, represented by 4 maps.



(b) Clusters for region 17, represented by 4 maps.



(c) Clusters for region 52, represented by 3 maps.

Figure 4.7: Depiction of contrastive representations in 2 dimensions for 3 different regions. The colours represent the cluster to which a representation is assigned. From left to right, the geocontrastive, visual and contextual representations.

The cluster depictions from Figure 4.7 were produced using UMAP + HDBSCAN (additional results in subsection C.7.7). I chose these 3 regions arbitrarily, but observed the same pattern throughout all other regions I tested: UMAP neatly separated contextual representations in embedding space according to the number of maps for a given region. Moreover, each cluster was composed predominantly by patches from a single map, with just 1 or 2 outliers. This strongly indicates that the contextual representations are capturing stylistic information from the maps, instead of contextual information.

Retrospectively, this makes sense, since it is likely that the contextual patches don't share many visual similarities with the reference patches; in order to make these 2 patches similar, GeoSimCLR might simply default to focusing on patch style, since things like map colour, line types, etc... will be shared between the reference and contextual patches. As can be seen in Figure 4.7 this can heavily affect the geocontrastive representations. For regions 8 and 52, despite the visual clustering showcasing clear visual differences, the geocontrastive representations get tightly clustered in similar fashion to the contextual representations.

Admittedly, one must be careful: conclusions about lower-dimensional representations might not apply in the original representation space. However, the consistency of the pattern across different regions, alongside the fact that even PCA can pick up on these distinct contextual clusters (see subsection C.7.7) provides good evidence that my conclusion is valid. Moreover, taking the geocontrastive representations for the different maps of a region, I computed their similarity score matrix, which revealed that contextual similarity is significantly higher amongst representations coming from the same map, whilst visual similarity remains relatively uniform (see Figure C.16).

#### 4.4.2.3 Segmentation Through Clustering

Another way of illustrating structure in latent space is by clustering patches belonging to a single map. By colouring the patches according to cluster, and then reconstructing the map, good representations should reveal the semantic differences between different regions of a map. For this, we took all patches corresponding to a given map from training, validation and testing data. I clustered using HDBSCAN and K-Means (with K = 4). I only used UMAP with 15 features for dimensionality reduction. Since the number of patches was small, I also used HDBSCAN to cluster directly in latent space.



(a) Segmentation using HDBSCAN in latent space.



(b) Segmentation using UMAP (15 features) + HDBSCAN.



(c) Segmentation UMAP (15 features) + K-Means (K = 4).

Figure 4.8: Segmentations for map 82877412. Outliers for HDBSCAN were assigned a cluster by using soft clustering. These are denoted by a red box around the patch. From left to right, the segmentations use SimCLR, BYOL and GeoSimCLR representations. Segmentations for different maps of the same region available at subsection C.7.9.

From Figure 4.8, it seems that the most semantically significant segmentations are obtained by using HDBSCAN directly with the BYOL representations. The dark blue cluster corresponds to nature, the orange cluster to buildings and the light blue cluster corresponds to railtracks. The other segmentations are either too simple (like using UMAP + HDBSCAN with SimCLR representations, or HDBSCAN directly on GeoSimCLR representations), or put semantically different patches in the same cluster (for example, the green cluster for BYOL representations using UMAP + K-Means, which contains nature, buildings and railtracks).

Notice that in all SimCLR segmentations, there is a distinct region corresponding to grasslands which is always generated as a cluster (the green cluster when using UMAP + K-Means). However, this isn't found by any other method (only by BYOL with UMAP + HDBSCAN). This hints at the high discriminative capacity of SimCLR, since the grassland patches are particularly distinctive.

It seems that the best segmentation strategy with SimCLR is using UMAP + K-Means, as it generally identifies the distinct regions of the map well (although some buildings are put as part of the nature cluster, and a part of the railtracks are included within the buildings). For GeoSimCLR, UMAP + HDBSCAN seems to work best.

# **Chapter 5**

# **Discussion and Conclusions**

## 5.1 Main Conclusions

In conclusion, I belive that I have reasonably demonstrated the effectiveness of CL in encoding features in temporally-spaced historical maps. I have systematically compared how different hyperparameter settings influence the capacity of SimCLR and BYOL to learn useful representations, which allow them to recall true positive patch pairs. I have observed that SimCLR is generally better at correctly identifying true positive pairs, and that it learns in a more robust manner than BYOL, which given certain hyperparameter settings, can collapse its representations. Nonetheless, I have showcased how the positive pair finding capacity of the models can be useful when exploring similarities across different maps, and I propose a framework for using these learnt representations to align unseen maps.

I visualised the similarity score distributions for the 2 models, and observed a stark differences with regards to the range and shape of the distributions, likely due to the choice of encoder, alongside the use of negative pairs during training.

I was quite surprised that, according to the PPIT, the best SimCLR model for these tasks seemed to employ a simple CNN encoder. Moreover, I have observed that even if I train more complex, ResNet-based models for longer, the performance in these evaluation metrics isn't too significant. On the other hand, for BYOL using a ResNet-based model substantially improved performance in these tasks.

I also sought to better understand how these different models encode the features. Throughout the proposed tasks, I have observed a significant contrast in the sort of patches that the different models find similar. SimCLR seems to be a lot more discriminative, which allows it to more decisively gauge whether 2 patches are similar or dissimilar. On the other hand, BYOL seems to learn more semantically oriented representations, as it seems to be more capable at abstracting away broad features from the patches.

SimCLR representations seem to have encoded stylistic nuances present in the maps (which allow it to more readily discriminate when 2 patches are from the same year of production), whilst BYOL seems to be fairly invariant to these stylistic features.

Moreover, SimCLR is also better at encoding the degree of information present within patches than BYOL.

To further explore the factors influencing the learnt features, I applied transformations to the patches, from which I observed that the representations for both models are most sensitive to the orientation and location of patch attributes, whereas transformations which change the colour or blurriness of a patch have the least effect. I believe that these features are highly desirable for an application involving maps. However, I observed that the representations are also fairly influenced by cropping, which can indicate that the models are not gauging local features well, and are instead encoding feature combinations as a whole.

I developed a new contrastive framework, GeoSimCLR, with the hope of learning contrastive representations which encode both visual and contextual information. I hoped that these representations would be more useful for map-based applications, were understanding the surroundings of a given patch can be critical to finding patches which are similar, and appear in similar contexts. However, I found that this combination of visual and contextual information ultimately hurt the performance of the representations, particularly since the contextual features were being weighted too heavily.

To investigate the structure of the representations in latent space, I employed clustering. I observed that BYOL-based clusters were more semantically consistent, whereas SimCLR clusters tended to have more outliers, which didn't follow the general semantic meaning of the cluster. When clustering GeoSimCLR, I discovered that the learnt contextual representations seemed to primarily encode stylistic attributes. This meant that similarity scores were biased towards patches coming from the same map type. Nonetheless, I showed that by more heavily weighting the visual representations, I obtained similarity scores which better aligned with human judgement.

I also used clustering to semantically segment individual maps. Generally, segmentations weren't perfect, with different clusters overlapping. However, the clusters do seem to encode some broad-sense awareness of semantics, particularly when clustering BYOL representations directly in latent space.

## 5.2 Future Work

There are several areas of future work that could improve upon the results of this project. First, in terms of data processing, exploring the use of overlapping patches could provide the model with more information and feature combinations, leading to more robust representations. Secondly, extending the project to include a wider variety of maps with distinct styles, such as Bartholomew maps, would increase the usability of these contrastive models, particularly given how challenging aligning maps with such different styles is. Lastly, a more fine-grained hyperparameter tuning could be employed to shed light on why CNN encoders work best with SimCLR for this purpose, and why certain BYOL settings lead to representation collapse. One could also evaluate a ResNet-based SimCLR model on temporal/information awareness tasks, to see whether this improves the semantic awareness of the model, in contrast to when a CNN is used.

When evaluating the contrastive representations, I have avoided a direct comparison between the two models (which is critical given their distributional difference), which motivated the development of the PPIT showcased in subsection 4.2.2, a novel, unsupervised way of evaluating different contrastive models. Nonetheless, these tasks are just a proxy for selecting "good enough" models and, for example, don't take into account the quality of negative pairs found by the contrastive models (for example, given a reference patch containing a building in a certain style/orientation, we should expect a better model to find negative pairs which share these features, but currently this information isn't taken into account). The dowsntream classification tasks presented in subsection 4.3.1 provided a simple way of analysing the difference in representations learnt by the two models, but these could be further refined.

For example, through hand-labelling, an object classifier such as YOLO [40] can be trained, to detect features such as trees, buildings or text. A classifier can then be trained to determine which objects are present in a patch, based solely on the contrastive representation, which would provide more concrete evidence on how these models encode visual information. Furthermore, the contrastive representations could be used to generate a canonical map patch in a given style; this could reveal which features are encoded by contrastive representations, and would provide a framework for homogenising different maps of the same region (as they'd all be converted into the "same" map). I tried this using a UNet [41] to convert OS map representations into both OSM and OS style maps, and found that simple shapes (such as roads) could be successfully extracted from the representations (including a relatively correct orientation and positioning). However, the model struggled with more complex features (such as building shapes), so perhaps more complex systems (such as diffusion models [42]) could be used.

I was able to exploit the nature of our representations, in order to derive somewhat meaningful conclusions from the clustering results. However, given the large number of parameters that both UMAP and HDBSCAN allow, the results could be further refined to reveal more information. This would be particularly relevant when using clustering for map segmentation, whereby the segmentations weren't always consistent, and they were only meaningful for maps with many different sets of features. Nonetheless, this could be an interesting route to further explore, particularly since segmenting these historical maps in a completely unsupervised manner can be quite useful (for example, exploring city development through time).

# Bibliography

- [1] Kasra Hosseini et al. *MapReader: A Computer Vision Pipeline for the Semantic Exploration of Maps at Scale*. 2021. DOI: 10.48550/ARXIV.2111.15592. URL: https://arxiv.org/abs/2111.15592.
- [2] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: CoRR abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/ 1512.03385.
- [3] Zekun Li. "Generating Historical Maps from Online Maps". In: Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. SIGSPATIAL '19. Chicago, IL, USA: Association for Computing Machinery, 2019, pp. 610–611. ISBN: 9781450369091. DOI: 10. 1145/3347146.3363463. URL: https://doi.org/10.1145/3347146. 3363463.
- [4] Jun-Yan Zhu et al. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. 2017. DOI: 10.48550/ARXIV.1703.10593. URL: https: //arxiv.org/abs/1703.10593.
- [5] OpenStreetMap contributors. *Planet dump retrieved from https://planet.osm.org*. 2017. URL: https://www.openstreetmap.org.
- [6] Zekun Li et al. "An Automatic Approach for Generating Rich, Linked Geo-Metadata from Historical Map Images". In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '20. Virtual Event, CA, USA: Association for Computing Machinery, 2020, pp. 3290–3298. ISBN: 9781450379984. DOI: 10.1145/3394486.3403381. URL: https://doi.org/10.1145/3394486.3403381.
- [7] Ashish Jaiswal et al. "A Survey on Contrastive Self-Supervised Learning". In: *Technologies* 9.1 (2021). ISSN: 2227-7080. DOI: 10.3390/technologies9010002. URL: https://www.mdpi.com/2227-7080/9/1/2.
- [8] Lanling Xu et al. *Negative Sampling for Contrastive Representation Learning: A Review.* 2022. arXiv: 2206.00212 [cs.IR].
- [9] S. Chopra, R. Hadsell, and Y. LeCun. "Learning a similarity metric discriminatively, with application to face verification". In: 1 (2005), 539–546 vol. 1. DOI: 10.1109/CVPR.2005.202.
- [10] Florian Schroff, Dmitry Kalenichenko, and James Philbin. "FaceNet: A unified embedding for face recognition and clustering". In: (June 2015). DOI: 10.1109/ cvpr.2015.7298682. URL: https://doi.org/10.1109%2Fcvpr.2015. 7298682.

- [11] Kihyuk Sohn. "Improved Deep Metric Learning with Multi-class N-pair Loss Objective". In: 29 (2016). Ed. by D. Lee et al. URL: https://proceedings. neurips.cc/paper/2016/file/6b180037abbebea991d8b1232f8a8ca9-Paper.pdf.
- [12] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. *Representation Learning with Contrastive Predictive Coding*. 2018. DOI: 10.48550/ARXIV.1807.03748.
   URL: https://arxiv.org/abs/1807.03748.
- [13] Ting Chen et al. "A Simple Framework for Contrastive Learning of Visual Representations". In: (2020). DOI: 10.48550/ARXIV.2002.05709. URL: https: //arxiv.org/abs/2002.05709.
- [14] Kaiming He et al. Momentum Contrast for Unsupervised Visual Representation Learning. 2019. DOI: 10.48550/ARXIV.1911.05722. URL: https://arxiv. org/abs/1911.05722.
- [15] Jean-Bastien Grill et al. *Bootstrap your own latent: A new approach to self-supervised Learning*. 2020. arXiv: 2006.07733 [cs.LG].
- [16] Jia Deng et al. *ImageNet: A large-scale hierarchical image database*. 2009. DOI: 10.1109/CVPR.2009.5206848.
- [17] Yuhao Zhang et al. Contrastive Learning of Medical Visual Representations from Paired Images and Text. 2020. DOI: 10.48550/ARXIV.2010.00747. URL: https://arxiv.org/abs/2010.00747.
- [18] Yawen Wu et al. Distributed Contrastive Learning for Medical Image Segmentation. 2022. DOI: 10.48550/ARXIV.2208.03808. URL: https://arxiv.org/ abs/2208.03808.
- [19] Shekoofeh Azizi et al. Big Self-Supervised Models Advance Medical Image Classification. 2021. DOI: 10.48550/ARXIV.2101.05224. URL: https:// arxiv.org/abs/2101.05224.
- [20] Krishna Chaitanya et al. Contrastive learning of global and local features for medical image segmentation with limited annotations. 2020. DOI: 10.48550/ ARXIV.2006.10511. URL: https://arxiv.org/abs/2006.10511.
- [21] Xu Xie et al. "Contrastive Learning for Sequential Recommendation". In: 2022 IEEE 38th International Conference on Data Engineering (ICDE). 2022, pp. 1259– 1273. DOI: 10.1109/ICDE53745.2022.00099.
- [22] Ishan Dave et al. "TCLR: Temporal contrastive learning for video representation". In: Computer Vision and Image Understanding 219 (2022), p. 103406. ISSN: 1077-3142. DOI: https://doi.org/10.1016/j.cviu.2022.103406. URL: https://www.sciencedirect.com/science/article/pii/S1077314222000376.
- [23] Patrick Helber et al. "Introducing Eurosat: A Novel Dataset and Deep Learning Benchmark for Land Use and Land Cover Classification". In: (2018), pp. 204– 207. DOI: 10.1109/IGARSS.2018.8519248.
- Yao-Yi Chiang et al. "Training Deep Learning Models for Geographic Feature Recognition from Historical Maps". In: Using Historical Maps in Scientific Studies: Applications, Challenges, and Best Practices. Cham: Springer International Publishing, 2020, pp. 65–98. ISBN: 978-3-319-66908-3. DOI: 10.1007/978-3-319-66908-3\_4. URL: https://doi.org/10.1007/978-3-319-66908-3\_4.
- [25] Keiron O'Shea and Ryan Nash. An Introduction to Convolutional Neural Networks. 2015. arXiv: 1511.08458 [cs.NE].

- [26] Chitra Agastya et al. Self-supervised Contrastive Learning for Irrigation Detection in Satellite Imagery. 2021. DOI: 10.48550/ARXIV.2108.05484. URL: https://arxiv.org/abs/2108.05484.
- [27] Gencer Sumbul et al. "Bigearthnet: A Large-Scale Benchmark Archive for Remote Sensing Image Understanding". In: *IGARSS 2019 2019 IEEE International Geoscience and Remote Sensing Symposium*. IEEE, July 2019. DOI: 10.1109/igarss.2019.8900532. URL: https://doi.org/10.1109% 2Figarss.2019.8900532.
- [28] Kumar Ayush et al. *Geography-Aware Self-Supervised Learning*. 2020. DOI: 10.48550/ARXIV.2011.09980. URL: 8.
- [29] Gordon Christie et al. *Functional Map of the World*. 2017. DOI: 10.48550/ ARXIV.1711.07846. URL: https://arxiv.org/abs/1711.07846.
- [30] Kihyuk Sohn. Improved Deep Metric Learning with Multi-class N-pair Loss Objective. Ed. by D. Lee et al. 2016. URL: https://proceedings.neurips. cc/paper/2016/file/6b180037abbebea991d8b1232f8a8ca9-Paper.pdf.
- [31] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: Advances in Neural Information Processing Systems 32. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips. cc/paper/9015-pytorch-an-imperative-style-high-performancedeep-learning-library.pdf.
- [32] National Library of Scotland. *Discover the World in Maps*. https://www.nls.uk/collections/maps/.URL: https://www.nls.uk/collections/maps/.
- [33] Ordnance Survey. Ordnance Survey: See a Better place. https://www.nls.uk/ collections/maps/. URL: https://www.ordnancesurvey.co.uk.
- [34] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000). URL: https://opencv.org.
- [35] Weiyuan Wu. *dovahcrow/patchify*. Mar. 2021. URL: https://github.com/ dovahcrow/patchify.py.
- [36] John Canny. "A computational approach to edge detection". In: *IEEE Transactions on pattern analysis and machine intelligence* 6 (1986), pp. 679–698.
- [37] Leland McInnes, John Healy, and Steve Astels. "hdbscan: Hierarchical density based clustering". In: *The Journal of Open Source Software* 2.11 (2017), p. 205.
- [38] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.
- [39] Leland McInnes, John Healy, and James Melville. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. 2018. DOI: 10.48550/ ARXIV.1802.03426. URL: https://arxiv.org/abs/1802.03426.
- [40] Joseph Redmon et al. You Only Look Once: Unified, Real-Time Object Detection. 2016. arXiv: 1506.02640 [cs.CV].
- [41] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV].
- [42] Jonathan Ho, Ajay Jain, and Pieter Abbeel. *Denoising Diffusion Probabilistic Models*. 2020. arXiv: 2006.11239 [cs.LG].
- [43] Sean Gillies et al. *Rasterio: geospatial raster I/O for Python programmers.* https://github.com/mapbox/rasterio. Mapbox, 2013. URL: https://github.com/mapbox/rasterio.

- [44] Alan D. Snow et al. pyproj4/pyproj: 3.0.1rc0. https://github.com/pyproj4/ pyproj. Version 3.0.1rc0. Mar. 2021. DOI: 10.5281/zenodo.4571637. URL: https://doi.org/10.5281/zenodo.4571637.
- [45] Sean Gillies et al. *Shapely: manipulation and analysis of geometric objects.* toblerity.org, 2007. URL: https://github.com/shapely/shapely.

# **Appendix A**

# **Additional Work**

# A.1 Aligning Maps of the Same Region from Georeference Information

A key aspect of this project is being able to automatically find whether 2 maps are meant to represent the same region, as without this, no alignment is possible for positive patch pairs. Thankfully, since I had the georeferenced maps, I could leverage the metadata of the maps to automatically group together maps which corresponded to the same area.

Our first approach to this was to employ the metadata inherent in the original (unprocessed) TIFF files. Amongst other things, this included the coordinate system for the map and a bounding box for the map in those coordinates. Using the package rasterio [43], the bounding box for the map could be extracted. Using pyproj [44], I then converted the bounding box into longitudes and latitudes (the NLS used the British National Grid system for encode coordinates), since these are easier to work with and understand. I thought that using these coordinates, I could then group together the maps which shared the same bounding box. However, upon discovering that no groups were found, I decided to plot the bounding boxes using geojson.io:



Figure A.1: Bounding boxes for 4 maps of the same region. Image generated using geojson.io (see B.1.2).

As one can see, the rotation observed in the maps seemed to affect the bounding boxes stored as metadata. To mitigate this, I could have tried rotating the bounding boxes by the angle used to rectify the original maps. However, this posed unnecessary complications, and wasn't likely to work: longitudes and latitudes correspond to coordinates on a spherical surface, so applying an affine transformation wouldn't lead to a perfect bounding box alignment. Instead, I chose to use the shapely [45] package, which allowed us to convert the bounding boxes into shapes. Critically, this package allows me to compute the intersection area between 2 shapes. I used this intersection area as a signal to determine if 2 maps represented the same region: for any 2 maps, I can compute the intersection area of each of the bounding boxes, then the 2 maps are likely to represent the same region. Using a threshold of 0.8 for the intersection area proportion, I was able to quickly and efficiently group maps together.

Once I had access to the metadata from Table B.1, I was able to confirm that this method found the exact same region alignments, which means that this method could be used in situations where such metadata might not be available.

# A.2 Aligning Maps at the Patch Level

The last step in handling the maps was to, given 2 maps of the same region, be able to generate pairs of patches which are aligned: that is, finding sets of patches which correspond to the same area. Initially, I worked under the assumption that this would be an additional pre-processing step, since the maps were of different shapes, and I were told that during scanning, slight warping could have been introduced into the maps. Our alignment algorithm was tested on the 4 OS maps corresponding to the region portrayed in Figure 3.1. I felt that this was a fairly representative map, since it contained a variety of interesting features, both from the city (buildings, roads, rail tracks) and from the countryside (mountains, lakes, grasslands). For these tests, the 4 OS maps

were split into 2,301  $64 \times 64$  patches, after applying the bilinear interpolator with kernel 4. Because of this, I already had a "rough" alignment.

Once I had a rough alignment, our alignment algorithm worked as follows. Say I have 2 maps: a reference map, and a query map. Our objective is that, for each  $i \in [1, 2301]$ , I can shift each patch  $q_i$  in the query map, such that it has the best possible alignment with its corresponding patch  $r_i$  in the reference map.



Figure A.2: I aim to, given this rough alignment of  $r_i$ ,  $q_i$ , find the best possible way of shifting  $q_i$ , such that it is as aligned with  $r_i$  as possible.

To define the best possible alignment, I considered a maximum number of vertical and horizontal shifts,  $v_{shift}$ ,  $h_{shift}$  (for our experiments, I used  $v_{shift} = h_{shift} = 7$ ). I then padded the query and reference patches, using white pixels. Finally, I applied a binary, adaptive threshold to  $q_i$ ,  $r_i$  (which converts each pixel into either black or white), and then iterated over all possible combinations  $(\Delta y, \Delta x) \in [-v_{shift}, v_{shift}] \times [-h_{shift}, h_{shift}]$ . For each  $(\Delta y, \Delta x)$ , I shifted  $q_i$  vertically by  $\Delta y$  pixels, and horizontally by  $\Delta x$  pixels (call the result  $q_i^{shift}$ ), and then computed a pixel-wise XOR of  $q_i^{shift}$  with  $r_i$ . Summing this then tells us the number of pixels in common between  $q_i^{shift}$  and  $r_i$ , which I use as our alignment measure: the higher this value, the stronger the alignment.



Figure A.3: One way of visualising the misalignment between 2 patches is by using the red channel of the reference, and the blue channel of the query. I can construct then overlap them as a single image. Regions which appear red/blue in the overlapped image correspond to misalignments. As can be seen, these 2 patches barely misalign. In fact, our algorithm assigns optimal alignment by using  $\Delta y = -1$  and  $\Delta x = 0$  (I just need to shift the query patch one pixel down).

I can then apply this procedure over each roughly aligned patch between 2 maps (taking care to instead of including the padding in  $q_i^{shift}$ , to crop out the corresponding map region). I can then visualise the misalignment between 2 maps, by counting the number of times a given shift had to be used to obtain the best possible alignment:



Figure A.4: Heatmap showing the number of times a given alignment shift  $(\Delta y, \Delta x)$  was used in our alignment algorithm, over all patches in 2 maps. The darker a region, the more times a given shift was used. I can see that by far, most patches required no shift at all, with the great majority of patches requiring at most 2 pixel shifts vertically and/or horizontally.

Based on Figure A.2, it seemed reasonable to not use the alignment algorithm: most patches were reasonably aligned, and the minor misalignments found shouldn't impact our contrastive models in any significant way (after all, CL involves applying transformations to positive pairs, in order to learn more robust representations, so I were getting these transformations directly from the data). Just in case, I decided to explore which patches required large shifts in order to align them. I found that (broadly) these were encompassed in 3 categories: largely empty patches (patches have some noise, so darker regions will appear in different places for different empty patches), patches containing words (since words tended to vary in positioning and size across the maps) and patches containing symbols (since both symbol style and placement varied across maps). Moreover, this method has a clear flaw, in that it relies on some degree of stylistic similarity between the patches. Because of this, it wouldn't be reliable if I wanted to align a Bartholomew and OS patch, for instance. Moreover, computing so many alignments over all the maps in our data would be extremely costly. Hence, I thought that it wouldn't be reasonable to continue pursuing the automatic alignment route, since the rough alignments obtained by just applying patchify without overlapping patches were good enough.

## A.3 Aligning OS Maps with Bartholomew Maps

One of the capabilities of rasterio is that it can be used to recover the pixel corresponding to a given geographical coordinate from GeoTIFFs. Because of this, I wanted to see whether it would be possible to extract the region of a Bartholomew map corresponding to the region of some other OS map. Before, with the faulty metadata inherent in the original TIFF files, this would have been pointless. However, now that I had the metadata table, this was a bit more feasible. Naturally, the alignment wouldn't be perfect, since the Bartholomew metadata is also faulty.



Figure A.5: To the left, a Bartholomew map before being pre-processed. After preprocessing, I crop out a region corresponding to the OS map showcased in Figure 3.1. The result is shown to the right. Notice, comparing this with Figure 3.1, they seem very similar, but the cropped out Bartholomew map is slightly larger. This is due to the fact that the Bartholomew map had to be rotated and cropped. But without this, the cutoff would have been rotated, and so, the alignment would be even worse.

Overall, this supports my decision to not work with Bartholomew maps. Firstly, they have a high variability, even amongst themselves (compare Figure 3.1 with Figure A.5). Secondly, as can be seen from Figure A.5, the borders of Bartholomew maps don't form a boundary with the padding. As such, our current pre-processing method doesn't work too well with Bartholomew maps, and a new technique would need to be developed these borders. However, this is highly non-trivial, since the different Bartholomew maps are inconsistent in their bordering (sometimes theres one border, other times there are multiple parallel border lines). Thirdly, aligning them correctly with OS maps, in an automatic and efficient manner would require a lot more effort and ingenuity. One way of doing this could be identifying common landmarks between the two, but again, this would be an arduous task which could constitute a project all by itself.

# A.4 Modifying Representations Through Learning Task

Initially, I tried to enforce geographical awareness by defining a classification task: given some patch p, and another patch r which appears in its context (that is, one of the 8 patches which will be immediately adjacent to p), can I predict the position of r, just from their contrastive representations?



To do this, I learnt 2 simple feedforward models. The first one encoded the latent representations for p and r, which were then concatenated, and passed to thesecond feedforward layer, which used a softmax activation to predict one of 8 classes (top left, top, top right, left, right, bottom left, bottom or bottom right). However, I didn't obtain great results (around 60% validation and test accuracy), even after 100 epochs of training. Moreover, after passing the contrastive embeddings through the network, the resulting representations didn't preserve the similarity features which made our original

representations desirable (for instance, a given patch still obtained a high similarity score with a corresponding positive pair, but its similarity score with other completely different patches also increased).

# **Appendix B**

# Maps and Metadata

# **B.1 GeoJSONs**

#### B.1.1 GeoJSON Containing Full Map Region

```
1
   {
 2
      "type": "FeatureCollection",
 3
      "features": [
 4
       {
         "type": "Feature",
 5
         "geometry": {
 6
 7
           "type": "Polygon",
 8
           "coordinates": [
 9
             [
               [-3.38374788940603,
10
11
                 55.8715983250667
12
               ],
13
               [
14
                 -3.07422210721001,
                 55.8715983250667
15
16
               ],
17
               [
18
                 -3.07422210721001,
19
                 56.0020051346329
20
               ],
21
               ſ
22
                 -3.38374788940603,
23
                 56.0020051346329
24
               ],
25
               Γ
26
                 -3.38374788940603,
27
                 55.8715983250667
28
               1
29
             ]
30
           ]
```

```
31
          },
32
          "properties": {
33
           "stroke": "#ff0000",
34
            "fill" : "none",
35
           "fill-opacity" : 0
36
          }
37
        }
38
      1
39
```

B.1.2 GeoJSON of Overlapping Bounding Boxes

```
1
    {
 2
      "type": "FeatureCollection",
 3
      "features": [
 4
        {
 5
          "type": "Feature",
          "geometry": {
 6
           "type": "GeometryCollection",
 7
 8
            "geometries": [
 9
             {
10
               "type": "Polygon",
               "coordinates": [
11
12
                 [
13
                   [
14
                     -3.192112616595414,
15
                     55.9423625178286
16
                   ],
17
                   ſ
18
                     -3.149118538110902,
19
                     55.94277108265303
20
                   ],
21
                    Γ
22
                     -3.149638014166486,
23
                     55.96030038664902
24
                   ],
25
                   Γ
26
                     -3.192651513330189,
27
                     55.95989155384999
28
                   ],
29
                   [
30
                     -3.192112616595414,
31
                     55.9423625178286
32
33
34
               ]
35
             },
36
             {
```

37	"type": "Polygon",
38	"coordinates": [
39	[
40	[
41	-3.1916248563612144,
42	55.94244979270547
43	],
44	[
45	-3.1486045560362697,
46	55.94285843127807
47	],
48	]
49	-3.149112581883714,
50	55.960009178369596
51	],
52	[
53	-3.1921518949151615,
54	55.959600277564434
55	],
56	[
57	-3.1916248563612144,
58	55.94244979270547
59	]
60	]
61	
61	
62	},
62 63	}, { !
62 63 64	<pre>}, {     "type": "Polygon",     "secondinates", }</pre>
62 63 64 65 66	<pre>}, {     "type": "Polygon",     "coordinates": [</pre>
62 63 64 65 66 67	<pre>}, {     "type": "Polygon",     "coordinates": [     [     [ </pre>
62 63 64 65 66 67 68	<pre>}, {     "type": "Polygon",     "coordinates": [     [</pre>
62 63 64 65 66 67 68 69	<pre>}, {     "type": "Polygon",     "coordinates": [        [         [         -3.1922529047804167,         55.942420425296774</pre>
62 63 64 65 66 67 68 69 70	<pre>}, {     "type": "Polygon",     "coordinates": [     [         [         -3.1922529047804167,         55.942420425296774 }</pre>
61 62 63 64 65 66 67 68 69 70 71	<pre>}, {     "type": "Polygon",     "coordinates": [     [         [         -3.1922529047804167,         55.942420425296774     ],     [ </pre>
62 63 64 65 66 67 68 69 70 71 72	<pre>}, {     "type": "Polygon",     "coordinates": [     [</pre>
62 63 64 65 66 67 68 69 70 71 72 73	<pre>}, {     "type": "Polygon",     "coordinates": [        [</pre>
62 63 64 65 66 67 68 69 70 71 72 73 74	<pre>}, {     "type": "Polygon",     "coordinates": [     [         [         -3.1922529047804167,         55.942420425296774     ],     [         -3.1494971017459723,         55.942826815554305     ],</pre>
62 63 64 65 66 67 68 69 70 71 72 73 74 75	<pre>}, {     "type": "Polygon",     "coordinates": [     [         [         -3.1922529047804167,         55.942420425296774     ],     [         -3.1494971017459723,         55.942826815554305     ],     [</pre>
62 63 64 65 66 67 68 69 70 71 72 73 74 75 76	<pre>}, {     "type": "Polygon",     "coordinates": [     [         [         -3.1922529047804167,         55.942420425296774     ],     [         -3.1494971017459723,         55.942826815554305     ],     [         -3.1499948839664507,</pre>
62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77	<pre>}, {     "type": "Polygon",     "coordinates": [     [         [         -3.1922529047804167,         55.942420425296774     ],     [         -3.1494971017459723,         55.942826815554305     ],     [         -3.1499948839664507,         55.95961890427993</pre>
62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78	<pre>}, {     "type": "Polygon",     "coordinates": [     [         [         -3.1922529047804167,         55.942420425296774     ],     [         -3.1494971017459723,         55.942826815554305     ],     [         -3.1499948839664507,         55.95961890427993     ],</pre>
62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79	<pre>}, {     "type": "Polygon",     "coordinates": [     [         [         -3.1922529047804167,         55.942420425296774     ],     [         -3.1494971017459723,         55.942826815554305     ],     [         -3.1499948839664507,         55.95961890427993     ],     [</pre>
62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80	<pre>}, {     "type": "Polygon",     "coordinates": [     [         [         -3.1922529047804167,         55.942420425296774     ],     [         -3.1494971017459723,         55.942826815554305     ],     [         -3.1499948839664507,         55.95961890427993     ],     [         -3.192769187420956,     ] }</pre>
62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81	<pre>}, {     "type": "Polygon",     "coordinates": [     [         [         -3.1922529047804167,         55.942420425296774     ],     [         -3.1494971017459723,         55.942826815554305     ],     [         -3.1499948839664507,         55.95961890427993     ],     [         -3.192769187420956,         55.959212258688744</pre>
61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82	<pre>}, {     "type": "Polygon",     "coordinates": [     [         [         -3.1922529047804167,         55.942420425296774     ],     [         -3.1494971017459723,         55.942826815554305     ],     [         -3.1499948839664507,         55.95961890427993     ],     [         -3.192769187420956,         55.959212258688744     ],</pre>
61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83	<pre>}, {     "type": "Polygon",     "coordinates": [     [         [         -3.1922529047804167,         55.942420425296774     ],     [         -3.1494971017459723,         55.942826815554305     ],     [         -3.1499948839664507,         55.95961890427993     ],     [         -3.192769187420956,         55.959212258688744     ],     [</pre>
62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84	<pre>}, {     "type": "Polygon",     "coordinates": [     [         [         -3.1922529047804167,         55.942420425296774     ],     [         -3.1494971017459723,         55.942826815554305     ],     [         -3.1499948839664507,         55.95961890427993     ],     [         -3.192769187420956,         55.959212258688744     ],     [         -3.1922529047804167,     ] </pre>



# B.2 Code for Rotating and Cropping the Badly Scanned Maps

This code assumes that the image is horizontal, and that the padding is white. This function was developed by adapting:

- How to rotate skewed fingerprint image to vertical upright position [closed]
- How to remove whitespace from an image in OpenCV?

```
import numpy as np
import cv2 as cv
def rotate_crop_map(map_img):
   gray = cv.cvtColor(map_img, cv.COLOR_RGB2GRAY)
   gray = 255 - gray
   thresh = cv.threshold(gray, 0, 255, cv.THRESH_BINARY + cv.THRESH_OTSU)
       [1]
   # Compute rotated bounding box
   coords = np.column_stack(np.where(thresh > 0))
   center_rect, dims, angle = cv.minAreaRect(coords)
   if angle < -45:
       angle = -angle
   else:
       angle = 90-angle
   print(angle)
   # Rotate image to deskew
   (h, w) = map_img.shape[:2]
   center = (w // 2, h // 2)
   M = cv.getRotationMatrix2D(center, angle, 1.0)
   rotated = cv.warpAffine(map_img, M, (w, h), flags=cv.INTER_CUBIC,
       borderMode=cv.BORDER_REPLICATE)
   gray = cv.cvtColor(rotated, cv.COLOR_BGR2GRAY)
   gray = 255*(gray < 128).astype(np.uint8) # To invert the text to white
   coords = cv.findNonZero(gray) # Find all non-zero points (text)
   x, y, w, h = cv.boundingRect(coords) # Find minimum spanning bounding
      box
   rect = rotated[y:y+h, x:x+w]
   rect = cv.cvtColor(rect, cv.COLOR_BGR2RGB)
```

return rect

# B.3 Metadata Table Columns

Table B.1: Description of the metadata table provided by the NLS. The original table has more columns, but they contained redundant information (i.e same information as other columns). I thus only list the key columns of the metadata table.

Column Name	Type of Data
WKT	Bounding boxes in Well-Known Text for- mat. Coordinates are stored as longitudes and latitudes.
SERIES	The type of map. For example: Scotland - 25 Inch 2nd and Later Editions.
COUNTY	The country represented in the map. All these maps corresponded to Edinburghshire.
SHEET_NO	The sheet number for the map. This helps identify the geographical area covered by the map. These are of the form 007_05, and there were 57 such unique numbers.
EDITION	The edition of the map. All the maps were a second, third, fourth or fifth edition.
IMAGEURL	A URL to visualise the map in the NLS website. For instance: https://maps.nls.uk/view/82877892.
YEAR	The year in which the map was published. As discussed, these range from 1894 to 1947.

# **B.4 Code for Removing Uninformative Patches**

The method for removing patches with uninformative features was part of the CLPatchDataset class, which is the class I used to store the patch dataset for the contrastive algorithms.

```
import numpy as np
import cv2 as cv
from torch.utils.data import Dataset
class CLPatchDataset(Dataset):
    ...
    @staticmethod
    def count_black_pixels(edges):
        black_pixel_idx = np.where(edges == 255)
        return len(black_pixel_idx[0])
```

return False

```
@staticmethod
def get_edges(patch):
   gray = cv.cvtColor(np.array(patch), cv.COLOR_RGB2GRAY)
   gray = cv.GaussianBlur(gray, (3, 3), 0)
   return cv.Canny(gray, 50, 150)
@staticmethod
def is_empty_patch(patch):
   edges = CLPatchDataset.get_edges(patch)
   patch_width = patch.shape[0]
   n_black_pixels = CLPatchDataset.count_black_pixels(edges)
   min_black_pixels = int(0.01 * patch_width * patch_width)
   edge_pixel_range = int(0.1 * patch_width)
   max_edge_pixels = int(0.5 * n_black_pixels)
   if n_black_pixels <= min_black_pixels:</pre>
      return True
   else:
       for row in range(edge_pixel_range):
          if CLPatchDataset.count_black_pixels(edges[row, :]) >=
              max_edge_pixels \
                 or CLPatchDataset.count_black_pixels(edges[len(edges
                     ) - 1 - row, :]) >= max_edge_pixels \
                 or CLPatchDataset.count_black_pixels(edges[:, row])
                     >= max_edge_pixels \
                 or CLPatchDataset.count_black_pixels(edges[:, len(
                     edges) - 1 - row]) >= max_edge_pixels:
              return True
```

# Appendix C

# **Experimental Results**

## C.1 Estimate for Runtime

We used the set of hypterparameter choices which we used for running our experiments:

- encoder ∈ {ResNet18, ResNet34}
- pretrain  $\in$  {*True*, *False*}
- $\tau \in \{0.80, 0.90, 0.95, 0.99\}$
- $\eta \in \{1 \times 10^{-3}, 1 \times 10^{-2}\}$
- batch size  $\in \{32, 64\}$
- patch size  $\in \{128, 224\}$

Based on preliminary experiments, we obtained a lenient estimate of 12 hours to run 5 epochs (including evaluating the model 100 times per epoch on the validation set), this would amount to around 64 days of training, for *each* of the contrastive models. Typically, BYOL took much longer to train than SimCLR, and the estimate of around 12 hours is the approximate runtime we observed for SimCLR experiments.

## C.2 All Experiments Run

Note, for the experiments using  $224 \times 224$  patches, our dataset changed. In particular, I obtained 24,093 training positive pairs, 2,938 validation positive pairs and 2,886 testing positive pairs. Notice, due to the patch size, certain maps weren't included in this dataset, since upon splitting them into corresponding patches, some regions contained maps with a different number of patches. Said regions weren't included in the final dataset.

Architecture	Encoder	Pretrained	τ	Learning Rate	Patch Size	Batch Size
BYOL	ResNet18	1	0.99	$1 \times 10^{-3}$	$128 \times 128$	64
BYOL	ResNet18	1	0.99	$1 \times 10^{-2}$	$128 \times 128$	64
BYOL	ResNet18	1	0.99	$1 \times 10^{-3}$	$128 \times 128$	32
BYOL	ResNet34	1	0.99	$1 \times 10^{-3}$	128  imes 128	64
BYOL	ResNet18	×	0.99	$1 \times 10^{-3}$	$128 \times 128$	64
BYOL	ResNet18	1	0.99	$1 \times 10^{-3}$	$224 \times 224$	64
BYOL	CNN	×	0.99	$1 \times 10^{-3}$	$128 \times 128$	64
BYOL	ResNet18	1	0.95	$1 \times 10^{-3}$	128  imes 128	64
BYOL	ResNet18	1	0.90	$1 \times 10^{-3}$	128  imes 128	64
BYOL	ResNet18	1	0.80	$1 \times 10^{-3}$	$128 \times 128$	64
SimCLR	ResNet18	1	0.99	$1 \times 10^{-3}$	$128 \times 128$	64
SimCLR	ResNet18	1	0.99	$1 \times 10^{-2}$	$128 \times 128$	64
SimCLR	ResNet18	1	0.99	$1 \times 10^{-3}$	$128 \times 128$	32
SimCLR	ResNet34	1	0.99	$1 \times 10^{-3}$	$128 \times 128$	64
SimCLR	ResNet18	×	0.99	$1 \times 10^{-3}$	$128 \times 128$	64
SimCLR	ResNet18	1	0.99	$1 \times 10^{-3}$	$224 \times 224$	64
SimCLR	CNN	×	0.99	$1 \times 10^{-3}$	128  imes 128	64
SimCLR	ResNet18	1	0.95	$1 \times 10^{-3}$	$128 \times 128$	64
SimCLR	ResNet18	1	0.90	$1 \times 10^{-3}$	$128 \times 128$	64
SimCLR	ResNet18	1	0.80	$1 \times 10^{-3}$	$128 \times 128$	64

Table C.1: Set of all experiments run.

# C.3 Losses for Experiments

## C.3.1 BYOL

Table C.2: BYOL training and validation loss during experimental runs.

Model	Total Training Time	Full Epochs Completed	Best Training Loss	Best Validation Loss
BYOL (Base)	23.055	5	0.042	0.227
BYOL ( $\tau = 0.95$ )	26.673	5	0.039	0.209
BYOL ( $\tau = 0.90$ )	18.157	4	0.056	0.206
BYOL ( $\tau = 0.80$ )	18.473	4	0.057	0.214
BYOL (Not Pretrained)	23.061	5	0.059	0.252
BYOL (ResNet 34)	22.479	3	0.084	0.237
BYOL (CNN)	8.169	5	0.070	0.287
BYOL (Patch Size 224)	7.409	5	0.034	0.240
BYOL (Batch Size 32)	36.925	8	0.001	0.028
BYOL $(\eta=1\times 10^{-2})$	9.325	2	0.079	0.154

## C.3.2 SimCLR

Table C.3: SimCLR training and validation loss during experimental runs. Notice, we expect a lower loss for models using  $\tau < 0.99$ , since a smaller  $\tau$  will lead to a smaller loss, everything else equal.

Model	Total Training Time (Hours)	Full Epochs Completed	Best Training Loss	Best Validation Loss
SimCLR (Base)	14.910	5	3.871	4.143
SimCLR ( $\tau = 0.95$ )	15.012	5	3.828	4.115
SimCLR ( $\tau = 0.90$ )	13.382	5	3.776	4.081
SimCLR ( $\tau = 0.80$ )	13.237	5	3.647	3.995
SimCLR (Not Pretrained)	14.509	5	3.877	4.150
SimCLR (ResNet34)	12.938	3	3.882	4.148
SimCLR (CNN)	5.260	5	3.877	4.159
SimCLR (Patch Size 224)	4.776	5	3.066	4.089
SimCLR (Batch Size 32)	13.873	5	3.164	3.541
$\label{eq:simCLR} \text{SimCLR} \; (\eta = 1 \times 10^{-2})$	13.489	5	3.884	4.139

# C.4 Results for Experiments Run Longer

## C.4.1 Training Results

Table C.4: SimCLR training and validation loss during experimental runs, when run for 15 epochs without early stopping.

Model Total Training Time (Hours)		Full Epochs Completed	Best Training Loss	Best Validation Loss	
SimCLR ( $\tau = 0.80$ )	39.97	15	3.617	3.974	
SimCLR (Not Pretrained)	40.42	15	3.855	4.135	

## C.4.2 PPIT Results

Table C.5: Comparing the best SimCLR model from the original runs with the second best models after 15 epochs of training (instead of early stopping).

	Model	Top-1 Accuracy	Top-5 Accuracy	Top-10 Accuracy	Positive Pair Accuracy
	SimCLR ( $ au=0.80$ )	0.22547	0.75158	0.80010	0.73515
Early Stopping	SimCLR (Not Pretrained)	0.22615	0.75790	0.79922	0.74210
	SimCLR (CNN)	0.22654	0.76305	0.80272	0.74905
15 Enochs	SimCLR ( $ au=0.80$ )	0.22372	0.75372	0.79951	0.73787
15 Epochs	SimCLR (Not Pretrained)	0.22761	0.75897	0.80165	0.74336

# C.5 Deriving Automatic Thresholds for Information Awareness

```
import numpy as np
def get_class_thresholds(pixel_counts):
   thresholds = []
   prev_min = 0
   for partition in pixel_counts:
       thresh = np.max(partition)
       if thresh > prev_min:
          thresholds.append(thresh)
       else:
          thresholds.append(np.min(partition[partition > thresh]))
   return thresholds
# pixel_counts is a list
# pixel_counts[i] contains the number of edge pixels
# in the ith patch from a dataset
# split pixel_counts after sorting
split_pixel_counts = np.array_split(np.sort(pixel_counts), n_classes)
# get the thresholds for the different classes
class_thresholds = get_class_thresholds(split_pixel_counts)
# generate the class labels for the task
# digitize: "Return the indices of the bins
# to which each value in input array belongs."
# the input array is pixel_counts, whilst the bins are class_thresholds
labels = np.digitize(pixel_counts, class_thresholds, right=True)
```

## C.6 Transformations Applied

#### C.6.1 Rotation (30°)

The use of a 30° rotation angle was arbitrary, and was simply chosen since we thought it would significanly distort the meaning of a patch.

import torchvision.transforms as T

#### C.6.2 Horizontal Flip

```
H_FLIP_PROBABILITY = 1
HORIZONTAL_FLIP_TRANSFORM = T.RandomHorizontalFlip(p = H_FLIP_PROBABILITY)
```

#### C.6.3 Random Rotation (90°, 180°, 270°)

The set of rotation angles were selected in agreement with what was used in the original SimCLR paper.

```
import torchvision.transforms as T
import numpy as np
ROTATION_RANGE = np.arange(90,360 + 90,90)
def ROTATION_RANDOM_TRANSFORM(img):
    angle = np.random.choice(ROTATION_RANGE)
    return T.RandomRotation(degrees = [angle, angle], fill = 1)(img)
```

#### C.6.4 Random Crop ( $64 \times 64$ )

We chose a random crop of size  $64 \times 64$ , whilst in the original papers, they apply a random crop size (ranging from 0.08 and 1.00 in terms of proportion).

#### C.6.5 Centre Crop ( $64 \times 64$ )

T.Resize(IMG\_WIDTH)

#### C.6.6 Gaussian Blur

In the SimCLR paper, they randomly sample  $\sigma \in [0.1, 2.0]$ , and the kernel size is set to 10% of the image width/height. In our case, we picked a fixed  $\sigma = 1$ ; since we had  $128 \times 128$  patches, we used a kernel sie of  $13 \times 13$ .

])

#### C.6.7 Colour Jittering

The colour jittering implementation is in accordance with the SimCLR, where they define a jitter strength. The choice of 0.5 as a jitter strength was arbitrary.

#### C.6.8 Grayscaling

```
import torchvision.transforms as T
GRAYSCALE_PROBABILITY = 1
GRAYSCALE_TRANSFORM = T.RandomGrayscale(p = GRAYSCALE_PROBABILITY)
```

# C.7 Clustering Results

#### C.7.1 Settings for UMAP

```
from umap import UMAP
UMAP(verbose = True,
    n_jobs = -1,
    metric = "cosine",
    n_neighbors = 23,
    n_components = 100 #2,3,5,10,15,
    densmap = False,
    random_state = 23,
    negative_sample_rate = 10,
    low_memory = False,
    min_dist = 0)
```

#### C.7.2 Settings for HDBSCAN

```
from hdbscan import HDBSCAN
HDBSCAN(min_cluster_size = 23,
    min_samples = 13,
    metric = "minkowski",
    core_dist_n_jobs = -1,
    prediction_data = True,
    cluster_selection_epsilon = 0,
    cluster_selection_method = "eom",
    p = 2)
```

#### C.7.3 Clustering on Validation Data

Table C.6: Number of clusters and percentage of outliers for different dimensionality reduction techniques and contrastive models.

Dimensionality Reduction	Features	Model	Clusters	Outliers	Percentage of Outliers
UMAP	15	Best SimCLR	40	516	9.32%
UMAP	15	Best BYOL	52	497	8.97%
UMAP	15	GeoSimCLR	45	701	12.66%
UMAP	100	Best SimCLR	48	693	12.51%
UMAP	100	Best BYOL	62	732	13.22%
UMAP	100	GeoSimCLR	46	756	13.65%
PCA	15	Best SimCLR	5	2,596	46.88%
PCA	15	Best BYOL	17	4,190	75.66%
PCA	15	GeoSimCLR	7	3,194	57.67%
PCA	100	Best SimCLR	5	2,889	52.17%
PCA	100	Best BYOL	19	4,324	78.01%
PCA	100	GeoSimCLR	5	2,453	44.29%

## C.7.4 Cluster Similarities Using 100 Features

		Cluster Similarity			Model Similarity Percentile		
Dimension Reducer	Model	Mean	Median	Outlier	95th	99th	99.9th
UMAP 100	Rest SimCI D	0.63649	0.62903	0.24381	0.50193	0.69975	0.90003
PCA 100	Dest Stillelik	0.67573	0.66517	0.23644			
UMAP 100	Best BYOI	0.90576	0.91141	0.70625	0.87720	0.95171	0.99755
PCA 100	Dest BIOL	0.94897	0.96135	0.66050	0.07729		
UMAP 100	CoosimCLP	0.71886	0.69242	0.35816	0.72689	0.85462	0.03467
PCA 100	Geostment	0.60515	0.62288	0.26928			0.93407
# C.7.5 Cluster Examples

### C.7.5.1 BYOL Clusters



Figure C.1: UMAP (15 features) on BYOL representations.



Figure C.2: UMAP (15 features) on BYOL representations.



Figure C.3: PCA (15 features) on  $\ensuremath{\texttt{BYOL}}$  representations.



Figure C.4: PCA (15 features) on BYOL representations.



Figure C.5: PCA (15 features) on BYOL representations.

#### C.7.5.2 SimCLR Clusters



Figure C.6: UMAP (15 features) on SimCLR representations.



Figure C.7: UMAP (15 features) on SimCLR representations.



Figure C.8: PCA (15 features) on SimCLR representations.



Figure C.9: PCA (15 features) on SimCLR representations.

### C.7.5.3 GeoSimCLR Clusters



Figure C.10: UMAP (15 features) on GeoSimCLR representations.



Figure C.11: UMAP (15 features) on GeoSimCLR representations.



Figure C.12: UMAP (15 features) on GeoSimCLR representations.

## C.7.6 Score Distributions for All Clusters

#### C.7.6.1 Cluster Similarity Statistics

To do this, for each cluster, I compute the average similarity amongst patches in the cluster, and then consider the average and median of these values across all clusters. This then gives us a numerical measure of how good the clusters are at encompassing similar patches. To see whether the similarity is due to the aptness of the clusters, I have compared these similarities with simple baselines, such as an *n*th percentile similarity score for a given contrastive method, or the similarity score for the outlier cluster. In

particular, for each model, I can compute the set of all unique similarity scores between each patch  $P_i \in P, Q_j \in Q$ ; using these I can then easily compute *n*th percentiles. I chose the 95th, 99th and 99.9th percentiles, since I found that all of our data fit in between these bins. I have also visualised the distribution of scores within a cluster, and compare these with the distributions showcased in Figure 4.3.

Table C.7: Cluster similarity results, after reducing contrastive representations to 15 dimensions, and clustering with HDBSCAN.

Dim. Reducer	Model	Cluster Similarity			Model Similarity Percentile		
		Mean	Median	Outlier	95th	99th	99.9th
UMAP 15	- Best SimCLR	0.63246	0.64370	0.28298	0.50193	0.69975	0.90003
PCA 15		0.62638	0.73072	0.24650			
UMAP 15	Best BYOL	0.89492	0.90474	0.69894	0.87729	0.95171	0.99755
PCA 15		0.94131	0.95836	0.66047			
UMAP 15	GeoSimCLR	0.72217	0.71604	0.35030	0.72689	0.85462	0.93467
PCA 15		0.64038	0.67589	0.27531			

The above results show that the clusters are, in general, encompassing the notion of similar patches (at least from the point of view of the contrastive model), with the mean cluster similarity always falling well above the 95th percentile of similarity scores, and the median cluster similarity sometimes going above the 99th percentile.

#### C.7.6.2 Cluster Distributions (100 features)

We can compute the similarity score for each cluster, for each model. We then plot these scores as a histogram, where the y-axis corresponds to the proportion (that is, what proportion of all similarity scores does a bin correspond to), whilst the x-axis corresponds to the similarity scores of the cluster.



## C.7.7 Clusters for Regions from GeoSimCLR Representations



(a) Clusters for region 8, using UMAP and K-Means with K = 4.



(b) Clusters for region 8, using PCA and HDBSCAN.



(c) Clusters for region 8, using PCA and K-Means with K = 4.

Figure C.13: Clusters for region 8, using different dimensionality reduction and clustering techniques. From left to right, I cluster the full geo-contrastive representation, the visual representation and the contextual representation.



(a) Clusters for region 17, using UMAP and K-Means with K = 4.



(b) Clusters for region 17, using PCA and HDBSCAN.



(c) Clusters for region 17, using PCA and K-Means with K = 4.

Figure C.14: Clusters for region 17, using different dimensionality reduction and clustering techniques. From left to right, I cluster the full geo-contrastive representation, the visual representation and the contextual representation.



(a) Clusters for region 52, using UMAP and K-Means with K = 3.



(b) Clusters for region 52, using PCA and HDBSCAN.



(c) Clusters for region 52, using PCA and K-Means with K = 3.

Figure C.15: Clusters for region 52, using different dimensionality reduction and clustering techniques. From left to right, I cluster the full geo-contrastive representation, the visual representation and the contextual representation.

# C.7.8 Similarity Score Matrices for Region Patches from GeoSimCLR Representations



(a) Similarity scores for patches in region 8.



(b) Similarity scores for patches in region 17.



(c) Similarity scores for patches in region 52.

Figure C.16: Similarity score matrices for different regions (using the same patches as positive pairs of each other). From left to right, I use the full geo-contrastive representation, the visual representation and the contextual representation.

# C.7.9 Map Segmentation Through Clustering

#### C.7.9.1 HDBSCAN



(a) Segmentation for 82877409.tiff.







(b) Segmentation for 82877412.tiff.







(c) Segmentation for 82877415.tiff.



(d) Segmentation for 82877418.tiff.

Figure C.17: Maps were segmented by using HDBSCAN directly on the contrastive representations. From left to right, the representations used were the best SimCLR, the best BYOL and GeoSimCLR.

#### C.7.9.2 UMAP + HDBSCAN



(a) Segmentation for 82877409.tiff.



(b) Segmentation for 82877412.tiff.



(c) Segmentation for 82877415.tiff.



(d) Segmentation for 82877418.tiff.

Figure C.18: Maps were segmented by using UMAP for dimensionality reduction of the contrastive representations, followed by HDBSCAN for clustering. From left to right, the representations used were the best SimCLR, the best BYOL and GeoSimCLR.

#### C.7.9.3 UMAP + KMeans



(a) Segmentation for 82877409.tiff.



(b) Segmentation for 82877412.tiff.



(c) Segmentation for 82877415.tiff.



(d) Segmentation for 82877418.tiff.

Figure C.19: Maps were segmented by using UMAP for dimensionality reduction of the contrastive representations, followed by KMeans for clustering, with K = 4. From left to right, the representations used were the best SimCLR, the best BYOL and GeoSimCLR.

# **Appendix D**

# **Understanding Similarity Distributions**

# **D.1 Similarity Score Distributions**

D.1.1 BYOL

D.1.1.1 BYOL (Base)





**D.1.1.2** BYOL ( $\tau = 0.95$ )



**D.1.1.3** BYOL ( $\tau = 0.80$ )







D.1.1.5 BYOL (ResNet34)



## D.1.1.6 BYOL (CNN)



D.1.1.7 BYOL (Patch Size 224)



D.1.1.8 BYOL (Batch Size 32)



D.1.1.9 byol ( $\eta=1\times 10^{-2}$ )



- D.1.2 SimCLR
- D.1.2.1 SimCLR (Base)



D.1.2.2 SimCLR ( $\tau = 0.95$ )





D.1.2.3 SimCLR ( $\tau = 0.90$ )



D.1.2.4 SimCLR ( $\tau = 0.80$ )



D.1.2.5 SimCLR (Not Pretrained)



D.1.2.6 SimCLR (ResNet34)







D.1.2.8 SimCLR (Batch Size 32)



D.1.2.9 SimCLR ( $\eta=1\times 10^{-2}$ )



# D.1.3 Weighted GeoSimCLR





D.1.3.2 Visual Weight = 0.9



# **D.2 Most Similar Patches**

# D.2.1 SimCLR and BYOL







Figure D.2: This example showcases how our positive pair creation process could be refined, since the second positive pair seems to have very little to do with the reference patch (this doesn't mean they don't represent the same region, this is likely due to the year differences). Nonetheless, both SimCLR and BYOL are capable of identifying suitable positive pairs. SimCLR is capable of correctly identifying the first positive pair, and with much higher similarity score than the other most similar patches, which all score very similarly. On the other, the more semantic representativity of BYOL means that it just finds patches containing a line with the correct positioning and orientation, and all such lines obtain a nearly identical similarity score.



Figure D.3: Again, both models correctly identify the true positive pairs correctly, and there is a drastic decrease in similarity score for the remaining patches. Again, SimCLR finds patches which aren't too related to the reference (except perhaps for the locationa nd direction of a main road). On the other hand, BYOL seems to have been able to identify the notion of a cross roads, alongside the dotted line patterns which are present in the positive pairs.



Figure D.4: This exemplifies the impressive ability that these models have of understanding the information contained within the patches. Given that these are validation patches, the model has never seen any such patches. If I look at the reference and the positive pairs, I struggled to see how these could constitute positive pairs, particularly given the reference. The only thing that they seem to have in common is a stripe pattern, alongside a general direction. Both models seems to be able to pick up on this, particularly SimCLR, which impressively is capable of finding a correct positive pair. BYOL seems to struggle a lot more with this task, as it seems to identify a mixture of rail tracks and plain roads, both features present within the reference patch.



Figure D.5: In this example, it is quite noticeable the effect that the band of black and yellow has on decreasing the similarity score for the positive patch pairs. Moreover, it is quite significant to see how beyond the positive pairs, SimCLR seems to ignore the presence of the rail track, instead focusing on the presence of buildings with a road in between. On the other hand, BYOL does pick up on both the railtracks, but also the presence of lateral roads at either side of the tracks.



Figure D.6: These examples showcases 2 key aspects. Firstly, the positional awareness of the representations: both models understand that the text is situated in the upper right corner of the patch. Secondly, BYOL is a lot more optimistic, assigning a similarity score above 0.999 for all patches, whilst SimCLR seems to be a lot more conservative, perhaps in understanding the difference between numbers and text.









Figure D.7: Given the specificity and uniqueness of the reference patch, we'd expect taht the true positive pairs should be fairly easy to find. What is most interesting is that for both models, the third most similar patch conserves information and shape relative to the reference (the fact that there are trees, and the general shapes that appear in the patches are fairly similar). However, the fourther and fifth most similar patches seem to be fairly unrelated, particularly for SiMCLR (with BYOL at least there is a notion of trees around black lines).



Figure D.8: Here I showcase how BYOL seems to better understand the combination of features present within the reference (empty space + sparse trees in an enclosure). Once again, both models find the correct positive paris, and there is a marked difference between these and the other patches found.



# D.2.2 GeoSimCLR

#### D.2.2.1 Weight = 0.7





#### D.2.2.2 Weight = 0.9










103

