

SELF-SUPERVISED LEARNING

LAND CLASSIFICATION OF SATELLITE IMAGERY

ROBERT SKOGLUND

Bachelor's thesis
2022:K19



LUND UNIVERSITY

Faculty of Science
Centre for Mathematical Sciences
Mathematical Statistics

Bachelor's Theses in Mathematical Sciences 2022:K19
ISSN 1654-6229
LUNFMS-4067-2022
Mathematical Statistics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden
<http://www.maths.lu.se/>

Abstract

The rise of self-supervised learning has granted a deeper level of generalized machine learning, capable of learning semantic representations without any use of labelling. With 700 satellites orbiting Earth and generating terabytes of unlabelled data daily, satellite imagery serves as a particularly enticing data set for self-supervised learning, containing rich information with many applicable domains, such as agriculture. Could one, for example, train a model to predict harvest yields on farmland based on representations learnt with self-supervised learning?

To explore the capacity of self-supervised learning in the context of agriculture and remote sensing, i.e. studying phenomena from a distance using technology such as drone and satellites, we attempt to build a simple binary classifier classifying satellite images as *farmland* or *not-farmland* using the self-supervised visual representation learning algorithm *SimCLR* developed by *Google*. The dataset used was the *BigEarthNet-S2* dataset captured by the *Copernicus Sentinel-2* satellites. For comparison, a multi-class classifier was trained using an identical procedure on the visual object recognition dataset *TinyImageNet*. Self-supervised training of the network and supervised training of linear classifier were performed simultaneously, as the *SimCLR* authors report that this achieves similar performance as sequential self-supervised and supervised training.

TinyImageNet training metrics revealed successful self-supervised learning, however it was evident that the model would benefit from longer training and further experimentation with hyperparameters. The highest top-1 accuracy achieved was 41.66%. As for *BigEarthNet-S2*: after training the linear classifiers, the evaluation metrics revealed poor predictive accuracy and generalization capacity, obtaining sensitivities of approximately 60%. The poor evaluation metrics were however not attributed to poor training or choice of hyperparameters, but rather to the poor pairing of the classification task and dataset. Namely, the multi-labelled *BigEarthNet-S2* dataset contained too semantically diverse information with regards to the binary classification problem. This problem is intrinsic to the nature of multi-labelled data, containing overlapping classes and an arbitrary level of relevance for each label. After the analysis, improvements and methodological changes are proposed, such as utilizing a dataset with semantically distinct classes or fine-tuning with another niched dataset for a specialized downstream task.

Acknowledgements

I would like to express my sincere gratitude to Alexandros Sopasakis for his unending support and supervision during my thesis, as well as introducing me to an exciting domain of research I had not yet discovered.

I would also like to thank my beloved family for their unwavering support and motivation throughout my studies.

Contents

1	Introduction	1
1.1	Aim	2
1.2	Context	2
2	Background & Theory	3
2.1	Machine Learning	3
2.1.1	Self-Supervised Learning	3
2.2	Artificial Neural Networks	4
2.2.1	Linear regression	4
2.2.2	Perceptron	5
2.2.3	Multilayer Perceptron	6
2.2.4	Classification	8
2.2.5	Convolutional Neural Networks	10
2.2.6	Residual Networks	12
2.3	Optimization	14
2.3.1	Gradient Descent	14
2.3.2	Stochastic Gradient Descent	14
2.3.3	Minibatch Stochastic Gradient Descent	15
2.3.4	Momentum	15
2.3.5	Adaptive Learning Rate	15
2.3.6	Layer-wise Adaptive Rate Scaling	16
2.4	Image Processing	16
2.5	SimCLR	18
3	Method	21
3.1	Data	21
3.2	Evaluation	22
3.2.1	Data partitioning	23
3.3	Model & Training	23
4	Results	24
4.1	TinyImageNet Results	25
4.2	BigEarthNet Results	29

5	Discussion	31
5.1	Results Discussion	31
5.2	Conclusion and Further Research	33
A	Appendix	35
A.1	TinyImageNet: Further Training	35
A.1.1	Batch size $N = 256$	35
A.1.2	Batch size $N = 1024$	37
A.1.3	Tabular Summary	38

Chapter 1

Introduction

Machine learning has proved its ability to solve many high-level problems, such as distinguishing between cats and dogs or learning how to drive a car. Humans are still however superior in one sense: we seem to learn faster. For example, children can distinguish between cats and dogs within a few instances of exposure, while machines need thousands of examples to complete the same task. Humans can also learn to drive a car in under 50 hours but autonomous cars need thousands of hours of training data in order to drive. This comparison however fails to highlight the innate components of human intelligence: common sense. It is unfair to assume that humans do not have previously acquired knowledge, either innate or experienced, that contributes to our ability to learn quickly. For this reason, the problem of forming generalised intelligence or common sense is significant within artificial intelligence research.

The demonstrated success of machine learning is particularly high in specialized tasks with large amounts of labelled data. Learning through such a supervised approach has an inherent bottleneck: it requires too much labelling. Labelling demands many resources, be it time, expertise or money, and is very inefficient. By developing some sort of common sense in machines, could machines learn with as few examples and as quickly as humans?

Self-supervised learning is considered one of the most promising areas to develop such a common sense in machines. The defining principle in self-supervised learning is that it constructs a supervised learning task from unlabelled data, where the supervised task is often chosen to leverage the underlying data structure of the data. To evaluate the ability of the self-supervised system, one has to define a supervised task, known as the downstream task.

The field of remote sensing, studying objects and phenomena from a distance often using satellite imagery, generates a massive amount of data. Over 700 satellites currently orbit Earth and generate terabytes of data daily [8]. Since labelled satellite imagery is limited, one can utilize self-supervised learning to leverage the massive amounts of unlabelled data to develop intelligent systems. Such systems could solve interesting problems within remote sensing, and in particular: agriculture.

With the rise of Internet of Things (IoT) and autonomous vehicles, the agricultural sector has become exceedingly more automated. For example, soil nutrition/moisture is measured in fields with sensor technology, tractors are equipped with self-driving and intelligent irrigation systems, etc. By further utilizing satellite imagery, measuring a wide range of the electromagnetic spectrum, farmers can obtain valuable insights of their fields, such as potential yields or harvest.

1.1 Aim

The aim of this thesis is to investigate the performance of self-supervised learning in the context of agriculture and remote sensing. Specifically, the performance of the *SimCLR* self-supervised learning algorithm by Google will be tested on satellite imagery, namely the *BigEarthNet-S2* dataset captured by the *Copernicus Sentinel-2* satellites, and compared to the performance when trained on the visual object recognition dataset *TinyImageNet*.

1.2 Context

The thesis is a small contribution to the agricultural AI project *AI i klimatets tjänst*, a collaboration between Lund University and companies *Hushållningssällskapet*, *T-kartor* and *Sensitive*. The hope is that by using feature representations of the satellite imagery, processed through the trained self-supervised network, one can predict crop yields. With accurate yield predictions, farmers are able to optimize the business operations such as pricing and storing.

Chapter 2

Background & Theory

This section is an introduction to self-supervised learning and relevant domains leveraged in the *SimCLR* algorithm.

2.1 Machine Learning

Machine learning is the study of algorithms that learn from data with the objective to perform a some sort of decision-making task, such as prediction or classification. Machine learning is typically categorized into two mainstream approaches: supervised and unsupervised, which are defined by their usage of labelled and unlabelled data sets respectively [9]. The focus of this paper, self-supervised learning, is a newer approach commonly thought to be an intermediate of the supervised and unsupervised machine learning.

2.1.1 Self-Supervised Learning

Self-supervised learning (SSL) is a machine learning approach composed of two phases or tasks: the *pretext* and *downstream* task. The pretext task uses unlabelled data to encode feature representations, which are later utilized for the main problem at hand, namely the downstream task [1]. In order to produced semantically meaningful features, the pretext task needs very be carefully designed [7]. For this reason, a common theme in many pretext tasks is to leverage the inherent properties of the data, such as predicting a hidden property of the data, or solving a puzzle-like problem. Examples of pretext task include masked word prediction in natural language processing and patch arrangement in computer vision [1]. In practice, training on the pretext task is often followed by further training on a smaller data set more relevant to the downstream task. For this reason, training via the pretext task is referred to as *pretraining* whilst the latter is called *fine-tuning*.

The unique advantage of SSL is its ability to utilize unlabelled data. Directly annotating data is time-consuming, requiring human labour and become particularly costly when needing expert knowledge. By leveraging transfer learning in SSL, i.e. knowledge transfer from the pretext to downstream task, one can generate a robust model despite having limited data in the objective task.

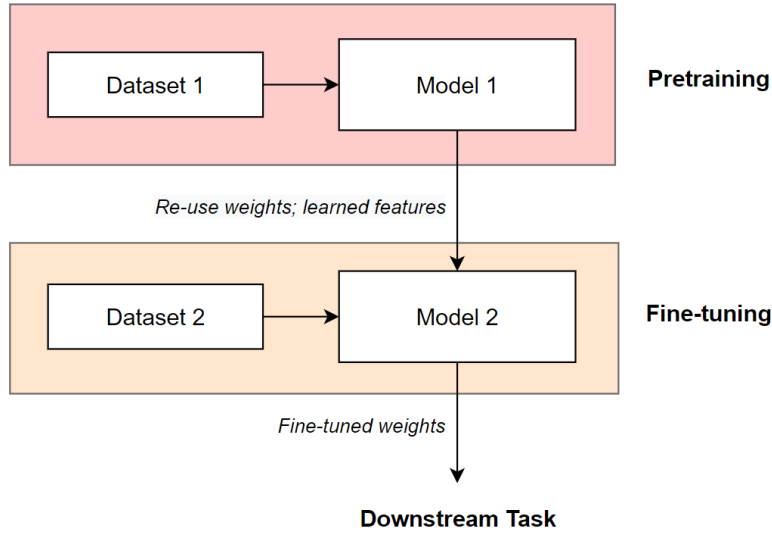


Figure 2.1: Pretraining and fine-tuning schema

2.2 Artificial Neural Networks

Within machine learning artificial neural networks (ANNs) are powerful tools for modelling tasks such as classification and prediction. In supervised learning, neural networks are used to train network parameters with respect to a data set with the hope of being able to predict a relevant set of features. Supervised machine learning problems can be divided into four components [11]:

- **Data:** the data we want to our model to learn from.
- **Model:** the model that we are training with respect to the labelled data.
- **Objective function:** a performance metric comparing our model prediction to the ground truth data set. Also known as cost function.
- **Optimization Algorithm:** the algorithm determining how to adjust out model’s parameters to optimize the objective function.

This sub-chapter will attempt to introduce ANN theory from a bottom-up approach, beginning with linear regression single-layer neural networks and ending with deep neural networks with more complex architectures.

2.2.1 Linear regression

Let $f(\mathbf{x})$ be an unknown function of interest defined on an unknown domain \mathcal{D} and $\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^n$ a set of n observations of $f(\mathbf{x})$, where $\mathbf{x}^{(i)} \in \mathbb{R}^d$ is a column vector,

$$\mathbf{x}^{(i)} = \begin{bmatrix} x_1^{(i)} & x_2^{(i)} & \dots & x_d^{(i)} \end{bmatrix}^T,$$

that is independently and identically distributed (i.i.d.) for $i = 0, \dots, n$, and $y \in \mathbb{R}$ a scalar. A linear regression model assumes an affine relationship between the explanatory variables $\{\mathbf{x}_i\}_{i=0}^n$ and response variable $\{y^{(i)}\}_{i=0}^n$, explaining the individual variability with a disturbance ϵ such that,

$$y = w_0 + x_1 w_1 + x_2 w_2 + \dots + x_d w_d + \epsilon = \mathbf{x}^T \mathbf{w} + \epsilon, \quad \mathbb{E}(\epsilon) = 0, \quad \text{Var}(\epsilon) < \infty,$$

where $\mathbf{w} \in \mathbb{R}^{d+1}$ is the column vector of weights. The model can also be written in matrix form,

$$\mathbf{y} = X\mathbf{w} + \boldsymbol{\epsilon},$$

where,

$$\mathbf{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix}, \quad X = \begin{bmatrix} 1 & -\mathbf{x}^{(1)T} \\ 1 & -\mathbf{x}^{(2)T} \\ \vdots & \vdots \\ 1 & -\mathbf{x}^{(n)T} \end{bmatrix} = \begin{bmatrix} 1 & x_1^{(1)} & \dots & x_d^{(1)} \\ 1 & x_1^{(2)} & \dots & x_d^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(n)} & \dots & x_d^{(n)} \end{bmatrix}, \quad \boldsymbol{\epsilon} = \begin{bmatrix} \epsilon^{(1)} \\ \epsilon^{(2)} \\ \vdots \\ \epsilon^{(n)} \end{bmatrix}.$$

In the context of machine learning, the constant w_0 is referred to as the bias whereas the set $\{w_i\}_{i=1}^n$ are referred to as the weights. We distinguish between the weights and bias, denoted b , by omitting the bias from the weight vector \mathbf{w} , reducing its dimension to \mathbb{R}^d .

The goal of linear regression is to find a function $h(x, w, b)$ that best approximates $f(x)$ over \mathcal{D} . We denote our model-based predictions \hat{y} , having the form,

$$\hat{y} = x_1 w_1 + x_2 w_2 + \dots + x_d w_d + b = \mathbf{x}^T \mathbf{w} + b =: h(\mathbf{x}, \mathbf{w}, b).$$

A common objective function for linear regression is the *mean square error* (MSE),

$$L(\mathbf{w}, b) = \text{MSE}(\mathbf{w}, b) := \frac{1}{n} \sum_{i=1}^n (\hat{\mathbf{y}}^{(i)} - \mathbf{y}^{(i)})^2 = \frac{1}{n} \sum_{i=1}^n (\mathbf{x}^{(i)T} \mathbf{w} + b - \mathbf{y}^{(i)})^2$$

The optimal parameters for the weight and bias (\mathbf{w}^*, b^*) of the model can be found by minimizing the MSE. The optimal parameters are therefore,

$$\mathbf{w}^*, b^* = \underset{\mathbf{w}, b}{\text{argmin}} L(\mathbf{w}, b).$$

2.2.2 Perceptron

A *perceptron* is a binary classifier with a structure corresponding to that of a neuron in a neural network. The perceptron's structure is analogous to that of a linear regression model, being composed of data, weights, a bias, and an output. The output of a perceptron i , denoted o_i , is defined as,

$$o_i = \begin{cases} 1, & \text{if } \mathbf{x}^T \mathbf{w} + b \geq 0, \\ 0, & \text{if } \mathbf{x}^T \mathbf{w} + b \leq 0. \end{cases}$$

A visual scheme of a perceptron is provided in Figure 2.2. When counting the depth or amount of layers in a neural network, we count the number of layers with tunable weights. Since the perceptron only has one layer with tunable weights, its depth is one. Single-layer neural networks can also have several outputs, as shown in Figure 2.3.

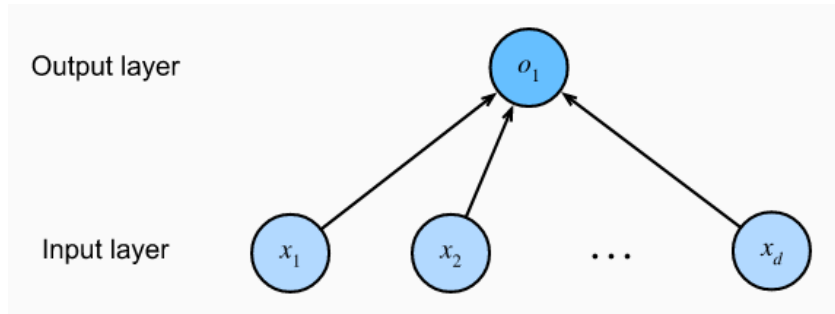


Figure 2.2: Perceptron

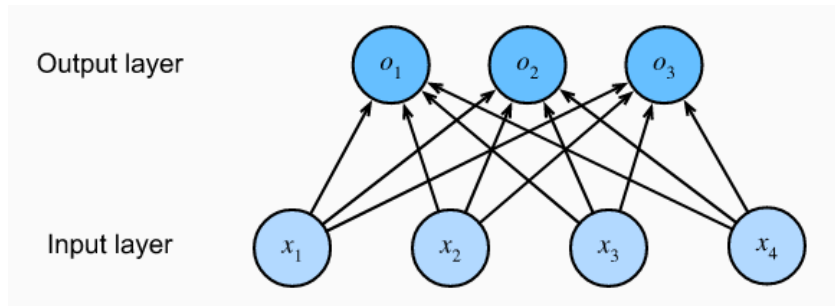


Figure 2.3: Single-layer neural network

2.2.3 Multilayer Perceptron

The *multilayer perceptron* (MLP) is a development of the perceptron, introducing more layers and non-linear activation functions. These properties are characteristic in deep learning, where one utilizes deep ANNs with many layers.

Layers

In MLPs, hidden layers are incorporated between the input layer and output layer. Often, all layers in an MLP are fully-connected, meaning that each node in one layer is connected to every node in the subsequent layer.

Nonlinearity & Activation Functions

An activation function defines the output of a neuron given the input, weights and bias. Since the perceptron is a binary classifier, the activation function is a binary step function. Denote each node in a hidden layer $h_i = \mathbf{x}^T \mathbf{w}^{(i)} + b^{(i)}$. In vector form, we can express the hidden layers $\{h_i\}_{i=1}^n$ as

$$\mathbf{h} = \mathbf{x}^T \mathbf{W}^{(1)} + \mathbf{b}^{(1)},$$

where $\mathbf{W} \in \mathbb{R}^{d \times q}$ and $\mathbf{b} \in \mathbb{R}^q$ represents the weights and biases of the hidden layers respectively, with q denoting the number of nodes in the hidden layer. Similarly, the outputs $\{o_i\}_{i=1}^n$ can be defined in vector form as,

$$\mathbf{o} = \mathbf{h}^T \mathbf{W}^{(2)} + \mathbf{b}^{(2)}.$$

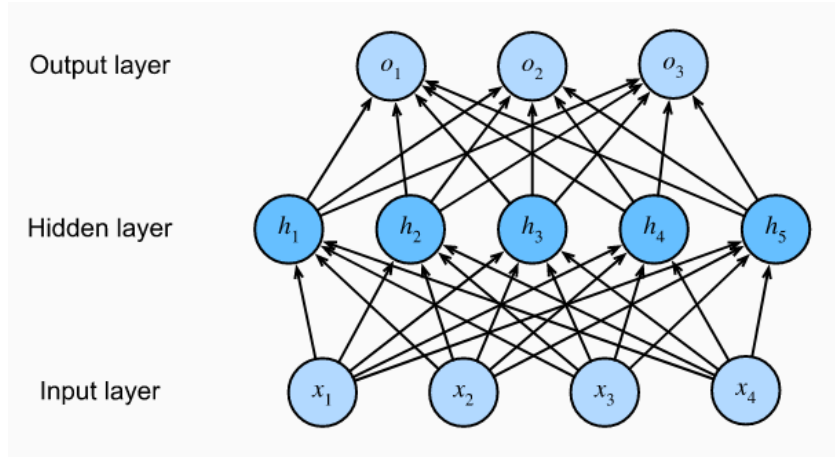


Figure 2.4: Multilayer Perceptron (MLP)

We can thus express the outputs linearly,

$$\begin{aligned}
 \mathbf{o} &= (\mathbf{x}^T \mathbf{W}^{(1)} + \mathbf{b}^{(1)}) \mathbf{W}^{(2)} + \mathbf{b}^{(2)} \\
 &= \mathbf{x}^T \mathbf{W}^{(1)} \mathbf{W}^{(2)} + \mathbf{b}^{(1)} \mathbf{W}^{(2)} + \mathbf{b}^{(2)} \\
 &= \mathbf{x}^T \mathbf{W} + \mathbf{b}
 \end{aligned}$$

where,

$$\mathbf{W} = \mathbf{W}^{(1)} \mathbf{W}^{(2)}, \quad \mathbf{b} = \mathbf{b}^{(1)} \mathbf{W}^{(2)} + \mathbf{b}^{(2)}.$$

Being able to express the outputs linearly causes the hidden layers to become redundant. To realize the full potential of the hidden layers, MLPs use non-linear activation functions. Denote $\sigma(\cdot)$ as multivariate function containing the activation functions for \mathbf{h} . By applying non-linear activation functions to the hidden layers \mathbf{h} , we can no longer express \mathbf{o} linearly, but instead as

$$\mathbf{o} = \mathbf{h}^T \mathbf{W}^{(2)} + \mathbf{b}^{(2)}, \quad \mathbf{h} = \sigma(\mathbf{x}^T \mathbf{W}^{(1)} + \mathbf{b}^{(1)}).$$

To build deeper MLPs, we stack the hidden layers, for example,

$$\mathbf{h}_1 = \sigma_1(\mathbf{x}^T \mathbf{W}^{(1)} + \mathbf{b}^{(1)}), \quad \mathbf{h}_i = \sigma_i(\mathbf{h}_{i-1} \mathbf{W}^{(i)} + \mathbf{b}^{(i)}), \quad i = 2, 3, \dots$$

One of the most common activation functions is the *rectified linear unit* (ReLU) function, defined as,

$$\text{ReLU}(x) = \max(0, x). \tag{2.1}$$

Others popular non-linear activation functions include the sigmoid functions such as the logistic function and hyperbolic tangent function.

2.2.4 Classification

In practice, many problems strive to classify inputs into a set of predefined categories, rather than predict. Given the observations $(\mathbb{X}, \mathbb{Y}) = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=0}^n$ and a class set $\mathcal{C} = \{c_i\}_{i=1}^m$, the goal of classification is to classify a new input into one of classes in the class set \mathcal{C} . To create such a classifier, one can train an ANN designed such that it has m outputs, each output corresponding to one class. The output o_i represents the probability of the input \mathbf{x} belonging to class c_i , denoted $P(y = c_i | \mathbf{x})$. In order for the outputs $\mathbf{o} = \{o_i\}_{i=1}^m$ to represent probabilities, they must be non-negative and sum to one. To guarantee these properties, the *softmax function* is applied, defined as,

$$\mathbf{softmax}(\mathbf{o})_i = \frac{e^{o_i}}{\sum_{k=1}^m e^{o_k}}.$$

When applied to output \mathbf{o} , the result is a vector $\hat{y} = [\hat{y}_1, \dots, \hat{y}_m]^T$ such that,

$$\hat{y}_i = \mathbf{softmax}(\mathbf{o})_i.$$

To represent the true class of a particular input x , we use a so-called *one-hot encoding*, a m -vector where m is the number of classes. The one-hot encoding contains zeroes at all entries except the k th entry, where k corresponds to the true class c_k , which contains a one, i.e.

$$\mathbf{onehot}(k)_j = \begin{cases} 1 & \text{if } k = j, \\ 0 & \text{if } k \neq j. \end{cases} \quad (2.2)$$

Maximum Likelihood Estimation

A common objective function within logistic regression and classification is the *maximum likelihood estimator*, where the objective is to maximize the likelihood of observing the training data (\mathbb{X}, \mathbb{Y}) given some model parameter(s) θ .

Definition 2.2.1. Let X be a discrete random variable X with outcome x . The likelihood function of the parameters $\theta = (\theta_1, \dots, \theta_d)$ given the outcome x of random variable X is defined as,

$$\mathcal{L}(\theta | x) = P(x | \theta) = p_\theta(x),$$

where $p_\theta(x)$ is a probability mass function dependent on model parameters θ .

Given many observations $\mathbb{X} = \{x_i\}_{i=1}^n$ that are conditionally independent, the likelihood can be represented as a product,

$$\mathcal{L}(\theta | \mathbb{X}) = P(x_1, x_2, \dots, x_n, | \theta) = \prod_{i=1}^n p_\theta(x_i) = \prod_{x \in X} p_\theta(x)^{Nq(x)},$$

where $q(x)$ denotes the relative frequency of outcome x in the observation set \mathbb{X} . By taking the negative log-likelihood, i.e. the negative of the natural logarithm of the likelihood $\mathcal{L}(\theta | X)$, and dividing by N one gets,

$$-\frac{1}{N} \log \mathcal{L}(\theta | \mathbb{X}) = -\frac{1}{N} \sum_{x \in X} \log p_\theta(x)^{Nq(x)} = -\sum_{x \in X} q(x) \log p_\theta(x).$$

The last expression is known as the *cross-entropy* of p relative to q , denoted $H(p, q)$, and is a common loss function used in the training of classification models [11]. It is evident that maximizing the likelihood is equivalent to minimizing the cross-entropy and negative log-likelihood since the logarithm is a monotonically increasing function.

Cross Entropy Loss

Cross-entropy originates from field of information theory, the study of transmission and processing of digital information. Before delving directly to cross-entropy, relevant terms from information theory are introduced along with some intuition.

Definition 2.2.2. Given a discrete random variable X and probability mass function $p_X(x)$, the *information content* or *surprisal* of measuring X as outcome x is defined as

$$I_X(x) = \log \left(\frac{1}{p_X(x)} \right) = -\log(p_X(x)).$$

The surprisal of an outcome x is meant to quantify the amount of surprise upon observing the outcome x , a basic quantity intrinsically related to probability that provides particular mathematical advantage within information theory. In information theory, surprisal encodes the amount of information provided by an event x , using the unit *bit* and *nat* when the logarithm has base 2 and base e respectively. In digital communications, for example, one can effectively use memory by encoding common events to messages containing less information.

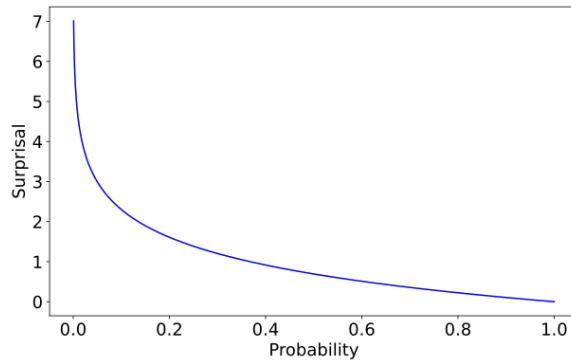


Figure 2.5: Probability *vs.* surprisal. Probabilities tending towards zero and one have surprisals tending to infinity and zero respectively.

Definition 2.2.3. Given a discrete random variable X with possible values x_1, x_2, \dots, x_i and probability mass function $p_X(x)$, the *entropy* $H(X)$ is defined as

$$H(X) = \mathbb{E} \log \left[\frac{1}{p_X(x)} \right] = -\sum_x p_X(x) \log p_X(x),$$

where \mathbb{E} denote the expectation.

Intuitively, the entropy quantifies the expected surprisal of a random variable.

Definition 2.2.4. Given a discrete random variable X with possible values x_1, x_2, \dots, x_i , and two probability distributions (p.d.'s) $p(x)$ and $q(x)$, the *cross-entropy* of p relative to q is defined as

$$H(p, q) = \mathbb{E}_p \log \left[\frac{1}{q(x)} \right] = - \sum_x p(x) \log q(x),$$

where \mathbb{E}_p denote the expectation with respect to p.d. p . Note that $H(p, q) \neq H(q, p)$.

Let \hat{y} and y represent an empirical and true p.d. respectively. In the context of training an ANN classification model with m classes $\{c_i\}_{i=1}^m$, \hat{y} would represent softmax layer output as an m -vector, where,

$$\hat{y}_i = \mathbf{softmax}(\mathbf{o})_i = \frac{e^{o_i}}{\sum_{j=1}^m e^{o_j}},$$

represents the probability that input x belongs to class c_i . If the input x belongs to class c_k , the true p.d. y would be a one-hot encoding of size m with the one placed at position k . Since the all weight is put on class c_k in the true p.d. y , one can simplify the cross entropy of the true relative to the empirical p.d. $H(y, \hat{y})$ to,

$$H(y, \hat{y}) = - \sum_{i=1}^m y_i \log \hat{y}_i = - \log \hat{y}_k = - \log \frac{e^{o_k}}{\sum_{i=1}^m e^{o_i}},$$

which will be a reoccurring representation in the contrastive loss function defined in Section 2.5 *SimCLR*. In practice, an objective function will calculate the average over many samples,

$$\frac{1}{n} \sum_{k=1}^n H(y^{(k)}, \hat{y}^{(k)}) = - \frac{1}{n} \sum_{k=1}^n \left(\sum_{i=1}^m y_i^{(k)} \log \hat{y}_i^{(k)} \right),$$

where n is the number of samples. Let the predicted label of a feature x with label y be denoted as $\hat{y} = f_\theta(x)$. Given the feature-label pairs $\{(x^{(i)}, y^{(i)})\}_{i=0}^n$, the optimal parameter vector θ^* for a model optimized with cross-entropy loss can then be defined as,

$$\theta^* = \operatorname{argmin}_{\theta} \frac{1}{n} \sum_{k=1}^n H(y^{(k)}, \hat{y}^{(k)}) = \operatorname{argmin}_{\theta} \frac{1}{n} \sum_{k=1}^n H(y^{(k)}, f_\theta(x^{(k)}))$$

2.2.5 Convolutional Neural Networks

Convolutional neural networks (CNNs) is a type of ANN designed mainly for applications within computer vision. CNN architectures can be split into to sections containing different types of layers: the feature learning and classification section. The feature learning section consists of two types of layers: convolutional and pooling layers.

- **Convolutional layers:** applying convolutions across image using kernel filters with a pre-determined size, stride and padding. Note that filters can have more than two dimensions, as in for example filters for RGB images. The term kernel refers to the convolution applied to each channel, whereas the term filter refers to the full set of kernels applied to all channels. Several convolutions can be applied in one layer, outputting an array of 'feature maps'.

- **Pooling layers:** reducing the dimension of an image or feature map by pooling batches of pixels together. This is often done either by taking choosing the maximum pixel of the batch (*max pooling*) or averaging the pixels of the batch (*average pooling*).

After a number of convolutional and pooling layers, the classification section begins with a flattening of the feature maps, i.e. consecutively storing the rows of the feature maps as one column. With the data now being in vector form, an MLP is applied to classify the original input image in a predetermined set of class. This is often done by outputting a vector with elements corresponding to probabilities of the image belonging to a certain class, classifying the image to the class with the highest probability after a (*softmax*) layer.

Definition 2.2.5. Given an image f and a kernel g , the 2D discrete convolution is defined as,

$$(f * g)[i, j] = \sum_a \sum_b f[a, b] \cdot g[i - a, j - b].$$

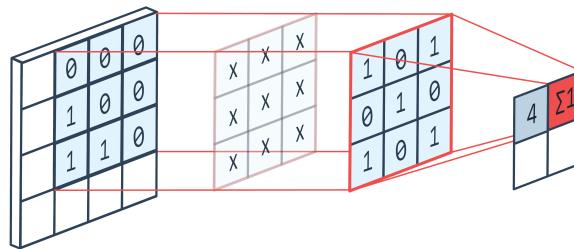


Figure 2.6: Convolution operation displayed visually

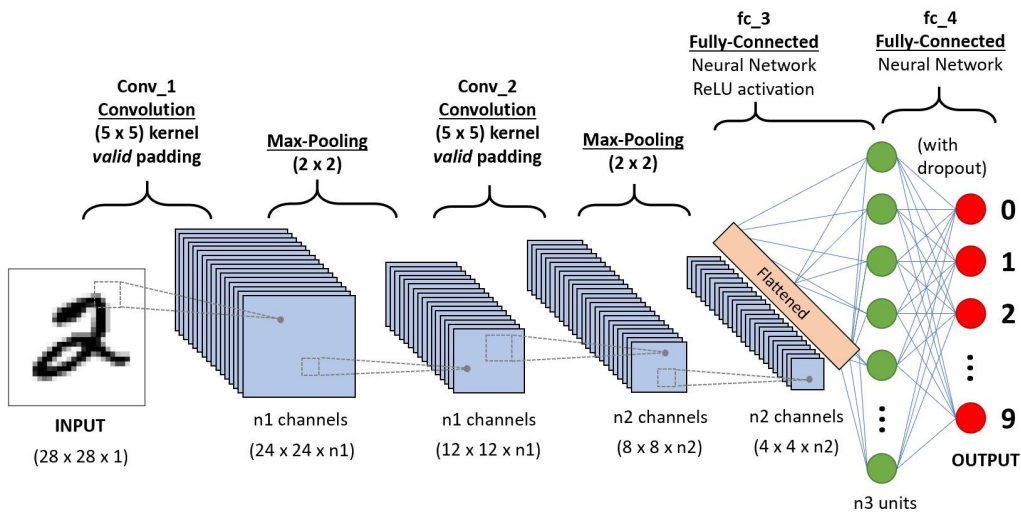


Figure 2.7: CNN Architecture Example

2.2.6 Residual Networks

The intention of stacking layers in deep ANNs is to enrich the features of the model, however stacking layers can also lead to the degrade features. This problem, known as the degradation problem, causes model accuracy to become saturated and eventually rapidly decrease. The cause of this problem is not a result of overfitting, but is rather attributed to poor weight initialization, a poor optimization function, or the notorious the problem of vanishing/exploding gradients.

One solution to the degradation problem are *residual blocks*. Residual blocks utilize skip-connections, which work by propagating activations to a layer two or three layers ahead in addition to the subsequent layer [5]. Skip-connections counteracts the degradation problem in two ways:

(i) alleviating the vanish gradient problem by providing shortcuts during back-propagation, and (ii) allowing the model to learn an identity function, making sure that higher layers do not perform worse than lower layers. Neural network architectures utilizing residual blocks are referred to as *residual networks* or *ResNets*.

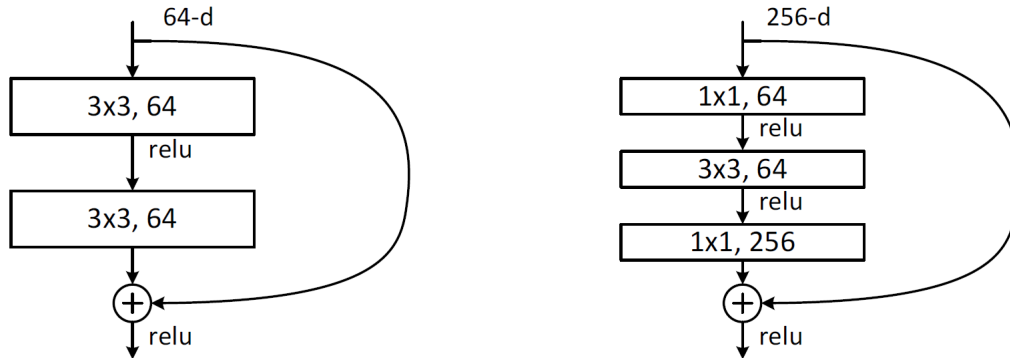


Figure 2.8: Left: Residual block. Right: Bottleneck residual block.

ResNet-50

The *ResNet-50* is a particular ResNet variant with 50-layers using a modified residual block referred to as a *bottleneck residual block*. Bottleneck residual blocks skips three layers (instead of two as in regular residual blocks), where the three intermediate layers are convolution layers with kernel sizes: 1x1, 3x3, and 1x1 from beginning to end respectively. The bottleneck variant was implemented to deeper ResNet architectures due concerns of training time. Findings show that the deeper ResNet variants (maximum 152-layer) perform better on top-1 and top-5 error rates on *ImageNet* relative to shallower variants.

Layer name	Output size	Layer contents
conv1	112x112	7x7, 64, stride 2
conv2	56x56	3x3 maxpool, stride 2
		$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3	14x14	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$
conv4	7x7	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$
conv5	7x7	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1x1	5

Table 2.1: ResNet-50 Architecture for *ImageNet* [5]. Matrices represent the bottleneck residual blocks, where each entry consists of the the kernel size followed by the number of kernels. The multiple after the matrix, say n , means that the block is repeated n times.

2.3 Optimization [11]

Given an objective function L and a training set $(\mathbb{X}, \mathbb{Y}) = \{(x^{(i)}, y^{(i)})\}_{i=1}^n$, the optimal parameters of the hypothesized model is found by solving the optimization problem,

$$\theta^* := \underset{\theta}{\operatorname{argmin}} L(\theta).$$

In words, the optimal parameters are the parameters that minimize the objective function. In this chapter, we introduce the most common optimization algorithms used in practice as well as the *LARS* optimizer, an optimizer particularly useful for self-supervised learning.

2.3.1 Gradient Descent

Gradient descent, also called *batch gradient descent*, is a first-order iterative optimization algorithm. Once a starting point θ_0 is initialized, gradient descent searches for the minimum by iteratively taking steps in the opposite direction of the gradient at a point with a set step size, also known as learning rate η . Mathematically, the iteration i can be expressed as,

$$\theta_{i+1} = \theta_i - \eta \nabla L(\theta_i),$$

where the objective function L is defined as a sum over each observation in the dataset, i.e.

$$L(\theta) = \frac{1}{n} \sum_{i=1}^n l^{(i)}(\theta) = \frac{1}{n} \sum_{i=1}^n l(h(x^{(i)}, \theta), y^{(i)}) = \frac{1}{n} \sum_{i=1}^n l(\hat{y}^{(i)}, y^{(i)}).$$

The function $l(\cdot, \cdot)$ is called the loss function and defines an error metric between a prediction $\hat{y} = h(x, \theta)$ and its true value y .

Gradient descent has several drawbacks. For large datasets, the calculation of the gradient $\nabla L(\theta)$ is very costly and redundant when data points are similar. Limited computer memory can also pose a problem with large datasets. Lastly, gradient descent doesn't allow us to add new data to our dataset during training, characterizing the algorithm as *offline*.

2.3.2 Stochastic Gradient Descent

Stochastic gradient descent (SGD) is a variant of gradient descent where each iteration evaluates the gradient of one observation at a time, rather than the gradient of the whole dataset/batch,

$$\begin{aligned} \theta_{i+1} &= \theta_i - \eta \nabla L(\theta_i), \\ L(\theta) &= l^{(k_i)}(\theta) = l(h(x^{(k_i)}, \theta), y^{(k_i)}), \end{aligned}$$

where $k_i \in \{1, \dots, n\}$ is a uniformly-random chosen index number at iteration i . It can be shown that the gradient for SGD is unbiased, that is,

$$\mathbb{E}[\nabla l^{(k_i)}(\theta)] = \frac{1}{n} \sum_{i=1}^n \nabla l^{(i)}(\theta).$$

The variance, however, is n times larger for SGD estimate compared to the estimate for batch gradient descent.

2.3.3 Minibatch Stochastic Gradient Descent

Minibatch stochastic gradient descent (minibatch-SGD) is another variants of gradient descent which addresses the problems in the aforementioned variants. In minibatch-SGD, a *minibatch* B of uniformly random sample of data points from the training data $\mathbb{X} = \{(x^{(i)}, y^{(i)})\}_{i=0}^n$ are utilized in the objective function. The size of the minibatch, denoted $|B|$, is a predetermined hyperparameter. The objective function L therefore has the form,

$$L(\theta) = \frac{1}{|B|} \sum_{i \in B} l(h(x^{(i)}, \theta), y^{(i)}).$$

It can be shown that the gradient estimate is unbiased, i.e.

$$\mathbb{E} \left[\frac{1}{|B|} \sum_{i \in B} \nabla l(h(x^{(i)}, \theta), y^{(i)}) \right] = \frac{1}{n} \sum_{i=1}^n \nabla l^{(i)}(\theta),$$

and has a variance $|B|$ times smaller than that of the SGD estimate.

By using $|B|$ data points per iteration, the gradient calculation is significantly less computationally expensive and the demand on memory is minimized. SGD also grants *online learning*, meaning that we can add new data to our dataset during training. When utilizing graphical processing units (GPUs) and parallelization, minibatch-SGD has a particularly high computational efficiency.

Note: To guarantee the convergence to an global or local minimum of gradient descent algorithms, one has to demand criteria such as convexity. In practice, these criteria are typically not fulfilled yet still demonstrate success.

2.3.4 Momentum

Adding momentum to the optimization is a common way of accelerating learning and gradient descent. Momentum implements an exponential moving average of the gradient terms from previous iterations, thereby preventing large oscillations in valley-like surfaces and increasing the rate of descent. With momentum implemented, one iteration of gradient descent is defined as,

$$\begin{aligned} v_{i+1} &= \gamma v_i + \eta \nabla L(\theta_i), \\ \theta_{i+1} &= \theta_i - v_{i+1}, \end{aligned}$$

where $v_0 = 0$ and $\gamma \in (0, 1)$ is a hyperparameter specifying the window of the exponential moving averages. The larger γ chosen, the larger the affect of previous steps on the current step.

2.3.5 Adaptive Learning Rate

In the previous optimization algorithms, the learning rate is kept the same for each iteration and for each of the individual parameters inside the parameter vector θ . When too large, constant learning rates will cause the algorithm to diverge from the minimum, whereas when too small, can substantially slow the learning process. By making learning rates adaptive, the learning rate can change for each iteration as well as for each individual parameter.

2.3.6 Layer-wise Adaptive Rate Scaling [4]

Layer-wise Adaptive Rate Scaling (LARS) is an optimizer specifically designed to facilitate large batch training of deep neural networks. Large batch training allows fewer number of iterations per epoch, and by parallelizing the workload, training becomes very effective. A common observation for large-batch training is however poor generalization error, attributed to the fact larger-batches favour sharper minima relative to smaller batch sizes. The *LARS* optimizer has proven to enable large-batch training of deep ANNs up to batch sizes of 32K. It manages to achieve this by scaling the learning rate for each layer of the neural network separately based on the ratio of the Euclidean norm of the weights to the Euclidean norm of the gradients for that layer.

2.4 Image Processing

Gaussian Blur

The Gaussian blur is a common blurring technique in image processing which convolves the image with a Gaussian kernel [10]. The Gaussian kernel has a predetermined integer size (odd, to ensure a center) and variance σ^2 . The continuous isotropic (i.e. circularly symmetric) Gaussian function $G(x, y)$ with variance σ centered at $(x, y) = (0, 0)$ is defined as,

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}.$$

To generate a Gaussian filter, the continuous Gaussian function needs to be discretized. This can be done by a number of different ways, such as picking the midpoint of each pixel or integrating over all values through each pixel, where the pixels have with some predetermined dimensions.

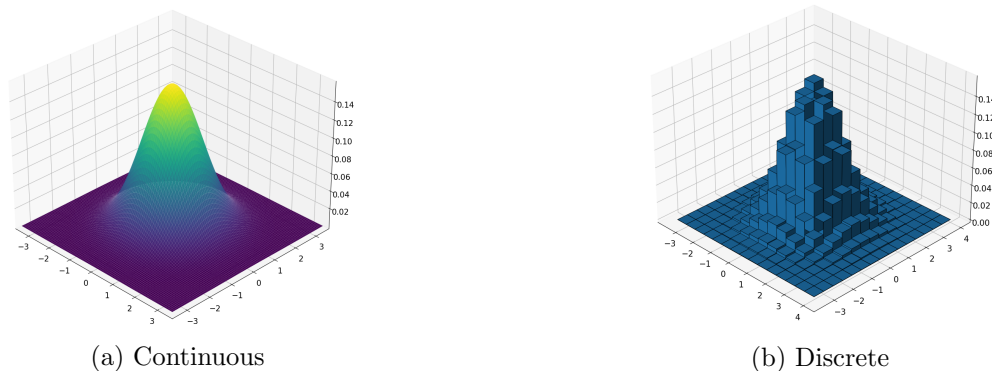


Figure 2.9: Continuous *vs.* discrete Gaussian functions. Centered at $(x, y) = (0, 0)$ with variance $\sigma^2 = 1$ and pixel size $(l, w) = (0.5, 0.5)$. Discretized by choosing midpoint of pixel.

The values of a discrete Gaussian distribution can also be approximated by binomial coefficients [3],

$$B(n, p) = \frac{n!}{(n-p)!p!}, \quad n, p \in \mathbb{N}, \quad 0 \leq p \leq n.$$

Such an approximation can be visualised in Figure 2.10. An example of a Gaussian kernel approximated by binomial coefficients is:

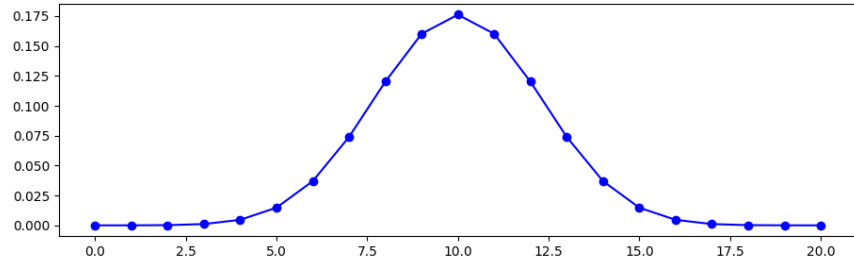


Figure 2.10: Normalized binomial coefficients $B(n, p)$, $n = 20$, $p = 0, 1, \dots, 20$.

$$\frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix} = \frac{1}{256} \begin{bmatrix} 1 \\ 4 \\ 6 \\ 4 \\ 1 \end{bmatrix} [1 \ 4 \ 6 \ 4 \ 1].$$

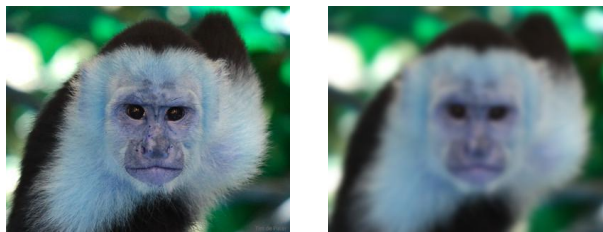
With a given Gaussian kernel and image, the image is blurred by applying a 2D discrete convolution.

Color Distortion

This subsection describes color distortion in the context of relevant framework, SimCLR, detailed in the next chapter.

Color distortion is composed of two transformations:

- **Color jitter:** defined by four consecutive random augmentations, manipulating: brightness, contrast, saturation and hue of the image. The augmentations have no particular order, and are shuffled upon each call in practice.
- **Color drop:** converting the RGB image to gray-scale.



(a) Normal

(b) Gaussian blur

Figure 2.11: Blur comparison

2.5 SimCLR

SimCLR is an algorithmic framework developed by *Google* for learning visual representations using *contrastive learning* [2]. Contrastive learning, a common theme within self-supervised learning, is an approach to learning based on the premise that similar data samples should have similar representations and dissimilar data samples should have dissimilar representations in the latent feature space. The *SimCLR* framework, visualized in Figure 2.12 and detailed in pseudocode in Algorithm 1, is composed of four major components:

1. Stochastic Data Augmentation:

The learning process begins with an image \mathbf{x} which is augmented in two ways, denoted $\tilde{\mathbf{x}}_i$ and $\tilde{\mathbf{x}}_j$, by applying a sequence of random transformation from a predefined family of augmentations \mathcal{T} . The operators $t, t' \sim \mathcal{T}$ represent the two different augmentations. The sequence of random transformations consists of: *random cropping* followed by *resize back to the original size*, *random color distortions* and *random Gaussian blur*.

2. Base Encoder:

A neural network base encoder $f(\cdot)$ that transforms that augmented images into vector representation, denoted \mathbf{h} . Thus, $\mathbf{h}_i = f(\tilde{\mathbf{x}}_i)$ and $\mathbf{h}_j = f(\tilde{\mathbf{x}}_j)$. The encoding function $f(\cdot)$ is the ResNet-50 neural network, as in Table 2.1.

3. Projection Head:

The projection head $g(\cdot)$ is a MLP with one-hidden layer, analogous to the MLP in Figure 2.4, mapping \mathbf{h} to the space where contrastive loss is taken place. The output of the projection head is \mathbf{z} , defined as $\mathbf{z} = \mathbf{w}^{(2)} \sigma(\mathbf{h}^T \mathbf{w}^{(1)})$, where σ is the ReLU function (Def. 2.1) and $\mathbf{w}_i, i = 1, 2$ represents the weights of the layers.

4. Contrastive Loss Function:

Since N images are augmented in two ways we are left with $2N$ images. During the training of the ANN model, an objective function $L(\cdot)$ looping through all $2N$ images is used,

$$L = \frac{1}{2N} \sum_{k=1}^N [l(2k-1, 2k) + l(2k, 2k-1)], \quad (2.3)$$

$$l(i, j) = -\log \frac{\exp(s_{i,j}/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{k \neq i} \exp(s_{i,k}/\tau)}, \quad s_{i,j} = \frac{z_i^T z_j}{\|z_i\| \|z_j\|},$$

where $l(\cdot, \cdot)$ is called the *normalized temperature-scaled cross entropy loss*, abbreviated to *NT-Xent*, and $s_{i,j}$ is the *cosine similarity* of z_i and z_j . Note the similarity between $l(i, j)$ and equation (2.2.4) representing the simplified cross entropy, with the most prominent difference being the addition of a temperature factor τ in $l(i, j)$. This contrastive loss function is chosen due to its correspondence to the contrastive pretext task: classifying the augmented images as having the same or different origins.

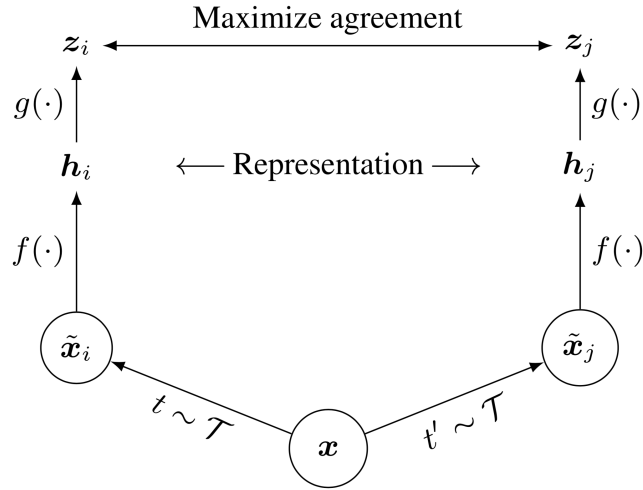


Figure 2.12: *SimCLR* Framework.

Comparison

SimCLR considerably outperforms other self-supervised and semi-supervised methods on classification on the *ImageNet* dataset, achieving a 76.5% and 93.2% accuracy for top-1 and top-5 classification respectively. The previous state-of-the-art was *CPCv2* with a 52.7% and 77.9% accuracy for top-1 and top-5 respectively. *SimCLR* even manages to match the performance of a supervised ResNet-50 classifier. A summary of the result is provided in Figure 2.13.

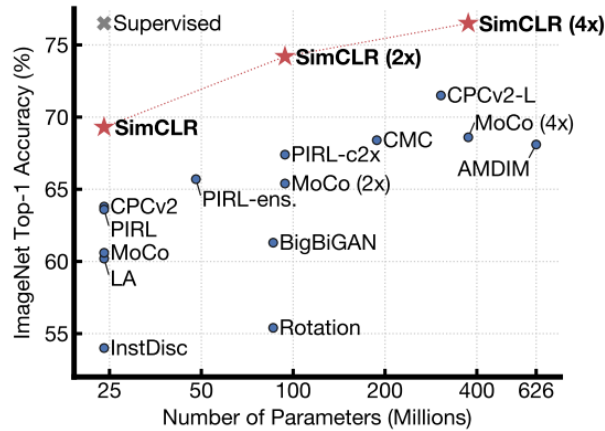


Figure 2.13: *ImageNet* Top-1 accuracy of linear classifiers trained on representations learned with different self-supervised methods (pretrained on *ImageNet*). Gray cross indicates supervised ResNet-50.

Algorithm 1 *SimCLR*'s main learning algorithm.

input: batch size N , constant τ , structure of f, g, \mathcal{T} .
for sampled minibatch $\{\mathbf{x}_k\}_{k=1}^N$ **do**
 for all $k \in \{1, \dots, N\}$ **do**
 draw two augmentation functions $t \sim \mathcal{T}, t' \sim \mathcal{T}$
 # the first augmentation
 $\tilde{\mathbf{x}}_{2k-1} = t(\mathbf{x}_k)$ # representation
 $\mathbf{h}_{2k-1} = f(\tilde{\mathbf{x}}_{2k-1})$ # projection
 $\mathbf{z}_{2k-1} = g(\mathbf{h}_{2k-1})$
 # the second augmentation
 $\tilde{\mathbf{x}}_{2k} = t'(\mathbf{x}_k)$ # representation
 $\mathbf{h}_{2k} = f(\tilde{\mathbf{x}}_{2k})$ # projection
 $\mathbf{z}_{2k} = g(\mathbf{h}_{2k})$
 end for
 for all $i \in \{1, \dots, 2N\}$ and $j \in \{1, \dots, 2N\}$ **do**
 $s_{i,j} = \mathbf{z}_i^T \mathbf{z}_j / \|\mathbf{z}_i\| \|\mathbf{z}_j\|$ # pairwise similarity
 end for
 define $l(i, j)$ **as** $l(i, j) = -\log \frac{\exp s_{i,j}/\tau}{\sum_{k=1}^{2N} \mathbb{1}_{k \neq i} \exp s_{i,k}/\tau}$
 $\mathcal{L} = \frac{1}{2N} \sum_{k=1}^N [l(2k-1, 2k) + l(2k, 2k-1)]$
 update networks f and g to minimize \mathcal{L}
end for
return encoder network $f(\cdot)$ and throw away $g(\cdot)$

Chapter 3

Method

3.1 Data

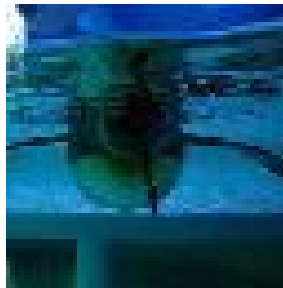
Two datasets will be used in this thesis: *TinyImageNet* and *BigEarthNet-S2*.

TinyImageNet

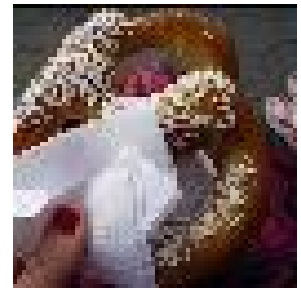
TinyImageNet is a dataset of 200 generic classes, such as *tractor*, *fur coat*, *barrel*, with each image having the dimensions 64×64 . Two dataset splits are used: the train and test split. The train split contains 200 images per class, corresponding to 100,00 images, whereas the test split contains 50 images per class, corresponding to 10,000 images. Example images/labels are shown in Figure 3.1



(a) *Bus*



(b) *Penguin*



(c) *Pretzel*

Figure 3.1: *TinyImageNet* examples

BigEarthNet

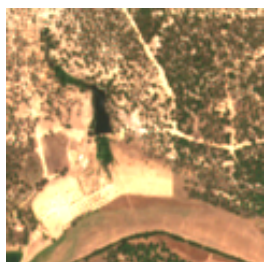
The dataset utilized in this thesis is the RGB-configured *BigEarthNet-S2* captured by the *Copernicus Sentinel-2* satellites. The original dataset contains satellite imagery of over 10 countries in Europe which are divided into 590326 non-overlapping image patches. Each image has the dimensions 120×120 pixels, corresponding to 1.2×1.2 km on the ground, thereby each pixel corresponding to 10 meters. The dataset is multi-labelled with 43 imbalanced labels. This implies that each image contains several labels and that the dataset cannot be divided into clearly divided into non-overlapping classes. Examples of images and their corresponding labels are in Figure 3.2.



(a) *Construction sites, Vineyards, Pastures, Complex cultivation patterns, Broad-leaved forest, Transitional woodland/shrub, Water courses.*



(b) *Discontinuous urban fabric, Mineral extraction sites, Pastures, Sea and ocean.*



(c) *Non-irrigated arable land, Permanently irrigated land, Agro-forestry areas, Coniferous forest, Mixed forest.*

Figure 3.2: BigEarthNet-examples

3.2 Evaluation

To evaluate the quality of the representations learnt by *SimCLR* a downstream task needs to be established. We choose to use the common linear evaluation protocol, where a linear classifier is used as an evaluation metric. The linear classifier is a single-layer neural network with a softmax function applied at the output layer. Training the linear classifier is relatively straight-forward for *TinyImageNet* since it has distinguished, non-overlapping classes, however *BigEarthNet-S2* it is a multi-labelled. Therefore each image needs to be categorized from a chosen set of non-overlapping classes. Categorizing semantically-diverse images with many labels into distinct classes requires a significant amount of generalization. To maintain simplicity, we choose to build a binary classifier classifying images as *farmland* or *not-farmland*. We define farmland as any image containing any of the following labels:

- *Complex cultivation patterns*
- *Land principally occupied by agriculture, with significant areas of natural vegetation*
- *Non-irrigated arable land*

These labels were chosen as they appear semantically similar when observing samples from the dataset.

3.2.1 Data partitioning

The *SimCLR* algorithm provided by *Google* gives the option to train the ResNet-50 base encoder and conduct supervised training of the linear classifier simultaneously, since this achieves similar performance as sequential training of base encoder and linear classifier [2]. Therefore only one data split is needed for training and another to evaluate the trained network, the testing set. One quarter of the *BigEarthNet-S2* dataset is partitioned with a 3:1 training to testing ratio.

Training	Testing
109830	36610

Table 3.1: *BigEarthNet-S2*: number of images in each split

TinyImageNet has the following pre-determined splits.

Training	Testing
100000	10000

Table 3.2: *TinyImageNet*: number of images in each split

3.3 Model & Training

Training occurs in two ways simultaneously: self-supervised pretraining and supervised training of the linear classifier. Pre-training will train the ResNet-50 base encoder and projection head as detailed in Algorithm 1. The training of the linear classifier has no influence on the base encoder and only trains the parameters of the single-layer neural network according to the downstream classification task. The two datasets have different purposes: *TinyImageNet* is used to verify that learning via *SimCLR* indeed works, whereas *BigEarthNet-S2* is experimental with the hope of confirming that contrastive learning is effective within remote sensing.

Training will be run for 50 epochs with a learning rate $\eta = 1.0$ for both datasets. We choose to train the model in six variations for *TinyImageNet*, with three and two distinct values of the batch size and the contrastive learning temperature respectively. *BigEarthNet-S2* on the other hand was trained with four variations, changing both the batch size and the contrastive learning temperature twice.

Batch size N	256	1024	- / 2048
Temperature τ	0.25	0.5	-

Table 3.3: Hyperparameter values to be tested

After the initial results, the most successful hyperparameters for *TinyImageNet* are chosen for further testing into training length, where the number of epochs trained are increased to 100, 200, 400 epochs.

As the original *SimCLR* paper warns that training may be unstable when using standard SGD/Momentum optimizers for large batch sizes, we opt to use the recommended *LARS* optimizer.

Chapter 4

Results

In this section, model training and evaluation metrics are presented. Training metrics consist of contrastive and supervised accuracies and loss, with definitions provided below.

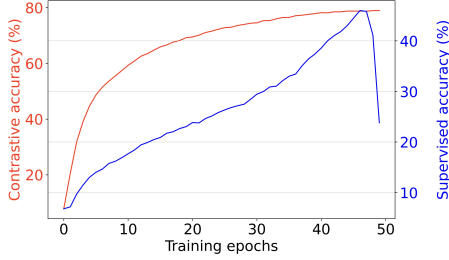
- **Contrastive accuracy:** the accuracy of the the contrastive prediction task of *SimCLR*'s pretext task.
- **Contrastive loss:** the loss contrastive function as defined in equation (2.3), facilitating the contrastive self-supervised learning.
- **Supervised accuracy:** the accuracy of the base encoder in the downstream classification task: classifying *farmland*.
- **Supervised loss:** the cross-entropy loss of the true classes with respect to the predicted classes, as detailed in equation (2.2.4).

For *TinyImageNet* the evaluation metrics consist of the top-1 and top-5 classification accuracy and a per-class top-1 accuracy bar chart. For *BigEarthNet-S2* the evaluation metrics consist of the classification accuracy as well as an normalized confusion matrix, displaying the relative predictive accuracy for *farmland* vs. *not-farmland* classes. Along with this, the final accuracy, sensitivity and specificity of the classifier are presented in a table for *BigEarthNet-S2*. Since the behaviour of the training and evaluation metrics appear very similar for many of the training variations, a cohesive analysis summarizing all the variations is performed, rather than analysing each training variation separately.

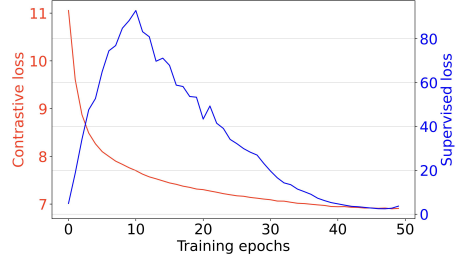
Note:

- The bar chart of top-1 accuracy per class in *TinyImageNet* is sorted in order from highest accuracy to lowest accuracy, meaning that the indices do not correspond to a common class throughout the results.
- A few of the plots begin at training epoch 10 due since training began from a checkpoint from a saved model.

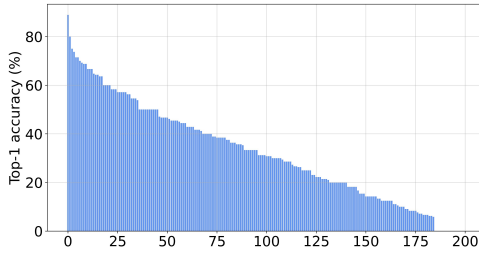
4.1 TinyImageNet Results



(a) Contrastive and supervised accuracy



(b) Contrastive and supervised loss

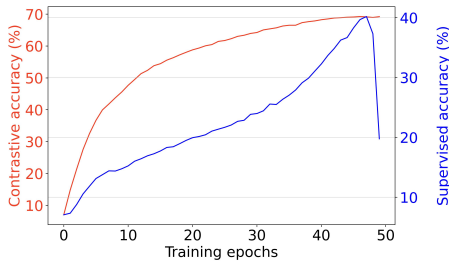


(c) Top-1 accuracies per class

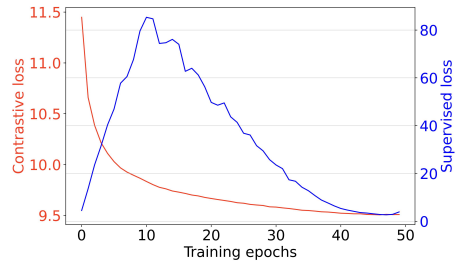
Top-1 Accuracy	32.26%
Top-5 Accuracy	58.08%

(d) Predictive metrics

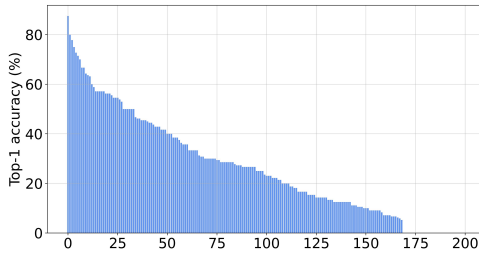
Figure 4.1: Learning with batch size $N = 256$ and temperature $\tau = 0.25$ for 50 epochs



(a) Contrastive and supervised accuracy



(b) Contrastive and supervised loss

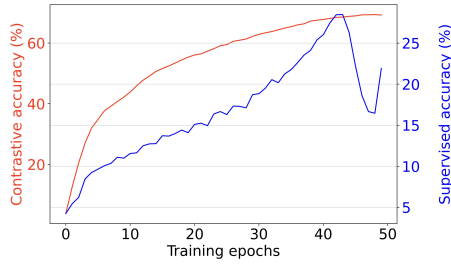


(c) Top-1 accuracies per class

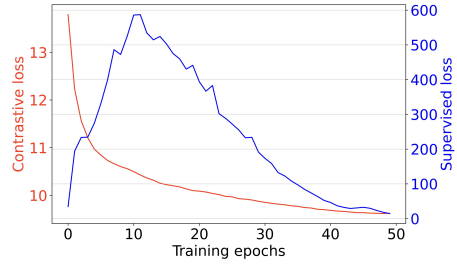
Top-1 Accuracy	26.38%
Top-5 Accuracy	52.41%

(d) Predictive metrics

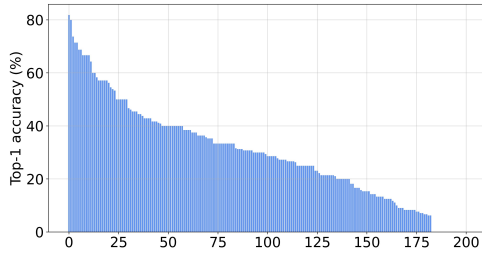
Figure 4.2: Learning with batch size $N = 256$ and temperature $\tau = 0.5$ for 50 epochs



(a) Contrastive and supervised accuracy



(b) Contrastive and supervised loss

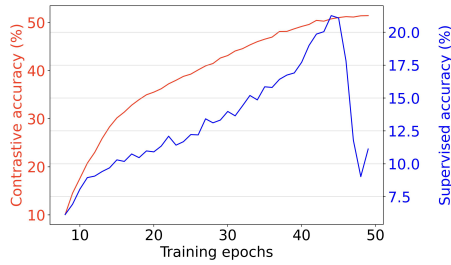


(c) Top-1 accuracies per class

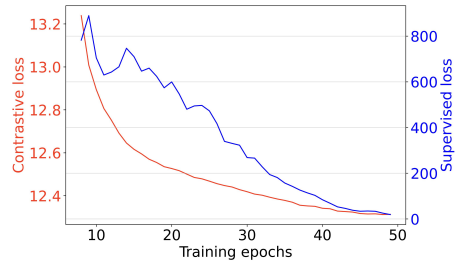
Top-1 Accuracy	29.10%
Top-5 Accuracy	54.65%

(d) Predictive metrics

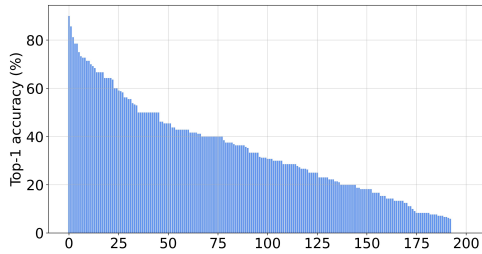
Figure 4.3: Learning with batch size $N = 1024$ and temperature $\tau = 0.25$ for 50 epochs



(a) Contrastive and supervised accuracy



(b) Contrastive and supervised loss

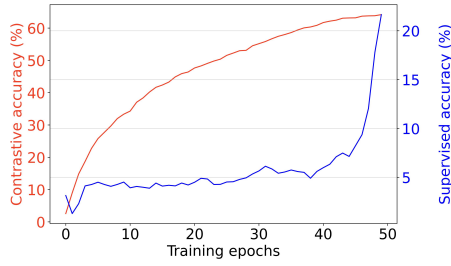


(c) Top-1 accuracies per class

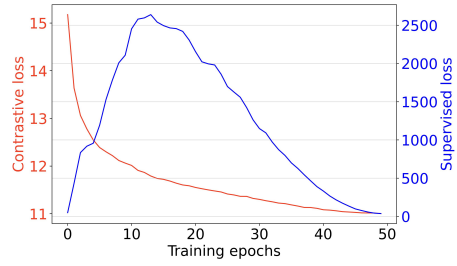
Top-1 Accuracy	18.05%
Top-5 Accuracy	39.62%

(d) Predictive metrics

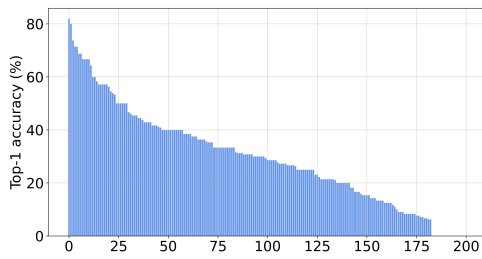
Figure 4.4: Learning with batch size $N = 1024$ and temperature $\tau = 0.5$ for 50 epochs



(a) Contrastive and supervised accuracy



(b) Contrastive and supervised loss

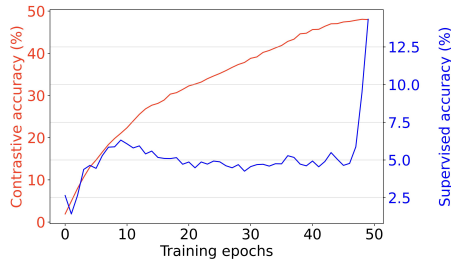


(c) Top-1 accuracies per class

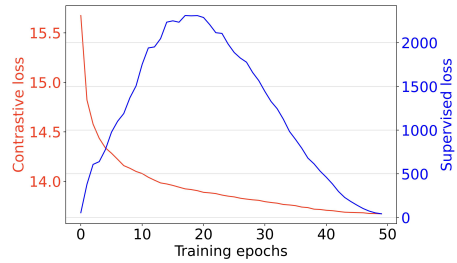
Top-1 Accuracy	26.47%
Top-5 Accuracy	51.02%

(d) Predictive metrics

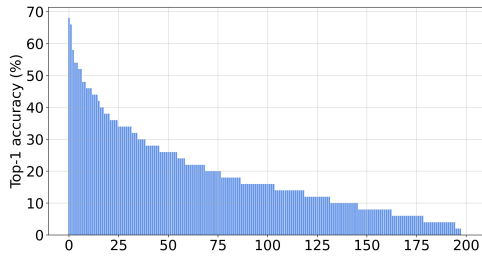
Figure 4.5: Learning with batch size $N = 2048$ and temperature $\tau = 0.25$ for 50 epochs



(a) Contrastive and supervised accuracy



(b) Contrastive and supervised loss

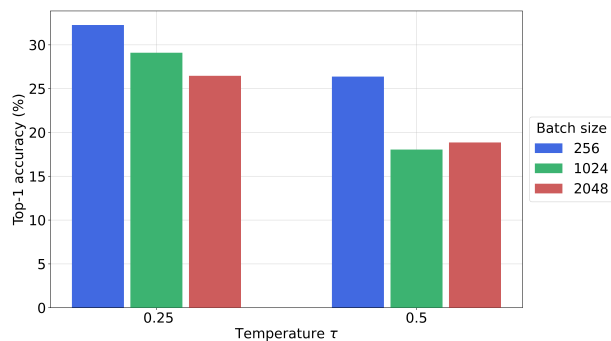


(c) Top-1 accuracies per class

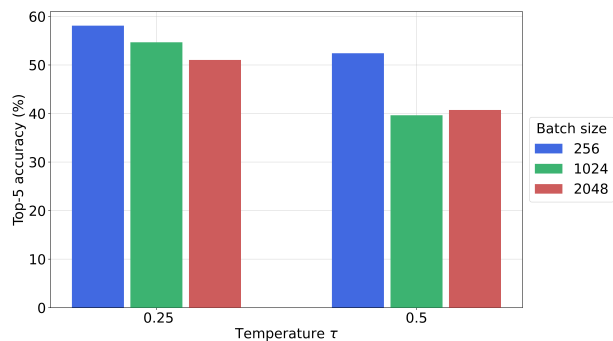
Top-1 Accuracy	18.85%
Top-5 Accuracy	40.73%

(d) Predictive metrics

Figure 4.6: Learning with batch size $N = 2048$ and temperature $\tau = 0.5$ for 50 epochs

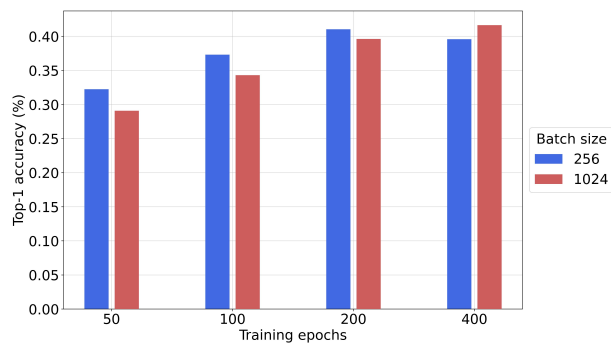


(a) Top-1 accuracy

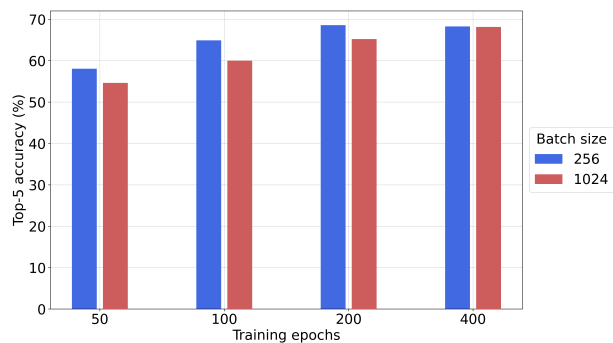


(b) Top-5 accuracy

Figure 4.7: Accuracies for different batch sizes and temperatures after 50 training epochs



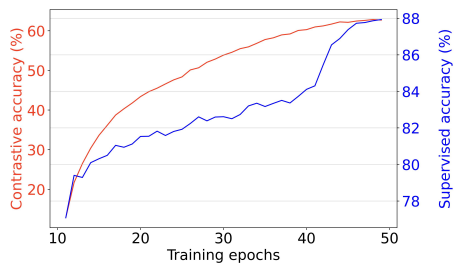
(a) Top-1



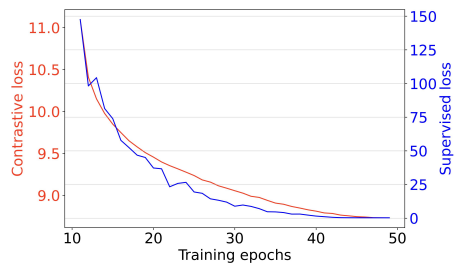
(b) Top-5

Figure 4.8: Accuracies for different number of training epochs and batch sizes with temperature $\tau = 0.25$

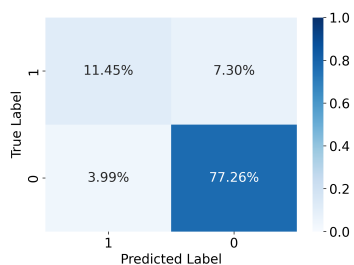
4.2 BigEarthNet Results



(a) Contrastive and supervised accuracy



(b) Contrastive and supervised loss

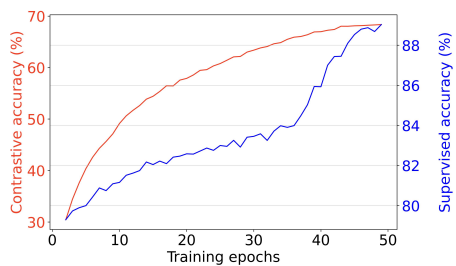


(c) Confusion matrix

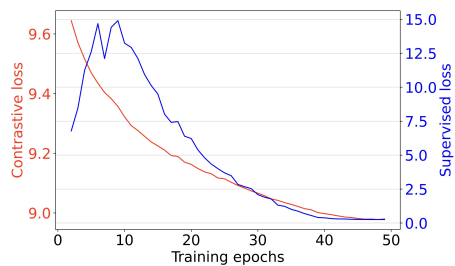
Sensitivity	61.06%
Specificity	95.09%
Accuracy	88.89%

(d) Predictive metrics

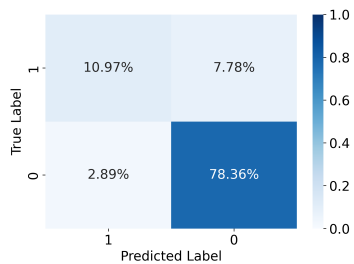
Figure 4.9: Learning with batch size $N = 256$ and temperature $\tau = 0.25$



(a) Contrastive and supervised accuracy



(b) Contrastive and supervised loss

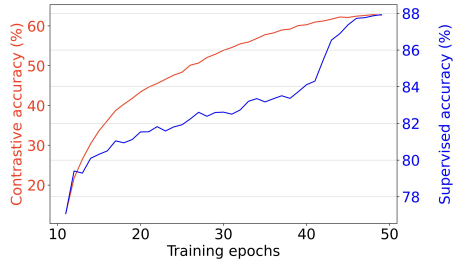


(c) Confusion matrix

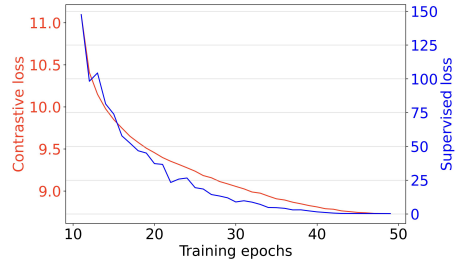
Sensitivity	58.51%
Specificity	96.44%
Accuracy	89.72%

(d) Predictive metrics

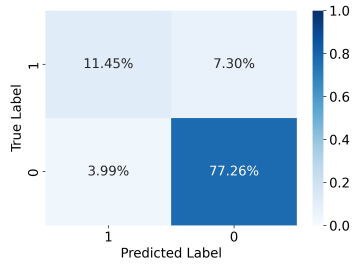
Figure 4.10: Learning with batch size $N = 256$ and temperature $\tau = 0.5$.



(a) Contrastive and supervised accuracy



(b) Contrastive and supervised loss

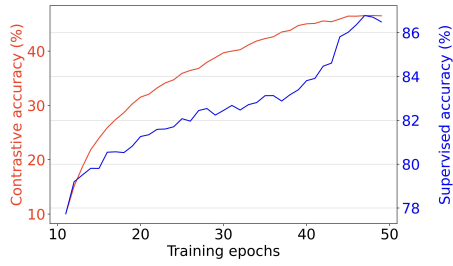


(c) Confusion matrix

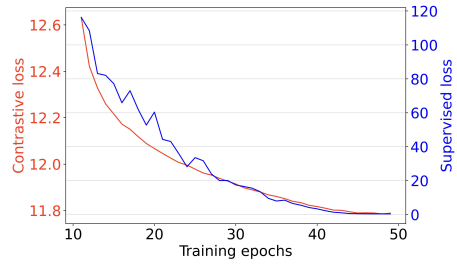
Sensitivity	61.06%
Specificity	95.09%
Accuracy	88.89%

(d) Predictive metrics

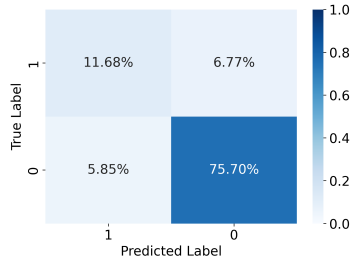
Figure 4.11: Learning with batch size $N = 1024$ and temperature $\tau = 0.25$.



(a) Contrastive and supervised accuracy



(b) Contrastive and supervised loss



(c) Normalized confusion matrix

Sensitivity	63.31%
Specificity	92.83%
Accuracy	87.91%

(d) Predictive metrics

Figure 4.12: Learning with batch size $N = 1024$ and temperature $\tau = 0.5$.

Chapter 5

Discussion

5.1 Results Discussion

TinyImageNet

For all the training varieties, the contrastive and supervised accuracies generally increase, giving a preliminary suggestion that the training is successful. The contrastive and supervised accuracies do however exhibit unique behaviours: the contrastive accuracy seemingly converges to around 50 – 70% whereas the supervised accuracy fluctuates more heavily in an upwards direction. Particularly notable in the supervised training is the large drop of training accuracy towards the end of training for batch sizes 256 and 1024, potentially linked to the inherent noise granted from the gradient estimation in smaller batch sizes [6], or from a larger learning rate enabling steps out of basins in the loss surface.

The contrastive and supervised loss functions also exhibit similar behaviour for all hyperparameters. Whereas the contrastive loss converges, the supervised loss increases substantially followed by a convergent decrease. This initial increase may be due to a too large learning rate, which is regularly scaled to an appropriate amount during the training by the *LARS* optimizer. Other dynamics in the learning rate, such as the linear warm up or weight decay may also be the cause, however further experimentation would be needed to confirm this. Another feasible explanation is the simultaneous training of the base encoder and linear classifier: the base encoder is still learning appropriate parameters for the pretext task, implying that that produced feature vectors are not yet semantically valuable. Only once the base encoder is trained *enough* can the linear classifier begin to successfully train with respect to the downstream task.

Based on the summary from Figure 4.7, it is evident that the lower batch sizes and contrastive temperatures yield better results, with batch size $N = 256$ and contrastive temperature $\tau = 0.25$ performing the best, acquiring a top-1 and top-5 accuracy of 32.26% and 58.07% respectively. This may be evidence for a finding by Nitish et al. [6], that claim smaller batch sizes tend to find flatter minima due to increased noise in gradient estimation, thereby generalizing better than larger batch sizes. The success of smaller contrastive temperatures, on the other hand, may be explained by multiplicative effect on the cosine similarity. This has the effect of amplifying the differences between similar features and enabling the model to more easily distinguish hard-negatives, i.e. features that are similar but correspond to another label [2].

In attempt to acquire higher accuracies, further training was performed for the temperature $\tau = 0.25$ and the batch sizes 256 and 1024. The results showed a steady increase of accuracy when the number of epochs of training increased. An interesting observation is that despite the batch size

$N = 256$ performing better than batch size $N = 1024$ when training for fewer epochs, batch size $N = 1024$ performs better when training for 400 epochs. This may be linked to an observation of the original SimCLR [2], namely that contrastive learning benefits more from larger batch sizes and more training steps. Upon observing the training plots however (see Appendix A.1), the contrastive accuracy is substantially higher for batch size $N = 256$ for all training-lengths, suggesting that contrastive representation learning is more successful for the smaller batch size. The resulting classification accuracies on the test split remain relatively similar for both batch sizes for training-lengths of 200 and 400 epochs. The best performing run models for top-1 and top-5 accuracy were the models with batch size/training-length 1024/400 (41.66%) and 256/200 (68.17%) respectively. Since the supervised accuracies still seems to fluctuate during training, it is presumed learning is still taking place. To determine the better hyperparameters, training should be performed longer.

BigEarthNet

As the training progresses, the contrastive and supervised accuracies increase consistently, suggesting that the model is learning successfully. The contrastive accuracy appears to converge steadily during training whereas the supervised accuracy does not seem to converge and has stronger fluctuations. The supervised accuracy is also substantially higher from the beginning of the training process, implying that the supervised binary classification task is easier than the pretext task. No clear differences are seen in the accuracies amongst the different training variations with the exception of the training with batch size $N = 1024$ and temperature $\tau = 0.5$, which displayed considerably smaller contrastive accuracy. This was unexpected as contrastive learning benefits from larger batch sizes, perhaps explained by the phenomena that larger batch sizes prefer sharper minimas in the parameter space which provide worse generalization ability [6].

Both the contrastive and supervised loss plots show convergent behaviour, indicative of successful optimization and training. Although the losses seemingly converge, longer training can most probably minimize the loss even more. No noticeable differences are seen between the different training variations in terms of loss, except the training variation with batch size $N = 256$ and temperature $\tau = 0.5$, which features an initial increase but then proceeds to decrease steadily. This was also seen in the *TinyImageNet* training, potentially a result of learning rate dynamics, simultaneous training of the base-encoder and classifier, etc.

Upon observing the model accuracy during evaluation, valued at approximately 89% for all training variations, one can be easily misled. Although the accuracy is high, the sensitivity, i.e. the probability of classifying an image as *farmland* given that it truly is *farmland*, lies at approximately 60%. The specificity on the other hand, i.e. the probability of classifying an image as *not-farmland* given that it truly is *not-farmland*, is very high, approximately valued at 95%. These results may be a result of an imbalanced dataset; only 18% of the data contains *farmland*. With relatively fewer instances of *farmland*, the model will have had less opportunity to learn the semantic representations found in *farmland*. The poor sensitivity may also be attributed to an data intrinsic issue: the instances of the classes are not semantically different enough. Although the satellite images may have contained the pre-defined *farmland* labels, they may also have contained other labels which semantically dominated the image. For example, a satellite image of an urban landscape with a small segment of farmland will still be granted as the class *farmland*.

5.2 Conclusion and Further Research

In this thesis, the learning capacity of the self-supervised *SimCLR* algorithm was investigated in the context of visual object recognition and remote sensing using the datasets *TinyImageNet* and *BigEarthNet-S2* respectively. Self-supervised contrastive learning took place successfully for both models, indicated by converging contrastive accuracy and loss, however the supervised accuracies continue to fluctuate upwards, sometimes heavily, with no sign of convergence. This suggests that the models are still learning and capable of learning more through continued training.

Despite smaller batch sizes and contrastive temperatures preferred for small training-lengths with *TinyImageNet*, no clear preference of hyperparameters were shown for longer training. The highest top-1 accuracy from all training variations models was 41.66%, obtained via training for 400 epochs with batch size 1024, contrastive temperature $\tau = 0.25$. Conducting a longer training with different hyperparameter variations would be especially interesting for *TinyImageNet*, as it shows no bottlenecks during learning.

As for *BigEarthNet-S2*: despite a high classification accuracy, poor sensitivities of approximately 60% of the models suggest very limited generalization capacity. This limited generalization capacity is not attributed to poor training or parameter tuning, but rather to a poorly chosen classification task with respect to the dataset. In this case, the semantically diverse satellite images were too complex for the binary classification task classifying farmland.

The intrinsic limitation of using *BigEarthNet-S2*, being multi-labelled, for a classification task is the semantically diverse information present in each sample. For an initial evaluation of self-supervised learning performance with satellite imagery, it would be a good idea to utilize a simple dataset containing images with non-overlapping, semantically distinct classes. This would entail that images contain strictly one label with one semantic context spanning the entire image.

If one insists to use multi-labelled dataset alternative downstream tasks should be considered. Since multi-labelled image-based datasets contain so much diverse information, one might want to explore the use of pixel-based downstream tasks such as semantic or instance segmentation.

Although studying the effect of other hyperparameters, such as learning rate, number of training epochs, and augmentation strength, could potentially bring a higher predictive accuracy for the land classification problem, it would be of higher interest to explore larger methodological changes. One feasible alternative is to fine-tune the base encoder with a niched dataset for a specialized classification task. In relation to remote sensing, one could fine-tune the pretrained network on satellite images of farmland, paired with their yields, for yield prediction using a limited dataset.

Bibliography

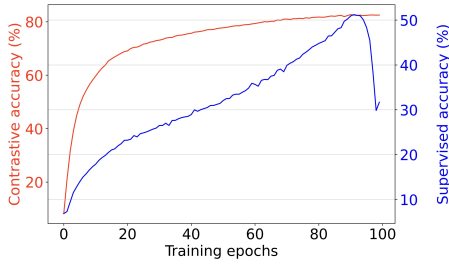
- [1] Alfredo Canziani and Yann LeCun. *NYU Deep Learning, Spring 2020*. 2020. URL: <https://atcold.github.io/pytorch-Deep-Learning/>.
- [2] Ting Chen et al. “A Simple Framework for Contrastive Learning of Visual Representations”. In: *CoRR* abs/2002.05709 (2020). arXiv: 2002.05709. URL: <https://arxiv.org/abs/2002.05709>.
- [3] Moo K. Chung. 3. *The Gaussian kernel*. 2007. URL: <https://pages.stat.wisc.edu/~mchung/teaching/MIA/reading/diffusion.gaussian.kernel.pdf.pdf>.
- [4] Vineet Gundecha. “Challenges of Large-batch Training of Deep Learning Models”. In: (2020). URL: <https://infohub.delltechnologies.com/p/challenges-of-large-batch-training-of-deep-learning-models/#:~:text=It%20has%20been%20consistently%20observed,perform%20poorly%20on%20test%20data..>
- [5] 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>.
- [6] Nitish Shirish Keskar et al. “On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima”. In: *CoRR* abs/1609.04836 (2016). arXiv: 1609.04836. URL: <http://arxiv.org/abs/1609.04836>.
- [7] Alexander Kolesnikov, Xiaohua Zhai, and Lucas Beyer. “Revisiting Self-Supervised Visual Representation Learning”. In: *CoRR* abs/1901.09005 (2019). arXiv: 1901.09005. URL: <http://arxiv.org/abs/1901.09005>.
- [8] Oscar Mañas et al. “Seasonal Contrast: Unsupervised Pre-Training From Uncurated Remote Sensing Data”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Oct. 2021, pp. 9414–9423.
- [9] Kevin P. Murphy. “Machine Learning A Probabilistic Perspective”. In: (2012).
- [10] Ashley Walker Robert Fisher Simon Perkins. *Gaussian Smoothing*. 2003. URL: <https://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm>.
- [11] Aston Zhang et al. *Dive into Deep Learning*. <https://d2l.ai>. 2020.

Appendix A

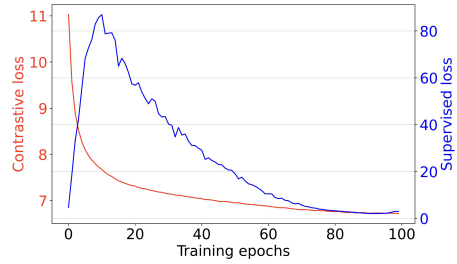
Appendix

A.1 TinyImageNet: Further Training

A.1.1 Batch size $N = 256$

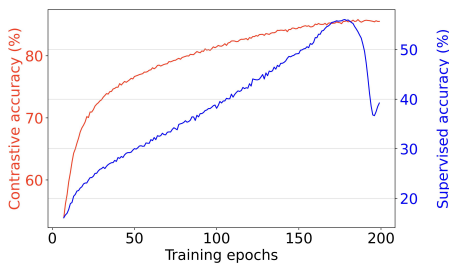


(a) Contrastive and supervised accuracy

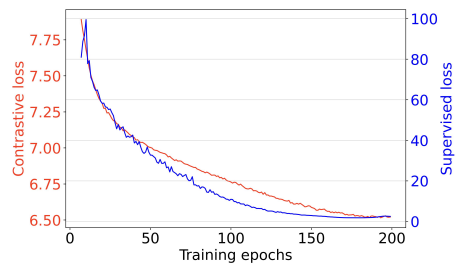


(b) Contrastive and supervised loss

Figure A.1: Learning with batch size $N = 256$ and temperature $\tau = 0.25$ for 100 epochs

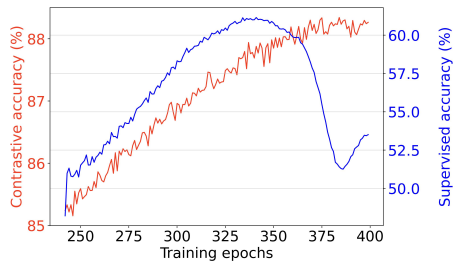


(a) Contrastive and supervised accuracy

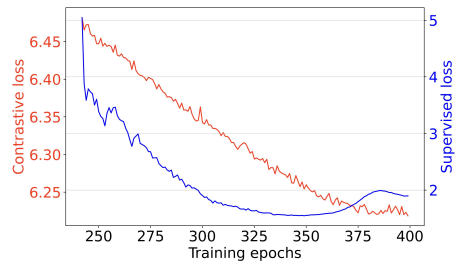


(b) Contrastive and supervised loss

Figure A.2: Learning with batch size $N = 256$ and temperature $\tau = 0.25$ for 200 epochs



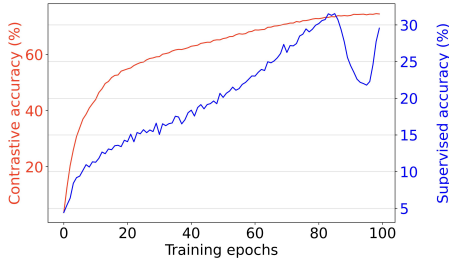
(a) Contrastive and supervised accuracy



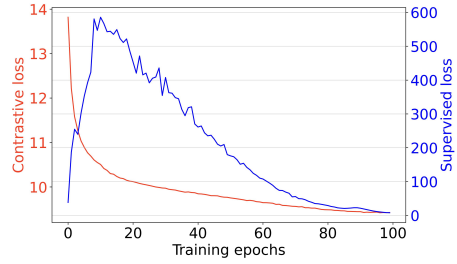
(b) Contrastive and supervised loss

Figure A.3: Learning with batch size $N = 256$ and temperature $\tau = 0.25$ for 400 epochs

A.1.2 Batch size $N = 1024$

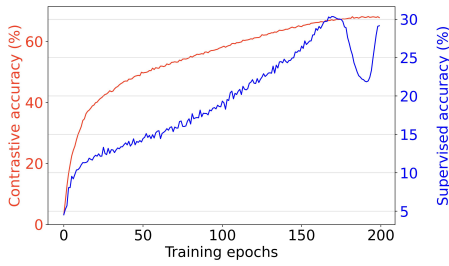


(a) Contrastive and supervised accuracy

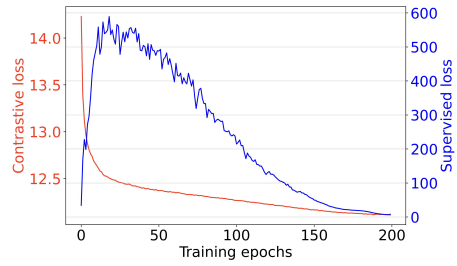


(b) Contrastive and supervised loss

Figure A.4: Learning with batch size $N = 1024$ and temperature $\tau = 0.25$ for 100 epochs

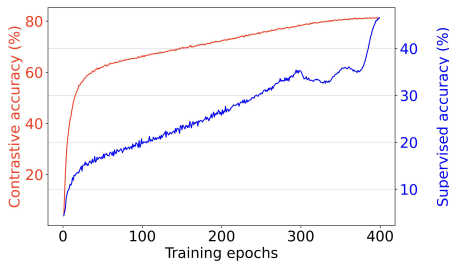


(a) Contrastive and supervised accuracy

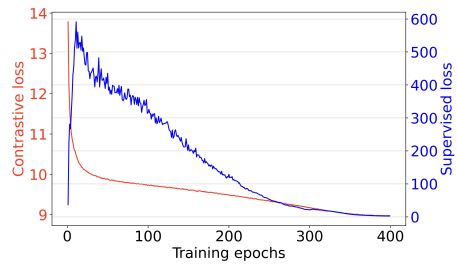


(b) Contrastive and supervised loss

Figure A.5: Learning with batch size $N = 1024$ and temperature $\tau = 0.25$ for 200 epochs



(a) Contrastive and supervised accuracy



(b) Contrastive and supervised loss

Figure A.6: Learning with batch size $N = 1024$ and temperature $\tau = 0.25$ for 400 epochs

A.1.3 Tabular Summary

<i>Batch size \ Epochs</i>	50	100	200	400
256	32.26	37.33	41.05	39.6
1024	29.1	34.32	39.62	41.66

Table A.1: Top-1 accuracy for different batch sizes and training epochs.

<i>Batch size \ Epochs</i>	50	100	200	400
256	58.08	64.93	68.62	68.29
1024	54.65	60.03	65.24	68.17

Table A.2: Top-5 accuracy for different batch sizes and training epochs.