# Self-Supervised Few Shot Learning

## Prototypical Contrastive Learning with Graphs

Ojas Kishorkumar Shirekar

# Self-Supervised Few Shot Learning

## Prototypical Contrastive Learning with Graphs

by

Ojas Kishorkumar Shirekar

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Wednesday August 31, 2022 at 14:30.


| | | |
|---|---|---|
| Student number: | 5225493 | |
| Project duration: | July 1, 2021 – July 31, 2022 | |
| Thesis committee: | Dr. ir. H. Jamali-Rad, | TU Delft and Shell, Daily supervisor |
| | Dr. ir. J. van Gemert, | TU Delft, Advisor |
| | Dr. E. Isufi, | TU Delft, External committee member |




TU Delft

# Preface

This report, along with the two scientific articles present in it, is the culmination of the work I did for my Master's thesis. First, I would like to thank my supervisor and mentor, Dr. Hadi Jamali-Rad, for the constant guidance and support. Much of this work would not have come to fruition if not for Hadi's drive and impeccable attention to detail. Not only has Hadi taught me whatever I know about the art of research, but he also imparted valuable life lessons that will always stay with me.

I would like to express my gratitude to my parents and brother. Without whose support, understanding and encouragement, I would not have the privilege of being able to pursue my interests or write this report while sitting in Delft.

Finally, I would like to thank the thesis committee chair, Dr. Jan van Gemert, for his fantastic and exciting Deep Learning and Computer Vision classes that I always looked forward to attending. I would also like to thank Dr. Elvin Isufi for their time and interest in my work, particularly during a busy, sweltering summer.

I am glad to have met some of the brightest people I know in Delft, and some have become my close friends. I cherish all my moments with them, especially the time spent in building 28. Thank you for making this journey colourful.

Working through my thesis, I have realised that my passion for computer science and research remains stronger than ever before.

This report has been structured to first show the two scientific articles containing the motivation, explanations, methods developed, and experimental results. The chapters following the articles present the fundamental concepts that have made this work a reality. The report has been designed to be as self-contained as possible.

*Ojas Kishorkumar Shirekar*
*Delft, August 2022*

# List of Publications

[1] Ojas Kishore Shirekar, Hadi Jamali-Rad, ”Self-Supervised Class-Cognizant Few-Shot Classification”, Accepted at The 29th IEEE International Conference on Image Processing (IEEE ICIP), 2022. arXiv: https://arxiv.org/abs/2202.08149, Code: https://github.com/ojss/c3lr.

[2] Ojas Kishore Shirekar, Anuj Singh, Hadi Jamali-Rad, ”Self-Attention Message Passing for Contrastive Few-Shot Learning”, submitted to IEEE/CVF Winter Conference on Applications of Computer Vision (WACV), 2023.

# Contents

# Notation

### Numbers and Arrays

| | |
|---|---|
| $\boldsymbol{A}$ | A matrix |
| $\mathrm{diag}(\boldsymbol{a})$ | A square, diagonal matrix with diagonal entries given by $\boldsymbol{a}$ |
| $\boldsymbol{I}$ | Identity matrix with dimensionality implied by context |
| $\boldsymbol{I}_n$ | Identity matrix with $n$ rows and $n$ columns |
| $\mathrm{a}$ | A scalar random variable |
| $A$ | A tensor |
| $\boldsymbol{a}$ | A vector |
| $a$ | A scalar (integer or real) |

### Linear Algebra Operations

| | |
|---|---|
| $\det(\boldsymbol{A})$ | Determinant of $\boldsymbol{A}$ |
| $\boldsymbol{A}^\top$ | Transpose of matrix $\boldsymbol{A}$ |
| $\boldsymbol{A} \odot \boldsymbol{B}$ | Element-wise (Hadamard) product of $\boldsymbol{A}$ and $\boldsymbol{B}$ |

### Indexing

| | |
|---|---|
| $A_{i,j}$ or $\boldsymbol{A}(i,j)$ | Element $i, j$ of matrix $\boldsymbol{A}$ |
| $\mathrm{a}_i$ | Element $i$ of the random vector $\mathbf{a}$ |
| $A_{i,j,k}$ | Element $(i, j, k)$ of a 3-D tensor $A$ |
| $a_i$ | Element $i$ of vector $\boldsymbol{a}$, with indexing starting at 1 |
| $a_{-i}$ | All elements of vector $\boldsymbol{a}$ except for element $i$ |
| $\boldsymbol{A}_{:,i}$ | Column $i$ of matrix $\boldsymbol{A}$ |
| $\boldsymbol{A}_{i,:}$ | Row $i$ of matrix $\boldsymbol{A}$ |
| $A_{:,:,i}$ | 2-D slice of a 3-D tensor |

### Calculus

| | |
|---|---|
| $\dfrac{\partial f}{\partial \boldsymbol{x}}$ | Jacobian matrix $\boldsymbol{J} \in \mathbb{R}^{m \times n}$ of $f : \mathbb{R}^n \to \mathbb{R}^m$ |

| | |
|---|---|
| $\frac{\partial y}{\partial x}$ | Partial derivative of $y$ with respect to $x$ |
| $\int f(\boldsymbol{x})d\boldsymbol{x}$ | Definite integral over the entire domain of $\boldsymbol{x}$ |
| $\int_{\mathbb{S}} f(\boldsymbol{x})d\boldsymbol{x}$ | Definite integral with respect to $\boldsymbol{x}$ over the set $\mathbb{S}$ |
| $\nabla_{\boldsymbol{x}} y$ | Gradient of $y$ with respect to $\boldsymbol{x}$ |
| $\nabla^2_{\boldsymbol{x}} f(\boldsymbol{x})$ or $\boldsymbol{H}(f)(\boldsymbol{x})$ | The Hessian matrix of $f$ at input point $\boldsymbol{x}$ |
| $\frac{dy}{dx}$ | Derivative of $y$ with respect to $x$ |

**Sets and Graphs**

| | |
|---|---|
| $(a, b]$ | The real interval excluding $a$ but including $b$ |
| $[a, b]$ | The real interval including $a$ and $b$ |
| $\mathcal{G}$ | A graph |
| $\mathbb{R}$ | The set of real numbers |
| $\mathbb{A}\backslash\mathbb{B}$ | Set subtraction, i.e., the set containing the elements of $\mathbb{A}$ that are not in $\mathbb{B}$ |
| $\mathbb{A}$ | A set |
| $\{0, 1, \dots, n\}$ | The set of all integers between $0$ and $n$ |
| $\{0, 1\}$ | The set containing 0 and 1 |

**Datasets and Distributions**

| | |
|---|---|
| $\boldsymbol{X}$ | The $m \times n$ matrix with input example $\boldsymbol{x}^{(i)}$ in row $\boldsymbol{X}_{i,:}$ |
| $p_{\text{data}}$ | The data generating distribution |
| $\hat{p}_{\text{data}}$ | The empirical distribution defined by the training set |
| $\boldsymbol{x}^{(i)}$ | The $i^{\text{th}}$ example (input) from a dataset |
| $y^{(i)}$ or $\boldsymbol{y}^{(i)}$ | The target associated with $\boldsymbol{x}^{(i)}$ for supervised learning |

# 1

# Introduction

In recent years, deep learning models have become larger and increasingly demand more data to perform their tasks satisfactorily. However, few-shot learning has garnered increasing interest recently because it underscores a fundamental gap between smart human adaptability and data-hungry supervised and unsupervised deep learning methods.

Today, machine learning models are capable of performing exceptionally well on a variety of tasks. Machine learning models already perform better than humans in several image related tasks such as object recognition, image super-resolution scaling, image analysis tasks in medicine and self-driving cars. Not only this, we have seen an explosive growth of large language models such as GPT-3 (Brown et al. 2020) and LaMDA (Cohen et al. 2022), that are capable of generating coherent sentences in a manner that was previously not seen.

**Figure 1.1:** As a human one can immediately figure out which object is most similar to the one bounded by a red box, but machines are quite poor at this type of abstraction and learning. Image borrowed from (Lake, Salakhutdinov, and Tenenbaum 2015).

However, even while being cognisant of these achievements, we must realise that most of the leading approaches today (and state-of-the-art approaches from a few years ago) are also one of the most data-hungry and parameter heavy approaches. To put this hunger into perspective, SEER

(Goyal et al. 2021) uses a model with $1.3$ billion parameters and trains it with SWaV (Caron, Misra, et al. 2020). However, this number is not even close to the league in which large language models (LLMs) operate. GPT-3 (Brown et al. 2020) has $175$-billion parameters and this number has only increased. Furthermore, such models require massive datasets such as ImageNet (Deng et al. 2009) for image models and datasets such as The Pile (Gao et al. 2020) for language models.

On the contrary, humans possess two crucial aspects of **conceptual knowledge**: first, humans generally require only a few examples from which they can learn natural or man-made concepts and categories, whereas machine learning models, as we discussed, require excessive amounts of examples to perform at a similar level. Second, humans learn far richer abstractions that machine learning models do not, and are capable of utilising these abstractions for a wide range of functions. A good example of this in action is when children acquire language; for instance, for many verbs in English the past tense is formed by adding "-ed" to the verb's stem (e.g. walked). Although children may not explicitly learn the past tense forms for every such word, they can still naturally regularise (and generalise) the past tense form and reuse it as required (Marcus et al. 1992). Similarly a visual example is shown in Figure 1.1, if a human is shown the object bounded int he red box they can immediately start associating it with other similar objects (shown below the object in question). A human can naturally parse major components of the object that make it what it is (a Segway[1] for those wondering). For instance, one can see the wheels and the handles which also happen to be components of a bicycle - as a human this relationship is easy to notice. Humans can also use their abstractions to generate new examples, parsing objects into their parts, and above all creating new abstractions based on existing ideas and abstractions. Machine learning models, on the other hand, struggle with generalisation and learning from fewer examples - this forms the core of the *few-shot learning problem*.

Inspired by the human ability to learn rich abstractions and relationships between objects from just a few examples, we develop two self-supervised learning methods, `C`$^3$`LR` and `SAMPTransfer`. This body of work aims to learn rich abstractions from the relationships between multiple images and uses those abstractions to tackle the few-shot learning problem. In both these works we strive to develop techniques in order to mimic the hallmark of human adaptability which is to learn quickly from a handful of examples.

[1] https://www.segway.com/

# 2

# Scientific Article 1 ($\mathrm{C}^3\mathrm{LR}$)

# SELF-SUPERVISED CLASS-COGNIZANT FEW-SHOT CLASSIFICATION

*Ojas Kishore Shirekar*[†], *Hadi Jamali-Rad*[†⋆]

[†] Faculty of EEMCS, Delft University of Technology (TU Delft), Delft, The Netherlands
[⋆] Shell Global Solutions International B.V., Amsterdam, The Netherlands

## ABSTRACT

Unsupervised learning is argued to be the dark matter of human intelligence[1]. To build in this direction, this paper focuses on unsupervised learning from an abundance of unlabeled data followed by few-shot fine-tuning on a downstream classification task. To this aim, we extend a recent study on adopting contrastive learning for self-supervised pre-training by incorporating class-level cognizance through iterative clustering and re-ranking and by expanding the contrastive optimization loss to account for it. To our knowledge, our experimentation both in standard and cross-domain scenarios demonstrate that we set a new state-of-the-art (SoTA) in (5-way, 1 and 5-shot) settings of standard mini-ImageNet benchmark as well as the (5-way, 5 and 20-shot) settings of cross-domain CDFSL benchmark. Our code and experimentation can be found in our GitHub repository: https://github.com/ojss/c3lr.

***Index Terms***— Few-shot classification, self-supervised learning, contrastive learning.

## 1. INTRODUCTION

Few-shot learning has received an upsurge of attention recently because it highlights a fundamental gap between human intelligence and data-hungry supervised deep learning methods. We humans can learn in a self-supervised fashion and/or with very little supervision. To tackle this challenge, few-shot classification is cast as the task of predicting class labels for a set of unlabeled data points (*query set*) given only a small set of labeled ones (*support set*). The query and support samples are typically drawn from the same distribution. Few-shot classification approaches are typically comprised of two sequential phases [1–4]: (i) *pre-training* on an abundant dataset (sometimes called "base"), followed by (ii) *fine-tuning* on an unseen dataset containing "novel" classes. Typically, the target classes in pre-training and fine-tuning phases are mutually exclusive. In this paper, we focus on self-supervised (also sometimes interchangeably called "unsupervised" in the literature) setting where we have no access to the class labels of the base dataset in the pre-training phase or their distribution.

The authors thank Delft University of Technology and Shell Global Solutions International B.V. for permission to publish this work.

[1] Yann LeCun's note; Meta AI blog post on self-supervised learning.

The art here is to devise a synthetic class label assignment technique and corresponding loss function in the pre-training phase to efficiently transfer the learning to the fine-tuning phase. To this aim, studies have proposed two different approaches. The first approach follows a *meta-learning* strategy to create (synthetic) "tasks" similar to the the downstream episodic training in the fine-tuning phase [5–7]. The second one follows some sort of *transfer learning* approach, where a representation learning step in the pre-training phase is followed by episodic fine-tuning [1, 8, 9]. In the latter case, typically a feature extractor (encoder) is trained using metric learning to capture the global structure of the unlabeled data. Next, a simple predictor (typically a linear layer) is adopted in conjunction with the extractor for quick adaptation to the novel classes in the fine-tuning phase. The better the feature extractor captures the global structure of the unlabeled data, the less the predictor requires training samples and the faster it adapts itself to the unseen classes in the fine-tuning phase.

Recent studies [1, 9, 10] demonstrate that the second approach based on transfer learning outperforms meta-learning based methods in cross-domain settings, where the training and novel classes come from totally different distributions. Their results also show that a properly-devised transfer learning based unsupervised approach comes pretty close to the performance of a fully supervised counterpart [1, 3], something that we will also confirm through experimentation. Most recently, a new state-of-the-art (SoTA) in self-supervised few shot classification has been set by extending the prototypical networks (ProtoNets) [11] using a *contrastive* loss [2]. This approach (called ProtoTransfer [1]) constructs a contrastive metric embedding that clusters unlabeled prototypical samples and their augmentations. Inspired by this idea, we propose class-cognizant contrastive learning ($C^3$LR, Algorithm 1) to further extend it to incorporate class-level insights from the global structure of data. This is done via an unsupervised iterative re-ranking and clustering step resulting in clusters of unlabeled embeddings followed by a modified contrastive loss now containing a term that specifically promotes this class-level global structure. Our experimentation demonstrates that $C^3$LR outperforms its predecessor ProtoTransfer in (5-way, 1 and 5-shot) settings of Ominglot [12] and mini-Imagenet [13] benchmarks by about $1\%$ and $2\%+$, respectively. The performance improvement goes up to $4.5\%$ in the cross-domain

setting of the CDFSL benchmark [14]. As a result, to our best knowledge, C$^3$LR sets a new SoTA for most challenging settings of mini-ImageNet and CDFSL benchmarks.

## 2. CLASS-COGNIZANT CONTRASTIVE LEARNING (C$^3$LR)

In this section, we first describe our problem formulation. We then discuss the two phases of the proposed approach: self-supervised pre-training and few-shot supervised fine-tuning. The mechanics of the proposed approach and a sketch of the training procedure is shown in Figure 1.

### 2.1. Preliminaries

Let us denote the training data of size $M$ as $\mathcal{D}_{\text{tr}} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{M}$ with $(\boldsymbol{x}_i, y_i)$ representing an image $\boldsymbol{x}_i$ and its class label $y_i$. In the pre-training phase, we take $L$ random samples from $\mathcal{D}_{\text{tr}}$ and augment each sample $Q$ times by drawing augmentation functions $\psi^q(.), \forall q \in [Q]$ from the set $\mathcal{A}$. This results in a batch of size $B = (Q+1)L$ total samples. Note that the data labels are unknown in the pre-training phase. In the fine-tuning phase, we deal with the so-called episodic training on a set of tasks $\mathcal{T}$ containing $N$ classes each with $K$ samples per task drawn from the test dataset $\mathcal{D}_{\text{tst}} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{M'}$ of size $M'$. From now on, we refer to this task construct as ($N$-way, $K$-shot) denoted by $(N, K)$. An episode consists of a labeled support set, $\mathcal{S}$, from which the model learns and an unlabeled query set, $\mathcal{Q}$, on which the model predicts. Note that both $\mathcal{S}$ and $\mathcal{Q}$ contain a set of tasks of the form $(N, K)$.

### 2.2. Self-Supervised Pre-Training

The fact that we do not have access to class labels calls for a self-supervised pre-training stage. As discussed earlier, we build upon the idea of employing contrastive learning for prototypical transfer learning following the footsteps of [1]. The **high-level idea** here is to not only enforce the latent embeddings of augmented images come close to that of the source image in the embedding space (the classical contrastive setting), but also enforce embeddings of the images belonging to each cluster (and their augmentations) come closer to each other, for which a preceding unsupervised cluster formation step is required. This can help enforce similar classes into separate clusters, which will in turn be used as additional information in a modified two-term contrastive loss in Algorithm 1. Let us walk you through the process in further details.

Algorithm 1 starts with **batch generation** (lines 2 to 7): each mini-batch consists of $L$ random samples $\{\boldsymbol{x}_i\}_{i=1}^{L}$ from $\mathcal{D}_{\text{tr}}$, where $\boldsymbol{x}_i$ is treated as a 1-shot support sample for which we create $Q$ randomly augmented versions $\tilde{\boldsymbol{x}}_{i,q}$ as query samples (line 5). This leads to a batch size of $B = (Q+1)L$. Then embeddings are generated by passing the samples through an encoder $f_\phi$ network. This is where the first major modification to ProtoTransfer [1] comes into play. Before the contrastive loss comes into action, we apply **re-ranking and clustering**

**Algorithm 1:** Class-Cognizant Contrastive Learning (C$^3$LR)

**Require:** $L$, $Q$, $f_\phi$, $\mathcal{A}$, $\alpha$, $d[\cdot,\cdot]$

1 **while** *not done* **do**
2 Sample minibatch $\{\boldsymbol{x}_i\}_{i=1}^{L}$
3 **forall** $i \in \{1, \ldots, L\}$ **do**
4 **forall** $q \in \{1, \ldots, Q\}$ **do**
5 $\tilde{\boldsymbol{x}}_{i,q} = \psi^q(\boldsymbol{x}_i)$; $\psi^q \sim \mathcal{A}$.
6 **end**
7 **end**
8 $\mathbf{R} = \texttt{ReRank}\Big(\Big[f_\phi\left(\{\boldsymbol{x}_i\}_{i=1}^{L}\right), f_\phi\left(\{\tilde{\boldsymbol{x}}_{i,\,q}\}_{i=1,q=1}^{L,Q}\right)\Big]\Big)$
9 $\mathcal{C} = \{\boldsymbol{C}_1, \boldsymbol{C}_2, \ldots, \boldsymbol{C}_P\} \leftarrow \texttt{HDBSCAN}(\mathbf{R})$
10 $\mathcal{M} = \{\mathbf{m}_p\}_{p=1}^{P}; \;\; \mathbf{m}_p = \frac{\sum_{x_j \in \boldsymbol{C}_p} x_j}{|\boldsymbol{C}_p|}$
11 **let** $r(i,q,p) = -\log \frac{\exp(-d[f_\phi(\tilde{\boldsymbol{x}}_{i,q}), \mathbf{m}_\mathbf{p}])}{\sum_{p=1}^{P} \exp(-d[f_\phi(\tilde{\boldsymbol{x}}_{i,q}), \mathbf{m}_\mathbf{p}])}$
12 **let** $\ell(i,q) = -\log \frac{\exp(-d[f_\phi(\tilde{\boldsymbol{x}}_{i,q}), f_\phi(\boldsymbol{x}_i)])}{\sum_{k=1}^{L} \exp(-d[f_\phi(\tilde{\boldsymbol{x}}_{i,q}), f_\phi(\boldsymbol{x}_k)])}$
13 $\mathcal{L}_1 = \frac{1}{LQ} \sum_{p=1}^{P} \sum_{i=1}^{L} \sum_{q=1}^{Q} r(i,q,p)$
14 $\mathcal{L}_2 = \frac{1}{LQ} \sum_{i=1}^{L} \sum_{q=1}^{Q} \ell(i,q)$
15 $\mathcal{L} = \mathcal{L}_1 + \mathcal{L}_2$
16 $\phi \leftarrow \phi - \alpha \nabla_\phi \mathcal{L}$
17 **end**

(lines 8 to 10) to discover class-level global structure of data and enforce similar classes into separate clusters in the embedding space. Note that this step remains to be unsupervised in that the class labels are not required. The re-ranking step (line 8) makes use of the $k$-reciprocal nearest neighbors as the distance metric between latent embeddings [15], which has been shown to outperform the Euclidean distance [3] when used for subsequent clustering. HDBSCAN clustering [16] is then applied on the re-ranked embeddings $\mathbf{R}$ and returns a set of clusters populated in $\mathcal{C}$. HDBSCAN is versatile enough to discover and create required number of clusters $P$. With clusters at hand, we are now in a position to extend the standard loss proposed in [1] to contain a **class-cognizant term** (in lines 11 and 13), with lines 12 and 14 reflecting on the classical contrastive loss of ProtoTransfer [1]. This new loss term $\mathcal{L}_1$ enables a progressive improvement in class-level cluster formation and in turn learning similar representations for cluster members, while $\mathcal{L}_2$ encourages clustering of the embeddings of the augmented query samples $\{f_\phi(\tilde{\boldsymbol{x}}_{i,q})\}$ around their prototypes $\{f_\phi(\boldsymbol{x}_i)\}$. Here, both terms use an Euclidean distance metric in the embedding space denoted by $d[\cdot,\cdot]$. Finally, the new loss $\mathcal{L} = \mathcal{L}_1 + \mathcal{L}_2$ is optimized with mini-batch stochastic gradient decent with respect to the parameters $\phi$ of the encoder networks $f_\phi$.

### 2.3. Supervised Fine-Tuning

The pre-trained encoder $f_\phi$ will be used for the downstream few-shot classification task. To this aim, following [1, 11], we

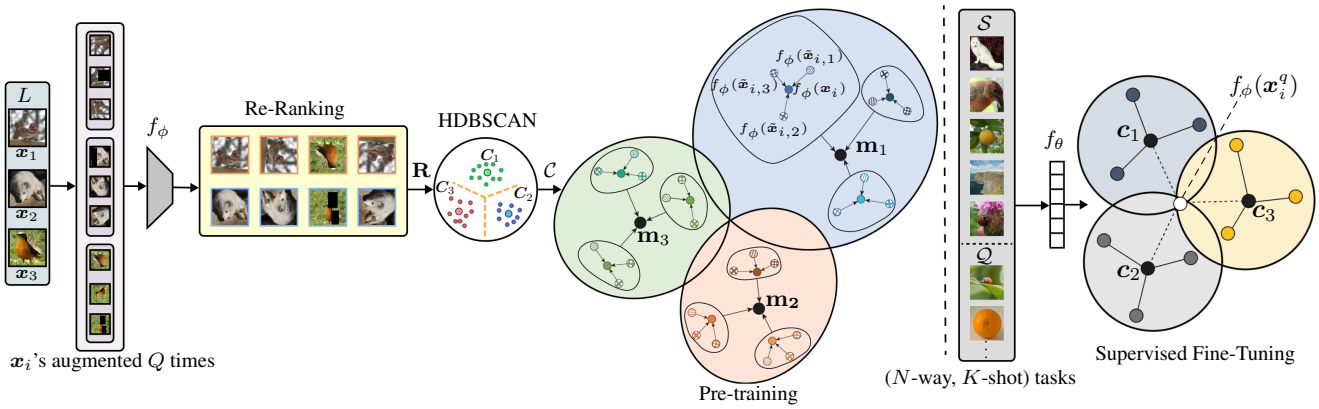



**Fig. 1**: C$^3$LR schematic view and training procedure. In the figure, $\boldsymbol{x}_i^q$ is an image sampled from the query set $\mathcal{Q}$.

concatenate $f_\phi$ with a single-layer nearest-neighbor classifier $f_\theta$ (resulting in a similar architecture as in ProtoNet [11]) and fine-tune this last layer. In this phase, we first calculate the class prototypes $\boldsymbol{c}_n$ (embeddings) for class $n$ using the encoder $f_\phi$ on the support set $\mathcal{S}_n$:

$$\boldsymbol{c}_n = \frac{1}{|\mathcal{S}_n|} \sum_{(\boldsymbol{x}_i, y_i) \in \mathcal{S}_n} f_\phi(\boldsymbol{x}_i).$$

These prototypes are then used to initialize the classifier $f_\theta$ following [1].

## 3. EXPERIMENTATION

In this section, we first discuss our experimental setup; we then present our numerical results.

### 3.1. Experimental Setup

**Datasets.** We conduct several in-domain experiments to benchmark C$^3$LR. For this purpose, we make use of commonly adopted datasets Omniglot [12] and mini-Imagenet [13] to compare against unsupervised few-shot learning approaches. Omniglot contains 1623 different handwritten characters borrowed from 50 unique alphabets out of which we use 1028 characters for training, 172 for validation and 423 for testing. We resize the grayscale images to $28 \times 28$ pixels. Mini-ImageNet contains 100 classes with 600 samples in each class amounting to a total of $60,000$ images that we resize to $84 \times 84$ pixels. Out of the 100 classes, we use 64 classes for training, 16 for validation and 20 for testing. For both datasets, the settings are the most commonly adopted ones in literature [1, 3, 7, 13]. The augmentations (in $\mathcal{A}$) used for the experimentations follow [1]. We also compare our method on a more challenging cross-domain few-shot learning (CDFSL) benchmark [14]. This benchmark consists of four datasets with increasing similarities to mini-ImageNet. In that order, we have grayscale chest X-ray images from ChestX [17], dermatological skin lesion images from ISIC2018 [18], satellite aerial images from EuroSAT [19], and crop disease images from CropDiseases [20]. We also use Caltech-UCSD Birds (CUB) dataset [21] for further analysis of cross-domain performance. CUB is composed of $11,788$ images from 200 unique bird species. We use 100 images for training, 50 for validation and 50 for test.

**Training.** The Conv4 model [13] is pre-trained on the respective training splits of the datasets, with an initial learning rate of 0.001, multiplied by 0.5 every $25,000$ steps via the Adam optimizer [22]. Based on the derivations in [11] and similar usage in [1], we initialize the classification layer $f_\theta$ with weights set to $\mathbf{W}_n = 2\boldsymbol{c}_n$ and biases set to $b_n = -\|\boldsymbol{c}_n\|^2$. For validation, we create 15 ($N$-way, $K$-shot) tasks using the validation split from which the corresponding validation accuracy and loss are calculated. Experiments involving CDFSL benchmark follow [1, 14], where we pre-train a ResNet10 encoder using C$^3$LR on mini-ImageNet images of size $224 \times 224$ for 400 epochs with the Adam optimizer and a constant learning rate of 0.001.

**Evaluation scenarios and baseline.** Our testing scheme uses 600 test episodes on which the pre-trained encoder (using C$^3$LR) is fine-tuned and tested. All our results indicate 95% confidence intervals over 3 runs each with 600 test episodes. The standard deviation values are thus calculated according to the 3 runs to provide more solid measures for comparison. For our in-domain benchmarks, we test on (5-way, 1-shot) and (5-way, 5-shot) classification tasks. While our cross-domain testing is done using (5-way, 5-shot) and (5-way, 20-shot) classification tasks. We compare our performance with a suit of recent self-supervised few-shot baselines such as ProtoTransfer [1], UFLST [3], LASIUM [23] and CACTUS [6], to name a few. Furthermore, we also compare with a set of supervised approaches (such as MAML [5], ProtoNet [11] , etc.) the best performing of which are obviously expected to outperform ours as well as other self-supervised methodologies.

### 3.2. Performance Evaluation

**In-domain evaluation.** Table 1 summarizes our performance evaluation results on Omniglot and mini-ImageNet datasets for ($N$-way, $K$-shot) scenarios with $N = 5$ and $K = 1, 5$. The top

**Table 3**: Accuracy (%± std.) of ($N$-way, $K$-shot) classification on the CDFSL benchmark. Style: **best** and second best.

| **Method**$(N, K)$ | (5,5) | (5,20) | (5,5) | (5,20) | (5,5) | (5,20) | (5,5) | (5,20) |
|---|---|---|---|---|---|---|---|---|
| | **ChestX** | | **ISIC** | | **EuroSAT** | | **CropDiseases** | |
| UMTRA-ProtoNet [1] | 24.94 ± 0.43 | 28.04 ± 0.44 | 39.21 ± 0.53 | 44.62 ± 0.49 | 74.91 ± 0.72 | 80.42 ± 0.66 | 79.81 ± 0.65 | 86.84 ± 0.50 |
| UMTRA-ProtoTune [1] | 25.00 ± 0.43 | 30.41 ± 0.44 | 38.47 ± 0.55 | 51.60 ± 0.54 | 68.11 ± 0.70 | 81.56 ± 0.54 | 82.67 ± 0.60 | 92.04 ± 0.43 |
| ProtoTransfer [1] | **26.71** ± 0.46 | **33.82** ± 0.48 | 45.19 ± 0.56 | 59.07 ± 0.55 | 75.62 ± 0.67 | 86.80 ± 0.42 | 86.53 ± 0.56 | 95.06 ± 0.32 |
| C$^3$LR (**ours**) | 26.00 ± 0.41 | 33.39 ± 0.47 | **45.93** ± 0.54 | **59.95** ± 0.53 | **80.32** ± 0.65 | **88.09** ± 0.45 | **87.90** ± 0.55 | **95.38** ± 0.31 |
| ProtoNet [14] (sup.) | 24.05 ± 1.01 | 28.21 ± 1.15 | 39.57 ± 0.57 | 49.50 ± 0.55 | 73.29 ± 0.71 | 82.27 ± 0.57 | 79.72 ± 0.67 | 88.15 ± 0.51 |
| Pre+Mean-Cent. [14] (sup.) | 26.31 ± 0.42 | 30.41 ± 0.46 | 47.16 ± 0.54 | 56.40 ± 0.53 | 82.21 ± 0.49 | 87.62 ± 0.34 | 87.61 ± 0.47 | 93.87 ± 0.68 |
| Pre+Linear [14] (sup.) | 25.97 ± 0.41 | 31.32 ± 0.45 | 48.11 ± 0.64 | 59.31 ± 0.48 | 79.08 ± 0.61 | 87.64 ± 0.47 | 89.25 ± 0.51 | 95.51 ± 0.31 |

**Table 1**: Accuracy (%± std.) for ($N$-way, $K$-shot) classification tasks. Style: **best** and second best.

| | **Omniglot** | | **mini-ImageNet** | |
|---|---|---|---|---|
| **Method**$(N, K)$ | (5,1) | (5,5) | (5,1) | (5,5) |
| CACTUs-MAML [6] | 68.84 ± 0.80 | 87.78 ± 0.50 | 39.90 ± 0.74 | 53.97 ± 0.70 |
| CACTUs-ProtoNet [6] | 68.12 ± 0.84 | 83.58 ± 0.61 | 39.18 ± 0.71 | 53.36 ± 0.70 |
| UMTRA [7] | 83.80 | 95.43 | 39.93 | 50.73 |
| AAL-ProtoNet [24] | 84.66 ± 0.70 | 89.14 ± 0.27 | 37.67 ± 0.39 | 40.29 ± 0.68 |
| AAL-MAML++ [24] | 88.40 ± 0.75 | 97.96 ± 0.32 | 34.57 ± 0.74 | 49.18± 0.47 |
| UFLST [3] | **97.03** | **99.19** | 33.77 ± 0.70 | 45.03 ± 0.73 |
| ULDA-ProtoNet [25] | - | - | 40.63 ± 0.61 | 55.41 ± 0.57 |
| ULDA-MetaOptNet [25] | - | - | 40.71 ± 0.62 | 54.49 ± 0.58 |
| U-SoSN+ ArL [26] | - | - | 41.13 ± 0.84 | 55.39 ± 0.79 |
| LASIUM [23] | 83.26 ± 0.55 | 95.29 ± 0.22 | 40.19 ± 0.58 | 54.56 ± 0.55 |
| ProtoTransfer $_{(L=50)}$ [1] | 88.00 ± 0.64 | 96.48 ± 0.26 | 45.67 ± 0.79 | 62.99 ± 0.75 |
| ProtoTransfer $_{(L=200)}$ | 88.37 ± 0.74 | 96.54 ± 0.41 | 44.17 ± 1.08 | 61.07 ± 0.82 |
| C$^3$LR (**ours**) | 89.30 ± 0.64 | 97.38 ± 0.23 | **47.92** ± 1.2 | **64.81** ± 1.15 |
| MAML [5] (supervised) | 94.46 ± 0.35 | 98.83 ± 0.12 | 46.81 ± 0.77 | 62.13± 0.72 |
| ProtoNet [11] (supervised) | 97.70± 0.29 | 99.28 ± 0.10 | 46.44± 0.78 | 66.33± 0.68 |
| MMC [27] (supervised) | 97.68± 0.07 | - | 50.41 ± 0.31 | 64.39 ± 0.24 |
| FEAT [4] (supervised) | - | - | 55.15 | 71.61 |
| Pre+Linear [1] (supervised) | 94.30 ± 0.43 | 99.08 ± 0.10 | 43.87 ± 0.69 | 63.01 ± 0.71 |

**Table 2**: Accuracy (%± std.) for ($N$-way, $K$-shot) classification on mini-ImageNet with pre-training on CUB.

| **Training** | **Testing** | (5,1) | (5,5) |
|---|---|---|---|
| ProtoTransfer ($L = 50$) [1] | ProtoTune [1] | 35.37 ± 0.63 | 52.38 ± 0.66 |
| ProtoTransfer ($L = 200$) | ProtoTune | 34.67 ± 0.84 | 51.45 ± 0.72 |
| C$^3$LR (**ours**) | ProtoTune | **39.61** ± 1.11 | **55.53** ± 1.42 |

section compares the performance of the proposed approach (C$^3$LR) with the most recent relevant self-supervised competitors. As can be seen, for Omniglot, we outperform ProtoTransfer [1] (which we build on) by about $1\%$ in both $K = 1, 5$ shot scenarios. We score the second overall best in (5-way, 1-shot) falling behind UFLST [3]. For the mini-ImageNet benchmark, to our knowledge, we set a new SoTA outperforming ProtoTransfer by $2\%+$. Interestingly, our performance beats some of the supervised baselines (bottom section of the table) adopting similar encoder architecture Conv4 for mini-ImageNet and comes close to $K = 5$-shot performances on Omniglot. Obviously, the SoTA supervised few-shot learning approaches have the advantage of having access to the all the labels, as such due to the supervision signal, are expected to outperform the unsupervised approaches like ours.

**Cross-domain evaluation.** So far we have demonstrated that the proposed approach excels for in-domain scenarios. The next step is to assess the performance under more challenging cross-domain scenarios (Table 2 and Table 3) where we pre-train on a certain dataset in an unsupervised fashion, then fine-tune and test on a different dataset. Table 2 illustrates the results of a Conv4 encoder trained on CUB and tested on tasks derived from mini-ImageNet. Here again C$^3$LR shows a clear improvement of $3\%+$ compared to ProtoTransfer (with pre-training sample sizes $L = 50, 200$). The important message here is that the proposed approach enhances ProtoTransfer in generalizing to truly unseen data. To further investigate the performance on cross-domain scenarios, we next focus on CDFSL benchmark [14] containing several datasets. Here, we pre-train on mini-ImageNet and fine-tune and test on ChestX [17], ISIC2018 [18], EuroSAT [19], and CropDiseases [20]. We compare the performance against ProtoTransfer and two of its variants with UMTRA [7] as pre-training strategy (all proposed in [1]). We also compare with a couple of closely related supervised approaches from [14], for the sake of reference. As can be seen, except for ChestX where we marginally come short of ProtoTransfer, for the other three datasets we outperform the second best competitor (ProtoTransfer) by about $0.5\%+$ to $4.5\%+$ with the most significant improvement in the case of EuroSAT. Interestingly, once again the performance of C$^3$LR is not far off that of the related supervised approaches (bottom of the table) even sometimes outperforming the supervised approaches especially in (5-way, 20-shot) scenarios.

## 4. CONCLUDING REMARKS

Inspired by the idea of using contrastive learning for unsupervised few-shot classification, we build upon the recently proposed idea of ProtoTransfer [1] by incorporating class cognizance through: (i) an unsupervised iterative re-ranking and clustering step, followed by (ii) an adjusted optimization loss formulation. We demonstrate that our proposed approach (C$^3$LR) offers considerable performance improvement above its predecessor ProtoTransfer in both in/cross-domain few-shot classification scenarios setting a new SoTA in mini-ImageNet and CDFSL benchmarks.

## 5. REFERENCES


[1] Carlos Medina, Arnout Devos, and Matthias Grossglauser, "Self-supervised prototypical transfer learning for few-shot classification," *arXiv preprint arXiv:2006.11325*, 2020.

[2] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton, "A simple framework for contrastive learning of visual representations," in *International conference on machine learning*. PMLR, 2020, pp. 1597–1607.

[3] Zilong Ji, Xiaolong Zou, Tiejun Huang, and Si Wu, "Unsupervised few-shot learning via self-supervised training," *arXiv preprint arXiv:1912.12178*, 2019.

[4] Han-Jia Ye, Hexiang Hu, De-Chuan Zhan, and Fei Sha, "Few-shot learning via embedding adaptation with set-to-set functions," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 8808–8817.

[5] Chelsea Finn, Pieter Abbeel, and Sergey Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International conference on machine learning*. PMLR, 2017, pp. 1126–1135.

[6] Kyle Hsu, Sergey Levine, and Chelsea Finn, "Unsupervised learning via meta-learning," *arXiv preprint arXiv:1810.02334*, 2018.

[7] Siavash Khodadadeh, Ladislau Boloni, and Mubarak Shah, "Unsupervised meta-learning for few-shot image classification," *Advances in neural information processing systems*, vol. 32, 2019.

[8] Yonglong Tian, Yue Wang, Dilip Krishnan, Joshua B Tenenbaum, and Phillip Isola, "Rethinking few-shot image classification: a good embedding is all you need?," in *European Conference on Computer Vision*. Springer, 2020, pp. 266–282.

[9] Guneet S Dhillon, Pratik Chaudhari, Avinash Ravichandran, and Stefano Soatto, "A baseline for few-shot image classification," *arXiv preprint arXiv:1909.02729*, 2019.

[10] Da Chen, Yuefeng Chen, Yuhong Li, Feng Mao, Yuan He, and Hui Xue, "Self-supervised learning for few-shot image classification," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 1745–1749.

[11] Jake Snell, Kevin Swersky, and Richard Zemel, "Prototypical networks for few-shot learning," *Advances in neural information processing systems*, vol. 30, 2017.

[12] Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum, "Human-level concept learning through probabilistic program induction," *Science*, vol. 350, no. 6266, pp. 1332–1338, 12 2015.

[13] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al., "Matching networks for one shot learning," *Advances in neural information processing systems*, vol. 29, 2016.

[14] Yunhui Guo, Noel CF Codella, Leonid Karlinsky, John R Smith, Tajana Rosing, and Rogerio Feris, "A New Benchmark for Evaluation of Cross-Domain Few-Shot Learning," *arXiv preprint arXiv:1912.07200*, 2019.

[15] Zhun Zhong, Liang Zheng, Donglin Cao, and Shaozi Li, "Re-ranking person re-identification with k-reciprocal encoding," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1318–1327.

[16] Leland McInnes, John Healy, and Steve Astels, "hdbscan: Hierarchical density based clustering.," *J. Open Source Softw.*, vol. 2, no. 11, pp. 205, 2017.

[17] Xiaosong Wang, Yifan Peng, Le Lu, Zhiyong Lu, Mohammadhadi Bagheri, and Ronald M Summers, "Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2097–2106.

[18] Noel Codella, Veronica Rotemberg, Philipp Tschandl, M Emre Celebi, Stephen Dusza, David Gutman, Brian Helba, Aadi Kalloo, Konstantinos Liopyris, Michael Marchetti, et al., "Skin lesion analysis toward melanoma detection 2018: A challenge hosted by the international skin imaging collaboration (isic)," *arXiv preprint arXiv:1902.03368*, 2019.

[19] Patrick Helber, Benjamin Bischke, Andreas Dengel, and Damian Borth, "Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification," 2017.

[20] Sharada P Mohanty, David P Hughes, and Marcel Salathé, "Using Deep Learning for Image-Based Plant Disease Detection," *Frontiers in Plant Science*, vol. 7, pp. 1419, 2016.

[21] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie, "The Caltech-UCSD Birds-200-2011 Dataset," Tech. Rep. CNS-TR-2011-001, California Institute of Technology, 2011.

[22] Diederik P Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[23] Siavash Khodadadeh, Sharare Zehtabian, Saeed Vahidian, Weijia Wang, Bill Lin, and Ladislau Bölöni, "Unsupervised meta-learning through latent-space interpolation in generative models," *arXiv preprint arXiv:2006.10236*, 2020.

[24] Antreas Antoniou and Amos Storkey, "Assume, augment and learn: Unsupervised few-shot meta-learning via random labels and data augmentation," *arXiv preprint arXiv:1902.09884*, 2019.

[25] Tiexin Qin, Wenbin Li, Yinghuan Shi, and Yang Gao, "Diversity helps: Unsupervised few-shot learning via distribution shift-based data augmentation," *arXiv preprint arXiv:2004.05805*, 2020.

[26] Hongguang Zhang, Piotr Koniusz, Songlei Jian, Hongdong Li, and Philip HS Torr, "Rethinking class relations: Absolute-relative supervised and unsupervised few-shot learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 9432–9441.

[27] Mengye Ren, Eleni Triantafillou, Sachin Ravi, Jake Snell, Kevin Swersky, Joshua B Tenenbaum, Hugo Larochelle, and Richard S Zemel, "Meta-learning for semi-supervised few-shot classification," *arXiv preprint arXiv:1803.00676*, 2018.

# 3

# Scientific Article 2 (SAMPTransfer)

# Self-Attention Message Passing for Contrastive Few-Shot Learning

Ojas Kishorkumar Shirekar[1], Anuj Singh[1], Hadi Jamali-Rad[1,2]
[1]TU Delft Delft, The Netherlands
[2]Shell Global Solutions International B.V., Amsterdam, The Netherlands
{o.k.shirekar, a.r.singh}@student.tudelft.nl, h.jamalirad@tudelft.nl

## Abstract

*A primary trait of humans is the ability to learn rich representations and relationships between entities from just a handful of examples without much guidance. Unsupervised few-shot learning is an undertaking aimed at reducing this fundamental gap between smart human adaptability and machines. We present a contrastive learning scheme for unsupervised few-shot classification, where we supplement a convolutional network's strong inductive prior with a self-attention based message passing neural network to exploit intra-batch relations between images. We also show that an optimal-transport (OT) based task-awareness algorithm generates task-representative prototypes that lead to more accurate classification and aid in elevating the robustness of pre-trained models. We show that our approach (`SAMPTransfer`) offers appreciable performance improvements over its competitors in both in/cross-domain few shot classification scenarios, setting new standards in the miniImagenet, tieredImagenet and CDFSL benchmarks.*

## 1. Introduction

In recent years, deep learning models have become larger and demand massive amounts of training data to perform their tasks satisfactorily. Meanwhile, few-shot learning has garnered increasing interest recently because it underscores a fundamental gap between smart human adaptability and data-hungry supervised and unsupervised deep learning methods. To address this gap, few-shot classification is cast as a task to predict class labels for a set of unlabeled data points (*query set*) given only a small set of labeled data points (*support set*). Typically, the query and support data points are drawn from the same distribution.

Few-shot classification methods usually consist of two sequential phases: (i) *pre-training* on a large dataset of "base" classes, regardless of the training being supervised or unsupervised. This is followed by (ii) *fine-tuning* on an unseen dataset consisting of "novel" classes. Normally, the classes used in the pre-training and fine-tuning are mutually exclusive. In this paper, our focus is on the self-supervised setting (also sometimes interchangeably called "unsupervised" in the literature) where we have no access to the actual class labels of the "base" dataset.

To this end, various methods have been proposed and are broadly classified into two different approaches. The first approach relies on the use of *meta-learning* and episodic training that involves the creation of synthetic "tasks" to mimic the subsequent episodic fine-tuning phase [1, 18, 25–27, 31, 56]. The second method follows a *transfer learning* approach, where the network is trained non-episodically to learn optimal representations in the pre-training phase, and is then followed by an episodic fine-tuning phase [16, 34, 43]. In this method, a feature extractor is trained to capture the structure of unlabeled data using a form of representation learning [6, 8, 34]. Next, a prediction layer (conventionally a linear layer) is utilized in conjunction with the pre-trained feature extractor for quick adaptation to the novel classes in the fine-tuning phase. The better the feature extractor models the distribution of the unlabeled data, the less the predictor requires training samples, and the faster it adapts itself to the unseen classes in the fine-tuning phase (also the testing phase).

Furthermore, supervised approaches that follow the episodic training paradigm may include a certain degree of *task awareness*. Such approaches exploit the information available in the query set during the training and testing phases [3, 11, 57] to alleviate the model's sample bias. As a result, the model learns to generate task-specific embeddings by better aligning the features of the support and query samples for optimal distance metric based label assignment. We also see a set of supervised approaches that do not rely purely on convolutional feature extractors. Instead, these approaches also use graph neural networks (GNN) [28, 39, 55, 58] to model instance-level and class-level relationships. Additionally, GNN's are equipped with the ability to exploit the manifold structure of the novel class space and then propagate labels from the support embeddings to the unlabeled query embeddings [53]. However, the majority of graph-based methods have eluded the unsupervised setting.

Several recent studies have questioned the necessity of

meta-learning for few-shot classification [4, 7, 16, 34, 41, 43, 62]. They report competitive performance on few-shot benchmarks without episodic training or few-shot task-based experiences during training. These methods follow the second approach and aim to solve the few-shot learning problem by fine-tuning a pre-trained feature extractor with a standard cross-entropy loss. Some of these methods [13, 34, 43] demonstrate that the transfer learning approach outperforms meta-learning based methods in standard in-domain and cross-domain settings, where training and novel classes come from totally different distributions. Similarly, our approach `SAMPTransfer` performs self-supervised pre-training on an unlabeled training domain and can transfer to a few-shot target domain task.

Many of the current unsupervised methods use a form of *contrastive learning* [8] in their self-supervised pre-training phase. Contrastive learning methods typically treat each image in a batch as its own class. The only other images that share the class are the augmentations of the image in question. Such methods enforce similarity of representations between pairs of an image and its augmentations (positive pairs), while enforcing dissimilarity between all other pairs of images (negative pairs) through a *contrastive loss*. Although these methods work well, they overlook the possibility that within a randomly sampled batch of images, there could be several images (apart from their augmentations) that in reality belong to the same class. By applying the contrastive loss, the network may inadvertently learn different representations for such images and classes. To avoid this problem, recent methods such as SimCLR use large batch sizes to maximize the number of negative samples [8]. However, there is still a failure to look beyond single instances to learn the representations of images. To overcome this pitfall, we propose `SAMP-CLR` in the pre-training phase of `SAMPTransfer`, that uses a form of *graph attention* coupled with traditional feature extractors. This design allows us to utilize the contrastive training scheme and simultaneously learn refined representations by looking beyond single-image instances in a task. In the few-shot downstream tasks, using graph attention also allows for a degree of task awareness to be induced in the feature extractor by using information from both support and query samples to better align feature representations. We propose an *optimal transport* based algorithm during the fine-tuning phase that aligns the distributions of the support and query samples to improve downstream adaptability of the pre-trained encoder, in a way that requires no additional parameters. Our **contributions** can be summarized as: *(i)* we propose a graph-based contrastive learning approach that helps learn representations while looking beyond single instances, *(ii)* we propose an optimal transport based fine-tuning phase that enhances feature refinement, which in turn helps in stabilizing and improving test time performance without the need for additional trainable parameters, *(iii)* we show that our unsupervised method outperforms its competitors on *mini*-ImageNet and *tiered*ImageNet (up to $7\%$ and $5\%$), while remaining competitive on the CDFSL-benchmark [22], *(iv)* we present a detailed analysis supporting our claims and demonstrating the robustness of our method.

## 2. Related Work

**Self-Supervised learning**. Self-supervised learning (SSL) is an umbrella term for a set of unsupervised methods that obtain supervisory signals from within the data itself, more often than not by leveraging the underlying structure in the data. The general technique of self-supervised learning is to predict any unobserved (or property) of the input from any observed part. Several recent advances in the SSL space have made waves by eclipsing their fully supervised counterparts [20]. Some examples of seminal works include SimCLR [8], BYOL [21], SWaV [6], MoCo [23], and SimSiam [9]. Our pre-training method `SAMP-CLR` is inspired by SimCLR and ProtoTransfer [34].

**Metric learning**. Metric learning aims to learn a representation function that maps the data into an embedding space. The distance between objects in the embedding space must preserve their similarity (or dissimilarity) - similar objects are closer, while dissimilar objects are farther. For example, unsupervised methods based on some form of contrastive loss, such as SimCLR [8] or NNCLR [17], guide objects belonging to the same potential class to be mapped to the same point and those from different classes to be mapped to different points. This process generally involves taking two crops of the same image and encouraging the network to emit an identical representation for the two while ensuring that the representations remain different from all other images in a given batch. Note that in an unsupervised setting, each image in a batch is its own class. Metric learning methods have been shown to work quite well for few-shot learning. AAL-ProtoNets [1], ProtoTransfer [34], UMTRA [27], and certain types of graph neural networks [39] are excellent examples that use metric learning for few-shot learning.

**Graph Neural Networks for FSL**. Since the first use of graphs for FSL in [39], there have been several advancements and continued interest in using graphs for supervised FSL. In [39], each node corresponds to one instance (labeled or unlabeled) and is represented as the concatenation of a feature embedding and a label embedding. The final layer of their model is a linear classifier layer that directly outputs the prediction scores for each unlabeled node. There has also been an increase in methods that use transduction. TPN [33] is one of those methods that uses graphs to propagate labels from labeled samples to unlabeled samples. Although methods such as EGNN [28] make use of edge and node features, earlier methods focused only on using node features. Graphs are attractive as they can model intra-batch

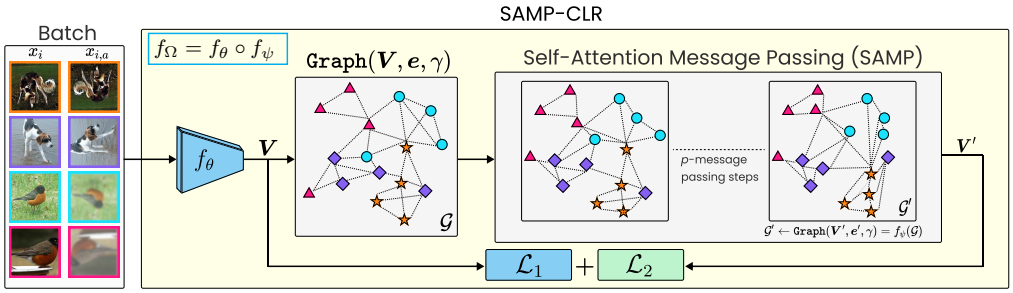



Figure 1: SAMP-CLR schematic view and pre-training procedure. In the figure, $x_{i,a}$ is an image sampled from the augmented set $A$. The $p$-message passing steps refine the features extracted using a CNN encoder.

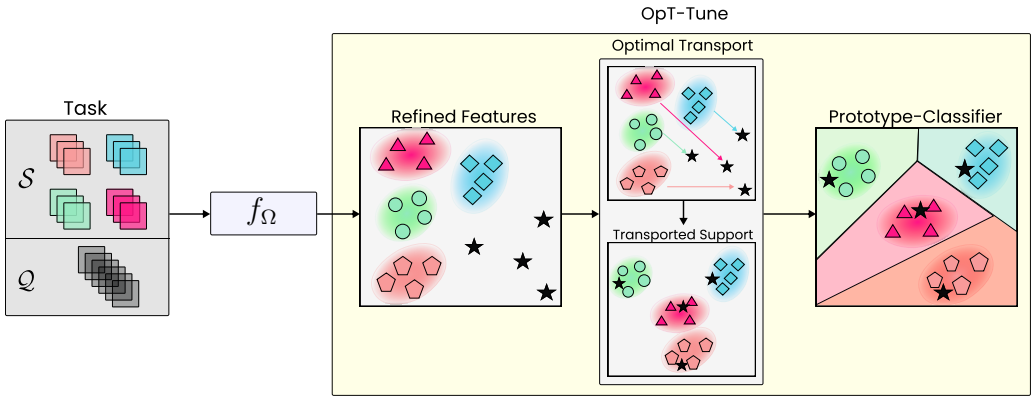



Figure 2: Features extracted from the pre-trained CNN are used to build a graph. Features refined using the pre-trained GAT layer(s). Optimal Transport based fine-tuning algorithm aligns support features with query features.

relations and can be extended for transduction, as used in [28, 33]. In addition to transduction and relation modeling, graphs are highly potent as task adaptation modules. HGNN [58] is an example where a graph is used to refine and adapt feature embeddings. To the best of our knowledge, it must be noted that most graph-based methods have been applied in the supervised FSL setting and that we are the first to use it in any form for unsupervised FSL.

## 3. Proposed Method (SAMPTransfer)

In this section, we first describe our problem formulation. We then discuss the two subsequent phases of the proposed approach: (i) self-supervised pre-training (SAMP-CLR), and (ii) the optimal transport based few-shot supervised fine-tuning (OpT-Tune). Together, these two phases constitute our overall approach, which we have coined as SAMPTransfer. The mechanics of the proposed pre-training and fine-tuning procedures are illustrated in Figs. 1 and 2, respectively.

### 3.1. Preliminaries

Let us denote the training data of size $D$ as $\mathcal{D}_{\text{tr}} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{D}$ with $(\boldsymbol{x}_i, y_i)$ representing an image and its class label, respectively. In the pre-training phase, we take $L$ random samples from $\mathcal{D}_{\text{tr}}$ and augment each sample $A$ times by randomly sampling augmentation functions $\zeta^a(.), \forall a \in [A]$ from the set $\mathcal{A}$. This results in a mini-batch of size $B = (A+1)L$ total samples. Note that in the unsupervised setting, we have no access to the data labels in the pre-training phase. Next, we fine-tune our model episodically [48] on a set of randomly sampled tasks $\mathcal{T}_i$ drawn from the test dataset $\mathcal{D}_{\text{tst}} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{D'}$ of size $D'$. A task, $\mathcal{T}_i$, is comprised of two parts: (i) the support set $\mathcal{S}$ from which the model learns, (ii) the query set $\mathcal{Q}$ on which the model is evaluated. The support set $\mathcal{S} = \{\boldsymbol{x}_i^s, y_i^s\}_{i=1}^{NK}$ is constructed by drawing $K$ labeled random samples from $N$ different classes, resulting in the so-called ($N$-way, $K$-shot) settings. The query set $\mathcal{Q} = \{\boldsymbol{x}_j^q\}_{j=1}^{NQ}$ then contains $NQ$ unlabeled samples. By convention, we denote the set $\mathcal{T}_i = \mathcal{S} \cup \mathcal{Q}$ with $(N, K)$.

### 3.2. Self-Attention Message Passing (SAMP)

Our network architecture consists of a convolutional (CNN) feature extractor $f_\theta$ and a message passing network based on self-attention, $f_\psi$. The CNN feature extractor $f_\theta$, parameterized by $\theta$, is used to extract features $\boldsymbol{V} = f_\theta(\boldsymbol{X})$, where $\boldsymbol{V} \in \mathbb{R}^{B \times d}$ is the set of $B$ features each of size $d$ and $\boldsymbol{X} \in \mathbb{R}^{B \times C \times H \times W}$ is a batch of $B$ images of size

$C \times H \times W$. To help refine the features and use batch-level relationships, we create a graph $\mathcal{G} = \texttt{Graph}(\boldsymbol{V}, \boldsymbol{e}, \gamma)$ where $\boldsymbol{V}$ is treated as a set of initial node features, $\boldsymbol{e}$ is the pairwise distance between all nodes based on a given distance metric, and $\gamma$ is a threshold on the values in $\boldsymbol{e}$ that determines whether two nodes will be connected or not. Note that $|\mathcal{G}| = B$, as we build the graph over the $B$ samples in our batch. We use a **s**elf-**a**ttention **m**essage **p**assing neural network (we call SAMP) $f_\psi$, parameterized by $\psi$, to refine the initial feature vectors by exchanging and amalgamating information between all pairs of connected nodes. From now on, as can be seen in Figs. 1 and 2, we refer to the combination of the feature extractor $f_\theta$ and the SAMP module $f_\psi$ as $f_\Omega = f_\psi \circ f_\theta$ where $\Omega = \{\theta, \psi\}$ is the collection of all parameters. The SAMP layers, $f_\psi$ operate on the graph $\mathcal{G}$.

To allow an effective exchange of information to refine initial node features $\boldsymbol{v}$, we make use of graph attention in a slightly different manner than the standard graph attention defined in [47]. The graph attention in [47] uses a single weight matrix $\boldsymbol{W}$ that acts as a shared linear transformation for all nodes. Instead, we choose to use scaled dot-product self-attention as defined in [40, 46]. The major benefit of this design choice is that it enhances the network with more expressivity, as shown in [5, 29]. Notably, the use of three separate representations (query, key, and value) instead of just a single weight matrix to linearly transform the data is key to modeling relationships between data points.

We apply $p$ successive message passing steps similar to [40, 47]. In each step, we pass messages between the connected nodes of $\mathcal{G}$ and obtain updated features in $\boldsymbol{V}^{p+1}$, at step $p+1$. Here, the $i$-th row of $\boldsymbol{V}^{p+1}$ is given by $\boldsymbol{V}_i^{p+1} = \sum_{j \in \mathcal{N}_i} \lambda_{ij}^p \boldsymbol{W}^p \boldsymbol{V}_j^p$, where $\lambda_{ij}$ is the attention score between the nodes $i$ and $j$, $\boldsymbol{W}^p$ is the message passing weight matrix at step $p$, and $\mathcal{N}_i$ denotes the set of neighboring nodes of node $i$. In this way, $\lambda_{i,j}$ allows our update mechanism to flexibly weight every samples w.r.t every other sample in the batch. We employ scaled dot-product self-attention to compute attention scores, leading to: $\lambda_{ij}^p = \text{softmax}(\boldsymbol{W}_q^p \boldsymbol{V}_i^p (\boldsymbol{W}_k^p \boldsymbol{V}_j^p)^T / \sqrt{d})$ where $\boldsymbol{W}_k^p$ and $\boldsymbol{W}_q^p$ are the weight matrices corresponding to the sending and receiving nodes, respectively. To allow the message-passing neural network to learn a diverse set of attention scores, we apply $H$ scaled dot-product self-attention heads in every message-passing step and concatenate their results. To this end, instead of using single weight matrices $\boldsymbol{W}_q^p, \boldsymbol{W}_k^p$ and $\boldsymbol{W}^p$, we use $\boldsymbol{W}_q^{p,h}, \boldsymbol{W}_k^{p,h}$ and $\boldsymbol{W}^{p,h}$ all $\in \mathbb{R}^{d/H \times d}$ for each attention head, resulting in:

$$\boldsymbol{V}_i^{p+1} = \left[ \sum_{j \in N_i} \lambda_{ij}^{p,1} \boldsymbol{W}^{p,1} \boldsymbol{V}_j^p, \ldots, \sum_{j \in N_i} \lambda_{ij}^{p,H} \boldsymbol{W}^{p,H} \boldsymbol{V}_j^p \right],$$

note that $\boldsymbol{V}_i^{p+1}$ still has the same dimension $\mathbb{R}^d$.

**Algorithm 1:** SAMP-CLR

**Require:** $\mathcal{A}$, $f_\theta$, $f_\psi$, $\Omega$, $\alpha$, $\beta$, $\eta$, $d[.]$, $d'[.]$

1 **while** *not done* **do**
2 Sample minibatch $\{\boldsymbol{x}_i\}_{i=1}^L$
3 Augment samples: $\bar{\boldsymbol{x}}_{i,a} = \zeta_a(\boldsymbol{x}_i)$; $\zeta_a \sim \mathcal{A}$.
4 $\boldsymbol{Z}, \bar{\boldsymbol{Z}} \leftarrow f_\theta\left(\{\boldsymbol{x}_i\}_{i=1}^L\right), f_\theta\left(\{\bar{\boldsymbol{x}}_{i,\,a}\}_{i=1,a=1}^{L,A}\right)$
5 $\boldsymbol{V} = [\boldsymbol{Z}^\top, \bar{\boldsymbol{Z}}^\top]^\top$, $\boldsymbol{e} = \{d'[\boldsymbol{V}_i, \boldsymbol{V}_j], \forall i, j \in [B]\}$
6 $\mathcal{G} \leftarrow \texttt{Graph}(\boldsymbol{V}, \boldsymbol{e}, \gamma)$
7 $\mathcal{G}' \leftarrow \texttt{Graph}(\boldsymbol{V}', \boldsymbol{e}', \gamma) = f_\psi(\mathcal{G})$
8 $\boldsymbol{Z}', \bar{\boldsymbol{Z}}' \leftarrow \boldsymbol{V}'_{1:L}, \boldsymbol{V}'_{L+1:B}$
9 $\ell(i,a) = -\log \frac{\exp\left(-d\left[\bar{\boldsymbol{Z}}_{(a-1)L+i}, \boldsymbol{Z}_i\right]\right)}{\sum_{k=1}^L \exp\left(-d\left[\bar{\boldsymbol{Z}}_{(a-1)L+i}, \boldsymbol{Z}_k\right]\right)}$
10 $r(i,a) = -\log \frac{\exp\left(-d\left[\bar{\boldsymbol{Z}}'_{(a-1)L+i}, \boldsymbol{Z}'_i\right]\right)}{\sum_{k=1}^L \exp\left(-d\left[\bar{\boldsymbol{Z}}'_{(a-1)L+i}, \boldsymbol{Z}'_k\right]\right)}$
11 $\mathcal{L}_1 = {}^1\!/_{LA} \sum_{i=1}^L \sum_{a=1}^A \ell(i,a)$
12 $\mathcal{L}_2 = {}^1\!/_{LA} \sum_{i=1}^L \sum_{a=1}^A r(i,a)$
13 $\mathcal{L} = \beta \mathcal{L}_1 + \mathcal{L}_2$
14 $\Omega \leftarrow \Omega - \eta \nabla_\Omega \mathcal{L}$
15 **end**

### 3.3. Self-Supervised Pre-Training (SAMP-CLR)

The fact that we do not have access to the true class labels of the training data underscores the need to use a self-supervised pre-training scheme. As briefly discussed in Section 1, we build on the idea of employing contrastive prototypical transfer learning with some inspiration from [34, 41]. Standard contrastive learning enforces embeddings of augmented images to be close to the embeddings of their source images in the representation space. The *key idea* of SAMP-CLR is not only to perform contrastive learning (the "CLR" component) on the source and augmented image embeddings but also to ensure that images in the mini-batch belonging to potentially the same class have similar embeddings. This is where the "SAMP" module comes to rescue, enabling the model look beyond single instances and their augmentations. SAMP allows the model to extract richer semantic information across multiple images present in a mini-batch. Concretely speaking, we apply a contrastive loss on the SAMP refined features (generated by $f_\psi$), and on the standard convolutional features (generated by $f_\theta$). Let us walk you through the process in more detail.

Algorithm 1 begins with batch generation: each mini-batch consists of $L$ random samples $\{\boldsymbol{x}_i\}_{i=1}^L$ from $\mathcal{D}_{\text{tr}}$, where $\boldsymbol{x}_i$ is treated as a 1-shot support sample for which we create $A$ randomly augmented versions $\tilde{\boldsymbol{x}}_{i,a}$ as query samples (lines 2 to 3), leading to a batch size of $B = (A+1)L$. Then embeddings $\boldsymbol{Z} \in \mathbb{R}^{L \times d}$ and $\bar{\boldsymbol{Z}} \in \mathbb{R}^{LA \times d}$ are generated (line 4) by passing the source images and augmented

images through a feature extraction network $f_\theta$, respectively. We then construct $\mathcal{G} = \texttt{Graph}(\boldsymbol{V}, \boldsymbol{e}, \alpha)$ with $\boldsymbol{V} = [\boldsymbol{Z}^\top, \bar{\boldsymbol{Z}}^\top]^\top$ of size $B \times d$ concatenating source and augmented image embeddings $\boldsymbol{Z}$ and $\bar{\boldsymbol{Z}}$ (line 5-6), $\boldsymbol{e}$ is the vector of centered shift/scale-invariant cosine similarities $d'[.]$ (line 5) [45], and $\gamma$ is defined earlier. The graph $\mathcal{G}$ is then passed through the SAMP layer(s) $f_\psi$ resulting in a updated graph $\mathcal{G}'$ with refined node features $\boldsymbol{V}'$ (line 7). $\boldsymbol{V}'$ is spliced into the updated source image and augmented image embeddings ($\boldsymbol{Z}'$ and $\bar{\boldsymbol{Z}}'$), respectively (lines 8). In lines 9 to 12, we then apply contrastive losses $\mathcal{L}_1$ (between $\boldsymbol{Z}$ and $\bar{\boldsymbol{Z}}$) and $\mathcal{L}_2$ (between $\boldsymbol{Z}'$ and $\bar{\boldsymbol{Z}}'$). Here, $\mathcal{L}_1$ encourages the feature extractor to cluster the embeddings of augmented query samples $\bar{\boldsymbol{Z}}$ around their prototypes (namely, source embeddings) $\boldsymbol{Z}$, which in turn provides a good initial set of embeddings for the SAMP projector module to refine. $\mathcal{L}_2$ enforces the same constraints as $\mathcal{L}_1$ but for embeddings generated by the SAMP layer. Both loss terms use a Euclidean distance metric in the embedding space, denoted by $d[.]$. Finally, the overall loss is given by $\mathcal{L} = \beta\mathcal{L}_1 + \mathcal{L}_2$, with $\beta$ being a scaling factor, is optimized with mini-batch stochastic gradient descent w.r.t all the parameters in $\Omega = \{\theta, \psi\}$ and the learning rate $\eta$.

### 3.4. Supervised Fine-tuning (OpT-Tune)

We propose a two-stage supervised fine-tuning consisting of (i) a transportation stage followed by (ii) a prototypical fine-tuning and classification stage. The transportation stage involves using optimal transport (OT) [12, 36]. As is sketched in Fig. 2, OT helps projecting embeddings of the support set, $\boldsymbol{Z}^s = f_\Omega(\{\boldsymbol{x}_i^s\}_{i=1}^{NK}) \in \mathbb{R}^{NK\times d}$, such that they overlaps better with the query set embeddings, $\boldsymbol{Z}^q = f_\Omega(\{\boldsymbol{x}_j^q\}_{j=1}^{NQ}) \in \mathbb{R}^{NQ\times d}$ upon transportation. This increases the spread of $\boldsymbol{Z}^s$ in the query set's domain which in turn creates more representative prototypes for each of the $N$ classes in $\mathcal{S}$. We demonstrate, in Section 6 that this brings about significant impact on the downstream classification performance.

**OT based feature alignment.** We provide a basic intuition for OT [12, 36] in the context of SAMPTransfer. Let $\boldsymbol{r} \in \mathbb{R}^{NK}$ and $\boldsymbol{c} \in \mathbb{R}^{NQ}$ be two probability simplexes defined over $\boldsymbol{Z}_i^s, \forall i \in [NK]$ and $\boldsymbol{Z}_j^q, \forall j \in [NQ]$, respectively. $\boldsymbol{r}$ denotes the distribution of the support embeddings allocated to each query embedding, whereas $\boldsymbol{c}$ denotes the distribution of the query embeddings allocated to each support embedding. Consider $\boldsymbol{\Pi}(\boldsymbol{r}, \boldsymbol{c})$ to be a set of $NK \times NQ$ doubly stochastic matrices where all rows sum up to $\boldsymbol{r}$ and all columns sum up to $\boldsymbol{c}$ as:

$$\boldsymbol{\Pi}(\boldsymbol{r}, \boldsymbol{c}) = \left\{ \boldsymbol{\pi} \in \mathbb{R}_+^{NK\times NQ} \mid \boldsymbol{\pi}\mathbf{1}_{NQ} = \boldsymbol{r}, \boldsymbol{\pi}^\top \mathbf{1}_{NK} = \boldsymbol{c} \right\}. \quad (1)$$

Intuitively, $\boldsymbol{\Pi}(\boldsymbol{r}, \boldsymbol{c})$ is a collection of all transport "plans", where a transport plan is defined as a potential strategy specifying how much of each support embedding is allocated to every query embedding and vice-versa. Our goal here is to find the most optimal transport plan, out of all possible transport plans $\boldsymbol{\Pi}(\boldsymbol{r}, \boldsymbol{c})$, that allocates $NK$ support embeddings to $NQ$ query embeddings with maximum overlap between their distributions.

**Algorithm 2:** OpT-Tune

**Require:** $d[\cdot]$, $\boldsymbol{Z}^s$, $\boldsymbol{Z}^q$

1 $\boldsymbol{M}_{i,j} = d[\boldsymbol{Z}_i^s, \boldsymbol{Z}_j^q], \forall i \in [NK], j \in [NQ]$

2 $\boldsymbol{\pi}^\star \leftarrow$ Solving Eq. (2) using Sinkhorn-Knopp

3 $\hat{\boldsymbol{\pi}}_{i,j}^\star \leftarrow \boldsymbol{\pi}_{i,j}^\star \,/\, \sum_j \boldsymbol{\pi}_{i,j}^\star, \forall i \in [NK], j \in [NQ]$

4 Solve Eq. (3)

**Return:** $\widehat{\boldsymbol{Z}^s}$

Given a cost matrix $\boldsymbol{M}$, the cost of mapping $\boldsymbol{Z}^s$ to $\boldsymbol{Z}^q$ using a transport plan $\boldsymbol{\pi}$ can be quantified as $\langle \boldsymbol{\pi}, \boldsymbol{M} \rangle_F$ and the OT problem can then be stated as,

$$\boldsymbol{\pi}^\star = \underset{\boldsymbol{\pi}\in\boldsymbol{\Pi}(\boldsymbol{r},\boldsymbol{c})}{\operatorname{argmin}} \langle \boldsymbol{\pi}, \boldsymbol{M} \rangle_F - \varepsilon \mathbb{H}(\boldsymbol{\pi}), \quad (2)$$

where $\boldsymbol{\pi}^\star$ denotes the most optimal transportation plan, $\langle \cdot, \cdot \rangle_F$ is the Frobenius dot product, and $\varepsilon$ is the weight on the entropic regularizer $\mathbb{H}$ [12]. The cost matrix $\boldsymbol{M}$ quantifies the overlap between the two distributions by measuring the distance between each support and query embedding pair: $\boldsymbol{M}_{i,j} = d[\boldsymbol{Z}_i^s, \boldsymbol{Z}_j^q]$. The entropic regularization promotes "smoother" transportation plans [12]. Equation (2) is then solved using the time-efficient Sinkhorn-Knopp algorithm. Notice that $\boldsymbol{\pi}^\star$ is also referred to as *Wasserstein metric* [12, 36]. To adapt $\boldsymbol{Z}^s$ to $\boldsymbol{Z}^q$ with cost matrix $\boldsymbol{M}$, we compute $\hat{\boldsymbol{Z}}^s$ as the *projected mapping* of $\boldsymbol{Z}^s$, given by:

$$\hat{\boldsymbol{Z}}^s = \hat{\boldsymbol{\pi}}^\star \boldsymbol{Z}^q, \quad (3)$$

$$\hat{\boldsymbol{\pi}}_{i,j}^\star = \frac{\boldsymbol{\pi}_{i,j}^\star}{\sum_j \boldsymbol{\pi}_{i,j}^\star}, \forall i \in [NK], j \in [NQ], \quad (4)$$

where $\widehat{\boldsymbol{\pi}}^\star$ is the normalized transport. The *projected support* embeddings $\hat{\boldsymbol{Z}}^s$ are an estimation of $\boldsymbol{Z}^s$ in the region occupied by the query embeddings $\boldsymbol{Z}^q$. Specifically, it is a barycentric mapping of the support features $\boldsymbol{Z}^s$. Algorithm 2 shows this process in a succinct manner.

**Prototypical classification**. The projected support embeddings, $\hat{\boldsymbol{Z}}^s$, are used for prototype creation and classification of the query points. To this end, following [34, 44] we concatenate $f_\Omega$ with a single layer nearest mean classifier $f_\phi$ (resulting in an architecture similar to ProtoNet [42]) and only fine-tune this last layer. In this stage, for each class $k \in \mathcal{C}$ in the support set, we compute the class prototype $\boldsymbol{c}_k$ for class $k$ using the projected support embeddings $\hat{\boldsymbol{Z}}^{s,k}$ belonging to class $k$:

$$\boldsymbol{c}_k = \frac{1}{\left|\hat{\boldsymbol{Z}}^{s,k}\right|} \sum_{\hat{\boldsymbol{z}}\in\hat{\boldsymbol{Z}}^{s,k}} \hat{\boldsymbol{z}}, \text{ for } k \in \mathcal{C}.$$

Following [34, 44], we initialize the classification layer $f_\phi$ with weights set to $\boldsymbol{W}_k = 2\boldsymbol{c}_k$ and biases set to $b_k = -\|\boldsymbol{c}_k\|^2$. To finetune this layer, we sample a subset of supports from $\mathcal{S}$ and train $f_\phi$ with a standard cross-entropy loss; more details are given in Section 4.

## 4. Experimental Setup

**Datasets.** To benchmark the performance of `SAMPTransfer`, we conduct "in-domain" experimentation on two most commonly adopted few-shot learning datasets: *mini*-ImageNet [48] and *tiered*ImageNet [38]. *Mini*-ImageNet contains 100 classes with 600 samples in each class. This equals a total of $60,000$ images that we resize to $84 \times 84$ pixels. Out of the 100 classes, we use 64 classes for training, 16 for validation, and 20 for testing. *Tiered*ImageNet is a larger subset of ILSVRC-12 [15] with 608 classes with a total of $779,165$ images of size $84 \times 84$. We use 351 for training, 97 for validation, and 8 for testing, out of the 608 classes. The augmentation strategy follows the one proposed in [2]. We also compare our method on a recent more challenging "cross-domain" few-shot learning (CDFSL) benchmark [22], which consists of several datasets. This benchmark has four datasets with increasing similarities to *mini*-ImageNet. In that order, we have grayscale chest X-ray images from ChestX [51], dermatological skin lesion images from ISIC2018 [10], aerial satellite images from EuroSAT [24], and crop disease images from CropDiseases [35]. We also used the Caltech-UCSD Birds (CUB) dataset [49] for further analysis of cross-domain performance. The CUB dataset is made up of $11,788$ images from 200 unique species of birds. We use 100 classes for training, 50 for both validation and testing.

**Training strategy.** In Fig. 1, as feature extractor, we use the standard `Conv4` model following [27, 34, 48]. It is followed by a single `SAMP` layer with 4 attention heads. Note that we also use a slightly modified version of the `Conv4` network which we call `Conv4b`, where we increase the number of filters from $(64, 64, 64, 64)$ to $(96, 128, 256, 512)$ [19] and average pool the final feature map returning a smaller embedding dimension $d = 512$ instead of $d = 1600$. The networks are pre-trained using `SAMP-CLR` on the respective training splits of the datasets, with an initial learning rate of $\eta = 0.0005$, annealed by a cosine scheduler via the Adam optimizer [30] and $L = 128$. Experiments involving CDFSL benchmark follow [22, 34, 41], where we pre-train a ResNet-10 encoder using `SAMP-CLR` on *mini*-ImageNet images of size $224 \times 224$ for 400 epochs with the Adam optimizer and a constant learning rate of $\eta = 0.0001$. Similar to the `Conv4` encoder, the ResNet-10 uses the same `SAMP` configuration.

During validation and testing, as defined in Section 3.4, we initialize and fine-tune $f_\phi$ for 15 iterations where we sample a subset of examples from $\mathcal{S}$ in each iteration. For validation, we create 15 ($N$-way, $K$-shot) tasks using the validation split of the respective dataset.

**Evaluation scenarios and baseline.** Our testing scheme uses 600 test episodes, each with 15 query shots per class, on which the pre-trained encoder (`SAMP-CLR`) is fine-tuned using `OpT-Tune` and tested. All our results indicate 95% confidence intervals over 3 runs, each with 600 test episodes. Therefore, the standard deviation values are calculated according to the 3 runs to provide more concrete measures for comparison. For our in-domain benchmarks, we test on (5-way, 1-shot) and (5-way, 5-shot) classification tasks, while our cross-domain testing is performed using (5-way, 5-shot) and (5-way, 20-shot) classification tasks following [22]. We compare our performance with a suite of recent unsupervised few-shot baselines such as U-MlSo [60], C$^3$LR [41], Meta-GMVAE [31], and Revisiting UML [56] to name a few. Furthermore, we also compare with a set of supervised approaches (such as MetaQDA [61] and TransductiveCNAPS [3]), the best of which are expected to outperform ours and other unsupervised methods.

Table 1: Accuracy (%± std.) for ($N$-way, $K$-shot) classification tasks. Style: **best** and second best.

| | | ***mini*-ImageNet** | |
|---|---|---|---|
| **Method**$(N, K)$ | **Backbone** | (5,1) | (5,5) |
| CACTUs-MAML [25] | Conv4 | 39.90 ± 0.74 | 53.97 ± 0.70 |
| CACTUs-Proto [25] | Conv4 | 39.18 ± 0.71 | 53.36 ± 0.70 |
| UMTRA [27] | Conv4 | 39.93 | 50.73 |
| AAL-ProtoNet [1] | Conv4 | 37.67 ± 0.39 | 40.29 ± 0.68 |
| AAL-MAML++ [1] | Conv4 | 34.57 ± 0.74 | 49.18± 0.47 |
| UFLST [26] | Conv4 | 33.77 ± 0.70 | 45.03 ± 0.73 |
| ULDA-ProtoNet [37] | Conv4 | 40.63 ± 0.61 | 55.41 ± 0.57 |
| ULDA-MetaNet [37] | Conv4 | 40.71 ± 0.62 | 54.49 ± 0.58 |
| U-SoSN+ArL [59] | Conv4 | 41.13 ± 0.84 | 55.39 ± 0.79 |
| U-MlSo [60] | Conv4 | 41.09 | 55.38 |
| ProtoTransfer [34] | Conv4 | 45.67 ± 0.79 | 62.99 ± 0.75 |
| CUMCA [54] | Conv4 | 41.12 | 54.55 |
| Meta-GMVAE [31] | Conv4 | 42.82 | 55.73 |
| Revisiting UML [56] | Conv4 | 48.12 ± 0.19 | 65.33 ± 0.17 |
| CSSL-FSL_Mini64 [32] | Conv4 | 48.53 ± 1.26 | 63.13 ± 0.87 |
| C$^3$LR [41] | Conv4 | 47.92 ± 1.2 | 64.81 ± 1.15 |
| `SAMPTransfer` (**ours**) | Conv4 | 55.75 ± 0.77 | 68.33 ± 0.66 |
| `SAMPTransfer`$^\star$ (**ours**) | Conv4b | **61.02** ± 1.0 | **72.52** ± 0.68 |
| *Supervised Methods* | | | |
| MAML [18] | Conv4 | 46.81 ± 0.77 | 62.13± 0.72 |
| ProtoNet [42] | Conv4 | 46.44± 0.78 | 66.33± 0.68 |
| MMC [38] | Conv4 | 50.41 ± 0.31 | 64.39 ± 0.24 |
| FEAT [57] | Conv4 | 55.15 | 71.61 |
| SimpleShot [52] | Conv4 | 49.69 ± 0.19 | 66.92 ± 0.17 |
| Simple CNAPS [3] | ResNet-18 | 53.2 ± 0.9 | 70.8 ± 0.7 |
| Transductive CNAPS [3] | ResNet-18 | 55.6 ± 0.9 | 73.1 ± 0.7 |
| MetaQDA [61] | Conv4 | 56.41 ± 0.80 | 72.64 ± 0.62 |
| Pre+Linear [34] | Conv4 | 43.87 ± 0.69 | 63.01 ± 0.71 |

Table 2: Accuracy (%± std.) for ($N$-way, $K$-shot) classification tasks. Style: **best** and second best.

| Method$(N, K)$ | Backbone | *tiered*ImageNet (5,1) | *tiered*ImageNet (5,5) |
|---|---|---|---|
| $C^3$LR [41] | Conv4 | 42.37 ± 0.77 | 61.77 ± 0.25 |
| ULDA-ProtoNet [37] | Conv4 | 41.60 ±0.64 | 56.28 ± 0.62 |
| ULDA-MetaOptNet [37] | Conv4 | 41.77 ± 0.65 | 56.78 ± 0.63 |
| U-SoSN+ArL [59] | Conv4 | 43.68 ± 0.91 | 58.56 ± 0.74 |
| U-MlSo [60] | Conv4 | 43.01 ± 0.91 | 57.53 ± 0.74 |
| `SAMPTransfer` (**ours**) | Conv4 | 45.25 ± 0.89 | 59.75 ± 0.66 |
| `SAMPTransfer`$^\star$ (**ours**) | Conv4b | **49.10** ± 0.94 | **65.19** ± 0.82 |

## 5. Performance Evaluation

**In-Domain Experiments.** Table 1 summarizes our performance evaluation results on the *mini*-ImageNet dataset for ($N$-way, $K$-shot) scenarios with $N = 5$ and $K = 1, 5$. The top section compares the performance of the proposed approach (`SAMP-CLR`) with the most recent unsupervised competitors. We outperform our closest competitors by approximately $7\%+$ and $2\%+$ in the (5-way, 1-shot) and (5-way, 5-shot) settings, respectively. More interestingly, our method matches or beats some of the supervised baselines (bottom section of the table), especially SimpleCNAPS which uses a much more powerful ResNet-18 backbone. Obviously, the state-of-the-art supervised few-shot learning approaches have the advantage of having access to the true labels. When it comes to *tiered*ImageNet, our approach shows considerable gains over recent competitors such as $C^3$LR [41] with a $3\%+$ improvement in the (5-way, 1-shot) setting and a $5\%+$ improvement in the (5-way,5 shot) setting. As such, `SAMPTransfer` sets a new state-of-the-art for both *tiered*ImageNet and *mini*-ImageNet datasets.

**Cross-Domain experiments.** We focus on the recent CDFSL benchmark [22] to investigate the performance of `SAMPTransfer` in cross-domain scenarios. This outcome is summarize in Table 3. Here, we pre-train on *mini*-ImageNet and fine-tune on ChestX [51], ISIC2018 [10], EuroSAT [24], and CropDiseases [35]. We compare the performance against $C^3$LR[41], ProtoTransfer [34] along with its two variants using UMTRA [27] (also proposed in [34]), as well as ConFeSS [13] and ATA [50] - two of the latest methods *dedicated* to solving the cross-domain few-shot learning problem. Note that we also compare with a couple of related supervised approaches from [22], as a reference. Our method consistently keeps up with ConFeSS [13], but scores higher in 5 and 20 shot CropDiseases tasks by $2\%+$ and about $1\%$, respectively. Except for EuroSAT, our method is consistently competitive ($\sim 1\%$ difference in accuracy) to the performance of ConFeSS in ChestX and ISIC. In ISIC, which is the second least similar dataset to *mini*-ImageNet, our method is better by $1\%+$ in the (5-way, 20-shot) setting. Note that `SAMPTransfer` outperforms another recent dedicated method ATA [50] in all CDFSL benchmark settings.

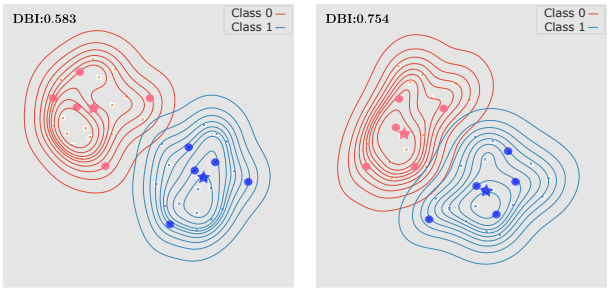



Figure 3: Before (left) and after applying OT (right). ★, ●, ✖ mark prototypes, supports and queries, respectively.

## 6. Ablation Study and Robustness Analysis

Table 4 investigates the performance of the proposed method against various choices of important hyperparameters. We use the (5-way, 5-shot) *mini*-ImageNet benchmark to analyze the robustness of our method and demonstrate the importance of our design choices.

**`OpT-Tune` is crucial.** To illustrate the effect of using `OpT-Tune` on the classification performance, we perform experiments with `OpT-Tune` disabled. For a fair comparison, we use the same pre-trained models in the test runs with `OpT-Tune` enabled or disabled. The best performing model (a `Conv4b`) uses 1 `SAMP` layer with 4 attention heads and a batch size of 128, resulting in accuracy of $72.52\%$ with `OpT-Tune` enabled. The same model, with `OpT-Tune` disabled, loses $9\%$ accuracy. Even with `OpT-Tune` disabled, our method remains competitive with some of the latest methods in Table 1. This observation suggests that the process described in Section 3.4 is an efficient technique to incorporate task awareness and improve the quality of prototypes. This is further corroborated in Fig. 3 where a task with $N = 2$ is used to show-case the effect of `OpT-Tune`. We observe that the support embeddings are more evenly spread out over the distribution of the query embeddings. This is also backed by the DBI score [14] which increases from $0.583$ to $0.754$ after `OpT-Tune` is applied.

**SAMP layers and attention heads.** In Table 4, we also investigate the robustness of our method when the number of `SAMP` layers ($p$) and attention heads ($H$) vary. The best performance is achieved with a single `SAMP` layer with four attention heads. Increasing $p$ leads to a significant decrease in performance; however, increasing $H$ leads to a small performance degradation. Notably, the observations here are consistent with those reported in [40, 47].

**Embedding dimension.** We measure the performance of the model in relation to two commonly used (by a majority of the existing baselines) embedding dimensions: 512 and

Table 3: Accuracy (%± std.) of ($N$-way, $K$-shot) classification on the CDFSL benchmark. Style: **best** and second best.

| **Method**$(N, K)$ | (5,5) | (5,20) | (5,5) | (5,20) | (5,5) | (5,20) | (5,5) | (5,20) |
|---|---|---|---|---|---|---|---|---|
| | **ChestX** | | **ISIC** | | **EuroSAT** | | **CropDiseases** | |
| UMTRA-ProtoNet [34] | 24.94 ± 0.43 | 28.04 ± 0.44 | 39.21 ± 0.53 | 44.62 ± 0.49 | 74.91 ± 0.72 | 80.42 ± 0.66 | 79.81 ± 0.65 | 86.84 ± 0.50 |
| UMTRA-ProtoTune [34] | 25.00 ± 0.43 | 30.41 ± 0.44 | 38.47 ± 0.55 | 51.60 ± 0.54 | 68.11 ± 0.70 | 81.56 ± 0.54 | 82.67 ± 0.60 | 92.04 ± 0.43 |
| ProtoTransfer [34] | 26.71 ± 0.46 | 33.82 ± 0.48 | 45.19 ± 0.56 | 59.07 ± 0.55 | 75.62 ± 0.67 | 86.80 ± 0.42 | 86.53 ± 0.56 | 95.06 ± 0.32 |
| $C^3$LR [41] | 26.00 ± 0.41 | 33.39 ± 0.47 | 45.93 ± 0.54 | 59.95 ± 0.53 | 80.32 ± 0.65 | 88.09 ± 0.45 | 87.90 ± 0.55 | 95.38 ± 0.31 |
| `SAMPTransfer` (**ours**) | 26.27 ± 0.44 | **34.15** ± 0.50 | 47.60 ± 0.59 | **61.28** ± 0.56 | 81.58 ± 0.63 | 88.52 ± 0.50 | **91.74** ± 0.55 | **96.36** ± 0.28 |
| ConFeSS [13] (dedicated) | **27.09** | 33.57 | **48.85** | 60.10 | **84.65** | **90.40** | 88.88 | 95.34 |
| ATA [50] (dedicated) | 24.43 ± 0.2 | - | 45.83 ± 0.3 | - | 83.75 ± 0.4 | - | 90.59 ± 0.3 | - |
| ProtoNet [22] (sup.) | 24.05 ± 1.01 | 28.21 ± 1.15 | 39.57 ± 0.57 | 49.50 ± 0.55 | 73.29 ± 0.71 | 82.27 ± 0.57 | 79.72 ± 0.67 | 88.15 ± 0.51 |
| Pre+Mean-Cent. [22] (sup.) | 26.31 ± 0.42 | 30.41 ± 0.46 | 47.16 ± 0.54 | 56.40 ± 0.53 | 82.21 ± 0.49 | 87.62 ± 0.34 | 87.61 ± 0.47 | 93.87 ± 0.68 |
| Pre+Linear [22] (sup.) | 25.97 ± 0.41 | 31.32 ± 0.45 | 48.11 ± 0.64 | 59.31 ± 0.48 | 79.08 ± 0.61 | 87.64 ± 0.47 | 89.25 ± 0.51 | 95.51 ± 0.31 |

Table 4: Ablation study of various parameters on accuracy.

| **Backbone** | $p$ | $H$ | $L$ | $\beta$ | **OT** | **Accuracy** |
|---|---|---|---|---|---|---|
| Conv4b | 1 | 4 | 64 | 1.0 | ✓ | 71.42 ± 0.73 |
| Conv4b | 1 | 4 | 64 | 0.7 | ✓ | 71.41 ± 0.71 |
| Conv4b | 1 | 8 | 64 | 1.0 | ✓ | 71.27 ± 0.75 |
| Conv4b | 1 | 8 | 64 | 0.7 | ✓ | 69.87 ± 0.72 |
| Conv4b | 2 | 1 | 64 | 0.7 | ✓ | 68.99 ± 0.71 |
| Conv4b | 2 | 4 | 64 | 0.7 | ✓ | 67.01 ± 0.69 |
| Conv4 | 1 | 4 | 64 | 0.7 | ✓ | 69.61 ± 0.71 |
| Conv4 | 1 | 4 | 64 | 1.0 | ✓ | 67.60 ± 0.62 |
| Conv4 | 1 | 8 | 64 | 1.0 | ✓ | 63.59 ± 0.68 |
| Conv4b | 1 | 4 | 128 | 0.7 | ✓ | 72.52 ± 0.72 |
| Conv4 | 1 | 4 | 128 | 0.7 | ✓ | 68.33 ± 0.71 |
| Conv4 | 1 | 4 | 128 | 0.0 | ✓ | 52.81 ± 0.66 |
| Conv4b | 1 | 4 | 128 | 0.0 | ✓ | 72.44 ± 0.69 |
| Conv4b | 1 | 4 | 64 | 0.7 | ✗ | 64.29 ± 0.63 |
| Conv4b | 1 | 4 | 128 | 0.7 | ✗ | 63.47 ± 0.64 |
| Conv4 | 1 | 4 | 64 | 0.7 | ✗ | 66.73 ± 0.65 |

Table 5: Accuracy (%± std.) for ($N$-way, $K$-shot) classification on *mini*-ImageNet with pre-training on CUB.

| **Training** | **Testing** | (5,1) | (5,5) |
|---|---|---|---|
| ProtoTransfer [34] | ProtoTune | 35.37 ± 0.63 | 52.38 ± 0.66 |
| $C^3$LR [41] | ProtoTune | 39.61 ± 1.11 | 55.53 ± 1.42 |
| `SAMPTransfer` (**ours**) | OpT-Tune | **49.32** ± 0.75 | **56.10** ± 0.60 |

1600. As can be seen in Table 4, the network performs best with an embedding dimension of 512. Performance is notably lower with an embedding dimension of 1600. We hypothesize that this behavior can be attributed to the lower number of channels in the final feature map of a `Conv4` network, which is limited to 64.

**Effect of loss scaling factor $\beta$ on $\mathcal{L}_1$.** We observe that when $\beta = 0$ the `Conv4` based model suffers the most as it loses 15% accuracy compared to $\beta = 0.7$, suggesting that training the CNN with a contrastive loss is crucial. However, the `Conv4b` model is not affected as strongly by the presence of this loss function. Regardless, we set $\beta = 0.7$ for both models (`Conv4` and `Conv4b`).

**Cross-domain robustness.** To further analyze the cross-domain performance characteristics of our method, in addition to Table 3, we trained a `Conv4` model on CUB and tested it on tasks derived from *mini*-ImageNet. CUB consists of 200 classes of only birds, while *mini*-ImageNet consists of 64 classes, of which only 3 training classes are birds. Thus, CUB has a diminished class diversity compared to *mini*-ImageNet. Table 5 shows that our method retains better transfer accuracy than other competing methods when training classes are diversity constrained.

## 7. Concluding Remarks

Inspired by the human mind's capacity to naturally make relevant connections between entities, we developed a contrastive learning scheme for unsupervised few-shot classification, where we supplemented a CNN's strong inductive prior with MPNNs to exploit intra-batch relations between images. Furthermore, we also showed that an optimal-transport-based task-awareness algorithm can aid in elevating the robustness of pre-trained models. We show that our approach (`SAMPTransfer`) offers appreciable performance improvements over its competitors in both in/cross-domain few-shot classification scenarios, setting new standards in the *mini*-ImageNet, *tiered*ImageNet and CDFSL benchmarks.

## References


[1] Antreas Antoniou and Amos Storkey. Assume, augment and learn: Unsupervised few-shot meta-learning via random labels and data augmentation. *arXiv preprint arXiv:1902.09884*, 2019.

[2] Philip Bachman, R Devon Hjelm, and William Buchwalter. Learning representations by maximizing mutual information across views. *Advances in neural information processing systems*, 32, 2019.

[3] Peyman Bateni, Jarred Barber, Jan-Willem van de Meent, and Frank Wood. Enhancing few-shot image classification with unlabelled examples. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2796–2805, 2022.

[4] Malik Boudiaf, Imtiaz Ziko, Jérôme Rony, José Dolz, Pablo Piantanida, and Ismail Ben Ayed. Information maximiza-

tion for few-shot learning. *Advances in Neural Information Processing Systems*, 33:2445–2457, 2020.

[5] Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? *arXiv preprint arXiv:2105.14491*, 2021.

[6] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments. *Advances in Neural Information Processing Systems*, 33:9912–9924, 2020.

[7] Da Chen, Yuefeng Chen, Yuhong Li, Feng Mao, Yuan He, and Hui Xue. Self-supervised learning for few-shot image classification. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1745–1749. IEEE, 2021.

[8] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[9] Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15750–15758, 2021.

[10] Noel Codella, Veronica Rotemberg, Philipp Tschandl, M Emre Celebi, Stephen Dusza, David Gutman, Brian Helba, Aadi Kalloo, Konstantinos Liopyris, Michael Marchetti, et al. Skin lesion analysis toward melanoma detection 2018: A challenge hosted by the international skin imaging collaboration (isic). *arXiv preprint arXiv:1902.03368*, 2019.

[11] Wentao Cui and Yuhong Guo. Parameterless transductive feature re-representation for few-shot learning. In *International Conference on Machine Learning*, pages 2212–2221. PMLR, 2021.

[12] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. *Advances in neural information processing systems*, 26, 2013.

[13] Debasmit Das, Sungrack Yun, and Fatih Porikli. ConfeSS: A framework for single source cross-domain few-shot learning. In *International Conference on Learning Representations*, 2022.

[14] David L. Davies and Donald W. Bouldin. A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1(2):224–227, 1979.

[15] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[16] Guneet S Dhillon, Pratik Chaudhari, Avinash Ravichandran, and Stefano Soatto. A baseline for few-shot image classification. *arXiv preprint arXiv:1909.02729*, 2019.

[17] Debidatta Dwibedi, Yusuf Aytar, Jonathan Tompson, Pierre Sermanet, and Andrew Zisserman. With a little help from my friends: Nearest-neighbor contrastive learning of visual representations. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9588–9597, 2021.

[18] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

[19] Spyros Gidaris, Andrei Bursuc, Nikos Komodakis, Patrick Pérez, and Matthieu Cord. Boosting few-shot visual learning with self-supervision. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 8059–8068, 2019.

[20] Priya Goyal, Mathilde Caron, Benjamin Lefaudeux, Min Xu, Pengchao Wang, Vivek Pai, Mannat Singh, Vitaliy Liptchinsky, Ishan Misra, Armand Joulin, and Piotr Bojanowski. Self-supervised pretraining of visual features in the wild. 3 2021.

[21] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent-a new approach to self-supervised learning. *Advances in neural information processing systems*, 33:21271–21284, 2020.

[22] Yunhui Guo, Noel CF Codella, Leonid Karlinsky, John R Smith, Tajana Rosing, and Rogerio Feris. A New Benchmark for Evaluation of Cross-Domain Few-Shot Learning. *arXiv preprint arXiv:1912.07200*, 2019.

[23] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9729–9738, 2020.

[24] Patrick Helber, Benjamin Bischke, Andreas Dengel, and Damian Borth. Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification, 2017.

[25] Kyle Hsu, Sergey Levine, and Chelsea Finn. Unsupervised learning via meta-learning. *arXiv preprint arXiv:1810.02334*, 2018.

[26] Zilong Ji, Xiaolong Zou, Tiejun Huang, and Si Wu. Unsupervised few-shot learning via self-supervised training. *arXiv preprint arXiv:1912.12178*, 2019.

[27] Siavash Khodadadeh, Ladislau Boloni, and Mubarak Shah. Unsupervised meta-learning for few-shot image classification. *Advances in neural information processing systems*, 32, 2019.

[28] Jongmin Kim, Taesup Kim, Sungwoong Kim, and Chang D Yoo. Edge-labeling graph neural network for few-shot learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11–20, 2019.

[29] Jinwoo Kim, Tien Dat Nguyen, Seonwoo Min, Sungjun Cho, Moontae Lee, Honglak Lee, and Seunghoon Hong. Pure transformers are powerful graph learners. *arXiv preprint arXiv:2207.02505*, 2022.

[30] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[31] Dong Bok Lee, Dongchan Min, Seanie Lee, and Sung Ju Hwang. Meta-gmvae: Mixture of gaussian vae for unsupervised meta-learning. In *ICLR*, 2021.

[32] Jianyi Li and Guizhong Liu. Few-shot image classification via contrastive self-supervised learning. *arXiv preprint arXiv:2008.09942*, 2020.

[33] Yanbin Liu, Juho Lee, Minseop Park, Saehoon Kim, Eunho Yang, Sung Ju Hwang, and Yi Yang. Learning to propagate labels: Transductive propagation network for few-shot learning. *arXiv preprint arXiv:1805.10002*, 2018.

[34] Carlos Medina, Arnout Devos, and Matthias Grossglauser. Self-supervised prototypical transfer learning for few-shot

classification. *arXiv preprint arXiv:2006.11325*, 2020.

[35] Sharada P Mohanty, David P Hughes, and Marcel Salathé. Using Deep Learning for Image-Based Plant Disease Detection. *Frontiers in Plant Science*, 7:1419, 2016.

[36] Gabriel Peyré, Marco Cuturi, et al. Computational optimal transport: With applications to data science. *Foundations and Trends® in Machine Learning*, 11(5-6):355–607, 2019.

[37] Tiexin Qin, Wenbin Li, Yinghuan Shi, and Yang Gao. Diversity helps: Unsupervised few-shot learning via distribution shift-based data augmentation. *arXiv preprint arXiv:2004.05805*, 2020.

[38] Mengye Ren, Eleni Triantafillou, Sachin Ravi, Jake Snell, Kevin Swersky, Joshua B Tenenbaum, Hugo Larochelle, and Richard S Zemel. Meta-learning for semi-supervised few-shot classification. *arXiv preprint arXiv:1803.00676*, 2018.

[39] Victor Garcia Satorras and Joan Bruna Estrach. Few-shot learning with graph neural networks. In *International Conference on Learning Representations*, 2018.

[40] Jenny Denise Seidenschwarz, Ismail Elezi, and Laura Leal-Taixé. Learning intra-batch connections for deep metric learning. In *International Conference on Machine Learning*, pages 9410–9421. PMLR, 2021.

[41] Ojas Kishore Shirekar and Hadi Jamali-Rad. Self-supervised class-cognizant few-shot classification. *arXiv preprint arXiv:2202.08149*, 2022.

[42] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. *Advances in neural information processing systems*, 30, 2017.

[43] Yonglong Tian, Yue Wang, Dilip Krishnan, Joshua B Tenenbaum, and Phillip Isola. Rethinking few-shot image classification: a good embedding is all you need? In *European Conference on Computer Vision*, pages 266–282. Springer, 2020.

[44] Eleni Triantafillou, Tyler Zhu, Vincent Dumoulin, Pascal Lamblin, Utku Evci, Kelvin Xu, Ross Goroshin, Carles Gelada, Kevin Swersky, Pierre-Antoine Manzagol, and Hugo Larochelle. Meta-Dataset: A Dataset of Datasets for Learning to Learn from Few Examples. *arXiv preprint arXiv:1903.03096*, 2020.

[45] Stijn Van Dongen and Anton J Enright. Metric distances derived from cosine similarity and pearson and spearman correlations. *arXiv preprint arXiv:1208.3145*, 2012.

[46] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[47] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

[48] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. *Advances in neural information processing systems*, 29, 2016.

[49] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The Caltech-UCSD Birds-200-2011 Dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011.

[50] Haoqing Wang and Zhi-Hong Deng. Cross-domain few-shot classification via adversarial task augmentation. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 1075–1081. International Joint Conferences on Artificial Intelligence Organization, 8 2021. Main Track.

[51] Xiaosong Wang, Yifan Peng, Le Lu, Zhiyong Lu, Mohammadhadi Bagheri, and Ronald M Summers. Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2097–2106, 2017.

[52] Yan Wang, Wei-Lun Chao, Kilian Q Weinberger, and Laurens van der Maaten. Simpleshot: Revisiting nearest-neighbor classification for few-shot learning. *arXiv preprint arXiv:1911.04623*, 2019.

[53] Zhu Xiaojin and Ghahramani Zoubin. Learning from labeled and unlabeled data with label propagation. *Tech. Rep., Technical Report CMU-CALD-02–107, Carnegie Mellon University*, 2002.

[54] Hui Xu, Jiaxing Wang, Hao Li, Deqiang Ouyang, and Jie Shao. Unsupervised meta-learning for few-shot learning. *Pattern Recognition*, 116:107951, 2021.

[55] Ling Yang, Liangliang Li, Zilun Zhang, Xinyu Zhou, Erjin Zhou, and Yu Liu. Dpgn: Distribution propagation graph network for few-shot learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 13390–13399, 2020.

[56] Han-Jia Ye, Lu Han, and De-Chuan Zhan. Revisiting Unsupervised Meta-Learning via the Characteristics of Few-Shot Tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2022.

[57] Han-Jia Ye, Hexiang Hu, De-Chuan Zhan, and Fei Sha. Few-shot learning via embedding adaptation with set-to-set functions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8808–8817, 2020.

[58] Tianyuan Yu, Sen He, Yi-Zhe Song, and Tao Xiang. Hybrid graph neural networks for few-shot learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 3179–3187, 2022.

[59] Hongguang Zhang, Piotr Koniusz, Songlei Jian, Hongdong Li, and Philip HS Torr. Rethinking class relations: Absolute-relative supervised and unsupervised few-shot learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9432–9441, 2021.

[60] Hongguang Zhang, Hongdong Li, and Piotr Koniusz. Multi-level second-order few-shot learning. *IEEE Transactions on Multimedia*, pages 1–1, 2022.

[61] Xueting Zhang, Debin Meng, Henry Gouk, and Timothy M. Hospedales. Shallow bayesian meta learning for real-world few-shot recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 651–660, October 2021.

[62] Imtiaz Ziko, Jose Dolz, Eric Granger, and Ismail Ben Ayed. Laplacian regularized few-shot learning. In *International conference on machine learning*, pages 11660–11670. PMLR, 2020.

# 4

# Deep Learning

Deep learning is an area of machine learning that uses Artificial Neural Networks (McCulloch and Pitts 1943) and has been applied to a wide variety of tasks such as image classification, object recognition, activity recognition, 3D depth estimation, and various natural language processing (NLP) tasks. In stark contrast to the classical machine learning approach of designing manual feature extraction methods, deep learning focuses on creating algorithms that can automatically learn to extract relevant features.

## 4.1. Deep Feedforward Networks

Deep feedforward networks, also called feedforward networks, or **multilayer perceptrons** (MLPs), are essential deep learning models. The goal of a feedforward network is to approximate some ideal function $f^\star$. For example, a classifier $y = f^\star(\boldsymbol{x})$ is a mapping between input $\boldsymbol{x}$ to a category $y$. A feedforward network has the ability to learn this mapping $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$ where $\boldsymbol{\theta}$ is a set of learnt parameters that results in the best approximation of function $f(\boldsymbol{x}; \boldsymbol{\theta}) \approx f^\star$. In other words, neural networks are function approximators.

These models are called **feedforward** because the information flows through the function that evaluates $\boldsymbol{x}$ and produces $\boldsymbol{y}$. The process of passing the input $\boldsymbol{x}$ through the function and through intermediate computations is known as **forward pass**. We then use a **loss function** to measure the difference between $f^\star$ and $f$ - that is, the difference between the ideal mapping $f^\star$ and the estimated mapping $f$. The parameters of the network, $\boldsymbol{\theta}$, are updated in a **backward pass** based on some optimisation criteria, given the guidance of the loss function.

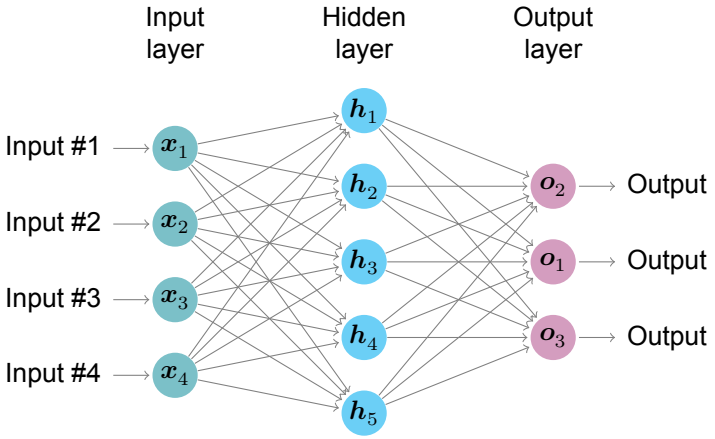



**Figure 4.1:** An MLP with a hidden layer consisting of $5$ hidden units.

Feedforward neural networks are called *networks* because they are typically composed of many different functions. The model is associated with a directed acyclic graph that describes how the functions are composed together. For example, we can have three functions $f^{(1)}, f^{(2)}$ and $f^{(3)}$ connected in a chain, to form $f = f^{(1)} \circ f^{(2)} \circ f^{(3)}$. Chain structures such as these are quite commonly used structures of neural networks. Here, $f^{(1)}$ is the **first layer**, $f^{(2)}$ is the **second layer** and so on. The final layer is called **output layer**, in this small example, $f^{(3)}$ is the output layer. The training data and training strategy determine what the output layer must produce for each given input $\boldsymbol{x}$. Intermediate layers such as $f^{(2)}$ that do not directly produce the output $\boldsymbol{y}$ are called **hidden layers**.

Each hidden layer of the network is typically vector valued. The dimensionality of these hidden layers determines **width** of the model. Each element in the vector plays a role analogous to a neuron. Instead of thinking of a layer as a vector-to-vector function, we can also think of the layer as consisting of many **units** that act in parallel (Pinker and Prince 1988), each representing a vector-to-scalar function. Figure 4.1 shows a small MLP with a single hidden layer. Each of the nodes in Figure 4.1 represents a value in a vector; this implies that the input $\boldsymbol{x}$ has a dimensionality of $4$, the hidden layer (intermediate values) has a dimensionality of $5$ and the output has a dimensionality of $3$.

Now, we must choose the form of our model, $f(\boldsymbol{x}; \boldsymbol{\theta})$. Let us choose a linear model parameterised by $\boldsymbol{\theta}$ consisting of $\boldsymbol{w}$ and $b$, where $\boldsymbol{w}$ provides the **weights** and $b$ provides the **biases** (or intercepts):

$$f(\boldsymbol{x}; \boldsymbol{\theta}) = f(\boldsymbol{x}; \boldsymbol{w}, b) = \boldsymbol{x}^\top \boldsymbol{w} + b. \tag{4.1}$$

For the model in Figure 4.1 whose hidden layer has $\boldsymbol{h}$ units, which are calculated by a function $f^{(1)}(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c})$ that is linear similar to the form of Equation (4.1). The values of these hidden units are then used as input to the second layer, which is also the output layer of the network. The output layer is also a linear function applied to $\boldsymbol{h}$ rather than to $\boldsymbol{x}$. The network now has two functions chained together $\boldsymbol{h} = f^{(1)}(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c})$ and $y = f(\boldsymbol{h}; \boldsymbol{w}, b)$, with the complete model being $f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = f^{(2)}\left(f^{(1)}(\boldsymbol{x})\right)$. It should be noted that a model parameterised by $\boldsymbol{\theta}$ implies that $\boldsymbol{\theta}$ is a collection of all the weights and biases present in the network.

Unfortunately, the complete model $f$ is a linear function of its input, since we have chosen both $f^{(1)}$ and $f^{(2)}$ to be linear. However, not all data spaces are linear. More often than not, they are non-linear in nature; accordingly, we must introduce a non-linear function to describe the data and its features. Most neural networks do so using an **affine transformation** controlled by the learned parameters (such as $\boldsymbol{\theta}$), followed by a fixed nonlinear function called an **activation function**.

## 4.2. Activation Functions

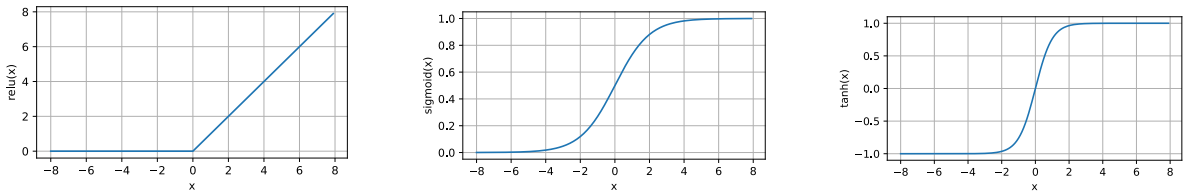
Activation functions decide whether or not a neuron (node in Figure 4.1) should be activated. More specifically, they are differentiable operators to transform inputs to outputs and add some form of non-linearity in the transformation. Formally, our hidden-layer output is now defined as:

$$\begin{aligned} \boldsymbol{h} &= g \circ f^{(1)}(\boldsymbol{x}) \\ &= g\left(\boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{c}\right) \end{aligned} \tag{4.2}$$

where $\boldsymbol{W}$ is the linear transformation, $\boldsymbol{c}$ the biases and $g(\cdot)$ is an **activation function** whose job is to introduce some non-linearity. Previously, we had used a vector of weights and a scalar bias parameter in Equation (4.1) to describe an affine transformation from a vector input to a scalar output. However, now we are describing an affine transformation from a vector $\boldsymbol{x}$ to a vector $\boldsymbol{h}$, so an entire vector of biases is needed.

One of the most widely used activation functions is **ReLU** (Rectified Linear Unit) (Fukushima 1975). This function simply returns the maximum between the given input value and zero, as given in Equation (4.3). As such, this simple function has a minimal computation cost and can be performed quite quickly.

$$\begin{aligned} g \circ f(\boldsymbol{x}) &= \max\left(0, f^{(1)}(\boldsymbol{x})\right) \\ &= \max\left(0, \boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{c}\right) \end{aligned} \tag{4.3}$$




**(a)** ReLU **(b)** Sigmoid **(c)** Tanh

**Figure 4.2:** Three common activation functions.

Another important and common function is the **Sigmoid** Equation (4.4) function. This function maps the input value to a real number between $(0, 1)$. In earlier forms of nerural networks (McCulloch and Pitts 1943), scientists where interested in neurons that either fire or do not, and so used thresholding units. A thresholding activation takes value $0$ when its input is below a certain threshold and takes value $1$ if its input is higher than the threshold. Sigmoids, on the other hand, are smooth and differentiable approximations of thresholding units; this is favourable to the gradient-based learning adopted in today's time. Sigmoids are used largely in output units when we want to interpret the output as probabilities for binary classification problems.

$$\text{sigmoid}(\boldsymbol{x}) = \sigma(\boldsymbol{x}) = \frac{1}{1 + \exp(-\boldsymbol{x})}. \tag{4.4}$$

Like the sigmoid function, the **tanh** (hyperbolic tan) function also squashes its inputs, mapping them between $-1$ and $1$:

$$\tanh(\boldsymbol{x}) = \frac{1 - \exp(-2\boldsymbol{x})}{1 + \exp(-2\boldsymbol{x})}. \tag{4.5}$$

## 4.3. Loss Function

During training, for each training sample in the training set $\{\boldsymbol{x}_i, y_i\}_{i=1}^N$ where $\boldsymbol{x}_i$ is an input vector and $y_i$ is the corresponding label, we present the input $\boldsymbol{x}$ to the neural network and compare the predicted output of the network $\hat{y}_i$ with the corresponding label $y_i$. We need to define a **loss function** to objectively measure how much the predicted output of the network $\hat{y}_i$ is different from the expected output $y_i$. For regression problems, the quadratic loss function called the mean squared error (MSE) is calculated as follows:

$$\mathcal{L}(\hat{y}, y) = \frac{1}{2}\|y - \hat{y}\|^2 \tag{4.6}$$

The loss function is calculated for each training example in the training set. The average of the calculated loss functions for all training examples in the training set is called the **cost function**. For MSE it is the average of the calculated loss functions for all training examples in the training set:

$$J(\boldsymbol{\theta}) = \frac{1}{2N}\sum_{i=1}^{N}\|y_i - \hat{y}_i\|^2 \tag{4.7}$$

where $N$ is the total number of training samples. In the following Section 4.4 we shall see another loss function that is important for multiclass classification.

## 4.4. Softmax

For convenience in the following section, we shall call the output of network $f(\boldsymbol{x}; \boldsymbol{\theta})$ as $\boldsymbol{o}$. Assume that we have a loss function such as mean squared error loss (MSE). When working with **multiclass classification** problems, where $\boldsymbol{y} \in \{1, \dots, K\}$; we expect the output of the model to be a vector of class probabilities where each value tells us the probability that a sample belongs to a particular class. Note that we generally employ the "one-hot" encoding mechanism where $\boldsymbol{y} = (0, ..., 0, 1, 0, ..., 0)$. We can try

to directly minimise the difference between $\boldsymbol{o}$ and $\boldsymbol{y}$. Although treating classification as a regression problem may work well, it still lacks in two main ways:

- There is no guarantee that $\boldsymbol{o}$ will sum up to $1$ as we have come to expect from probabilities
- There is no guarantee that $\boldsymbol{o}$ takes strictly non-negative values regardless of the sum of $\boldsymbol{o}$

Both of these issues make the problem difficult to solve in addition to making the solution highly sensitive to outliers. If we were to presume a positive linear dependency between the horsepower of a car and the probability that someone will buy it, the probability might exceed $1$ when it comes to buying a Bugatti Chiron[1]! Of course, this breaks the mathematical and logical rules of probability theory; therefore, we need a technique to map the values of $\boldsymbol{o}$ between $(0, 1)$.

One way to accomplish these goals (ensuring that the output sums up to $1$ and its values are non-negative), is to use an exponential function $P(y = i) \propto \exp o_i$. The $\exp$ function does ensure that the output will always be non-negative. We can further **normalise** these values so that they add up to $1$ by dividing each by the sum of the whole. This gives us the **softmax** function:

$$\hat{\boldsymbol{y}} = \operatorname{softmax}(\boldsymbol{o}) \quad \text{where} \quad \hat{y}_i = \frac{\exp(o_i)}{\sum_{j=1}^{K} \exp(o_j)}. \tag{4.8}$$

Note that the highest value in $\boldsymbol{o}$ corresponds to the most likely class according to $\hat{\boldsymbol{y}}$. Softmax also preserves the ordering of its arguments as shown in the example:

$$\operatorname{softmax}\left(\begin{bmatrix} 8 \\ 5 \\ 0 \end{bmatrix}\right) = \begin{bmatrix} 0.9523 \\ 0.0474 \\ 0 \end{bmatrix}.$$

### 4.4.1. Softmax and Cross-Entropy Loss

The softmax function gives us a vector $\hat{\boldsymbol{y}}$, which we can interpret as (estimated) conditional probabilities of each class, given any input $\boldsymbol{x}$, like $P(y = \text{class}|\boldsymbol{x})$. The cross-entropy loss measures the difference between two probability distributions; in this case, it measures the difference between $\hat{\boldsymbol{y}}$ and $\boldsymbol{y}$. The cross-entropy loss is given as:

$$\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_{j=1}^{K} \boldsymbol{y}_j \log \hat{\boldsymbol{y}}_j. \tag{4.9}$$

Note that the loss defined in Equation (4.9), has a lower bound of $0$ when $\hat{\boldsymbol{y}}$ is a probability vector, that is, no single entry is greater than $1$, and hence its negative logarithm cannot be lower than $0$. For multiclass classification problems, the cost function is calculated as follows:

$$J(\boldsymbol{\theta}) = -\frac{1}{N} \sum_{j=1}^{N} \sum_{i=1}^{K} \boldsymbol{y}_{\boldsymbol{i}}^{(\boldsymbol{j})} \log(\hat{\boldsymbol{y}}_{\boldsymbol{i}}^{(\boldsymbol{j})}) \tag{4.10}$$

where $\boldsymbol{y}_{\boldsymbol{i}}^{(\boldsymbol{j})}$ and $\hat{\boldsymbol{y}}_{\boldsymbol{i}}^{(\boldsymbol{j})}$ are the true and predicted output for the $j^{\text{th}}$ input sample $\boldsymbol{x}_j$. To better understand the origins of the cross-entropy loss, please refer to A. Zhang et al. (2021).

### 4.4.2. Softmax vs Sigmoid

The softmax and sigmoid functions are similar, as they help squeeze the network output within a certain range. The softmax function operates on a vector input, while the sigmoid takes a scalar value as input. In fact, the sigmoid function is a special case of the softmax function for a classifier with only two input classes (binary classification problem). We can show this if we set the input vector to $\boldsymbol{z} = [x, 0]$ and

[1] https://www.bugatti.com/models/chiron-models/

calculate the first output element with the usual softmax formula:

$$\begin{aligned}
\text{softmax}(\boldsymbol{z})_1 &= \frac{\exp(z_1)}{\exp(z_1) + \exp(z_2)} \\
&= \frac{\exp(x)}{\exp(x) + \exp(0)} \\
&= \frac{\exp(x)}{\exp(x) + 1} && \text{divide numerator and denominator by } \exp(x) \\
&= \frac{1}{1 + \exp(-x)}
\end{aligned}$$

## 4.5. Universal Approximation Theory

MLPs are of extreme importance in deep learning, as they form the basis for many applications and advanced neural architectures. To do this, MLPs and neural networks, in general, must be able to approximate a wide range of functions, given enough input data to learn from. Neural networks started as a way to mimic the neural structure of the brain (McCulloch and Pitts 1943), and as we know our brain is capable of complex statistical analysis, among other things. As such, a worthwhile question to ask here is *just how powerful can a deep neural network be?* Fortunately for us, this question has been answered several times in the context of MLPs and radial basis functions (RBF) (Micchelli 1986; Cybenko 1989; Hornik 1991). These works suggest that even with a single hidden layer, given enough nodes and the right set of parameters $\boldsymbol{\theta}$, we can potentially model any function. In simple terms, having enough nodes and stacking enough affine transformations followed by non-linear transforms repetitively, a neural network should be able to model a highly rich class of functions. Although neural networks are capable of expressing arbitrary continuous functions, it may not be easy to learn the function and its parameters.

## 4.6. Optimisation and Backpropagation

Until now, we have only discussed the forward pass of a neural network, where the network processes the input $\boldsymbol{x}$. However, to effectively approximate $f^\star$, a neural network must learn an ideal set of values for the parameters, $\boldsymbol{\theta}$. To do so, we must **optimise** the weights so that they are suitable for the task and the data at hand. However, the optimisation algorithms used to train deep learning models differ from traditional optimisation algorithms in multiple ways. In most machine tasks, we are concerned with some performance measure $P$, which is defined with respect to the test set and may very well be intractable. Therefore, we optimise $P$ indirectly through a cost function $J(\boldsymbol{\theta})$.

Typically, the cost function is written as an average over the training set (see Section 4.3):

$$\begin{aligned}
J(\boldsymbol{\theta}) &= \mathbb{E}_{(\boldsymbol{x},\mathrm{y})\sim\hat{p}_{\text{data}}} \mathcal{L}(f(\boldsymbol{x};\boldsymbol{\theta}), y) \\
&= \mathbb{E}_{(\boldsymbol{x},\mathrm{y})\sim\hat{p}_{\text{data}}} \mathcal{L}(\hat{y}, y)
\end{aligned} \tag{4.11}$$

where $\mathcal{L}$ is the loss function per sample, $\hat{y} = f(\boldsymbol{x};\boldsymbol{\theta})$ is the predicted output, the input is $\boldsymbol{x}$ and $\hat{p}_{\text{data}}$ is the empirical (observed) distribution. In the straightforward supervised case, $y$ is the true target output. It takes minimal effort to adapt this formulation to exclude $y$ as an argument, as we shall see this in Chapter 7.

### 4.6.1. Gradient Descent

We begin by assuming that we have a function $y = f(x)$ where $x, y \in \mathbb{R}$ and that it has a derivative $f'(x)$. The derivative of a function gives its slope, also called **gradient** at a certain input point $x$. The gradient indicates how adding a small change $\epsilon$ to $x$ will affect the output: $f(x + \epsilon) \approx f(x) + \epsilon f'(x)$. Gradient descent is a first-order iterative optimisation algorithm used to find the local minimum or maximum of a function (Robbins and Monro 1951; Kiefer and Wolfowitz 1952). The optimisation algorithm works by iteratively moving in the direction of steepest descent, as defined by the negative of the gradient.

In Figure 4.3 the function $f(x) = \frac{1}{2}x^2$ depends only on the input $x$ and has a derivative $f'(x) = x$. In order to find the minimum of $f(x)$ we must find the value of $x$ where $f'(x) = 0$. The useful property of

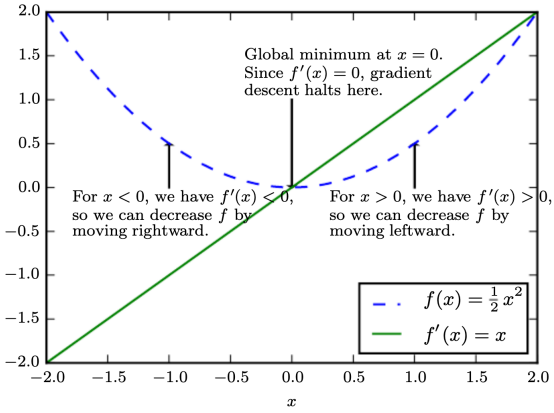



**Figure 4.3:** Gradient descent algorithm illustration for finding the minimum of a function $f(x) = \frac{1}{2}x^2$. Beginning with either positive of negative values of $x$, the gradient descent algorithm will find a minima at $x = 0$ by moving in the direction opposite the sign of the gradient given by $f'(x) = x$. Figure courtesy of Goodfellow, Bengio, and Courville (2016).

the gradient is that it tells us how to change $x$ to make a small change to $y$. For instance, we can infer that $f\left(x - \epsilon \operatorname{sign}\left(f'(x)\right)\right) \ll f(x)$ for a small $\epsilon$. Therefore, to reduce $f(x)$, we can begin this process by starting with positive or negative values of $x$ and moving in the direction opposite to the sign of the gradient $f'(x)$. The process is known as **gradient descent**, more specifically this update step reads:

$$\begin{aligned} x' &= x - \epsilon \operatorname{sign}\left(f'(x)\right) \\ &= x - \epsilon \operatorname{sign} \nabla_x f(x) \end{aligned} \tag{4.12}$$

### 4.6.2. Backpropagation

The backpropagation as a learning algorithm was put forth by Rumelhart, Hinton, and Williams (1986), and is firmly based on the concept of gradient descent. We discussed gradient descent in the context of a simple function $f(x)$ that was dependent only on its input $x$, however, neural networks depend on the input $\boldsymbol{x}$ and a set of parameters $\boldsymbol{\theta}$, $f(\boldsymbol{x}; \boldsymbol{\theta})$. Furthermore, the learning goal of neural networks is not to directly minimise the function $f(\boldsymbol{x}; \boldsymbol{\theta})$. Rather, the goal of learning in neural networks is to minimise the cost function (or loss function) given the training set.

Consider a network parameterised by $\boldsymbol{\theta} = \{\boldsymbol{w}, b\}$. The goal of backpropagation is to calculate partial derivatives $\partial J / \partial \boldsymbol{w}$ and $\partial J / \partial b$ of the cost function $J$ with respect to weight $\boldsymbol{w}$ and bias $b$. To further concretize the notion, we shall use the MSE loss from Section 4.3:

$$\mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{2} \|\boldsymbol{y} - \hat{\boldsymbol{y}}\|^2 \tag{4.13}$$

$$J(\boldsymbol{\theta}) = \frac{1}{2N} \sum_{i=1}^{N} \mathcal{L}(f(\boldsymbol{x}; \boldsymbol{\theta}), \boldsymbol{y}) \tag{4.14}$$

where $N$ is the total number of training samples. To calculate $\partial J / \partial \boldsymbol{w}$ and $\partial J / \partial b$, we apply the **chain rule** and calculate, in turn, the gradient of each intermediate variable and parameter. The first step is to calculate $\partial J / \partial \mathcal{L}$ which are the gradients of the cost function in Equation (4.14) with respect to the loss term $\mathcal{L}$. Next, we must compute the gradient of the cost function with respect to the output variable $\hat{\boldsymbol{y}}$ according to the chain rule:

$$\frac{\partial J}{\partial \hat{\boldsymbol{y}}} = \frac{\partial J}{\partial \mathcal{L}} \times \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{y}}}. \tag{4.15}$$

Now we can calculate the gradient $\partial J / \partial \boldsymbol{w}$, using the chain rule this yields:

$$\frac{\partial J}{\partial \boldsymbol{w}} = \left( \frac{\partial J}{\partial \hat{\boldsymbol{y}}} \times \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{w}} \right). \tag{4.16}$$

where $\partial\hat{\boldsymbol{y}}/\partial\boldsymbol{w} = \boldsymbol{x}$. Notice that the order of calculations are now **reversed** relative to those performed in the forward pass. Here we start with the output emitted by the neural network and work our way backwards towards the parameters. We can repeat the above steps for computing $\partial J/\partial b$. Intuitively, this tells us how to update $\boldsymbol{w}$ and $b$ in order to reduce $J(\boldsymbol{\theta})$.

We must also be aware of the fact that we optimise for the loss function $\mathcal{L}$, that is, we are trying to find the optimal set of parameters $\boldsymbol{\theta}$ that will give us the lowest value of the loss function across $N$ training samples. In other words, we are trying to find the minima of $\mathcal{L}$ with respect to the input $\boldsymbol{x}$ and the parameters $\boldsymbol{\theta}$. The gradient $\nabla_{\boldsymbol{\theta}} J$ across all samples is given as:

$$\nabla_\theta J(\boldsymbol{\theta}) = \frac{1}{2N}\sum_{i=1}^{N} \nabla_{\boldsymbol{\theta}} \mathcal{L}(f(\boldsymbol{x};\boldsymbol{\theta}), y) \tag{4.17}$$

where $\nabla_\theta J(\boldsymbol{\theta})$ is the average of the gradients of the loss function with respect to $\boldsymbol{x}$ and $\boldsymbol{\theta}$. The parameter update using the gradient descent algorithm with step size $\eta$ can be written as

$$\boldsymbol{\theta}' = \boldsymbol{\theta} - \eta \nabla_\theta J(\boldsymbol{\theta}). \tag{4.18}$$

However, this method is intractable for very large values of $N$. Therefore, we choose to compute gradients for a smaller set of $m$ samples such that $m \ll N$:

$$\nabla_\theta J(\boldsymbol{\theta}) = \frac{1}{2m}\sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} \mathcal{L}(f(\boldsymbol{x};\boldsymbol{\theta}), y). \tag{4.19}$$

This is the idea of stochastic approximation behind **stochastic gradient descent** (SGD). It is an approximation of the gradient from a small number of input samples (Bottou, Curtis, and Nocedal 2016). The SGD update step in Equation (4.19) is similar to Equation (4.18).

Finally, for a network with $l$ layers, there are just as many gradients with respect to the loss function $\mathcal{L}$:

$$\boldsymbol{\nabla}_{\boldsymbol{\theta}} \mathcal{L}(f(\boldsymbol{x};\boldsymbol{\theta}), y) = \left[\frac{\partial \mathcal{L}}{\partial w^{(l)}}, \frac{\partial \mathcal{L}}{\partial b^{(l)}}, \frac{\partial L}{\partial w^{(l-1)}}, \frac{\partial \mathcal{L}}{\partial b^{(l-1)}}, \ldots, \frac{\partial \mathcal{L}}{\partial w^{(1)}}, \frac{\partial \mathcal{L}}{\partial b^{(1)}}\right]^\top \tag{4.20}$$

where $\boldsymbol{\nabla}_{\boldsymbol{\theta}} L(f(\boldsymbol{x};\boldsymbol{\theta}), y)$ is a vector of partial derivatives with respect to the weights in the neural network model while $w^{(l)}$ and $b^{(l)}$ are the weights and biases of layer $l$.

# 5

# Convolutional Neural Networks

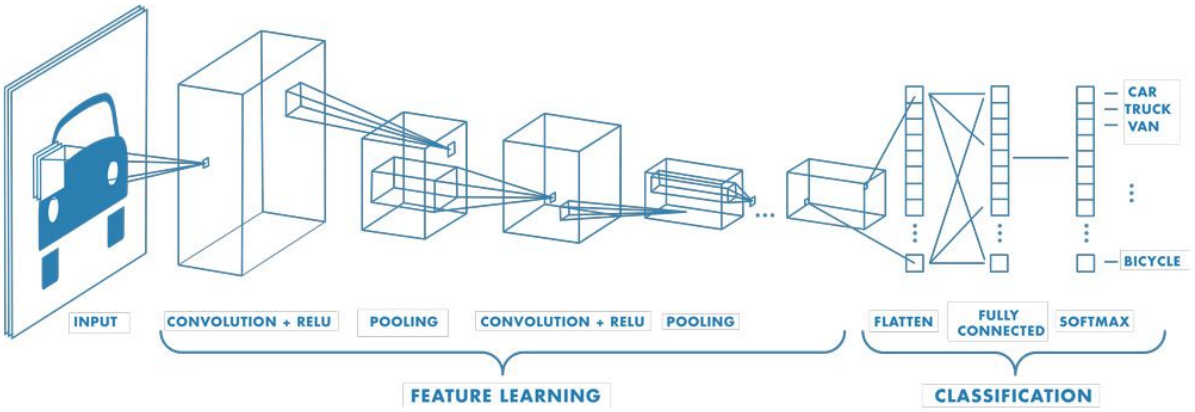



**Figure 5.1:** An overview of the internals of CNNs. Image sourced from Matlab[1]

LeCun et al. proposed the novel **convolutional neural network** (CNN) architecture to tackle computer vision tasks. The specific task LeCun et al. chose to solve was the recognition of handwritten digits (today a similar dataset is the MNIST[2] dataset). The LeNet-$5$ (LeCun et al. 1989) was inspired by Neocognitron (Fukushima 1975; Fukushima and Miyake 1982) that allowed extraction of **features** such as certain orientations, edges, and corners. The LeNet-$5$ was designed to use backpropagation for learning and proved backpropagation allowed neural nets to be used in the real world. Yann LeCun estimates that at one point this system was being used to read more than $10\%$ of all cheques in the U.S.A (Chow 2021).

CNNs were originally inspired by early findings in the study of natural biological vision. They have since become the tools of choice in computer vision and as models of neural activity in visual tasks (Eickenberg et al. 2017; Kuzovkin et al. 2018; Lindsay 2021). The CNN architecture and its derivatives continue to be one of the best performing models (alongside newly minted vision transformers (Dosovitskiy et al. 2020)) for computer vision tasks ranging from image classification to 3D depth estimation.

Although CNNs were first introduced in $1989$, it was AlexNet (Krizhevsky, Sutskever, and Hinton 2012) that really launched CNN-based models into the spotlight. AlexNet outperformed its competitors by about $10\%$ on the ImageNet (Deng et al. 2009) benchmark. Surpassing its competitors by such a great margin, AlexNet and CNNs emerged as the dominant force in computer vision problems.

[1] https://www.mathworks.com/discovery/convolutional-neural-network-matlab.html
[2] http://yann.lecun.com/exdb/mnist/

An archetypal CNN (Figure 5.1) is made up of two major components, (i) a **feature extractor** and (ii) a **classification head**. The feature extractor portion is made up of multiple convolutional layers, followed by non-linear activation functions (Section 4.2) and finally some form of **pooling** is applied. The classification head consists of an MLP that uses the output of a feature extractor, followed by the application of the **softmax** function to get the class probabilities. We shall go over each of the operations in the following sections.

## 5.1. Convolution

As the name implies, the basic building block of a CNN is a **convolution operator**, which we denote by an asterisk $*$. In its most generic form, a convolution is an operation on two functions of a real-valued argument. Suppose that we have a water-level sensor installed in a massive fish tank. Our sensor provides a single output $x(t)$, the level of water in the tank at the time $t$. Now, our sensor is not perfect and is somewhat noisy. To decrease the noise in the measurements, we take averages of several measurements. The more recent the measurements, the more relevant they would be, so we would rather want this to be a weighted average that gives more importance to recent measurements. We can achieve this by using a weighting function $w(a)$, where $a$ is the age of a given measurement. Applying the weighting function produces a new function called $S$, which provides a smooth estimate of the water level in the tank:

$$s(t) = \int x(a)w(t-a)da. \tag{5.1}$$

This operation is called **convolution** and Equation (5.1) is often succinctly written as:

$$s(t) = (x * w)(t). \tag{5.2}$$

In Equation (5.1) and Equation (5.2) $x$ is referred to as the **input**, the function $w$ is called the **kernel**. In the terminology of the convolutional network, the output is termed **feature map**.

We can also use convolutions over multiple axes simultaneously. If we choose a two-dimensional image $\boldsymbol{I}$ as input, it stands to reason that we would probably want to use a two-dimensional kernel $\boldsymbol{K}$ over it:

$$\boldsymbol{S}(i,j) = (\boldsymbol{I} * \boldsymbol{K})(i,j) = \sum_m \sum_n \boldsymbol{I}(m,n)\boldsymbol{K}(i-m, j-n). \tag{5.3}$$

As convolution is a commutative operation, we can rewrite Equation (5.3) for a convenient implementation:

$$\boldsymbol{S}(i,j) = (\boldsymbol{K} * \boldsymbol{I})(i,j) = \sum_m \sum_n \boldsymbol{I}(i-m, j-n)\boldsymbol{K}(m,n). \tag{5.4}$$

The commutative property of convolution appears as we have **flipped** the kernel relative to the input, in the sense that as $m$ increases, the index into the input increases, but the index into the kernel decreases. While useful for mathematical proofs, its not necessary to do in terms of actual implementation. Instead, many neural network libraries implement the **cross-correlation** function, which, for all intents and purposes, is the same as a convolution without the kernel being flipped:

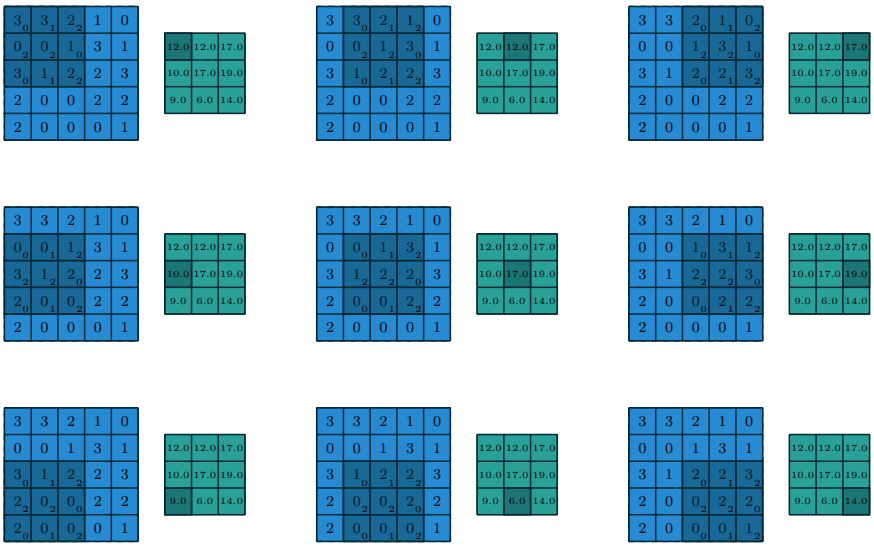
$$\boldsymbol{S}(i,j) = (\boldsymbol{K} * \boldsymbol{I})(i,j) = \sum_m \sum_n \boldsymbol{I}(i+m, j+n)\boldsymbol{K}(m,n). \tag{5.5}$$

| | | |
|---|---|---|
| 0 | 1 | 2 |
| 2 | 2 | 0 |
| 0 | 1 | 2 |

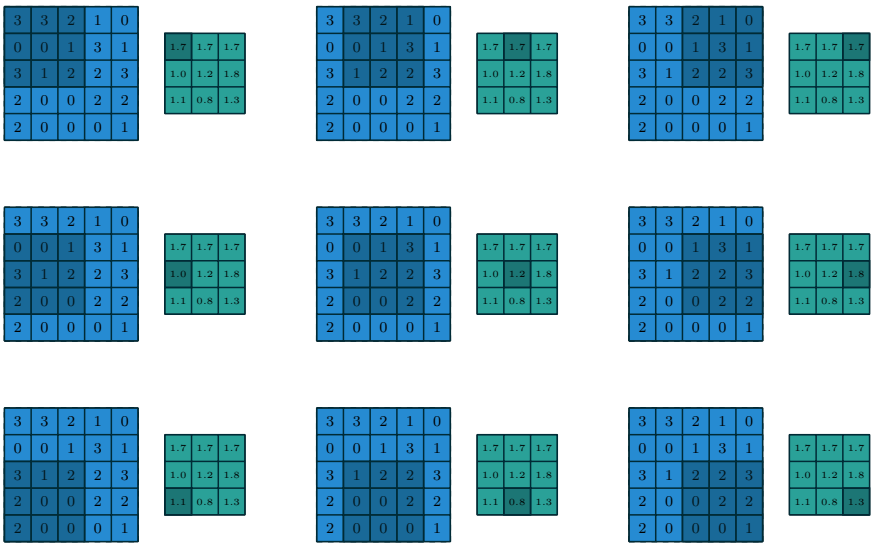
**Figure 5.2:** A two-dimensional kernel.

An example of a two-dimensional *kernel* is shown in Figure 5.2, and Figure 5.3a shows an example of a discrete convolution using the kernel in Figure 5.2. The blue grid is the two-dimensional input, also called the **input feature map**. To keep the drawing simple, a single input feature map is shown; however, it is quite common to have multiple feature maps stacked on top of each other. The

kernel slides over the input feature map and, at each location, we take a product between each element of the kernel and the input element that coincides with it. The results of the products are summarised to obtain the output at the current location (the centre of the kernel). This process can be repeated using different kernels to generate as many **output feature maps** as required (Figure 5.4). It must be noted that the kernel can be learnt through back-propagation.



**(a)** Computing the output values of a discrete convolution.



**(b)** Computing the output values of a $3 \times 3$ average pooling operation on a $5 \times 5$ input using $1 \times 1$ strides.

**Figure 5.3:** Examples of convolution and average pooling operations (Dumoulin, Visin, and Box 2016).

## 5.2. Pooling

In addition to the convolution operation described in Section 5.1, **pooling** operations make up another important aspect of a CNN. Pooling operations are meant to reduce the size of the feature maps by using some function to **summarise** regions of the feature map, such as taking the average or maximum value of a collection of values in a feature map present in a window.

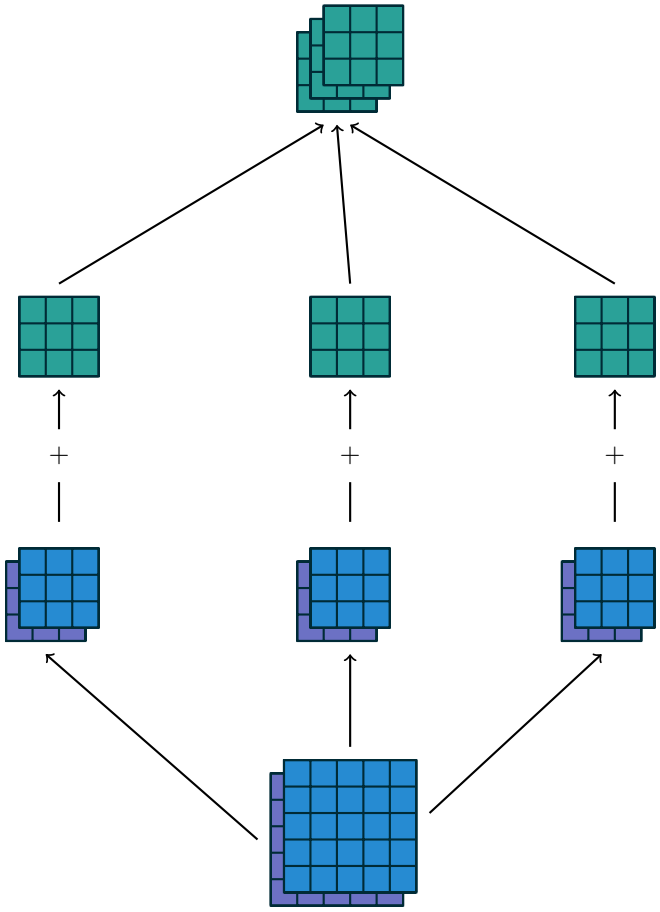


**Figure 5.4:** A convolution mapping from two input feature maps to three output feature maps using a $3 \times 2 \times 3 \times 3$ collection of kernels $\mathbf{w}$. In the left pathway, input feature map 1 is convolved with kernel $\mathbf{w}_{1,1}$ and input feature map 2 is convolved with kernel $\mathbf{w}_{1,2}$, and the results are summed together elementwise to form the first output feature map. The same is repeated for the middle and right pathways to form the second and third feature maps, and all three output feature maps are grouped together to form the output. Figure courtesy Dumoulin, Visin, and Box (2016).

Pooling operations work similarly to a convolution by sliding a window across the input and applying a pooling function to the contents of the window. Pooling operations replace the linear combination described by the kernel with some other function. Figure 5.3b shows an example of the **average pooling** operation, which applies the averaging function on the contents of the sliding window. Similarly, **maximum pooling** is another pooling operation that takes the maximum of the content present in the sliding window.

## 5.3. Kernels as Feature Extractors

Each convolutional layer is characterised by the size of the kernels, the number of kernels, the padding, and **stride**; the relationship between these components is not trivial to intuitively infer. Every kernel's weights act as a particular feature detector, and thus learning the kernel's weights translates into learning the features. Figure 5.5 shows the types of features and patterns learnt by a fully trained model in various layers. Note that as the number of layers increases, the network is learning more specific and complex patterns. In layer $5$ we can see the network as learnt the shape of a human head as well those of digs and other animals. While in layer $2$, we see that some generic lines and other shapes have been learnt.


**Figure 5.5:** Visualisation of features learnt in various layers by a fully trained CNN. Image from Zeiler and Fergus (2013).

# 6

# Basics of Geometric Deep Learning and Graph Neural Networks

## 6.1. Graphs

In various branches of science, from biology to particle physics, graphs are often used as models of systems of relations and interactions. In this work, graphs are important as they engender a fundamental type of permutation invariance.

A basic **graph** $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a collection of **nodes** $\mathcal{V}$ and **edges** $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ between pairs of nodes. Depending on the application or field, the nodes can also be called **vertices**, and the edges can be called **links** or **relations**.

For the purpose of an intuitive explanation, we assume that the nodes are associated with $s$-dimensional *node features*, denoted by $\boldsymbol{x}_u$ for all $u \in \mathcal{V}$. Social networks are one of the most studied and best examples of graphs in the real world. Here, nodes represent users, edges correspond to friendship relations between them, and node features ($\boldsymbol{x}_u \in \mathbb{R}^d$) model user information such as age, location, last active time, etc. It should be noted that it is often possible to equip edges with features.

A key property of graphs is that nodes in $\mathcal{V}$ are not typically assumed to be provided in any particular ordering, and thus, by extension, any operations performed on graph structures should not depend on any assumed inherent ordering of nodes. The desirable property that functions acting on graphs must possess is called **permutation invariance**, and implies that for any two **isomorphic** graphs, the results of such functions must be identical. *Isomorphism* is an edge-preserving bijection between two graphs. The two isomorphic graphs shown in Figure 6.1 are identical up to the reordering of their nodes.

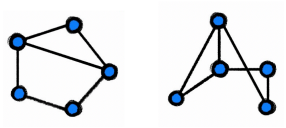


**Figure 6.1:** Two isomorphic graphs.

Permutation invariance is important in `SAMPTransfer`(see Chapter 3). One of the main motivations behind `SAMPTransfer` is to allow the network to learn from multiple samples made available to the network in the form of a mini-batch. The ordering of the samples is of little interest to us, as we are more concerned with learning by looking beyond single instances. The exact mechanism of this will be explained in more detail in subsequent sections.

## 6.2. Janossy Pooling

Let us first illustrate the concept of permutation invariance on **sets**, a special case of graphs without edges ($\mathcal{E} = \emptyset$). We illustrate this with the help of a framework provided to us in the form of Janossy pooling (Murphy et al. 2018).

Currently, there are a few ways to incorporate permutation invariance. The first is the "permute and average" paradigm. It works by considering all possible permutations of input elements, then passing each permutation through a permutation-sensitive model or function, and then taking an average of all the results. If two inputs $\boldsymbol{X}$ and $\tilde{\boldsymbol{X}}$ are permutations of each other, this process will give the same result for both inputs. Mathematically, this can be written as follows:

$$\hat{f}(\mathbf{X}) = \frac{1}{|T_n|} \sum_{\pi \in T_n} \phi(\pi(\mathbf{X})) \tag{6.1}$$

where $\boldsymbol{X} = [\boldsymbol{x}_1^\top, \ldots, \boldsymbol{x}_n^\top]^\top$ is a stack of node features and has $n$ elements with $d$ dimensionality, $T_n$ is the set of all permutations of $n$ elements, and $\phi$ is a permutation-sensitive model. Therefore, we construct a permutation-invariant $\hat{f}$ from a permutation-sensitive $\phi$.

This idea is called **Janossy pooling** and was first introduced by Murphy et al. (2018). Indeed, it is a straightforward and easy way to achieve permutation invariance, but it happens to be very computationally intensive. The computational cost is dominated by the sum over $T_n$ in Equation (6.1), where the number of elements in $T_n$ is $n!$. As one can imagine, if $\boldsymbol{X}$ is a mini-batch with very large $n$, Equation (6.1) quickly becomes intractable.

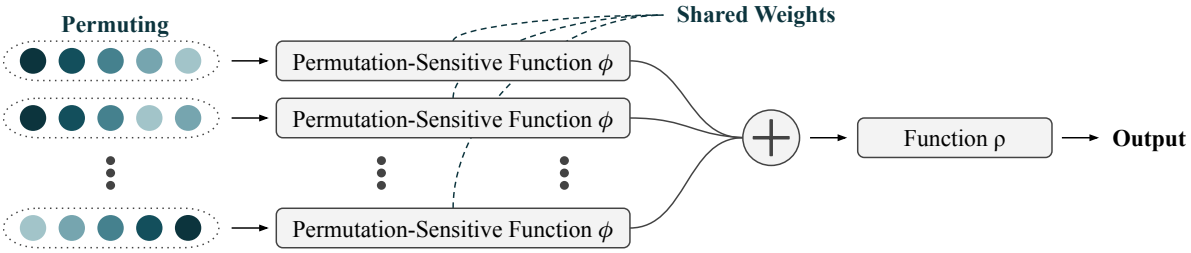



**Figure 6.2:** $\hat{f}$ comprises everything up to and including the sum. The function $\rho$ applied after the sum is optional and does not need to follow any constraints to guarantee invariance because its input, $\hat{f}(x)$, is already invariant. Image courtesy of Wagstaff et al. (2022).

### 6.2.1. A More Efficient Janossy Pooling

To save computational costs, we may give up the calculation of $n!$ permutations. In the above Figure 6.2, we look at all possible $n$-tuples from the set of $n$ elements; instead, we can consider all $k$-tuples where $k < n$ (Murphy et al. 2018). We can then update Equation (6.1) as:

$$\hat{f}(\boldsymbol{X}) = \frac{1}{|P(n,k)|} \sum_{\boldsymbol{X}_{\{k\}}} f(\boldsymbol{X}_{\{k\}}) \tag{6.2}$$

where $\boldsymbol{X}_{\{k\}}$ stands for a $k$-tuple of $\boldsymbol{X}$.

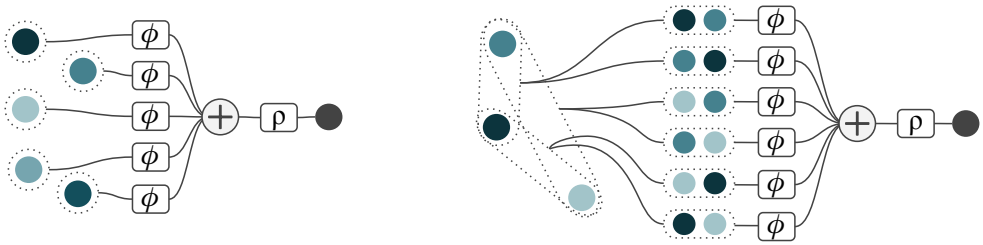
Let us take a short example to understand how the count of values came about in Equation (6.2). Consider the input set $w, x, y, z$ with $n = 4$. When we set $k = 2$, then our sum will be applied over all 2-tuples:

$$(w,x),\ (x,w), (w,y),\ (y,w),\ (w,z),\ (z,w), (x,y),\ (y,x),\ (x,z),\ (z,x),\ (y,z),\ (z,y)$$

A sum over all these tuples is clearly invariant to permutations of elements in the input set, that is, each tuple appears exactly once in the sum no matter the order of individual elements. Therefore, the number of $k$-tuples from a set of $n$ elements is:

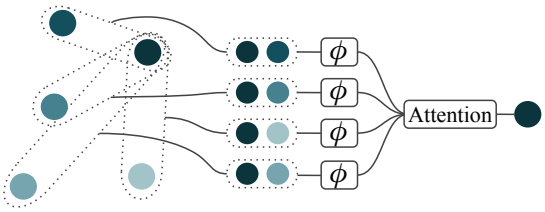
$$P(n,k) = \frac{n!}{(n-k)!} \tag{6.3}$$

For $k \ll n$, this gives us a much more tractable method. Setting $k = n$ gives us the most expressive and most expensive model. Setting $k = 1$ gives us a model whose cost is linear in the size of the input set. Increasing $k$ allows us to take into account higher-order interactions between elements in the set; we will come back to this idea of interactions in Section 6.2.3.




**(a)** Janossy pooling with $k = 1$ (*Deep Sets*)

**(b)** Janossy pooling with $k = 2$




**(c)** Self-attention

**Figure 6.3:** Different variations of Janossy pooling. Permutation invariance is guaranteed by processing all combinations of $k$ elements and then aggregating via a sum (or softmax in the case of attention). Self-attention, a variant of Janossy pooling with $k = 2$, focuses on one node at a time (the darkest node here), computing an output for this specific node. It is often employed for all nodes in parallel in a permutation-equivariant manner, mapping sets of points to sets of points (J. Lee et al. 2019). Image borrowed from Wagstaff et al. (2022).

### 6.2.2. Deep Sets

When $k = 1$ and we include the optional function $\rho$, we obtain a popular special case known as Deep Sets (Zaheer et al. 2017). Zaheer et al. propose a neural architecture in which each input is first transformed by a neural network model $\phi$ individually (see Figure 6.3a), then the results are aggregated through a $\mathrm{sum}$ operator and further processed by a second neural network $\rho$.

The aforementioned functions produce a "global" graph-wise result, but we are frequently more interested in functions that work "locally" or node-wise. In order to acquire the collection of latent node features, for instance, we would wish to use some function to *update* the features in each node. In other words, we may wish for a **node-wise** function to be permutation equivariant and want a permutation invariant **graph-wise** function.

We can now generalise the notions of permutation-invariance and equivariance from sets to graphs. In the generic setting $\mathcal{E} \neq \emptyset$, the graph connectivity can be represented by a $n \times n$ **adjacency matrix** $\boldsymbol{A}$, defined as

$$a_{uv} = \begin{cases} 1 & (u, v) \in \mathcal{E} \\ 0 & \text{otherwise} . \end{cases} \tag{6.4}$$

Note that now the adjacency and input (or feature) matrices $\boldsymbol{A}$ and $\boldsymbol{X}$ are "synchronised", $a_{u,v}$ specifies the adjacency information between the nodes described by the $u^{\text{th}}$ and $v^{\text{th}}$ rows of $\boldsymbol{X}$. Therefore, applying a permutation matrix $\boldsymbol{P}$ to node features $\boldsymbol{X}$ directly implies applying it to $\boldsymbol{A}$'s rows and columns, $\boldsymbol{P}\boldsymbol{A}\boldsymbol{P}^\top$. We then say that a graph-wise function $f$ is permutation invariant if

$$f\left(\boldsymbol{P}\boldsymbol{X}, \boldsymbol{P}\boldsymbol{A}^\top\right) = f(\boldsymbol{X}, \boldsymbol{A}) \tag{6.5}$$

and a node-wise function $\rho$ is permutation equivariant if

$$\rho\left(\boldsymbol{P}\boldsymbol{X}, \boldsymbol{P}\boldsymbol{A}\boldsymbol{P}^\top\right) = \boldsymbol{P}\rho(\boldsymbol{X}, \boldsymbol{A}) \tag{6.6}$$

### 6.2.3. Modelling Relations and Interactions

`SAMPTransfer` has yet another fundamental idea that is essential for its success and operation. It is able to exploit the interactions (or relations) between elements in a mini-batch, which really means that we are trying to capture the fact that our model output may depend not only on the individual contribution of each element, but it may be crucial to also take into account the fact that certain elements appear together in the same set. Our goal is simply to model and use these interactions.

To illustrate this with a simple example, imagine the task of evaluating how well a set of ingredients work together to cook a meal. Setting $k = 1$ allows the function $\phi$ to consider related individual attributes, but cannot detect conflicts between ingredients (e.g., garlic vs. caramel). A larger $k$ allows $\phi$ to see more than one item at a time, allowing it to make relational inferences about pairs of ingredients, allowing for a more expressive model of deliciousness. Considering $\phi$ as an encoder and $\rho$ as a decoder, $\phi$ can encode information about interactions, which $\rho$ can use for decoding.

Similar to the above example, `SAMPTransfer` uses contrastive learning (see Section 7.3) in order to learn to place semantically similar images closer together in an embedding space. When we increase $k$, we give our model the expressivity to learn to recognise semantic similarities between images and to explicitly take these relations into account. By allowing the model to consider relations between images in a mini-batch, it can better decide which images belong closer together in the feature space and in turn learn better discriminative semantic features. Bear in mind that the image is first passed through a CNN (see Chapter 5), and all the other operations explained here are applied to the flattened output of the CNN.

## 6.3. Janossy Pooling and Self-Attention

Many of the current neural architectures resemble Janossy pooling with $k = 2$. **Self-attention** (Vaswani et al. 2017) is one such algorithm that is also used in the context of graphs in `SAMPTransfer`. Self-attention works by comparing two elements of a set at a time, usually by performing a scalar product. The results of the scalar products are used as **attention weights** to aggregate information from different points using a weighted permutation invariant sum. Although it is similar to binary ($k = 2$) Janossy pooling, self-attention also uses $\mathrm{softmax}$ to ensure that the attention weights sum up to $1$ (see Section 4.4).

To substantiate the relationship between permutation-invariant binary Janossy pooling and self-attention, we first define a natural extension of $k$-ary Janossy pooling from permutation *invariance to equivariance*. In normal binary Janossy pooling, the function $\phi$ acts on every two-tuple, followed by the $\mathrm{sum}$-pooling operation. For the purpose of visualisation and intuition, we will write these two-tuples as a matrix:

$$\begin{bmatrix} \phi(\boldsymbol{x}_1, \boldsymbol{x}_1) & \phi(\boldsymbol{x}_2, \boldsymbol{x}_1) & \cdots & \phi(\boldsymbol{x}_n, \boldsymbol{x}_1) \\ \phi(\boldsymbol{x}_1, \boldsymbol{x}_2) & \phi(\boldsymbol{x}_2, \boldsymbol{x}_2) & \cdots & \phi(\boldsymbol{x}_n, \boldsymbol{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi(\boldsymbol{x}_1, \boldsymbol{x}_n) & \phi(\boldsymbol{x}_2, \boldsymbol{x}_n) & \cdots & \phi(\boldsymbol{x}_n, \boldsymbol{x}_n) \end{bmatrix} \tag{6.7}$$

If we pool over the *entire matrix* we get an invariant output. However, by pooling over *each row individually* we get a permutation equivariant output. In general, we define $k$-ary equivariant Janossy pooling as follows: the $i^{\text{th}}$ output is obtained by aggregating the overall ($\phi$-transformed) $k$-tuples which start with the element $\boldsymbol{x}_i$. A second network $\rho$ may then further transform each output individually. Mathematically, this reads as follows:

$$\boldsymbol{z}_i = f_i(\boldsymbol{x}) = \rho\left(\sum_j \phi(\boldsymbol{x}_i, \boldsymbol{x}_j)\right). \tag{6.8}$$

In practise, however, the process is slightly more involved. Typically, $\phi(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is divided into **attention weights** $a(\boldsymbol{x}_i, \boldsymbol{x}_j)$ and **values** $v(\boldsymbol{x}_j)$ and a softmax is applied on the attention weights. Furthermore, the attention weights themselves would be born from the scaled dot-product between two affine transformations called the **query** and **key**. However, the elegant view from the lens of binary

Janossy pooling remains clear. We shall come back to self-attention and its usage in `SAMPTransfer` in later sections.

### 6.3.1. Query, Key and Value

We have already alluded to queries, keys, and values in Equation (6.8), these three entities form the major components of self-attention. These entities are also what makes self-attention based GNNs more expressive than the standard attention scheme (Equation (6.16)) (Dwivedi and Bresson 2020; Brody, Alon, and Yahav 2021; Bronstein et al. 2021).

Every graph attention layer uses each input node $\boldsymbol{x}_i$ in three different ways:

- It is compared to every other vector to establish the weights for its own output $\boldsymbol{z}_i$
- It is compared to every other vector to establish the weights for the output of the $j^{\text{th}}$ vector $\boldsymbol{z}_j$
- It is used as part of the weighted sum to compute each output vector once the weights have been established.

These roles are called **query**, **key**, and **value**, respectively. Each input node $\boldsymbol{x}_i$ is linearly transformed three different times to derive a new vector for each of the roles. In other words, we use three $d \times d$ weight matrices $\boldsymbol{W}_q$, $\boldsymbol{W}_k$ and $\boldsymbol{W}_v$ to compute three linear transformations:

$$\boldsymbol{q}_i = \boldsymbol{W}_q \boldsymbol{x}_i \quad \boldsymbol{k}_i = \boldsymbol{W}_k \boldsymbol{x}_i \quad \boldsymbol{v}_i = \boldsymbol{W}_v \boldsymbol{x}_i \tag{6.9}$$

these transformations allow the attention module a fair degree of flexibility, allowing it to project the input to suit the roles they must be a part of.

The softmax function can be sensitive to very large input values. These kill the gradient, slow down learning, or cause it to stop altogether. Since the average value of the dot product increases with embedding dimension $d$, it helps to scale the dot product back a little to prevent the inputs to the softmax function from growing too large.

$$\begin{aligned} a'_{i,j} &= \frac{\boldsymbol{q}_i^\top \boldsymbol{k}_j}{\sqrt{d}} \\ a_{i,j} &= \operatorname{softmax}\left(a'_{i,j}\right) \end{aligned} \tag{6.10}$$

**Why $\sqrt{d}$?**

Consider a vector in $\mathbb{R}^d$ with all values $c$. Its Euclidean length is $\sqrt{d}c$. Therefore, we are dividing out the amount by which the increase in dimension increases the length of the average vectors.

We then use the attention weights calculated in Equation (6.10), to compute a weight sum to generate the output vector. Note that the output vector $\boldsymbol{z}_i$ depends on $\boldsymbol{v}_j, \forall j$ however not every $\boldsymbol{v}_j$ contributes equally to output $\boldsymbol{z}_i$. The contribution to the output is controlled by the attention weight $a_{i,j}$:

$$\boldsymbol{z}_i = \sum_j a_{i,j} \boldsymbol{v}_j. \tag{6.11}$$

As stated in Sections 6.2 and 6.4, self-attention is a special case of graph neural networks and was first proposed by Vaswani et al. (2017). We can see this from Equations (6.8) and (6.11). We can also view this from the message-passing perspective presented in Section 6.4 to realise that every output is determined by all its neighbours; it is this structure that `SAMPTransfer` successfully makes use of in order to "look" beyond single instances and refine features.

## 6.4. Graph Neural Networks

Now that we have described graphs and permutation invariant and equivariant functions, we can begin to talk about Graph Neural Networks (GNNs). GNNs are perhaps among the most **general** class of deep learning architectures in existence at this moment. Some deep learning architectures can be understood as special cases of the GNN with additional geometric structure. As it happens, we have already covered one such specific case, self-attention in Section 6.3.

Notice that in Section 6.3, the output of $i^{\text{th}}$ depends on all pairs of $(i, j)$. However, we can also constrain this so that the $i^{\text{th}}$ output is more dependent on **local** nodes. This means that instead of considering all pairs, the output of node $i$ instead depends on a smaller set of neighbours. We can formalise this to define what we mean when a node is a "neighbour" of another node.

The **neighbourhood** of $i$ is defined as:

$$\mathcal{N}_i = \{j : (i, j) \in \mathcal{E} \text{ or } (j, i) \in \mathcal{E}\} \tag{6.12}$$

and the neighbourhood features as the multiset:

$$\mathbf{X}_{\mathcal{N}_i} = \{\{\boldsymbol{x}_j : j \in \mathcal{N}_i\}\}. \tag{6.13}$$

Further generalising on our perspective developed in Section 6.2 and Section 6.3, we can specify a function $\varphi(\boldsymbol{x}_i, \mathbf{X}_{\mathcal{N}_i})$ that is locality aware and can operate over a node and its neighbourhood. Then a permutation equivariant function $h$ can be constructed by applying $\varphi$ to every node's neighbourhood separately:

$$h(\mathbf{X}, \boldsymbol{A}) = \begin{bmatrix} \varphi(\boldsymbol{x}_1, \mathbf{X}_{\mathcal{N}_1}) \\ \varphi(\boldsymbol{x}_2, \mathbf{X}_{\mathcal{N}_2}) \\ \vdots \\ \varphi(\boldsymbol{x}_n, \mathbf{X}_{\mathcal{N}_n}) \end{bmatrix}. \tag{6.14}$$

As $h$ is constructed by applying a shared function $\varphi$ to each node locally, its permutation equivariance rests on the output of $\varphi$ being independent of the ordering of the nodes in $\mathcal{N}_i$. Therefore, if $\varphi$ is built to be permutation invariant, then this property will be satisfied. Observe that the role $\varphi$ plays here is that of $\hat{f}$ in Equation (6.1) and role played by $h$ is similar to that of $f$ in Equation (6.8).

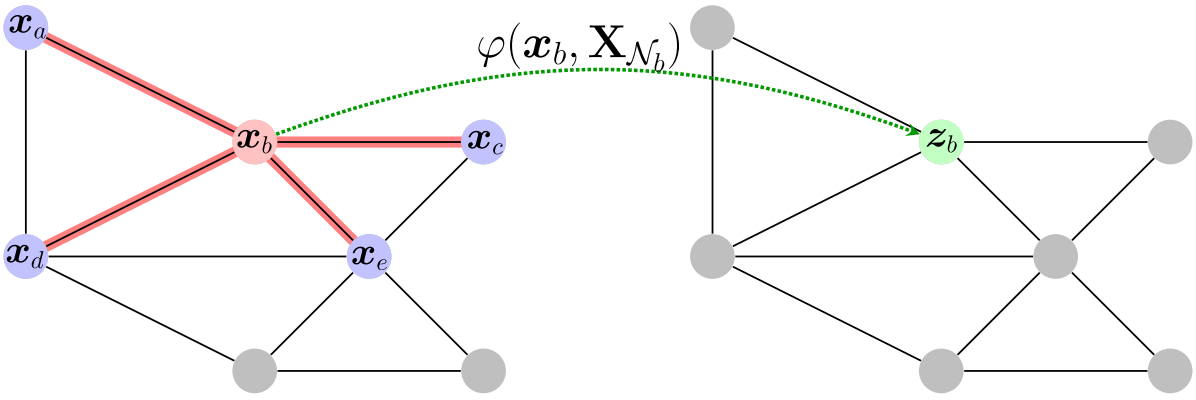



**Figure 6.4:** By applying a permutation-invariant function $\varphi$ to every neighbourhood. In this case, $\varphi$ is applied to the features $\boldsymbol{x}_b$ of node $b$ as well as the multiset of its neighbourhood features, $\mathbf{X}_{\mathcal{N}_b} = \{\{\boldsymbol{x}_a, \boldsymbol{x}_b, \boldsymbol{x}_c, \boldsymbol{x}_d, \boldsymbol{x}_e\}\}$. Applying $\varphi$ in this manner to every node's neighbourhood recovers the rows of the resulting matrix of latent features $\mathbf{Z} = h(\mathbf{X}, \boldsymbol{A})$. Image modified from (Bronstein et al. 2021).

A GNN is a optimisable transformation on the attributes of a graph (nodes, edges, global context) that preserves permutation invariance. Based on our discussion in Section 6.2.2, we consider a graph to be specified with an adjacency matrix $\boldsymbol{A}$ and node features $\mathbf{X}$. We will discuss GNN architectures that are **permutation equivariant** functions $h(\mathbf{X}, \boldsymbol{A})$ constructed by applying shared permutation invariant

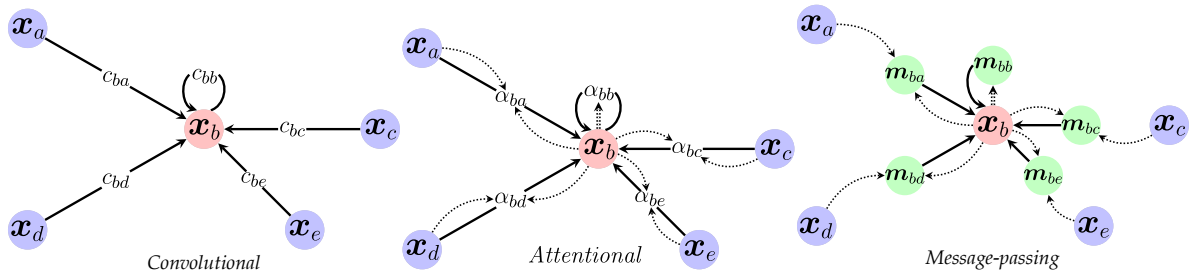



**Figure 6.5:** A visualisation of the dataflow for the three flavours of GNN layers, $g$. We use the neighbourhood of node $b$ from Figure 6.4 to illustrate this. Left-to-right: **convolutional**, where sender node features are multiplied with a constant, $c_{uv}$; **attentional**, where this multiplier is *implicitly* computed via an attention mechanism of the receiver over the sender: $\alpha_{ij} = a(\vec{x}_i, \vec{x}_j)$; and **message-passing**, where vector-based messages are computed based on both the sender and receiver: $\vec{m}_{ij} = \psi(\vec{x}_i, \vec{x}_j)$. Image modified from (Bronstein et al. 2021).

functions $\varphi(\boldsymbol{x}_i, \mathbf{X}_{\mathcal{N}_i})$ where $\mathbf{X}_{\mathcal{N}_i}$ is the neighbourhood of $\boldsymbol{x}_i$. This local function $\varphi$ can be referred to as "diffusion", "propagation", or "message-passing", and the computation $h$ is referred to as a GNN layer.

There are three types of GNN layers that are most common, in all three of them permutation invariance is satisfied by **aggregating** features from $\mathbf{X}_{\mathcal{N}_i}$ (potentially transformed by another function $\psi$) with some permutation-invariant function $\bigoplus$, and then **updating** the features of node $i$, by means of some function $\varphi$. Usually, $\varphi$ and $\psi$ are learnable affine transformations with activation functions (see Chapter 4) and $\bigoplus$ is an operation such as $\mathrm{sum}$, $\mathrm{mean}$, or $\max$. These operations are analogous to pooling functions discussed in Section 5.2.

In the **convolutional** kind of GNNs (Kipf and Welling 2016), the features of the neighbourhood nodes are directly aggregated with fixed weights,

$$z_u = \varphi\left(\boldsymbol{x}_i, \bigoplus_{j \in \mathcal{N}_i} c_{ij} \psi\left(\boldsymbol{x}_j\right)\right). \tag{6.15}$$

Here, $c_{ij}$ specifies the importance of node $i$ to node $j$'s representation.

In the **attentional** scheme (Veličković et al. 2018; Brody, Alon, and Yahav 2021), the interactions are implicit.

$$z_i = \varphi\left(\boldsymbol{x}_u, \bigoplus_{j \in \mathcal{N}_i} a\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) \psi\left(\boldsymbol{x}_j\right)\right) \tag{6.16}$$

Here, $a$ is a learnable **self-attention** mechanism that computes the importance coefficients $\alpha_{ij} = a(\boldsymbol{x}_i, \boldsymbol{x}_j)$ implicitly. They are often softmax-normalised across all neighbours. If the aggregation operation $\bigoplus$ is summation, then the aggregation is a linear combination of the neighbourhood node features, but the weights are feature-dependent. The reader is urged to recognise the similarities between Equation (6.16) and Equation (6.8).

Finally, the **message-passing** type amounts to computing arbitrary vectors ("messages") across edges,

$$z_i = \varphi\left(\boldsymbol{x}_i, \bigoplus_{j \in \mathcal{N}_i} \psi\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right)\right) \tag{6.17}$$

Here, $\psi$ is a learnable message function, computing $j$'s vector sent to $i$, and the aggregation can be considered as a form of message passing on the graph.

It should be noted that the relationship between the three approaches is as follows *convolution* $\subseteq$ *attention* $\subseteq$ *message-passing*. The attentional mechanism can represent convolutional GNNs by an attention-mechanism implemented as a lookup table $a(\boldsymbol{x}_i, \boldsymbol{x}_j) = c_{ij}$. Message-passing can represent both convolutional and attentional GNNs where the messages are only the sender node's features: $\psi(\boldsymbol{x}_i, \boldsymbol{x}_j) = c_{ij}\psi(\boldsymbol{x}_j)$ for convolutional GNNs and $\psi(\boldsymbol{x}_i, \boldsymbol{x}_j) = a\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right)\psi(\boldsymbol{x}_j)$ for attentional GNNs.

`SAMPTransfer` makes use of the attentional semantics in the message-passing context to implement its GNN, however, the implementation of the attention mechanism follows (Vaswani et al. 2017) instead of the mechanism suggested in (Veličković et al. 2018).

# 7

# Self Supervised Learning

In recent years, the field of artificial intelligence has progressed leaps and bounds at an astronomical rate. However, the vast majority of progress has been made in the **supervised** domain, where we have massive amounts of carefully labelled data available. Such models perform extremely well in the singular task for which they were trained.

However, the reality is such that we cannot always possess massive amounts of labelled data for a given task at hand. There are some tasks for which there are simply not enough high-quality labelled data. Examples of such tasks include modelling esoteric languages for which we do not have enough resources or several image classification tasks in the medical domain. Furthermore, supervised learning has shown us that it is constrained when it comes to building easily generalisable models. The idea is that if AI systems can gain a more nuanced understanding of the reality beyond what is specified and constrained by training labels, they will ultimately be closer to human-level understanding.

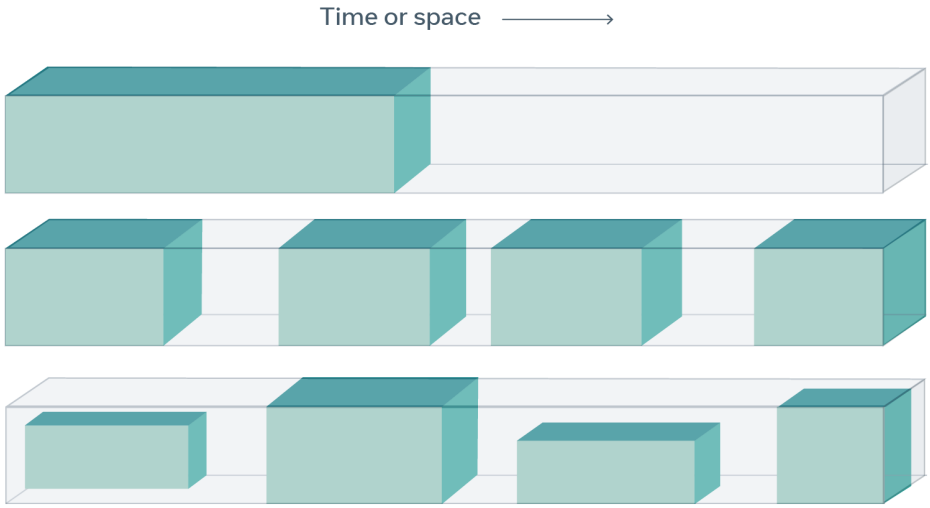



**Figure 7.1:** In self-supervised learning, the system is trained to predict hidden parts of the input (in gray) from visible parts of the input (in green).

Self-supervised learning (SSL) is currently the most promising form of **unsupervised learning** alternative to supervised learning, which is perhaps capable of guiding AI systems to learn general knowledge and an approximate form of common sense. Self-supervision typically involves formulating a specialised supervised task to predict only a subset of information from the rest (see Figure 7.1). This idea has been used extensively in language modelling. The default task of a language model is to predict the next word given a sequence of words (or a partial sentence). We then use the trained model to help us solve another related task. This concept is called **transfer learning**, where we the **pre-train** the model on a large dataset, then save the weights and apply it to another possibly related problem. Often we may also want to **fine-tune** the saved model weights by training it for a few epochs at a slow learning rate on the dataset pertinent to the new problem. The pre-training can be either supervised or unsupervised; however, we shall only be focusing on the unsupervised (self-supervised) aspect. It is also worth noting that most self-supervised training schemes focus on **representation learning**, and we shall cover this in a bit more detail in Section 7.1.

SEER (Goyal et al. 2021), a research project from Meta AI, made use of SWaV (Caron, Misra, et al. 2020) to train a computer vision model that has outperformed state-of-the-art supervised models on tasks such as image classification, object detection and image segmentation. SEER's performance demonstrates quite conclusively that SSL methods can excel at computer vision tasks while learning generalisable features. The work done on SEER parallels the work being done in the NLP field for a while now, where state-of-the-art models frequently use billions of parameters and train using SSL methods on huge datasets.

## 7.1. Representation Learning

The performance of machine learning methods is highly dependent on the choice of data representation (or features) on which they are applied. For that reason, much of the actual effort in deploying machine learning algorithms goes into the design of preprocessing pipelines and data transformations that result in a representation of the data that can support effective machine learning. Our goal with representation learning is to allow the model to learn robust representations with minimal human interference.

Representation learning aims to train machine learning algorithms to learn useful representations, e.g. interpretable representations, useful latent features, or allow transfer learning to be used. Deep neural networks can be considered representation learning models that typically encode information that is projected into a different subspace. These representations are then usually passed on to a linear classifier to, for instance, train a classifier.

At a high level, a network has two components:

1. an encoder (also called a feature extractor in Figure 5.1)
2. and a linear classifier.

The encoder transforms the input data and projects them into a subspace that we refer to as **latent space**. Then the representation (output of the encoder) is passed to a linear classifier. Typically, in a supervised setting, we would like to map representations to labels, which a classical classifier would do. However, representation learning aims at mapping representations to other representations. These learned representations are often dense and compact; they can also generalise to similar data modalities. In other words, these representations can **transfer** well for other tasks and have been the principal method for solving problems in which data annotations are hard or even impossible to obtain. The learned representations are stored in the weights of the model, Figure 7.2 shows a small example where the hidden layers have learnt to recognise specific components of an image. One can save these learnt weights and reuse them for other tasks.

## 7.2. Self-supervision with Images

As mentioned above in this text, self-supervision usually involves formulating a specialised supervised task; this task is also called a **pretext task**. However, we normally do not care about the performance of the model on this artificial task. Rather, we are more concerned with the intermediate learned repre-

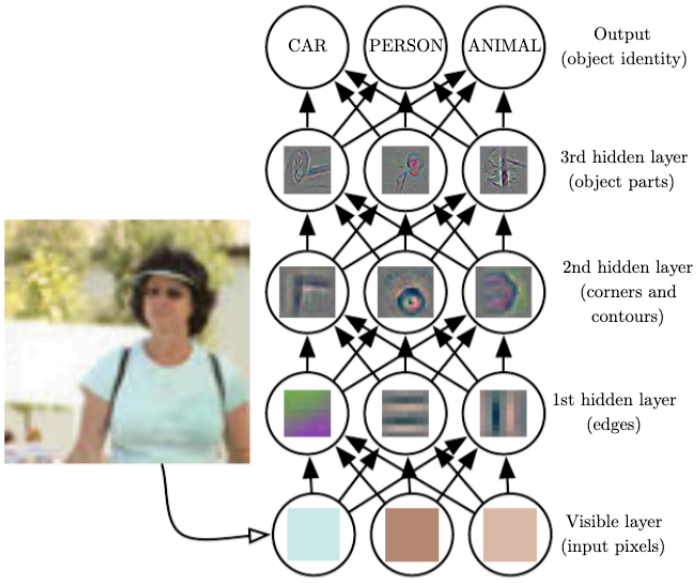



**Figure 7.2:** Deep neural networks combine simple concepts to learn representations and derive complex structures in a hierarchical pipeline. Each layer refines information from the previous layer. Finally, the classifier takes the transformed representation and draws boundaries between classes. See Figure 5.5 for more. Image from (Goodfellow, Bengio, and Courville 2016).

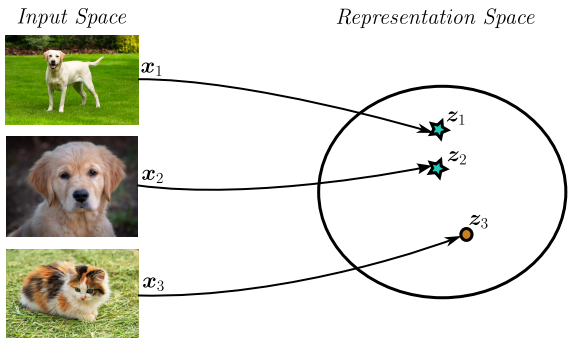



**Figure 7.3:** We have three images $x_1, x_2$ and $x_3$. The first two images depict a dog, and the third image depicts a cat. The network should ideally learn to place the dogs closer together ($z_1$ and $z_2$) in the representation space and the cat ($z_3$) farther away from the dogs.

sentations with the hope that this representation carries with it good semantic or structural meanings that be beneficial to a wide variety of real word tasks.

For example, we might rotate images at four random angles and train a model to predict how each image has been rotated. Here, the task of predicting rotations is the pretext task, so the actual accuracy on this task is irrelevant to us (Gidaris, Singh, and Komodakis 2018). Instead, we expect the model to learn high-quality latent representations that are useful for other real-world tasks. However, this is not the only way, and in the next section we shall discuss one of the most popular techniques in this space.

## 7.3. Contrastive Representation Learning

The goal of contrastive representation learning is to learn a feature subspace in which similar (positive) pairs of data items stay close to each other while dissimilar (negative) pairs are farther apart. Contrastive learning can be applied to both supervised and unsupervised settings. For unsupervised learning, contrastive learning remains one of the most powerful approaches to self-supervised learning.

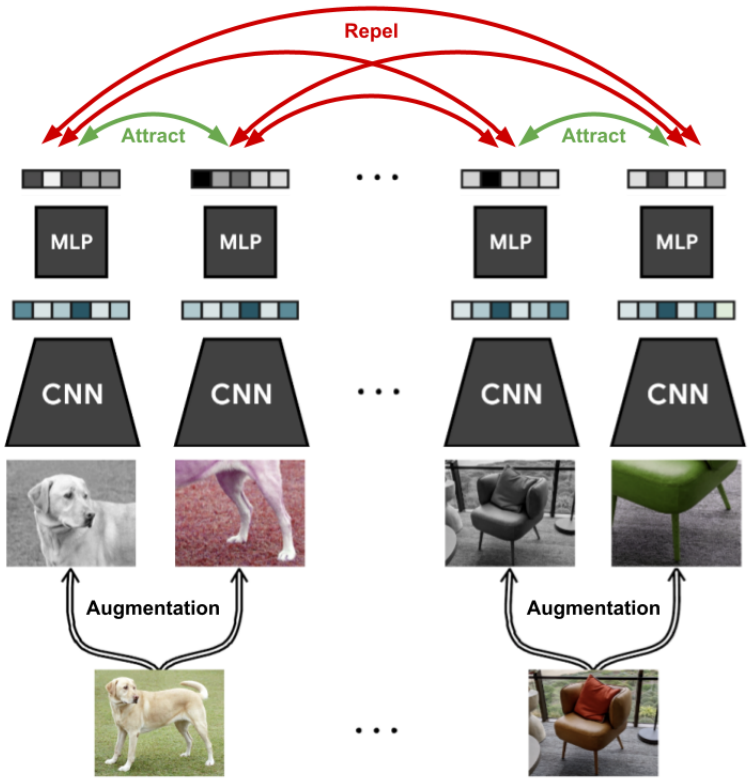



**Figure 7.4:** SimCLR works by augmenting an image twice and ensuring their representations are close to each other while being farther away from representations of other augmented images.

When working in a supervised setting, it is easy to find positive pairs of the data; one only needs to fetch images that are associated with the same true label. However, in an unsupervised setting, we have no such label information; therefore, we must get creative in order to teach the network what constitutes as a positive pair of images. This brings us to one of the most intuitive techniques, SimCLR (T. Chen et al. 2020).

### 7.3.1. SimCLR

The idea behind SimCLR is quite simple and elegant. An image $x$ is randomly sampled from the dataset, and then two random transformations are applied to the image to obtain two augmented images $x_i$ and $x_j$. Each image is passed through the encoder to obtain the representations $z_i$ and $z_j$, respectively. The goal here is to maximise the similarity between $z_i$ and $z_j$, see Figure 7.4 for a small overview of SimCLR. Figure 7.5 shows a few examples of the type of image transformations used by SimCLR.

In order to train the network to bring the said representations closer, we first measure the similarity between them using a **similarity metric** such as cosine similarity or Euclidean distance. The cosine similarity metric between $z_i$ and $z_j$ is given as

$$s_{i,j} = \frac{\boldsymbol{z}_i^\top \boldsymbol{z}_j}{(\|\boldsymbol{z}_i\| \|\boldsymbol{z}_j\|)} \tag{7.1}$$

We calculate the pairwise cosine similarity between each augmented image in a mini-batch using Equation (7.1). Figure 7.6 shows an example of what positive pairs might look like. In the ideal case, the similarity between all *positive* pairs must be high and low between all *negative* pairs.

Next, we use pairwise cosine similarities to calculate the loss. The loss is called **NT-Xent** (nor-

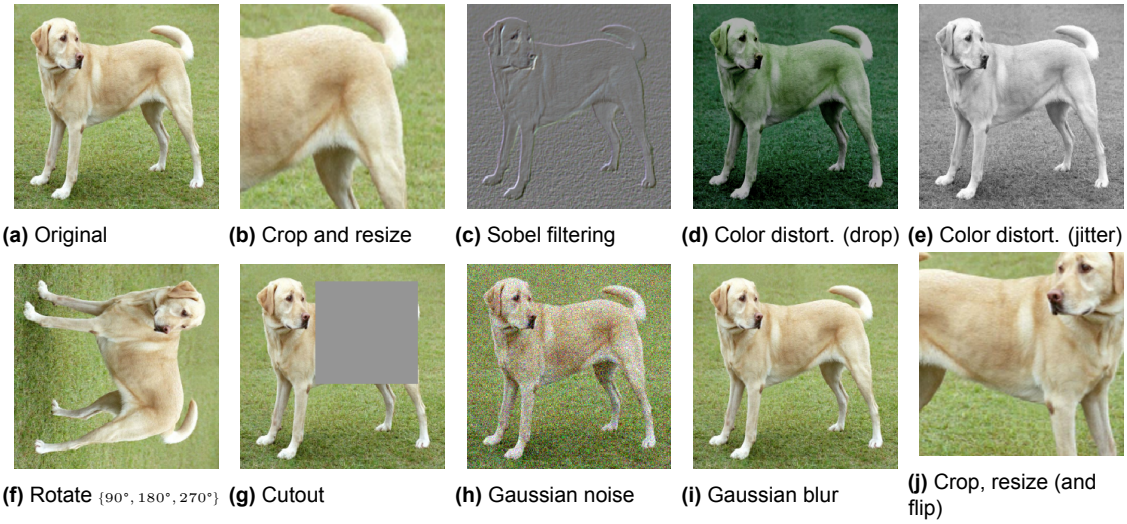



**Figure 7.5:** Illustrations of data augmentation operators. (Original image cc-by: Von.grzanka)

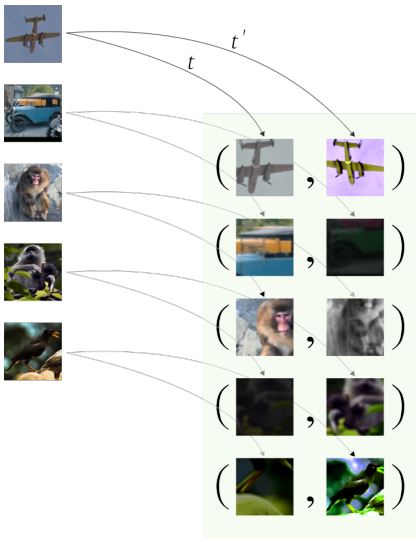



**Figure 7.6:** This figure shows a set of images on the left being transformed using functions $t$ and $t'$ giving us pairs of positve images shown on the right. Image from (Silva 2021).

malised temperature-scaled cross-entropy loss) or NCE (noise contrastive estimator) and is given as

$$\ell(i,j) = -\log \frac{\exp\left(s_{i,j}/\tau\right)}{\sum_{k=1}^{2n} \mathbb{1}_{[k \neq i]} \exp\left(s_{i,k}/\tau\right)} \tag{7.2}$$

where $\tau$ is the temperature scaling factor. The reader is urged to recognise the similarities between Equation (7.2) and Equation (4.8). Equation (7.2) is indeed just the normal $\mathrm{softmax}$ (Equation (4.8)) with an additional temperature scaling factor.

Normally a mini-batch of $n$ samples is randomly sampled, since the contrastive pretext task is defined on pairs of augmented images derived from the mini-batch, it results in a total of $2n$ data points. The negative samples are not sampled explicitly. Instead, given a positive pair, the other $2(n-1)$ augmented samples in a mini-batch are treated as negative samples.

You can choose $\tau$ to be any value; higher values of $\tau$ will lead to a "softer" output distribution of probabilities, for example, $[0.01, 0.01, 0.98]$. As $\tau$ approaches $0$, it will lead to a distribution "sharper", for example, $[0.2, 0.2, 0.6]$. A distribution of "softer" implies that the model is less confident in its predictions,

while “sharper” implies that the model is very confident. A low temperature ($< 1$) discourages predictions from collapsing to a uniform distribution, which is undesirable. When we talk about collapse, we are talking about **representation collapse**. Representation collapse is a phenomenon where in the network outputs the same representation regardless of the input, as one can imagine if every pair of images has the same representation, the loss would be incredibly low. However, if the network returns the same representation for all input, its not very useful for downstream tasks that require the features to be discriminative and have semantic information.

**Intuition for Temperature**

The temperature parameter penalises the larger logits more than the smaller logits. The exponential function is an “increasing function”. So, if a term is already large, penalising it by a small amount would make it much smaller (% wise) than if that term were small.

# 8

# Few-Shot Learning

A good machine learning model often requires training with a large number of samples. Humans, on the contrary, learn new concepts and skills much faster and more efficiently. Children who have only seen cats and birds a few times can quickly tell them apart. People who know how to ride a bike are likely to discover how to ride a motorcycle quickly with little or even no demonstration. Is it possible to design a machine learning model with similar properties - can we have it learn new concepts and skills quickly with a few training examples? That is essentially what few-shot learning methods aim to solve.

Despite notable advances in the field of artificial intelligence, two essential aspects of human conceptual intelligence have consistently eluded machine learning and artificial intelligence (AI) systems. First, for most interesting kinds of natural and man-made categories of entities, humans can learn a new concept from just one or a few handful examples, whereas many AI models would require several thousands of examples to perform satisfactorily. Second, people learn far richer representations than machines do, even for seemingly simple concepts. Humans use these representations for a wide variety of tasks such as creating new entities based on the exemplars, classifying objects into parts, grasping relations between concepts, and creating new abstract categories (concepts) by combining existing ideas and concepts. These rich representations also allow us to differentiate between entities. In stark contrast to this, the best machine learning models and neural networks cannot perform these additional functions using their specialised learnt representations. The challenge arises when we wish for AI models to learn new concepts and representations from few examples and ensure that these representations are abstract and flexible.

In this body of work the focus is on few-shot classification of images. Few-shot classification is cast as a task of predicting class labels for a set of unlabelled data points (*query set*) given only a small set of labelled data points (*support set*). Typically, the query and support data points are drawn from the same distribution.

Few-shot classification methods typically consist of two sequential phases: (i) *pre-training* on a large dataset of “base” classes, regardless of whether the training is supervised or unsupervised. This is followed by (ii) *fine-tuning* on an unseen dataset consisting of “novel” classes. Normally, the classes used in the pre-training and fine-tuning are mutually exclusive. In this paper, our focus is on the self-supervised (also sometimes interchangeably called “unsupervised” in the literature) setting where we have no access to the actual class labels of the “base” dataset.

To this end, various methods have been proposed and broadly categorised under two different approaches:

- **Meta-learning**. Where an algorithm “learns how to learn” and aims to find a quickly generalisable model

- **Transfer Learning**. Where the model is trained to learn optimal representations that are generalisable with minimal updates

The first approach, *meta-learning*, relies on episodic training that involves creating synthetic "tasks" to mimic the downstream episodic fine-tuning phase (Finn, Abbeel, and Levine 2017; Hsu, Levine, and Finn 2018; Antoniou and Storkey 2019; Ji et al. 2019; Khodadadeh, Boloni, and Shah 2019; D. B. Lee et al. 2021; Ye, Han, and Zhan 2022). The second method follows a *transfer learning* approach, where the network trained non-episodically to learn optimal representations in the pre-training phase, which is then followed by an episodic fine-tuning phase (Dhillon et al. 2019; Medina, Devos, and Grossglauser 2020; Tian et al. 2020). In this method, a feature extractor (encoder) is trained using a form of metric learning to capture the structure of the unlabelled data. Next, a simple predictor (conventionally a linear layer) is utilised in conjunction with the pre-trained feature extractor for quick adaptation to the novel classes in the fine-tuning phase. The better the feature extractor captures the global structure of the unlabelled data, the less the predictor requires training samples and the faster it adapts itself to the unseen classes in the fine-tuning phase (also the testing phase).

We also see a set of supervised approaches that do not rely purely on a convolutional feature extractor. Instead, these approaches also make use of graphs and graph neural networks (Satorras and Estrach 2018; Kim et al. 2019; Yang et al. 2020; Yu et al. 2022). Using a graph neural network (GNN) can help in modelling instance-level and class-level relationships. Graphs can also help propagate labels between labelled and unlabelled samples, this was proposed by Xiaojin and Zoubin (2002) and has been applied by Liu et al. (2018) in their work. However, it should be noted that graph-based methods have eluded the unsupervised setting.

Several recent studies have questioned the necessity of meta-learning for few-shot classification (Dhillon et al. 2019; Boudiaf et al. 2020; Medina, Devos, and Grossglauser 2020; Tian et al. 2020; Ziko et al. 2020; D. Chen et al. 2021). They report competitive performance on few-shot benchmarks without episodic training or few-shot task-based experiences during training. These methods follow the second approach and aim to solve the few-shot learning problem by fine-tuning a pre-trained feature extractor with a standard cross-entropy loss. Some of these methods (Medina, Devos, and Grossglauser 2020; Tian et al. 2020; Das, Yun, and Porikli 2022) in the space demonstrate that the transfer learning approach outperforms meta-learning based methods in standard in-domain and cross-domain settings - where the training and novel classes come from totally different distributions.

## 8.1. Formalising the Few-Shot Learning Problem

In a few-shot setting, the model aims to generalise well and quickly enough on a variety of new and potentially unseen tasks after being trained for optimal performance on various learning tasks. Each task is associated with a dataset $\mathcal{D}$, containing both inputs and true labels. The model parameters of an optimal generalisable model are defined as follows:

$$\boldsymbol{\theta}^{\star} = \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{\mathcal{D}\sim p_{\text{data}}}[\mathcal{L}_{\boldsymbol{\theta}}(\mathcal{D})] \tag{8.1}$$

First, we look at how $\mathcal{D}$ is structured in a standard supervised few-shot setting. Consider a labelled dataset of size $M$, $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{M}$ of images $\boldsymbol{x}_i$ and class labels $y_i$. This dataset $\mathcal{D}$ is divided into three disjoint subsets: $\left\{\mathcal{D}^{\text{tr}} \cup \mathcal{D}^{\text{val}} \cup \mathcal{D}^{\text{tst}}\right\} \in \mathcal{D}$, referring to the training, validation, and test subsets, respectively. Next, we define a set of randomly sampled tasks $\mathcal{T}_i$ drawn from the training dataset $\mathcal{D}^{\text{tr}} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{M^{\text{tr}}}$ of size $M^{\text{tr}}$. A task, $\mathcal{T}_i$, is comprised of two parts: (i) the support set $\mathcal{S}$ from which the model learns, (ii) the query set $\mathcal{Q}$ on which the model is evaluated. The support set $\mathcal{S} = \{\boldsymbol{x}_i^s, y_i^s\}_{i=1}^{NK}$ is constructed by drawing $K$ labelled random samples from $N$ different classes, resulting in the so-called ($N$-way, $K$-shot) settings. The query set $\mathcal{Q} = \{\boldsymbol{x}_j^q\}_{j=1}^{NQ}$ then contains $NQ$ unlabeled samples. By convention, we denote the set $\mathcal{T}_i = \mathcal{S} \cup \mathcal{Q}$ with $(N, K)$. The model is expected to quickly learn the optimal parameters from the $NK$ support data points and apply the learnt weights to classify the $NQ$ unlabelled query data points, on which performance is evaluated. Figure 8.1 shows

[1] https://www.borealisai.com/research-blogs/tutorial-2-few-shot-learning-and-meta-learning-i/

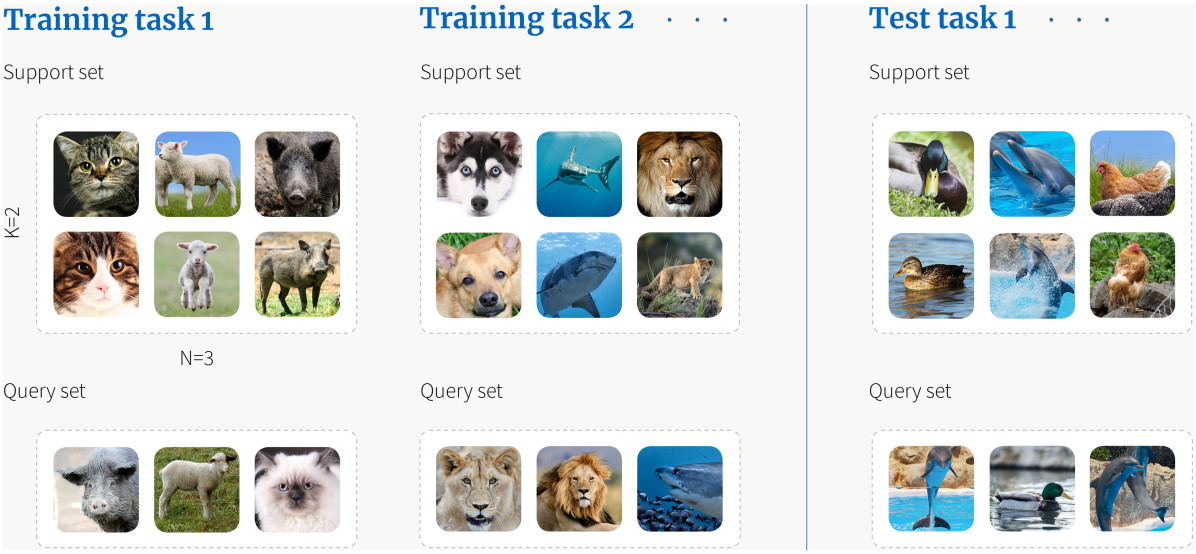



**Figure 8.1:** An example of (3-way, 2-shot) image classification. Image from Borealis AI[1]

(3-way, 2-shot) training tasks, it should be noted that even during testing (downstream classification task) the model is presented with tasks that are in the same (3-way, 2-shot) form as the training tasks.

## 8.2. Model Agnostic Meta Learning (MAML)

**Meta-learning** is most commonly understood as "learning to learn", which refers to the process of improving a learning algorithm over multiple learning episodes and was first proposed by Schmidhuber (1987). On the contrary, conventional ML improves model predictions on multiple data instances. During base learning, an *inner* learning algorithm solves a task such as image classification, defined by a dataset and an objective. During meta-learning, an *outer* algorithm updates the inner learning algorithm so that the model it learns improves an outer objective. The learning episodes of the base task, namely tuples of the image $\boldsymbol{x}_i$ and associated label $y_i$, can be seen to provide the instances that the outer algorithm needs to learn the base learning algorithm

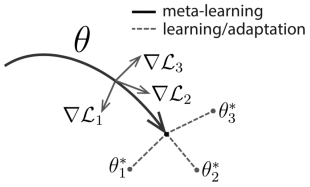



**Figure 8.2:** MAML tries to find optimal generalised weights, by first finding optimal weights in the inner learning phase. Image from (Finn, Abbeel, and Levine 2017).

MAML is a gradient-based **meta learning** method. It is a technique that works with any model that learns through gradient descent. It trains a model to find an optimal set of weights that can be quickly adapted to a new task, through a few gradient descent steps. The meta-learner tries to find an initial set of weights that are not only useful for adapting to various problems, but also can be adapted quickly (in a small number of steps) and efficiently (using only a few examples). Figure 8.2 illustrates how MAML should work at meta-test time. We are looking for a pre-trained parameter that can reach near-optimal parameters for every task in one or a few gradient steps.

This approach is fairly simple and has several advantages. It does not make any assumptions about the form of the model. It is quite efficient - no additional parameters are introduced for meta-learning, and the learner strategy uses a known optimisation process (gradient descent), rather than having to come up with one from scratch. It can also be easily applied to a number of domains, including classification, regression, and reinforcement learning. However, it can be computationally expensive due to the use of higher-order gradients, as we will see.

### 8.2.1. Meta-training

More formally, we define a **base model** to be a neural network $f_{\boldsymbol{\theta}}$ with meta-parameters $\boldsymbol{\theta}$. We want to learn an initial $\boldsymbol{\theta}^{\star} = \boldsymbol{\theta}_0$ that, after a small number $n$ of gradient update steps on data from a support set $\mathcal{S}_b$ to obtain $\boldsymbol{\theta}_n$, the network performs well on the query set of that task $\mathcal{Q}_b$. Here, $b$ is the index of a particular task $\mathcal{T}_b$. The $n$ gradient update steps form what is known as the **inner-loop update process**. The updated base-network parameters after $i$ steps on data from the support task $\mathcal{S}_b$ can be expressed as

$$\boldsymbol{\theta}_i^b = \boldsymbol{\theta}_{i-1}^b - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}_{S_b} \left( f_{\boldsymbol{\theta}_{i-1}^b} \right), \tag{8.2}$$

where $\eta$ is the learning rate, $\boldsymbol{\theta}_i^b$ are the weights of the base network after $i$ steps with task $\mathcal{T}_b$, $\mathcal{L}_{S_b} \left( f_{\boldsymbol{\theta}_{i-1}^b} \right)$ is the loss on the support set of task $b$ after $(i-1)$ (the previous step) update steps. Assuming that our *task batch* (a batch of tasks instead of images) size is $B$ we can define a meta-objective, which can be expressed as:

$$\mathcal{L}_{meta} \left( \boldsymbol{\theta}_0 \right) = \sum_{b=1}^{B} \mathcal{L}_{\mathcal{Q}_b} \left( f_{\boldsymbol{\theta}_n^b(\boldsymbol{\theta}_0)} \right), \tag{8.3}$$

where we explicitly show the dependence of $\boldsymbol{\theta}_n^b$ on $\boldsymbol{\theta}_0$, given by unrolling Equation (8.2). The meta-objective in Equation (8.3) measures the quality of an initialisation $\boldsymbol{\theta}_0$ in terms of the total loss of using that initialisation over a collection of tasks (i.e. a batch of tasks). This meta objective is now minimised to optimise the initial parameter value $\boldsymbol{\theta}_0$, this $\boldsymbol{\theta}_0$ contains within it the across-task experiences and knowledge. This process of optimising for the meta-objective is called the **outer-loop** update process. The resulting update process is given by:

$$\boldsymbol{\theta}_0 = \boldsymbol{\theta}_0 - \beta \nabla_{\boldsymbol{\theta}} \sum_{b=1}^{B} \mathcal{L}_{\mathcal{Q}_b} \left( f_{\boldsymbol{\theta}_N^b(\boldsymbol{\theta}_0)} \right), \tag{8.4}$$

where $\beta$ is a learning rate and $\mathcal{L}_{\mathcal{Q}_b}$ denotes the loss on the query set for task $\mathcal{T}_b$. For more details, please refer to (Finn, Abbeel, and Levine 2017; Antoniou, Edwards, and Storkey 2018).

## 8.3. Prototypical Networks

Prototypical networks are a type of **metric learning** models. The core idea in metric learning is similar to nearest neighbours algorithms ($k$-NN classifier and $k$-means clustering) and kernel density estimation. The goal of Metric Learning is to learn a representation function that maps objects into an embedded space. The distance in the embedded space should preserve the objects' similarity; similar objects get closer, and dissimilar objects get far away. The similarity metric is learnt by a kernel function $f_{\boldsymbol{\theta}}$, parameterised by $\boldsymbol{\theta}$. The distances measured by the kernel function are converted into probabilities on a set of labels $y_i \forall i$ by means of a softmax or sigmoid function. Both `C3LR` and `SAMPTransfer` make use of prototypical networks for classification because this method is simple, yet robust, as evidenced by the performance of `C3LR` and `SAMPTransfer`.

Prototypical networks learn a non-linear function, $f_{\boldsymbol{\theta}}$, that maps the input to a feature vector in an embedding space (or representation space) using a parameterised neural network in a supervised manner. The embeddings of the $K$-shots for each of the $N$-classes are averaged to compute the **class prototypes**. Traditionally, a *prototype* is an entity that is the archetypal form of "something", for example, Boeing 777[2] jets are prototypes of modern large passenger jets. Here, a class prototype $\boldsymbol{c}_k$ plays a similar role; it is a feature vector that is defined for every class $k \in \mathcal{C}$, as the mean vector of the embedded support set samples in class $k$, and is calculated as:

$$\boldsymbol{c}_k = \frac{1}{|\mathcal{S}_k|} \sum_{(\boldsymbol{x}_i, y_i) \in \mathcal{S}_k} f_{\boldsymbol{\theta}}(\boldsymbol{x}_i). \tag{8.5}$$

The distribution over classes for a given query input $\boldsymbol{x}$ is a softmax over the inverse of the distances between the embedding of the query data and the prototype vectors:

$$P(y = k | \boldsymbol{x}) = \text{softmax} \left( -d(f_\theta(\boldsymbol{x}), \boldsymbol{c}_k) \right) = \frac{\exp(-d(f_\theta(\boldsymbol{x}), \boldsymbol{c}_k))}{\sum_{k' \in \mathcal{C}} \exp(-d(f_\theta(\boldsymbol{x}), \boldsymbol{c}_{k'}))} \tag{8.6}$$

[2] https://www.boeing.com/commercial/777/

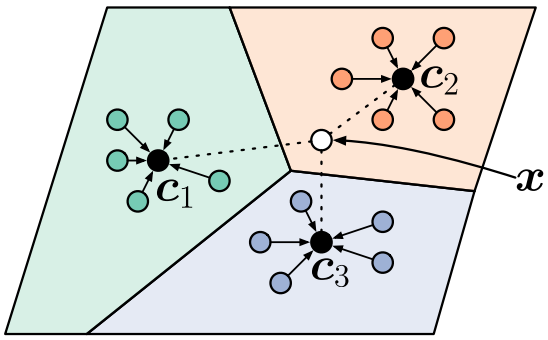



**Figure 8.3:** Few-shot prototypes $c_k$ are computed as the mean of embedded support examples for each class. Image sourced from (Snell, Swersky, and Zemel 2017).

where $d$ can be any distance function, as long as it is a differentiable operation to allow gradient-based learning. In the original work, Snell, Swersky, and Zemel (2017) use the Euclidean distance metric.

Learning is carried out by minimising the negative log likelihood $J(\boldsymbol{\theta})$ and performing stochastic gradient descent over the same to update the kernel's parameters $\boldsymbol{\theta}$:

$$J(\boldsymbol{\theta}) = -\log P_{\boldsymbol{\theta}}(y = k|\boldsymbol{x}). \tag{8.7}$$

### 8.3.1. Re-interpretation of the Prototypical Classifier as a Linear Model

As explained by Snell, Swersky, and Zemel (2017), Prototypical Networks can be re-interpreted as a linear classifier applied to a learned representation $f_{\boldsymbol{\theta}}(\boldsymbol{x})$. Using the Euclidean distance $d(\boldsymbol{z}_i, \boldsymbol{z}_j) = \|\boldsymbol{z}_i - \boldsymbol{z}_j\|^2$ in Equation (8.6) makes the model equivalent to a linear model and the output logits can be expressed as:

$$\begin{aligned} -\|f_{\boldsymbol{\theta}}(\boldsymbol{x}) - \boldsymbol{c}_k\|^2 &= -f_{\boldsymbol{\theta}}(\boldsymbol{x})^\top f_{\boldsymbol{\theta}}(\boldsymbol{x}) + 2\boldsymbol{c}_k^\top f_{\boldsymbol{\theta}}(\boldsymbol{x}) - \boldsymbol{c}_k^\top \boldsymbol{c}_k \\ 2\boldsymbol{c}_k^\top f_{\boldsymbol{\theta}}(\boldsymbol{x}) - \boldsymbol{c}_k^\top \boldsymbol{c}_k &= \boldsymbol{W}_k^\top f_{\boldsymbol{\theta}}(\boldsymbol{x}) + b_k, \text{ where } \boldsymbol{W}_k = 2\boldsymbol{c}_k \text{ and } b_k = -\boldsymbol{c}_k^\top \boldsymbol{c}_k \end{aligned} \tag{8.8}$$

Based on Equation (8.8), we can see that the $k^{\text{th}}$ unit of an equivalent linear layer therefore has weights $\boldsymbol{W}_k = 2\boldsymbol{c}_k$ and biases $b_k = -\boldsymbol{c}_k^\top \boldsymbol{c}_k = -\|\boldsymbol{c}_k\|^2$, which are both differentiable with respect to $\boldsymbol{\theta}$ as they are a function of $f_{\boldsymbol{\theta}}$. Snell, Swersky, and Zemel (2017) hypothesise that this linear classifier is effective because all the required non-linearity can be learnt within the embedding function $f_{\boldsymbol{\theta}}$.

This interpretation also allows us to create a task-specific linear classifier layer for each episode where the layer weights are initialised based on the equivalent weights and biases of the prototypical network as defined above and in Equation (8.8). Subsequently, this layer can then be optimised as usual on the given support set. When computing the update for $\boldsymbol{\theta}$, we allow gradients to flow through the Prototypical Network-equivalent linear layer; we may also **freeze** the embedding network $f_{\boldsymbol{\theta}}$ if we only wish to train the Prototypical Network-equivalent linear layer.

## 8.4. Unsupervised Few-Shot Learning

So far, we have only discussed few-shot learning in a supervised setting. However, a more challenging scenario is the unsupervised setting. Recall that the dataset in the supervised setting is given as $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^M$ which implies that we have the true class labels $y_i$ available during training time. In the unsupervised setting, we no longer have access to the true labels during the training period.

Then a question naturally arises: How do we create valid tasks (see Section 8.1) if we do not have access to the true labels? This is an important question; consider Figure 8.1 where we illustrate ($3$-way, $2$-shot) shots, we see that each of the $3$ classes has exactly $2$ samples. Tasks like this would be difficult to create without labels. To overcome this, we shall discuss some creative methods in the following sections.

### 8.4.1. CACTUs

CACTUs was one of the first methods aimed at solving the unsupervised few-shot learning problem (Hsu, Levine, and Finn 2018). CACTUs uses MAML as part of its training mechanism to obtain a generalisable model. CACTUS also retains the episodic task-based training strategy introduced by MAML (Finn, Abbeel, and Levine 2017). Unlike MAML, however, a major aspect of CACTUs is dedicated to a clever **task creation** technique, as it cannot be fed standard tasks, as discussed above. The idea of

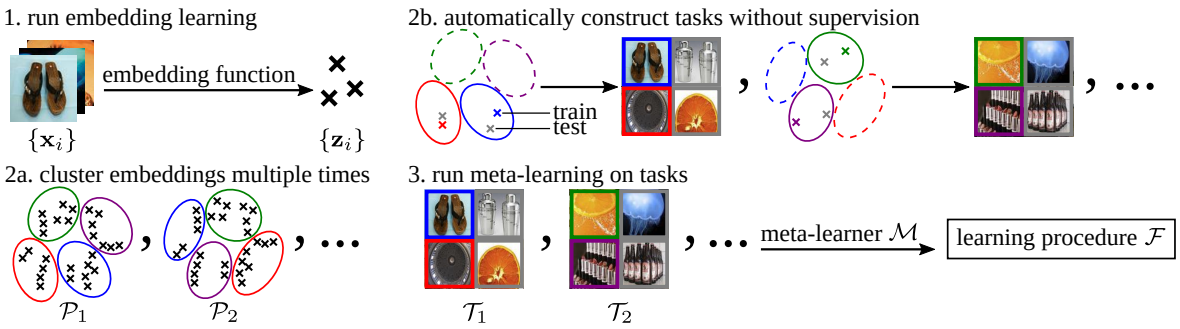



**Figure 8.4:** Illustration of CACTUs, image from (Hsu, Levine, and Finn 2018).

CACTUs is to use a separate embedding learning algorithm $\mathcal{E}$ on $\boldsymbol{x}_i \in \mathcal{D}$ to produce embeddings $\{\boldsymbol{z}_i\}$. It then uses $k$-means clustering on $\{\boldsymbol{z}_i\}$ $P$ times to generate a set of partitions $\mathcal{P}_P = \{\mathcal{C}_P\}$. CACTUs can technically make use of any self-supervised (more generally, unsupervised) representation learning method to train $\mathcal{E}$. Hsu, Levine, and Finn (2018) choose to use DeepCluster (Caron, Bojanowski, et al. 2018), BiGAN (Berthelot et al. 2018), ACAI (Donahue, Krähenbühl, and Darrell 2016), and InfoGAN (Xi Chen et al. 2016).

CACTUs then samples a partition, $\mathcal{P}$, randomly from the set of partitions $\{\mathcal{P}_P\}$, following which a cluster $\mathcal{C}_n$ is randomly sampled from the partition $\mathcal{P}$. The cluster sampling process is carried out $N$ times for each of the classes desired in a ($N$-way, $K$-shot) task. Similarly, the process is repeated for $Q$ query images. Finally, the randomly sampled support and query images are used to create a **synthetic** task that is fed into MAML.

A major concern here is that CACTUs is dependent on bigger models, like AlexNet (Krizhevsky, Sutskever, and Hinton 2012), using powerful self-supervised learning methods on a small dataset like *mini*-ImageNet. The embeddings (representations) generated by this larger model are then used to create synthetic tasks to train a simpler Conv4 architecture. It has been known for quite some time that larger models consistently offer better performance and representations (K. He, X. Zhang, et al. 2015; Dosovitskiy et al. 2020), these better representations indirectly aid a smaller Conv4 to learn better. The entire premise in the few-shot learning space is to learn a generalisable model quickly. Therefore, by using parameter heavy models and computationally intensive algorithms we defeat the purpose of a self-contained end-to-end few-shot learning algorithm.

### 8.4.2. UMTRA

Unsupervised Meta-learning with Tasks constructed by Random sampling and Augmentation (UMTRA), is another method that uses a MAML style training strategy with inner and outer objectives in an unsupervised fashion. Unlike CACTUs (Section 8.4.1), the focus of UMTRA is not on generating ideal tasks. Instead, the idea behind UMTRA can be summarised in one line; it is an unsupervised method that uses augmentations of the images in a given episode to create a valid query set. This allows it to generate a "labelled" query set for the outer objective and can perform well if we span the entire space of the classes.

The functioning of UMTRA is illustrated in Figure 8.5, where $N$ data points are randomly sampled, each of the $N$ random samples is given its own unique class label. Subsequently, each sample $\boldsymbol{x}_i$ is augmented using an augmentation function $\mathcal{A}$ that gives an augmented image $\boldsymbol{x}'_i$. The reason this works is that each augmented image will share its class label with the original image. Therefore, the outer objective will always have samples of the same class available for it to learn from.

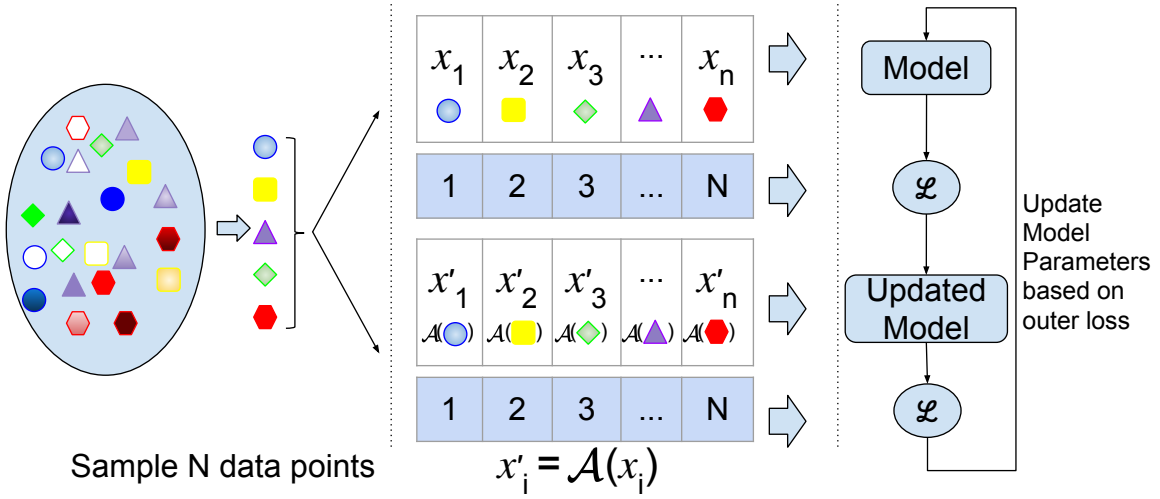



**Figure 8.5:** We start from a dataset of unlabelled data. The training data is created by randomly choosing $N$ samples $\boldsymbol{x}_i$ from the dataset. The query data is created by applying the augmentation function $\mathcal{A}$ to each sample from the training data. Image borrowed from (Khodadadeh, Boloni, and Shah 2019).

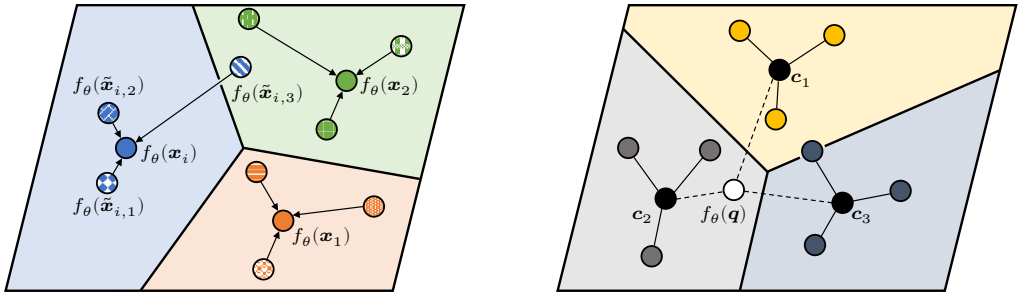
### 8.4.3. ProtoTransfer

Until now, we have discussed CACTUs and UMTRA, both of which use Model Agnostic Meta Learning (MAML) and episodic training to solve the few-shot classification problem. However, several recent studies have questioned the necessity of meta-learning for few-shot classification (Dhillon et al. 2019; Boudiaf et al. 2020; Medina, Devos, and Grossglauser 2020; Tian et al. 2020; Ziko et al. 2020; D. Chen et al. 2021). They report competitive performance on few-shot benchmarks without episodic training or few-shot task-based experiences during training. These methods follow an approach where they aim to solve the few-shot learning problem using transfer-learning. First, they pre-train a feature-extractor and then fine-tune a simple classification layer (usually a linear layer) with standard cross-entropy loss. In fact, both our methods (`C`$^3$`LR` and `SAMPTransfer`) are based on this paradigm and comfortably outperform unsupervised/supervised MAML based methods.

Some of these methods (Medina, Devos, and Grossglauser 2020; Tian et al. 2020; Das, Yun, and Porikli 2022) in the space demonstrate that the transfer learning approach outperforms meta-learning based methods in standard in-domain and cross-domain settings, where the training and novel classes come from totally different distributions.

One such method that we shall focus on is ProtoTransfer (Medina, Devos, and Grossglauser 2020). Unlike CACTUs and UMTRA ProtoTransfer does not rely on a task-based episodic training regime and does not use MAML for its training. Instead, ProtoTransfer uses a form of contrastive representation learning that is similar to and inspired by SimCLR. ProtoTransfer consists of two phases: (i) **pre-training** on a large dataset of "base" classes. This is followed by (ii) **fine-tuning** on an unseen dataset consisting of "novel" classes.

Medina, Devos, and Grossglauser (2020) call their pre-training method **ProtoCLR**, and in principle it is extremely similar to SimCLR. ProtoCLR uses the original image, $\boldsymbol{x}_i$, and $Q$ augmentations, $\{\tilde{\boldsymbol{x}}_{i,q}\}_{q=1}^{Q}$. In order to learn the metric embedding function, $f_{\boldsymbol{\theta}}(\cdot)$, a contrastive loss is used here to ensure that all $Q$ augmentations have the same representation as the original image. SimCLR on the other hand, uses two augmentations of the same image and ensures similarity between them by means of a contrastive loss. Although not the same, one can see that ProtoCLR is very inspired by methods such as SimCLR.




(a) Self-Supervised Prototypical Pre-Training (b) Prototypical Fine-Tuning & Inference

**Figure 8.6:** Self-Supervised Prototypical Transfer Learning. (a): In the embedding, original images $\boldsymbol{x}_i$ serve as class prototypes around which their $Q$ augmented views $\tilde{\boldsymbol{x}}_{i,q}$ should cluster. (b): Prototypes $\boldsymbol{c}_k$ are the means of embedded support examples for each class $n$ and initialise a final linear layer for fine-tuning. An embedded query point $\boldsymbol{q}$ is classified via a softmax over the fine-tuned linear layer. Image sourced from (Snell, Swersky, and Zemel 2017).

**Link between self-supervision and an episode**

Consider a mini-batch that contains $n$ random samples $\{\boldsymbol{x}_i\}_{i=1\ldots n}$ from the training set $\mathcal{D}^{\text{tr}}$. As our self-supervised setting does not assume any knowledge about the base class labels $\boldsymbol{y} \in \mathcal{D}^{\text{tr}}$, we treat each sample as its own class. Thus, each sample $\boldsymbol{x}_i$ serves as a $1$-shot support sample and class prototype. For every prototype $\boldsymbol{x}_i$, $Q$ randomly augmented samples $\tilde{\boldsymbol{x}}_{i,q}$ are used as query samples.

Following the ProtoCLR pretraining phase, we have a fine-tuning phase called **ProtoTune** and it is a supervised phase. In the fine-tuning phase, the learnt embedding $f_{\boldsymbol{\theta}}(\cdot)$ is used to address the **target problem** of few-shot image classification, where we are presented with test tasks episodically, each with their own support and query. ProtoTune is based on the linear layer interpretation of the Prototypical classifier (see Section 8.3.1). It uses each task's support set to initialise the linear classifier and fine-tune it on subsets of the support set for some iterations, randomly sampling different subsets of support images in each iteration.

ProtoTransfer was chosen as the base for `C`$^3$`LR` and `SAMPTransfer` due to its robustness and simplicity of self-supervision. Although both `C`$^3$`LR` and `SAMPTransfer` are inspired by ProtoTransfer, the similarities only extend to their shared self-supervised nature and in the case of `SAMPTransfer` there is a reimagined network and fine-tuning phase based on optimal transport that has the potential to be useful even outside of few-shot learning.

# 9

# Optimal Transport

In most decisions pertaining to life and sciences, the "shortest path" approach acts as the guiding strategy. Whenever a commodity, a person, a single bit of information, or any entity that is available at a given point and needs to be sent to a target point, one should ideally favour a way that uses the least effort possible. Typically, one would achieve this by moving an entity along a straight line from point A to point B on a plane. The theory of **optimal transport** (OT) generalises that intuition in the case where, instead of moving only item at a time, we are concerned with the problem of simultaneously moving several items (or a distribution) from configuration onto another.

As vacation planners can attest, planning the transportation for a group of individuals involved in the event, with the added constraint that they reach a given target configuration upon arrival, is considerably more tortuous than carrying it out for a single individual. In fact, thinking in terms of groups or rather distributions requires more advanced mathematical thinking and formalism, which was first explored by Monge (1781). Regardless of how complicated this formalism might be, it has deep rooted connections to our daily life. Transportation, whether it be of people, goods, or information, rarely involves moving items. All major problems in economics, such as supply chain logistics, production planning, or warehousing, involve moving distributions, and that motif consistently appears in all seminal references on optimal transport.

`SAMPTransfer` uses an optimal transport based module in the fine-tuning phase to "transport" the support set embeddings into the domain of the query set embeddings. The transportation step "moves" the support embeddings in such a way that they overlap more areas of the query embedding domain, more specifically, we are trying to align the support embedding distribution with the query embeddings. This transportation step is followed by prototypical classification which makes use of the *transported support* embeddings. Using the transported support embeddings allows to create prototypes that are more representative of the items they are archetypes of.

## 9.1. A brief historical context of Optimal Transport

The inception of optimal transport began in the late $18^{\text{th}}$ century under the rule of Louis XVI, around $1781$, almost a decade before the French revolution of $1789$. The French physicist and mathematician Gaspard Monge[1] was the first to formulate the mathematical problem of "Excavation and embankments" ("Mémoire sur la théorie des déblais et des remblais") - how to transport soil during the construction of forts and roads with minimal transport expenses. But what does shoveling dirt have to do with AI, statistics, or machine learning? Interestingly, the framework devised by Monge provides a formulation for comparing probability distributions. Let us think of probability density functions as piles of dirt, where the "height" of the pile corresponds to the probability density at that point, and the shoveling of dirt between the piles as moving probability from one point to another, at a cost proportional to the distance

[1] https://mathshistory.st-andrews.ac.uk/Biographies/Monge/

between these two points.

The modern form of optimal transport and the one that is in use today in various applications was first formulated by the nobel laureate Leonid Vitaliyevich Kantorovich[2], the father of linear programming. It was his work in 1942 which cast the transportation as a probablistic measure on a metric space $\mathcal{X}$, where in a measure $\alpha$ is the intial one that needs to be transported to the second and final measure $\beta$, that is, the desired distribution after transportation. A transportation plan, as it is called by Leonid Kantorovich, is also some probability measure $\pi$ on the Cartesian product of the space with itself $\mathcal{X} \times \mathcal{X}$. We shall go into a bit more depth in later sections of this chapter.

It would be several years later that this theory was rediscovered because of the work of Brenier (1991) and others that this theory provided a crucial basis for research, with strong links to convexity, partial differential equations, and statistics. Today, researchers in computer science disciplines, imaging, and more generally data scientists recognise that optimal transport theory grants powerful tools to study distributions in an abstract context, such as comparing readily available distributions.

## 9.2. Discrete Optimal Transport

Since our work mostly deals with atomic items, we chose to focus on the discrete formulation instead of the continuous formulation, although it must be noted that the concepts spoken about here do not change drastically in the continuous domain.

### 9.2.1. Asssignment Problem

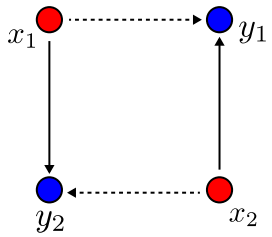



**Figure 9.1:** Non-unique assignments. The other solution is dashed. Image sourced from (Peyré, Cuturi, et al. 2019).

In its most basic form, OT can be viewed as an assignment problem between sets of entities, that is, among all possible configurations, which is the best? This question is quite restrictive in the sense that we can only work with two sets of the same total size, that is, an *initial* set and a *target* set.

Each set can be represented as a histogram (or vector) $\boldsymbol{r}$, that belongs to the probability simplex - the components of the vector sum up to $1$:

$$\boldsymbol{r} \in \left\{ x = (x_1, ..., x_N) \in \mathbb{R}^N : \sum_{i=1}^{N} x_i = 1 \right\} \tag{9.1}$$

If we consider $\mathbf{C}_{i,j}$ as the cost of moving an element from $i$ to $j$, then the quantity we wish to minimise is $\sum_i \mathbf{C}_{i,\sigma(i)}$ , where $\sigma$ is a **permutation** of the set $\{1, \dots, N\}$. This permutation represents an assignment of the bin $i$ of the first histogram to the output positions $j$ in the second histogram.

In this form optimal transport is fundamentally a combinatorial problem, and may be summarised as: How can we assign every element $i \in \{1, \dots, N\}$ to elements $j \in \{1, \dots, N\}$ in order to minimise $\sum_i \mathbf{C}_{i,\sigma(i)}$.

The result of this search is called **optimal assignment**. As you may have already guessed, there can be $N!$ possible solutions to this problem; therefore, as $N$ grows, this problem quickly becomes intractable. An important aspect is that this assignment is not unique. Figure 9.1 is an example where two elements are assigned to two other elements that together form the four corners of a square.

[2] https://www.nobelprize.org/prizes/economic-sciences/1975/kantorovich/biographical/

### 9.2.2. Working with Asymmetric Distributions

Requiring two equally sized histograms is a very strong constraint, hardly any real-world problems present themselves in this manner. By expanding the previous problem definition to a broader class of histograms, we obtain the Monge problem. In this version of the problem, several points $\boldsymbol{x}_i$ can map to the same $y_j$ and their weights will be summed.

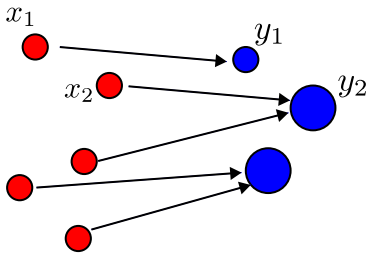



**Figure 9.2:** The Monge problem. Image from (Peyré, Cuturi, et al. 2019).

In this case, the mapping between inputs and outputs is no longer a combinatorial permutation, but a **surjective** map $\pi$. If points $\{\boldsymbol{x}_1, \dots, \boldsymbol{x}_n\}$ have weights $\boldsymbol{r} = (r_1, \dots, r_n)$ and points $\{y_1, \dots, y_m\}$ have weights $\boldsymbol{c} = (c_1, \dots, c_m)$, $\pi$ must verify:

$$\forall j \in \{1, \dots m\}, \quad c_j = \sum_{i:\pi(x_i)=y_j} a_i \tag{9.2}$$

**Surjective functions**

A surjective function is a function $f$ that maps an element $\boldsymbol{x}$ to every element $y$; that is, for every $y$, there is an $\boldsymbol{x}$ such that $f(\boldsymbol{x}) = y$. In other words, every element of the function's codomain is the image of at least one element of its domain. It is not required that $\boldsymbol{x}$ be unique; the function $f$ may map one or more elements of $X$ to the same element of $Y$.

Even with this formulation, it does not make our job easier, the mass conservation constraint stated in Equation (9.2) must be satisfied, while our problem still remains an assignment problem. We are still assigning element $\boldsymbol{x}_i$ to element $\boldsymbol{y}_j$.

### 9.2.3. The Kantorovich relaxation

Even with the above extension, this formulation of the optimal transport problem is still too constrained to be practically useful in many cases. As alluded to earlier in Section 9.1, in 1942 Leonid Kantorovich was instrumental in making OT viable for practical use. Kantorovich introduced a key idea, which is to **relax** the deterministic portion of the transportation. Points in the source histogram (or domains or distributions), $\boldsymbol{x}_i$, no longer have to map to a single target point and can be *fragmented* into *partial* assignments, this is called **mass splitting**.

**Relaxation**

Relaxation refers to the modelling strategy in mathematical optimisation and associated fields. Relaxation stands to be the approximation in relation to the difficult problem with regard to a nearby problem which stands to be easy to compute/solve.

This new relaxed formulation is much more suitable for real-world situations, such as logistical problems. Hitchcock (1941) stated a version of this problem as follows: When several factories supply a product to a number of cities, we seek the least costly means of distribution. Due to freight rates and other expenses, the cost of a ton of product to a particular city will vary according to the factory that supplies it and will also vary from city to city.

To reflect this change, we will slightly modify our previous formulation by replacing the permutation function $\sigma$ by a coupling matrix $\boldsymbol{\pi} = \boldsymbol{\pi}_{ij} \in \mathbb{R}_+^{n \times m}$. In Figure 9.3, each $\boldsymbol{\pi}_{ij}$ would be the weight of

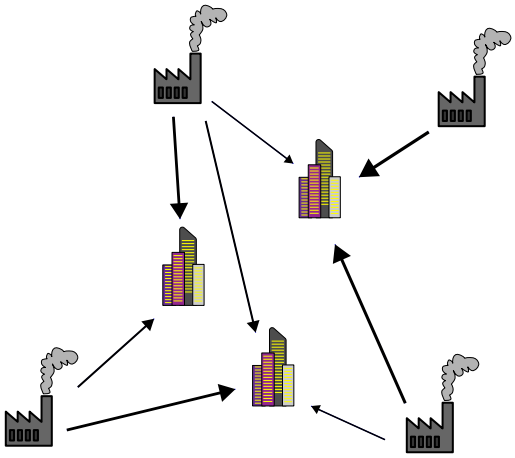


**Figure 9.3:** Factories with different supply capacities have to deliver goods to cities with various demands.

the arrow from factory $i$ to city $j$. As we have mentioned earlier, $\boldsymbol{r}$ is a probability simplex, the same is applicable to $\boldsymbol{c}$. With this established, all possible assignments can be written as:

$$\boldsymbol{\Pi}(\boldsymbol{r}, \boldsymbol{c}) = \left\{ \boldsymbol{\pi} \in \mathbb{R}_{+}^{n \times m} \mid \boldsymbol{\pi} \mathbf{1}_m = \boldsymbol{r}, \boldsymbol{\pi}^\top \mathbf{1}_n = \boldsymbol{c} \right\}. \tag{9.3}$$

$\boldsymbol{\Pi}(\boldsymbol{r}, \boldsymbol{c})$ contains all non-negative $n \times m$ matrices for which all rows sum up to $\boldsymbol{r}$ and all columns sum up to $\boldsymbol{c}$. This makes $\boldsymbol{\Pi}(\boldsymbol{r}, \boldsymbol{c})$ a **polytope** of $\boldsymbol{r}$ and $\boldsymbol{c}$, representing a polyhedral set of $n \times m$ matrices. $\boldsymbol{\Pi}(\boldsymbol{r}, \boldsymbol{c})$ is then essentially a collection of **transport plans** (coupling matrix) of which some are better than others. Keep in mind that both $\boldsymbol{r}$ and $\boldsymbol{c}$ can only be accessed through a finite set of samples each, in Figure 9.3 that means a limited set of factories and cities.

**Polytope**

The word polytope is used to mean a number of related, but slightly different, mathematical objects. A convex polytope may be defined as the convex hull of a finite set of points that are always bounded. “Convex” implying that there is a minimum point.

Cuturi (2013) also gives a probabilistic interpretation for $\boldsymbol{\Pi}(\boldsymbol{r}, \boldsymbol{c})$: for $A$ and $B$ two multinomial random variables taking values in $\{\boldsymbol{x}_1, \dots, \boldsymbol{x}_n\}$ and $\{\boldsymbol{y}_1, \dots, \boldsymbol{y}_m\}$ each with distribution $\boldsymbol{r}$ and $\boldsymbol{c}$, respectively, the set $\boldsymbol{\Pi}(\boldsymbol{r}, \boldsymbol{c})$ contains all possible *joint probabilities* of $(A, B)$. Any matrix $\boldsymbol{\pi} \in \boldsymbol{\Pi}(\boldsymbol{r}, \boldsymbol{c})$ can be identified with a joint probability for $(A, B)$ such that $p(A = i, B = j) = p_{ij}$.

Given a cost matrix $\boldsymbol{C}$, the cost of mapping $\boldsymbol{r}$ to $\boldsymbol{c}$ using a transport plan (or joint probability) $\boldsymbol{\pi}$ can be quantified as $\langle \boldsymbol{\pi}, \boldsymbol{C} \rangle_F$, we can now formulate the problem in a much cleaner fashion:

$$\boldsymbol{\pi}^\star = \min_{\boldsymbol{\pi} \in \boldsymbol{\Pi}(r,c)} \sum_{i,j} \boldsymbol{C}_{i,j} \pi_{i,j} = \min_{\boldsymbol{\pi} \in \boldsymbol{\Pi}(r,c)} \langle \boldsymbol{\pi}, \boldsymbol{C} \rangle_F \tag{9.4}$$

where $\boldsymbol{\pi}^\star$ is the optimal transport plan and $\langle \cdot, \cdot \rangle_F$ is the Frobenius dot product. When the cost matrix is based on a valid distance metric, the optimum $\boldsymbol{\pi}^\star$ is known as **Wasserstein distance**. It is basically a distance between two probability distributions. It is sometimes also called **earth mover distance** as it can be interpreted as how much “dirt” you have to move to change one “landscape” (distribution) in another.

### 9.2.4. Entropic Regularisation

Regularising the optimal transport problem was originally proposed by Hitchcock (1941). It is a method for **approximating** solutions to the optimal transport problem by adding a regularising term to the objective function in Equation (9.4).

We start by defining the entropy of the coupling matrix, $\boldsymbol{\pi}$:

$$\mathbb{H}(\boldsymbol{\pi}) = -\sum_{ij} \boldsymbol{\pi}_{ij} \log \boldsymbol{\pi}_{ij}. \tag{9.5}$$

In information theory, the entropy of a random variable is the average level of "information", "surprise", or "uncertainty" inherent to the variable's possible outcomes. Based on this understanding, a matrix with low entropy will be sparser, with most of its non-zero values concentrated in a few points. Conversely, a matrix with high-entropy will be *smoother*, with the maximum entropy achieved with a uniform distribution of values across its elements. With a regularisation coefficient $\varepsilon$, we can include this in the optimal transport problem to encourage smoother coupling matrices:

$$\begin{aligned} \boldsymbol{\pi}^\star = \min_{\boldsymbol{\pi} \in \boldsymbol{\Pi}(r,c)} & \langle \boldsymbol{\pi}, \boldsymbol{C} \rangle_F - \varepsilon \mathbb{H}(\boldsymbol{\pi}) \\ \text{subject to } & \boldsymbol{\pi}\mathbf{1} = \boldsymbol{r} \\ & \boldsymbol{\pi}^\top \mathbf{1} = \boldsymbol{c}. \end{aligned} \tag{9.6}$$

By increasing the value of $\varepsilon$, the resulting coupling matrix will be smoother, and as $\varepsilon \to 0$ the coupling matrix will be sparser (or sharper) and the solution will be closer to that of the original relaxed OT problem in Equation (9.4). The intuition behind entropy regularisation is similar to the temperature-normalised cross-entropy (NT-Xent) discussed in Section 7.3.1.

The addition of this entropic regularisation makes this optimisation problem convex. Therefore, there is a unique optimal solution $\boldsymbol{\pi}^\star$. It can be shown that the solution to this regularised problem has the following form:

$$\forall (i,j) \in \{1,...,n\} \times \{1,...,m\}, \quad \boldsymbol{\pi}_{i,j} = \mathbf{u}_i \boldsymbol{K}_{i,j} \mathbf{v}_j, \quad \boldsymbol{\pi} = \operatorname{diag}(\mathbf{u}) \boldsymbol{K} \operatorname{diag}(\mathbf{v}) \tag{9.7}$$

where $\boldsymbol{K}$ is a kernel matrix where $\boldsymbol{K}_{i,j} = \exp\left(-\boldsymbol{C}_{i,j}/\varepsilon\right)$ calculated with $\boldsymbol{C}$ and $\mathbf{u}$ and $\mathbf{v}$ are unknown scaling variables. The formulation in Equation (9.7) is important because we now have an explicit formula for an optimal coupling matrix.

### 9.2.5. Sinkhorn-Knopp Algorithm

The problem in Equation (9.7) is known as the matrix scaling problem, we are trying to find two scaling diagonal matrices, $\mathbf{u}$ and $\mathbf{v}$, that when multiplied with $\boldsymbol{K}$ give $\boldsymbol{\pi}$. It must be noted that $\boldsymbol{\pi}$ is what we call a **doubly stochastic matrix**, by extension $\operatorname{diag}(\mathbf{u}) \boldsymbol{K} \operatorname{diag}(\mathbf{v})$ is also doubly stochastic. We can find these diagonal matrices by alternatively updating $\mathbf{u}$ and $\mathbf{v}$ using **Sinkhorn's algorithm**:

$$\mathbf{u}^{(k+1)} = \frac{\boldsymbol{r}}{\boldsymbol{K}\mathbf{v}^{(k)}} \tag{9.8}$$

$$\mathbf{v}^{(k+1)} = \frac{\boldsymbol{c}}{\boldsymbol{K}^\top \mathbf{u}^{(k+1)}} \tag{9.9}$$

This simple iterative method of approaching the double stochastic matrix, alternately rescales all rows and all columns of $\boldsymbol{\pi}$ to sum to $1$. The convergence proof of this algorithm is attributed to Sinkhorn and Knopp (1967). The algorithm not only converges, but does so at a linear rate. Since these iterations are solving a regularised version of the original problem, the corresponding Wasserstein distance that results is sometimes called the Sinkhorn distance. It is easy to see that these iterations form a sequence of linear operations. Therefore, it is straightforward for deep learning models to backpropagate through these iterations; in other words, the Sinkhorn-Knopp algorithm is differentiable.

Following a paper by Cuturi (2013) called "Sinkhorn distances: Lightspeed computation of optimal transport" that showed Sinkhorn updates were efficient and scalable approximations to OT, there has been renewed interest in the community. Several performant ideas in the self-supervised learning space, such as SWaV (Caron, Misra, et al. 2020) and SeLa (Asano, Rupprecht, and Vedaldi 2019) have made use of the Sinkhorn-Knopp algorithm in their methods. Furthermore, the supervised few-shot learning space has also seen the usage of Sinkhorn-Knopp in methods like PT-MAP (Hu, Gripon, and Pateux 2021) which use the algorithm to assign labels to unlabelled query samples.

# 10

# C$^3$LR: Additional Materials

This chapter deals with the justification of certain design choices made for C$^3$LR (Chapter 2), that could not be covered in the original article due to space constraints. The algorithm from the article is made available here for convenience.

**Algorithm 1:** Class-Cognizant Contrastive Learning (C$^3$LR)

**Require:** $L$, $Q$, $f_\phi$, $\mathcal{A}$, $\alpha$, $d[\cdot,\cdot]$

**while** *not done* **do**
  Sample minibatch $\{\boldsymbol{x}_i\}_{i=1}^{L}$
  **forall** $i \in \{1,\dots,L\}$ **do**
    **forall** $q \in \{1,\dots,Q\}$ **do**
      $\tilde{\boldsymbol{x}}_{i,q} = \psi^q(\boldsymbol{x}_i)$; $\psi^q \sim \mathcal{A}$.
    **end**
  **end**
  $\mathbf{R} = \texttt{ReRank}\Big(\Big[f_\phi\left(\{\boldsymbol{x}_i\}_{i=1}^{L}\right), f_\phi\left(\{\tilde{\boldsymbol{x}}_{i,\,q}\}_{i=1,q=1}^{L,Q}\right)\Big]\Big)$
  $\mathcal{C} = \{\boldsymbol{C}_1, \boldsymbol{C}_2, \dots, \boldsymbol{C}_P\} \leftarrow \texttt{HDBSCAN}(\mathbf{R})$
  $\mathcal{M} = \{\mathbf{m}_p\}_{p=1}^{P}; \quad \mathbf{m}_p = \frac{\sum_{x_j \in \boldsymbol{C}_p} x_j}{|\boldsymbol{C}_p|}$
  **let** $r(i,q,p) = -\log \frac{\exp(-d[f_\phi(\tilde{\boldsymbol{x}}_{i,q}), \mathbf{m}_p])}{\sum_{p=1}^{P} \exp(-d[f_\phi(\tilde{\boldsymbol{x}}_{i,q}), \mathbf{m}_p])}$
  **let** $\ell(i,q) = -\log \frac{\exp(-d[f_\phi(\tilde{\boldsymbol{x}}_{i,q}), f_\phi(\boldsymbol{x}_i)])}{\sum_{k=1}^{L} \exp(-d[f_\phi(\tilde{\boldsymbol{x}}_{i,q}), f_\phi(\boldsymbol{x}_k)])}$
  $\mathcal{L}_1 = \frac{1}{LQ} \sum_{p=1}^{P} \sum_{i=1}^{L} \sum_{q=1}^{Q} r(i,q,p)$
  $\mathcal{L}_2 = \frac{1}{LQ} \sum_{i=1}^{L} \sum_{q=1}^{Q} \ell(i,q)$
  $\mathcal{L} = \mathcal{L}_1 + \mathcal{L}_2$
  $\phi \leftarrow \phi - \alpha \nabla_\phi \mathcal{L}$
**end**

## 10.1. Choice of Clustering Algorithm

As we can see from Algorithm 1 and Chapter 2, the re-ranking and clustering steps are crucial to its functioning. The purpose of this section is to show why HDBSCAN (McInnes, Healy, and Astels 2017) was chosen in combination with the $k$-reciprocal Jaccardian distance (Zhong et al. 2017). To motivate our choices we shall refer to Table 10.1.

| **Method** | Clustering Algorithm | UMAP | UMAP $\mathbb{R}^{\cdot}$ | $L$ | $B$ | Avg Test Acc (%) |
|---|---|---|---|---|---|---|
| ProtoCLR | None | ✗ | - | 50 | 200 | 60.66 |
| ProtoCLR | None | ✗ | - | 200 | 800 | 62.32 |
| C$^3$LR (Oracle) | None | ✗ | - | 200 | 800 | **66.86** |
| C$^3$LR | $K$-means ($K = 5$) | ✓ | 3 | 200 | 800 | 62.14 |
| C$^3$LR | $K$-means ($K = 5$) | ✗ | - | 200 | 800 | 62.19 |
| C$^3$LR | $K$-means ($K = 10$) | ✗ | - | 200 | 800 | 61.69 |
| C$^3$LR | $K$-means ($K = 25$) | ✓ | 3 | 200 | 800 | 62.68 |
| C$^3$LR | $K$-means ($K = 25$) | ✗ | - | 200 | 800 | 63.60 |
| C$^3$LR | HDBSCAN | ✓ | 3 | 200 | 800 | 62.46 |
| C$^3$LR | HDBSCAN | ✗ | - | 200 | 800 | 62.44 |
| C$^3$LR | HDBSCAN + $k$-rJd | ✓ | 2 | 200 | 800 | 63.79 |
| C$^3$LR | HDBSCAN + $k$-rJd | ✗ | - | 200 | 800 | <u>64.81</u> |

**Table 10.1:** Table comparing design choices for C$^3$LR though accuracy (%) on (5-way, 5-shot) classification tasks. $L$ is number of source images, $B$ is the total number of images including augmentations. Style: **best** and <u>second best</u>.

Table 10.1 shows the performance of C$^3$LR with $K$-means and HDBSCAN as clustering algorithms of choice. The best performing variant of C$^3$LR is the **oracle** variant, this variant of the model provided access to the true labels of the data. The true labels allowed the creation of the ideal cluster centres. To understand why this is, we shall elaborate on lines 9 and 10 in Algorithm 1. In line 9, HDBSCAN processes the embeddings and groups them, when we say that an element belongs to cluster $\mathbf{C}_p$ it also implies that $\mathbf{C}_p$ functions as a predicted label for that element. Instead of using predicted labels, the oracle variant uses actual labels. Since we know the actual grouping of elements, we can calculate the ideal cluster centres on line 10. The purpose of the oracle model is to gauge the potential maximum performance that could be achieved using the training algorithm in question if the model were given all available label information without changing any other aspects of the training algorithm. It gives us a theoretical maximum that we know the training regime is capable of reaching. Ideally we want non-oracle models to be as close as possible to the performance of the oracle variant.

As shown in Table 10.1, the $K$-means algorithm has been tested with various values of $K$. We tried values of $K$ that seemed reasonable, however the results were not promising and were the same as that of ProtoCLR. The choice of $K$ was kept to at most 25, because we wanted a cluster to have sufficient samples for the calculation of loss $\mathcal{L}_1$. Choosing the ideal value for $K$ is also an additional overhead; for example, a value of $K$ that works well with *mini*-ImageNet may not work well for *tiered*ImageNet. We were unable to test the combination of $K$-means with $k$-reciprocal Jaccardian distance ($k$-rJd) because the implementation of $K$-means in Scikit-learn does not take a custom distance matrix as input (Pedregosa et al. 2011).

With the shortcomings of $K$-means in mind, we wanted to use a smarter clustering algorithm that could take custom distance matrices and automatically discover the ideal number of clusters in a given batch of size $B$. We decided to explore HDBSCAN (McInnes, Healy, and Astels 2017) as it can find clusters with variable densities and is also less prone to noise than DBSCAN (Ester et al. 1996; Schubert et al. 2017).

Choosing HDBSCAN did not immediately pay off; however, we did notice that the performance of C$^3$LR was more stable with HDBSCAN since we no longer needed to determine the optimal number of clusters required by trial and error. The stability given to us by HDBSCAN gave us the confidence to investigate other changes in C$^3$LR that could improve performance. A natural thought was to refine the neighbourhood of each sample so that it is close to elements that are similar to itself, since this would allow HDBSCAN to discover better clusters. Therefore, we took inspiration from the method developed by Ji et al. (2019) and utilised $k$-reciprocal Jaccardian distance ($k$-rJd) based re-ranking of elements (Zhong et al. 2017). $k$-rJd takes into account the reciprocal relationship between data points and is

therefore a stricter rule to measure whether two feature vectors match or not, thereby aiding HDBSCAN in finding better clusters.

It is well known that both $K$-means and HDBSCAN suffer from the curse of dimensionality (Kriegel, Kröger, and Zimek 2009; McInnes, Healy, and Astels 2017); to mitigate the effects of dimensionality, we explored the use of UMAP (McInnes, Healy, and Melville 2018) prior to the re-ranking and clustering steps on the lines $8-9$. The idea was to apply UMAP to $f_\phi(\boldsymbol{x}) \in \mathbb{R}^{1600}$ and reduce the dimensionality to $\mathbb{R}^3$ or $\mathbb{R}^2$. Unfortunately, we did not see significant performance improvements. The results of this are also given in Table 10.1. The column "UMAP $\mathbb{R}^{\cdot}$" refers to the number of UMAP dimensions.

## 10.2. Shortcomings of C$^3$LR

Although C$^3$LR is better than its competitors, it still has some shortcomings that we shall discuss in this section.

### 10.2.1. Unstable Clusters

As discussed in Section 10.1, we decided to use HDBSCAN as it provided a *relatively* more stable performance than $K$-means. However, HDBSCAN frequently generated massive clusters, as there is no way to constrain how many elements can be placed in a cluster. This meant that members of certain clusters were not really similar, leading to a sub-optimal effect of loss $\mathcal{L}_1$ on the learning process. From Section 10.2.2 and Figure 10.1 we can see the sheer amount of points classified as noise hovers around the $400$ mark. It must be noted that C$^3$LR has an effective batch size $B = 800$, this means in every training iteration roughly half the data was discarded as noise by HDBSCAN. Only the remaining amount of data played an active role in calculation of $\mathcal{L}_1$. Moreover, running the HDBSCAN algorithm for every batch is a time consuming and computationally heavy process.

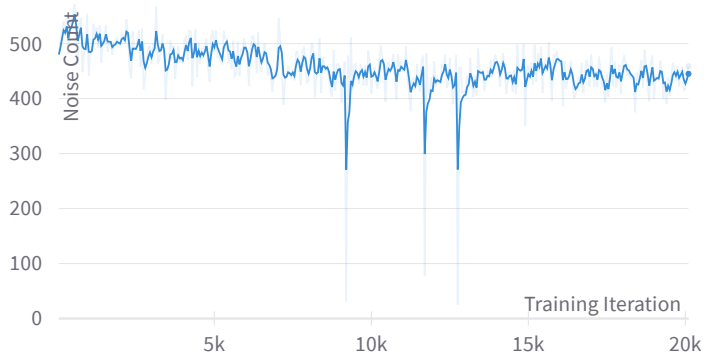



**Figure 10.1:** This graph shows the amount of points clustered as noise by HDBSCAN. We can see that roughly $400$ points are always classified as noise, even after $20,000$ training iterations.

### 10.2.2. Partial Compatibility with Gradient Based Learning

A major drawback of C$^3$LR is that the re-ranking and clustering steps are not compatible with gradient based learning. This means that any operations performed by these steps are not recorded in the computational graph tracked by PyTorch (Paszke et al. 2019). A computational graph keeps track of all mathematical operations performed on a variable; during backpropagation this computational graph is used to calculate the gradients. Due to this, information that is valuable to the model is lost between lines $8$ and $9$. The loss $\mathcal{L}_1$ is partially able to address this issue, however, it is far from ideal.

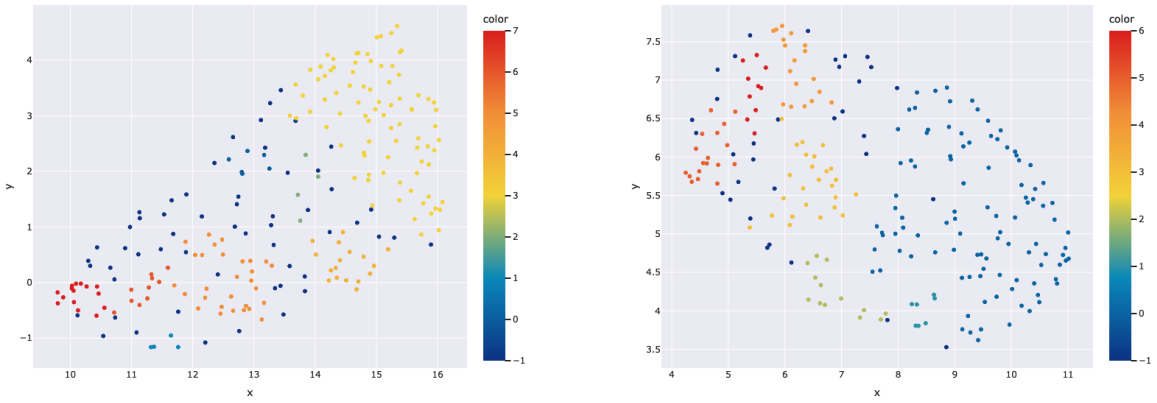



**Figure 10.2:** Two UMAP plots showing the clusters generated by HDBSCAN (McInnes, Healy, and Astels 2017). Notice the massive yellow (left) and blue (right) clusters, it is unlikely all elements in them are truly similar. Unfortunately, due to the variable number of clusters generated by HDBSCAN, the legend is shown as a continuous colour scale to avoid repetition of colours. Cluster "$-1$" contains noisy points.

# 11

# SAMPTransfer: Additional Materials

This chapter covers the motivation for SAMPTransfer's design choices as well as additional experiments with different types of graph neural networks.

Based on the discussion in Chapter 10, there are three main issues with C$^3$LR that we have to address:

1. There is no constraint on the amount of elements that can be placed within a cluster
2. It is computationally heavy to compute new clusters in each training step
3. Partial compatibility with gradient-based learning hinders the network's learning ability (previous knowledge about re-ranking and clustering does not carry over).

These problems with C$^3$LR formed the basis for certain design choices with respect to SAMPTransfer. In order to tackle problem (1) and (2), we do away with the clustering step and instead rely on self-attention based message passing (SAMP) to discover and refine features that belong to the same overarching class. In C$^3$LR, the re-ranking and clustering steps were used to find similar data points in a batch. The same can be achieved by means of a SAMP layer. SAMP is able to detect similar images in a batch which replaces clustering and re-ranking. It does this by creating a graph out of the image embeddings and then running a message-passing step. This also allows it to further refine their features to bring them even closer together, which was previously explicitly enforced through a modified contrastive loss $\mathcal{L}_1$ in C$^3$LR. Coming to problem (3), as described in Chapter 6, graph neural networks are fully compatible with gradient based learning and optimisation.

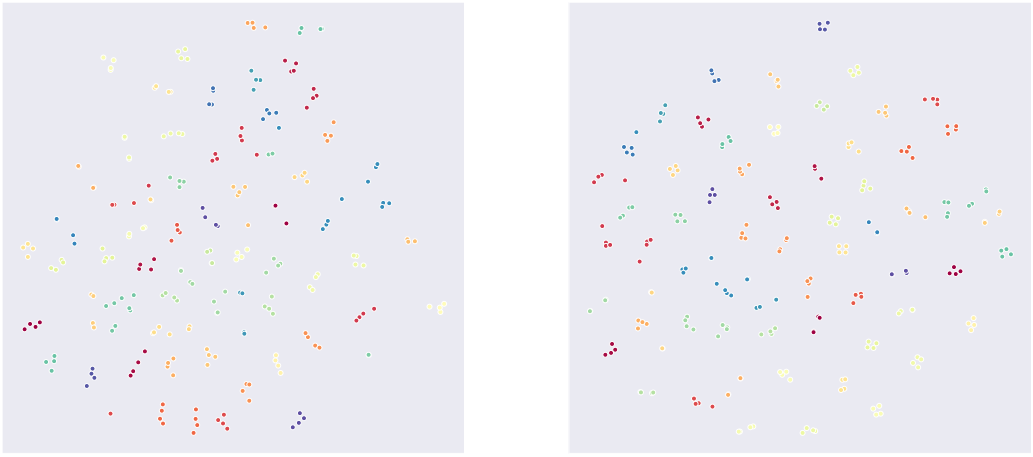
## 11.1. Why SAMP?

Table 11.1 shows the performance of three different types of GNNs. Before arriving on SAMP, we also tested the dynamic edge convolutions (Wang et al. 2019) and LatentGNN (S. Zhang, X. He, and Yan 2019) layers.

| Backbone | GNN Type | Accuracy |
|---|---|---|
| Conv4 | EdgeConv (2019) | 57.76 $\pm$ 0.81 |
| Conv4 | LatentGNN (2019) | 61.07 $\pm$ 0.66 |
| Conv4 | SAMP | **64.67** $\pm$ 2.65 |

**Table 11.1:** Comparision between three types of GNN layers. Accuracy ($\%$ $\pm$ std.) values are for ($5$-way, $5$-shot) *mini*-ImageNet classification tasks.

We chose the EdgeConv layer because it has the ability to dynamically update the graph at



**(a)** CNN features. **(b)** SAMP refined features.

**Figure 11.1:** Each point is an image representation that has three neighbours which are its augmentations. The UMAP plot on the left shows features generated by the Conv4b backbone, the points are clearly more spread out. The flattened output of the Conv4b backbone are used as input to the SAMP layer. The UMAP plot on the right generated by using the SAMP layer outputs.

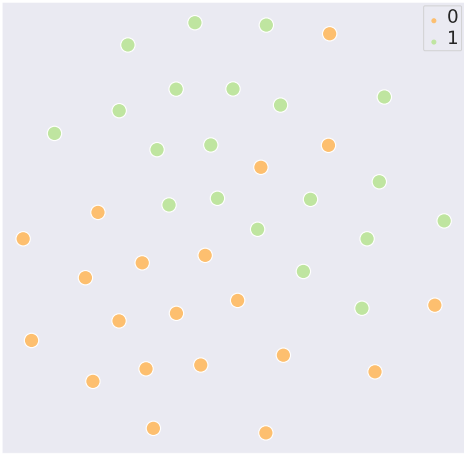
every step. EdgeConv was originally developed to be used in processing 3D point cloud data, where it was meant to grasp the 3D structure and relations between the points of a scanned item. We expected it to perform well with image feature vectors; however, due to the data changing too rapidly between training steps, it proved challenging for the EdgeConv layer to learn the similarities and relationships between images.

The next layer that we tested was the LatentGNN layer (S. Zhang, X. He, and Yan 2019). It was developed for the express purpose of modelling non-local contextual relations between visual feature maps. The key idea is to introduce a latent space to reduce the complexity of the graph, which allows the use of a low-rank representation for the graph affinity matrix. S. Zhang, X. He, and Yan (2019) insert this layer at intermediate points in a ResNet-50 and ResNet-101 (K. He, X. Zhang, et al. 2015). The layer projects square feature maps to latent space as feature vectors and then applies message passing steps to refine the feature vectors and finally converts them back to square feature maps for further processing by convolution blocks. Although this layer showed promising results, it requires a LatentGNN layer to be placed between convolutional layers, making a simple Conv4 unreasonably complex. However, the results were close enough to $C^3$LR and other competitors that it made natural sense to explore this direction further.

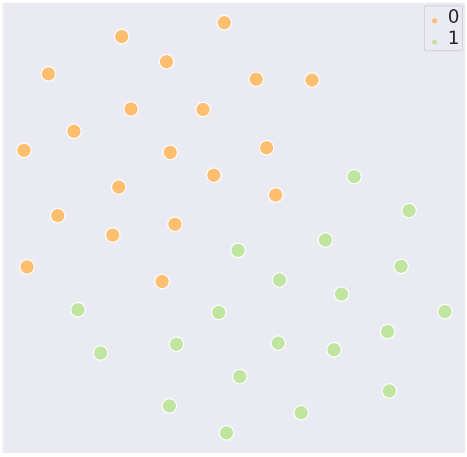
Ultimately, we arrive at a variant of the graph attention (GAT) layer (Veličković et al. 2018) that we have coined as SAMP. Layers similar to SAMP have been formulated under various names by (Brody, Alon, and Yahav 2021; Seidenschwarz, Elezi, and Leal-Taixé 2021), among others. To the best of our knowledge, no one seems to have used it in the context of unsupervised few-shot learning. It is simpler to use than LatentGNN (S. Zhang, X. He, and Yan 2019) and with minimal tuning the performance was within the margin of error of $C^3$LR.

### 11.1.1. Link with $C^3$LR

In $C^3$LR the re-ranking and clustering steps are used to find similar images in the embedding space and the loss $\mathcal{L}_1$ is used to bring these similar images closer together. In essence, we are looking beyond single instances by using these two steps. In SAMPTransfer, both these steps can be replaced by a SAMP layer. Conceptually, a SAMP layer is looking for similar images present in the batch and bringing them closer together by refining the features of the similar images. It doesn't need a modified contrastive loss to do so and instead only needs a standard contrastive loss (T. Chen et al. 2020).




**(a)** UMAP plot of CNN features.




**(b)** UMAP plot of SAMP refined features.

**Figure 11.2:** The figure on the left shows the CNN features of a support and query set from (2-way, 5-shot) task drawn from the *mini*-ImageNet validation set. Note that there are 15 query samples per class. The figure on the right shows the SAMP refined features where we can see two clearly separated clusters of points whereas on the right there are a few stray orange points. We choose $N = 2$ so that the effect of SAMP is more prominent and immediately visible.

### 11.1.2. SAMP in Action

In Figure 11.1 the plot on the right shows image augmentations tightly grouped around the source image. From this we can infer that the SAMP layer is able to recognise similar images and refine their features so that they are closer in the representation space.

Similar to Figure 11.1, Figure 11.2 shows the embeddings of a support and query set in a (2-way, 5-shot) task. We can see that the CNN features on the left aren't as separable, several orange dots are in the area of the green cluster. However, when the CNN features are passed through a SAMP layer we can see that clusters form that are almost perfectly separable, indicating that the SAMP layer is working by looking beyond single instances and has learnt to refine features as required. The legend is not shown as there 64 classes on the figure.

## 11.2. Shortcomings of SAMPTransfer

Although SAMPTransfer puts forth an impressive performance, there are still some problems that need to be addressed.

### 11.2.1. Reliance on Batch Size

Like other contrastive learning approaches, such as SimCLR (T. Chen et al. 2020), SAMPTransfer too needs a reasonable batch size to be performant. SimCLR achieves its best performance with a batch size of 8192 (T. Chen et al. 2020), similarly we too notice that as our batch size is increased from 64 to 128 there is a noticeable benefit. This is because a larger batch size means access to a greater number of negative pairs that can be used in the contrastive loss. Some methods like MoCo (K. He, Fan, et al. 2020) and NNCLR (Dwibedi et al. 2021) make use of a memory module that helps reduce dependence on the batch size.

### 11.2.2. Loss of Spatial Information

Currently, the SAMP layers work on the flattened feature maps generated by the convolutional encoder. Although, this design is not inherently wrong it does lose the spatial information present in the 2-d feature maps. For example, this could be important when working on features extracted from two images depicting a cat and a bobcat[1]. Information regarding the specifics about the shape of their ears or body could all be lost with the flattening operation.

[1] https://www.nationalgeographic.com/animals/mammals/facts/bobcat

| Backbone | $p$ | $H$ | $L$ | $\beta$ | OT | Accuracy |
|---|---|---|---|---|---|---|
| Conv4b | 1 | 4 | 64 | 1.0 | ✓ | 71.42 $\pm$ 0.73 |
| Conv4b | 1 | 4 | 64 | 0.7 | ✓ | 71.41 $\pm$ 0.71 |
| Conv4 | 1 | 4 | 64 | 0.7 | ✓ | 69.61 $\pm$ 0.71 |
| Conv4 | 1 | 4 | 64 | 1.0 | ✓ | 67.60 $\pm$ 0.62 |
| Conv4 | 1 | 8 | 64 | 1.0 | ✓ | 63.59 $\pm$ 0.68 |
| Conv4b | 1 | 4 | 128 | 0.7 | ✓ | 72.52 $\pm$ 0.72 |
| Conv4 | 1 | 4 | 128 | 0.7 | ✓ | 68.33 $\pm$ 0.71 |
| Conv4 | 1 | 4 | 128 | 0.0 | ✓ | 52.81 $\pm$ 0.66 |
| Conv4b | 1 | 4 | 128 | 0.0 | ✓ | 72.44 $\pm$ 0.69 |
| Resnet-12 | 1 | 2 | 128 | 0.7 | ✓ | 69.83 $\pm$ 0.71 |
| ResNet-18 | 1 | 2 | 128 | 0.7 | ✓ | 70.91 $\pm$ 0.69 |
| ResNet-50 | 1 | 2 | 128 | 0.7 | ✓ | 74.05 $\pm$ 0.64 |
| Resnet-12 | 1 | 4 | 128 | 0.7 | ✓ | 71.93 $\pm$ 0.73 |
| ResNet-18 | 1 | 4 | 128 | 0.7 | ✓ | 71.2 $\pm$ 0.68 |
| ResNet-50 | 1 | 4 | 128 | 0.7 | ✓ | 73.27 $\pm$ 0.68 |

**Table 11.2:** Ablation study of various parameters on accuracy.

### 11.2.3. Poor Scaling with Larger Backbones

We see that SAMPTransfer scales poorly with larger backbones such as ResNet-12, ResNet-18, and ResNet-50 (K. He, X. Zhang, et al. 2015). The results for these backbones are shown in Table 11.2, which is a slightly modified table present in Chapter 3. From Table 11.2, we can see that only the largest ResNet, ResNet-50 manages to outperform a much smaller Conv4b network.

Although it is natural to presume that scaling to a larger backbone would also improve performance (K. He, X. Zhang, et al. 2015; Goyal et al. 2021), we see the opposite in action here. We hypothesise that this could be due to sub-optimal hyperparameters, including the choice of optimiser. However, due to the lack of time and resources we were unable to extensively investigate this behaviour.

# 12

# Conclusions

## 12.1. Future Work

Based on the limitations of our approaches discussed in Chapter 10 and Chapter 11, we have some future directions for future research. One of the simplest directions to explore would be to swap the SAMP layer in favour of Graph Attention V2 (Brody, Alon, and Yahav 2021) or Non-Parametric Transformers (Kossen et al. 2021). Both these methods claim to be more powerful than the variety of attention we are using in SAMP, and it would be interesting to see if these methods can learn to refine embeddings better than SAMP.

We also think that a memory module would be beneficial for SAMPTransfer, so that it can learn the manifold of the data space better. Similar to NNCLR (Dwibedi et al. 2021) or MoCo (K. He, X. Zhang, et al. 2015), SAMPTransfer could access examples that were seen in previous batches and use it to learn better semantic features. Another related possibility is to use a memory bank to learn prototypes based on the dataset. With each batch that is seen by the network, this memory bank can be updated with more representative prototypes. The prototypes can then be used in downstream tasks and even be fine-tuned if required. Furthermore, we could build a graph based memory system that can summarise and keep track of relations between seen entities. The summarisation process could involve storing prototypes as a graph, along with a few ($< 5$) examples each. These relations can be exploited during the training or testing process to find the optimal representation or class, respectively.

It is also worth exploring different kinds of pre-text tasks for self-supervision. One particularly interesting pre-text task would be a form of masked image modelling (K. He, Xinlei Chen, et al. 2022; Xie et al. 2022) with the aid of graphs. The high-level idea is to make patches out of each of the input images in the batch and have the network reconstruct it. We continue using the SAMP layer so that the network learns to reconstruct images by using information available in other images. The hope is that the network would be forced to look beyond single instances and "borrow" information available from other images in the batch.

## 12.2. Conclusion

Inspired by the human mind's propensity to quickly learn rich representations and relationships between known and unknown elements, this body of work introduces two novel ideas for self-supervised few-shot learning namely SAMPTransfer and C$^3$LR.

C$^3$LR looks beyond single instances by incorporating class cognisance through: (i) an unsupervised iterative re-ranking and clustering step, followed by (ii) an adjusted optimisation loss formulation. We demonstrate that our proposed approach (C$^3$LR) offers considerable performance improvement over its predecessor ProtoTransfer in both in-domain benchmarks like *mini*-ImageNet and Omniglot. C$^3$LR also showcases competitive performance on the more challenging cross-domain few-shot learn-

ing (CDFSL) benchmark.

While in SAMPTransfer, the proposed method utilises a variant of the graph attention network that we call SAMP in a contrastive learning scheme. The SAMP layer helps the model look beyond a single instance and find other similar items in an entire batch and jointly refine their embeddings. SAMP allows the model to extract richer semantic information across multiple images present in a mini-batch. Unlike $C^3$LR, SAMPTransfer no longer requires the re-ranking and clustering steps, making it more computationally efficient. Furthermore, SAMPTransfer also proposes the use of OpT-Tune, an optimal transport based task adaptation framework which helps in aligning support and query embeddings during test time without requiring any trainable parameters. With OpT-Tune, we prove that fine-tuning does not have to be unremarkable, instead it can play an active role in elevating the robustness of a pre-trained network.

Through SAMPTransfer and $C^3$LR, we have successfully shown that unsupervised few-shot learning can match the performance some of latest supervised methods. By having these approaches look beyond single instances, we mimicked a form of the human learning process and applied it to the unsupervised few-shot learning problem. With both these methods, we firmly believe to have taken a step forward in the long and arduous journey to close the gap between human and machine.

# Bibliography


[1] Gaspard Monge. “Mémoire sur la théorie des déblais et des remblais”. In: *Mem. Math. Phys. Acad. Royale Sci.* (1781), pp. 666–704 (cit. on p. 56).

[2] Frank L Hitchcock. “The distribution of a product from several sources to numerous localities”. In: *Journal of mathematics and physics* 20.1-4 (1941), pp. 224–230 (cit. on pp. 58, 59).

[3] Leonid Kantorovich. “On the transfer of masses (in Russian)”. In: *Doklady Akademii Nauk* 37.2 (1942), pp. 227–229 (cit. on pp. 57, 58).

[4] Warren S McCulloch and Walter Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133 (cit. on pp. 22, 24, 26).

[5] Herbert Robbins and Sutton Monro. “A Stochastic Approximation Method”. In: *The annals of mathematical statistics* 22 (3 Sept. 1951), pp. 400–407. ISSN: 0003-4851. DOI: 10.1214/AOMS/1177729586. URL: https://projecteuclid.org/journals/annals-of-mathematical-statistics/volume-22/issue-3/A-Stochastic-Approximation-Method/10.1214/aoms/1177729586.full (cit. on p. 26).

[6] J. Kiefer and J. Wolfowitz. “Stochastic Estimation of the Maximum of a Regression Function”. In: *The annals of mathematical statistics* 23 (3 Sept. 1952), pp. 462–466. ISSN: 0003-4851. DOI: 10.1214/AOMS/1177729392. URL: https://projecteuclid-org.tudelft.idm.oclc.org/journals/annals-of-mathematical-statistics/volume-23/issue-3/Stochastic-Estimation-of-the-Maximum-of-a-Regression-Function/10.1214/aoms/1177729392.full%20https://projecteuclid-org.tudelft.idm.oclc.org/journals/annals-of-mathematical-statistics/volume-23/issue-3/Stochastic-Estimation-of-the-Maximum-of-a-Regression-Function/10.1214/aoms/1177729392.short (cit. on p. 26).

[7] Richard Sinkhorn and Paul Knopp. “Concerning nonnegative matrices and doubly stochastic matrices”. In: *Pacific Journal of Mathematics* 21.2 (1967), pp. 343–348. ISSN: 00308730. DOI: 10.2140/PJM.1967.21.343 (cit. on p. 60).

[8] Kunihiko Fukushima. “Cognitron: A self-organizing multilayered neural network”. In: *Biological Cybernetics 1975 20:3* 20 (3 Sept. 1975), pp. 121–136. ISSN: 1432-0770. DOI: 10.1007/BF00342633. URL: https://link-springer-com.tudelft.idm.oclc.org/article/10.1007/BF00342633 (cit. on pp. 23, 29).

[9] Kunihiko Fukushima and Sei Miyake. “Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition”. In: *Competition and cooperation in neural nets*. Springer, 1982, pp. 267–285 (cit. on p. 29).

[10] Charles A. Micchelli. “Interpolation of scattered data: Distance matrices and conditionally positive definite functions”. In: *Constructive Approximation 1986 2:1* 2 (1 Dec. 1986), pp. 11–22. ISSN: 1432-0940. DOI: 10.1007/BF01893414. URL: https://link-springer-com.tudelft.idm.oclc.org/article/10.1007/BF01893414 (cit. on p. 26).

[11] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning representations by back-propagating errors”. In: *nature* 323.6088 (1986), pp. 533–536 (cit. on p. 27).

[12] Jurgen Schmidhuber. “Evolutionary Principles in Self-Referential Learning. On Learning now to Learn: The Meta-Meta-Meta...-Hook”. Diploma Thesis. Technische Universitat Munchen, Germany, 14 May 1987. URL: http://www.idsia.ch/~juergen/diploma.html (cit. on p. 50).

[13] Steven Pinker and Alan Prince. “On language and connectionism: Analysis of a parallel distributed processing model of language acquisition”. In: *Cognition* 28 (1-2 Mar. 1988), pp. 73–193. ISSN: 0010-0277. DOI: 10.1016/0010-0277(88)90032-7 (cit. on p. 23).

[14] G. Cybenko. “Approximation by superpositions of a sigmoidal function”. In: *Mathematics of Control, Signals and Systems 1989 2:4* 2 (4 Dec. 1989), pp. 303–314. ISSN: 1435-568X. DOI: 10.

1007/BF02551274. URL: https://link-springer-com.tudelft.idm.oclc.org/article/10.1007/BF02551274 (cit. on p. 26).
[15] Y. LeCun et al. “Backpropagation Applied to Handwritten Zip Code Recognition”. In: *Neural Computation* 1 (4 Dec. 1989), pp. 541–551. ISSN: 0899-7667. DOI: 10.1162/NECO.1989.1.4.541 (cit. on p. 29).
[16] Yann Brenier. “Polar factorization and monotone rearrangement of vector-valued functions”. In: *Communications on Pure and Applied Mathematics* 44.4 (June 1991), pp. 375–417. ISSN: 1097-0312. DOI: 10.1002/CPA.3160440402. URL: https://onlinelibrary-wiley-com.tudelft.idm.oclc.org/doi/full/10.1002/cpa.3160440402%20https://onlinelibrary-wiley-com.tudelft.idm.oclc.org/doi/abs/10.1002/cpa.3160440402%20https://onlinelibrary-wiley-com.tudelft.idm.oclc.org/doi/10.1002/cpa.3160440402 (cit. on p. 57).
[17] Kurt Hornik. “Approximation capabilities of multilayer feedforward networks”. In: *Neural Networks* 4 (2 Jan. 1991), pp. 251–257. ISSN: 0893-6080. DOI: 10.1016/0893-6080(91)90009-T (cit. on p. 26).
[18] Gary F Marcus et al. “Overregularization in language acquisition”. In: *Monographs of the society for research in child development* (1992), pp. i–178 (cit. on p. 4).
[19] Martin Ester et al. “A density-based algorithm for discovering clusters in large spatial databases with noise.” In: *kdd*. Vol. 96. 34. 1996, pp. 226–231 (cit. on p. 62).
[20] Zhu Xiaojin and Ghahramani Zoubin. “Learning from labeled and unlabeled data with label propagation”. In: *Tech. Rep., Technical Report CMU-CALD-02–107, Carnegie Mellon University* (2002) (cit. on p. 49).
[21] Jia Deng et al. “Imagenet: A large-scale hierarchical image database”. In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255 (cit. on pp. 4, 29).
[22] Hans-Peter Kriegel, Peer Kröger, and Arthur Zimek. “Clustering High-Dimensional Data: A Survey on Subspace Clustering, Pattern-Based Clustering, and Correlation Clustering”. In: *ACM Trans. Knowl. Discov. Data* 3.1 (Mar. 2009). ISSN: 1556-4681. DOI: 10.1145/1497577.1497578. URL: https://doi-org.tudelft.idm.oclc.org/10.1145/1497577.1497578 (cit. on p. 63).
[23] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on p. 62).
[24] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf (cit. on pp. 29, 53).
[25] Marco Cuturi. “Sinkhorn distances: Lightspeed computation of optimal transport”. In: *Advances in neural information processing systems* 26 (2013) (cit. on pp. 59, 60).
[26] Matthew D. Zeiler and Rob Fergus. “Visualizing and Understanding Convolutional Networks”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 8689 LNCS (PART 1 Nov. 2013), pp. 818–833. ISSN: 16113349. DOI: 10.48550/arxiv.1311.2901. URL: https://arxiv.org/abs/1311.2901v3 (cit. on p. 33).
[27] Kaiming He, Xiangyu Zhang, et al. “Deep Residual Learning for Image Recognition”. In: (Dec. 2015). URL: http://arxiv.org/abs/1512.03385 (cit. on pp. 53, 66, 68, 69).
[28] Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. “Human-level concept learning through probabilistic program induction”. In: *Science* 350.6266 (Dec. 2015), pp. 1332–1338. ISSN: 10959203. DOI: 10.1126/SCIENCE.AAB3050/SUPPL{_}FILE/LAKE-SM.PDF. URL: https://www.science.org (cit. on p. 3).
[29] Léon Bottou, Frank E. Curtis, and Jorge Nocedal. “Optimization Methods for Large-Scale Machine Learning”. In: (June 2016). DOI: 10.48550/arxiv.1606.04838. URL: https://arxiv.org/abs/1606.04838 (cit. on p. 28).
[30] Xi Chen et al. “Infogan: Interpretable representation learning by information maximizing generative adversarial nets”. In: *Advances in neural information processing systems* 29 (2016) (cit. on p. 53).

[31] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. “Adversarial feature learning”. In: *arXiv preprint arXiv:1605.09782* (2016) (cit. on p. 53).
[32] Vincent Dumoulin, Francesco Visin, and George E P Box. “A guide to convolution arithmetic for deep learning”. In: (Mar. 2016). DOI: 10.48550/arxiv.1603.07285. URL: https://arxiv.org/abs/1603.07285v2 (cit. on pp. 31, 32).
[33] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016 (cit. on pp. 27, 44).
[34] Thomas N Kipf and Max Welling. “Semi-supervised classification with graph convolutional networks”. In: *arXiv preprint arXiv:1609.02907* (2016) (cit. on p. 40).
[35] Michael Eickenberg et al. “Seeing it all: Convolutional network layers map the function of the human visual system”. In: *NeuroImage* 152 (May 2017), pp. 184–194. ISSN: 1053-8119. DOI: 10.1016/J.NEUROIMAGE.2016.10.001 (cit. on p. 29).
[36] Chelsea Finn, Pieter Abbeel, and Sergey Levine. “Model-agnostic meta-learning for fast adaptation of deep networks”. In: *International conference on machine learning*. PMLR. 2017, pp. 1126–1135 (cit. on pp. 49–51, 53).
[37] Leland McInnes, John Healy, and Steve Astels. “hdbscan: Hierarchical density based clustering.” In: *J. Open Source Softw.* 2.11 (2017), p. 205 (cit. on pp. 61–64).
[38] Erich Schubert et al. “DBSCAN revisited, revisited: why and how you should (still) use DBSCAN”. In: *ACM Transactions on Database Systems (TODS)* 42.3 (2017), pp. 1–21 (cit. on p. 62).
[39] Jake Snell, Kevin Swersky, and Richard Zemel. “Prototypical networks for few-shot learning”. In: *Advances in neural information processing systems* 30 (2017) (cit. on pp. 52, 55).
[40] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017) (cit. on pp. 37, 38, 41).
[41] Manzil Zaheer et al. “Deep sets”. In: *Advances in neural information processing systems* 30 (2017) (cit. on p. 36).
[42] Zhun Zhong et al. “Re-ranking person re-identification with k-reciprocal encoding”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1318–1327 (cit. on pp. 61, 62).
[43] Antreas Antoniou, Harrison Edwards, and Amos Storkey. “How to train your MAML”. In: *arXiv preprint arXiv:1810.09502* (2018) (cit. on p. 51).
[44] David Berthelot et al. “Understanding and improving interpolation in autoencoders via an adversarial regularizer”. In: *arXiv preprint arXiv:1807.07543* (2018) (cit. on p. 53).
[45] Mathilde Caron, Piotr Bojanowski, et al. “Deep clustering for unsupervised learning of visual features”. In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 132–149 (cit. on p. 53).
[46] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. “Unsupervised representation learning by predicting image rotations”. In: *arXiv preprint arXiv:1803.07728* (2018) (cit. on p. 44).
[47] Kyle Hsu, Sergey Levine, and Chelsea Finn. “Unsupervised learning via meta-learning”. In: *arXiv preprint arXiv:1810.02334* (2018) (cit. on pp. 49, 53).
[48] Ilya Kuzovkin et al. “Activations of deep convolutional neural networks are aligned with gamma band activity of human visual cortex”. In: *Communications Biology 2018 1:1* 1 (1 Aug. 2018), pp. 1–12. ISSN: 2399-3642. DOI: 10.1038/S42003-018-0110-Y. URL: https://www-nature-com.tudelft.idm.oclc.org/articles/s42003-018-0110-y (cit. on p. 29).
[49] Yanbin Liu et al. “Learning to propagate labels: Transductive propagation network for few-shot learning”. In: *arXiv preprint arXiv:1805.10002* (2018) (cit. on p. 49).
[50] Leland McInnes, John Healy, and James Melville. “Umap: Uniform manifold approximation and projection for dimension reduction”. In: *arXiv preprint arXiv:1802.03426* (2018) (cit. on p. 63).
[51] Ryan L. Murphy et al. “Janossy Pooling: Learning Deep Permutation-Invariant Functions for Variable-Size Inputs”. In: *7th International Conference on Learning Representations, ICLR 2019* (Nov. 2018). URL: https://arxiv.org/abs/1811.01900v3 (cit. on pp. 34, 35).
[52] Victor Garcia Satorras and Joan Bruna Estrach. “Few-Shot Learning with Graph Neural Networks”. In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=BJj6qGbRW (cit. on p. 49).
[53] Petar Veličković et al. “Graph Attention Networks”. In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=rJXMpikCZ (cit. on pp. 40, 41, 66).

[54] Antreas Antoniou and Amos Storkey. “Assume, augment and learn: Unsupervised few-shot meta-learning via random labels and data augmentation”. In: *arXiv preprint arXiv:1902.09884* (2019) (cit. on p. 49).
[55] Yuki Markus Asano, Christian Rupprecht, and Andrea Vedaldi. “Self-labelling via simultaneous clustering and representation learning”. In: *arXiv preprint arXiv:1911.05371* (2019) (cit. on p. 60).
[56] Guneet S Dhillon et al. “A baseline for few-shot image classification”. In: *arXiv preprint arXiv:1909.02729* (2019) (cit. on pp. 49, 54).
[57] Zilong Ji et al. “Unsupervised few-shot learning via self-supervised training”. In: *arXiv preprint arXiv:1912.12178* (2019) (cit. on pp. 49, 62).
[58] Siavash Khodadadeh, Ladislau Boloni, and Mubarak Shah. “Unsupervised meta-learning for few-shot image classification”. In: *Advances in neural information processing systems* 32 (2019) (cit. on pp. 49, 54).
[59] Jongmin Kim et al. “Edge-labeling graph neural network for few-shot learning”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 11–20 (cit. on p. 49).
[60] Juho Lee et al. “Set Transformer: A Framework for Attention-Based Permutation-Invariant Neural Networks”. In: *International Conference on Machine Learning (ICML)* (2019) (cit. on p. 36).
[61] 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-performance-deep-learning-library.pdf (cit. on p. 64).
[62] Gabriel Peyré, Marco Cuturi, et al. “Computational optimal transport: With applications to data science”. In: *Foundations and Trends® in Machine Learning* 11.5-6 (2019), pp. 355–607 (cit. on pp. 57, 58).
[63] Yue Wang et al. “Dynamic graph cnn for learning on point clouds”. In: *Acm Transactions On Graphics (tog)* 38.5 (2019), pp. 1–12 (cit. on p. 65).
[64] Songyang Zhang, Xuming He, and Shipeng Yan. “Latentgnn: Learning efficient non-local relations for visual recognition”. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 7374–7383 (cit. on pp. 65, 66).
[65] Malik Boudiaf et al. “Information maximization for few-shot learning”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 2445–2457 (cit. on pp. 49, 54).
[66] Tom Brown et al. “Language models are few-shot learners”. In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901 (cit. on pp. 3, 4).
[67] Mathilde Caron, Ishan Misra, et al. “Unsupervised learning of visual features by contrasting cluster assignments”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 9912–9924 (cit. on pp. 4, 43, 60).
[68] Ting Chen et al. “A simple framework for contrastive learning of visual representations”. In: *International conference on machine learning*. PMLR. 2020, pp. 1597–1607 (cit. on pp. 45, 66, 67).
[69] Alexey Dosovitskiy et al. “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”. In: (Oct. 2020). DOI: 10.48550/arxiv.2010.11929. URL: https://arxiv.org/abs/2010.11929v2 (cit. on pp. 29, 53).
[70] Vijay Prakash Dwivedi and Xavier Bresson. “A Generalization of Transformer Networks to Graphs”. In: (Dec. 2020). DOI: 10.48550/arxiv.2012.09699. URL: https://arxiv.org/abs/2012.09699v2 (cit. on p. 38).
[71] Leo Gao et al. “The pile: An 800gb dataset of diverse text for language modeling”. In: *arXiv preprint arXiv:2101.00027* (2020) (cit. on p. 4).
[72] Kaiming He, Haoqi Fan, et al. “Momentum contrast for unsupervised visual representation learning”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 9729–9738 (cit. on p. 67).
[73] Carlos Medina, Arnout Devos, and Matthias Grossglauser. “Self-supervised prototypical transfer learning for few-shot classification”. In: *arXiv preprint arXiv:2006.11325* (2020) (cit. on pp. 49, 54).
[74] Yonglong Tian et al. “Rethinking few-shot image classification: a good embedding is all you need?” In: *European Conference on Computer Vision*. Springer. 2020, pp. 266–282 (cit. on pp. 49, 54).

[75] Ling Yang et al. “Dpgn: Distribution propagation graph network for few-shot learning”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 13390–13399 (cit. on p. 49).
[76] Imtiaz Ziko et al. “Laplacian regularized few-shot learning”. In: *International conference on machine learning*. PMLR. 2020, pp. 11660–11670 (cit. on pp. 49, 54).
[77] Shaked Brody, Uri Alon, and Eran Yahav. “How attentive are graph attention networks?” In: *arXiv preprint arXiv:2105.14491* (2021) (cit. on pp. 38, 40, 66, 69).
[78] Michael M. Bronstein et al. “Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges”. In: (2021). URL: http://arxiv.org/abs/2104.13478 (cit. on pp. 38–40).
[79] Da Chen et al. “Self-supervised learning for few-shot image classification”. In: *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2021, pp. 1745–1749 (cit. on pp. 49, 54).
[80] Rony Chow. *Yann Lecun: An early AI prophet*. Dec. 2021. URL: https://www.historyofdatascience.com/yann-lecun/ (cit. on p. 29).
[81] Debidatta Dwibedi et al. “With a little help from my friends: Nearest-neighbor contrastive learning of visual representations”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 9588–9597 (cit. on pp. 67, 69).
[82] Priya Goyal et al. “Self-supervised Pretraining of Visual Features in the Wild”. In: (Mar. 2021). DOI: 10.48550/arxiv.2103.01988. URL: https://arxiv.org/abs/2103.01988v2 (cit. on pp. 4, 43, 68).
[83] Yuqing Hu, Vincent Gripon, and Stéphane Pateux. “Leveraging the feature distribution in transfer-based few-shot learning”. In: *International Conference on Artificial Neural Networks*. Springer. 2021, pp. 487–499 (cit. on p. 60).
[84] Jannik Kossen et al. “Self-Attention Between Datapoints: Going Beyond Individual Input-Output Pairs in Deep Learning”. In: *arXiv:2106.02584* (2021) (cit. on p. 69).
[85] Dong Bok Lee et al. “Meta-GMVAE: Mixture of Gaussian VAE for Unsupervised Meta-Learning”. In: *ICLR*. 2021 (cit. on p. 49).
[86] Grace W. Lindsay. “Convolutional Neural Networks as a Model of the Visual System: Past, Present, and Future”. In: *Journal of Cognitive Neuroscience* 33 (10 Sept. 2021), pp. 2017–2031. ISSN: 0898-929X. DOI: 10.1162/JOCN_A_01544. URL: https://direct-mit-edu.tudelft.idm.oclc.org/jocn/article/33/10/2017/97402/Convolutional-Neural-Networks-as-a-Model-of-the (cit. on p. 29).
[87] Jenny Denise Seidenschwarz, Ismail Elezi, and Laura Leal-Taixé. “Learning intra-batch connections for deep metric learning”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 9410–9421 (cit. on p. 66).
[88] Thalles Santos Silva. “A Few Words on Representation Learning”. In: *https://sthalles.github.io* (2021). URL: https://sthalles.github.io/a-few-words-on-representation-learning/ (cit. on p. 46).
[89] Aston Zhang et al. “Dive into Deep Learning”. In: *arXiv preprint arXiv:2106.11342* (2021) (cit. on p. 25).
[90] Aaron Daniel Cohen et al. “LaMDA: Language Models for Dialog Applications”. In: (2022) (cit. on p. 3).
[91] Debasmit Das, Sungrack Yun, and Fatih Porikli. “ConFeSS: A Framework for Single Source Cross-Domain Few-Shot Learning”. In: *International Conference on Learning Representations*. 2022. URL: https://openreview.net/forum?id=zRJu6mU2BaE (cit. on pp. 49, 54).
[92] Kaiming He, Xinlei Chen, et al. “Masked autoencoders are scalable vision learners”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 16000–16009 (cit. on p. 69).
[93] Edward Wagstaff et al. “Universal approximation of functions on sets”. In: *Journal of Machine Learning Research* 23.151 (2022), pp. 1–56 (cit. on pp. 35, 36).
[94] Zhenda Xie et al. “Simmim: A simple framework for masked image modeling”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 9653–9663 (cit. on p. 69).
[95] Han-Jia Ye, Lu Han, and De-Chuan Zhan. “Revisiting Unsupervised Meta-Learning via the Characteristics of Few-Shot Tasks”. In: *IEEE Transactions on Pattern Analysis and Machine Intelli-*

*gence* (2022), pp. 1–1. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2022.3179368. URL: https://ieeexplore.ieee.org/document/9786650/ (cit. on p. 49).

[96] Tianyuan Yu et al. “Hybrid Graph Neural Networks for Few-Shot Learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 36. 3. 2022, pp. 3179–3187 (cit. on p. 49).