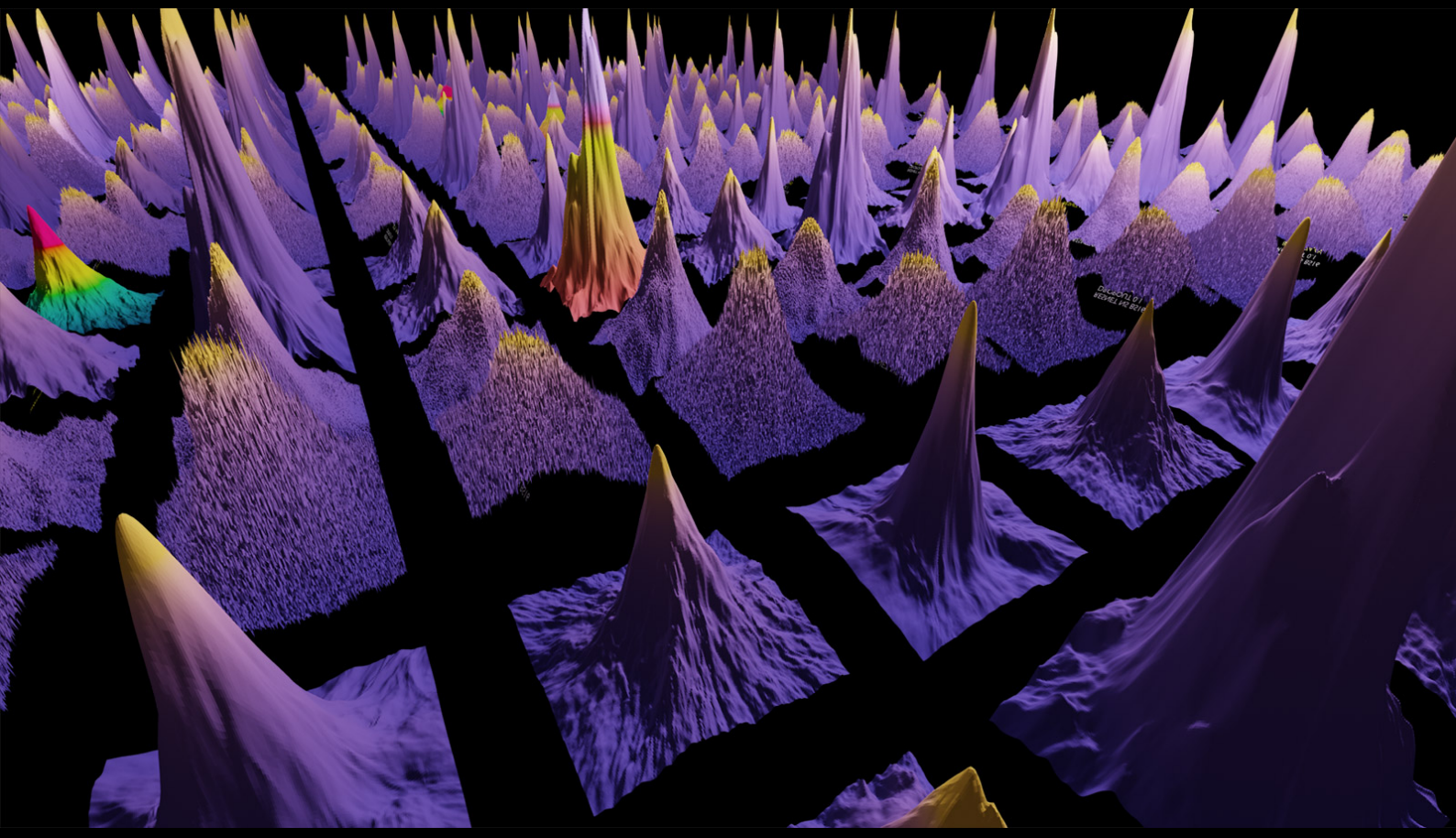
TRIDENT

Transductive Variational Inference of Decoupled Latent Variables for Few Shot Classification

Anuj Singh



TRIDENT

Transductive Variational Inference of Decoupled Latent Variables for Few Shot Classification

by

Anuj Singh

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Friday August 26, 2022 at 15:00.

Student number:	5305926	
Project duration:	November 16, 2021 – August 26, 2022	
Thesis committee:	Dr. ir. H. Jamali-Rad,	TU Delft and Shell, Supervisor
	Dr. ir. J.C. van Gemert,	TU Delft, Advisor
	Prof. dr. ir. G.J.T. Leus ,	TU Delft, External comittee member

TU Delft

Preface

The current report "TRIDENT: Transductive Variational Inference of Decoupled Latent Variables for Few Shot Classification" presents the work done for my Master thesis project. The research was conducted within the Computer Vision Lab at TU Delft, under the supervision of Dr. J.C. van Gemert and the daily supervision of Dr. H. Jamali-Rad.

This report has been structured in a way that the first part shows the scientific article, containing the motivation, related work, methods, experiments, and the results of the research. The latter part is used to discuss the relevant fundamental concepts and domain specific information useful in understanding the scientific paper.

The past two years of my masters account for the steepest learning curve I have ever been on in my life. This duration also accounts for the most rewarding two years of my life, where I discovered my love for machine learning research.

First and foremost, I would like to express my deepest appreciation for Dr. Hadi Jamali-Rad, who has not only been my daily supervisor, but also my mentor. You have been like a wise elder brother to me by looking out for me and guiding me on every step of the path. I can't thank you enough for the time and effort you have invested in this thesis and the support you continue to provide me in my career. I want to thank Dr. J.C. van Gemert for his crystal-clear teaching and thought provoking courses on Deep Learning, Computer Vision. I would also like to thank Dr. G.J.T. Leus for his interest in evaluating this work by participating in my defense committee.

Finally, and most importantly, I would like to express my gratitude for my family. They have afforded me the luxury of focusing on my research and education by supporting me emotionally, morally, financially and by loving me unconditionally.

Anuj Singh
Delft, August 2022

List of Publications

[1] Anuj Singh, Hadi Jamali-Rad, "Transductive Decoupled Variational Inference for Few-Shot Classification", submitted to Proceedings of the AAAI Conference on Artificial Intelligence, 2023.

[2] Ojas 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.

[3] Hadi Jamali-Rad, Mohammad Abdizadeh and Anuj Singh, "Federated Learning With Taskonomy for Non-IID Data," in IEEE Transactions on Neural Networks and Learning Systems(TNNLS), 2022.

Contents

1

Introduction

Deep learning algorithms are usually data hungry and require massive amounts of training data to reach a satisfactory level of performance on any task. In contrast, humans learn new skills much more efficiently and can also use learned concepts in richer ways than conventional algorithms, for example: imagination, explanation and taking action. Kids who have seen cats and birds only a few times can quickly tell them apart. People who know how to ride a bike are likely to learn riding a motorcycle faster with little or even no demonstration. The fundamental challenge for deep learning algorithms lies in understanding and inculcating the two major aspects (Lake, Salakhutdinov, and Tenenbaum 2015) of human-level concept learning:

- How do people learn new concepts from just one or a few examples?
- How do people learn such abstract, rich, and flexible representations that generalize well across several tasks?

This is essentially what few-shot learning aims to solve. An ideal few-shot learning model is expected to be capable of generalizing or adapting well to new tasks and environments that have never been encountered during training time. The adaptation process on the unseen test tasks is essentially a mini learning session, which takes place with very limited exposure to the new task configurations. Ideally, the learnt model can complete new unseen tasks by learning the underlying concepts and generalizable abstractions behind the tasks. For this reason, few-shot learning is also referred to as *learning to learn*. An instantiation of few-shot learning is few-shot classification where the idea is to learn to predict correct class labels for a set of unlabeled data points (query set), given only a small set of labeled data points (support set). This query and support set together are called an *episode* or a *task*, and are drawn from the same data distribution.

A wealth of research methods have been proposed, aiming to solve this problem from various perspectives. **Metric learning** proposes to learn a shared feature extractor to embed the samples into a metric space of class prototypes (Vinyals et al. 2016; Snell, Swersky, and Zemel 2017; Sung et al. 2018; Wang et al. 2019; Bateni, Goyal, et al. 2020; J. Liu, Song, and Qin 2020; Bateni, Barber, et al. 2022). The input image samples into a lower dimensional embedding space to then classify the unlabelled samples based on the computation of some distance or similarity based metric. By parameterizing these embedding-space mappings with neural networks and using differentiable similarity based metrics for classification, the network can be made to classify images in a data-deficient setting by training it in an episodic manner. This facilitates quick and efficient generalisation on new and unseen tasks. Due to limited data per class, these prototypes suffer from sample-bias and fail to efficiently represent class characteristics. Furthermore, sharing a feature extractor across tasks implies that the discriminative information learnt from the seen classes is equally effective on any arbitrary unseen classes, which is not true in most cases.

Task-aware few-shot learning approaches (K. Lee et al. 2019; Li et al. 2019; Requeima et al. 2019; Ye et al. 2020; Bateni, Barber, et al. 2022) address these limitations by exploiting information hidden in the unlabeled data. This exploitation is facilitated by an adaptation mechanism that makes the support embeddings extracted from the seen classes useful for classification of the unseen query classes, particularly for that given task. This type of *customization* of embedding spaces helps align support and query feature vectors for better representation of task-specific discriminative information. This not only improves the discriminative ability of classifiers across tasks, but also alleviates the problem of overfitting on limited support set data since now information from the query set is also used in extracting features for each image in a task. Since the alignment of these embeddings is still subject to the relevance of the characteristics captured by the shared feature extractors, task-aware methods sometimes fail to extract meaningful representations particularly relevant to classification.

Apart from using transduction for task-aware feature extraction, there are methods that use **Transductive Inference** to classify the query samples by making use of the patterns and additional structural information present in them. In these approaches, labels are assigned to the query samples only and not beyond them, as opposed to their inductive counterparts where prediction is done on the entire feature space and not just on a few test samples (Yanbin Liu et al. 2019; Boudiaf et al. 2020; Dhillon et al. 2020; Ziko et al. 2020; Bateni, Barber, et al. 2022).

Optimization-Based Learning methods search for model parameters that are sensitive to the task objective functions for fast gradient-based adaptation to new tasks. Approaches like MAML (Finn, Abbeel, and Levine 2017) and its computationally light variants perform fast adaptation by backpropagating the meta-loss through the inner-loop of gradient descent steps computed on the support samples. SNAIL (Mishra et al. 2018) uses a combination of temporal convolutions and soft attention to aggregate information from past experience and to pinpoint specific pieces of information. LEO (Rusu et al. 2019) learns a low-dimensional, data-dependent latent embedding of model parameters and performs optimization-based meta learning in this space. However, these gradient-based meta learning techniques usually don't scale well with larger backbones and neural nets such as ResNet's and WRN's for feature extractors due their computational complexity of operating on second-order gradients, and are prone to overfitting as well Mishra et al. 2018. Furthermore, MAML inspired methods are also very sensitive to neural network architectures, often leading to instability during training, requiring arduous hyper-parameter searches to stabilize training and achieve high generalization (Antreas Antoniou, Harrison Edwards, and Amos J. Storkey 2019)

Probabilistic methods address sample-bias by relaxing finding point estimates to approximating data-dependent distributions of either high-dimensional model weights (Gordon et al. 2019; Nguyen, Do, and Carneiro 2019; Ravi and Beatson 2019; Hu et al. 2020) or lower-dimensional class prototypes (J. Zhang et al. 2019; Sun et al. 2021). Gradient-based meta-learning methods estimate parameters which have a high variance due to the small sample sizes of the available dataset. To work with this variance, a natural extension to finding point-estimates of parameters is modelling this uncertainty as the variance by treating these parameters as latent variables in a Bayesian framework. However, inferring a high-dimensional posterior of model parameters is inefficient in low-data regimes. Moreover, estimating distributions of class prototypes involves using hand-crafted non-parametric aggregation techniques which may not be well suited for every unseen task.

2

Scientific Article (TRIDENT)

Transductive Decoupled Variational Inference for Few-Shot Classification

Anuj Singh [1], **Hadi Jamali-Rad** [1, 2]

[1]Delft University of Technology, The Netherlands,
[2]Shell Global Solutions International B.V., Amsterdam, The Netherlands

Abstract

The versatility to learn from a handful of samples is the hallmark of human intelligence. Few-shot learning is an endeavour to transcend this capability down to machines. Inspired by the promise and power of probabilistic deep learning, we propose a novel variational inference network for few-shot classification (coined as TRIDENT) to decouple the representation of an image into *semantic* and *label* latent variables, and simultaneously infer them in an intertwined fashion. To induce *task-awareness*, as part of the inference mechanics of TRIDENT, we exploit information across both query and support images of a few-shot task using a novel built-in attention-based transductive feature extraction module (we call AttFEX). Our extensive experimental results corroborate the efficacy of TRIDENT and demonstrate that, using the simplest of backbones, it sets a new state-of-the-art in the most commonly adopted datasets *mini*ImageNet and *tiered*ImageNet (offering up to 4% and 5% improvements, respectively), as well as for the recent challenging cross-domain *mini*Imagenet $\rightarrow$ CUB scenario offering a significant margin (up to 20% improvement) beyond the best existing cross-domain baselines.

Introduction

Deep learning algorithms are usually data hungry and require massive amounts of training data to reach a satisfactory level of performance on any task. To tackle this limitation, few-shot classification aims to learn to classify images from various unseen tasks in a data-deficient setting. In this exciting space, *metric learning* proposes to learn a shared feature extractor to embed the samples into a metric space of class prototypes (Sung et al. 2018; Vinyals et al. 2016; Snell, Swersky, and Zemel 2017; Wang et al. 2019; Liu, Song, and Qin 2020; Bateni et al. 2020). Due to limited data per class, these prototypes suffer from sample-bias and fail to efficiently represent class characteristics. Furthermore, sharing a feature extractor across tasks implies that the discriminative information learnt from the seen classes are equally effective on any arbitrary unseen classes, which is not true in most cases. *Task-aware* few-shot learning approaches (Bateni et al. 2022; Ye et al. 2020) address these limitations by exploiting information hidden in the unlabeled data. As a result, the model learns task-specific embeddings by aligning the features of the labelled and unlabelled task instances for optimal distance metric based label assignment. Since the alignment of these embeddings is still subject to the relevance of the characteristics captured by the shared feature extractors, task-aware methods sometimes fail to extract meaningful representations particularly relevant to classification. *Probabilistic* methods address sample-bias by relaxing the need to find point estimates to approximate data-dependent distributions of either high-dimensional model weights (Nguyen, Do, and Carneiro 2019; Ravi and Beatson 2019; Gordon et al. 2019; Hu et al. 2020) or lower-dimensional class prototypes (Sun et al. 2021; Zhang et al. 2019). However, inferring a high-dimensional posterior of model parameters is inefficient in low-data regimes and estimating distributions of class prototypes involves using hand-crafted non-parametric aggregation techniques which may not be well suited for every unseen task.

Although fit for purpose, all these approaches seem to overlook an important perspective. An image is composed of different attributes such as style, design, and context, which are not necessarily relevant discriminative characteristics for classification. Here, we refer to these attributes as *semantic* information. On the other hand, other class-characterizing attributes (such as wings of a bird, trunk of an elephant, hump on a camel's back) are critical for classification, irrespective of context. We refer to such attributes as *label* information. Typically, contextual information is majorly governed by semantic attributes, whereas the label characteristics are subtly embedded throughout an image. In other words, semantic information can be predominantly present across an image, whereas *attending* to subtle label information determines how effective a classification algorithm would be. Thus, we argue that attention to label-specific information should be ingrained into the mechanics of the classifier, decoupling it from semantic information. This becomes even more important in a few-shot setting where the network has to quickly learn from little data. Building upon this idea, we propose **tr**ansductive variational **i**nference of **de**coupled late**nt** variables (coined as TRIDENT), to simultaneously infer decoupled label and semantic information using two intertwined variational networks. To induce task-awareness while constructing the variational inference mechanics of TRIDENT, we introduce a novel **at**ention-based **t**ransductive **f**eature **ex**traction module (we call AttFEX) which further enhances the discriminative power of the inferred label attributes. This way TRIDENT infers distribu-

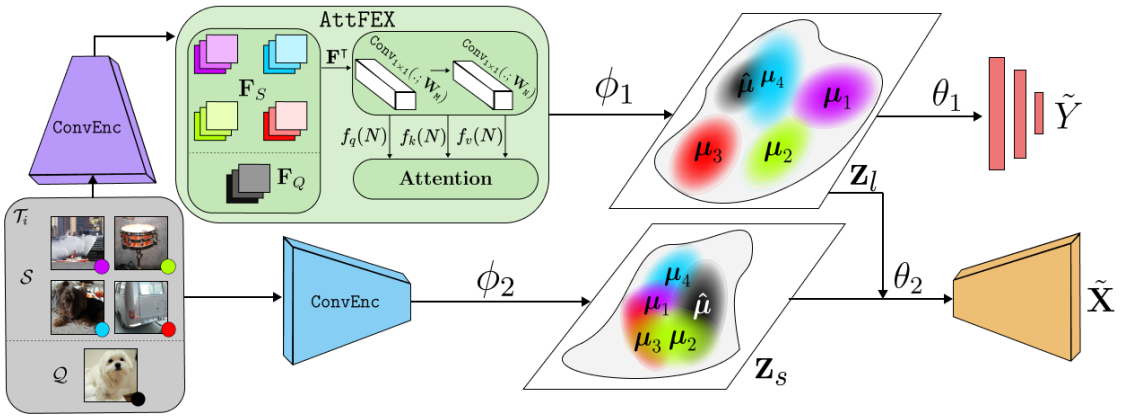


Figure 1: High-level process flow of TRIDENT. Inferred label latent variable $\mathbf{z}_l$ contains class-characterizing information, as is reflected by better separation of the distributions when compared to their semantic latent counterparts $\mathbf{z}_s$. AttFEX module generates *task-aware* feature maps by exploiting information from both support and query images, which compensates for the lack of label vectors Y in inferring $\mathbf{z}_l$.

tions instead of point estimates and injects a handcrafted inductive-bias into the network to guide the classification process. Our main contributions can be summarized as:

1. We propose TRIDENT, a variational inference network to simultaneously infer two salient *decoupled* attributes of an image (*label* and *semantic*), by inferring these two using two intertwined variational sub-networks (Fig. 1).
2. We introduce an attention-based transductive feature extraction module, AttFEX, to enable TRIDENT see through and compare all images within a task, inducing task-cognizance in the inference of label information.
3. We perform extensive evaluations to demonstrate that TRIDENT sets a new state-of-the-art by outperforming all existing baselines on the most commonly adopted datasets *mini*Imagenet and *tiered*Imagenet (up to 4% and 5%), as well as for the challenging cross-domain scenario of *mini*Imagenet $\rightarrow$ CUB (up to 20% improvement).

Related Work

Metric-based learning. This body of work revolves around mapping input samples into a lower-dimensional embedding space and then classifying the unlabelled samples based on a distance or similarity metric. By parameterizing these mappings with neural networks and using differentiable similarity metrics for classification, these networks can be trained in an episodic manner (Vinyals et al. 2016) to perform few-shot classification. Prototypical Nets (Snell, Swersky, and Zemel 2017), Simple Shot (Wang et al. 2019), Relation Networks (Sung et al. 2018), Matching Networks (Vinyals et al. 2016) variants of Graph Neural Nets (Satorras and Estrach 2018; Yang et al. 2020), Simple CNAPS (Bateni et al. 2020), are a few examples of seminal ideas in this space.

Transductive Feature-Extraction and Inference. Transductive feature extraction or task-aware learning is a variant of the metric-learning with an adaptation mechanism that *aligns* support and query feature vectors in the embedding space for better representation of task-specific discriminative information. This not only improves the discriminative ability of classifiers across tasks, but also alleviates the problem of overfitting on limited support set since information from the query set is also used for extracting features of images in a task. CNAPS (Requeima et al. 2019), Transductive-CNAPS (Bateni et al. 2022), FEAT (Ye et al. 2020), Assoc-Align (Afrasiyabi, Lalonde, and Gagné 2020), TPMN (Wu et al. 2021) and CTM (Li et al. 2019) are prime examples of such methods. Next to transduction for task-aware feature extraction, there are methods that use *transductive inference* to classify all the query samples at once by jointly assigning them labels, as opposed to their inductive counterparts where prediction is done on the samples one at a time. This is either done by iteratively propagating labels from the support to the query samples or by fine-tuning a pre-trained backbone using an additional entropy loss on all query samples, which encourages confident class predictions at query samples. TPN (Liu et al. 2019), Ent-Min (Dhillon et al. 2020), TIM (Boudiaf et al. 2020), Transductive-CNAPS (Bateni et al. 2022), LaplacianShot (Ziko et al. 2020), DPGN (Yang et al. 2020) and ReRank (SHEN et al. 2021) are a few notable examples in this space that usually report state-of-the-art results in certain few-shot classification settings (Liu et al. 2019).

Optimization-based meta-learning. These methods search for model parameters that are sensitive to task objective functions for fast gradient-based adaptation to new tasks. MAML (Finn, Abbeel, and Levine 2017), its variants (Rajeswaran et al. 2019; Nichol, Achiam, and Schulman 2018a) and SNAIL (Mishra et al. 2018) are a few prominent examples while LEO (Rusu et al. 2019) efficiently meta-updates its parameters in a lower dimensional latent space.

Probabilistic learning. The estimated parameters of typical gradient-based meta-learning methods discussed earlier (Finn, Abbeel, and Levine 2017; Rusu et al. 2019; Mishra et al. 2018; Nichol, Achiam, and Schulman 2018a; Rajeswaran et al. 2019), have high variance due to the small task sample size. To deal with this, a natural extension is to model the uncertainty by treating these parameters as latent variables in a Bayesian framework as proposed in Neural Statistician (Edwards and Storkey 2017), PLATIPUS (Finn, Xu, and Levine 2018), VAMPIRE (Nguyen,

Do, and Carneiro 2019), ABML (Ravi and Beatson 2019), VERSA (Gordon et al. 2019), SIB (Hu et al. 2020), SAMOVAR (Iakovleva, Verbeek, and Alahari 2020). Methods like ABPML (Sun et al. 2021) and VariationalFSL (Zhang et al. 2019) infer latent variables of class prototypes to perform classification and avoid inferring high-dimensional model parameters. ABPML (Sun et al. 2021) and VariationalFSL (Zhang et al. 2019) are the closest to our approach. In contrast to these two methods, we avoid handcrafting class-level aggregations, as well as we enhance variational inference by incorporating an inductive bias through decoupling of label and semantic information.

Problem Definition

Consider a labelled dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i) \,|\, i \in [1, N']\}$ of images $\mathbf{x}_i$ and class labels y_i. This dataset $\mathcal{D}$ is divided into three disjoint subsets: $\mathcal{D} = \{\mathcal{D}^{tr} \cup \mathcal{D}^{val} \cup \mathcal{D}^{test}\}$, respectively, referring to the training, validation, and test subsets. The validation dataset $\mathcal{D}^{val}$ is used for model selection and the testing dataset $\mathcal{D}^{test}$ for final evaluation. Following standard few-shot classification settings (Vinyals et al. 2016; Sung et al. 2018; Snell, Swersky, and Zemel 2017), we use episodic training on a set of tasks $\mathcal{T}_i \sim p(\mathcal{T})$. The tasks are constructed by drawing K random samples from N different classes, which we denote as an (N-way, K-shot) task. Concretely, each task $\mathcal{T}_i$ is composed of a *support* and a *query* set. The support set $\mathcal{S} = \{(\mathbf{x}^S_{kn}, y^S_{kn}) \,|\, k \in [1, K], n \in [1, N]\}$ contains K samples per class and the query set $\mathcal{Q} = \{(\mathbf{x}^Q_{kn}, y^Q_{kn}) \,|\, k \in [1, Q], n \in [1, N]\}$ contains Q samples per class. For a given task, the NQ query and NK support images are mutually exclusive to assess the generalization performance.

The Proposed Method: TRIDENT

Let us start with the high-level idea. The proposed approach is devised to learn meaningful representations that capture two pivotal characteristics of an image by modelling them as separate latent variables: (i) $\mathbf{z}_s$ representing *semantics*, and (ii) $\mathbf{z}_l$ embodying class *labels*. Inferring these two latent variables simultaneously allows $\mathbf{z}_l$ to learn meaningful distributions of class-discriminating characteristics *decoupled* from semantic features represented by $\mathbf{z}_s$. We argue that learning $\mathbf{z}_l$ as the sole latent variable for classification results in capturing a mixture of true label and other semantic information. This in turn can lead to sub-optimal classification performance, especially in a few-shot setting where the information per class is scarce and the network has to adapt and generalize quickly. By inferring decoupled label and semantics latent variables, we inject a handcrafted inductive-bias that incorporates only relevant characteristics, and thus, ameliorates the network's classification performance.

Generative Process

The directed graphical model in Fig. 2 illustrates the common underlying generative process p such that $p_i = p(\mathbf{x}_i, y_i \,|\, \mathbf{z}_{li}, \mathbf{z}_{si})$. For the sake of brevity, in the following we drop the sample index i as we always refer to terms associated with a single data sample. We work on

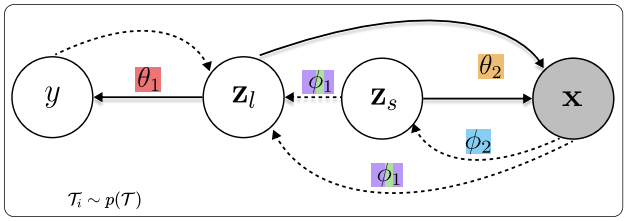


Figure 2: Generative Model of TRIDENT. Dotted lines indicate variational inference and solid lines refer to generative processes. The inference and generative parameters are color coded to correspond to their respective architectures indicated in Fig.1 and Fig.4.

the logical premise that the label latent variable $\mathbf{z}_l$ is responsible for generating class label as well as for image reconstruction, whereas the semantic latent variable $\mathbf{z}_s$ is only responsible for image reconstruction (solid lines in the figure). Formally, the data is explained by the generative processes: $p_{\theta_1}(y \,|\, \mathbf{z}_l) = \text{Cat}(y \,|\, \mathbf{z}_l)$ and $p_{\theta_2}(\mathbf{x} \,|\, \mathbf{z}_l, \mathbf{z}_s) = g_{\theta_2}(\mathbf{x}; \mathbf{z}_l, \mathbf{z}_s)$, where Cat(.) refers to a multinomial distribution and $g_{\theta_2}(\mathbf{x}; \mathbf{z}_l, \mathbf{z}_s)$ is a suitable likelihood function such as a Gaussian or Bernoulli distribution. The likelihoods of both these generative processes are parameterized using deep neural networks and the priors of the latent variables are chosen to be standard multivariate Gaussian distributions (Kingma and Welling 2014; Kingma et al. 2014): $p(\mathbf{z}_s) = \mathcal{N}(\mathbf{z}_s \,|\, \mathbf{0}, \mathbf{I})$ and $p(\mathbf{z}_l) = \mathcal{N}(\mathbf{z}_l \,|\, \mathbf{0}, \mathbf{I})$.

Variational Inference of Decoupled $\mathbf{Z}_l$ and $\mathbf{Z}_s$

Computing exact posterior distributions is intractable due to high dimensionality and non-linearity of the deep neural network parameter space. Following (Kingma and Welling 2014; Kingma et al. 2014), we instead construct an approximate posterior over the latent variables by introducing a fixed-form distribution $q(\mathbf{z}_l, \mathbf{z}_s \,|\, \mathbf{x}, y)$ parameterized by ϕ. By using $q_\phi(.)$ as an inference network, the inference is rendered tractable, scalable and amortized since ϕ now acts as the global variational parameter. We assume q_ϕ has a factorized form $q_\phi(\mathbf{z}_s, \mathbf{z}_l \,|\, \mathbf{x}, y) = q_{\phi_1}(\mathbf{z}_l \,|\, \mathbf{x}, \mathbf{z}_s)\, q_{\phi_2}(\mathbf{z}_s \,|\, \mathbf{x})$, where $q_{\phi_1}(.), q_{\phi_2}(.)$ are assumed to be multivariate Gaussian distributions. As is also depicted in Fig. 2, we use $\mathbf{z}_s$ as input to $q_{\phi_1}(.)$ to infer $\mathbf{z}_l$ because of their conditional dependence given $\mathbf{x}$. This way we forge a path to allow *necessary* semantic latent information flow through the label inference network. On the other hand, the opposite direction (using $\mathbf{z}_l$ to infer $\mathbf{z}_s$) is unnecessary, because label information does not directly contribute to the extraction of semantic features. We will further reflect on this design choice in the next subsection. Neural networks are then used to parameterize both inference networks as:

$$
\begin{aligned}
q_{\phi_2}(\mathbf{z}_s \,|\, \mathbf{x}) &= \mathcal{N}\left(\mathbf{z}_s \,|\, \boldsymbol{\mu}_{\phi_2}(\mathbf{x}), diag(\boldsymbol{\sigma}^2_{\phi_2}(\mathbf{x}))\right), \\
q_{\phi_1}(\mathbf{z}_l \,|\, \mathbf{x}, \mathbf{z}_s) &= \mathcal{N}\left(\mathbf{z}_l \,|\, \boldsymbol{\mu}_{\phi_1}(\mathbf{x}, \mathbf{z}_s), diag(\boldsymbol{\sigma}^2_{\phi_1}(\mathbf{x}, \mathbf{z}_s))\right).
\end{aligned} \tag{1}
$$

To find the optimal *approximate* posterior, we derive the evidence lower bound (ELBO) on the marginal likelihood of

the data to form our objective function:

$$
\begin{aligned}
p(\mathbf{x}, y) &= \iint p(\mathbf{x}, y \,|\, \mathbf{z}_s, \mathbf{z}_l)\, p(\mathbf{z}_s, \mathbf{z}_l)\, d\mathbf{z}_s\, d\mathbf{z}_l, \\
&= \mathbb{E}_{q(\mathbf{z}_s, \mathbf{z}_l \,|\, x)} \left[\frac{p(\mathbf{x} \,|\, \mathbf{z}_l, \mathbf{z}_s) p(y \,|\, \mathbf{z}_l) p(\mathbf{z}_l) p(\mathbf{z}_s)}{q(\mathbf{z}_l, \mathbf{z}_s \,|\, \mathbf{x})} \right]. \\
\ln p(\mathbf{x}, y) &\geqslant \mathbb{E}_{q(\mathbf{z}_s, \mathbf{z}_l | \mathbf{x})} \left[\ln \left(\frac{p(\mathbf{x} \,|\, \mathbf{z}_l, \mathbf{z}_s) p(y \,|\, \mathbf{z}_l) p(\mathbf{z}_l) p(\mathbf{z}_s)}{q(\mathbf{z}_s, \mathbf{z}_l \,|\, \mathbf{x})} \right) \right], \\
&= \mathbb{E}_{q_{\phi_2}} \left[\mathbb{E}_{q_{\phi_1}} \left[\ln \left(\frac{p(\mathbf{x} \,|\, \mathbf{z}_s, \mathbf{z}_l) p(y \,|\, \mathbf{z}_l) p(\mathbf{z}_s) p(\mathbf{z}_l)}{q(\mathbf{z}_s \,|\, \mathbf{x}) q(\mathbf{z}_l \,|\, \mathbf{x}, \mathbf{z}_s)} \right) \right] \right].
\end{aligned}
$$

Denoting $\Psi = (\theta_1, \theta_2, \phi_1, \phi_2)$, the ELBO can be given by

$$
\begin{aligned}
&\mathcal{L}(\Psi) = -\mathbb{E}_{q_{\phi_2}} \mathbb{E}_{q_{\phi_1}} \left[\ln p_{\theta_2}(\mathbf{x} \,|\, \mathbf{z}_s, \mathbf{z}_l) + \ln p_{\theta_1}(y \,|\, \mathbf{z}_l) \right] + \\
&D_{KL}\big(q_{\phi_1}(\mathbf{z}_l \,|\, \mathbf{x}, \mathbf{z}_s) \| \, p(\mathbf{z}_l)\big) + D_{KL}\big(q_{\phi_2}(\mathbf{z}_s \,|\, \mathbf{x}) \| \, p(\mathbf{z}_s)\big),
\end{aligned} \tag{2}
$$

where the second line follows the graphical model in Fig 2, and $\mathbb{E}(.)$ and $\ln(.)$ denote the expectation operator and the natural logarithm, respectively. We avoid computing biased gradients by following the re-parameterization trick from (Kingma and Welling 2014). Assuming Gaussian distributions for the priors as well as the variational distributions allows us to compute the KL Divergences of $\mathbf{z}_l$ and $\mathbf{z}_s$ (last two terms in (2)) analytically (Kingma and Welling 2014). By considering a multivariate Gaussian distribution and a multinomial distribution as the likelihood functions for $p_{\theta_2}(\mathbf{x} \,|\, \mathbf{z}_s, \mathbf{z}_l)$ and $p_{\theta_1}(y \,|\, \mathbf{z}_l)$, respectively, the negative log-likelihood of $\mathbf{x}$ becomes the mean squared error (MSE) between the reconstructed images $\tilde{\mathbf{x}}$ and the ground-truth images $\mathbf{x}$ while the negative log-likelihood of y becomes the cross-entropy between the actual labels y and the predicted labels $\tilde{y}$. After working (2) out, we arrive at our overall objective function $\mathcal{L} = \mathcal{L}_R + \mathcal{L}_C$, where:

$$
\begin{aligned}
\mathcal{L}_R &= \alpha_1 \|\mathbf{x} - \tilde{\mathbf{x}}\|^2 - KL(\mu_s, \sigma_s), \\
\mathcal{L}_C &= -\alpha_2 \sum_{n=1}^{N} y_n \ln p_{\theta_1}(\tilde{y} = n \,|\, \mathbf{z}_l) - KL(\mu_l, \sigma_l),
\end{aligned} \tag{3}
$$

where $KL(\mu, \sigma) = \frac{1}{2} \sum_{d=1}^{D} \big(1 + 2\ln(\sigma^d) - (\mu^d)^2 - (\sigma^d)^2\big)$, D denotes the dimension of the latent space, N is the total number of classes in an (N-way, K-shot) task, α_1, α_2 are constant scaling factors, μ_s and σ_s^2 denote the mean and variance vectors of semantic latent distribution, and μ_l and σ_l^2 denote the mean and variance vectors of label latent distribution. The hyper-parameters α_1, α_2 only scale the evidence lower-bound appropriately, since the reconstruction loss is in practice three orders of magnitude greater than the cross-entropy loss. As such, this scaling helps convergence but impacts the tightness of the ELBO slightly; nonetheless, (2) and (3) are still considered variational inference by consensus among the literature (Higgins et al. 2017; Joy et al. 2021; Mathieu et al. 2019; Dupont 2018). The loss is calculated for each given task on query and support sets separately; i.e., $\mathcal{L}^g = \mathcal{L}^g_R + \mathcal{L}^g_C$ with $g \in \{\mathcal{S}_i, \mathcal{Q}_i\}$. Note that in (1) we deliberately choose to exclude the label information y as input to $q_{\phi_1}(.)$ to be able to exploit the associated generative network $p_{\theta_1}(y \,|\, \mathbf{z}_l)$ as a classifier. The consequence and the proposed solution to accommodate this design choice are discussed in the next subsection.

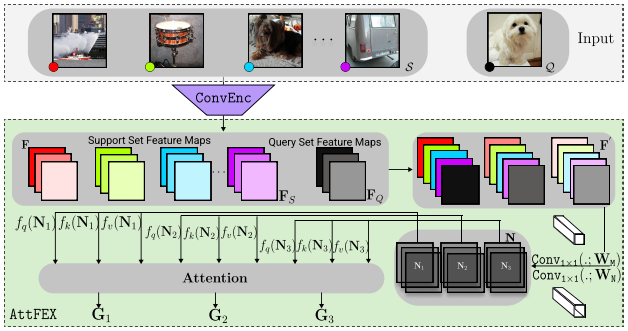


Figure 3: `AttFEX` module depicting colors as images and shades as feature maps. We illustrate only 3 image feature maps and 3 channels instead of 32 for $\mathbf{N}$, for the sake of simplicity.

`AttFEX` for Transductive Feature Extraction

We first extract the feature maps of all images in the task using a convolutional block $\mathbf{F} = \texttt{ConvEnc}(\mathbf{X})$ where $\mathbf{X} \in \mathbb{R}^{N(K+Q) \times C \times W \times H}$, $\mathbf{F} \in \mathbb{R}^{N(K+Q) \times C' \times W' \times H'}$. The feature map tensor $\mathbf{F}$ is then transposed into $\mathbf{F}' \in \mathbb{R}^{C' \times N(K+Q) \times W' \times H'}$ and fed into two consecutive 1×1 convolution blocks. This helps the network utilize information across corresponding pixels of all images in a task $\mathcal{T}_i$ which acts as a parametric comparison of classes. We leverage the fact that `ConvEnc` already extracts local pixel information by using larger kernels, and thus, use parameter-light 1×1 convolutions subsequently to focus only on individual pixels. Let $\mathbf{F}'_i$ denote the i^{th} channel (or feature map layer) out of total of C' available and `ReLU` denote the rectified linear unit activation. The 1×1 convolution block ($\texttt{Conv}_{1\times1}$) is formulated as follows:

$$
\begin{aligned}
\mathbf{M}_i &= \texttt{ReLU}\big(\texttt{Conv}_{1\times1}(\mathbf{F}'_i, \mathbf{W}_M)\big), \forall i \in [1, C']; \\
\mathbf{N}_j &= \texttt{ReLU}\big(\texttt{Conv}_{1\times1}(\mathbf{M}_j, \mathbf{W}_N)\big), \forall j \in [1, C'];
\end{aligned} \tag{4}
$$

where $\mathbf{N} \in \mathbb{R}^{C' \times 32 \times W' \times H'}$ and $\mathbf{W}_M \in \mathbb{R}^{64 \times N(K+Q) \times 1 \times 1}$, $\mathbf{W}_N \in \mathbb{R}^{32 \times 64 \times 1 \times 1}$ denote the learnable weights. Next, we want to blend information across feature maps for which we use a self-attention mechanism (Vaswani et al. 2017) across $\mathbf{N}_j, \forall j \in [1, 32]$. To do so, we feed $\mathbf{N}$ to query, key and value extraction networks $f_q(, ; \mathbf{W}_Q)$, $f_k(.; \mathbf{W}_K)$, $f_v(.; \mathbf{W}_V)$ which are also designed to be 1×1 convolutions as:

$$
\begin{aligned}
\mathbf{Q}_i &= \texttt{ReLU}\left(\texttt{Conv}_{1\times1}(\mathbf{N}_i, \mathbf{W}_Q)\right), \quad \forall i \in [1, C']; \\
\mathbf{K}_i &= \texttt{ReLU}\left(\texttt{Conv}_{1\times1}(\mathbf{N}_i, \mathbf{W}_K)\right), \quad \forall i \in [1, C']; \\
\mathbf{V}_i &= \texttt{ReLU}\left(\texttt{Conv}_{1\times1}(\mathbf{N}_i, \mathbf{W}_V)\right), \quad \forall i \in [1, C'];
\end{aligned} \tag{5}
$$

where $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{1 \times 32 \times 1 \times 1}$ are the learnable weights and $\mathbf{Q}$, $\mathbf{K}$, $\mathbf{V} \in \mathbb{R}^{C' \times 1 \times W' \times H'}$ are the query, key and value tensors. Next, each feature map $\mathbf{N}_j$ is mapped to its output tensor $\mathbf{G}_j$ by computing a weighted sum of the values, where each weight (within parentheses in (6)) measures the compatibility (or similarity) between the query and its corresponding key tensor using an inner-product:

$$
\mathbf{G}_i = \sum_{j=1}^{C'} \left(\frac{\exp\left(\mathbf{Q}_i \cdot \mathbf{K}_j\right)}{\sqrt{d_k} \cdot \sum_{k=1}^{C'} \exp\left(\mathbf{Q}_i \cdot \mathbf{K}_k\right)} \right) \mathbf{V}_i, \tag{6}
$$

where $d_k = W' \times H'$, and $\mathbf{G}_i \in \mathbb{R}^{1\times C'\times W'\times H'}$, $\forall i$. Finally, we transform the original feature maps $\mathbf{F}$ by applying a Hadamard product between the feature mask $\mathbf{G}$ and $\mathbf{F}$, thus, rendering the required feature maps transductive:

$$\tilde{\mathbf{F}}^S = \mathbf{G} \circ \mathbf{F}^S \quad or \quad \tilde{\mathbf{F}}^Q = \mathbf{G} \circ \mathbf{F}^Q.$$

Here, $\mathbf{F}^S$ and $\mathbf{F}^Q$ represent the feature maps corresponding to the support and query images, respectively. As a result of operating on this channel-pixel distribution across images in a task, $\mathbf{F}^S$ and $\mathbf{F}^Q$ are rendered task-aware. Note that the query tensor $\mathbf{Q}$ must not be confused with the query set $\mathcal{Q}$ of a task.

Algorithmic Overview and Training Strategy

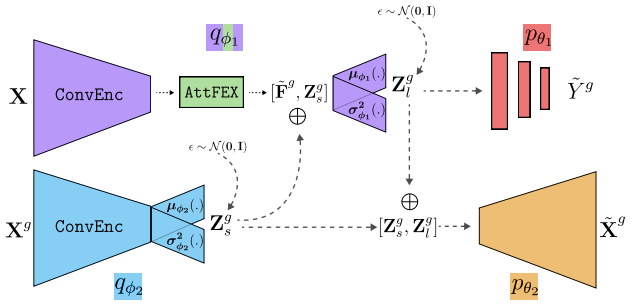


Figure 4: TRIDENT is comprised of two intertwined variational networks. $\mathbf{Z}_s^g$ is concatenated with the output of AttFEX, and used for inferring $\mathbf{Z}_l^g$, where $g \in \{\mathcal{S}, \mathcal{Q}\}$. Next, both $\mathbf{Z}_l^g$ and $\mathbf{Z}_s^g$ are used to reconstruct images $\tilde{\mathbf{X}}^g$ while $\mathbf{Z}_l^g$ is used to extract $\tilde{Y}^g$.

Overview of TRIDENT. The complete architecture of TRIDENT is illustrated in Fig. 4. The ConvEnc feature extractor and the linear layers $\mu_{\phi_2}(.)$, $\sigma^2_{\phi_2}(.)$ constitute the inference network q_{ϕ_2} of the semantic latent variable (bottom row of Fig. 4). The AttFEX module, another ConvEnc, and linear layers $\mu_{\phi_1}(.)$ and $\sigma^2_{\phi_1}(.)$ make up the inference network q_{ϕ_1} of the label latent variable (top row of Fig. 4). The proposed approach, TRIDENT, is described in Algorithm 1. Note that TRIDENT is trained in a MAML (Finn, Abbeel, and Levine 2017) fashion, where depending on the inner or outer loop, the support or query set ($g \in \{\mathcal{S}, \mathcal{Q}\}$) will be the reference, respectively. First, the lower ConvEnc block extracts feature maps $\mathbf{X}^g_{\text{CE}} = \texttt{ConvEnc}(\mathbf{X}^g)$. $\mathbf{X}^g_{\text{CE}}$'s are then flattened and passed onto $\mu_{\phi_2}(.)$, $\sigma^2_{\phi_2}(.)$, which respectively output the mean and variance vectors of the *semantic* latent distribution, as discussed in (1). This is done either for the entire support or the query images $\mathbf{X}^g$, where $g \in \{S, Q\}$ for a given task $\mathcal{T}_i$. We then sample a set of vectors $\mathbf{Z}^g_s$ (subscript s for *semantic*) from their corresponding Gaussian distributions using the re-parameterization trick (line 1, Algorithm 1). Upon passing $\mathbf{X} = \mathbf{X}^{\mathcal{S}} \cup \mathbf{X}^{\mathcal{Q}}$ through the upper ConvEnc, the AttFEX module of q_{ϕ_1} comes into play to create *task-cognizant* feature maps $\tilde{\mathbf{F}}^g$ for either $\mathcal{S}$ or $\mathcal{Q}$ (line 2). $\mathbf{Z}^g_s$ together with $\tilde{\mathbf{F}}^g$ are passed onto the linear layers $\mu_{\phi_1}(.)$, $\sigma^2_{\phi_1}(.)$ to generate the mean and variance vectors of the *label* latent Gaussian distributions (line 3). After sampling the set of vectors $\mathbf{Z}^g_l$ (subscript l for *label*) from their corresponding distributions, we use $\mathbf{Z}^g_l$ and $\mathbf{Z}^g_s$ to reconstruct the input images $\tilde{\mathbf{X}}^g$ using the generative network p_{θ_2} (line 4). Next, $\mathbf{Z}^g_l$'s are input to the classifier network p_{θ_1} to generate the class logits, which are normalized using a softmax(.), resulting in class-conditional probabilities $p(\tilde{Y}^g \,|\, \mathbf{Z}^g_l)$ (line 5). Finally (in line 6), using the outputs of all the components discussed earlier, we calculate the loss $\mathcal{L}^g$ as formulated in (2) and (3). **Training strategy.** An important aspect of the training procedure of TRIDENT is that its set of parameters $\Psi = (\theta_1, \theta_2, \phi_1, \phi_2)$ are meta-learnt by back-propagating through the adaptation procedure on the support set, as proposed in MAML (Finn, Abbeel, and Levine 2017) and illustrated here in Algorithm 2. This increases the sensitivity of the parameters Ψ towards the loss function for fast adaptation to unseen tasks and reduces generalization errors on the query set $\mathcal{Q}$. First, we randomly initialize the parameters Ψ (line 1, Algorithm 2) to compute the objective function over the support set $\mathcal{L}^{\mathcal{S}_i}(\Psi)$ using equation (3) in the main manuscript, and perform a number of gradient de-

Algorithm 1: TRIDENT

Require: $\mathbf{X}^{\mathcal{S}}, \mathbf{X}^{\mathcal{Q}}, Y^g, \mathbf{X}^g_{\text{CE}}$, where $g \in \{\mathcal{S}, \mathcal{Q}\}$

1 Sample: $\mathbf{Z}^g_s \sim q_{\phi_2}(\mathbf{Z}_s \,|\, \boldsymbol{\mu}_{\phi_2}(\mathbf{X}^g_{\text{CE}}), \, diag(\boldsymbol{\sigma}^2_{\phi_2}(\mathbf{X}^g_{\text{CE}})))$

2 Compute *task-cognizant* embeddings:
$[\tilde{\mathbf{F}}^{\mathcal{S}}, \tilde{\mathbf{F}}^{\mathcal{Q}}] = \texttt{AttFEX}(\texttt{ConvEnc}(\mathbf{X})); \mathbf{X} = \mathbf{X}^{\mathcal{S}} \cup \mathbf{X}^{\mathcal{Q}}$

3 Concatenate $\mathbf{Z}^g_s$ and $\tilde{\mathbf{F}}^g$ into $[\tilde{\mathbf{F}}^g, \mathbf{Z}^g_s]$ and sample:
$\mathbf{Z}^g_l \sim q_{\phi_1}(\mathbf{Z}_l \,|\, \boldsymbol{\mu}_{\phi_1}([\tilde{\mathbf{F}}^g, \mathbf{Z}^g_s]), \, diag(\boldsymbol{\sigma}^2_{\phi_1}([\tilde{\mathbf{F}}^g, \mathbf{Z}^g_s])))$

4 Reconstruct $\mathbf{X}^g$ using $\tilde{\mathbf{X}}^g = p_{\theta_2}(\mathbf{X} \,|\, \mathbf{Z}^g_l, \mathbf{Z}^g_s)$

5 Extract class-conditional probabilities using:
$p(\tilde{Y}^g \,|\, \mathbf{Z}^g_l) = \texttt{softmax}(p_{\theta_1}(Y^g \,|\, \mathbf{Z}^g_l))$

6 Compute $\mathcal{L}^g = \mathcal{L}^g_R + \mathcal{L}^g_C$ using (3)

Return: $\mathcal{L}^g$

Algorithm 2: End to End Meta-Training of TRIDENT

Require: $\mathcal{D}^{tr}$, α, β, B

1 Randomly initialise $\Psi = (\phi_1, \phi_2, \theta_1, \theta_2)$

2 **while** *not converged* **do**

3 Sample B tasks $\mathcal{T}_i = \mathcal{S}_i \cup \mathcal{Q}_i$ from $\mathcal{D}^{tr}$

4 **for** *each task* $\mathcal{T}_i$ **do**

5 **for** *number of adaptation steps* **do**

6 Compute $\mathcal{L}^{\mathcal{S}_i}(\Psi) = \texttt{TRIDENT}(\mathcal{T}_i - \{Y^{\mathcal{Q}_i}\})$

7 Evaluate $\nabla_{(\Psi)} \mathcal{L}^{\mathcal{S}_i}(\Psi)$

8 $\Psi \leftarrow \Psi - \alpha \nabla_{\Psi} \mathcal{L}^{\mathcal{S}_i}(\Psi)$

9 **end**

10 $(\Psi')_i = \Psi$

11 **end**

12 Compute
$\mathcal{L}^{\mathcal{Q}_i}(\Psi'_i) = \texttt{TRIDENT}(\mathcal{T}_i - \{Y^{\mathcal{S}_i}\}); \forall i \in [1, B]$

13 Meta-update on $\mathcal{Q}_i$: $\Psi \leftarrow \Psi - \beta \nabla_{\Psi} \sum_{i=1}^{B} \mathcal{L}^{\mathcal{Q}_i}(\Psi'_i)$

14 **end**

Table 1: Accuracies in (% ± std). The predominant methodology of the baselines: `Ind.`: inductive inference, `TF`: transductive feature extraction methods, `TI`: transductive inference methods. `Conv`: convolutional blocks, `RN`: `ResNet` backbone, †: extra data. Style: **best** and second best. `TRIDENT` employs a transductive feature extraction module (`TF`), and the simplest of backbones (`Conv4`).

			*mini*Imagenet		*tiered*Imagenet		*mini*→CUB	
Methods	**Backbone**	**Approach**	**5-way 1-shot**	**5-way 5-shot**	**5-way 1-shot**	**5-way 5-shot**	**5-way 1-shot**	**5-way 5-shot**
MAML (Finn, Abbeel, and Levine 2017)	Conv4	Ind.	48.70 ± 1.84	63.11 ± 0.92	51.67 ± 1.81	70.30 ± 0.08	34.01 ± 1.25	48.83 ± 0.62
ABML (Ravi and Beatson 2019)	Conv4	Ind.	40.88 ± 0.25	58.19 ± 0.17	-	-	31.51 ± 0.32	47.80 ± 0.51
OVE(PL) (Patacchiola et al. 2020)	Conv4	Ind.	48.00 ± 0.24	67.14 ± 0.23	-	-	37.49 ± 0.11	57.23 ± 0.31
DKT+Cos (Patacchiola et al. 2020)	Conv4	Ind.	48.64 ± 0.45	62.85 ± 0.37	-	-	40.22 ± 0.54	55.65 ± 0.05
BOIL (Oh et al. 2021)	Conv4	Ind.	49.61 ± 0.16	48.58 ± 0.27	66.45 ± 0.37	69.37 ± 0.12	-	-
LFWT(Tseng et al. 2020)	RN10	TF+TI	66.32 ± 0.80	81.98 ± 0.55	-	-	47.47 ± 0.75	66.98 ± 0.68
FRN(Wertheimer, Tang, and Hariharan 2021)	RN12	Ind.	66.45 ± 0.19	82.83 ± 0.13	71.16 ± 0.22	86.01 ± 0.15	54.11 ± 0.19	77.09 ± 0.15
DPGN(Yang et al. 2020)	RN12	TF+TI	67.77	84.6	72.45	87.24	-	-
PAL(Ma et al. 2021)	RN12	TF+TI	69.37 ± 0.64	84.40 ± 0.44	72.25 ± 0.72	86.95 ± 0.47	-	-
Proto-Completion(Zhang et al. 2021a)	RN12	TF+TI	73.13 ± 0.85	82.06 ± 0.54	81.04 ± 0.89	87.42 ± 0.57	-	-
TPMN(Wu et al. 2021)	RN12	TF+TI	67.64 ± 0.63	83.44 ± 0.43	72.24 ± 0.70	86.55 ± 0.63	-	-
LIF-EMD(Li, Wang, and Hu 2021)	RN12	TF+TI	68.94 ± 0.28	85.07 ± 0.50	73.76 ± 0.32	87.83 ± 0.59	-	-
Transd-CNAPS(Bateni et al. 2022)	RN18	TF+TI	55.6 ± 0.9	73.1 ± 0.7	65.9 ± 1.0	81.8 ± 0.7	-	-
Baseline++(Chen et al. 2019)	RN18	TF	51.87 ± 0.77	75.68 ± 0.63	-	-	42.85 ± 0.69	62.04 ± 0.76
FEAT(Ye et al. 2020)	RN18	TF	66.78	82.05	70.80	84.79	50.67 ± 0.78	71.08 ± 0.73
SimpleShot(Wang et al. 2019)	WRN	Ind.	63.32	80.28	69.98	85.45	48.56	65.63
Assoc-Align(Afrasiyabi, Lalonde, and Gagné 2020)	WRN	TF	65.92 ± 0.60	82.85 ± 0.55	74.40 ± 0.68	86.61 ± 0.59	47.25 ± 0.76	72.37 ± 0.89
ReRank(SHEN et al. 2021)	WRN	TF+TI	72.4±0.6	80.2±0.4	79.5±0.6	84.8±0.4	-	-
TIM-GD(Boudiaf et al. 2020)	WRN	TI	77.8	87.4	82.1	89.8	-	71
LaplacianShot(Ziko et al. 2020)	WRN	TI	74.9	84.07	80.22	87.49	55.46	66.33
S2M2(Mangla et al. 2020)	WRN	TF	64.93 ± 0.18	83.18 ± 0.11	73.71 ± 0.22	88.59 ± 0.14	48.24 ± 0.84	70.44 ± 0.75
MetaQDA(Zhang et al. 2021b)	WRN	TF	67.83 ± 0.64	84.28 ± 0.69	74.33 ± 0.65	89.56 ± 0.79	53.75 ± 0.72	71.84 ± 0.66
PT+MAP(Hu, Gripon, and Pateux 2021)	WRN	TF+TI	82.92 ± 0.26	88.82 ± 0.13	85.67 ± 0.26	90.45 ± 0.14	62.49 ± 0.32	76.51 ± 0.18
PEM_nE-BMS(Hu, Pateux, and Gripon 2022)	WRN	TF+TI	83.35 ± 0.25	89.53 ± 0.13	86.07 ± 0.25	91.09 ± 0.14	63.90 ± 0.31	79.15 ± 0.18
Transd-CNAPS+FETI(Bateni et al. 2022)	RN18†	TF+TI	79.9 ± 0.8	91.50 ± 0.4	73.8 ± 0.1	87.7 ± 0.6	-	-
TRIDENT(Ours)	Conv4	TF	**86.11 ± 0.59**	**95.95 ± 0.28**	**86.97 ± 0.50**	**96.57 ± 0.17**	**84.61 ± 0.33**	**80.74 ± 0.35**

scent steps on the parameters Ψ to adapt them to the support set (lines 5 to 9). This is called the *inner-update* and is done separately for all the support sets corresponding to their B different tasks (line 3). Once the inner-update is computed for each of the B parameter sets, the loss is evaluated on the query set $\mathcal{L}^{\mathcal{Q}_i}(\Psi_i')$ (line 12), following which a *meta-update* is conducted over all the corresponding query sets, which involves computing a gradient through a gradient procedure as described in (Finn, Abbeel, and Levine 2017) (line 13).

Experimental Evaluation

The goal of this section is to address the following four questions: (i) How well does `TRIDENT` perform when compared against the state-of-the-art methods for few-shot classification? (ii) How reliable is `TRIDENT` in terms of the confidence and uncertainty metrics? (iii) How well does `TRIDENT` perform in a cross-domain setting where there is a domain shift between the training and testing datasets? (iv) Does `TRIDENT` actually decouple latent variables?

Benchmark Datasets. We evaluate `TRIDENT` on the three most commonly adopted datasets: *mini*Imagenet (Ravi and Larochelle 2017), *tiered*Imagenet (Ren et al. 2018) and CUB (Welinder et al. 2010). *mini*Imagenet and *tiered*Imagenet are subsets of ImageNet (Deng et al. 2009) utilized for few-shot classification. Further details on these datasets can be found in the Appendix.

Implementational Details. We use PyTorch (Paszke et al. 2019) and learn2learn (Arnold et al. 2020) for all our implementations. We use a commonly adopted `Conv4` architecture (Ravi and Larochelle 2017; Finn, Abbeel, and Levine 2017; Patacchiola et al. 2020; Afrasiyabi, Lalonde, and Gagné 2020; Wang et al. 2019; Boudiaf et al. 2020) as `ConvEnc` to obtain the generic feature maps. Following the standard setting in the literature (Finn, Abbeel, and Levine 2017; Ravi and Larochelle 2017), the `Conv4` has four convolutional blocks where each block has a 3×3 convolution layer with 32 feature maps, followed by a batch normalization (BN) (Ioffe and Szegedy 2015) layer, a 2×2 max-pooling layer and a `LeakyReLU(0.2)` activation. The generative network p_{θ_1} for $\mathbf{z}_l$ is a classifier with two linear layers and a `LeakyReLU(0.2)` activation in between, while p_{θ_2} for $\mathbf{z}_s$ consists of four blocks of a 2-D upsampling layer, followed by a 3×3 convolution and `LeakyReLU(0.2)` activation. Both latent variables $\mathbf{z}_l$ and $\mathbf{z}_s$ have a dimensionality of 64. Following (Nichol, Achiam, and Schulman 2018b; Liu et al. 2019; Vaswani et al. 2017), images are resized to 84×84 for all configurations and we train and report test accuracy of (5-way, 1 and 5-shot) settings with 10 query images per class for all datasets. Hyperparameter settings can be found in the Appendix.

Evaluation Results

We report test accuracies indicating 95% confidence intervals over 600 tasks for *mini*Imagenet, and 2000 tasks for both *tiered*Imagenet and CUB, as is customary across the literature (Chen et al. 2019; Dhillon et al. 2020; Bateni et al. 2022). We compare our performance against a wide variety of state-of-the-art few-shot classification methods such as: (i) metric-learning (Wang et al. 2019; Bateni et al. 2020; Afrasiyabi, Lalonde, and Gagné 2020; Yang et al. 2020), (ii) transductive feature-extraction based (Oreshkin, Rodríguez López, and Lacoste 2018; Ye et al. 2020; Li et al. 2019; Xu et al. 2021), (iii) optimization-based (Finn, Abbeel, and Levine 2017; Mishra et al. 2018; Oh et al. 2021; Lee et al. 2019; Rusu et al. 2019), (iv) transductive inference-based (Bateni et al. 2022; Boudiaf et al. 2020;

Table 2: Parameter count of TRIDENT against competitors.

	Conv4	μ_ϕ	σ_ϕ	AttFEX	**TRIDENT**	Conv4	RN18	WRN
q_{ϕ_1}	28896	51264	51264	6994	**412,238**	190, 410	12.4M	36.482M
q_{ϕ_2}	28896	51264	51264	-				
$p_{\theta_1} + p_{\theta_2}$	2245 + 132009							

Ziko et al. 2020; Liu et al. 2019), and (v) Bayesian (Iakovleva, Verbeek, and Alahari 2020; Zhang et al. 2019; Hu et al. 2020; Patacchiola et al. 2020; Ravi and Beatson 2019) approaches. Previous works (Liu et al. 2019), (Hou et al. 2019) have demonstrated the superiority of transductive inference methods over their inductive counterparts. In this light, we compare against a larger number of transductive (18 baselines) rather than inductive (7 baselines) methods for a fair comparison.

It is important to note that TRIDENT is only a *transductive feature-extraction* based method as we utilize the query set images to extract task-aware feature embeddings; it is not a transductive inference based method since we perform inference of class-labels over the entire domain of definition and not just for the selected query samples (Vapnik 2006; Gammerman, Vovk, and Vapnik 1998). The results on *mini*Imagenet and *tiered*Imagenet for both (5-way, 1 and 5-shot) settings are summarized in Table 1. We accentuate on the fact that we also compare against Transd-CNAPS+FETI (Bateni et al. 2022), where the authors pretrain the ResNet-18 backbone on the entire train split of Imagenet. We, however, avoid training on additional datasets, in favor of fair comparison with the rest of literature. Regardless of the choice of backbone (simplest in our case), TRIDENT sets a new state-of-the-art on *mini*Imagenet and *tiered*Imagenet for both (5-way, 1 and 5-shot) settings, offering up to 5% gain over the prior art. Recently, a more challenging *cross-domain* setting has been proposed for few-shot classification to assess its generalization capabilities to unseen datasets. The commonly adopted setting is where one trains on *mini*Imagenet and tests on CUB (Chen et al. 2019). The results of this experiment are also presented in Table 1. We compare against *any existing baselines* for which this cross-domain experiment has been conducted. As can be seen, and to the best of our knowledge, TRIDENT again sets a new state-of-the-art by a significant margin of 20% for (5-way, 1-shot) setting, and 1.5% for (5-way, 5-shot) setting.

Computational Complexity. Most of the reported baselines in Table 1 use stronger backbones such as ResNet12, ResNet18 and WRN which contain 11.5, 12.4 and 36.4 millions of parameters respectively. On the other hand, we use three Conv4s along with two fully connected layers and an AttFEX module which accounts for 410,958 and 412,238 parameters in the (5-way, 1-shot) and (5-way, 5-shot) scenarios, respectively. This is summarized in details in Table 2. Even though we are more parameter heavy than approaches that use a single Conv4 as feature extractor, TRIDENT's total parameters still lies in the same order of magnitude as these approaches. In summary, when it comes to complexity in parameter space, we are considerably more efficient than the vast majority of the cited competitors.

Reliability Metrics. A complementary set of metrics are typically used in probabilistic settings to measure the uncertainty and reliability of predictions. More specifically, expected calibration error (ECE) and maximum calibration error (MCE) respectively measure the expected and maximum binned difference between confidence and accuracy (Guo et al. 2017). This is illustrated in Table 3 where TRIDENT offers superior calibration on *mini*Imagenet (5-way, 1 and 5-shot) as compared to other probabilistic approaches, and MAML (Finn, Abbeel, and Levine 2017).

Table 3: Calibration errors of TRIDENT. Style: **best** and second best.

	Metrics	MAML	PLATIPUS	ABPML	ABML	BMAML	VAMPIRE	**TRIDENT**
5-way, 1-shot	**ECE**	0.046	0.032	0.013	0.026	0.025	0.008	**0.0036**
	MCE	0.073	0.108	0.037	0.058	0.092	0.038	**0.029**
5-way, 5-shot	**ECE**	0.032	-	0.006	-	0.027	-	**0.0015**
	MCE	0.044	-	0.030	-	0.049	-	**0.018**

Ablation Study

We analyze the classification performance of TRIDENT across various paramaters and hyper-parameters, as is summarized in Table 4. We use *mini*Imagenet (5-way, 1-shot) setting to carry out ablation study experiments. To cover different design perspectives, we carry out ablation on: (i) MAML-style training parameters: meta-batch size B and number of inner adaption steps n, (ii) latent space dimensionality: $\mathbf{z}_l$ and $\mathbf{z}_s$ to assess the impact of their size, (iii) AttFEX features: number of features extracted by $\mathbf{W}_M$, $\mathbf{W}_N$. Looking at the results, TRIDENT's performance is directly proportional to the number of tasks and inner-adaptation steps, as is previously demonstrated in (Antreas Antoniou, Harrison Edwards, and Amos J. Storkey 2019; Finn, Abbeel, and Levine 2017) for MAML based training. Regarding latent space dimensions, a correlation between a higher dimension of $\mathbf{z}_l$ and $\mathbf{z}_s$ and a better performance can be observed. Even though, the results show that increasing both dimensions beyond 64 leads to performance degradation. As such, $(64, 64)$ seems to be the sweet spot. Finally, on feature space dimensions of AttFEX, the performance improves when $\mathbf{W}_M > \mathbf{W}_N$, and the best performance is achieved when the parameters are set to (64, 32). Notably, the exact set of parameters return the best performance for (5-way, 5-shot) setting.

Decoupling Analysis

As a qualitative demonstration, we visualize the *label* and *semantic* latent means (μ_l and μ_s) of query images for a randomly selected (5-way, 5-shot) task from *mini*Imagenet, before and after the MAML meta-update procedure. The UMAP (McInnes, Healy, and Melville 2018) plots in Fig. 5 illustrate significant improvement in class-conditional separation of query samples for *label* latent space upon meta-adaption, whereas negligible improvement is visible on the semantic latent space. This is a qualitative evidence that $\mathbf{Z}_L$ captures more class-discriminating information as compared to $\mathbf{Z}_S$. To substantiate this quantitatively, the clustering capacity of these latent spaces is also measured by the Davies-Bouldin score (DBI) (Davies and Bouldin 1979), where, the lower the DBI score, the better both the inter-cluster separation and intra-cluster "tightness". Fig. 5 shows that the DBI score drops significantly more after meta-adaptation in the case of $\mathbf{Z}_L$ as compared to $\mathbf{Z}_S$, indicating better clustering of features in the former than the latter. This aligns with the proposed decoupling strategy of TRIDENT and corroborates the validity of our proposition to put an emphasis on

Table 4: Ablation study for *mini*Imagenet (5-way, 1-shot) tasks. Accuracies in (% ± std.).

(B, n)	**(5, 3)**	**(5, 5)**		**(10, 3)**	**(10, 5)**		**(20, 3)**	**(20, 5)**	
	-	67.43 ± 0.75		69.21 ± 0.66	74.6 ± 0.84		80.82 ± 0.68	**86.11 ± 0.59**	
$(dim(\mathbf{z}_l), dim(\mathbf{z}_s))$	**(32, 32)**	**(32, 64)**	**(32, 128)**	**(64, 32)**	**(64, 64)**	**(64, 128)**	**(128, 32)**	**(128, 64)**	**(128, 128)**
	76.29 ± 0.72	75.44 ± 0.81	79.1 ± 0.57	82.93 ± 0.8	**86.11 ± 0.59**	85.62 ± 0.52	81.49 ± 0.65	82.89 ± 0.48	84.42 ± 0.59
$(dim(\mathbf{W}_M), dim(\mathbf{W}_N))$	**(32, 32)**	**(32, 64)**	**(32, 128)**	**(64, 32)**	**(64, 64)**	**(64, 128)**	**(128, 32)**	**(128, 64)**	**(128, 128)**
	78.4 ± 0.23	77.89 ± 0.39	79.55 ± 0.87	**86.11 ± 0.59**	84.87 ± 0.45	82.11 ± 0.35	84.67 ± 0.7	85.8 ± 0.58	83.92 ± 0.63

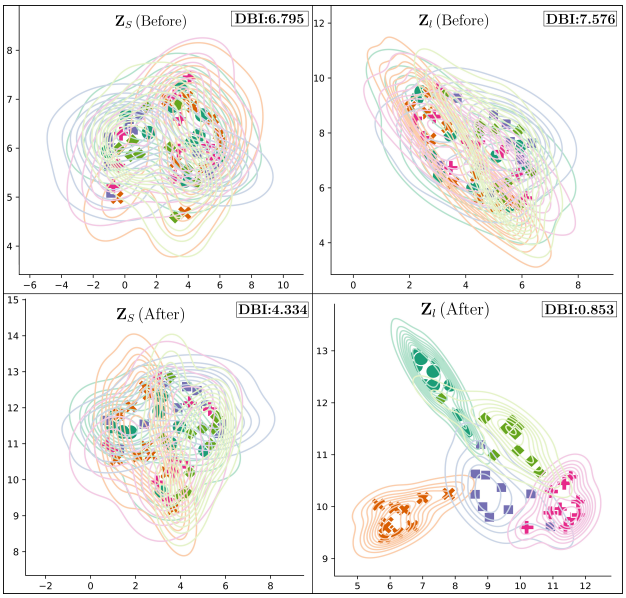


Figure 5: Better class separation upon meta-update is confirmed by lower DBI scores. Different colors/markers indicated classes.

label latent information for the downstream few-shot task.

Concluding Remarks

We introduce a novel variational inference network (coined as TRIDENT) that simultaneously infers decoupled latent variables representing semantic and label information of an image. The proposed network is comprised of two intertwined variational sub-networks responsible for inferring the semantic and label information separately, the latter being enhanced using an attention-based transductive feature extraction module (AttFEX). Our extensive experimental results corroborate the efficacy of this transductive decoupling strategy on a variety of few-shot classification settings demonstrating superior performance and setting a new state-of-the-art for the most commonly adopted dataset *mini* and *tiered*Imagenet as well as for the recent challenging cross-domain scenario of *mini*Imagenet → CUB. As future work, we plan to demonstrate the applicability of TRIDENT in semi-supervised and unsupervised settings by including the likelihood of unlabelled samples derived from the graphical model. This would render TRIDENT as an all-inclusive holistic approach towards solving few-shot classification.

References

Afrasiyabi, A.; Lalonde, J.-F.; and Gagné, C. 2020. Associative alignment for few-shot image classification. In *European Conference on Computer Vision*, 18–35. Springer.

Antreas Antoniou; Harrison Edwards; and Amos J. Storkey. 2019. How to train your MAML. In *ICLR (Poster)*. OpenReview.net.

Arnold, S. M. R.; Mahajan, P.; Datta, D.; Bunner, I.; and Zarkias, K. S. 2020. learn2learn: A Library for Meta-Learning Research.

Bateni, P.; Barber, J.; van de Meent, J.-W.; and Wood, F. 2022. Enhancing Few-Shot Image Classification With Unlabelled Examples. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2796–2805.

Bateni, P.; Goyal, R.; Masrani, V.; Wood, F.; and Sigal, L. 2020. Improved Few-Shot Visual Classification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Boudiaf, M.; Ziko, I.; Rony, J.; Dolz, J.; Piantanida, P.; and Ben Ayed, I. 2020. Information Maximization for Few-Shot Learning. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M. F.; and Lin, H., eds., *Advances in Neural Information Processing Systems*, volume 33, 2445–2457. Curran Associates, Inc.

BRIER, G. W. 1950. VERIFICATION OF FORECASTS EXPRESSED IN TERMS OF PROBABILITY. *Monthly Weather Review*, 78(1): 1 – 3.

Chen, W.-Y.; Liu, Y.-C.; Kira, Z.; Wang, Y.-C.; and Huang, J.-B. 2019. A Closer Look at Few-shot Classification. In *International Conference on Learning Representations*.

Davies, D. L.; and Bouldin, D. W. 1979. A Cluster Separation Measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1(2): 224–227.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, 248–255. Ieee.

Dhillon, G. S.; Chaudhari, P.; Ravichandran, A.; and Soatto, S. 2020. A Baseline for Few-Shot Image Classification. In *International Conference on Learning Representations*.

Dupont, E. 2018. Learning Disentangled Joint Continuous and Discrete Representations. In Bengio, S.; Wallach, H.; Larochelle, H.; Grauman, K.; Cesa-Bianchi, N.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Edwards, H.; and Storkey, A. J. 2017. Towards a Neural Statistician. *ArXiv*, abs/1606.02185.

Finn, C.; Abbeel, P.; and Levine, S. 2017. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In

Proceedings of the 34th International Conference on Machine Learning, volume 70 of *Proceedings of Machine Learning Research*, 1126–1135.

Finn, C.; Xu, K.; and Levine, S. 2018. Probabilistic Model-Agnostic Meta-Learning. In Bengio, S.; Wallach, H.; Larochelle, H.; Grauman, K.; Cesa-Bianchi, N.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Galy-Fajou, T.; Wenzel, F.; Donner, C.; and Opper, M. 2020. Multi-Class Gaussian Process Classification Made Conjugate: Efficient Inference via Data Augmentation. In Adams, R. P.; and Gogate, V., eds., *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*, volume 115 of *Proceedings of Machine Learning Research*, 755–765. PMLR.

Gammerman, A.; Vovk, V.; and Vapnik, V. 1998. Learning by transduction, vol UAI'98.

Gordon, J.; Bronskill, J.; Bauer, M.; Nowozin, S.; and Turner, R. 2019. Meta-Learning Probabilistic Inference for Prediction. In *International Conference on Learning Representations*.

Guo, C.; Pleiss, G.; Sun, Y.; and Weinberger, K. Q. 2017. On Calibration of Modern Neural Networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, 1321–1330. PMLR.

Higgins, I.; Matthey, L.; Pal, A.; Burgess, C.; Glorot, X.; Botvinick, M.; Mohamed, S.; and Lerchner, A. 2017. beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework. In *International Conference on Learning Representations*.

Hou, R.; Chang, H.; MA, B.; Shan, S.; and Chen, X. 2019. Cross Attention Network for Few-shot Classification. In Wallach, H.; Larochelle, H.; Beygelzimer, A.; d'Alché-Buc, F.; Fox, E.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Hu, S. X.; Moreno, P.; Xiao, Y.; Shen, X.; Obozinski, G.; Lawrence, N.; and Damianou, A. 2020. Empirical Bayes Transductive Meta-Learning with Synthetic Gradients. In *International Conference on Learning Representations (ICLR)*.

Hu, Y.; Gripon, V.; and Pateux, S. 2021. Leveraging the feature distribution in transfer-based few-shot learning. In *International Conference on Artificial Neural Networks*, 487–499. Springer.

Hu, Y.; Pateux, S.; and Gripon, V. 2022. Squeezing backbone feature distributions to the max for efficient few-shot learning. *Algorithms*, 15(5): 147.

Iakovleva, E.; Verbeek, J.; and Alahari, K. 2020. Meta-Learning with Shared Amortized Variational Inference. In *Proceedings of the 37th International Conference on Machine Learning*, Proceedings of Machine Learning Research. PMLR.

Ioffe, S.; and Szegedy, C. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, 448–456. PMLR.

Joy, T.; Schmon, S.; Torr, P.; N, S.; and Rainforth, T. 2021. Capturing Label Characteristics in {VAE}s. In *International Conference on Learning Representations*.

Kingma, D. P.; and Ba, J. 2015. Adam: A Method for Stochastic Optimization. In *ICLR (Poster)*.

Kingma, D. P.; Mohamed, S.; Jimenez Rezende, D.; and Welling, M. 2014. Semi-supervised Learning with Deep Generative Models. In *Advances in Neural Information Processing Systems*, volume 27.

Kingma, D. P.; and Welling, M. 2014. Auto-Encoding Variational Bayes. arXiv:1312.6114.

Lee, K.; Maji, S.; Ravichandran, A.; and Soatto, S. 2019. Meta-Learning with Differentiable Convex Optimization. In *CVPR*.

Li, H.; Eigen, D.; Dodge, S.; Zeiler, M.; and Wang, X. 2019. Finding Task-Relevant Features for Few-Shot Learning by Category Traversal. In *CVPR*.

Li, J.; Wang, Z.; and Hu, X. 2021. Learning Intact Features by Erasing-Inpainting for Few-shot Classification. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(9): 8401–8409.

Liu, J.; Song, L.; and Qin, Y. 2020. Prototype Rectification for Few-Shot Learning. In *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part I*.

Liu, Y.; Lee, J.; Park, M.; Kim, S.; Yang, E.; Hwang, S.; and Yang, Y. 2019. Learning to Propagate Labels: Transductive Propagation Network for Few-shot Learning. In *International Conference on Learning Representations*.

Long, M.; CAO, Z.; Wang, J.; and Jordan, M. I. 2018. Conditional Adversarial Domain Adaptation. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Ma, J.; Xie, H.; Han, G.; Chang, S.-F.; Galstyan, A.; and Abd-Almageed, W. 2021. Partner-assisted learning for few-shot image classification. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 10573–10582.

Mangla, P.; Kumari, N.; Sinha, A.; Singh, M.; Krishnamurthy, B.; and Balasubramanian, V. N. 2020. Charting the right manifold: Manifold mixup for few-shot learning. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, 2218–2227.

Mathieu, E.; Rainforth, T.; Siddharth, N.; and Teh, Y. W. 2019. Disentangling Disentanglement in Variational Autoencoders. In Chaudhuri, K.; and Salakhutdinov, R., eds., *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, 4402–4412. PMLR.

McInnes, L.; Healy, J.; and Melville, J. 2018. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*.

Mishra, N.; Rohaninejad, M.; Chen, X.; and Abbeel, P. 2018. A Simple Neural Attentive Meta-Learner. In *International Conference on Learning Representations*.

Nguyen, C. C.; Do, T.; and Carneiro, G. 2019. Uncertainty in Model-Agnostic Meta-Learning using Variational Inference. *CoRR*.

Nichol, A.; Achiam, J.; and Schulman, J. 2018a. On First-Order Meta-Learning Algorithms. arXiv:1803.02999.

Nichol, A.; Achiam, J.; and Schulman, J. 2018b. On First-Order Meta-Learning Algorithms. *CoRR*, abs/1803.02999.

Oh, J.; Yoo, H.; Kim, C.; and Yun, S.-Y. 2021. BOIL: Towards Representation Change for Few-shot Learning. In *International Conference on Learning Representations*.

Oreshkin, B.; Rodríguez López, P.; and Lacoste, A. 2018. TADAM: Task dependent adaptive metric for improved few-shot learning. In *Advances in Neural Information Processing Systems*, volume 31.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Kopf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; and Chintala, S. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*.

Patacchiola, M.; Turner, J.; Crowley, E. J.; O'Boyle, M. F. P.; and Storkey, A. J. 2020. Bayesian Meta-Learning for the Few-Shot Setting via Deep Kernels. In *NeurIPS*.

Rajeswaran, A.; Finn, C.; Kakade, S. M.; and Levine, S. 2019. Meta-Learning with Implicit Gradients. In Wallach, H.; Larochelle, H.; Beygelzimer, A.; d'Alché-Buc, F.; Fox, E.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Ravi, S.; and Beatson, A. 2019. Amortized Bayesian Meta-Learning. In *International Conference on Learning Representations*.

Ravi, S.; and Larochelle, H. 2017. Optimization as a Model for Few-Shot Learning. In *ICLR*.

Ren, M.; Ravi, S.; Triantafillou, E.; Snell, J.; Swersky, K.; Tenenbaum, J. B.; Larochelle, H.; and Zemel, R. S. 2018. Meta-Learning for Semi-Supervised Few-Shot Classification. In *International Conference on Learning Representations*.

Requeima, J.; Gordon, J.; Bronskill, J.; Nowozin, S.; and Turner, R. E. 2019. Fast and Flexible Multi-Task Classification using Conditional Neural Adaptive Processes. In *Advances in Neural Information Processing Systems*, volume 32.

Rusu, A. A.; Rao, D.; Sygnowski, J.; Vinyals, O.; Pascanu, R.; Osindero, S.; and Hadsell, R. 2019. Meta-Learning with Latent Embedding Optimization. In *International Conference on Learning Representations*.

Satorras, V. G.; and Estrach, J. B. 2018. Few-Shot Learning with Graph Neural Networks. In *International Conference on Learning Representations*.

SHEN, X.; Xiao, Y.; Hu, S. X.; Sbai, O.; and Aubry, M. 2021. Re-ranking for image retrieval and transductive few-shot classification. In Beygelzimer, A.; Dauphin, Y.; Liang, P.; and Vaughan, J. W., eds., *Advances in Neural Information Processing Systems*.

Snell, J.; Swersky, K.; and Zemel, R. 2017. Prototypical Networks for Few-shot Learning. In *Advances in Neural Information Processing Systems*, volume 30.

Snell, J.; and Zemel, R. 2021. Bayesian Few-Shot Classification with One-vs-Each Pólya-Gamma Augmented Gaussian Processes. In *International Conference on Learning Representations*.

Sun, Z.; Wu, J.; Li, X.; Yang, W.; and Xue, J.-H. 2021. Amortized Bayesian Prototype Meta-learning: A New Probabilistic Meta-learning Approach to Few-shot Image Classification. In *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, Proceedings of Machine Learning Research.

Sung, F.; Yang, Y.; Zhang, L.; Xiang, T.; Torr, P. H.; and Hospedales, T. M. 2018. Learning to Compare: Relation Network for Few-Shot Learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Tseng, H.-Y.; Lee, H.-Y.; Huang, J.-B.; and Yang, M.-H. 2020. Cross-domain few-shot classification via learned feature-wise transformation. *arXiv preprint arXiv:2001.08735*.

Vapnik, V. N. 2006. Estimation of Dependences Based on Empirical Data. *Estimation of Dependences Based on Empirical Data*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L. u.; and Polosukhin, I. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30.

Vinyals, O.; Blundell, C.; Lillicrap, T.; kavukcuoglu, k.; and Wierstra, D. 2016. Matching Networks for One Shot Learning. In *Advances in Neural Information Processing Systems*, volume 29.

Wang, Y.; Chao, W.-L.; Weinberger, K. Q.; and van der Maaten, L. 2019. SimpleShot: Revisiting Nearest-Neighbor Classification for Few-Shot Learning. arXiv:1911.04623.

Welinder, P.; Branson, S.; Mita, T.; Wah, C.; Schroff, F.; Belongie, S.; and Perona, P. 2010. Caltech-UCSD Birds 200. Technical Report CNS-TR-2010-001, California Institute of Technology.

Wertheimer, D.; Tang, L.; and Hariharan, B. 2021. Few-Shot Classification With Feature Map Reconstruction Networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 8012–8021.

Wu, J.; Zhang, T.; Zhang, Y.; and Wu, F. 2021. Task-Aware Part Mining Network for Few-Shot Learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 8433–8442.

Xu, W.; yifan xu; Wang, H.; and Tu, Z. 2021. Attentional Constellation Nets for Few-Shot Learning. In *International Conference on Learning Representations*.

Yang, L.; Li, L.; Zhang, Z.; Zhou, X.; Zhou, E.; and Liu, Y. 2020. DPGN: Distribution Propagation Graph Network for Few-Shot Learning. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 13387–13396.

Ye, H.-J.; Hu, H.; Zhan, D.-C.; and Sha, F. 2020. Few-Shot Learning via Embedding Adaptation with Set-to-Set Functions. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 8808–8817.

Yoon, J.; Kim, T.; Dia, O.; Kim, S.; Bengio, Y.; and Ahn, S. 2018. Bayesian Model-Agnostic Meta-Learning. In *Advances in Neural Information Processing Systems*, volume 31.

Zhang, B.; Li, X.; Ye, Y.; Huang, Z.; and Zhang, L. 2021a. Prototype Completion With Primitive Knowledge for Few-Shot Learning. In *CVPR*, 3754–3762.

Zhang, J.; Zhao, C.; Ni, B.; Xu, M.; and Yang, X. 2019. Variational Few-Shot Learning. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 1685–1694.

Zhang, X.; Meng, D.; Gouk, H.; and Hospedales, T. M. 2021b. Shallow bayesian meta learning for real-world few-shot recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 651–660.

Ziko, I. M.; Dolz, J.; Granger, E.; and Ayed, I. B. 2020. Laplacian Regularized Few-Shot Learning. In *ICML*, 11660–11670.

Appendix

Impact of AttFEX

In order to study the impact of the transductive feature extractor AttFEX, we exclude it during training and train the remaining architecture. Training proceeds exactly as mentioned before in the manuscript. As can be seen in Table 5, the exclusion of AttFEX from TRIDENT results in a substantial drop in classification performance across both datasets and task settings. Empirically, this further substantiates the importance of AttFEX's ability to render the feature maps transductive/task-aware. As explained earlier in the main manuscript, it is imperative to include y in the input to $q_{\phi_1}(.)$ for mathematical correctness of the variational inference formulation. However, in order to utilize TRIDENT as a classification and not a label reconstruction network, we choose not to input y to $q_{\phi_1}(.)$, but rather do so indirectly by inducing a semblance of label characteristics in the features extracted from the images in a task. Thus, it is important to realize that this ability of AttFEX to render feature maps transductive is not just an adhoc performance enhancer, but rather an essential part of TRIDENT since it allows us to not violate our generative and inference mechanics.

Table 5: Impact of AttFEX on classification accuracies.

	*mini*Imagenet		*tiered*Imagenet	
	(5-way, 1-shot)	**(5-way, 5-shot)**	**(5-way, 1-shot)**	**(5-way, 5-shot)**
AttFEX OFF	67.68 ± 0.55	78.53 ± 0.21	69.32 ± 0.76	79.32 ± 0.76
AttFEX ON	86.11 ± 0.59	95.95 ± 0.28	86.97 ± 0.50	96.57 ± 0.17

Additional Details of Datasets

***mini*Imagenet** (Vinyals et al. 2016) is a subset of ImageNet (Deng et al. 2009) for few-shot classification. It contains 100 classes with 600 samples each. We follow the predominantly adopted settings of (Ravi and Larochelle 2017; Chen et al. 2019) where we split the entire dataset into 64 classes for training, 16 for validation and 20 for testing. ***tiered*Imagenet** is a larger subset of ImageNet (Deng et al. 2009) with 608 classes and $779,165$ total images, which are grouped into 34 higher-level nodes in the *ImageNet* human-curated hierarchy. This set of nodes is partitioned into 20, 6, and 8 disjoint sets of training, validation, and testing nodes, and the corresponding classes form the respective meta-sets. **CUB** (Welinder et al. 2010) dataset has a total of 200 classes, split into training, validation and test sets following (Chen et al. 2019). We use this dataset to simulate the effect of a domain shift where the model is first trained on a (5-way, 1 or 5-shot) configuration of *mini*Imagenet and then tested on the test classes of CUB, as used in (Chen et al. 2019; Boudiaf et al. 2020; Ziko et al. 2020; Long et al. 2018).

Implementational Details

Let α_1 and α_2 respectively denote the scaling factors of the MSE and cross-entropy terms in our objective functions $\mathcal{L}_R$ and $\mathcal{L}_C$, as already defined in Subsection 4.2. The terms α and β respectively denote the learning rates of the *inner* and *meta* updates whereas B and n respectively denote the number of sampled tasks and adaptation steps of the *inner*-update of our end-to-end training process, as described in Algorithm 2. The hyperparameter values (**H.P.**) used for training TRIDENT on *mini*Imagenet and *tiered*Imagenet are shown in Table 6. We apply the same hyperparameters for the cross-domain testing scenario of *mini*Imagenet $\rightarrow$ CUB used for training TRIDENT on *mini*Imagenet, for the given (N-way, K-shot) configuration. Hyperparameters are kept fixed throughout training, validation and testing for a given configuration. Adam (Kingma and Ba 2015) optimizer is used for inner and meta-updates. Finally, the query, key and value extraction networks $f_q(,;\mathbf{W}_Q)$, $f_k(.;\mathbf{W}_K)$, $f_v(.;\mathbf{W}_V)$ of the AttFEX module only use $\texttt{Conv}_{1\times1}(.)$ and not the LeakyReLU(0.2) activation function for (5-way, 1-shot) tasks, irrespective of the dataset. We observed that utilizing BatchNorm (Ioffe and Szegedy 2015) in the decoder of z_s (p_{θ_2}) to train TRIDENT on (5-way, 5-shot) tasks of *mini*Imagenet and on (5-way, 1-shot) tasks of *tiered*Imagenet leads to better scores and improved stability during training. We used the ReLU activation function instead of LeakyReLU(0.2) to carry out training on (5-way, 1-shot) tasks of *tiered*Imagenet. Meta-learning objectives can lead to unstable optimization processes in practice, especially when coupled with stochastic sampling in latent spaces, as also previously observed in (Antreas Antoniou, Harrison Edwards, and Amos J. Storkey 2019; Rusu et al. 2019). For ease of experimentation we clip the meta-gradient norm at an absolute value of 1. TRIDENT converges in $82,000$ and $22,500$ epochs for (5-way, 1-shot) and (5-way, 5-shot) tasks of *mini*Imagenet, respectively and takes $67,500$ and $48,000$ epochs for convergence on (5-way, 1-shot) and (5-way, 5-shot) tasks of *tiered*Imagenet, respectively. This translates to an average training time of 110 hours on an 11GB NVIDIA 1080Ti GPU. Note that we did not employ any data augmentation, feature averaging or any other data apart from the corresponding training subset $\mathcal{D}^{tr}$, during training and evaluation.

Table 6: Hyperparameter values when training TRIDENT.

	*mini*Imagenet		*tiered*Imagenet	
H.P.	**5-way, 1-shot**	**5-way, 5-shot**	**5-way, 1-shot**	**5-way, 5-shot**
α_1	1e-2	1e-2	1e-2	1e-2
α_2	100	100	150	150
α	1e-3	1e-3	1.5e-3	1.7e-3
β	1e-4	1e-4	1.5e-4	1.7e-4
B	20	20	20	20
n	5	5	5	5

Additional Calibration Results

To further examine the reliability and calibration of our method, we assess the ECE, MCE (Guo et al. 2017) and Brier scores (BRIER 1950) of TRIDENT on the challenging *cross-domain* scenario of *mini*Imagenet $\rightarrow$ CUB for (5-way, 5-shot) tasks. Note that this an extension to Table 1 from the Experimental Evaluation section of the main manuscript and not an additional test on a new dataset since we already compute and compare the accuracies of TRIDENT on

Table 7: Style: **best** and second best.

Methods	ECE	MCE	Brier
Feature Transfer(Chen et al. 2019)	0.275	0.646	0.772
Baseline(Chen et al. 2019)	0.315	0.537	0.716
Matching Nets(Vinyals et al. 2016)	0.030	0.079	0.630
Proto Nets(Snell, Swersky, and Zemel 2017)	**0.009**	0.025	0.604
Relation Net(Sung et al. 2018)	0.234	0.554	0.730
DKT+Cos(Patacchiola et al. 2020)	0.236	0.426	0.670
BMAML(Yoon et al. 2018)	0.048	0.077	0.619
BMAML+Chaser(Yoon et al. 2018)	0.066	0.260	0.639
LogSoftGP(ML)(Galy-Fajou et al. 2020)	0.220	0.513	0.709
LogSoftGP(PL)(Galy-Fajou et al. 2020)	0.022	0.042	0.564
OVE(ML)(Snell and Zemel 2021)	0.049	0.066	0.576
OVE(PL)(Snell and Zemel 2021)	0.020	0.032	0.556
TRIDENT(Ours)	**0.009**	**0.02**	**0.276**

*mini*Imagenet → CUB in the main manuscript. When compared against other baselines that report these metrics on the aforementioned scenario, TRIDENT proves to be the most calibrated with the best reliability scores. This is shown in Table 7.

3

Statistical Modelling

Learning from experiences and feedback forms the cornerstone of human development. Scientists have long dreamed of making machines that mimic the characteristic human traits of making observations, thinking and learning. Machine learning is a field of study that lies at the confluence of statistics, computer science and psychology, aiming to make machines learn from experiences. In order to make machines extract meaningful insights and patterns from raw data, we need a few necessary ingredients:

- *Data* captured by observing reality.
- A *statistical model* to transform and process the data.
- An *objective function* that quantifies how well the model is performing.
- An *algorithm* that adjusts the models parameters to optimize for the objective function.

Data are the observations that the model experiences. In the context of statistical modelling, a dataset is a collection of many examples. Each example is a collection of features (dimensions) that are a result of quantitative measurement from some event or process that the machine learning algorithm must process. An example is typically represented as a vector $\boldsymbol{x}_i \in \mathbb{R}^d$, where $i \in [1, N]$, N denotes the total number of feature vectors in the dataset and d denotes the dimensionality of a feature vector. These examples are also referred to as data points, datums or samples. Based on the type of data captured and available for modelling, machine learning can be broadly categorized into Supervised and Unsupervised.

Supervised learning algorithms experience a dataset that not only contains feature vectors describing a task/event, but also a label or target associated to each example. For example, the Iris dataset (Fisher 1936) contains feature vectors describing properties of iris plants and also the target species of each of these feature vectors, represented as a discrete variable with as many dimensions as the total number of species in the dataset. Apart from discrete target variables, it is also possible to have continuous target variables, for example, a dataset containing important properties about the stock of a company at multiple timestamps and the target variable being the price of that stock for each of the recorded timestamps. The goal of a supervised machine learning algorithm is to process the feature vectors and correctly predict the target variable for ideally all possible test examples. Predicting a discrete target variable or category is called **Classification** and predicting a continuous real valued target variable is called **Regression**.

Unsupervised Learning algorithms experience a dataset containing only feature vectors describing the useful properties of the structure of this dataset. In cases like these, the goal of the machine learning algorithm is to process this dataset and ideally learn the underlying distribution of

the entire dataset, to then be used for density estimation at new, unseen data points, clustering examples into categories of similar examples, or implicit tasks of denoising and synthesis of useful examples. In both, Supervised and Unsupervised cases, the idea is to achieve the goal of solving a task not only on the seen examples (training set), but also to generalize on the unseen examples (test set).

3.1. Frequentist Statistics

Learning from data requires a principled approach for estimating the true unknown distribution of the dataset. As discussed earlier, unsupervised learning involves observing several examples of a random vector $\boldsymbol{x}$, and attempting to implicitly or explicitly learn the probability distribution $p(\boldsymbol{x})$, or some interesting properties of that distribution, while supervised learning involves observing several examples of a random vector $\boldsymbol{x}$ along with an associated value or vector $\boldsymbol{y}$, and learning to predict $\boldsymbol{y}$ from $\boldsymbol{x}$, usually by estimating the conditional distribution $p(\boldsymbol{y}|\boldsymbol{x})$.

Frequentist statistics involves modelling a parametric form of the true distribution and then maximizing the likelihood of the observed data with respect to the parameters. This perspective assumes that the true parameter value $\boldsymbol{\theta}$ is fixed but unknown, while the data is observed and known. Thus, the task now is to find a *point estimate* $\hat{\boldsymbol{\theta}}$ that maximizes the likelihood of the observed data, while $\hat{\boldsymbol{\theta}}$ being a random variable on account of being a function of the dataset. The estimate $\hat{\boldsymbol{\theta}}$ is known as the **Maximum Likelihood Estimator (MLE)** of the true parameter $\boldsymbol{\theta}$.

Consider the dataset to be a collection of N samples $\boldsymbol{X} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_N\}$ drawn independently from the same true but unknown data generating distribution $p_{data}(\boldsymbol{x})$. Let the model be defined by the parameters $\boldsymbol{\theta}$ and $p_{model}(\boldsymbol{x}|\boldsymbol{\theta})$ be a parametric family of distributions over the same space indexed by $\boldsymbol{\theta}$. The maximum likelihood estimator (MLE) for $\boldsymbol{\theta}$ is then given by:

$$\begin{aligned} \boldsymbol{\theta}_{\text{MLE}} &= \underset{\boldsymbol{\theta}}{argmax}\, p_{model}(\boldsymbol{X}|\boldsymbol{\theta}) \\ &= \underset{\boldsymbol{\theta}}{argmax} \prod_{i=1}^{N} p_{model}\left(\boldsymbol{x}_i|\boldsymbol{\theta}\right) \end{aligned} \tag{3.1}$$

The likelihood of the entire dataset can be written as a product of the likelihood of all individual samples since it is assumed that the data is sampled independently and from identical true distributions. This is known as the **i.i.d.** assumption. In order to avoid numerical underflow due to the multiplication of several small probability values, a logarithm of the likelihood is taken to conveniently transform the product into a sum while keeping the argmax same:

$$\begin{aligned} \boldsymbol{\theta}_{\text{MLE}} &= \underset{\boldsymbol{\theta}}{argmax} \sum_{i=1}^{N} \log p_{model}\left(\boldsymbol{x}_i|\boldsymbol{\theta}\right) \\ \boldsymbol{\theta}_{\text{MLE}} &= \underset{\boldsymbol{\theta}}{argmax}\, \mathbb{E}_{\boldsymbol{x}\sim\hat{p}_{\text{data}}} \log p_{\text{model}}\left(\boldsymbol{x}|\boldsymbol{\theta}\right) \end{aligned} \tag{3.2}$$

Since the argmax doesn't change when scaling the objective function, dividing the expression by N gives the expectation of the log-likelihood with respect to the empirical distribution $\hat{p}_{\text{data}}$. In practice, it is common to minimize the negative log-likelihood rather than maximizing the log-likelihood as the objective function:

$$\boldsymbol{\theta}_{\text{MLE}} = \underset{\boldsymbol{\theta}}{argmin}\, -\mathbb{E}_{\boldsymbol{x}\sim\hat{p}_{\text{data}}} \log p_{\text{model}}\left(\boldsymbol{x}|\boldsymbol{\theta}\right) \tag{3.3}$$

In the case of **Supervised** Learning, the negative log-likelihood (NLL) to be minimized takes the form:

$$\boldsymbol{\theta}_{\text{MLE}} = \underset{\boldsymbol{\theta}}{argmin} -\sum_{i=1}^{N} \log p_{model}\left(\boldsymbol{y}_i|\boldsymbol{x}_i, \boldsymbol{\theta}\right), \tag{3.4}$$

while for **Unsupervised** Learning, the negative log-likelihood (NLL) to be minimized takes the form:

$$\boldsymbol{\theta}_{\text{MLE}} = \underset{\boldsymbol{\theta}}{argmin} -\sum_{i=1}^{N} \log p_{model}\left(\boldsymbol{x}_i|\boldsymbol{\theta}\right), \tag{3.5}$$

3.1.1. Linear Regression using MLE

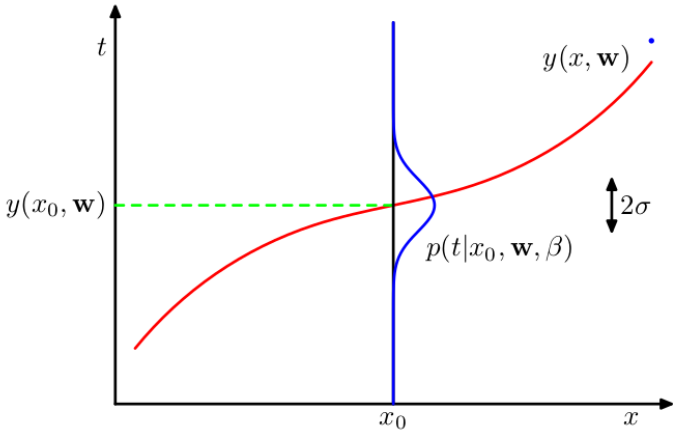


Figure 3.1: Linear Regression using a Gaussian distribution as the likelihood of y at a given $\boldsymbol{x}_0$ with model parameters $\boldsymbol{w}$ and $\beta^{-1} = \sigma^2$. Figure courtesy Bishop (n.d.)

Linear regression is a widely used method for predicting a real-valued continuous target variable $y \in \mathbb{R}$ given a feature vector $\boldsymbol{x} \in \mathbb{R}^D$. Since regression deals with modelling a continuous valued target variable, the likelihood distribution (p_{model}) is assumed to be Gaussian, with its mean parameterized by the model function $\hat{y} = \boldsymbol{w}^\top + b$. In other words, a linear regressor tries to fit a Gaussian distribution centered at the prediction value $\hat{y}$ of a given input feature vector $\boldsymbol{x}$, as depicted in Fig. 3.1. The likelihood thus takes the form:

$$p(y \mid \boldsymbol{x}, \boldsymbol{\theta}) = \mathcal{N}\left(y \mid b + \boldsymbol{w}^\top \boldsymbol{x}, \sigma^2\right) \tag{3.6}$$

where $\boldsymbol{\theta} = (b, \boldsymbol{w}, \sigma^2)$ are all the parameters of the model, with b being the bias term. However, a straight line will not provide a good fit to most data sets. To this end, a nonlinear transformation can be applied to the input features, by replacing $\boldsymbol{x}$ with $\phi(\boldsymbol{x})$ to get

$$p(y \mid \boldsymbol{x}, \boldsymbol{\theta}) = \mathcal{N}\left(y \mid b + \boldsymbol{w}^\top \phi(\boldsymbol{x}), \sigma^2\right) \tag{3.7}$$

Note that the method would still be called linear regression since the relationship between the model parameters and the target variable is still linear. The likelihood of the entire dataset is given by the product of likelihoods of all individual datums since we work under the i.i.d. assumption. To find the MLE of the model parameters, the NLL of the dataset is minimized, which is given by:

$$\begin{aligned} \mathrm{NLL}\left(\boldsymbol{w}, \sigma^2\right) &= -\sum_{i=1}^{N} \log\left[\left(\frac{1}{2\pi\sigma^2}\right)^{\frac{1}{2}} \exp\left(-\frac{1}{2\sigma^2}\left(y_i - \boldsymbol{w}^\top \boldsymbol{x}_i\right)^2\right)\right] \\ &= \frac{1}{2\sigma^2}\sum_{i=1}^{N}\left(y_i - \hat{y}_i\right)^2 + \frac{N}{2}\log\left(2\pi\sigma^2\right) \end{aligned} \tag{3.8}$$

where the model predictions $\hat{y} = \boldsymbol{w}^\top \boldsymbol{x}_i$. The NLL in Equation (3.8) is minimized by setting the derivatives of the NLL with respect to the parameters equal to 0. Computing the derivative of the NLL with respect to the model's weight parameter $\boldsymbol{w}$ gives the Ordinary Least Squares (OLS) objective function:

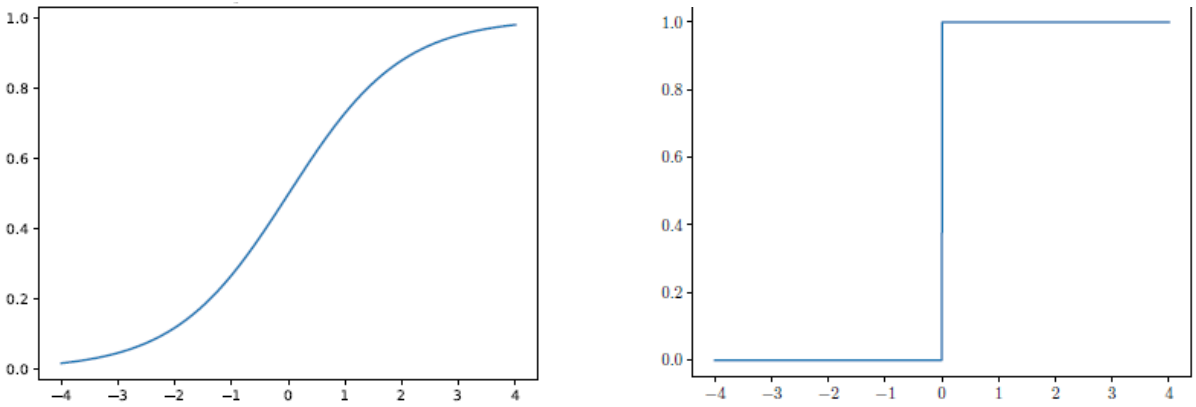
$$\mathrm{OLS}(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{N}\left(y_i - w^\top \boldsymbol{x}_i\right)^2 = \frac{1}{2}\|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|_2^2 = \frac{1}{2}(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})^\top(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}) \tag{3.9}$$

Optimizing this loss function for linear regression is also commonly called the method of **sum of least squares**. Solving the OLS objective gives the MLE of $\boldsymbol{w}$:

$$\boldsymbol{w}_{MLE} = \left(\boldsymbol{X}^\top \boldsymbol{X}\right)^{-1} \boldsymbol{X}^\top \boldsymbol{y} \tag{3.10}$$

By simply plugging in the $\boldsymbol{w}_{MLE}$ into the NLL expression and equating the derivative with respect to σ^2 to zero, we obtain σ^2_{MLE}:

$$\sigma^2_{MLE} = \left(\boldsymbol{y}^\top \boldsymbol{y} - \boldsymbol{y}^\top \boldsymbol{X} \boldsymbol{w}_{MLE}\right) \tag{3.11}$$



(a) Sigmoid **(b)** Heaviside

Figure 3.2: Activation functions to output probability values.

3.1.2. Logistic Regression using MLE

Logistic regression is a widely used discriminative classification model $p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})$ where $\boldsymbol{x}$ is a feature vector, $y \in [1,....,C]$ is the class label, and $\boldsymbol{\theta}$ are the model parameters. If $C = 2$, this is known as binary logistic regression, and if $C > 2$, it is known as multi-class logistic regression. For ease of representation and explanation, only binary logistic regression has been discussed in this text, with straight forward extension to the multi-class case. Since classification deals with the modelling of discrete target variables ($0, 1$ in the case of binary classification), it is suitable to model the likelihood distribution (p_{model}) as a Bernoulli distribution of $\boldsymbol{y}$, dependent on the feature vectors $\boldsymbol{x}$ and the model's parameters $\boldsymbol{w}$, b

$$p(y \mid \boldsymbol{x};\boldsymbol{\theta}) = \mathrm{Ber}\left(y \mid \sigma\left(\boldsymbol{w}^{\top}\boldsymbol{x} + b\right)\right) \tag{3.12}$$

where σ and b denote the sigmoid function and the model's bias term, respectively. The total parameters of the model are denoted by $\boldsymbol{\theta} = \{\boldsymbol{w}, b\}$. The role of the sigmoid function is to squash the output of the model $\boldsymbol{w}^{\top}\boldsymbol{x} + b$ into the interval $(0, 1)$. This is done because the Bernoulli distribution expects a probability value $\in [0, 1]$ as an input for $y = 1$ by default. Therefore, the modelling task for binary classification boils down to correctly output a value > 0 for probability value > 0.5 when $y = 1$, and a value < 0 for probability value < 0.5 for $y = 0$, as can be seen in Fig 3.2. The probability values estimated by the model are given by:

$$p(y = 1 \mid \boldsymbol{x};\boldsymbol{\theta}) = \sigma(a) = \frac{1}{1 + e^{-a}} \tag{3.13}$$

where $a = \boldsymbol{w}^{\top}\boldsymbol{x} + b$ are called the log-odds ($\log \frac{p}{1-p}$), where $p = p(y = 1 \mid \boldsymbol{x};\boldsymbol{\theta})$. The sigmoid function can be considered as a "soft" version of the Heaviside function and is thus fit for statistical modelling due to its differentiability. Here, the model's output is a linear function of the input. This is not effective for modelling data that is not linearly separable. To include non-linearity in the model's predictions, the input can be transformed using a basis function $\phi(\boldsymbol{x})$, and can then be used as the input in the model's

function. The negative log-likelihood of the Bernoulli distribution $p(y \mid \boldsymbol{x}; \boldsymbol{\theta})$ is thus given by:

$$\begin{aligned}
\text{NLL}(\boldsymbol{\theta}) &= -\frac{1}{N} \log p(Y \mid \boldsymbol{X}; \boldsymbol{\theta}) = -\frac{1}{N} \log \prod_{i=1}^{N} \text{Ber}\left(y_i \mid \hat{y}_i\right) \\
&= -\frac{1}{N} \sum_{i=1}^{N} \log \left[\hat{y}_i^{y_i} \times (1-\hat{y}_i)^{1-y_i}\right] \\
&= -\frac{1}{N} \sum_{i=1}^{N} \left[y_i \log \hat{y}_i + (1-y_i) \log\left(1-\hat{y}_i\right)\right] \\
&= \frac{1}{N} \sum_{i=1}^{N} \mathbb{H}_{ce}\left(y_i, \hat{y}_i\right)
\end{aligned} \tag{3.14}$$

where $\hat{y} = \sigma(\boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}) + b)$ and $\mathbb{H}_{ce}(p, q) = -[p \log q + (1-p) \log(1-q)]$ is the binary **cross-entropy** loss.

3.2. Bayesian Statistics

The frequentists view probabilities in terms of frequencies of random and repeatable events. On the other hand, the Bayesian perspective is to view probabilities as a measure for providing a quantification of uncertainty. In the frequentist setting, parameters are assumed to be fixed and their values are determined by an estimator that maximizes the likelihood of the observed data with respect to the parameters. The variance of these estimates are calculated by considering the distribution of all possible datasets given the parameters. Rather than assuming the parameters to be fixed, Bayesians view the actual observed data to be the only single dataset, and the uncertainty in the data generating parameters is expressed through a probability distribution over them. This involves making assumptions about the parameters apriori. The assumptions about the parameters are then updated by making precise revisions of uncertainty in the parameters, in light of new evidence (data).

Consider the observed data $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i) \forall i \in [1, N]\}$ for a supervised model and $\mathcal{D} = \{(\boldsymbol{x}_i) \forall i \in [1, N]\}$ for unsupervised models. In contrast to the point estimate MLE of the parameters $\boldsymbol{\theta}$, Bayesian statistics involve computing a probability distribution over the parameters in light of evidence $\mathcal{D}$. This is done using the Bayes rule:

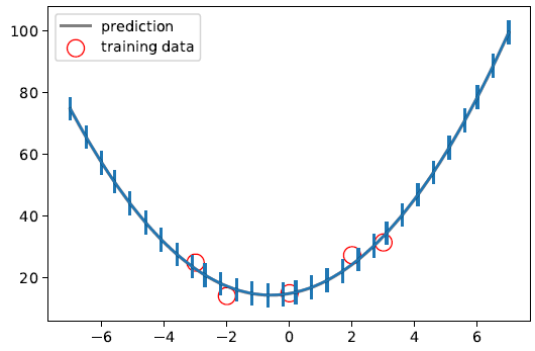
$$p(\boldsymbol{\theta} \mid \mathcal{D}) = \frac{p(\boldsymbol{\theta}) p(\mathcal{D} \mid \boldsymbol{\theta})}{p(\mathcal{D})} = \frac{p(\boldsymbol{\theta}) p(\mathcal{D} \mid \boldsymbol{\theta})}{\int p\left(\boldsymbol{\theta}'\right) p\left(\mathcal{D} \mid \boldsymbol{\theta}'\right) d\boldsymbol{\theta}'} \tag{3.15}$$

The term $p(\boldsymbol{\theta} \mid \mathcal{D})$ is known as the **posterior** distribution to represent the uncertainty in $\boldsymbol{\theta}$. The **prior** distribution $p(\boldsymbol{\theta})$ reflects the assumption/information about the parameters before seeing the evidence/data. The **likelihood** function $p(\mathcal{D} \mid \boldsymbol{\theta})$ reflects the data we expect to see for each setting of the parameters. The **marginal likelihood** $p(\mathcal{D})$ can be interpreted as the average probability of the data over all possible prior parameters. $p(\mathcal{D})$ is a constant and can be ignored when computing the relative probabilities of $\boldsymbol{\theta}$ values. The **posterior predictive** distribution can also be computed once the uncertainty measure over $\boldsymbol{\theta}$ is updated (posterior distribution) by marginalizing out the parameters:

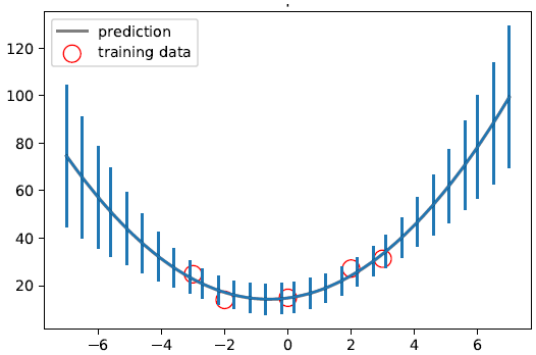
$$p(y \mid \boldsymbol{x}, \mathcal{D}) = \int p(y \mid \boldsymbol{x}, \boldsymbol{\theta}) p(\boldsymbol{\theta} \mid \mathcal{D}) d\boldsymbol{\theta} \tag{3.16}$$

3.2.1. Bayesian Linear Regression

To compute the posterior over parameters $p(\boldsymbol{\theta} \mid \mathcal{D})$ and the posterior predictive distribution $p(y \mid \boldsymbol{x}, \mathcal{D})$, an assumption about the prior $p(\boldsymbol{\theta})$ must be made. For simplicity and reasons that will be stated later in this chapter, a Gaussian prior is assumed: $p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w} \mid \widetilde{\boldsymbol{w}}, \widetilde{\boldsymbol{\Sigma}})$. σ^2 is assumed to be known for simplicity and thus only $\boldsymbol{w}$ is optimized.

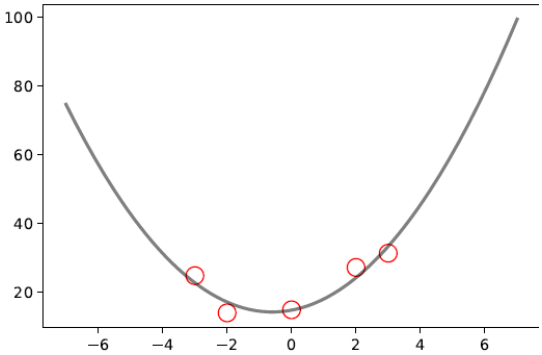


(a) MLE Posterior Predictive.

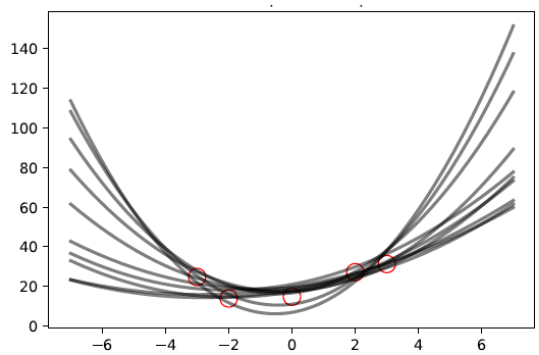


(b) Bayesian Posterior Predictive.

Figure 3.3: Prediction Densities of MLE and Bayesian approaches. Figure courtesy Murphy (2022).



(a) MLE Posterior.



(b) Bayesian Posterior.

Figure 3.4: Posterior Densities of MLE and Bayesian approaches. Figure courtesy Murphy (2022).

Due to the **i.i.d.** assumption, the likelihood of the dataset can be written as:

$$p\left(\mathcal{D} \mid \boldsymbol{w}, \sigma^2\right) = \prod_{i=1}^{N} p\left(y_i \mid \boldsymbol{w}^\top \boldsymbol{x}, \sigma^2\right) = \mathcal{N}\left(\boldsymbol{y} \mid \boldsymbol{X}\boldsymbol{w}, \sigma^2 \boldsymbol{I}_N\right) \tag{3.17}$$

where $\boldsymbol{I}_N$ is the NxN identity matrix. Since both the prior and likelihood are Gaussians, by conjugacy, the posterior is Gaussian too, which reduces the burden of computing the marginal likelihood. The posterior can be derived as follows:

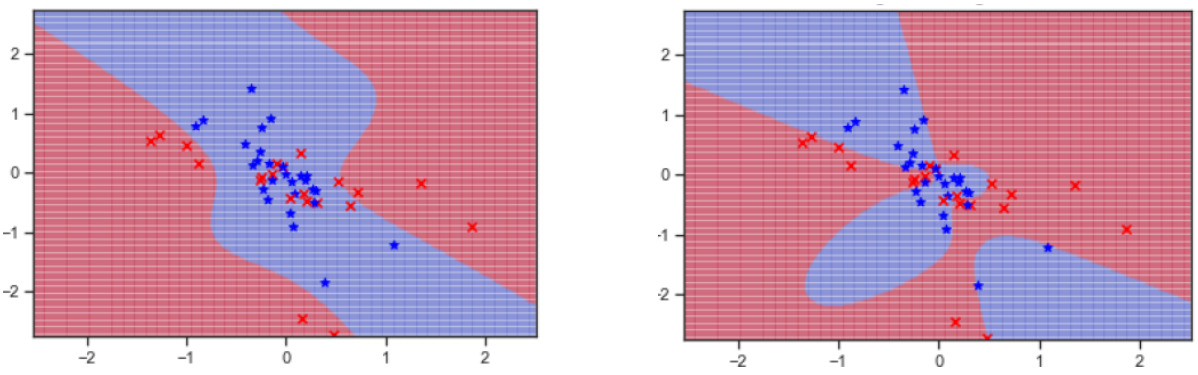
$$\begin{aligned} p\left(\boldsymbol{w} \mid \boldsymbol{X}, \boldsymbol{y}, \sigma^2\right) &\propto \mathcal{N}(\boldsymbol{w} \mid \breve{\boldsymbol{w}}, \breve{\boldsymbol{\Sigma}}) \mathcal{N}\left(\boldsymbol{y} \mid \boldsymbol{X}\boldsymbol{w}, \sigma^2 \boldsymbol{I}_N\right) = \mathcal{N}(\boldsymbol{w} \mid \widehat{w}, \widehat{\boldsymbol{\Sigma}}) \\ \widehat{w} &\triangleq \widehat{\boldsymbol{\Sigma}}\left(\breve{\boldsymbol{\Sigma}}^{-1}\breve{w} + \frac{1}{\sigma^2}\boldsymbol{X}^\top y\right) \\ \widehat{\boldsymbol{\Sigma}} &\triangleq \left(\breve{\boldsymbol{\Sigma}}^{-1} + \frac{1}{\sigma^2}\boldsymbol{X}^\top \boldsymbol{X}\right)^{-1} \end{aligned} \tag{3.18}$$

where $\breve{\boldsymbol{w}}$ and $\breve{\boldsymbol{\Sigma}}$ are the posterior mean and covariance, respectively.

Now, to compute the uncertainty about the predictions of future outputs, the posterior predictive distribution can be calculated as follows:

$$\begin{aligned} p\left(y \mid \boldsymbol{x}, \mathcal{D}, \sigma^2\right) &= \int \mathcal{N}\left(y \mid \boldsymbol{x}^\top \boldsymbol{w}, \sigma^2\right) \mathcal{N}(\boldsymbol{w} \mid \widehat{\boldsymbol{\mu}}, \widehat{\boldsymbol{\Sigma}}) dw \\ &= \mathcal{N}\left(y \mid \widehat{\boldsymbol{\mu}}^\top \boldsymbol{x}, \widehat{\boldsymbol{\sigma}}^2(x)\right) \end{aligned} \tag{3.19}$$

where $\widehat{\sigma}^2(\boldsymbol{x}) \triangleq \sigma^2 + \boldsymbol{x}^\top \widehat{\boldsymbol{\Sigma}} \boldsymbol{x}$ is the variance of the posterior predictive distribution at a given point $\boldsymbol{x}$ using the updated $\boldsymbol{\theta}$ using N training data points. This variance in the posterior predictive distribution



(a) $\lambda = 1$. **(b)** $\lambda = 10^{-5}$.

Figure 3.5: Regularized Logistic Regression with multiple λ values. Figure courtesy Murphy (2022).

is a function of the given observation variance (σ^2) and the parameter's variance $\widehat{\boldsymbol{\Sigma}}$. This parameter variance grows as one moves away from the training samples, representing uncertainty (Fig. 3.3b). This indication and measure of uncertainty lacks from the MLE estimation of the parameters (Fig. 3.3a). In the case of MLE, the optimal $\boldsymbol{w}_{MLE}$ and σ^2_{MLE} from Equations (3.10), (3.11) are plugged into the posterior predictive distribution to obtain $p\left(y \mid \boldsymbol{x}^\top \boldsymbol{w}_{\hat{MLE}}, \sigma^2_{MLE}\right)$. The variance of this posterior predictive distribution is independent of the new data and thus remains constant throughout (Fig. 3.3a). Moreover, the MLE method only gives one unique function for the posterior (Fig. 3.4a) since the estimator only optimizes for a point estimate. The Bayesian posterior allows for the sampling of a range of different functions due to the availability of an entire distribution (Fig. 3.4b).

3.2.2. Bayesian Logistic Regression

In the previous subsection, we assumed that the prior distribution over parameters is a Gaussian. This allowed for the analytical computation of the exact posterior distribution and the need to compute the marginal likelihood by evaluating an integral was avoided. This property of the prior distribution makes it a **conjugate** prior for the Gaussian likelihood distribution. This is not possible with all cases of probability distributions. Such a case can be observed here, with the Logistic Regression setup where the likelihood is assumed to be a Bernoulli distribution. With the loss of analytical tractability of the exact posterior distribution, it is either possible to compute certain approximations to the exact posterior or to compute a point estimate of the optimal parameter value that maximizes the posterior distribution $\boldsymbol{\theta}_{MAP}$. An approach to the former is discussed in the chapter 5 and the latter is called the **Maximum Aposteriori Estimation** (MAP). Firstly, the posterior distribution is directly proportional to the product of the likelihood and the prior:

$$p\left(\boldsymbol{w} \mid \boldsymbol{X}, \boldsymbol{y}\right) \propto Ber(\boldsymbol{y} \mid \boldsymbol{w}, \boldsymbol{X}) p\left(\boldsymbol{\theta}\right) \tag{3.20}$$

Assuming that the prior distribution is a standard normal distribution with zero mean, the MAP estimate is given by:

$$\hat{\boldsymbol{w}}_{\text{MAP}} = \underset{\boldsymbol{w}}{\text{argmin}} \text{NLL}(\boldsymbol{w}) + \lambda \|\boldsymbol{w}\|_2^2 \tag{3.21}$$

This is a consequence of taking a negative log on both sides of the equation, as has been described in earlier subsections. Thus, the MAP estimate of the parameters is just an l_2 **regularized** version of the MLE estimate of $\boldsymbol{w}$. This is simply a consequence of assuming a standard normal prior for the weights $p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w} \mid 0, \boldsymbol{I})$. The larger the value of λ, the more the parameters are penalized for being large (deviating from the zero-mean prior), and thus, lesser the flexibility of the model (Fig. 3.5).

The MAP estimate obtained for Linear Regression using the same assumption over the prior weights is called **Ridge Regression** and follows the same negative log-likelihood, but with an additional regularization term of the l_2 norm of the weights, as in equation (3.21). Therefore, the final training objective/ loss function for MAP estimation becomes:

$$\mathcal{L}(w) = \mathbb{H}_{ce}(\boldsymbol{y}, \sigma(\boldsymbol{X}\boldsymbol{w})) + \lambda \|\boldsymbol{w}\|_2^2 \tag{3.22}$$

for Logistic Regression and

$$\mathcal{L}(w) = \frac{1}{2}\|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|_2^2 + \lambda\|\boldsymbol{w}\|_2^2 \tag{3.23}$$

for Linear Regression.

3.3. Why Frequentist Statistics Works

The maximum likelihood estimator can be shown to be the best asymptotic estimator, as the number of examples $N \to \infty$. The maximum likelihood estimator of a parameter has the property that as the number of samples approaches infinity, the maximum likelihood estimate of the parameter value converges to the true value of the data generating parameter, given certain appropriate conditions. These appropriate conditions are:

- The true distribution p_{data} must lie within the model family $p_{model}(\cdot;\theta)$. Otherwise, it is impossible to recover p_{data} using any estimator.
- The true distribution p_{data} must correspond to exactly one value of θ. Otherwise, MLE can recover the correct p_{data} , but will be unable to choose the true θ parameter responsible for generating the data.

MLE can also be viewed as a simple point approximation to the Bayesian posterior $p(\boldsymbol{\theta} \mid \mathcal{D})$ using a uniform prior, in which case the MAP and MLE estimates become equal since the prior $p(\boldsymbol{\theta})$ is independent of the parameter. Another way of reasoning for the use of MLE on why it works is the following. The predictive distribution of MLE $p(\boldsymbol{y} \mid \hat{\boldsymbol{\theta}}_{MLE})$ is actually the closest possible distribution to the **empirical distribution** of the observed data. To prove this, first, the empirical distribution is defined by the Dirac Delta distribution:

$$p_{\mathcal{D}}(\boldsymbol{X}, \boldsymbol{y}) = p_{\mathcal{D}}(\boldsymbol{y} \mid \boldsymbol{X}) p_{\mathcal{D}}(\boldsymbol{x}) = \frac{1}{N}\sum_{i=1}^{N} \delta\left(x - x_i\right) \delta\left(y - y_i\right) \tag{3.24}$$

Now, a model must be created that has a distribution $q(\boldsymbol{Y} \mid \boldsymbol{X})$ as close to the true $_{\mathcal{D}}(\boldsymbol{y} \mid \boldsymbol{X})$ as possible. A suitable metric to measure this is the **KL Divergence**:

$$\begin{aligned} D_{\mathrm{KL}}(p\|q) &= \sum_{\boldsymbol{y}} p(\boldsymbol{y}) \log \frac{p(\boldsymbol{y})}{q(\boldsymbol{y})} \\ &= \underbrace{\sum_{\boldsymbol{y}} p(\boldsymbol{y}) \log p(\boldsymbol{y})}_{\mathbb{H}(p)} - \underbrace{\sum_{\boldsymbol{y}} p(\boldsymbol{y}) \log q(\boldsymbol{y})}_{\mathbb{H}_{ce}(p,q)} \end{aligned} \tag{3.25}$$

where $\mathbb{H}(p)$ is the entropy of p and $\mathbb{H}_{ce}(p,q)$ is the cross-entropy between p, q. $D_{\mathrm{KL}}(p\|q) \geq 0$, with equality iff $p = q$. Defining $q = q(\boldsymbol{y} \mid \boldsymbol{X})$ and $p = p_{\mathcal{D}}(\boldsymbol{y} \mid \boldsymbol{X})$, the expected KL divergence becomes:

$$\begin{aligned} \mathbb{E}_{p_{\mathcal{D}(\boldsymbol{x})}}\left[D_{\mathbb{K}}\left(p_{\mathcal{D}}(Y \mid x)\|q(Y \mid x)\right)\right] &= \sum_{\boldsymbol{x}} p_{\mathcal{D}}(\boldsymbol{x}) \left[\sum_{\boldsymbol{y}} p_{\mathcal{D}}(\boldsymbol{y} \mid x) \log \frac{p_{\mathcal{D}}(\boldsymbol{y} \mid x)}{q(\boldsymbol{y} \mid \boldsymbol{x})}\right] \\ &= \text{ const } - \sum_{\boldsymbol{x},\boldsymbol{y}} p_{\mathcal{D}}(\boldsymbol{x}, \boldsymbol{y}) \log q(\boldsymbol{y} \mid \boldsymbol{x}) \\ &= \text{ const } - \frac{1}{N}\sum_{i=1}^{N} \log p\left(\boldsymbol{y}_i \mid \boldsymbol{x}_i, \boldsymbol{\theta}\right) \end{aligned} \tag{3.26}$$

Minimizing this is equivalent to minimizing the NLL in Equations (3.14), (3.8).

3.4. The problem with Bayesian Statistics

Given a likelihood $p(\mathcal{D} \mid \boldsymbol{\theta})$ and a prior $p(\theta)$, we can compute the posterior $p(\theta \mid \mathcal{D})$ using Bayes rule (Equation (3.15)). However, actually performing this computation is usually analytically intractable, except for the cases where the prior selected is a conjugate to the likelihood distribution (Section 3.2.1), or models where all the latent variables come from a small finite set of possible values. This limits our selection of priors to only the conjugate cases, depending on the likelihood distribution. This also means that the choice of the prior is now made on the basis of mathematical convenience rather than as a reflection of prior beliefs. Moreover, in cases where there is no prior information available about the parameters, a poor choice of the prior distribution can consequently give poor results with high confidence. This leads to drawing misleading conclusions about the true data distribution.

3.5. Structural Causal Models

In order to study the questions of causality, there must be a principled way to set the assumptions about the causal story behind a data set. To do so, the concept of the structural causal model (SCM) comes into the picture, which is a way of describing the relevant features of the world and how they interact with each other. Specifically, a structural causal model describes how nature assigns values to variables of interest. Every SCM is associated with a *graphical causal model*, which consists of a set of nodes representing the variables in the dataset and edges representing the relationship between the nodes. This text only deals with graphical models that are **directed acyclic graphs (DAGs)**. Because of the association between SCM and graphs, graphs can provide the definition of causation in a given dataset. Specifically, if, in a graphical model, a variable $\boldsymbol{X}$ is the child of another variable $\boldsymbol{Y}$, then $\boldsymbol{Y}$ is a direct cause of $\boldsymbol{X}$ and if $\boldsymbol{X}$ is a descendant of $\boldsymbol{Y}$, then $\boldsymbol{Y}$ is a potential cause of $\boldsymbol{X}$.

3.5.1. Decomposition Rules

For any model whose graph is acyclic, the joint distribution of the variables in the model is given by the product of the conditional distributions $P(child \mid parents)$ over all the families in the graph. Formally, this is written as:

$$P\left(x_1, x_2, \ldots, x_n\right) = \prod_i P\left(x_i \mid pa_i\right) \tag{3.27}$$

where pa_i stands for the parents of child nodes x_i, and the index i is over all possible nodes it the graph (n). For example, consider the simple chain graph $\boldsymbol{X} \rightarrow \boldsymbol{Y} \rightarrow \boldsymbol{Z}$. For this, the joint distribution can be decomposed as:

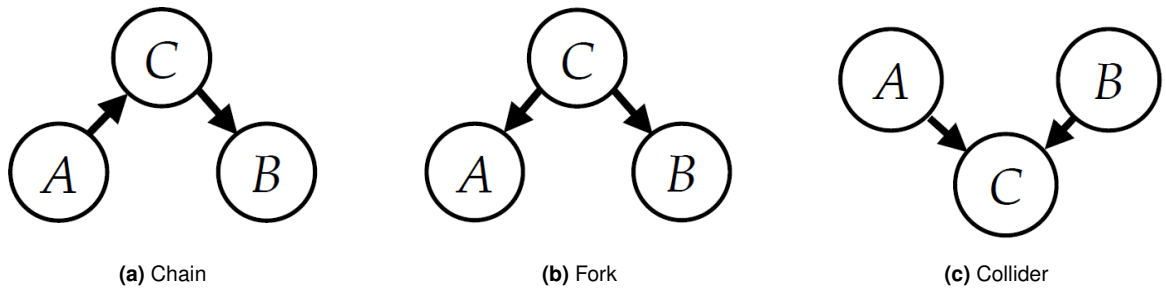
$$P(X = x, Y = y, Z = z) = P(X = x)P(Y = y \mid X = x)P(Z = z \mid Y = y) \tag{3.28}$$

3.5.2. Rules of Conditional Independence

In most practical cases, we do not have a probabilistically specified causal model, but only a graphical structure of the model based on our assumptions of the underlying generative process of the data. We know the causal relationships of the variables, but we do not know the strength or nature of the relationships. Even with such limited information, we can discern a great deal about the data set generated by the model. From an unspecified graphical causal model, i.e., one in which we know which variables are functions of which others, but not the specific nature of the functions that connect them, we can learn which variables in the data set are independent of each other and which are independent of each other conditional on other variables. These independencies will be true of every data set generated by a causal model with that graphical structure, regardless of the specific functions attached to the SCM (Glymour, Pearl, and Jewell 2016).

Rule 1: Conditional Independence in Chains Two variables, $\boldsymbol{A}$ and $\boldsymbol{C}$, are conditionally independent given $\boldsymbol{B}$, if there is only one unidirectional path between $\boldsymbol{A}$ and $\boldsymbol{C}$, and $\boldsymbol{B}$ is any set of variables that intercepts that path (Fig. 3.6a).

Rule 2: Conditional Independence in Forks If a variable $\boldsymbol{C}$ is a common cause of variables $\boldsymbol{A}$ and $\boldsymbol{B}$ (Fig. 3.6b), and there is only one path between $\boldsymbol{A}$ and $\boldsymbol{B}$, then $\boldsymbol{A}$ and $\boldsymbol{B}$ are independent conditional on $\boldsymbol{C}$.



(a) Chain **(b)** Fork **(c)** Collider

Figure 3.6: Types of DAGs. Figure courtesy Glymour, Pearl, and Jewell (2016)

Rule 3: Conditional Independence in Colliders If a variable $\boldsymbol{C}$ is the collision node between two variables $\boldsymbol{A}$ and $\boldsymbol{B}$ (Fig. 3.6c), and there is only one path between $\boldsymbol{A}$ and $\boldsymbol{B}$, then $\boldsymbol{A}$ and $\boldsymbol{b}$ are unconditionally independent but are dependent conditional on $\boldsymbol{C}$ and any descendants of $\boldsymbol{C}$.

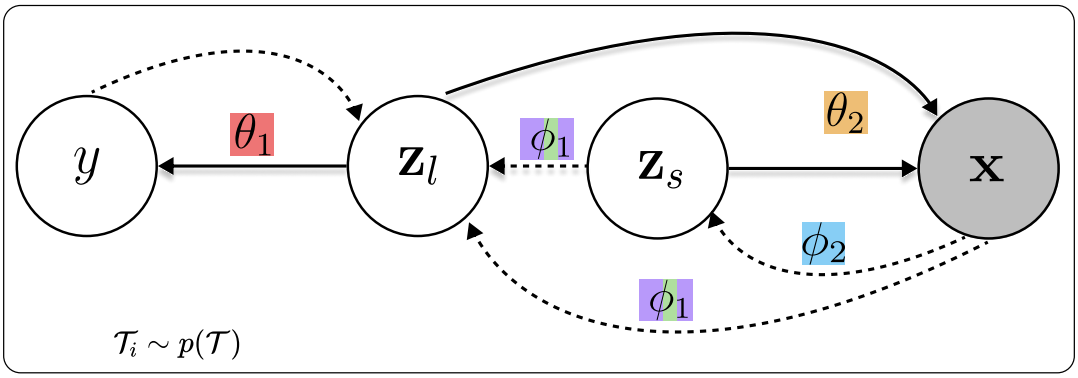


Figure 3.7: The DAG of TRIDENT with dotted lines indicate inference and solid lines refer to generative processes. The dependency of $\boldsymbol{z}_l$ on $\boldsymbol{z}_s$ has been introduced in order to incorporate their conditional dependence given $\boldsymbol{x}$.

The DAG of TRIDENT has been specified in Fig. 3.7 of the scientific article. It is important to notice that $\boldsymbol{x}$ acts as the collider node for $\boldsymbol{z}_l$ and $\boldsymbol{z}_s$ since these two latent variables are parents of $\boldsymbol{x}$, as is the case in Fig. 3.6c. This means that $q(\boldsymbol{z}_l, \boldsymbol{z}_s \mid \boldsymbol{x}) \neq q(\boldsymbol{z}_l \mid \boldsymbol{x}) q(\boldsymbol{z}_s \mid \boldsymbol{x})$ since $\boldsymbol{z}_l$ and $\boldsymbol{z}_s$ are dependent on each other conditional on $\boldsymbol{x}$. This is why the inference mechanics are assumed to have a factorized form $q(\boldsymbol{z}_l, \boldsymbol{z}_s \mid \boldsymbol{x}) = q(\boldsymbol{z}_s \mid \boldsymbol{x}) q(\boldsymbol{z}_l \mid \boldsymbol{x}, \boldsymbol{z}_s)$. This way a path is forged to allow *necessary* semantic latent information to flow through the label inference network. This could have been assumed to be the other way round as: $q(\boldsymbol{z}_l, \boldsymbol{z}_s \mid \boldsymbol{x}) = q(\boldsymbol{z}_l \mid \boldsymbol{x}) q(\boldsymbol{z}_s \mid \boldsymbol{x}, \boldsymbol{z}_l)$, but this does not hold logical relevance because label information does not directly contribute to the extraction of semantic features.

4

Deep Learning

Deep neural networks are biologically inspired function approximators that are designed to perform a wide array of tasks such as text modelling, speech synthesis, tabular data prediction, image classification, activity recognition, 3D scene generation etc. These networks are typically compositions of multiple functions (neurons) stacked on top of each other that operate in parallel. Each of these functions/units can be analogously thought of as neurons (McCulloch and Pitts 1943) that receive an input from the previous network of units as a vector and output a scalar activation value.

4.1. Deep Feedforward Networks

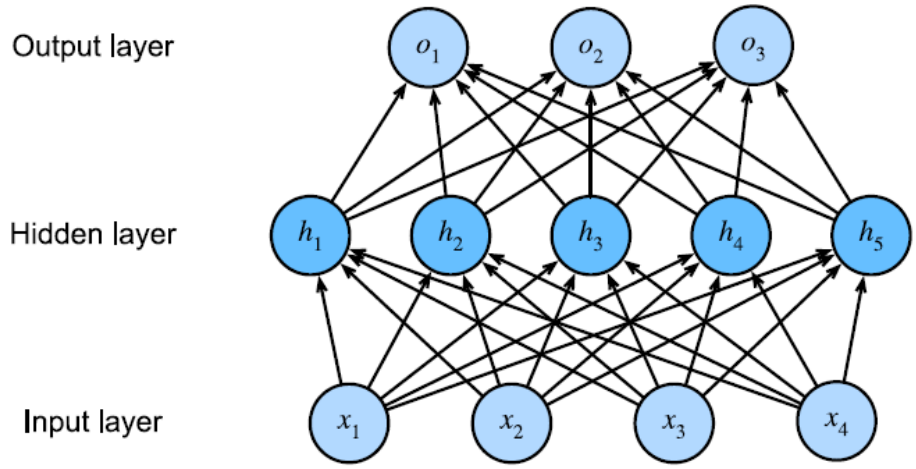
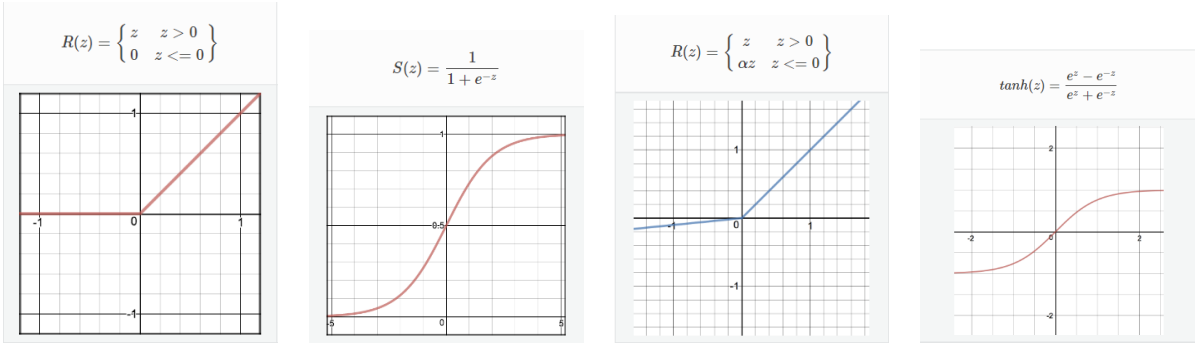


Figure 4.1: Multilayer Perceptron with 1 layer. Figure courtesy A. Zhang et al. (2021)

Deep feedforward networks, also called multilayer perceptrons (MLPs), process data to find patterns and extract meaningful information for statistical generalization. This type of networks are composed of multiple functions, connected in a chain, to give the final output. The length of the chain is called the depth of the network while each component in the chain is called a layer. For eg: $f(\boldsymbol{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\boldsymbol{x})))$ represents a neural network of depth 3 where $f^{(1)}$, $f^{(2)}$ and $f^{(3)}$ are the first, second and third layers of the neural network. The goal of this network $f(\boldsymbol{x})$ is to approximate some true function $y = f^*(\boldsymbol{x})$ that maps the input $\boldsymbol{x}$ to the output y. This is done by optimizing the parameters $\boldsymbol{\theta}$ of the neural network function $f(\boldsymbol{x}; \boldsymbol{\theta})$ that result in the best approximation of the true function. A question that arises is that why do we even need these deep neural networks particularly? This is because of their unique ability to be **Universal Function Approximators** Hornik, Stinchcombe, and White 1989.



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

Figure 4.2: Activation functions commonly used in practice.

It has been proven in prior literature that even with a single hidden layer, given enough nodes and the right set of parameters $\boldsymbol{\theta}$, we can potentially model any function using MLPs. In other words, given enough number of nodes stacked on top of each other along with non-linear transformations, a neural network can potentially model any function possible. It is important to note here that the universal approximation theory holds true only in the presence of non-linear transformations and not the affine transforms alone. This is because it is impossible to have all data distributions, sampled from reality, to be fit with only a linear parameterization of the given distribution. A few of the commonly used **activation functions** have been illustrated in Fig. 4.2.

Formally, the output of an MLP is calculated as follows:

$$\begin{aligned} \boldsymbol{H} &= g_1(\boldsymbol{X}\boldsymbol{W}^{(1)} + \boldsymbol{b}^{(1)}) \\ \boldsymbol{O} &= g_2(\boldsymbol{H}\boldsymbol{W}^{(2)} + \boldsymbol{b}^{(2)}) \end{aligned} \tag{4.1}$$

where $\boldsymbol{X} \in \mathbb{R}^{Nxd}$ represents the dataset with N samples and d features, $\boldsymbol{H} \in \mathbb{R}^{Nxh}$ represents the outputs of the hidden layer with h hidden units or dimensionality of the hidden layer output, $\boldsymbol{W}^{(1)} \in \mathbb{R}^{dxh}$ and $\boldsymbol{b}^{(1)} \in \mathbb{R}^{1xh}$ being the weights and biases of the first layer, $\boldsymbol{W}^{(2)} \in \mathbb{R}^{hxq}$ and $\boldsymbol{b}^{(2)} \in \mathbb{R}^{1xq}$ being the weights and biases of the second layer, $\boldsymbol{O} \in \mathbb{R}^{nxq}$ being the output of the MLP with dimensionality q and g_1, g_2 being non-linear activation functions. To build more general and computationally heavy MLPs that posses greater capacity for approximation, more hidden layers can be stacked on top of each other instead of just one, as is illustrated in Equation (4.1) and Fig. 4.1.

4.1.1. Stochastic Gradient Descent

Until now, objective functions have been derived using multiple perspectives of distribution modelling and function approximators have also been discussed for optimal modeling of these distributions. However, all the estimators of network parameters have to be optimized for minimizing the negative log-likelihoods of multiple tasks. To optimize for the minimum loss with respect to the network parameters, **Gradient Descent** and its variants are the most used methods in practice. As the name suggests, gradient descent is a gradient based optimization algorithm that iteratively operates on convex functions to find the optimal parameters. Suppose that the function to be optimized is $y = f(x)$. The derivative of this function is given by $f'(x) = \frac{dy}{dx}$, which gives the slope of the function (for 1 dimensional domain and co-domain) $f(x)$ at the given point x. The slope also indicates the change in y, if a change is made in x. This is a useful property because now, with this derivative information, it is possible to indicate the direction of movement for the parameter in order to "settle" in a local minima where the derivative $f'(x) = 0$. This is illustrated in Fig. 4.3. Finally, the gradient descent procedure involves taking a step in the opposite direction of the gradient in order to settle in a minima:

$$\boldsymbol{x}' = \boldsymbol{x} - \epsilon \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) \tag{4.2}$$

where ϵ is the size of the step taken in each iteration.

As far as deep learning is concerned, the parameter space consists of thousands of dimensions

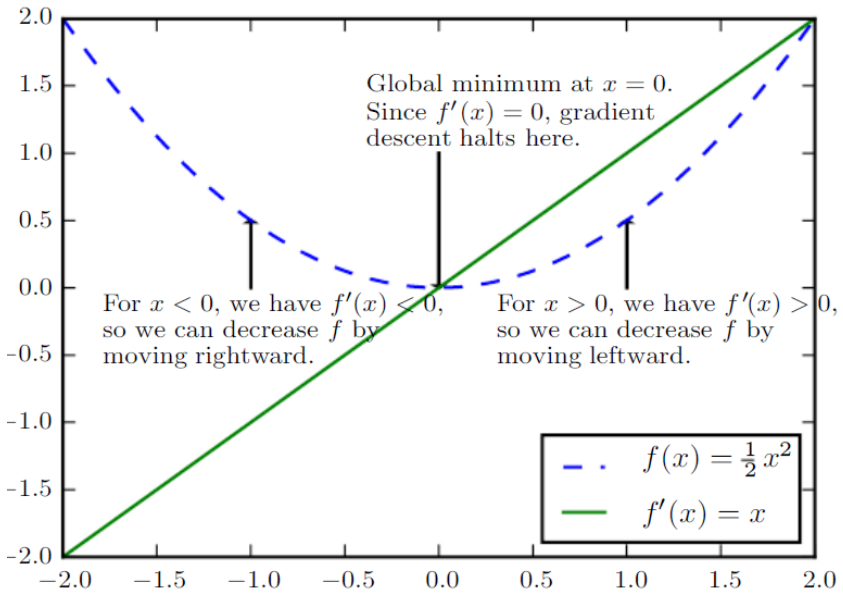


Figure 4.3: Gradient descent algorithm uses the derivative of the cost function to find the optimal direction of movement for the parameters. Figure courtesy Goodfellow, Bengio, and Courville (2016)

and thus the loss functions contain many more local minima and saddle points than a global minimum. Therefore, it is common to settle for finding local minima that has a value very close to the actual global minimum. Moreover, deep learning involves using large training datasets for good generalization, but large datasets are also much more computationally difficult to process. In order to run the gradient descent algorithm on the entire dataset, a computation cost of the order $\mathcal{O}(N)$ is required:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\boldsymbol{\theta}} L\left(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}\right) \tag{4.3}$$

This operation becomes computationally intractable for training datasets containing billions of samples. Thus, **stochastic gradient descent** is used instead of gradient descent where, the gradient is computed for only a small mini-batch containing a few $m << N$ samples. This works since the gradient is an expectation of the derivative of the loss function, over the distribution of the dataset. This expectation can be computed with a much smaller number of samples, as opposed to using the entire dataset. The expectation is computed as follows:

$$\boldsymbol{g} = \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m} L\left(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}\right) \tag{4.4}$$

Since the algorithm now operates on only m instead of N samples, multiple random sampling passes are carried out where m random samples are drawn from the entire training dataset. Finally, steps are taken in the opposite direction of the gradient in the parameter space, until convergence. This is represented as:

$$\theta \leftarrow \theta - \epsilon g \tag{4.5}$$

4.1.2. Backpropagation

The stochastic gradient descent procedure described in the previous subsection requires to compute the derivative of the loss function with respect to all the function parameters. When using a deep neural network, the derivative must be computed with respect all the thousands of parameters. To do so, an efficient method is developed called backpropagation which involves computing the gradients of outputs with respect to inputs and parameters, starting from the last output of the network which is

the loss function value, with respect to input, which is the output of the neural network function. This computation is carried out recursively till the computation reaches the input layer of the neural network and thus all the necessary derivatives have been calculated. This procedure, in other words, is also called the **chain rule of derivatives** in calculus. Finally, all the derivatives are collected in a vector of gradients as follows:

$$\nabla_{\boldsymbol{\theta}} L(f(\boldsymbol{x};\boldsymbol{\theta}),y) = \left[\frac{\partial L}{\partial w_h}, \frac{\partial L}{\partial b_h}, \frac{\partial L}{\partial w_{h-1}}, \frac{\partial L}{\partial b_{h-1}}, \ldots \frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial b_1}\right]^\top \tag{4.6}$$

where h is the total number of layers in the neural network.

4.2. Convolutional Neural Networks

Convolutional neural networks (CNNs) are a special class of neural networks that are designed particularly to process data that has a grid-like topology. This makes it a natural fit for operating on image data ($2D$ grids) or even time-series. CNNs are ubiquitous in the realm of computer vision because of their powerful and efficient ability to process an image's rich spatial structure. MLPs are fully-connected networks that would operate on the image's grid of pixels by flattening it first and then treating each pixel as a separate dimension. This is massively inefficient because of two reasons:

- An image grid comprises of a large number of pixels in totality, generally anywhere between 84×84 to 1024×1024, or even more, depending on the resolution of the image. Operating on a massive number of dimensions requires that many more parameters, one for each in the first layer of the MLP, and then lesser or more, depending on the network architecture. This makes the optimization procedure computationally costly and inefficient.

- By operating on each pixel separately, the MLP disregards the rich spatial structure of the image. This spatial structure is of great importance since it forms the essence of what an image depicts.

Convolutional neural networks are neural networks that use the convolution operation instead of standard matrix multiplication in at least one of their layers, as opposed to MLPs that simply use matrix multiplication. (Goodfellow, Bengio, and Courville 2016)

To start with CNNs, it is apt to first understand what the convolution operation is. The convolution function is computed between two real valued functions $x, w : \mathbb{R}^d \to \mathbb{R}$ and is defined as:

$$s(t) = (x * w)(t) = \int x(a)w(t-a)da \tag{4.7}$$

In other words, the convolution operation measures the overlap between x and w when one function is "flipped" and shifted by t. The first argument is known as the **input** (in this case x) and the second one, the **kernel** (in this case w). The output is known as the **feature map**. For discrete functions, the integral becomes a sum and thus the convolution operation is defined as:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{4.8}$$

These definitions of convolution are for the single-dimensional case. However, as discussed earlier, in deep learning applications, the use is for multi-dimensional data such as images which are 2-dimensional. In these cases, the convolution is defined over multiple axes as:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n) \tag{4.9}$$

where I and K are two-dimensional input and kernel, respectively. The second line in Equation (4.9) indicates that the convolution operation is commutative. What is common in practice is to use the **cross-correlation** function, which is the same as convolution, but without the flipping of the kernel:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n). \tag{4.10}$$

Deep learning libraries such as PyTorch (Paszke et al. 2019) actually implement **cross-correlation** but call it convolution. This is what has been followed in the rest of the text too. A visual representation

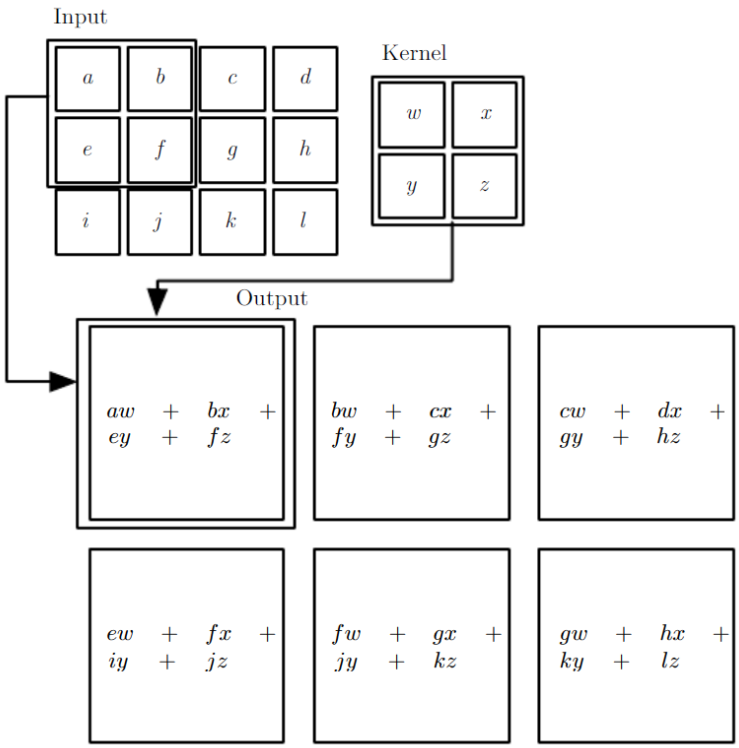


Figure 4.4: Convolution operation (cross-correlation without flipping). Figure courtesy Goodfellow, Bengio, and Courville (2016)

of this convolution operation applied to a $2D$ grid (can be interpreted as an image) has been illustrated in Fig. 4.4. CNNs not only use one such kernel, but multiple of these with different weight values to extract a large variety of features at every step/layer of the convolutional neural network. The number of kernels in a convolutional operation is referred to as the number of **channels** in a convolutional operation.

4.2.1. Why Convolutions?

As mentioned earlier, the unique ability of convolutions to incorporate the structural information of an image in its processing helps improve a deep learning algorithm since it encodes the prior knowledge that nearby pixels in an image are similar and thus need similar processing, rather than having separate weights for them. This property of CNNs is called **Parameter Sharing**.

In standard MLPs, all elements of a weight layer are used once when computing the output of a layer and is multiplied by one element of the input once only. But in CNNs, each value in a kernel is is used at every position of the input grid (except boundary values in some cases where the input grid is not padded). This means that only one set of parameters (all values in a kernel) are learnt for CNNs, rather than learning a different weight value for each and every location as is the case with MLPs (Fig. 4.5).

Parameter sharing also makes CNNs **equivariant** to translation. This means that the output changes in exactly the same way the input changes (Fig. 4.6). This is particularly relevant for processing images since if an object in the image changes its location, then the feature map representation of the image also changes accordingly. This helps design particular convolutions that extract specific types of features, since they function in the same way throughout the space of the image and extract those features from all parts of the image.

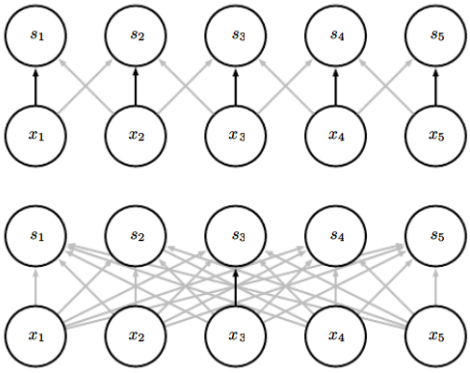


Figure 4.5: Top: Black arrow indicates use of the central weight in 3-element kernel in a CNN. Bottom: Black arrow indicates the use of a weight matrix element in MLP. Figure courtesy Goodfellow, Bengio, and Courville (2016)

$$[100,\ 101,\ 99,\ 200,\ 199,\ 201] \star [-1,\ +1] = [1,\ -2,\ 101,\ -1,\ 2]$$
$$[101,\ 99,\ 200,\ 199,\ 201,\ 201] \star [-1,\ +1] = [-2,\ 101,\ -1,\ 2,\ 0]$$

Figure 4.6: An example indicating the same leftward shift in output as the input.

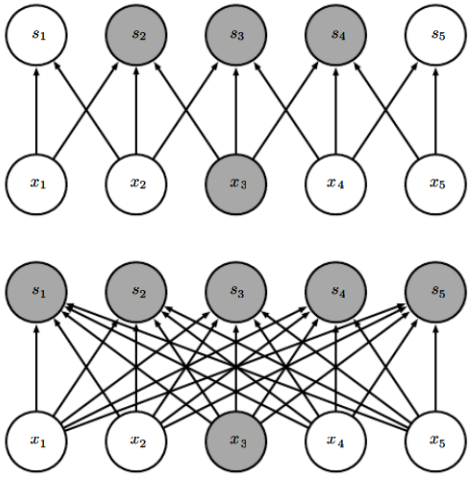


Figure 4.7: Highlighting the output units s that are influenced by the input unit x in the case of a 3-element kernel convolution (top) and fully connected matrix multiplication (bottom). Figure courtesy Goodfellow, Bengio, and Courville (2016)

Finally, CNNs typically have **sparse connections** or sparse weights instead of fully connected weights. This is because the kernel's size is typically set to be much smaller than the image's size. This is done so as to process an image and extract local features from small neighbourhood of pixels throughout the image, with the help of specialized convolutions. For example, a kernel can be designed to detect edges and contours in an image that only occupy a few hundreds of pixels. This is much more efficient than processing the image entirely all at once using millions of pixel values. This also makes CNNs less computationally expensive and memory light (Fig. 4.7).

Although convolution is equivariant to translation, it is not equivariant to other transformations such as scale or rotation. In order to handle this, **Pooling** is used after a convolutional layer in standard CNNs. Pooling operates on small neighbourhoods (just like a kernel) of an image and returns one output for the neighbour. One such pooling operation is **average pooling** that returns the average of the input neighbourhood of pixels. Pooling helps making a CNN approximately invariant to small

translations. Invariance is useful when we care about detecting certain features in the image grid rather than actually locating their exact position in the final feature map.

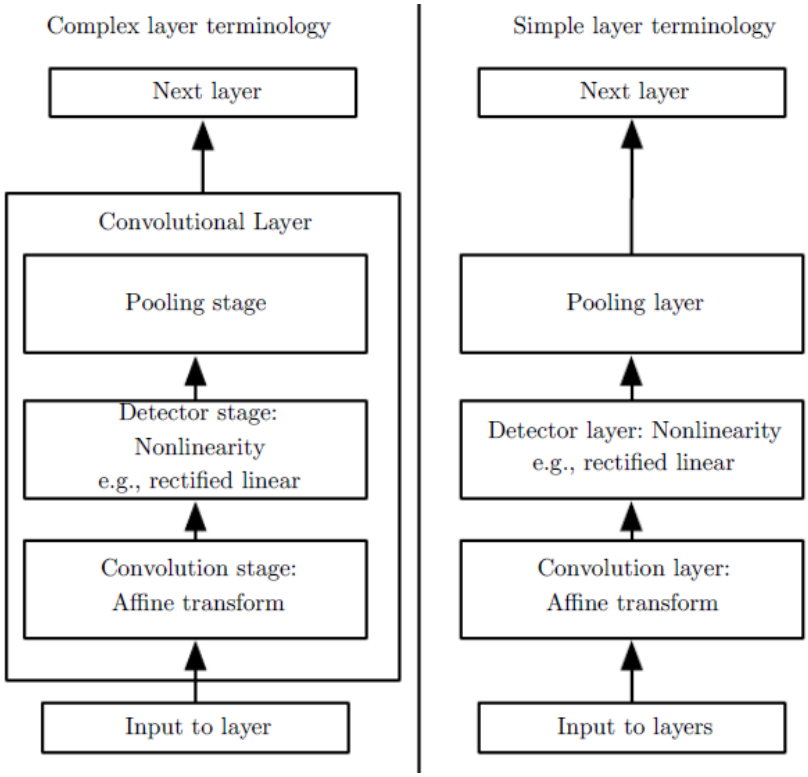


Figure 4.8: Typical convolutional block in a CNN comprises of a convolution operation, followed by a non-linearity induced by an activation function and then finally a pooling operation. Figure courtesy Goodfellow, Bengio, and Courville (2016).

A **Convolution block** in a CNN generally comprises of the components depicted in Fig. 4.8. Multiple of these convolutional blocks are stacked layer after layer, with different kernel and pooling kernel sizes to extract meaningful feature maps from input images to be used for the downstream task. One of the first such CNNs was developed by Yann LeCun, called the LeNet (LeCun et al. 1998), as

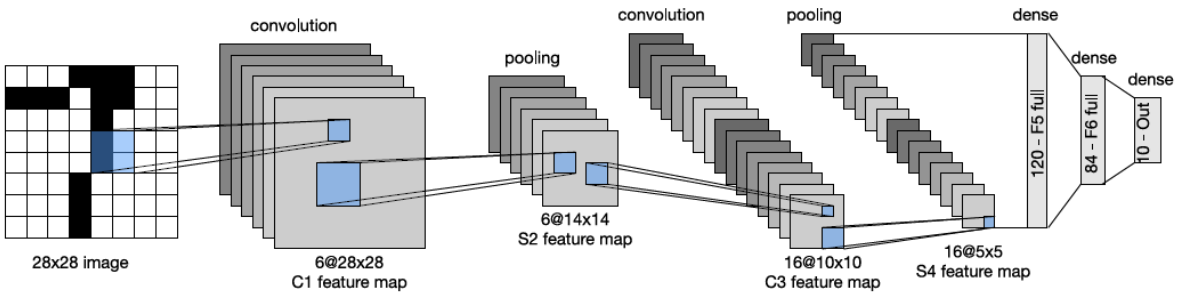


Figure 4.9: Flow of data in the LeNet. The input image shows a handwritten digit '7' and output is the probability over 10 possible classes. Figure courtesy A. Zhang et al. (2021).

shown in Fig. 4.9. An example of feature maps that can be extracted from CNNs has been illustrated in Fig. 4.10.

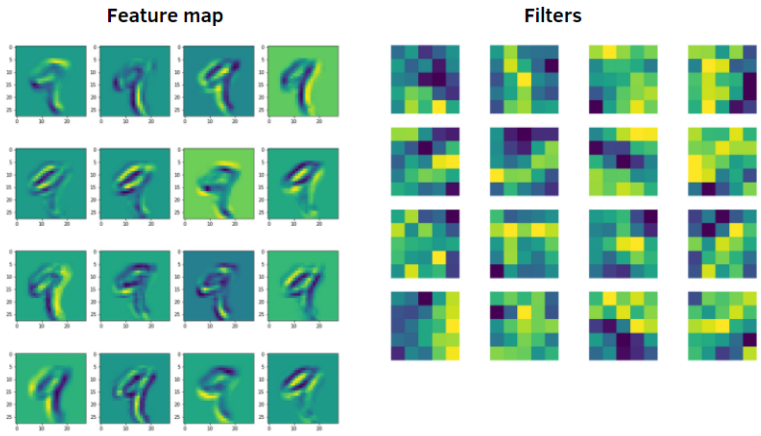


Figure 4.10: Feature maps (left) extracted from a CNN with 4 convolutional blocks with the learnt convolutional kernels (right). The rows indicate 4 experiments run in parallel for the same handwritten digit classification task, with columns indicating the different feature maps extracted in each experiment.

4.3. Attention

The optical nerve of a human's visual system receives massive amounts of sensory input, far more than what it can process. Thankfully for us, we posses the ability to only *attend* to certain stimuli and let the other slide. Focus and concentration grant humans the ability to consciously divert their attention to certain matters of interest, which has had a great part to play in the natural selection of species too. Inspired by this biological ability of human beings to attend to relevant stimuli, **Attention** mechanisms are built mathematically that power *Transformers*, responsible for creating large language models such as GPT-3 (Brown et al. 2020).

The idea of attention dates back to 1964, when Nadarya and Watson developed the Nadarya-Watson kernel regression (Nadaraya 1964; Watson 1964). In this version of regression, the output that must be predicted $y = f(x)$ is the normalized, kernel-weighted sum of the input locations:

$$f(x) = \sum_{i=1}^{n} \frac{K\left(x - x_i\right)}{\sum_{j=1}^{n} K\left(x - x_j\right)} y_i \tag{4.11}$$

where n equals the total number of observed/training data points and K is the kernel that is used. This can be re-written from the perspective of attention:

$$f(x) = \sum_{i=1}^{n} \alpha\left(x, x_i\right) y_i \tag{4.12}$$

where x is the query and (x_i, y_i) are the key-value pairs. Notice that the weight that each y_i contributes to the final output value $y = f(x)$ is decided by the attention weight $\alpha(x, x_i)$ in Equation (4.12). The attention weights over all possible keys for a given query forms a probability distribution due to their normalization and non-negative values. The kernel can be chosen to be any relevant kernel such as the Gaussian or Softmax kernel, depending on the downstream application.

A very similar version of this attention has been proposed in Vaswani et al. 2017, which is also known as the **scaled dot-product attention**. As the name suggests, the attention weighting function is a simple dot-product that is scaled by the square root of the dimensionality of the query/key representations. These representations are generally extracted by separate neural networks for query, key and values. For a mini-batch of n queries and m key-value pairs, where the query mini-batch matrix $\boldsymbol{Q}$ has a dimensionality of $\mathbb{R}^{n \times d}$, value matrix $\boldsymbol{K}$ has a dimensionality of $\mathbb{R}^{m \times d}$ and the value mini-batch

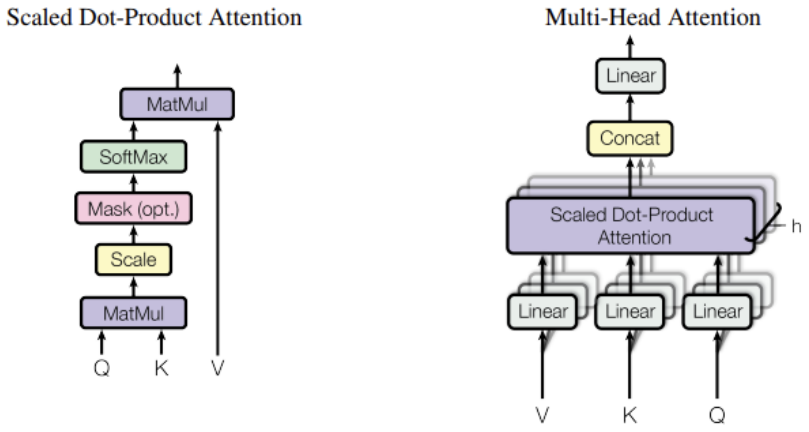


Figure 4.11: Illustration of Single and Multi-head attention as proposed in Vaswani et al. (2017).

matrix $\boldsymbol{V}$ has a dimensionality of $\mathbb{R}^{m\times v}$, the scaled dot-product attention can be computed as follows:

$$\text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\mathsf{T}}}{\sqrt{d}}\right)\mathbf{V} \in \mathbb{R}^{n\times v}. \tag{4.13}$$

This attention mechanism is now widely used in almost all attention implementations due to its impressive efficacy and widespread generalizability across multiple downstream tasks. This version of attention is used in most *Transformers*, Devlin et al. 2018; Yinhan Liu et al. 2019; Brown et al. 2020 being a few notable examples. There is also a possibility to have multiple different versions of the query, key and value representations for each data point. This helps the network extract multiple representations for the same data points that may be relevant in different contexts for the attention function to then operate on. In other words, multiple representations of self attention allows the network to jointly attend to information from different representation subspaces at different positions. This is known as **Multi-headed self-attention**, and has been depicted in Fig. 4.11.

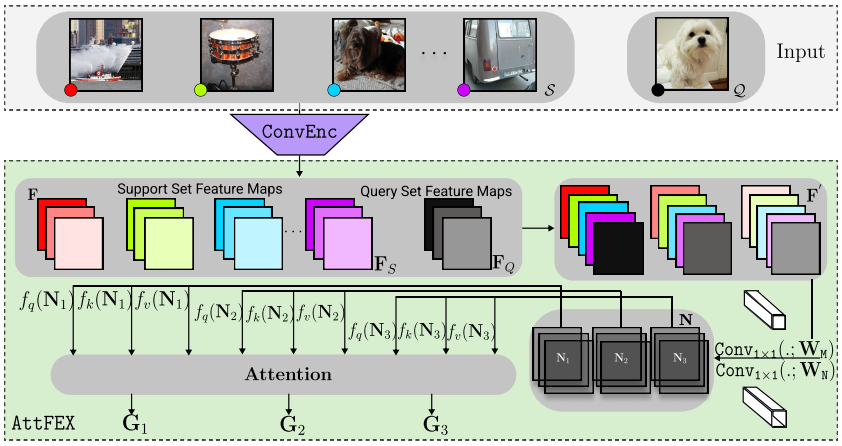


Figure 4.12: Illustration of the AttFEX operating point-wise on pixel distributions across corresponding feature maps. The self attention module then fuses information extracted across the various feature maps.

TRIDENT utilizes a special version of convolutional neural networks in its transductive feature-extraction module AttFEX (Fig. 4.12). These networks have a kernel size of 1×1, and are thus called Conv1 $\times$ 1. This specific class of convolutions is used in the AttFEX module since it is meant to only see through and compare the feature maps (and not images) across all images in a task. The feature maps already contain local spatial features from images since they are extracted using larger (3×3) convolutional kernels. The AttFEX module now only operates in a point-wise manner on pixel distributions of these information rich feature maps since all of these have been extracted using identical

sets of kernels. Thus, operating on identically processed corresponding pixels of feature maps is much more efficient and appropriate than using a larger convolutional kernel again to pool neighbouring pixel information. AttFEX makes use of a scaled dot-product attention mechanism (Vaswani et al. 2017) as well, to then fuse information extracted across the various feature maps in the previous step. Finally, we end up with masks that contain task-relevant class-discriminating information which is multiplied with the original feature maps to render them task-specific.

5

Variational Inference

All fields of science rely on hypothesizing theories about the underlying processes of nature and then testing these theories by means of experimentation. Hypothesizing theories is equivalent to assuming a structural causal model for the process in question, as discussed in detail in Chapter 3, Section 3.5. The graphical model explains causal relationships between all the variables. Effectively, the aim is to learn a joint distribution over all the variables in the SCM and then testing them against observations collected from reality. This perspective is known as *generative modeling*, as opposed to *discriminative modeling*, where the aim is to only learn a predictor given the data (Diederik P Kingma, Welling, et al. 2019).

Generative modeling provides an intuitive and interpretable model of the assumptions made about the underlying laws and constraints of the generative process. These are reflected in the variables and their relationships, while other information that is considered to be irrelevant is discarded as noise. The inferred joint distribution of the variables along with the graphical model naturally expresses the causal structure which in turn generalizes much better to unseen data distributions/ situations when compared to correlations. This is because of the fact that correlation between two variables is an insufficient metric to conclude any cause and effect relationship between them. The correlation could be completely coincidental or maybe influenced by some other confounding latent factor. Moreover, a generative model can also be used as a discriminative one by plugging in the inferred joint probabilities into the Bayes rule (Equation 3.15).

Discriminative models optimize for a mapping between inputs and future predictions. A generative model, however, first identifies the future output and then understands the process behind it by inferring distributions of all the variables involved. While this characteristic of generative models is useful for inferring and discovering important traits and patterns from data, it also is disadvantageous in the sense that it makes stronger assumptions about the data. This leads to higher asymptotic bias (Banerjee 2007) in generative models, as compared to their discriminative counterparts. Thus, when the causal assumptions about the data are incorrect (which to some degree they are in almost all cases), and there is sufficient amount of data available at one's disposal, a discriminative model would typically give fewer errors in solely discriminative tasks. However, depending on the amount of data and labels available, and the nature of the downstream task, it maybe worthwhile to study and make assumptions about the generative process (Durk P Kingma et al. 2014; Sohn, H. Lee, and Yan 2015; Sønderby et al. 2016).

5.1. Why Approximate Inference?

When designing an SCM for a process, it is not necessary to always only have fully-observed variables. In other words, it is possible that one may need/want to include certain factors of variation/ random variables in the directed graphical model that are not a part of the dataset. These unobserved variables are called **latent variables** that are a part of the model, but not of the dataset. Usually, in practice, and

throughout this text (from now on), z is used to denote such variables. In order to infer a joint probability distribution over the variables, one needs to consider the Bayesian perspective for statistical modeling 3.2. However, this requires to compute some form of the marginal likelihood of observed data, which in most cases, is analytically intractable 3.4. For example, in a simple latent variable model where $z \to x$, the computation of the exact posterior distribution $p_\theta(z \mid x) = \frac{p_\theta(z,x)}{p_\theta(x)}$ is intractable due to the intractability of the marginal $p_\theta(x) = \int p_\theta(x,z)dz$. Considering an example regarding deep learning models where the model parameters $\boldsymbol{\theta}$ are the latent variables, and the exact posterior distribution $p(\boldsymbol{\theta} \mid \mathcal{D})$ must be computed. This involves computing the marginal likelihood of the dataset $\mathcal{D}$ by evaluating an integral defined over the extremely high-dimensional space of the model parameters $\boldsymbol{\theta}$. Computing this integral w.r.t. millions of neural network parameters is analytically intractable. MAP estimation is a workaround to this intractability by estimating a prior regularized point estimate, however it avoids the computation of the distribution completely 3.2. Thus, it also fails to provide any measure of uncertainty about the inferred variables. This motivates the need to resort to **Approximate Inference** techniques.

5.2. Variational Free Energy and Evidence Lower Bound

Variational inference is an approximate inference technique that has gained attention due to its relatively low computational cost and good generalizational capabilities. It allows one to infer probability distributions over variables by turning an intractable inference problem into a tractable optimization problem.

Consider a generative model with latent variables $\boldsymbol{z}$ and observed variables $\boldsymbol{x}$ while the fixed model parameters for the generative distribution are $\boldsymbol{\theta}$. Since computing the posterior distribution $p_{\boldsymbol{\theta}}(\boldsymbol{z} \mid \boldsymbol{x})$ is intractable, an *approximation* to this posterior $q(\boldsymbol{z})$ is assumed. Since goal is to have an approximate posterior that is as close as possible to the true posterior, the following objective/loss is minimized:

$$q = \underset{q \in \mathcal{Q}}{\operatorname{argmin}} D_{\boldsymbol{KL}}\left(q(\boldsymbol{z}) \| p_{\boldsymbol{\theta}}(\boldsymbol{z} \mid \boldsymbol{x})\right) \tag{5.1}$$

In order to optimize the approximate posterior over a family of distributions $\mathcal{Q}$, the distributions are parameterized using the **variational parameters** $\boldsymbol{\psi}$. Now, the optimization problem is shifted from minimizing with respect to q, to $\boldsymbol{\psi}$. The best variational parameters are given by:

$$\begin{aligned}
\psi^* &= \underset{\psi}{\operatorname{argmin}} D_{\mathbb{KL}}\left(q(\boldsymbol{z} \mid \psi) \| p_{\boldsymbol{\theta}}(\boldsymbol{z} \mid \boldsymbol{x})\right) \\
&= \underset{\psi}{\operatorname{argmin}} \mathbb{E}_{q(\boldsymbol{z}|\psi)}\left[\log q(\boldsymbol{z} \mid \psi) - \log\left(\frac{p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid z) p_{\boldsymbol{\theta}}(\boldsymbol{z})}{p_{\boldsymbol{\theta}}(\boldsymbol{x})}\right)\right] \\
&= \underset{\psi}{\operatorname{argmin}} \underbrace{\mathbb{E}_{q(\boldsymbol{z}|\psi)}\left[\log q(\boldsymbol{z} \mid \psi) - \log p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z}) - \log p_{\boldsymbol{\theta}}(\boldsymbol{z})\right]}_{\mathcal{L}(\psi \mid \boldsymbol{\theta}, \boldsymbol{x})} + \log p_{\boldsymbol{\theta}}(\boldsymbol{x})
\end{aligned} \tag{5.2}$$

Note that the variational parameters are optimized for a given $\boldsymbol{x}$. The last term $\log p_{\boldsymbol{\theta}}(\boldsymbol{x})$ is independent of $\boldsymbol{\psi}$ and thus can be dropped from the optimization problem. The remaining loss function can now be reduced to:

$$\mathcal{L}(\psi \mid \boldsymbol{\theta}, \boldsymbol{x}) = D_{\boldsymbol{KL}}\left(q(\boldsymbol{z} \mid \psi) \| p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})\right) \triangleq \mathbb{E}_{q(\boldsymbol{z}|\psi)}\left[\log q(\boldsymbol{z} \mid \psi) - \log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})\right] \tag{5.3}$$

Analogous to the Helmholtz Free Energy of a thermodynamic system, this loss function represents the **Variational Free Energy** (VFE), where:

$$\mathcal{L}(\psi \mid \boldsymbol{\theta}, \boldsymbol{x}) = \mathbb{E}_{q(\boldsymbol{z}|\psi)}[\mathcal{E}(\boldsymbol{z})] - \mathbb{H}(q) \tag{5.4}$$

and $\mathcal{E}(\boldsymbol{z})$ represents the expected energy and $\mathbb{H}(q)$ is the entropy of the approximate latent distribution. The VFE is an upper bound on the free energy $-\log p_{\boldsymbol{\theta}}(\boldsymbol{x})$, since the KL divergence between any two quantities is greater than zero:

$$D_{\mathbb{K}}(q \| p) = \mathrm{VFE}(\psi \mid \boldsymbol{\theta}, \boldsymbol{x}) + \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) \geq 0 \tag{5.5}$$

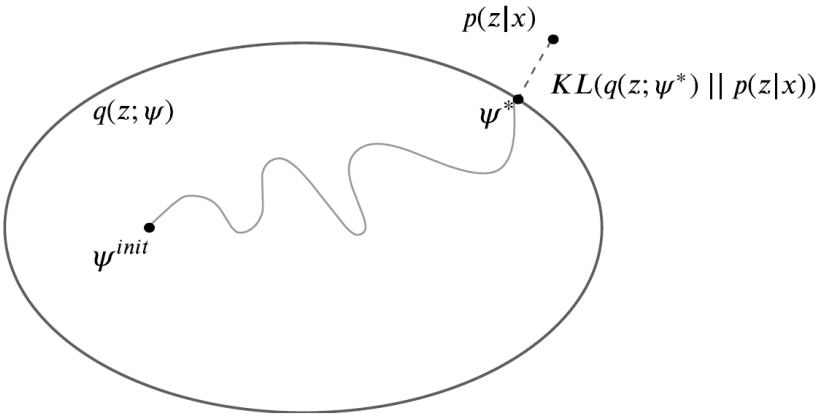


Figure 5.1: The oval represents the set of all variational distributions parameterized by $\boldsymbol{\psi} \in \mathbb{R}^k$. The aim is to find a point inside the oval that parameterizes a $q(\boldsymbol{z} \mid \boldsymbol{\psi})$ that is closest to the true posterior $p_{\boldsymbol{\theta}}(\boldsymbol{z} \mid \boldsymbol{x})$. Figure courtesy Murphy (2022).

By minimizing the VFE, the free energy (NLL of the data) is minimized, which in turn gives the best variational parameters. This eventually results in the best approximate posterior $q(\boldsymbol{z} \mid \boldsymbol{\psi})$, thus turning the inference problem into an optimization problem.

The negative of the VFE can be interpreted as the **evidence lower bound** or **ELBO** since:

$$L(\psi \mid \boldsymbol{\theta}, \boldsymbol{x}) \triangleq \mathbb{E}_{q(\boldsymbol{z}|\psi)} \left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}) - \log q(\boldsymbol{z} \mid \psi)\right] \leq \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) \tag{5.6}$$

where the log-likelihood of the data is called the “evidence". The ELBO can be rewritten as:

$$L(\psi \mid \boldsymbol{\theta}, \boldsymbol{x}) = \mathbb{E}_{q(\boldsymbol{z}|\psi)} \left[\log p_{\boldsymbol{\theta}}(x \mid \boldsymbol{z})\right] - D_{\mathbb{KL}} \left(q(\boldsymbol{z} \mid \psi) \| p_{\boldsymbol{\theta}}(\boldsymbol{z})\right) \tag{5.7}$$

where, by maximizing the ELBO, the log-likelihood of the data is increased. This is done by increasing the expected log-likelihood of the data conditional on the latent variable, while reducing the KL divergence between posterior and prior as a regularization term, as depicted in Fig. 5.1 .

5.3. Amortization of Inference

In the previous section, it was stated that the VFE and ELBO are only optimized for a given $\boldsymbol{x}$. This means that the variational parameters $\boldsymbol{\psi}$ have to be initialized and then the optimization must be repeated separately for each and every data point $\boldsymbol{x}$ in the dataset $\mathcal{D}$. Solving an optimization problem for each $q(\boldsymbol{z}_i \mid \boldsymbol{\psi}_i) \forall i \in [1, N]$ is slow and computationally expensive, especially in the context of deep learning where datasets typically contain a few million samples. To avoid this, the inference can be *amortized* by training a model with parameters $\boldsymbol{\phi}$ to predict the per-sample variational parameters $\boldsymbol{\psi}_i$ from the data point $\boldsymbol{x}_i$. In this way, the cost of finding per-sample $\boldsymbol{\psi}_i$ is reduced to finding a common $\boldsymbol{\phi}$ for the entire dataset. The model with parameters $\boldsymbol{\phi}$ is called the **inference network** $\boldsymbol{\psi}_i = f_{\boldsymbol{\phi}}^{inf}(\boldsymbol{x}_i)$. This effectively changes the approximate posterior as follows:

$$q\left(\boldsymbol{z}_i \mid \boldsymbol{\psi}_i\right) = q\left(\boldsymbol{z}_i \mid f_{\boldsymbol{\phi}}^{\boldsymbol{inf}}\left(\boldsymbol{x}_i\right)\right) = q_{\boldsymbol{\phi}}\left(\boldsymbol{z}_i \mid x_i\right) \tag{5.8}$$

To find the **amortized ELBO** for the entire dataset, the ELBO for each data point (or log-likelihood) can be summed since it is the lower bound of the log-likelihood of each data point (Equation (5.6)). The dataset ELBO thus becomes:

$$L(\boldsymbol{\phi}, \boldsymbol{\theta} \mid \mathcal{D}) = \frac{1}{N} \sum_{i=1}^{N} \left[\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}_i|\boldsymbol{x}_i)} \left[\log p_{\boldsymbol{\theta}}\left(\boldsymbol{x}_i, \boldsymbol{z}_i\right) - \log q_{\phi}\left(\boldsymbol{z}_i \mid \boldsymbol{x}_i\right)\right]\right] \tag{5.9}$$

The inner expectation over a single sample latent posterior can be approximated by sampling a $\boldsymbol{z}_i$ from the inferred distribution $q_{\phi}\left(\boldsymbol{z}_i \mid \boldsymbol{x}_i\right)$. Amortized inference is widely used in latent variable models and

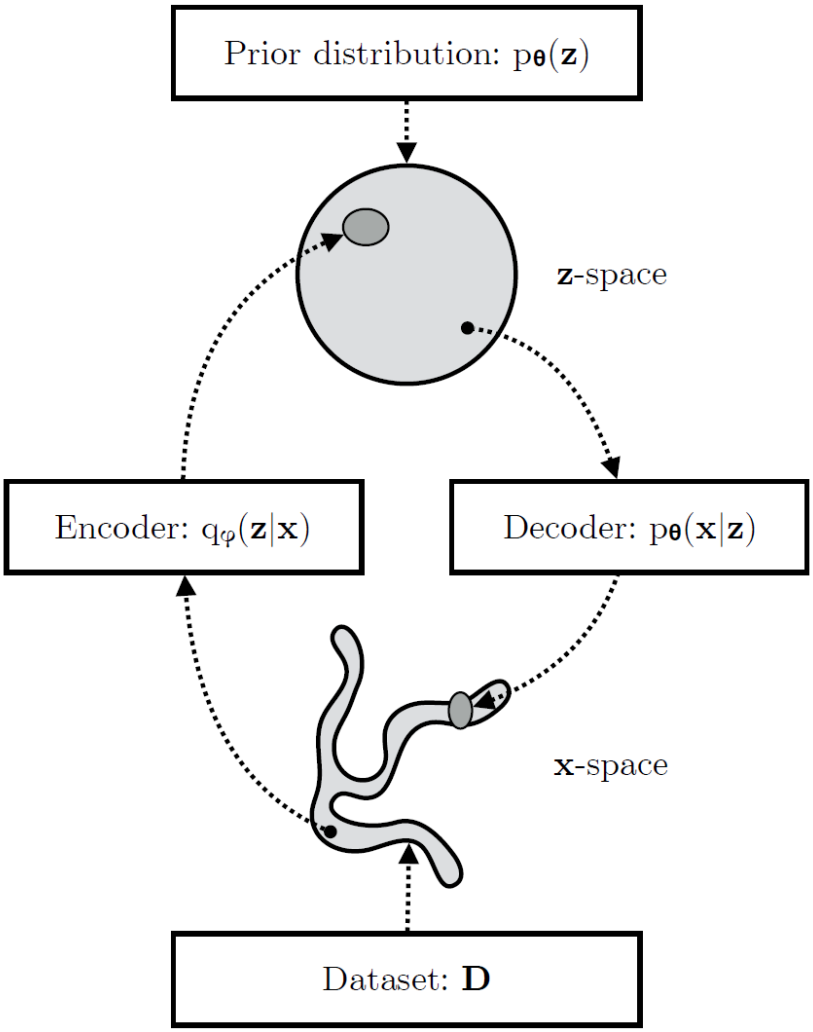


Figure 5.2: VAEs learn a neural network that acts as a stochastic mapping between the observed space of $\boldsymbol{x}$ and the latent space of $\boldsymbol{z}$. Typically the observed space has a rather complicated empirical distribution $q_{\mathcal{D}}(\boldsymbol{x})$ while the latent space has a relatively simpler distribution (depicted as a sphere). Figure courtesy Diederik P Kingma, Welling, et al. (2019).

probabilistic programming, however, this may result in sub-optimal $\boldsymbol{\psi}_i$'s since an implicit assumption is made about the relationship between the data point $\boldsymbol{x}_i$ and the variational parameters $\boldsymbol{\psi}_i$ through the inference network.

5.4. Variational Auto-Encoders

Variational autoencoders (VAEs) (Diederik P Kingma and Welling 2014) are a class of deep learning models that are built to variationally infer latent variables in a computationally efficient way using amortized inference and stochastic gradient descent. More specifically, VAEs use variational inference to compute the intractable posterior over latent variables by assuming an approximate posterior of the form $q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) \approx q_{\boldsymbol{\theta}}(\boldsymbol{z} \mid \boldsymbol{x})$. The approximate posterior $q(\cdot)$ is parameterized using a neural network $\boldsymbol{\phi}$, thus effectively *amortizing* the cost of computing a per-sample variational parameter $\boldsymbol{\psi}_i$. The optimization objective of the VAE is the *evidence lower bound (ELBO)*, just like any other variational method.

$$\begin{aligned} L_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x}) &= \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) - D_{KL}\left(q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x}) \| p_{\boldsymbol{\theta}}(\boldsymbol{z} \mid \boldsymbol{x})\right) \\ &\leq \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) \end{aligned} \tag{5.10}$$

Using the derivations from Equation (5.6), (5.7), a relationship can be established between the KL

divergence of the approximate and true posterior. Obviously, the KL divergence gives a measure of the amount of information (negative entropy) lost when the true posterior is replaced by the approximate posterior. Thus, maximizing the ELBO is equivalent to maximizing the log-likelihood of the data, which in turn pushes the KL divergence to be as close to zero. This also means that the KL divergence measures the gap between the ELBO $L(\boldsymbol{\phi}, \boldsymbol{\theta} \mid \boldsymbol{x})$ and the marginal likelihood $\log p_{\boldsymbol{\theta}}(\boldsymbol{x})$. Thus, the better the approximation of the approximate posterior, the better the log-likelihood of the data. Reframing the ELBO objective a little reflects the overall structure of the VAE much better, and can be given as:

$$\begin{aligned} L_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x}) &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z}) - \log q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})\right] \\ L_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x}) &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z}) + \log p_{\boldsymbol{\theta}}(\boldsymbol{z}) - \log q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})\right] \\ L_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x}) &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})\right] - D_{KL}\left(q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x}) \| p_{\boldsymbol{\theta}}(\boldsymbol{z})\right) \end{aligned} \tag{5.11}$$

where $q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})$ forms the inference network and $p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})$ forms the generative network, that reconstructs $\boldsymbol{x}$ using $\boldsymbol{z}$ as input and neural network parameters $\boldsymbol{\theta}$. A more intuitive explanation of the functioning of a VAE has been depicted in Fig. 5.2.

5.4.1. Stochastic Gradient Descent Optimization of the ELBO

By having the variational and fixed parameters $(\boldsymbol{\phi}, \boldsymbol{\theta})$ in the same network architecture, the VAE jointly optimizes both the sub-networks using stochastic gradient descent (Section 4.1.1). Assuming that the data has been independently sampled from identical distributions (i.i.d.), the ELBO for the entire dataset can be written as:

$$L_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathcal{D}) = \sum_{\boldsymbol{x}\in\mathcal{D}} \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x}) \tag{5.12}$$

For SGD, the gradients of the ELBO must be computed w.r.t. the inference and generative parameters. The generative parameter gradients are calculated as follows:

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} L_{\boldsymbol{\theta},\phi}(\boldsymbol{x}) &= \nabla_{\boldsymbol{\theta}} \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z}) - \log q_{\phi}(\boldsymbol{z} \mid \boldsymbol{x})\right] \\ &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\nabla_{\boldsymbol{\theta}}\left(\log p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z}) - \log q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})\right)\right] \\ &\simeq \nabla_{\boldsymbol{\theta}}\left(\log p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z}) - \log q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})\right) \\ &= \nabla_{\boldsymbol{\theta}}\left(\log p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z})\right) \end{aligned} \tag{5.13}$$

The last line is a Monte Carlo approximation of the second line in Equation (5.13). Note that the gradients of the generative parameters are unbiased since the expectation is w.r.t. to the inference parameters and thus the gradient operation can be easily computed for the terms inside the expectation operation. In the case of the variational parameters, the gradients are:

$$\begin{aligned} \nabla_{\phi} L_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x}) &= \nabla_{\phi} \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z}) - \log q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})\right] \\ &\neq \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\nabla_{\phi}\left(\log p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z}) - \log q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})\right)\right]. \end{aligned} \tag{5.14}$$

These gradients are unbiased since the expectation and the gradient operation, both are to be computed w.r.t. the inference parameters $\boldsymbol{\phi}$.

5.4.2. Reparameterization Trick

Due to the intractability of the unbiased gradient of the variational parameters, a change of variables is introduced in the sampling of the approximate latent posterior, called the reparameterization trick. Concretely, the random variable $\boldsymbol{Z} \sim q_{\boldsymbol{\phi}}(\boldsymbol{Z} \mid \boldsymbol{x})$ is transformed to now be a function of another random variable $\boldsymbol{\epsilon}$, given $\boldsymbol{x}, \boldsymbol{\phi}$:

$$\boldsymbol{z} = \boldsymbol{g}(\epsilon, \boldsymbol{\phi}, \boldsymbol{x}) \tag{5.15}$$

where $\boldsymbol{\epsilon}$ is independent of $\boldsymbol{x}, \boldsymbol{\phi}$. Now, the expectation is transformed into:

$$\mathbb{E}_{q_{\phi}(\boldsymbol{z}|\boldsymbol{x})}[f(\boldsymbol{z})] = \boldsymbol{E}_{p(\epsilon)}[f(\boldsymbol{z})] \tag{5.16}$$

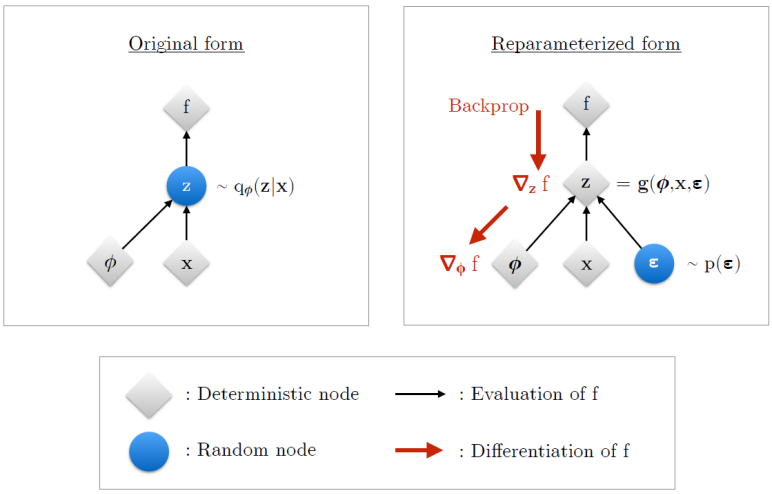


Figure 5.3: Gradients w.r.t. ϕ can now be unbiased since the expectation has a change of variable to ϵ. Figure courtesy Diederik P Kingma, Welling, et al. (2019).

and the gradient now becomes unbiased as follows:

$$\begin{aligned} \nabla_{\phi} \mathbb{E}_{q_{\phi}(\boldsymbol{z}|\boldsymbol{x})}[f(\boldsymbol{z})] &= \nabla_{\phi} \mathbb{E}_{p(\boldsymbol{\epsilon})}[f(\boldsymbol{z})] \\ &= \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\nabla_{\phi} f(\boldsymbol{z})\right] \\ &\simeq \nabla_{\phi} f(\boldsymbol{z}). \end{aligned} \tag{5.17}$$

$\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$ can be simulated using random noise. An illustration for the reparameterization trick can be seen in Fig. 5.3.

5.4.3. Practical Applications

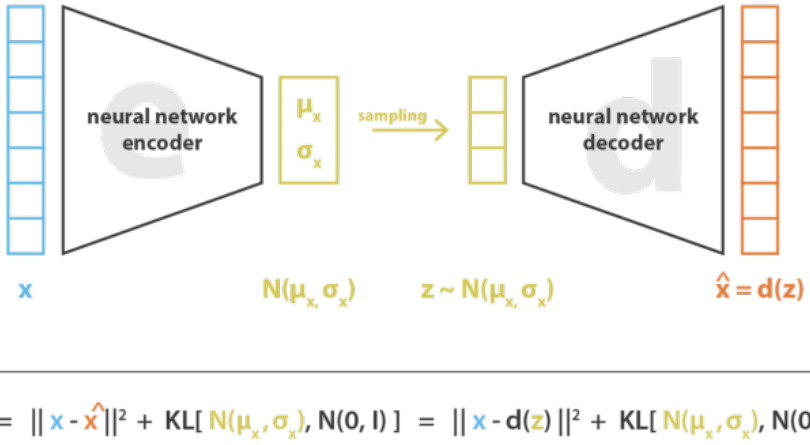


Figure 5.4: A VAE's objective function is comprised of a reconstruction term that makes the inference and generator network efficient, and a KL divergence term that acts as a regularizer for the inferred latent distributions. Figure courtesy Joseph Rocca.[1]

In practical applications of the VAE, the likelihood of the data conditional on the latent variable $p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})$ is assumed to either be Gaussian or Bernoulli, thus reducing the log-likelihood term into the mean squared error (MSE) or the binary cross-entropy loss (BCE), as derived in Chapter 3, Section 3.1.2, 3.1.1. Also, the prior for latent variable $p_{\boldsymbol{\theta}}(\boldsymbol{z})$ is considered to be a standard normal distribution while

[1] https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

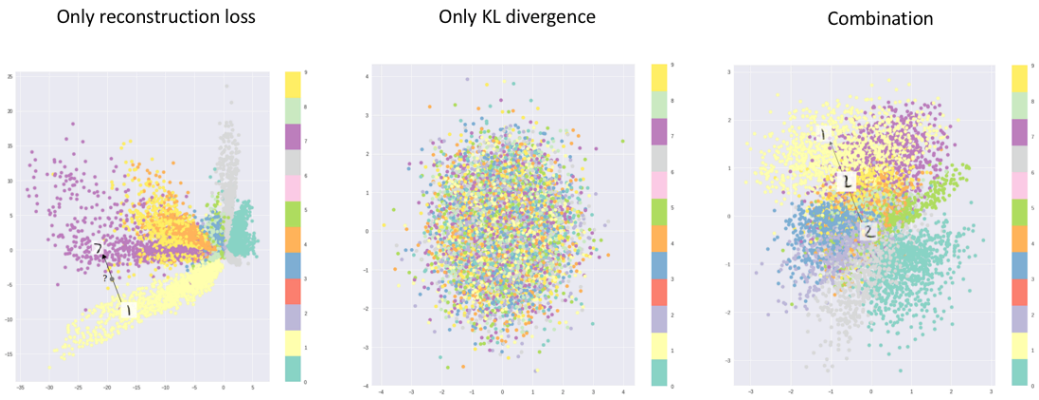
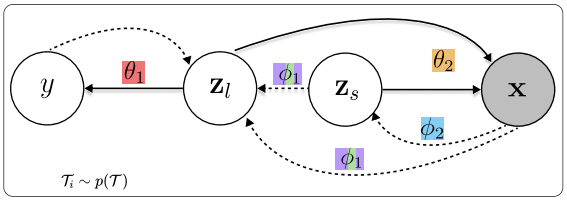
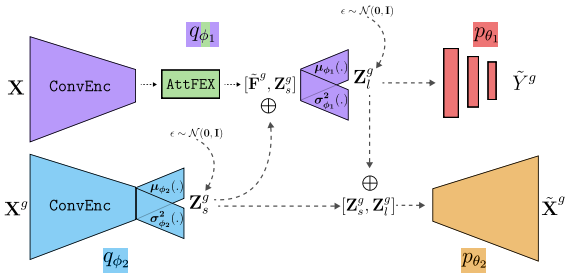


Figure 5.5: The KL divergence term prevents the network from “cheating" by forcing it to learn distributions instead of very high variance narrow distributions that only optimize reconstruction loss. Figure courtesy Irhum Shafkat.[2]



(a) TRIDENT's DAG illustrating variational inference (dotted lines) and generative process (solid lines). The inference and generative parameters are color coded to correspond to their respective architectures.



(b) TRIDENT's two intertwined variational networks that infer $\mathbf{Z}_l$ and $\mathbf{Z}_s$.

Figure 5.6: TRIDENT's generative model and corresponding variational network.

the approximate posterior is assumed to be a Gaussian distribution, with its parameters $\boldsymbol{\mu}, \boldsymbol{\sigma}$ parameterized using the inference neural network $\boldsymbol{\phi}$. The KL divergence term of the VAE ELBO term (Fig. 5.4) acts as a regularization term for the approximate posterior. When the prior is assumed to be standard normal Gaussian with zero mean and unit variance, the KL term forces the latent distribution to be close to this normal distribution by penalizing it when the mean deviates a lot from the origin just to maintain high reconstruction quality. The KL divergence term also helps learn well spread distributions instead of high-variance narrow distributions that only optimize for the reconstruction quality. This ensures that the inferred latent space is evenly distributed with minimally small regions that represent no input data. This is visualized using an example of the handwritten digits dataset in Fig. 5.5.

TRIDENT follows the same principles of amortized variational inference where the variational parameters ϕ_1 infer the latent variable $\mathbf{Z}_l$ and ϕ_2 infers the latent variable $\mathbf{Z}_s$. The generative model of TRIDENT makes assumptions about the causal structure of an image and its label that are reflected in its DAG (Fig. 5.6a). It is assumed that there are two salient latent factors that describe the essence of an image and its label: the *semantic* ($\mathbf{Z}_s$) and *label* ($\mathbf{Z}_l$) latent variables. The semantic and label latent variables are assumed to be the generative factors of the image while the label latent variable is responsible for generating the label of the image. The inference mechanics of the latent variables, decided by this DAG, are given in equation (1) of the scientific article. Label information is avoided as input to the inference network q_{ϕ_1} to utilize TRIDENT for classification and not label reconstruction. To compensate for the lack of important label information in the inference of the label latent variable, AttFEX is used to induce task-specificity in the feature maps of input images. This injects a semblance of label characteristics as input in the inference network q_{ϕ_1}, thus acting as an approximation to the true one-hot encoded label vectors Y. Finally, the variational network of TRIDENT is designed that follows

[2] https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf

the inference and generative mechanics of our DAG. The loss function to be optimized is decided by the ELBO derivation in equation (2) of 2, where the log-likelihood of images $\log p_{\theta_2}(\mathbf{x} \mid \mathbf{z}_s, \mathbf{z}_l)$ is an MSE term and the log-likelihood of labels $\log p_{\theta_1}(y \mid \mathbf{z}_l)$ is the cross-entropy term.

6

Few-Shot Learning

Deep learning has made tremendous advances in vision, speech, language and even natural sciences, yet still remains dependent on massive amounts of data. This is far from the goal of creating artificial “intelligence" since human beings, that have set the upper limit for intelligence, learn new skills and abstract concepts with very few data examples. Few shot learning is an endeavour to transcend this capability of humans onto machines, where the model must learn to generalize to new classes and concepts, given only a few samples per class during training. Few shot classification is an instantiation of the broader few shot learning problem statement, where the classifier is trained on a dataset containing several classes, but only a few handful (typically $1-5$) of samples per class. Given this data-deficient setting, an ideal few shot learning algorithm must learn to generalize well and quickly enough to classify images from new unseen test-data distributions.

6.1. Problem Definition

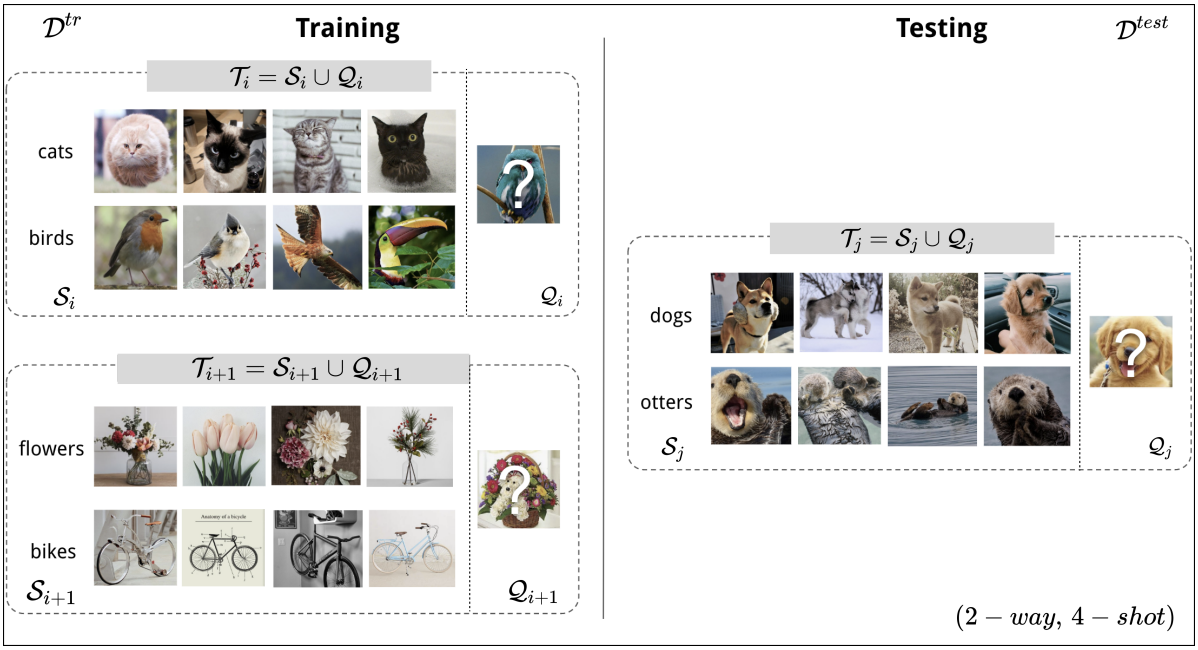


Figure 6.1: Illustration of a (2-way, 4-shot) setting where the each rectangular dotted box depicts a task $\mathcal{T}_i$, comprised of a support ($\mathcal{S}_i$) and query ($\mathcal{Q}_i$) set. During testing, typically $600-2000$ tasks are sampled for an unbiased estimate of the performance but only 1 task $\mathcal{T}_j$ is illustrated for the sake of simplicity.

Consider a labelled dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i) \,|\, i \in [1, N']\}$ of images $\mathbf{x}_i$ and class labels y_i. This

dataset $\mathcal{D}$ is divided into three disjoint subsets: $\mathcal{D} = \{\mathcal{D}^{tr} \cup \mathcal{D}^{val} \cup \mathcal{D}^{test}\}$, respectively, referring to the training, validation, and test subsets. The validation dataset $\mathcal{D}^{val}$ is used for model selection and the testing dataset $\mathcal{D}^{test}$ for final evaluation. Following standard few-shot classification settings Vinyals et al. 2016; Snell, Swersky, and Zemel 2017; Sung et al. 2018, we use episodic training on a set of tasks (also called episodes) $\mathcal{T}_i \sim p(\mathcal{T})$. The tasks are constructed by drawing K random samples from N different classes, which we denote as an (N-way, K-shot) task. Concretely, each task $\mathcal{T}_i$ is composed of a *support* and a *query* set, as illustrated in Fig. 6.1. The support set $\mathcal{S} = \{(\mathbf{x}^S_{kn}, y^S_{kn}) \mid k \in [1, K], n \in [1, N]\}$ contains K samples per class and the query set $\mathcal{Q} = \{(\mathbf{x}^Q_{kn}, y^Q_{kn}) \mid k \in [1, Q], n \in [1, N]\}$ contains Q samples per class. For a given task, the NQ query and NK support images are mutually exclusive to assess the generalization performance.

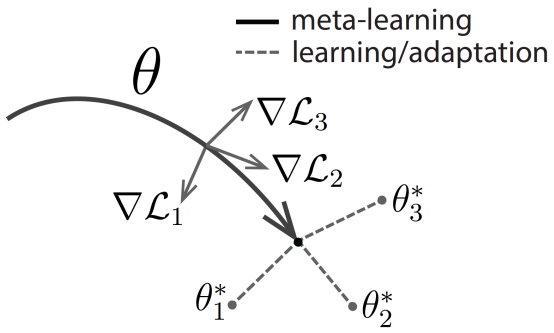
Episodic training was proposed by Vinyals et al. (2016) to make the training process mimic the testing phase, where the classifier's generalization ability is tested on multiple tasks with limited labeled samples. One might wonder why *Transfer Learning* based approaches are not suitable for the few shot classification problem setting, where a deep neural network is first trained on training dataset $\{\mathcal{D}^{tr}$, and then *fine-tuned* on the support sets of the testing dataset ($\{\mathcal{D}^{test}$) tasks. The issue with this approach is that it would severely overfit on few-shot tasks since the limited amount of labeled support samples (1-5) would not be sufficient enough to accurately represent the true distribution of classes. This results in high-variance fine-tuned models that fail to generalize on new unseen tasks. This claim has been empirically verified in Vinyals et al. (2016) and Finn, Abbeel, and Levine (2017). However, more recent works have shown the effectiveness of transfer learning based approaches (Boudiaf et al. 2020; Dhillon et al. 2020; Tian et al. 2020; Ziko et al. 2020) by achieving state-of-the-art accuracies on standard few-shot classification benchmarks. These works show that it is important to use *transductive inference*, where information from the unlabeled query samples is used in adjusting the decision boundary, either by iteratively propagating labels from the support to the query set or by using an additional measure of entropy on the query samples to ensure that the decision boundary is located in a low-density region.

The following sections discuss one seminal work from each category of methodologies developed for solving few-shot image classification.

6.2. Model Agnostic Meta Learning

Model agnostic meta learning (MAML), is an **optimization based** few shot learning method that learns to learn (meta-learn) by improving the learning strategy over multiple episodes/tasks (Finn, Abbeel, and Levine 2017). More specifically, MAML trains to find an initial set of weights that can be quickly adapted to new tasks, via just a few gradient descent steps. This quick adaptation is achieved by training the network to maximize the sensitivity of the loss function of new tasks with respect to the parameters, so that even small local changes in the parameters can lead to large improvements in the task loss (Finn, Abbeel, and Levine 2017). Consider a neural network f_θ that must be optimized for few shot classification. The goal is to find a set of parameters θ^* that generalize quickly within a few steps of gradient descent (Fig. 6.2a). To do so, first the neural network performs a few steps of gradient descent on the support set of each task from a randomly sampled meta-batch consisting of a small number of tasks, say B (line 7, Fig. 6.2b). This is known as the inner-update which results in a set of parameters $\boldsymbol{\theta} = \theta'_i \forall i \in [1, B]$. Using these parameters $\boldsymbol{\theta}$, the loss is computed over all the corresponding query sets using their respective inner-updated parameters (line 10, Fig. 6.2b). This is known as the meta-update step. This entire process makes up one iteration of the MAML algorithm. Multiple of these iterations are carried out until convergence, in order to find the optimal meta-parameters θ^*.

A VAE at least requires a few hundred samples to learn a good latent space for reconstruction. With only $1 - 5$ samples available per class in few-shot settings, the variational network would be handicapped in estimating a good latent posterior mapping that generalizes well across tasks. Thus, the variational and fixed parameters $\{\boldsymbol{\phi}, \boldsymbol{\theta}\}$ are made sensitive to small changes across tasks while learning the conditional distributions of latent variables, using a MAML-style training approach.



(a) MAML optimizes for a θ that is capable of quick adaptation to new tasks. Figure from Finn, Abbeel, and Levine (2017).

Algorithm 2 MAML for Few-Shot Supervised Learning

Require: $p(\mathcal{T})$: distribution over tasks
Require: α, β: step size hyperparameters
randomly initialize θ
while not done **do**
  Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
  for all $\mathcal{T}_i$ **do**
    Sample K datapoints $\mathcal{D} = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$ from $\mathcal{T}_i$
    Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ using $\mathcal{D}$ and $\mathcal{L}_{\mathcal{T}_i}$ in Equation (2) or (3)
    Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
    Sample datapoints $\mathcal{D}_i' = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$ from $\mathcal{T}_i$ for the meta-update
  end for
  Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$ using each $\mathcal{D}_i'$ and $\mathcal{L}_{\mathcal{T}_i}$ in Equation 2 or 3
end while

(b) MAML computes inner-updates on the support set and backpropagates through the gradient descent steps of the support adaptation, as a part of the meta-update on the query set. Figure from Finn, Abbeel, and Levine (2017).

6.3. Prototypical Networks

Prototypical nets or proto-nets (Snell, Swersky, and Zemel 2017) is an example of the **metric-learning** approach for few-shot classification. The idea here is to learn a common mapping across tasks, that projects images to an embedding space that is optimal for classification using a distance metric. Proto-nets solve the problem of few shot learning by addressing the key issue of overfitting on a small number of samples, leading to poor generalization on new unseen tasks. To this end, the authors propose to work under an assumption that a classifier should have a simple inductive bias. The aim is to learn an embedding space where the data points cluster around a single prototype representation of each class. First, each image is transformed into an embedding using a neural network f_ϕ that acts as the

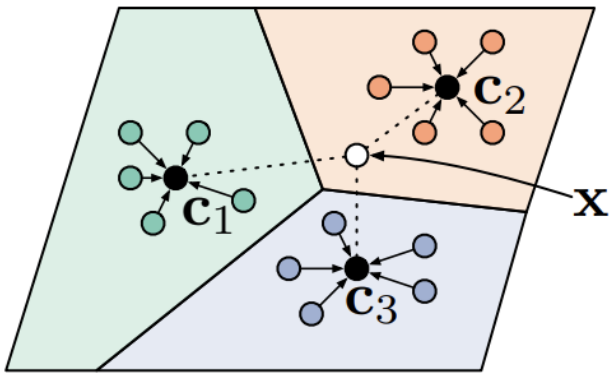


Figure 6.3: Prototypes c_k are the mean embeddings of the support examples and a query sample x is classified based on the distance between all pairs of c_k, x. Figure from Snell, Swersky, and Zemel (2017).

common embedding function used across tasks. Then, the support embeddings of each class $\mathcal{S}_n$ are averaged to compute the prototypes for each class:

$$\boldsymbol{c}_n = \frac{1}{|\mathcal{S}_n|} \sum_{(\boldsymbol{x}_i, y_i) \in \mathcal{S}_n} f_\phi(\boldsymbol{x}_i) \tag{6.1}$$

Finally, the class conditional probabilities of the query samples are computed using a given distance metric, which is the Euclidean distance metric in this case.

$$p_\phi(y = n \mid \boldsymbol{x}) = \frac{\exp\left(-d\left(f_\phi(\boldsymbol{x}), \boldsymbol{c}_n\right)\right)}{\sum_{n'} \exp\left(-d\left(f_\phi(\boldsymbol{x}), \boldsymbol{c}_{n'}\right)\right)} \tag{6.2}$$

The distances are normalized using the softmax activation so that these represent the probabilities of classes. The NLL of the class-conditional probabilities is minimized using SGD, which is equivalent to minimizing the cross-entropy. Since the euclidean distance between query samples and the mean of each class is used to compute the class-conditionals, this method is equivalent to the **nearest-mean classifier**, where the implicit assumption is that the class distributions are Gaussians with equal covariance matrices, all being equal to I. This also reflects in the linear decision boundaries (Fig. 6.3) between class distributions in the embedding space.

6.4. Transductive CNAPS

Transductive CNAPS (Bateni, Barber, et al. 2022), is a **transductive inference** and **task-aware** (also called **transductive feature extraction**) approach that builds on Simple CNAPS (Bateni, Goyal, et al. 2020). Simple CNAPS is a metric learning approach that builds on Proto-nets by using a Mahalanobis distance (Mahalanobis 1936) metric instead of a Euclidean distance metric. This involves estimating the class-covariance matrices, which in turn lead to quadratic decision boundaries. An advantage of using these is that the decision-boundary now takes into account the differences in class-distributions resulting in improved non-linear boundaries (Fig. 6.4). The decision-boundaries also improve since the two major flawed assumptions of unit and identical class-covariance matrices of proto-nets is relaxed in this case. Finally, the class conditional distributions are given by:

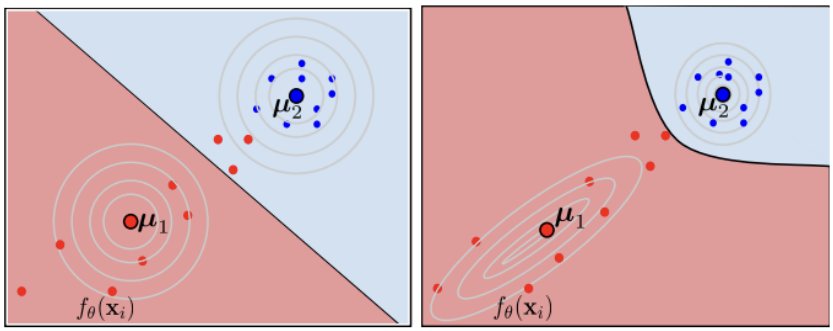


Figure 6.4: Euclidean norm (left) assumes same I covariance matrices whereas Mahalanobis distance (right) incorporates different covariances. Figure from Bateni, Goyal, et al. (2020).

$$p\left(y_i^* = n \mid f_\theta^\tau\left(\boldsymbol{x}_i^*\right), S^\tau\right) = \operatorname{softmax}\left(-d_n\left(f_\theta^\tau\left(\boldsymbol{x}_i^*\right), \boldsymbol{\mu}_n\right)\right) \tag{6.3}$$

where $\boldsymbol{x}_i^*$ is the query sample, $\boldsymbol{\mu}_n$ is the mean of class n and d_n is given by the Mahalanobis distance metric:

$$d_k(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{2}(\boldsymbol{x} - \boldsymbol{y})^T \left(\boldsymbol{Q}_n^\tau\right)^{-1} (\boldsymbol{x} - \boldsymbol{y}) \tag{6.4}$$

where $\boldsymbol{Q}_n^\tau$ is the class-covariance matrix. This can also be interpreted as the **quadratic discriminant classifier** since it assumes different Gaussian distributions for each class and then performs classification by computing the probability density function for all query-class pairs.

Transductive CNAPS introduces a transductive feature extraction module and an iterative transductive inference technique, that use information from the query samples to extract feature embeddings and perform inference, respectively. In the transductive feature extraction module, first the embeddings for support and query examples are extracted using the encoder CNN. Then, these are mean pooled to get $\boldsymbol{e}_S, \boldsymbol{e}_Q$ for support and query sets, respectively. Then, these are processed through two steps of a Long Short Term Memory (LSTM) network in order to get the final task-specific embeddings, as illustrated in Fig. 6.5.

To carry our transductive inference, an expectation-maximization routine is carried out where first inference is performed as in Equation (6.3), and then these class-conditional probabilities are used to compute weighted estimates of mean and covariance metrics of each corresponding class using the query features. These two steps are repeated for a few iterations until the convergence. This

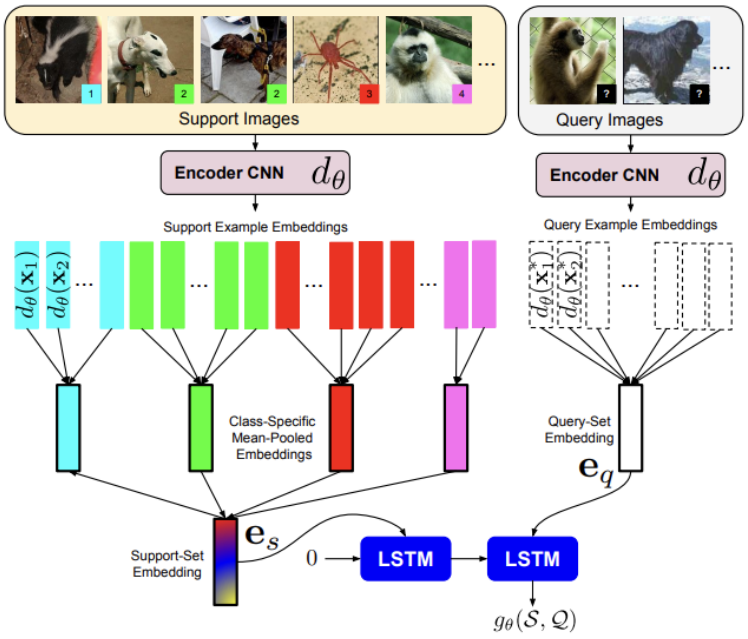


Figure 6.5: Information from both, support and query samples is used by g_θ to compute task-specific embeddings. Figure from Bateni, Goyal, et al. (2020).

iterative classification and distribution refinement is known as the **soft k-means** algorithm, the details of which can be found in (Dunn 1973). Since the information from the query samples is also used in classifying them, and the inference is performed only on the selected query data points instead of the entire domain of definition, this method is called transductive inference (Gammerman, Vovk, and V. Vapnik 1998; V. N. Vapnik 2006).

6.5. Transductive Propagation Networks

Transductive propagation networks was the first method to model transductive inference (Gammerman, Vovk, and V. Vapnik 1998; V. N. Vapnik 2006) explicitly in the few-shot classification problem. The aim here is to construct a graph structure with nodes as the support and query feature embeddings and the edges weighted by the distance between nodes. The distance is measured using an appropriate kernel metric, Gaussian in this case. Once the graph is constructed, labels are iteratively propagated from the support nodes to the unlabelled query nodes based on the distance/similarity between the node pairs (Fig. 6.6). A CNN f_ψ is used for extracting the features, which in turn form the node embeddings.

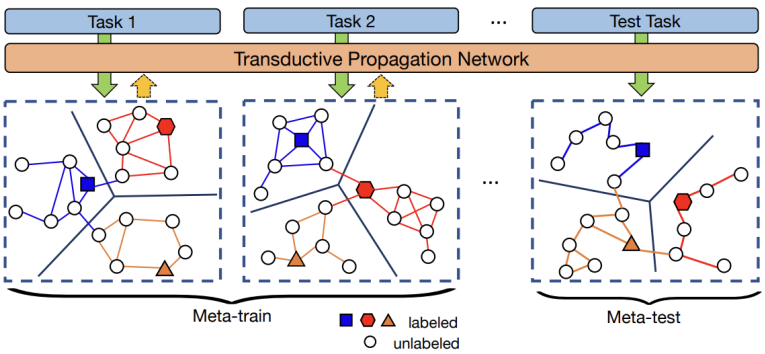


Figure 6.6: The labels are iteratively propagated from the support samples (colored) to the query samples (hollow) by exploiting the graph structure between embeddings as nodes and edge weights as distances. Figure from Yanbin Liu et al. (2019).

These node embeddings are then passed through another CNN g_ϕ that extract a set of embeddings responsible for the variance estimates σ for each node-pair, as shown in Fig. 6.7. These are used in the edge weight calculation as follows:

$$W_{ij} = \exp\left(-\frac{1}{2}d\left(\frac{f_\varphi(\boldsymbol{x_i})}{\sigma_i}, \frac{f_\varphi(\boldsymbol{x_j})}{\sigma_j}\right)\right). \tag{6.5}$$

Finally, the labels are propagated from support to query nodes using the manifold smoothing closed-

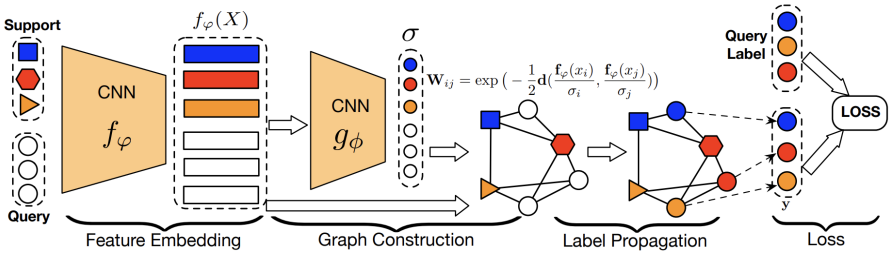


Figure 6.7: The architecture diagram of TPN illustrating the data flow from input images to the computation of the cross-entropy loss. Figure from Yanbin Liu et al. (2019).

form formulation:

$$F^* = (I - \alpha S)^{-1} Y \tag{6.6}$$

where $S = D^{-1/2} W D^{-1/2}$ with D being the diagonal matrix with its diagonals being the sum of rows of W, Y being a vector of labels (with 0s in place of the query labels) and I being the identity matrix. The matrix S is known as the graph's Laplacian. The propagated labels in F^* form the logits and are then softmax normalized to give the class-conditional probabilities F^*_{ij} for class j and sample i.

6.6. Meta Learning Probabilistic Inference

ML-PIP (Gordon et al. 2019) is a hierarchical **probabilistic** inference method for few shot learning that replaces gradient based meta-updates with forward passes through the inference networks at test time, for quick adaptation. The hierarchical probabilistic structure allows ML-PIP to share information across tasks via the shared latent variable θ, which is also responsible for meta-learning. The hierarchical DAG structure (Fig. 6.8) contains task-specific parameters ψ^t, where each task is comprised of a support set $\mathcal{D}^t = \{(x_n^t, y_n^t) : n = 1 : N_t\}$ and a query set $\mathcal{D}^t_{test} = \{(\tilde{x}_m^t, \tilde{y}_m^t) : n = 1 : M_t\}$. A point

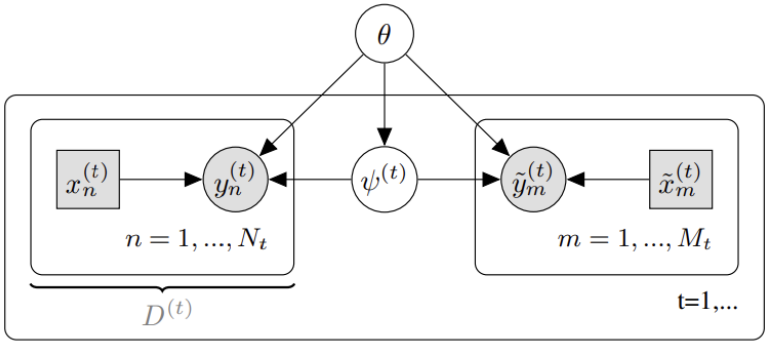


Figure 6.8: Directed graphical model for multi-task learning. Figure from Gordon et al. (2019)

estimate is learnt for the shared parameter θ since it has little uncertainty due to the presence of large amounts of data for its estimation. An approximate posterior distribution is estimated for ψ^t and is denoted by $p(\psi^t \mid \tilde{x}^t, \mathcal{D}^t, \theta)$. The posterior predictive distribution is a result of marginalizing out the task-specific parameters, and is given by:

$$p\left(\tilde{y}^t \mid \tilde{x}^t, \mathcal{D}^t, \theta\right) = \int p\left(\tilde{y}^t \mid \tilde{x}^t, \psi^t, \theta\right) p\left(\psi^t \mid \mathcal{D}^t, \theta\right) d\psi^t. \tag{6.7}$$

As discussed in earlier chapters, it is intractable to compute the exact posterior and thus an amortized approximate posterior is learnt, denoted by $q_\phi(\psi^t \mid \tilde{x}^t, \mathcal{D}^t, \theta)$. The best parameters are estimated by maximizing the expected accuracy of the approximate posterior predictive distribution, given any dataset:

$$\begin{aligned}\phi^* &= \underset{\phi}{\operatorname{argmin}} \mathbb{E}_{p(\mathcal{D},\tilde{x})}\left[D_{\mathbb{KL}}\left(p(\tilde{y} \mid \tilde{x}, \mathcal{D}, \theta) \| q_\phi(\tilde{y} \mid \tilde{x}, \mathcal{D}, \theta)\right)\right] \\ &= \underset{\phi}{\operatorname{argmin}} \mathbb{E}_{p(\mathcal{D},\tilde{x},\tilde{y})}\left[\log \int p(\tilde{y} \mid \tilde{x}, \psi, \psi) q_\phi(\psi \mid \mathcal{D}, \theta) d\psi\right].\end{aligned} \tag{6.8}$$

The outer expectation can be computed by sampling multiple tasks from $p(\mathcal{D})$, which acts as a Monte Carlo approximation. The inner expectation (integral) is computed by drawing S samples from the task-specific parameter posterior $\psi_s^t \sim q_\phi(\psi^t \mid \mathcal{D}^t, \theta)$. The resulting training objective now takes the form:

$$\mathcal{L}_{\text{ML-PIP}}(\theta, \phi) = \frac{1}{MT} \sum_{m=1}^{M} \sum_{t=1}^{T} \log\left(\frac{1}{S} \sum_{s=1}^{S} p\left(\tilde{y}_m^t \mid \tilde{x}_m^t, \psi_s^t\right)\right) \tag{6.9}$$

Note that this is different from standard amortized *variational* inference since the emphasis is only on minimizing the difference between the true and approximate posterior predictive distribution, and not the difference between the approximate and true posterior distributions $q_\phi(\psi^t \mid \mathcal{D}_{all}, \theta)$ and $p(\psi^t \mid \theta)$.

7

TRIDENT Additional Discussion

In this chapter, we talk about architectural details and modelling design choices that failed to work. Learning from these failures and building on the observations consequently made TRIDENT what it is now. Hopefully, this provides additional insight and further motivation to support the claims and hypothesis that TRIDENT makes.

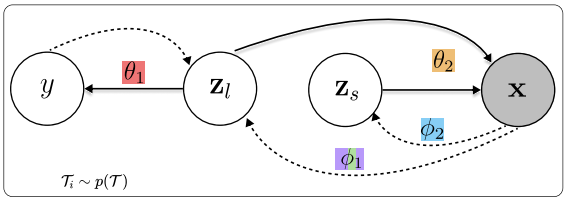
The idea of inferring decoupled information from data, based on the needs of the specific downstream task has been the key driving factor behind this research. From the perspective of a human being, we observe multiple details about the environment around us, however, we only choose to utilize specific parts of the observed information for a given task. This specific information is dependent on the nature of the task at hand. We apply the same line of thought to the problem of image classification. The essence of an image is comprised of various different aspects such as the context it was captured in, the context it depicts and the general style in which it is portrayed (a 3D render, an oil painting, a vapor-wave design, digital art, hand drawn sketch, an animated caricature etc.). Depending on the other images it may be discriminated against, not all of these aspects and characteristics are relevant to the image's classification. However, there are subtle details in images that set them apart from each other, which we call class-characterizing label attributes. These label attributes are decided based on the context of the task, of course. It can be argued that optimizing a simple cross-entropy loss for classification already acts a sufficient objective to guide the neural network's information extraction process. But in a few-shot scenario, there is extremely limited data for the network to learn from and then generalize on new unseen classes. In such a case, it is sub-optimal to leave it to the network to find the perfect combination of neural connections, since the amount of data is simply not enough for the network to learn the data distributions and find satisfactory decision boundaries. Thus, it is important to manually engineer an inductive bias by making assumptions about the causal structure of the data and deciding on which variables to infer. To this end, we decided to use semantic and label variables as the latent generative variables for the image and the label respectively. The inference and generative mechanics, and the variational lower bound followed from this line of thought.

7.1. A False Independence Assumption

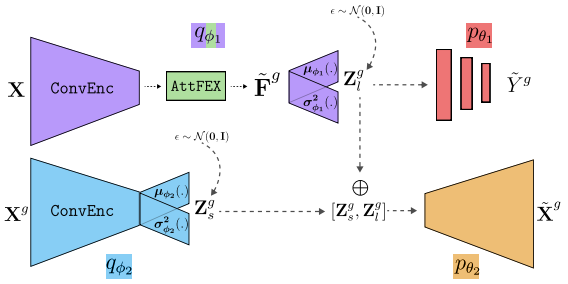
Initially, we disregarded the rule of conditional dependence on collider nodes (Chapter 3, Section 3.5) in the inference mechanics of TRIDENT. Concretely, there was no dependence considered between $\boldsymbol{z}_s$ and $\boldsymbol{z}_l$ given $\boldsymbol{x}$, when inferring the two latent variables. With this assumption, the factorized form for q_ϕ is $q_\phi\left(\boldsymbol{z}_s, \boldsymbol{z}_l \mid \boldsymbol{x}, y\right) = q_{\phi_1}\left(\boldsymbol{z}_l \mid \boldsymbol{x}\right) q_{\phi_2}\left(\boldsymbol{z}_s \mid \boldsymbol{x}\right)$. Thus, the amortized variational inference networks were:

$$q_{\phi_2}\left(\boldsymbol{z}_s \mid \boldsymbol{x}\right) = \mathcal{N}\left(\boldsymbol{z}_s \mid \mu_{\phi_2}(\boldsymbol{x}), diag(\sigma^2_{\phi_2}(\boldsymbol{x}))\right);\ q_{\phi_1}\left(\boldsymbol{z}_l \mid \boldsymbol{x}, \boldsymbol{z}_s\right) = \mathcal{N}\left(\boldsymbol{z}_l \mid \mu_{\phi_1}(\boldsymbol{x}, \boldsymbol{z}_s), diag(\sigma^2_{\phi_1}(\boldsymbol{x}, \boldsymbol{z}_s))\right). \tag{7.1}$$

The DAG corresponding to this has been illustrated in Fig. 7.1a. This makes a strong, and rather incorrect, assumption about the latent variables. This is that even though $\boldsymbol{z}_s$ and $\boldsymbol{z}_l$ together decide



(a) TRIDENT's DAG illustrating variational inference (dotted lines) and generative process (solid lines). The inference and generative parameters are color coded to correspond to their respective architectures.



(b) TRIDENT's two variational networks, intertwined only in their generative structures, that infer $\mathbf{Z}_l$ and $\mathbf{Z}_s$.

Figure 7.1: Generative model and corresponding variational network with the Conditional Independence Assumption.

the generative process of $\boldsymbol{x}$, knowing something about $\boldsymbol{x}$ does not tell anything about $\boldsymbol{z}_s$ while trying to infer $\boldsymbol{z}_l$ or vice-versa. Proceeding with this assumption, we formulated the ELBO which looked like:

$$\begin{aligned}\mathcal{L}(\Psi) = &-\mathbb{E}_{q_{\phi_2}}\left[\ln p_{\theta_2}(\boldsymbol{x} \mid \boldsymbol{z}_s, \boldsymbol{z}_l)\right] - \mathbb{E}_{q_{\phi_1}}\left[\ln p_{\theta_1}(y \mid \boldsymbol{z}_l)\right] + \\ &D_{KL}\big(q_{\phi_1}(\boldsymbol{z}_l \mid \boldsymbol{x}, \boldsymbol{z}_s) \| \, p(\boldsymbol{z}_l)\big) + D_{KL}\big(q_{\phi_2}(\boldsymbol{z}_s \mid \boldsymbol{x}) \| \, p(\boldsymbol{z}_s)\big),\end{aligned} \tag{7.2}$$

The implications of this assumption on the network architecture and the data flow are reflected in Fig. 7.1b. Finally, the accuracies achieved for the benchmark datasets and the cross-domain scenario are reflected in Table 7.1, where the accuracies with the incorrect assumption are shown in the first row and the scores with the correct model in the second row. Note that TRIDENT is equipped with AttFEX in both these scenarios since the compensation for the lack of label input for $\boldsymbol{z}_l$'s inference remains consistent throughout. The massive drop in generalization capability of TRIDENT, not only

Table 7.1: Accuracies in (% ± std).

		***mini*Imagenet**		***tiered*Imagenet**		***mini*→CUB**	
Methods	**Backbone**	**5-way 1-shot**	**5-way 5-shot**	**5-way 1-shot**	**5-way 5-shot**	**5-way 1-shot**	**5-way 5-shot**
TRIDENT**(Assumption)**	Conv4	**59.78 ± 0.45**	**67.55 ± 0.8**	**64.89 ± 0.52**	**73.89 ± 0.78**	**44.56 ± 0.59**	**48.90 ± 0.37**
TRIDENT	Conv4	**86.11 ± 0.59**	**95.95 ± 0.28**	**86.97 ± 0.50**	**96.57 ± 0.17**	**84.61 ± 0.33**	**80.74 ± 0.35**

for the cross-domain scenario, but also for in-domain unseen examples, corroborates the fact that this assumption of conditional independence on a collider node is incorrect. In other words, making a fallacious assumption about the causal structure of the underlying true data distribution hampers the inference capabilities of the model, which reflects in its generalization capability.

7.2. Experimenting with Manifold Smoothing

Before engineering AttFEX, we experimented with the idea of manifold smoothing as a way of inducing transduction in the feature extraction process. Inspired by the work of Yanbin Liu et al. (2019), the idea was to use propagation of information using feature embeddings in a graph structure. We constructed a graph with nodes as the feature embeddings extracted by the upper *ConvEnc* and edge weights decided by the Gaussian distance metric between node pairs. The σ value was calculated as $\sigma^2 = Var(d_{ij})$ where d_{ij} denotes the Gaussian distance metric between node i, j. Finally, the graph's Laplacian was used to propagate information between the node features themselves, rather than the label vectors. This acted as a smoothing operation between the node features, since the embeddings were now a weighted sum of all the node embeddings, with the weight depending on the distance/similarity between the node features. This removed noise from the manifold that the node embeddings sat on, thus acting as a manifold smoothing operation. The results for this are depicted in Table 7.2. The network fails to generalize as well as it does with AttFEX. Our hypothesis for this is that the convolutional inference network, on its own, isn't equipped enough to learn an embedding space for inferring a latent $\boldsymbol{z}_l$ that is close enough to the true posterior, while at the same time, optimizing for task-specificity. Thus, the efficient utilization of extra parameters from AttFEX for task-specificity is

Table 7.2: Accuracies in (% ± std).

		*mini*Imagenet		*tiered*Imagenet		*mini*→CUB	
Methods	**Backbone**	**5-way 1-shot**	**5-way 5-shot**	**5-way 1-shot**	**5-way 5-shot**	**5-way 1-shot**	**5-way 5-shot**
TRIDENT**(with Manifold Smoothing)**	Conv4	**69.84 ± 0.5**	**80.15 ± 0.67**	**74.12 ± 0.90**	**82.44 ± 0.23**	**61.89 ± 0.73**	**72.82 ± 0.31**
TRIDENT**(with** AttFEX**)**	Conv4	**86.11 ± 0.59**	**95.95 ± 0.28**	**86.97 ± 0.50**	**96.57 ± 0.17**	**84.61 ± 0.33**	**80.74 ± 0.35**

vital to the performance of TRIDENT by compensating for the lack of label input.

8

Conclusions and Future Directions

This work introduces a novel variational inference network (coined as TRIDENT) that simultaneously infers decoupled latent variables representing semantic and label information of an image. The proposed network is comprised of two intertwined variational networks responsible for inferring the semantic and label information separately, the latter being enhanced using an attention-based transductive feature extraction module (AttFEX). Our extensive experimental results corroborate the importance of this transductive decoupling strategy on a variety of few-shot settings showing superior performance and setting a new state-of-the-art for the most commonly adopted datasets *mini* and *tiered*Imagenet as well as for the recent challenging cross-domain scenario of *mini*Imagenet → CUB.

As future work, the aim is to make TRIDENT more modular. By doing so, this decoupling strategy can also be incorporated into other few-shot classification methodologies that employ different inference techniques. Along the same lines, AttFEX is already a plug-and-play module and can readily be adopted by other baselines to induce task-specificity in their extracted embedding spaces. Another avenue to explore is to demonstrate the applicability of TRIDENT in semi-supervised and unsupervised settings. This would be done by including the likelihood of unlabelled samples through marginalization over the label variable in the probabilistic objective function in the case of semi-supervised settings. Whereas in the unsupervised setting, augmentations of images in a task can be used to form the K-shots associated to classes, with each image being a class of its own. The label variable can now be interpreted as a 'pseudo'-label variable and the negative log-likelihood can be modelled by using a contrastive loss on all images in the task. This would render TRIDENT as an all-inclusive holistic approach towards solving few-shot classification problems.

Bibliography

[1] Rory A. Fisher. “THE USE OF MULTIPLE MEASUREMENTS IN TAXONOMIC PROBLEMS”. In: *Annals of Human Genetics* 7 (1936), pp. 179–188 (cit. on p. 17).

[2] Prasanta Chandra Mahalanobis. “On the generalized distance in statistics”. In: National Institute of Science of India. 1936 (cit. on p. 48).

[3] 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 p. 27).

[4] Elizbar Nadaraya. “On Estimating Regression”. In: *Theory of Probability and Its Applications* 9 (1964), pp. 141–142 (cit. on p. 34).

[5] Geoffrey S Watson. “Smooth regression analysis”. In: *Sankhy: The Indian Journal of Statistics, Series A* (1964), pp. 359–372 (cit. on p. 34).

[6] Joseph C Dunn. “A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters”. In: (1973) (cit. on p. 49).

[7] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer feedforward networks are universal approximators”. In: *Neural networks* 2.5 (1989), pp. 359–366 (cit. on p. 27).

[8] A Gammerman, V Vovk, and V Vapnik. *Learning by transduction, vol UAI98*. 1998 (cit. on p. 49).

[9] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324 (cit. on p. 33).

[10] Vladimir Naumovich Vapnik. “Estimation of Dependences Based on Empirical Data”. In: *Estimation of Dependences Based on Empirical Data* (2006) (cit. on p. 49).

[11] Arindam Banerjee. “An analysis of logistic models: Exponential family connections and online performance”. In: *Proceedings of the 2007 SIAM International Conference on Data Mining*. SIAM. 2007, pp. 204–215 (cit. on p. 37).

[12] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2014. arXiv: 1312.6114 [stat.ML] (cit. on p. 40).

[13] Durk P Kingma et al. “Semi-supervised Learning with Deep Generative Models”. In: *Advances in Neural Information Processing Systems*. Vol. 27. 2014 (cit. on p. 37).

[14] Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. “Human-level concept learning through probabilistic program induction”. In: *Science* 350.6266 (2015), pp. 1332–1338. DOI: 10.1126/science.aab3050. eprint: https://www.science.org/doi/pdf/10.1126/science.aab3050. URL: https://www.science.org/doi/abs/10.1126/science.aab3050 (cit. on p. 1).

[15] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. “Learning Structured Output Representation using Deep Conditional Generative Models”. In: *Advances in Neural Information Processing Systems*. Ed. by C. Cortes et al. Vol. 28. Curran Associates, Inc., 2015. URL: https://proceedings.neurips.cc/paper/2015/file/8d55a249e6baa5c06772297520da2051-Paper.pdf (cit. on p. 37).

[16] Madelyn Glymour, Judea Pearl, and Nicholas P Jewell. *Causal inference in statistics: A primer*. John Wiley & Sons, 2016 (cit. on pp. 25, 26).

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016 (cit. on pp. 29–33).

[18] Casper Kaae Sønderby et al. “Ladder variational autoencoders”. In: *Advances in neural information processing systems* 29 (2016) (cit. on p. 37).

[19] Oriol Vinyals et al. “Matching Networks for One Shot Learning”. In: *Advances in Neural Information Processing Systems*. Vol. 29. 2016 (cit. on pp. 1, 46).

[20] Chelsea Finn, Pieter Abbeel, and Sergey Levine. “Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks”. In: *Proceedings of the 34th International Conference on Machine Learning*. Vol. 70. Proceedings of Machine Learning Research. 2017, pp. 1126–1135 (cit. on pp. 2, 46, 47).

[21] Jake Snell, Kevin Swersky, and Richard Zemel. “Prototypical Networks for Few-shot Learning”. In: *Advances in Neural Information Processing Systems*. Vol. 30. 2017 (cit. on pp. 1, 46, 47).

[22] Ashish Vaswani et al. “Attention is All you Need”. In: *Advances in Neural Information Processing Systems*. Vol. 30. 2017 (cit. on pp. 34–36).
[23] Jacob Devlin et al. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *arXiv preprint arXiv:1810.04805* (2018) (cit. on p. 35).
[24] Nikhil Mishra et al. “A Simple Neural Attentive Meta-Learner”. In: *International Conference on Learning Representations*. 2018 (cit. on p. 2).
[25] Flood Sung et al. “Learning to Compare: Relation Network for Few-Shot Learning”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2018 (cit. on pp. 1, 46).
[26] Antreas Antoniou, Harrison Edwards, and Amos J. Storkey. “How to train your MAML.” In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019 (cit. on p. 2).
[27] Jonathan Gordon et al. “Meta-Learning Probabilistic Inference for Prediction”. In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=HkxStoC5F7 (cit. on pp. 2, 50).
[28] Diederik P Kingma, Max Welling, et al. “An introduction to variational autoencoders”. In: *Foundations and Trendső in Machine Learning* 12.4 (2019), pp. 307–392 (cit. on pp. 37, 40, 42).
[29] Kwonjoon Lee et al. “Meta-Learning with Differentiable Convex Optimization”. In: *CVPR*. 2019 (cit. on p. 2).
[30] Hongyang Li et al. “Finding Task-Relevant Features for Few-Shot Learning by Category Traversal”. In: *CVPR*. 2019 (cit. on p. 2).
[31] Yanbin Liu et al. “Learning to Propagate Labels: Transductive Propagation Network for Few-shot Learning”. In: *International Conference on Learning Representations*. 2019 (cit. on pp. 2, 49, 50, 53).
[32] Yinhan Liu et al. “Roberta: A robustly optimized bert pretraining approach”. In: *arXiv preprint arXiv:1907.11692* (2019) (cit. on p. 35).
[33] Cuong C. Nguyen, Thanh-Toan Do, and Gustavo Carneiro. “Uncertainty in Model-Agnostic Meta-Learning using Variational Inference”. In: *CoRR* (2019). arXiv: 1907.11864 (cit. on p. 2).
[34] Adam Paszke et al. “Pytorch: An imperative style, high-performance deep learning library”. In: *Advances in neural information processing systems* 32 (2019) (cit. on p. 31).
[35] Sachin Ravi and Alex Beatson. “Amortized Bayesian Meta-Learning”. In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=rkgpy3C5tX (cit. on p. 2).
[36] James Requeima et al. “Fast and Flexible Multi-Task Classification using Conditional Neural Adaptive Processes”. In: *Advances in Neural Information Processing Systems*. Vol. 32. 2019 (cit. on p. 2).
[37] Andrei A. Rusu et al. “Meta-Learning with Latent Embedding Optimization”. In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=BJgklhAcK7 (cit. on p. 2).
[38] Yan Wang et al. *SimpleShot: Revisiting Nearest-Neighbor Classification for Few-Shot Learning*. 2019. arXiv: 1911.04623 [cs.CV] (cit. on p. 1).
[39] Jian Zhang et al. “Variational Few-Shot Learning”. In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 1685–1694. DOI: 10.1109/ICCV.2019.00177 (cit. on p. 2).
[40] Peyman Bateni, Raghav Goyal, et al. “Improved Few-Shot Visual Classification”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020 (cit. on pp. 1, 48, 49).
[41] Malik Boudiaf et al. “Information Maximization for Few-Shot Learning”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 2445–2457. URL: https://proceedings.neurips.cc/paper/2020/file/196f5641aa9dc87067da4ff90fd81e7b-Paper.pdf (cit. on pp. 2, 46).
[42] 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. 34, 35).
[43] Guneet Singh Dhillon et al. “A Baseline for Few-Shot Image Classification”. In: *International Conference on Learning Representations*. 2020 (cit. on pp. 2, 46).

[44] Shell Xu Hu et al. “Empirical Bayes Transductive Meta-Learning with Synthetic Gradients”. In: *International Conference on Learning Representations (ICLR)*. 2020. URL: https://openreview.net/forum?id=Hkg-xgrYvH (cit. on p. 2).

[45] Jinlu Liu, Liang Song, and Yongqiang Qin. “Prototype Rectification for Few-Shot Learning”. In: *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part I*. 2020 (cit. on p. 1).

[46] 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 p. 46).

[47] Han-Jia Ye et al. “Few-Shot Learning via Embedding Adaptation with Set-to-Set Functions”. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 8808–8817 (cit. on p. 2).

[48] Imtiaz Masud Ziko et al. “Laplacian Regularized Few-Shot Learning”. In: *ICML*. 2020, pp. 11660–11670 (cit. on pp. 2, 46).

[49] Zhuo Sun et al. “Amortized Bayesian Prototype Meta-learning: A New Probabilistic Meta-learning Approach to Few-shot Image Classification”. In: *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*. Proceedings of Machine Learning Research. 2021 (cit. on p. 2).

[50] Aston Zhang et al. “Dive into deep learning”. In: *arXiv preprint arXiv:2106.11342* (2021) (cit. on pp. 27, 33).

[51] Peyman Bateni, Jarred Barber, et al. “Enhancing Few-Shot Image Classification With Unlabelled Examples”. In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. Jan. 2022, pp. 2796–2805 (cit. on pp. 1, 2, 48).

[52] Kevin P Murphy. *Probabilistic machine learning: an introduction*. MIT press, 2022 (cit. on pp. 22, 23, 39).

[53] Christopher M Bishop. *Pattern recognition and machine learning*. Vol. 4. 4. Springer (cit. on p. 19).