Making the Best of What We Have: Novel Strategies for Training Neural Networks under Restricted Labeling Information

by

Sachin Chhabra

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved April 2024 by the
Graduate Supervisory Committee:

Baoxin Li, Chair
Hemanth Venkateswara
Yezhou Yang
Teresa Wu
Yingzhen Yang

ARIZONA STATE UNIVERSITY

May 2024

ABSTRACT

Recent advancements in computer vision models have largely been driven by supervised training on labeled data. However, the process of labeling datasets remains both costly and time-intensive. This dissertation delves into enhancing the performance of deep neural networks when faced with limited or no labeling information. I address this challenge through four primary methodologies: domain adaptation, self-supervision, input regularization, and label regularization.

In situations where labeled data is unavailable but a similar dataset exists, domain adaptation emerges as a valuable strategy for transferring knowledge from the labeled dataset to the target dataset. This dissertation introduces three innovative domain adaptation methods that operate at pixel, feature, and output levels. Another approach to tackle the absence of labels involves a novel self-supervision technique tailored to train Vision Transformers in extracting rich features. The third and fourth approaches focus on scenarios where only a limited amount of labeled data is available. In such cases, I present novel regularization techniques designed to mitigate overfitting by modifying the input data and the target labels, respectively.

DEDICATION

To my parents, Dr. Vijay Chhabra & Dr. Nanda Chhabra.

ACKNOWLEDGMENTS

This dissertation stands as a testament to the collective support, encouragement, and contributions of numerous individuals, each of whom has left a lasting mark on my academic journey.

First and foremost, I would like to express my deepest gratitude to my advisor, Dr. Baoxin Li, who has been not only a mentor but also a guiding light throughout my academic journey. I was a lost Master's student who was trying to learn about the field of machine learning and Dr. Li graciously took me under his wing. His unwavering guidance, invaluable insights, and strong support have been instrumental in shaping me into the researcher I am today.

I am also grateful to Dr. Hemanth Venkateswara, who treated me as one of his own students. Our collaborative brainstorming sessions and his insightful guidance have proven invaluable in overcoming obstacles and deepening my understanding of machine learning. Dr. Venkateswara's pivotal role in guiding me out of every encountered *local minima* throughout my PhD journey cannot be overstated.

I extend my gratitude to my dissertation committee, Dr. Yezhou Yang, Dr. Teresa Wu, and Dr. Yingzhen Yang, for their invaluable advice and thorough evaluation of my work.

Throughout this journey, numerous individuals have left a lasting impact, knowingly or unknowingly, contributing to the completion of this dissertation. It all began with my friend Diptanshu Purwar, who introduced me to the field of machine learning and encouraged me to pursue it wholeheartedly. Without his constant push, I might not have chosen this path. Additionally, I extend my gratitude to Mahfuzur Siddiquee for his invaluable assistance in solidifying this decision and for helping me strengthen the foundational concepts of deep learning. Their support and mentorship have been indispensable in shaping my academic trajectory.

I am deeply appreciative of my academic family, including Nupur Thakur, Riti Paul, Yaoxin Zhuo, Prasanth Sai Gouripeddi, Sahil Vora, Jiuxu Chen, Vijetha Gattupalli, Yuzhen Ding, Kevin Ding, Yikang Li, and Tianshu Yu. Their friendship, technical expertise and unwavering support have been invaluable throughout my journey. Working alongside you all has been a pleasure, and your assistance with various technical and non-technical challenges has been immensely valuable.

I extend my heartfelt gratitude to my friends - Shalini Gangadhara, Usha Ranganathaiah, Roopa Jayashankar, Akhil Sahdev, Varun Singh, Shashank Jindal, Shivam Bajaj, Kuldeep Singh, Shrey Mehta, Prabal Bijoy Dutta, Mayank Darbari, Vishwesh Nath, Anya Chaturvedi and Apurva Kharate. Also, a huge shoutout to my gaming crew - Harsh Kothari, Varun Baluni, Kurmaya Gari Sai Siddharth Reddy, Eshaan Mathew, Mohamad Mudassar Shekh, Santosh Bidve, Utkarsh Pandey and Abhishek Throat for the awesome memories and endless fun. Each one of you has provided me with invaluable moments of joy and relaxation, which have been essential in maintaining my sanity throughout these years.

Lastly, but above all, I want to express my deepest gratitude to my parents Dr. Vijay Chhabra, Dr. Nanda Chhabra, and my brother Dr. Nitin Chhabra for their unwavering support, love, and encouragement through both good and bad times. It is because of them that I stand here today, having completed my Ph.D. degree.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

Chapter 1

INTRODUCTION

Recent advancements in deep learning techniques have consistently achieved state-of-the-art performance across various tasks. However, this performance often hinges on the availability of labeled data, which can be expensive to obtain. While unlabeled data is abundant, the process of labeling it proves to be prohibitively costly. In this dissertation, I present innovative approaches aimed at training networks with limited labels.

The first chapter addresses the challenge of missing labels by domain adaptation, where the target dataset lacks labels, but a similar labeled dataset, known as the source, is available. Here, the focus is on transferring knowledge from the labeled dataset (source) to the unlabeled dataset (target) through domain adaptation. I discuss three pioneering domain adaptation techniques, each operating at pixel, feature, and output levels.

The second chapter addresses the challenge of missing labels by introducing a novel self-supervision technique specifically tailored for training Vision Transformers, enabling the extraction of intricate features.

Subsequent third and fourth chapters, confront situations where only a restricted amount of labeled data is available. In such contexts, regularization techniques prove indispensable in mitigating overfitting induced by small datasets. These chapters introduce innovative input regularization techniques, modifying both input data and target labels, respectively.

1.1 Domain Adaptation

I address the challenge of missing labels by domain adaptation, where the target dataset lacks labels but a similar labeled dataset, known as the source, is available. One can train a model on the labeled dataset (source) and use it on the unlabeled dataset (target). However, such models often underperform on target datasets. This is due to the underlying domain differences between the two datasets.

While one approach to address this is by training the model afterward on the target dataset, it demands labeling the target data, which can be costly. Domain adaptation techniques aim to mitigate this challenge by adapting the classifier to the target dataset without relying on target labels. Typically, the prevailing method involves classifying samples based on features that remain consistent or invariant across both the source and target domains. Here, the focus is on transferring knowledge from the labeled dataset to the unlabeled dataset through domain adaptation.

To attain this objective, most techniques attempt to reduce the distance between the generated features of these domains using distance metrics such as maximum mean discrepancy, Wasserstein distance, or adversarially using a discriminator. My work focuses on resolving this issue by reducing differences between the domains either at the pixel level, feature level, or target level.

1.2 Self-Supervised learning

In case a similar dataset is unavailable, self-supervision emerges as a powerful solution that can be used to train the network to extract generalizable features. The features learned by such a network are highly transferable and can be used for multiple tasks. Hence, self-supervised training of networks is increasingly gaining a lot of traction. These techniques train networks by formulating auxiliary tasks that necessitate an understanding of the presented objects. Consequently, the features extracted from such networks are not only rich but also highly adaptable to diverse tasks. My objective revolves around enhancing existing self-supervised techniques. I aim to contribute to the refinement and advancement of these methods, further optimizing the capability of networks to learn and extract intricate features without the dependency on annotated data.

1.3 Regularization

When a limited amount of labeled data is available, the neural networks often struggle with overfitting, leading to diminished generalization abilities. In such cases, a robust regularization is required to counter overfitting. Regularization techniques aim to alleviate this by imposing additional constraints on the network. These constraints are strategically crafted to aid and improve the generalization capabilities of the network. Various regularization constraints operate differently: some directly impact the network itself, such as dropout [Srivastava *et al.* (2014)] and weight decay [Hanson and Pratt (1988)], while others influence the input data, employing techniques like cutout [Devries and Taylor (2017)] and mixup [Zhang *et al.* (2018)]. I intend to create an innovative regularization method specifically designed for Vision Transformers that leverages its distinctive strengths to enhance its generalization capabilities.

1.4 Label Regularization

Another way to regularize a neural network is label regularization and such regularizations are independent of the network and the input modality. The most popular label regularization technique is Label Smoothing, which adjusts target labels during training by adding a uniform label distribution over the categories to the one-hot target [Szegedy *et al.* (2016a)]. Training with Label Smoothing has proven effective in enhancing generalization and has been widely adopted. Despite the advantages of Label Smoothing, it is known to disrupt the relationships between categories [Müller *et al.* (2019)]. I aim to develop a novel label regularization technique that utilizes the optimal training vectors to enhance inter-class relationships and improve network regularization.

Chapter 2

DOMAIN ADAPTATION

Domain adaptation techniques address situations where labels for a dataset are unavailable, but there exists another dataset similar to the target dataset. In such scenarios, a classifier can be trained using the available labeled dataset (referred to as the source) and then applied to the target dataset. However, models trained on the source dataset often perform poorly on the target dataset, despite their similarity. This performance gap is attributed to the covariance shift, which denotes the underlying distribution difference between the domains of the two datasets.

In response to this challenge, the conventional approach is to train the model directly on the target dataset. Nevertheless, in cases where labels are absent, unsupervised domain adaptation (UDA) techniques offer a convenient solution. These methods utilize knowledge from the source dataset to adapt the classifier to the target dataset without relying on target labels.

Domain adaptation methods typically involve classifying samples based on a common ground that is consistent or invariant across both the source and target domains. This common ground can be achieved by utilizing shared features between the two domains or by translating images from one domain to another. My research focuses on addressing this issue by minimizing differences between the domains at the feature level, pixel level, or output level. The overview of these approaches is in Figure 2.1.

Feature-level domain adaptation is the most popular type of domain adaptation. These techniques focus on creating features that are common between the source and target domain. Such invariant features are achieved by aligning features generated by networks for both the source and target datasets using distance metrics like MMD

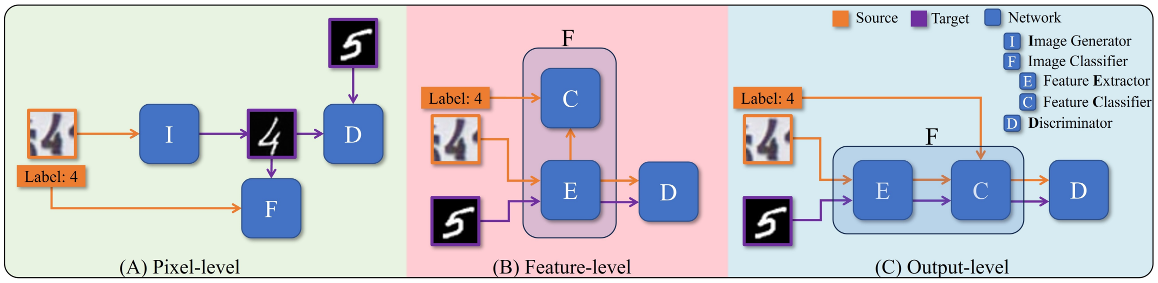


Figure 2.1: Variations types of domain adaptation approaches. **A.** Pixel-level alignment: Adversarial image translation is applied to translate source images into target images before using a common classifier. **B.** Feature-level alignment: The source and target features of a deep neural network are aligned before applying a common classifier. The figure depicts an adversarial feature alignment architecture. **C.** Generative Alignment of Posterior probabilities (GAP): I to align the posterior probabilities of the source and target classifiers using an adversarial framework. Although the image indicates the presence of a source, the GAP model can be used for both source-free and unsupervised domain adaptation.

or adversarial strategies. My work focuses on improving existing adversarial domain alignment techniques by performing domain alignment at the category levels resulting in a better global alignment.

Pixel-level domain adaptation methods translate images from one domain to domain to another. My work translates images from the source domain to the target domain. These translated images are used to learn a classifier on the target domain using translated images and source labels. My technique uses the classifier to guide the image translation process and uses translated images to guide the classifier iterative. I also combine feature-level adaptation with pixel-level adaptation for additional improvement. Here, domain alignment is performed among the features of three domains - source, source-to-target translated images, and target images. This trained classifier can classify both the source and target domain.

I showcase a new branch of domain adaptation called output-level domain adaptation which uses output probabilities of the network. Output/posterior probabilities show the relations of a sample to the other categories. My approach focuses on modeling this relation at category-level (inter-category relationships) and aligning them between source and target domain. This method can work with and without the source data. This can be helpful when dealing with sensitive data where only a trained model on source data is available and there is no access to the source data after the model has been trained. In this case, I designed a source replicator that is trained to mimic probabilities of different categories of the source domain. The source replicator aligns the posterior probabilities of the target to the source domain using the source replicator. This alignment results in an improved target-domain classifier. If source data is available, it omits the need for a source replicator. In this case, the posterior probabilities are directly aligned between the domains. This allows for additional components like label smoothing, and temperature scaling for further improvements.

2.1 Feature-Level

Feature-level domain adaptation aims to extract features that are common between the source and target domain. To achieve such invariant features these techniques attempt to reduce the distance such as maximum mean discrepancy, Wasserstein distance, or adversarially using a discriminator between the generated features of the domains. However, existing methods make the features domain-invariant but the alignment is done without considering the category information of the features and often leads to jumbled category features.

To overcome this limitation and refine adversarial global domain alignment, I present a novel technique called "Glocal alignment." Inspired by the ethos of "Think Globally, Act Locally," this method aligns data points locally by leveraging category information [Chhabra *et al.* (2021a)]. The process begins by partitioning samples from both the source and target domains based on their respective categories. However, in this scenario the target labels are unavailable. Hence, I employ pseudo-labels in place of the original label. Pseudo-labels is a technique where the network's predictions serve as labels if the predicted category probability surpasses a predefined threshold (as detailed in [Lee (2013)]). Subsequently, data points within each category from both domains are aligned. This local alignment within categories consequently ensures a more cohesive global alignment across domains. This straightforward yet impactful glocal adjustment can be seamlessly integrated into various adversarial domain alignment techniques, such as DANN [Ganin *et al.* (2016a)], MDC [Tzeng *et al.* (2015)], and GAN loss functions [Goodfellow *et al.* (2014)], yielding significantly improved alignment outcomes.

2.1.1 Background

Let $S = \{(x_i^s, y_i^s)\}_{i=1}^{N_s}$ be the source domain consisting of N_s labeled images sampled from distribution P_s. Likewise the target domain is denoted as $T = \{(x_i^t)\}_{i=1}^{N_t}$ consisting of N_t unlabeled images sampled from distribution P_t. The goal of unsupervised domain adaptation is to learn the target labels $\{y_i^t\}_{i=1}^{N_t}$ using S and T. It is assumed that S and T have an identical label space of K categories but since $P_s \neq P_t$, a classifier trained using S will underperform when trying to predict the target data labels.

I plan to align the source and target domains using adversarial feature alignment based on the Generative Adversarial Network (GAN) [Goodfellow *et al.* (2014)]. The standard GAN model consists of a Generator network $G(.)$, and a Discriminator network $D(.)$ which are pitted against each other in min-max optimization. Traditionally, the generator $G(.)$ takes noise as an input and outputs an image but in the case of unsupervised domain adaptation, the feature extractor E acts as the generator. It takes an image x as the input and output features $E(x)$ which can be classified by a classification module $C(.)$ to appropriate classes. $E(.)$ and $C(.)$ are representation of components of a traditional image classifier $F(.)$. The Discriminator network $D(.)$ attempts to distinguish between the $E(x^s)$ source and $E(x^t)$ target features and the feature extractor $E(.)$ attempts to align the features so that they are indistinguishable by the discriminator. Upon convergence, the marginal distributions of the source and target are said to be aligned. In the following, I outline 4 popular variants of adversarial alignment.

Vanilla GAN

The first model GAN_1 is the vanilla GAN model where the feature extractor $E(.)$ attempts to align the source and target features and the Discriminator $E(.)$ learns to distinguish between them. The objective function for this model is,

$$\begin{aligned}&\min_{D} -\mathbb{E}_{x\sim S}[\ln D(E(x))] - \mathbb{E}_{x\sim T}[\ln(1-D(E(x)))]\\ &\min_{E,C} -\lambda_{g_1}\,\mathbb{E}_{x\sim T}[\ln D(E(x))] + \mathcal{L}^{s}_{ce},\end{aligned} \tag{2.1}$$

where $\mathcal{L}_{ce} = -\mathbb{E}_{(x,y)\sim S}[y \cdot \ln C(E(x))]$ is the cross entropy loss and λ_{g_1} models the importance of the alignment loss. In Eq. 2.1, the feature extractor attempts to align target features to a fixed source distribution. Alternatively, better alignment can be done in feature space by modifying both the source and target features [Shu *et al.* (2018)]. While the objective function for training $D(.)$ is the same as in Eq. 2.1, the objective function to train the feature extractor $E(.)$ and Classifier $C(.)$ is,

$$\min_{G,C} -\lambda_{g_2}\big\{\,\mathbb{E}_{x\sim S}[\ln(1-D(E(x)))] + \mathbb{E}_{x\sim T}[\ln D(E(x))]\big\} + \mathcal{L}^{s}_{ce} \tag{2.2}$$

I refer to Eq. 2.1 as GAN$_1$ and to Eq. 2.2 as GAN$_2$.

DANN

My third model is the popular Domain-Adversarial Training of Neural Networks (DANN) [Ganin *et al.* (2016b)]. In DANN the feature extractor $E(.)$ uses a reversed gradient to update parameters during training in order to confuse the Discriminator $D(.)$. The objective function for the feature extractor differs from Eq. 2.1 as,

$$\min_{G,C} -\lambda_d \big\{ \mathbb{E}_{x\sim S}[\ln D(E(x))] + \mathbb{E}_{x\sim T}[\ln(1 - D(E(x)))] \big\} + \mathcal{L}^s_{ce}. \tag{2.3}$$

MDC

The 4th model for comparison is the Maximum Domain Confusion (MDC) [Tzeng *et al.* (2015)]. The MDC introduces maximum domain confusion through a cross-entropy loss between the output of the discriminator and the uniform distribution. This results in even the best discriminator performing poorly, thereby aligning the domains. The loss function for the Discriminator is identical to the one in Eq. 2.1. The objective function for the feature extractor and Classifier is given by,

$$\min_{G,C} -\lambda_m \big\{ \mathbb{E}_{x\sim(S\cup T)} \big[\frac{1}{2} \ln D(E(x)) + \frac{1}{2} \ln(1 - D(E(x)))\big] \big\} + \mathcal{L}^s_{ce}. \tag{2.4}$$

The summary of these loss functions is presented in Table 2.1.

Method	**Discriminator Loss**	**Feature extractor Loss**
DANN	$-\mathbb{E}_{x\sim S}[\ln D(E(x))] - \mathbb{E}_{x\sim T}[\ln(1 - D(E(x)))]$	Gradient Reversal
MDC	$-\mathbb{E}_{x\sim S}[\ln D(E(x))] - \mathbb{E}_{x\sim T}[\ln(1 - D(E(x)))]$	$-\mathbb{E}_{x\sim(S\cup T)}[\frac{1}{2}\ln D(E(x)) + \frac{1}{2}\ln(1 - D(E(x)))]$
GAN_1	$-\mathbb{E}_{x\sim S}[\ln D(E(x))] - \mathbb{E}_{x\sim T}[\ln(1 - D(E(x)))]$	$-\mathbb{E}_{x\sim T}[\ln D(E(x))]$
GAN_2	$-\mathbb{E}_{x\sim S}[\ln D(E(x))] - \mathbb{E}_{x\sim T}[\ln(1 - D(E(x)))]$	$-\mathbb{E}_{x\sim S}[\ln(1 - D(E(x)))] - \mathbb{E}_{x\sim T}[\ln D(E(x))]$

Table 2.1: Traditional Domain Alignment loss functions.

2.1.2 Glocal Method

Traditional domain alignment aligns the deep features without any regard to their category. Such alignment generally results in aligning mismatched classes and hurts target performance. To overcome this issue, I present Glocal alignment which splits data into classes and performs category-level (local) adversarial alignment. Since in the case of unsupervised domain adaptation target labels are not available, I rely on the pseudo label generated by the network. This way Glocal alignment achieves global alignment by aligning data points from each category for the two domains.

First, The classifier is trained on the source dataset S using standard cross-entropy loss $\mathcal{L}^s_{ce}$. Next, I determine the pseudo-labels for the target data by applying a threshold on the classifier prediction,

$$\hat{y}^t_i = \begin{cases} \underset{y}{\operatorname{argmax}}\, p(y|C(x^t_i)), & \text{if } \max\, p(y|C(E(x^t_i))) > \tau \\ -1 & \text{otherwise.} \end{cases} \tag{2.5}$$

I define $\bar{D}^t := \{(x^t_i, \hat{y}^t_i)\}^{n'_t}_{i=1} | \hat{y}^t_i \neq -1\}$ as the pseudo-labeled target dataset. To apply alignment on each class, K discriminators would be needed and the amount of data available to train each discriminator will also reduce by a factor of K, making this approach impractical when K is large. To address this concern, I use a multi-task learning approach and modify the discriminator to multi-headed logit [Ruder (2017)]. Specifically, I change the number of outputs of the discriminator from 1 (Global) to K and each output head acts as a decision function for one of K categories. D_i represents the sigmoid output of the multi-headed discriminator at the i^{th} head. The glocal discriminator loss is defined as,

$$\min_D -\mathbb{E}_{x\sim S}[\sum_{i=1}^{K} \mathbb{1}(i=y) \ln D^i(E(x))] - \mathbb{E}_{x\sim T}[\sum_{i=1}^{K} \mathbb{1}(i=\hat{y}^t) \ln(1-D^i(E(x)))] \tag{2.6}$$

Another issue that arises is the class imbalance problem due to the use of a subset

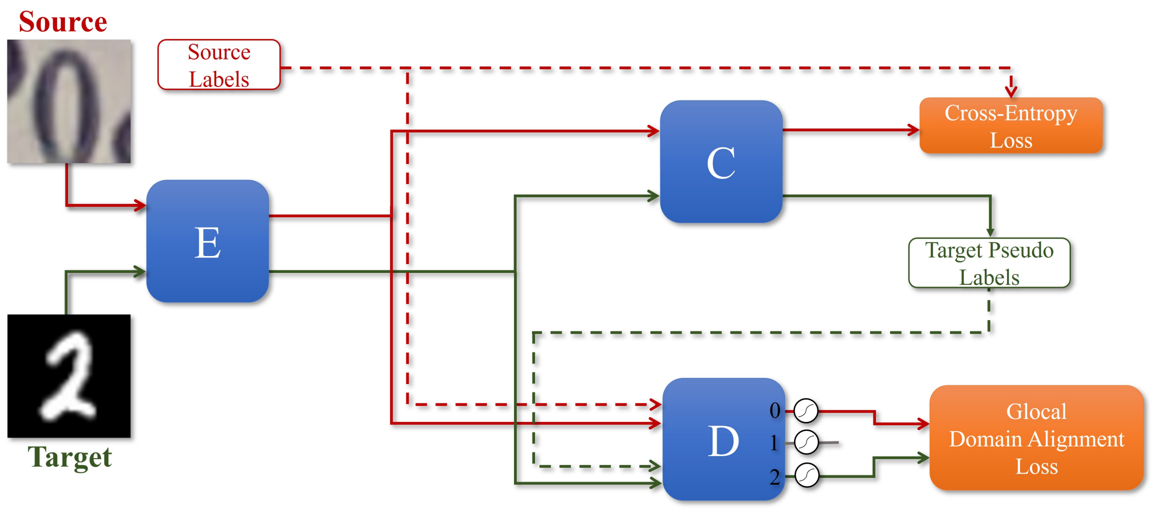


Figure 2.2: Model diagram of Glocal alignment. The flow of the source is denoted in **Red** and target using **Green**. The dotted lines indicate the flow of the labels. The feature extractor E generates features that are classified by the classifier C and Glocally aligned by the discriminator D. The classifier C is trained using the cross-entropy loss on the source dataset and generates pseudo-labels for target samples. The discriminator D is a multi-headed binary classifier and is trained using the Glocal domain-alignment loss function. The discriminator head D^i to train is selected using the input label and trained to classify between source and target's features belonging to i^{th} category. The feature extractor E uses both Glocal domain-alignment loss and cross-entropy loss for training. It is trained to fool the discriminator and minimize the source cross-entropy loss on the classification task.

of the target samples only. The reduced number of target samples introduces a bias in the discriminator towards the source domain. To overcome this issue, I do not use the threshold τ while training the discriminator. The discriminator heads are trained using all the target samples T based on their pseudo labels, enabling the discriminator D to learn the actual distributions without any bias. Only confident samples T' are used for training the feature extractor E which are aligned with the source. Similar to the discriminator, I modify the feature extractor loss functions by adding the condition to select the appropriate discriminator head and use T' instead of T. Using these simple adjustments, any global alignment technique can be used at a local level.

The glocal loss functions are in Table 2.2. The discriminator head is selected based on the source label and target pseudo label. The feature extractor is trained with the cross-entropy loss along with the glocal feature extractor loss. The weighing coefficients for the feature extractor are the same as the traditional alignment losses.

2.1.3 Experiments and Results

Datasets

I test my approach on the following classification tasks:

Digits and Traffic Signs: For digits experiments, I use 5 datasets - SVHN, MNIST, USPS, MNIST-M, and Synthetic digits (SyDigits). All the digits datasets have 10 classes and present various visual variations in the images. Synthetic Signs (SynSigns) and GTSRB are traffic sign datasets containing 43 classes.

I test my approach on the standard tasks: MNIST $\leftrightarrow$ USPS, MNIST $\rightarrow$ MNIST-M, SVHN $\rightarrow$ MNIST, SynDigits $\rightarrow$ SVHN and SynSigns $\rightarrow$ GTSRB. Digits and Traffic sign images are scaled to 32×32 using bilinear interpolation and normalized to be in the range $[-1, 1]$. The E(Small) network is trained for these tasks.

Office-31 [Saenko *et al.* (2010)] is a real-world object classification dataset that contains 31 classes in three different domains – Amazon, DSLR, and Webcam. The experiments are conducted using ResNet-50 pretrained on ImageNet. Standard random-crop and horizontal flips are applied for training and center-crop for testing.

Method	**Discriminator Loss**	**Feature extractor Loss**
$\mathcal{G}$-$DANN$	$-\mathbb{E}_{x\sim S}[\sum_{i=1}^{K} \mathbb{1}(i=y)\ln D^i(E(x))]$ $-\mathbb{E}_{x\sim T}[\sum_{i=1}^{K} \mathbb{1}(i=\hat{y}^t)\ln(1-D^i(E(x)))]$	Gradient Reversal using $-\mathbb{E}_{x\sim S}[\sum_{i=1}^{K} \mathbb{1}(i=y)\ln D^i(E(x))]$ $-\mathbb{E}_{x\sim T'}[\sum_{i=1}^{K} \mathbb{1}(i=\hat{y}^t)\ln(1-D^i(E(x)))]$
$\mathcal{G}$-MDC	$-\mathbb{E}_{x\sim S}[\sum_{i=1}^{K} \mathbb{1}(i=y)\ln D^i(E(x))]$ $-\mathbb{E}_{x\sim T}[\sum_{i=1}^{K} \mathbb{1}(i=\hat{y}^t)\ln(1-D^i(E(x)))]$	$-\mathbb{E}_{x\sim(S\cup T')}[\frac{1}{2}\sum_{i=1}^{K} \mathbb{1}(i=y)\ln D^i(E(x))$ $+\frac{1}{2}\sum_{i=1}^{K} \mathbb{1}(i=y)\ln(1-D^i(E(x)))]$
$\mathcal{G}$-GAN_1	$-\mathbb{E}_{x\sim S}[\sum_{i=1}^{K} \mathbb{1}(i=y)\ln D^i(E(x))]$ $-\mathbb{E}_{x\sim T}[\sum_{i=1}^{K} \mathbb{1}(i=\hat{y}^t)\ln(1-D^i(E(x)))]$	$-\mathbb{E}_{x\sim T'}[\sum_{i=1}^{K} \mathbb{1}(i=\hat{y}^t)\ln D^i(E(x))]$
$\mathcal{G}$-GAN_2	$-\mathbb{E}_{x\sim S}[\sum_{i=1}^{K} \mathbb{1}(i=y)\ln D^i(E(x))]$ $-\mathbb{E}_{x\sim T}[\sum_{i=1}^{K} \mathbb{1}(i=\hat{y}^t)\ln(1-D^i(E(x)))]$	$-\mathbb{E}_{x\sim S}[\sum_{i=1}^{K} \mathbb{1}(i=y)\ln(1-D^i(E(x)))]$ $-\mathbb{E}_{x\sim T'}[\sum_{i=1}^{K} \mathbb{1}(i=\hat{y}^t)\ln D^i(E(x))]$

Table 2.2: Glocal Domain Alignment loss functions.

Office-Home [Venkateswara *et al.* (2017)] is another challenging real-world object classification dataset with 4 domains - Art (Ar), Clipart (Cl), Product (Pr) and Real-world (Rw) images. It consists of 65 different categories from Office and Home settings. ResNet-50 pretrained on ImageNet is used for these experiments. Standard random-crop and horizontal flips are applied for training and center-crop for testing.

E (Small): `K(32)`→`K(32)`→`P(2,2)`→`K(64)`→ `K(64)`→`K(128)`→`K(128)`

`K(128)`→`K(128)`→`P(2,2)`→`FC(128)`

E (Office-31): `ResNet50`→`FC(512)`

E (Office-Home): `ResNet50`→`FC(512)`

D(*Global*): `FC(500)`→`FC(500)`→`FC(1)`

D(*Glocal*): `FC(500)`→`FC(500)`→`FC(`K`)`

C: `FC(`K`)`

Training Setup

To ensure a fair comparison, I train all the alignment loss approaches (including the baselines) using the above architectures. textttK(n) represents `n` kernels of size 3 with padding 1, `P(2,2)` is a max pool with kernel size 2 and stride 2. `FC(n)` is a fully connected layer with `n` neurons. Classifier C and Discriminator D use the output of the feature extractor E as the input. Discriminator uses 1 neuron for global alignment and K neurons for local alignment. I use ReLU activation in the feature extractor and Softmax activation for the classifier. The discriminator uses Leaky ReLU with α = 0.2 in the hidden layers and Sigmoid activation for the output.

Method	A→W	D→W	W→D	A→D	D→A	W→A	Mean
Source	68.4	96.7	99.3	68.9	62.5	60.7	76.1
DANN	82.5	97.6	99.6	81.5	67.9	71.7	83.5
$\mathcal{G}$-DANN	**92.6**$_\uparrow$	97.5$_\downarrow$	99.7$_\uparrow$	**89.8**$_\uparrow$	69.8$_\uparrow$	72.6$_\uparrow$	87.0$_\uparrow$
MDC	87.0	97.4	**99.8**	83.3	70.0	72.7	85.0
$\mathcal{G}$-MDC	90.4$_\uparrow$	**98.5**$_\uparrow$	**99.8**	88.6$_\uparrow$	**73.7**$_\uparrow$	72.8$_\uparrow$	**87.3**$_\uparrow$
GAN$_1$	85.5	97.1	**99.8**	84.3	68.4	72.4	84.6
$\mathcal{G}$-GAN$_1$	90.1$_\uparrow$	97.9$_\uparrow$	**99.8**	88.6$_\uparrow$	69.1$_\uparrow$	72.9$_\uparrow$	86.4$_\uparrow$
GAN$_2$	85.5	97.2	**99.8**	83.1	69.7	72.8	84.7
$\mathcal{G}$-GAN$_2$	92.0$_\uparrow$	98.2$_\uparrow$	**99.8**	88.2$_\uparrow$	72.6$_\uparrow$	**73.2**$_\uparrow$	**87.3**$_\uparrow$

Table 2.3: Target classification accuracies for Office-31 using ResNet-50. The Glocal model is denoted as $\mathcal{G}$-(model).

The E(Small) and E(Office-31 & Office-Home) are trained with a batch size of 128 and 36 respectively. I use the Adam optimizer with 10^{-4} learning rate. The learning rate for pretrained layers of ResNet-50 is set to 10^{-5} to ensure smooth fine-tuning. I set $\lambda_{g_1} = \lambda_{g_2} = \lambda_m = 0.01$, $\lambda_d = 1$ and $\tau = 0.9$ for all my experiments.

Target Classification Accuracy

My experiment results are shown in Table 2.4, 2.3 and 2.5. In all cases, the Glocal alignment outperforms the category-agnostic global alignment. For the small-resolution image experiments, the performance gain is the highest in the case of SVHN → MNIST, which is the hardest adaptation task among them. In the case of Office-31 and Office-Home experiments, Glocal alignment improves over global alignment with an average increase of 2.5%.

Method	MNIST→USPS	USPS→MNIST	MNIST→MNISTM	SVHN→MNIST	SyDigits→SVHN
Source	81.4	54.0	59.3	64.8	86.2
DANN	93.5	96.2	83.2	75.3	91.7
$\mathcal{G}$-DANN	$96.7_{\uparrow}$	$97.3_{\uparrow}$	**85.3**$_{\uparrow}$	$88.2_{\uparrow}$	$92.8_{\uparrow}$
MDC	96.3	97.9	–	72.8	91.9
$\mathcal{G}$-MDC	$97.3_{\uparrow}$	$98.4_{\uparrow}$	–	**89.7**$_{\uparrow}$	$92.8_{\uparrow}$
GAN_1	96.8	98.1	81.7	65.9	92.3
$\mathcal{G}$-GAN_1	$97.1_{\uparrow}$	98.1	$82.0_{\uparrow}$	$88.9_{\uparrow}$	$93.2_{\uparrow}$
GAN_2	97.1	98.3	81.9	73.1	92.5
$\mathcal{G}$-GAN_2	**97.2**$_{\uparrow}$	**98.7**$_{\uparrow}$	$83.0_{\uparrow}$	$89.1_{\uparrow}$	**93.4**$_{\uparrow}$

Table 2.4: Target classification accuracies on Digits and Traffic-Signs. The Glocal model is denoted as $\mathcal{G}$-(model). ("–" did not converge.)

Source	Ar			Cl			Pr			Rw			Mean
Target	Cl	Pr	Rw	Ar	Pr	Rw	Ar	Cl	Rw	Ar	Cl	Pr	
Source	34.9	50.0	58.0	37.4	41.9	46.2	38.5	31.2	60.4	53.9	41.2	59.9	46.1
DANN	48.3	63.5	73.2	56.3	65.4	64.4	53.7	50.0	72.9	68.5	55.6	80.5	62.7
$\mathcal{G}$-DANN	$\mathbf{55.6}_\uparrow$	$68.0_\uparrow$	$75.7_\uparrow$	$\mathbf{61.6}_\uparrow$	$68.6_\uparrow$	$\mathbf{72.0}_\uparrow$	$\mathbf{58.8}_\uparrow$	$\mathbf{55.5}_\uparrow$	$78.4_\uparrow$	$\mathbf{70.8}_\uparrow$	$\mathbf{58.8}_\uparrow$	$\mathbf{82.6}_\uparrow$	$\mathbf{67.2}_\uparrow$
MDC	48.3	67.8	74.7	56.7	65.4	65.6	56.6	51.6	74.0	68.8	58.6	81.1	64.1
$\mathcal{G}$-MDC	$55.3_\uparrow$	$\mathbf{68.5}_\uparrow$	$75.5_\uparrow$	$57.9_\uparrow$	$\mathbf{69.0}_\uparrow$	$70.5_\uparrow$	$56.2_\downarrow$	$54.0_\uparrow$	$78.2_\uparrow$	$69.9_\uparrow$	$\mathbf{58.8}_\uparrow$	$82.1_\uparrow$	$66.3_\uparrow$
GAN_1	48.5	67.0	74.8	57.1	65.9	66.7	54.8	53.8	75.4	69.6	57.9	79.8	64.3
$\mathcal{G}$-GAN_1	$53.5_\uparrow$	$66.5_\downarrow$	$75.3_\uparrow$	$57.4_\uparrow$	$67.5_\uparrow$	$68.9_\uparrow$	$55.0_\uparrow$	$54.5_\uparrow$	$75.8_\uparrow$	$69.8_\uparrow$	$57.5_\downarrow$	$80.8_\uparrow$	$65.2_\uparrow$
GAN_2	48.4	67.4	74.7	56.5	66.0	66.7	55.6	52.9	74.5	68.5	58.1	80.4	64.1
$\mathcal{G}$-GAN_2	$55.3_\uparrow$	$68.3_\uparrow$	$\mathbf{75.9}_\uparrow$	$58.3_\uparrow$	$68.2_\uparrow$	$71.4_\uparrow$	$57.4_\uparrow$	$54.7_\uparrow$	$\mathbf{78.6}_\uparrow$	$69.9_\uparrow$	$58.4_\uparrow$	$81.7_\uparrow$	$66.5_\uparrow$

Table 2.5: Results of Glocal domain alignment on Office-Home. The Glocal model is denoted as $\mathcal{G}$-(model).

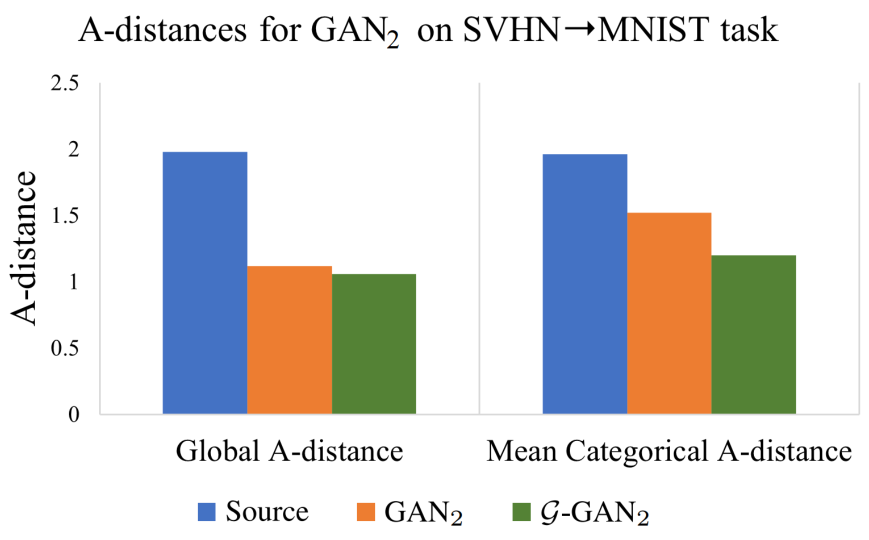


Figure 2.3: $\mathcal{A}$-distances for $\mathcal{G}$-GAN$_2$ on SVHN$\rightarrow$ MNIST task.

2.1.4 Analysis

A-distance

$\mathcal{A}$-distance is a metric to measure the domain gap, defined as $2\times(1-\varepsilon)$ where ε is the generalization error of a classifier trained to distinguish the features of the domains [Ben-David *et al.* (2010)]. I perform 5-fold cross-validation using a linear SVM for GAN$_2$ on SVHN $\rightarrow$ MNIST transfer task. As depicted in Fig. 2.3, Global and Glocal alignments achieve similar low $\mathcal{A}$-distance (Fig. 2.3 (left)), which signifies that domains are well-aligned. However, when compared using the mean $\mathcal{A}$-distance for each category (Fig. 2.3 (right)), I observe a significant domain gap.

Pseudo Label Accuracy

In almost all cases, the pseudo-label accuracy of the mini-batch was similar to the mini-batch accuracy. Even with moderate pseudo-label accuracy, the Glocal method

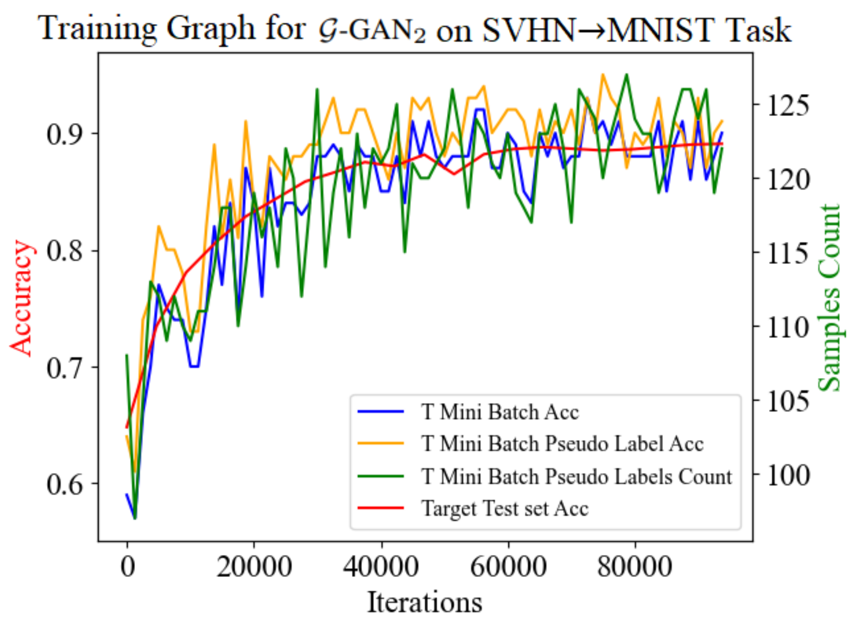


Figure 2.4: Training graphs comparing pseudo-label and target accuracies for $\mathcal{G}$-GAN$_2$ on SVHN$\rightarrow$ MNIST task.

achieves excellent performance. Fig. 2.4 shows training progress comparing pseudo-label accuracy with target accuracy for $\mathcal{G}$-GAN$_2$ on SVHN$\rightarrow$MNIST task.

Domain Alignment Feature Visualization

I use TSNE [Maaten and Hinton (2008)] plots to visualize feature alignment of the Glocal model for the SVHN$\rightarrow$MNIST task in Fig. 2.5. Global alignment mixes the two domains well but also misaligns the individual categories, whereas my approach provides better alignment for individual categories along with the global alignment.

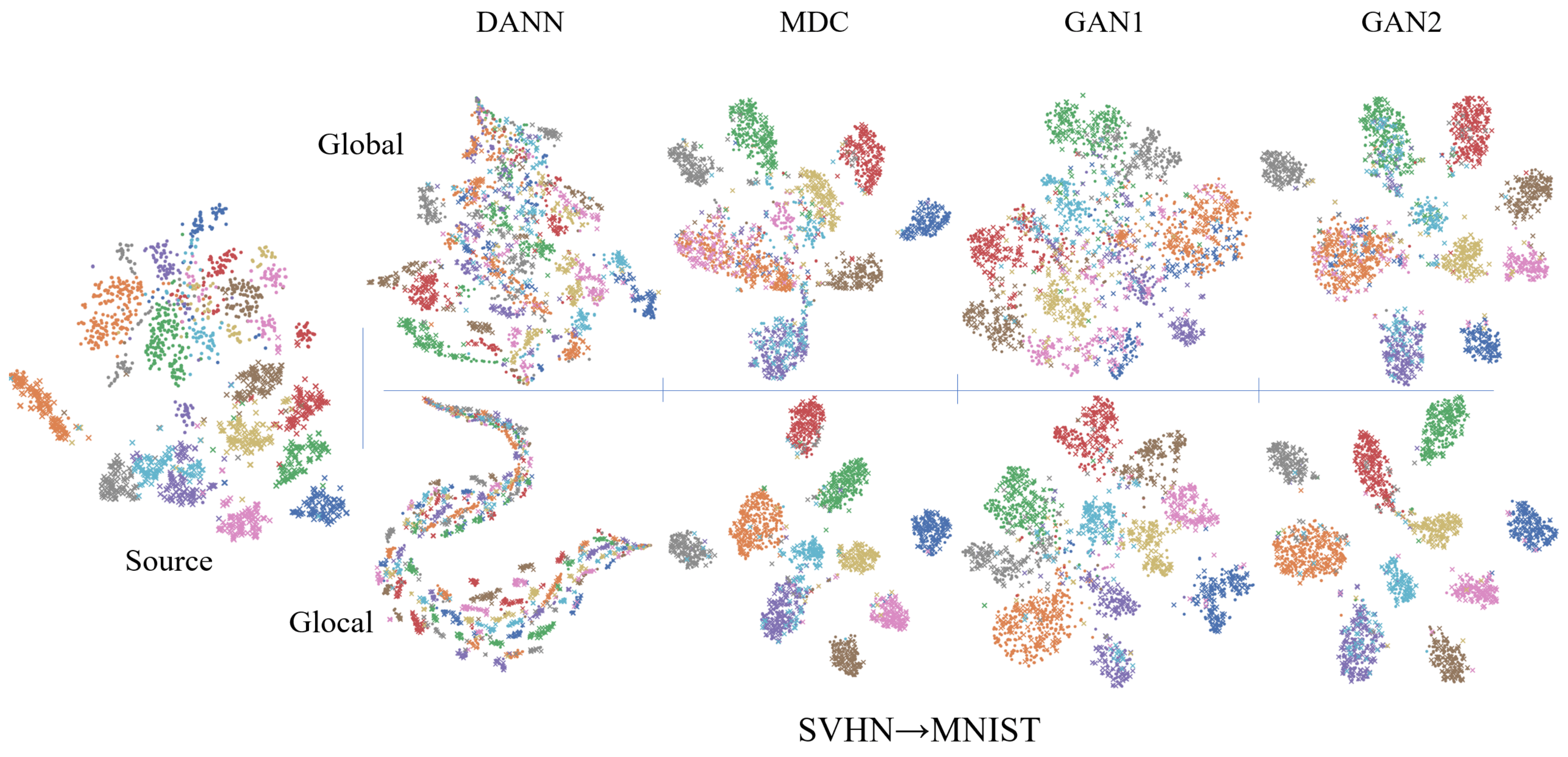


Figure 2.5: TSNE plots for SVHN→MNIST task. Each color represents a class. Source is represented by ● and target by +.

2.2 Pixel-Level Adaptation

Alternatively, domain adaptation can also be accomplished at the pixel level using image translation where images in one domain are translated to resemble images from the other domain. But pixel-level adaptation works well only on simple problems like digit classification because the images have limited variations which are nearly all captured in the datasets. Also, digit datasets from different domains vary mostly in the background with small changes to the foreground. On the other hand, real-world object classification datasets for domain adaptation have limited variations of the objects captured in the datasets. The intra-domain variations in terms of background are diverse and the inter-domain differences between the foregrounds (objects) are large [Saenko *et al.* (2010); Venkateswara *et al.* (2017)]. For effective pixel-based domain adaptation, large datasets are required that capture all the variations of the object, or the domains need to be very close. Due to these reasons, pixel-based domain adaptation approaches are mostly limited to digits, traffic signs, and segmentation datasets [Bousmalis *et al.* (2017); Hoffman *et al.* (2018); Russo *et al.* (2018)].

Standard approaches in image-translation-based unsupervised domain adaptation area use coupled generator-discriminator pairs from a GAN framework for image translation [Zhu *et al.* (2017)]. While one GAN translates source images to the target, another GAN translates target images to the source. There is a cyclic loss to ensure consistency in translation [Hoffman *et al.* (2018); Russo *et al.* (2018); Murez *et al.* (2018)]. The consistency loss however does not preclude the image from changing its category upon translation. For example, an image of digit 5 (source domain) can get converted to digit 7 in the target domain and back to digit 5 in the source domain all the while satisfying the cyclic consistency loss. The translated images are also of poor quality because the cyclic loss forces the network to embed non-relevant information

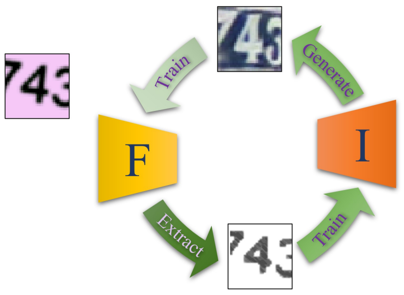


Figure 2.6: The learning paradigm of Iterative Image Translation (IIT). The image classifier F extracts content from the input source image. Using the content extracted from the classifier, the image generator I is trained to generate target-like images. In the next phase, the target-like generated images are used to train the classifier on the target domain. The cyclic process continues until convergence.

like background and style into the translated image to be able to reconstruct it later.

I showcase an approach "Iterative image translation" to overcome these problems with a single GAN framework that translates source images to the target domain [Chhabra *et al.* (2021b)]. The classifier module of my model retains the content (category) information from the source image into the translated target images using a content consistency loss. The generator in my GAN framework ensures the translated image appears to be from the target domain while preserving the content of the source image. In an iterative process, I train the classifier on the generated target images along with source labels, making it a better content extractor with each update. This way the classifier and the generator are used to train each other iteratively. Since there is no need for cyclic translation, the need to preserve the domain-specific content is eliminated while the generator can introduce target-specific content resulting in superior quality in the translated images. The procedure is depicted in Fig. 2.6.

The contributions of my model are as follows: (1) A novel image translation framework using a classifier to guide the image generator and vice-versa, (2) a content-consistency loss to retain source information in the translated image, (3) a three-way discriminator loss to align the features of the source, target and target-like source images giving more leeway to the GAN framework in the alignment space, (4) extensive empirical and subjective analysis to demonstrate the superiority of my translation framework.

2.2.1 Iterative Image Translation

Problem Statement

Let $S = \{x_i^s, y_i^s\}_{i=1}^{N_s}$ denote the source dataset, where N_s is number of samples drawn from the source distribution p_s and let $T = \{x_t^i\}_{i=1}^{N_t}$ be the target dataset, where N_t is number of samples drawn from the target distribution q_t. The goal is to learn the target labels $\{y_t^i\}_{i=1}^{N_t}$ using S and T. It is known that S and T share the same number of K classes but $p_s \neq q_t$ which is why a classifier trained on S is not sufficient to predict the target labels. To solve this problem, I use an image classifier F, a feature-level discriminator D_f, and a set of Image generator I and discriminator D_p. I pass the Source dataset S from the image generator I to generate fake target dataset $I(S) = \{I(x_i^s), y_i^s\}_{i=1}^{N_s}$.

I present an iterative approach involving two phases, A and B, to translate a source image to a target-like image using a single GAN framework. In Phase A, I train an image generator to generate a target-like image using a source image as input while retaining only the content (category) information from the source image. In Phase B, I train a classifier to extract the content information from the input images to be used for generation. I further outline the steps of my approach procedure below.

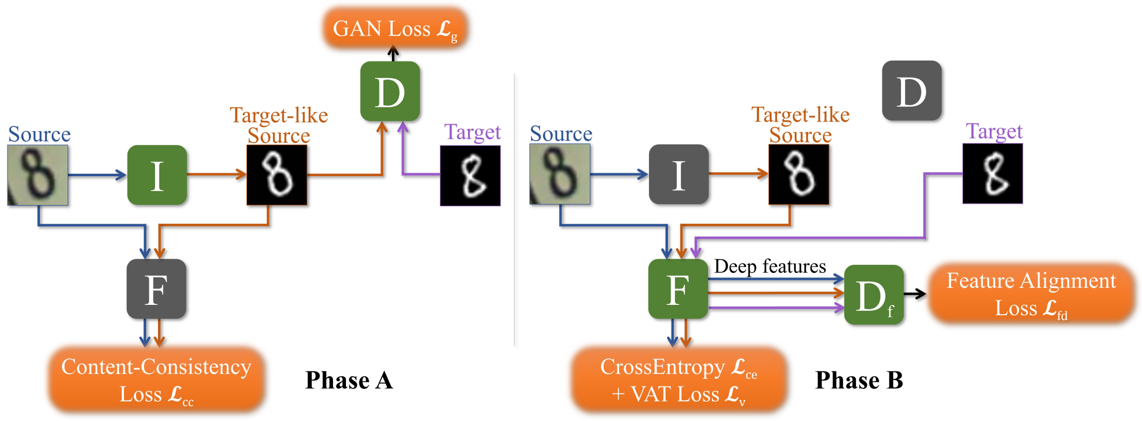

GAN Loss $\mathcal{L}_g$
D
Source
I
Target-like Source
Target
F
Content-Consistency Loss $\mathcal{L}_{cc}$
Phase A
D
Source
I
Target-like Source
Target
F
Deep features
D_f
Feature Alignment Loss $\mathcal{L}_{fd}$
CrossEntropy $\mathcal{L}_{ce}$ + VAT Loss $\mathcal{L}_v$
Phase B

Figure 2.7: Model Diagram of Iterative Image Translation. Green denotes the network is trained and Gray denotes the network is not trained during that phase. During Phase-A, Image Translator I is trained to minimize GAN loss $\mathcal{L}_g$ along with Content-Consistency loss $\mathcal{L}_{cc}$. During Phase-B, the source and the generated target-like source image with source labels are used to train the image classifier F using cross-entropy loss $\mathcal{L}_{ce}$ and Virtual Adversarial Training loss $\mathcal{L}_v$. Additionally, the deep features of the source, target-like source, and target images are aligned using feature alignment loss $\mathcal{L}_{fd}$.

Content Consistency Loss

I begin with an image classifier module F trained with standard cross-entropy loss on the labeled source images S,

$$\mathcal{L}^s_{ce} = -\,\mathbb{E}_{\{x^s,y^s\}\sim S}[y \cdot \log F(x^s)]. \tag{2.7}$$

I retain content information from the source image using the principle of consistency regularization, which is a regularization technique from semi-supervised learning [Sajjadi *et al.* (2016)]. I would like the source image and its corresponding target-like transformed image to have the same content. Content in this context refers to the category of the image. Having extracted the source content from the classifier in the form of a probability distribution over the categories, $F(x)$, I train the generator $I(.)$ of my GAN to retain this content information in the generated target-like image with the content consistency loss,

$$\mathcal{L}_{cc} = \mathbb{E}_{x \sim S}\left[||F(x) - F(I(x))||_2 \right]. \tag{2.8}$$

Matching the probability distributions alone will not result in identical content for the source image and its corresponding target-like image because the generated images can achieve the same probability distribution adversarially as well. To avoid this, I train the classifier using Virtual Adversarial Training [Miyato *et al.* (2018)] by minimizing the Kullback-Leibler divergence over the classifier predictions using a tiny perturbation $r < \epsilon$,

$$\mathcal{L}_v^s = \mathbb{E}_{x \sim S}[\max_{||r||<\epsilon} D_{KL}(F(x)||F(x+r))]. \tag{2.9}$$

Image Generation

For training the generator to produce real-looking target-like images, I use the least squared loss [Mao *et al.* (2017)] GAN objective along with the content consistency loss. The discriminator D is trained to distinguish between fake target $I(x^s)$ images generated by the image generator I and real target images x^t. The image generator I inputs a source image along with a noise vector $z \in \mathcal{N}(0, I)$ to fool the discriminator D. The input noise vector is sampled from a random Gaussian distribution and is up-scaled to the image size using a linear layer. It is then added as a new channel in the input image. The noise vector z provides the seed for the variations the generator produces in the target-like images,

$$\mathcal{L}_g = \mathbb{E}_{x\sim T}[(D(x)-1)^2] + \mathbb{E}_{\substack{x\sim S,\\ z\in\mathcal{N}(0,I)}}[(D(I(x,z)))^2]. \tag{2.10}$$

Learning Target Domain

Although the generated target images are constrained to have the same content as the source images, all of the images might not have content preserved. Therefore, select a subset of the most content-preserving generated images using content-consistency loss as a filter with a threshold τ,

$$S' := \big\{x_s \ni \mathcal{L}_{cc}(x^s) < \tau\big\}. \tag{2.11}$$

Initially, the classifier was trained using only the source images. The classifier can now be trained to classify target images using the generated target-like images. Although the generated images are not the actual target images, they are the best representation of the target domain learned by the generator given the constraints. These images have the same content as that of the source and are the closest representation to a labeled target domain. I exploit this fact and further train the classifier on the filtered generated images along with source labels so that it learns to classify target-like images. I train the classifier on these subsets of generated samples along with source labels using cross-entropy loss. I add VAT loss to it as well for the same reason as the source dataset,

$$\mathcal{L}_v^t = \mathbb{E}_{\substack{x\sim S',\\ z\in\mathcal{N}(0,1)}}[\max_{||r||<\epsilon} D_{KL}(F(I(x,z))||F(I(x)+r))], \tag{2.12}$$

$$\mathcal{L}_{ce}^t = -\,\mathbb{E}_{\{x,y\}\sim S'}[y\cdot\log F(I(x))]. \tag{2.13}$$

When the classifier is trained on the generated samples, it helps the generator to produce images that are even closer to the target domain. This way, I train the generator and the classifier alternatively and both the networks can learn from each other. With each update, the generator produces better target-like images and the classifier learns more about the target from those images.

Feature Alignment

The iterative pixel-level training does not guarantee that the classifier will have good accuracy on the actual target images because the generated images may not represent the entire target distribution [Hoffman *et al.* (2018)]. The pixel-level alignment alone can be achieved easily by aligning the generator to only a subset of the target domain (partial mode collapse). Also, the content variations in the target may not be entirely represented in the source images. Hence, I integrate a feature-level alignment loss to align the deep features of the domains. I consider target-like source images as a separate domain and implement domain alignment for all three domains - source, target-like source, and target,

$$\mathcal{L}_{fd} = -\mathbb{E}_{x\sim S}\log D_f^1(E(x)) - \mathbb{E}_{\substack{x\sim S' \\ z\in\mathcal{N}(0,I)}}\log D_f^2(E(I(x,z))) - \mathbb{E}_{x\sim T}\log D_f^3(E(x)), \tag{2.14}$$

where E is the submodule of the image classifier F till the penultimate layer and extracts the deep features. D_f^i is a 3-class classification network and $D_f^i(x)$ is the probability of x belonging to i^{th} class. My ternary feature alignment is similar to [Taigman *et al.* (2016)]. While training the discriminator, I use the same loss but while training the feature extractor, I maximize the loss with respect to all the domains instead of aligning them all to one domain. Eq. 2.14 helps in the further alignment of the distributions. The content extractor starts with knowing only about the source domain and ends up becoming an expert predictor of the target domain. The overall objective function brings together the cross-entropy loss ($\mathcal{L}_{ce}$), the virtual adversarial loss ($\mathcal{L}_v$), feature alignment ($\mathcal{L}_{fd}$), content consistency ($\mathcal{L}_{cc}$) and GAN loss ($\mathcal{L}_g$)

for training the network with corresponding λ hyper-parameters controlling relative importance of the terms,

$$\min_{C} \max_{D_f} \quad \lambda_{sce}\mathcal{L}^s_{ce}(C) + \lambda_{tce}\mathcal{L}^t_{ce}(C) + \lambda_{vs}\mathcal{L}^s_v(C) + \lambda_{vt}\mathcal{L}^t_v(C) - \lambda_{fd}\mathcal{L}_{fd}(D_f, C),$$

$$\min_{G} \max_{D} \quad \lambda_{cc}\mathcal{L}_{cc}(G) + \lambda_g\mathcal{L}_g(G, D). \tag{2.15}$$

Final Training Procedure & Algorithm

I start with training the classifier using Eq.1 and Eq.4. Following that, the generator is trained with the image generation loss (Eq.3) and content matching loss (Eq.2). During this time, the generator is warmed up to learn to generate target-like images and match the content information. Instead of using fixed λ_{ct}, I increase it gradually using the $e^{-5(1-p)^2}$ ramp-up function from [Tarvainen and Valpola (2017)], where p is the ratio of the iterations completed. Lastly, the network is trained using Eq.9 iteratively. Figure 2.7 depicts the overall framework of my approach and the complete algorithm is in Algorithm 1.

2.2.2 Experiments and Results

Dataset I test my approach on the following tasks: MNIST $\leftrightarrow$ USPS, MNIST $\rightarrow$ MNIST-M, SVHN $\leftrightarrow$ MNIST, SynDigits $\rightarrow$ SVHN and SynSigns $\rightarrow$ GTSRB. MNIST is a handwritten digits dataset with a black background [LeCun *et al.* (1998)]. SVHN contains an RGB digits dataset extracted from real-world house number images [Netzer *et al.* (2011)]. MNIST-M was created by combining MNIST images with patches randomly extracted from color photos of BSDS500. USPS is a digits dataset developed by recognizing the digits on the envelopes. Synthetic Digits (SynDigits) is a synthetically created digits dataset consisting of various English fonts on random

```
Algorithm 1: Iterative Image Translation
Input  : Source dataset $S = \{(x_i^s, y_i^s)\}_{i=1}^{N_s}$, target dataset $T = \{(x_i^t)\}_{i=1}^{N_t}$ and
         networks $F$, $I$, $D$, $D_f$
Output: Target labels $\{y_i^t\}_{i=1}^{N_t}$
Train $F$ using Cross Entropy (Eq. 1) + VAT loss (Eq. 4)
Further, train $I$ and $D$ using GAN loss (Eq. 3) + Content-Consistency loss
 (Eq. 2) using the ramp-up function.
for $M$ iterations do
    $m_s = miniBatch(S)$
    $m_t = miniBatch(T)$
    $m_{st} = I(m_s)$
    Train $D$ using Eq. 9
    Train $I$ using Eq. 9
    $m_{st} = I(m_s)$
    $m'_{st}$ = Filter $m_{st}$ using Eq. 5
    Train $F_d$ using Eq. 9
    Train $F$ using Eq. 9
end
Predict target labels $y_t$ using $F$
Return $\{y_t^i\}_{i=1}^{N_t}$
```

backgrounds. Synthetic Signs (SynSigns) and GTSRB are traffic sign datasets where SynSigns contain images from Wikipedia whereas GTSRB has real-world traffic sign images [Houben *et al.* (2013)]. All the digit datasets have 10 classes and present different visual variations in the domains. The traffic signs dataset provides a larger classification task of 43 classes.

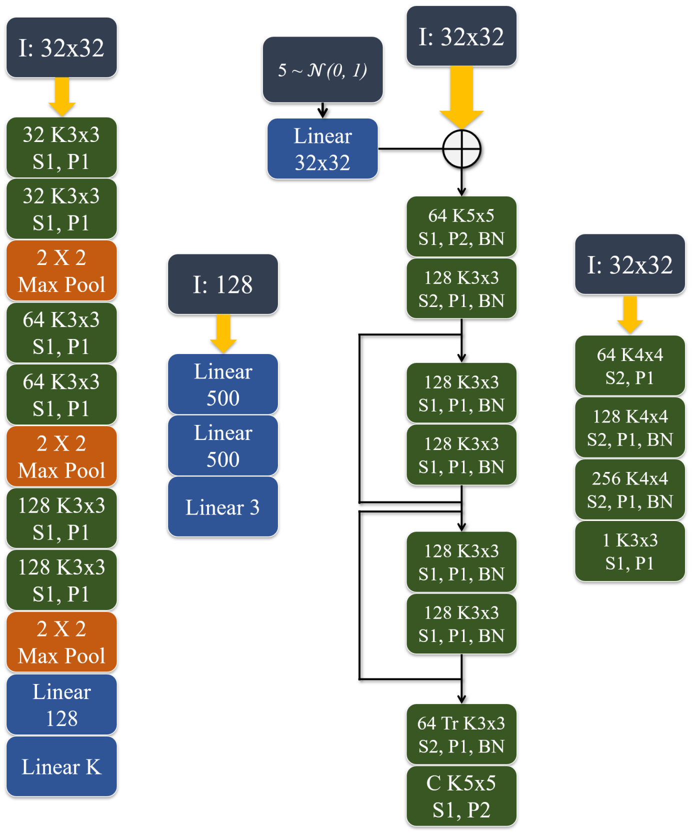


Figure 2.8: Network Architectures used in Iterative Image Translation experiments. From left to right: Classifier, Feature discriminator, Generator, and Pixel-level discriminator. All input images are resized to 32×32. K $3{\times}3$ refers to 3×3 convolution with K feature maps and Tr K refers to transposed convolution. I, S, P, BN, and C stand for Input, Stride, Pad, Batch normalization, and channels respectively. All the layers use ReLU activation except the Pixel-level discriminator which uses a lReLU with α=0.2.

Training details

I follow the standard unsupervised domain adaptation protocol i.e. train using the labeled source and unlabelled target training sets and evaluate on the target test set. All the input images are resized to 32×32. The network architectures are displayed in Figure 2.8. In the figure, From left to right: Classifier, Feature discriminator, Generator, and Pixel-level discriminator. K 3×3 refers to 3×3 convolution with K feature maps and Tr K refers to transposed convolution. I, S, P, BN, and C stand for Input, Stride, Pad, Batch normalization, and channels respectively. All the layers use ReLU activation except the Pixel-level discriminator which uses a lReLU with α=0.2.

The classifier has a generic architecture of six 3×3 convolutional layers containing 32, 32, 64, 64, 128, and 128 feature maps followed by one fully-connected layer of 128 hidden units and a classification layer of size K. The Feature discriminator uses the output of the first fully connected layer of the classifier as the input. The generator and the pixel-level discriminator architecture are inspired by [Zhu *et al.* (2017); Isola *et al.* (2017); Russo *et al.* (2018)]. The generator uses a 5-dimensional noise vector sampled from $\mathcal{N}(0, 1)$ which is then scaled up to the size of the image using a linear layer and concatenated channel-wise with the input image. For MNIST and USPS image generation, I used a smaller image discriminator by excluding the 256 feature map layer.

All the networks are trained using a batch size of 128 and Adam optimizer with $\beta_1 = 0.5$ and $\beta_2 = 0.99$. I use a learning rate of $1e^{-4}$ to train the classifier, $2e^{-4}$ for training the generator and discriminator (default from CycleGAN [Zhu *et al.* (2017)]). For the feature discriminator, I used a lower learning rate of 10^{-5} to prioritize pixel-level adaptation over feature-level adaptation.

I set the hyperparameters λ_{sce}, λ_{tce}, λ_g to 1. the The hyperparameters λ_{vs}, λ_{vt}, λ_{fd}, ϵ, λ_{cc} are set to 0.1, 0.1, 0.01, 3.5 and K^2s respectively based on existing literature which has used similar loss functions. τ is the only hyperparameter tuned by use and is empirically set to 10^{-6}.

Method	MNIST→USPS	USPS→MNIST	MNIST→MNISTM	SVHN→MNIST	MNIST→SVHN	SyDig→SVHN	SySigns→GTSRB
Source only	83.1	67.2	63.3	60.9	30.8	88.7	94.2
DANN	85.1	73.0	76.7	73.9	35.7	91.1	88.7
MMD	81.1	-	76.9	71.1	-	-	91.1
ADDA	89.4	90.1	-	76.0	-	-	-
ATT	-	-	94.2	86.2	52.8	93.1	96.2
DRCN	91.8	73.7	-	82.0	40.0	-	-
PixelDA	95.9	-	98.2	-	-	-	-
DSN	-	-	83.2	82.7	-	91.2	93.1
I2I Adapt	95.1	92.2	-	92.1	-	-	-
UNIT	96.0	93.6	-	90.5	-	-	-
PLR	90.7	91.8	94.3	97.3	63.4	-	-
CYCADA	95.6	96.5	-	90.4	-	-	-
DupGAN	96.0	98.8	-	92.5	62.7	-	-
SBADA	97.6	95.0	**99.4**	76.1	61.1	-	96.7
IIT	**97.8**	**99.1**	**99.4**	**97.4**	**66.5**	**95.4**	**97.2**

Table 2.6: Comparison of classification accuracy of IIT (Iterative Image Translation) with different domain adaptation methods. Pixel-based approaches are below the dashed line.

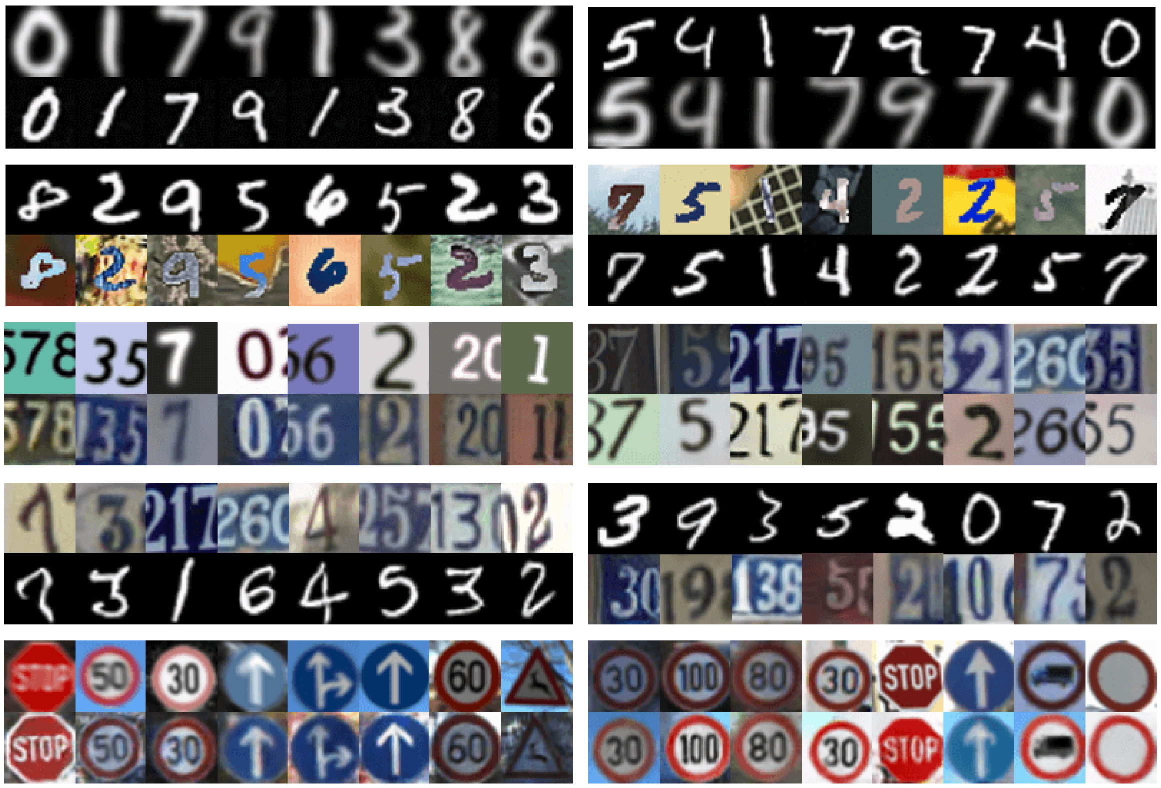


Figure 2.9: Sample generated images. Top to Bottom: USPS ↔ MNIST, MNIST ↔ MNIST-M, SyDigits ↔ SVHN, SVHN ↔ MNIST, SySigns ↔ GTSRB. I show images for each combination on every row. Upper row: Original source images and Lower row: Generated target images.

Results

The results of my experiments are in Table 2.6. I compare my approach with Image translation approaches like CYCADA [Hoffman *et al.* (2018)], SBADA [Russo *et al.* (2018)], as well as feature alignment, approaches like DANN [Ganin *et al.* (2016b)], MMD [Long *et al.* (2015)], ADDA [Tzeng *et al.* (2017)] and my approach outperforms all the compared methods on all the combinations. Even for the difficult combination of MNIST → SVHN, my approach is effective and beats all the compared baselines.

CC	VAT	FA	TOT	M→U	U→M	M→MM	S→M
✗	✗	✗	✗	83.1	67.2	63.3	60.9
✗	✓	✓	✓	94.2	97.4	98.3	15.5
✓	✗	✓	✓	96.5	98.1	97.7	77.4
✓	✓	✗	✓	96.2	95.1	91.3	77.8
✓	✓	✓	✓	**97.8**	**99.1**	**99.4**	**97.4**

Table 2.7: Ablation study of Iterative Image Translation on MNIST(M) $\rightarrow$ USPS(U), MNIST(M) $\rightarrow$ MNIST-M(MM) and SVHN(S) $\rightarrow$ MNIST(M). CC: Content-consistency loss; VAT: Virtual adversarial training for source and target; FA: feature-level alignment; TOT: Train on target. The first row represents source-only.

The generated target-like images can represent the target variations and is an effective way to learn the target domain. The sample translated images can be found in Fig. 2.9. In the figure, Top to Bottom are USPS $\leftrightarrow$ MNIST, MNIST $\leftrightarrow$ MNIST-M, SyDigits $\leftrightarrow$ SVHN, SVHN $\leftrightarrow$ MNIST, and SySigns $\leftrightarrow$ GTSRB. I show images for each combination on every row. The upper row is the original source images and the lower row is the generated target images.

2.2.3 Analysis

Ablation Study

To analyze the significance of each of the loss components, I perform an ablation study by removing them, one loss component at a time, from my approach. I use MNIST $\leftrightarrow$ USPS, SVHN $\rightarrow$ MNIST, and MNIST $\rightarrow$ MNIST-M combinations for this study. The results are presented in Table 2.7. The losses are abbreviated as CC:

Content-consistency loss; VAT: Virtual adversarial training for source and target; FA: feature-level alignment; TOT: Train on target. The first row in the table represents source-only. I perform the following three experiments

1. No content-consistency loss: I replace the content consistency loss with the cycle consistency loss (used for training CycleGAN) by adding another image generator [Zhu *et al.* (2017)]. Though this produces decent results for small variation combinations - MNIST $\leftrightarrow$ USPS, MNIST $\rightarrow$ MNIST-M, but results in shuffling of labels for the high variation combinations like SVHN $\rightarrow$ MNIST.

2. No virtual adversarial training losses: No adversarial training allows the generator to fool the consistency loss without matching the content and results in negative transfer, thereby impacting the final accuracy.

3. No feature alignment: It is clear from the results that pixel-level alignment is not sufficient to train the classifier on the target domain.

Feature Visualization

I use TSNE plots to visualize the deep features generated by the classifier before and after the adaptation in Figure 2.10 and Figure 2.13. The deep features are spread across for source, target-like, and target domains before the adaptation. After the adaptation, all three of them are aligned together to get an indistinguishable representation across domains.

Generated Images Spectrum

I use the task of MNIST$\rightarrow$MNIST-M transfer to understand the variations in generated images. Fig.2.10 shows the generated images before and after the adaptation. Before adaptation, the generated images look similar to MNIST-M but they are only

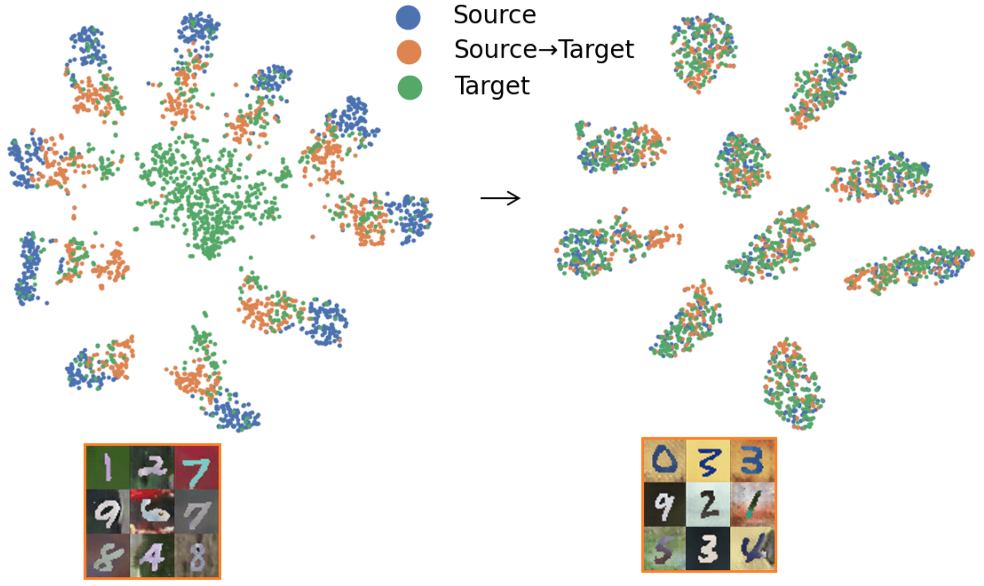


Figure 2.10: TSNE plots of the deep features of source, target-like generated, and target images along with generated target-like images for the MNIST → MNIST-M task. Left: depicts image features and generated target-like images before convergence. Right: depicts image features and generated target-like images after convergence.

restricted to having a darker background and lighter foreground, resembling MNIST images. I believe it is because the classifier can only detect dark-to-light edges. Hence, the generator needs to generate such images to keep the content loss low. The deep features of these generated images lie between those of the source (MNIST) and the target (MNIST-M) samples. They act as a bridge between the source and target domain. After adaptation, the features are aligned and the generated images cover the full spectrum of the target images.

Domain Gap

[Ben-David *et al.* (2010)] defined the $\mathcal{A}$-distance metric for evaluating testing the domain discrepancy as $2\times(1-\varepsilon)$ where ε is the generalization error of a classifier trained

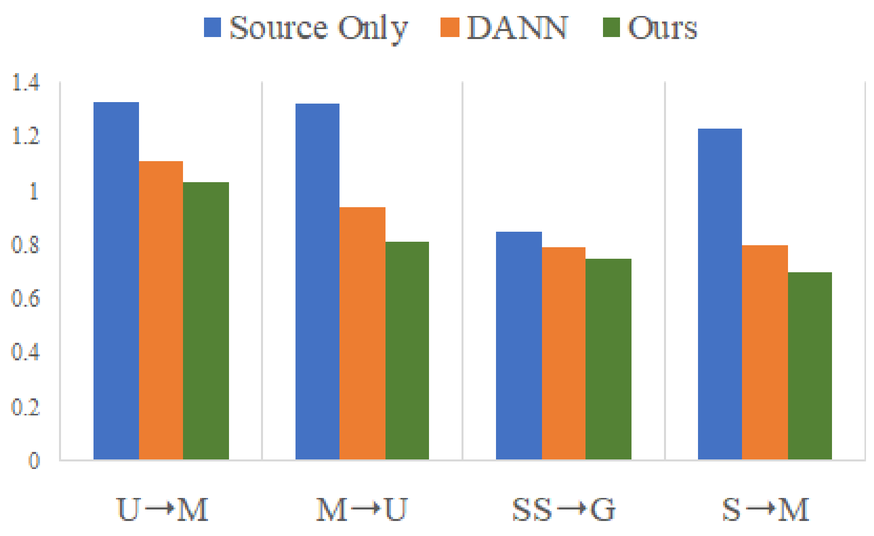


Figure 2.11: The domain discrepancy $\mathcal{A}$-distance between deep features for the MNIST(M) $\leftrightarrow$ USPS(U), SySigns(SS) $\rightarrow$ GTSRB(G) and SVHN(S) $\rightarrow$ MNIST(M). Smaller is better.

to classify deep features. I use 5-fold cross-validation using a linear SVM to compute the A-distances. Fig. 2.11 shows the $\mathcal{A}$-distance between three domains on different tasks using Source only, DANN-alignment [Ganin *et al.* (2016b)] and my method. My approach brings the domains closer while achieving superior performance.

Content Extraction

I perform linear interpolation between two input noise vectors for 3 pairs of source-target combinations and the results are in Figure 2.12. The first column is the source image and the other columns are generated target images. The images in the middle are generated by linear interpolating the random noise vector used for the first and the last columns of the generated images. For each pair, I use the same noise vectors, and the linear interpolation results in the same changes in style. For MNIST $\rightarrow$ MNIST-M combination, the color variations are similar across the two inputs. For

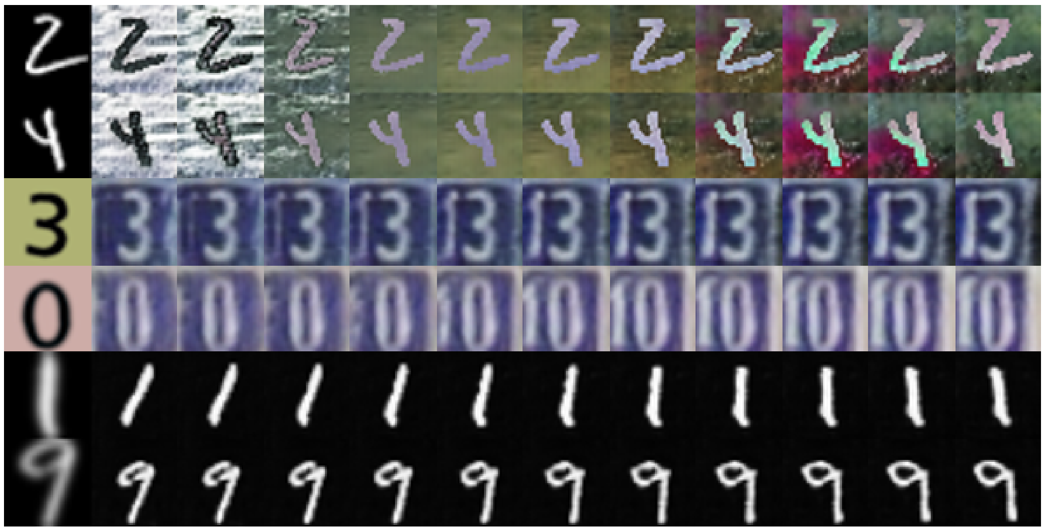


Figure 2.12: Linear Interpolation between two random noise vectors on MNIST → MNIST-M (Top two rows), SynDigits → SVHN (Middle two rows), and USPS → MNIST (Bottom two rows) pairs. Each pair uses the same input noise vector. The first column is the source image and the other columns are generated target images. The images in the middle are generated by linear interpolating the random noise vector used for the first and the last columns of the generated images.

SynDigits → SVHN, '1' can be seen appearing on the left and slowly transforming the background to a box. Similarly, for USPS → MNIST, it can be observed that equivalent angle rotations for the inputs. This confirms that my model extracts the content and uses the noise vector as the style component.

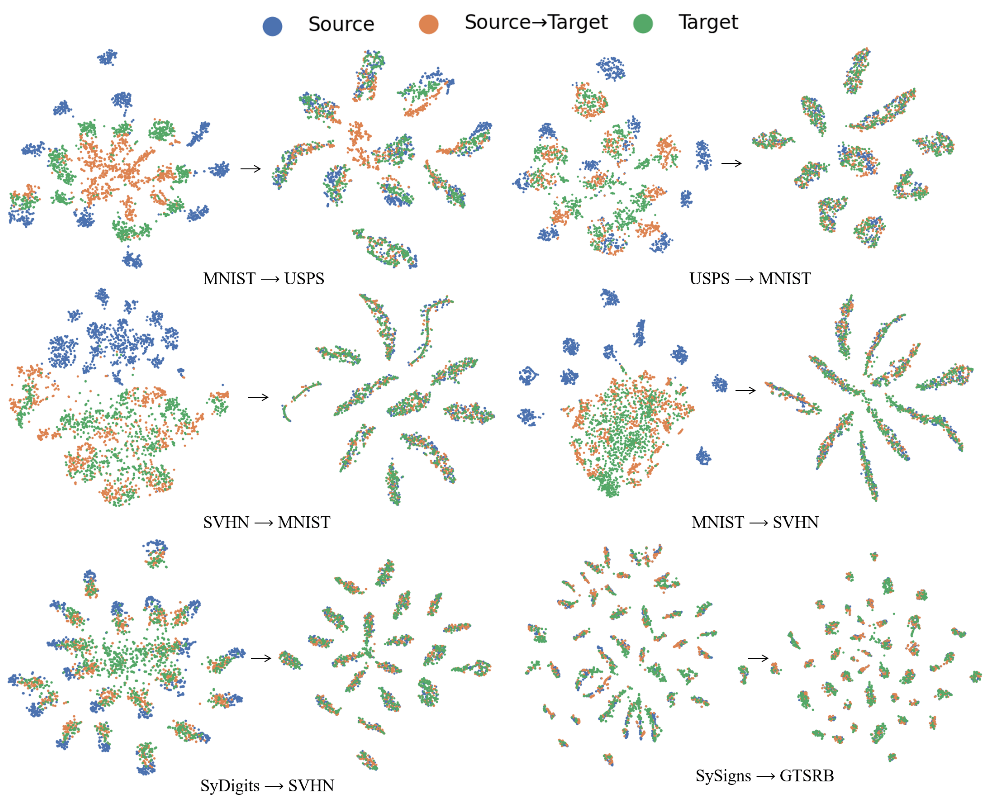


Figure 2.13: TSNE plots for MNIST → USPS, USPS → MNIST, SVHN → MNIST, MNIST → SVHN, SyDigits → SVHN and SySigns → GTSRB in English reading order. For every combination, left is before adaptation and right is after adaptation. ● denotes source domain, ● denotes source to target-like source translated images, and ● denotes Target domain samples.

2.3 Output-Level

Unsupervised domain adaptation requires both the labeled source dataset and the unlabeled target dataset when learning a classifier for the target. Access to a source dataset may not be available owing to security and privacy constraints. Source-free domain adaptation assumes only the presence of a pre-trained source classifier and the unlabeled target dataset [Liang *et al.* (2020)]. Unsupervised domain adaptation uses source-target alignment approaches like feature alignment [Ganin *et al.* (2016b); Pei *et al.* (2018); Tzeng *et al.* (2017); Shen *et al.* (2018)] and pixel alignment [Hoffman *et al.* (2018); Bousmalis *et al.* (2017); Russo *et al.* (2018)] to perform domain adaptation. These approaches are generally not possible in a source-free setting.

In this dissertation, I present a novel approach that models a generative paradigm governed by a joint probability $p(x, y)$ where x is the visible data and y are latent (labels) variables. I hypothesize an approximation $q_\theta(y|x)$ to the unknown posterior probabilities $p(y|x)$. I demonstrate that a good approximation of the posterior probability $q_\theta(y|x) \approx p(y|x)$ can be learned by aligning the predicted posterior distribution $q_\theta(y|x)$ with the class prior $p(y)$. I present arguments to establish that the generative paradigm is equivalent to the source-free domain alignment setting when I align the source and target posterior probabilities using adversarial alignment. Specifically, I circumvent the need for source data by generating the source category distribution $\hat{p}_s(y)$ using a conditional generative adversarial framework. Domain adaptation is achieved by enforcing the target posterior distribution to align with the source category distribution using adversarial alignment. In place of the traditional image or feature alignment, I proceed with alignment in the label space.

I present a **G**enerative model for the **A**lignment of **P**osterior probabilities (GAP) of the source and target to perform source-free and unsupervised domain adaptation [Chhabra *et al.* (2023)]. Some of the highlights of the GAP model are: (1) GAP does not introduce any new hyper-parameters and hence, does not require any additional hyperparameter tuning; (2) GAP is robust to variations in batch size (3) GAP can effectively exploit source data (when present) to enforce inter-class relationships through knowledge distillation. [Hinton *et al.* (2015)].

2.3.1 Generative Alignment of Posterior Probabilities

Problem Statement

Let $S = \{x_i^s, y_i^s\}_{i=1}^{N_s}$ be the source dataset where N_s represents the number of labeled training samples and y_s is the one-hot representation of the source label with K categories. The unlabeled target dataset is $T = \{x_i^t\}_{i=1}^{N_t}$ with N_t samples. The datasets S and T are drawn from distributions p_s and p_t with $p_s(x, y) \neq p_t(x, y)$, but they share the same label space with identical K categories. The goal is to estimate the labels $\{\hat{y}_i^t\}_{i=1}^{N_t}$ corresponding to elements in T. Source-free domain adaptation is a more restricted setup where the source and target data are not accessible at the same time. In source-free domain adaptation, once the source image classifier $F_\theta(.)$ has been trained using S, I lose access to S. I aim to predict target labels using T and the source classifier $F_\theta(.)$.

Generative Model

I present a generative paradigm to estimate the labels for the target dataset. Let the images from the target dataset be sampled from $x \in X$, where X is the space of images. The corresponding labels are one-hot binary vectors of the type $y \in \{0, 1\}^K$, where $\sum_k y_k = 1$. K is the number of distinct categories in the target dataset. The images and labels are sampled from an unknown target distribution $p_t(x, y)$. Given a target sample x, I intend to estimate the posterior $p_t(y|x)$ using which I arrive at the label y. I hypothesize to approximate $p_t(.)$ using a parametric model $q_\theta(.)$. In essence, I seek to estimate parameter θ such that $p_t(y|x) \approx q_\theta(y|x)$.

In order to estimate θ, I begin with estimating the reverse Kullback-Leibler (KL) divergence $\text{KL}(q_\theta(y|x)||p_t(y|x))$,

$$\begin{aligned}
\text{KL}(q_\theta(y|x)||p_t(y|x)) &= \mathbb{E}_{q_\theta(y|x)}\big[\text{log}q_\theta(y|x) - \text{log}p_t(y|x)\big] \\
&= \mathbb{E}_{q_\theta(y|x)}\big[\text{log}q_\theta(y|x) - \text{log}p_t(x|y) - \text{log}p_t(y) + \text{log}p_t(x)\big] \\
&= \text{KL}(q_\theta(y|x)||p_t(y)) - \mathbb{E}_{q_\theta(y|x)}\big[\text{log}p_t(x|y)\big] + \text{log}p_t(x) \\
\text{KL}(q_\theta(y|x)||p_t(y|x)) &\leq \text{KL}(q_\theta(y|x)||p_t(y)) - \mathbb{E}_{q_\theta(y|x)}\big[\text{log}p_t(x|y)\big]. \qquad (2.16)
\end{aligned}$$

In deriving Eq.2.16 I have used $p_t(y|x) = \frac{p_t(x|y)p_t(y)}{p_t(x)}$ and $\mathbb{E}_{q_\theta(y|x)}[\log\ p_t(x)] = \log\ p_t(x) \leq 0$. The R.H.S in Eq.2.16 is an upper bound for the KL-divergence between the distributions $q_\theta(y|x)$ and $p_t(y|x)$. The value of θ which will minimize the R.H.S will align the two distributions. The 1st term on the R.H.S. is the measure of alignment between the unknown prior distribution $p_t(y)$ and the generative model $q_\theta(y|x)$. The second term can be viewed as the expected reconstruction error converting from y to x.

Model Assumptions

A few assumptions are made to further simplify the model. $p_t(y)$ is unknown, but the source and the target have the same label space. I assume $p_s(y) \approx p_t(y)$, which is a reasonable assumption along the lines of assuming covariate-shift ($p_s(y|x) \approx p_t(y|x)$) [Ben-David *et al.* (2010)] or concept-shift ($p_s(x) \approx p_t(x)$) [Vorburger and Bernstein (2006); Popovič (2011)]. I model $q_\theta(.)$ using a neural network which takes

x as input and yields the posterior $q_\theta(y|x)$ as output. The presented model has issues of *identifiability.* A model is identifiable when $p(x)$ has a unique decomposition in $\sum_y p(x|y)p(y)$ [Chapelle *et al.* (2006)]. If a swapping of labels can yield the same marginal $p(x)$, the problem setup is not identifiable and the resulting solution is not consistent (different assignment of labels under different initial conditions). The 2nd term on the R.H.S of Eq. 2.16 can be viewed as a reconstruction term that can ensure identifiability. However, $p_t(x|y)$ is unknown. I address both problems by dropping the 2nd term and initializing the parameters in the network $q_\theta(.)$ with the parameters of a pretrained source classifier. This is a robust initialization mechanism that will bias the model $q_\theta(.)$ to yield a consistent mapping between x and y. Under these assumptions, I minimize $\mathrm{KL}(q_\theta(y|x)||p_s(y))$ in an attempt to align the distributions $q_\theta(y|x)$ and $p_t(y|x)$.

Source Replicator

In the absence of the source data, I cannot align the probabilities for the source and target directly. Therefore, I aim to replicate source prior probability vectors for target alignment. I perform this additional step of source replication after training a source network. Specifically, I design a conditional generative adversarial framework [Mirza and Osindero (2014)] - a generator $G_c(.;\theta_c)$ that takes fake label ($y^f \in Y$, where $Y = \{0,1\}^K$) and noise ($z \in \mathcal{N}(0, I)$) as input to generate a probability vector of K-dimensions (using softmax activation at the last layer). The conditional discriminator $D_c(.;\phi_c)$ is trained to discriminate between generated probabilities and source probabilities from the pre-trained source classifier $F(.,\theta)$. The conditional framework is preferred over a vanilla GAN primarily to avoid partial mode collapse and also to have control over the prior class distribution. I refer to the conditional generator as the Source Replicator in this dissertation. On account of its stability, I train the conditional GAN using the least squared loss function [Mao *et al.* (2017)],

$$\begin{aligned}\min_{\phi_c} \frac{1}{2}\mathbb{E}_{(x,y)\sim S}[(D_c(F(x;\theta), y; \phi_c) - 1)^2] \\ + \frac{1}{2}\mathbb{E}_{\substack{y^f \sim Y \\ z \in \mathcal{N}(0,I)}}[(D_c(G_c(z, y^f; \theta_c), y^f; \phi_c))^2], \\ \min_{\theta_c} \frac{1}{2}\mathbb{E}_{\substack{y^f \sim Y \\ z \in \mathcal{N}(0,I)}}[(D_c(G_c(z, y^f; \theta_c), y^f; \phi_c) - 1)^2]. \qquad (2.17)\end{aligned}$$

The conditional framework captures the statistical variations of the source predictions and their inter-class relations. I use the trained conditional generator G_c to replicate (generate) source probabilities $p_s(y)$ when minimizing $\text{KL}(q_\theta(y|x)||p_s(y))$. Figure 2.14A is the pre-trained source classifier. Figure 2.14B depicts the training of the Source Replicator G_c.

Source-free Domain Alignment

Domain adaptation approaches rely on the source data to perform source-target alignment either in the pixel space through image translation [Hoffman *et al.* (2018); Bousmalis *et al.* (2017)] or in the feature space [Tzeng *et al.* (2014); Ganin *et al.* (2016b)] (see Figure 2.1). The presented model can be interpreted as the alignment of posterior probabilities $p_s(y|x)$ and $p_t(y|x)$, where I align the source and target in the final stage of the classification process. As the input image x propagates through the classification framework (neural network), it is transformed from an image to a feature vector and finally to a probability vector while decreasing its information content and complexity. I hypothesize that source-target alignment in the high-dimensional pixel space and feature space is complex and less effective compared to the alignment of probability vectors. Also, the feature space is constantly changing as the network trains whereas, the probability space has fewer variations.

I implement a Generative Adversarial Network (GAN) to align the distributions $q_\theta(y|x)$ and $p_s(y)$ [Goodfellow *et al.* (2014)]. The image classifier network $F(.;\theta)$ mod-

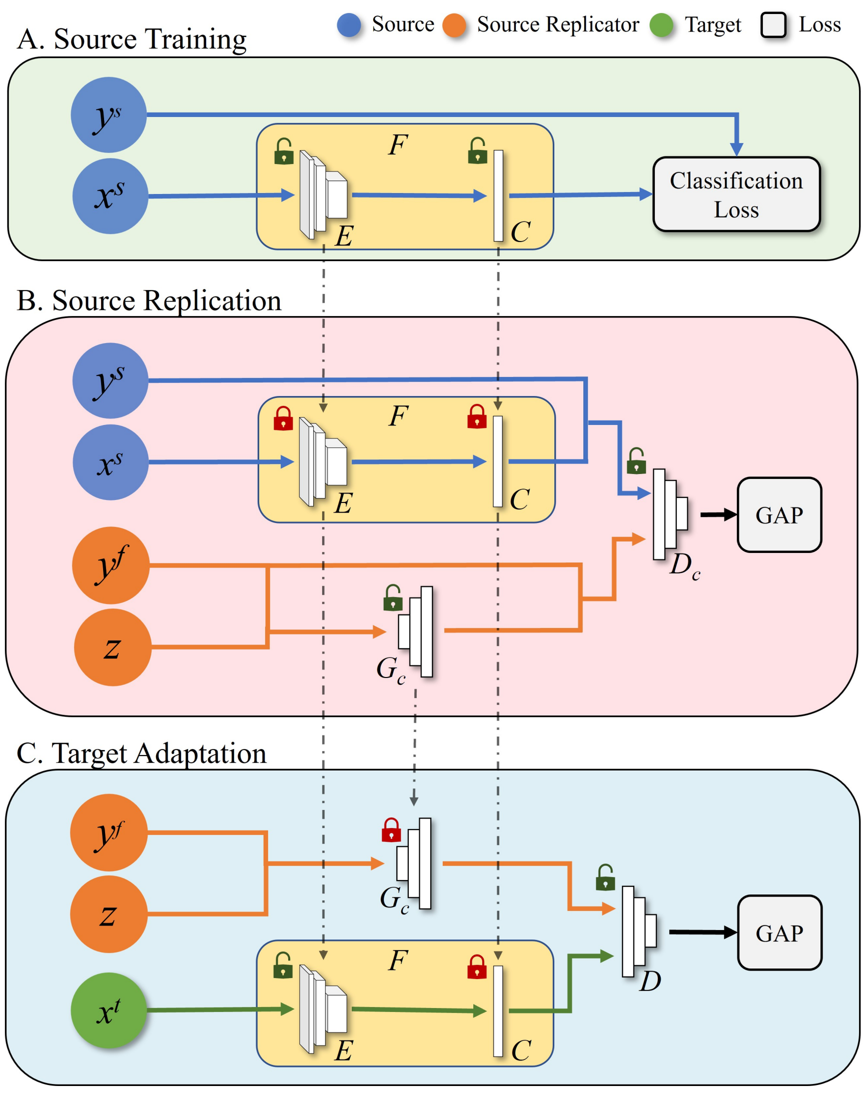


Figure 2.14: Model diagram for the GAP. **(A)** The feature extractor E and Classifier C are trained using the source data. **(B)** Source replicator (conditional generator) G_c is trained to replicate source predictions. $\boldsymbol{y}_f$ represents the fake label. **(C)** During the target adaptation, the C component is frozen to retain the source classification boundaries. The feature extractor E and classifier C represent the probability generator F and are initialized using the source network. The Source replication is performed using Eq.2.17 and the target adaptation uses Eq.2.18.

els $q_\theta(y|x)$, where θ are the parameters of $F(.;\theta)$, x is the target input image and y is the predicted label. The source prior $p_s(y)$ is modeled by the Source Replicator $G_c(.)$. I initialize the image classifier $F(.;\theta)$ with the parameters of a pre-trained source classifier. The similarity between the source and target domains is exploited to provide a near-optimal initialization for θ. The Discriminator network $D(.;\phi)$, with parameters ϕ, is trained to distinguish between the image classifier output $G(x;\theta) \rightarrow q_\theta(y|x)$ and outputs of the Source Replicator, $G_c(z, y^f; \theta_c)$. Figure 2.14C depicts target adaptation where only the feature extractor (E) in the pre-trained classifier $F(.)$ is trained. The parameters of the classifier (C) are frozen to anchor the classifier and ensure the target alignment does not drift away to yield a trivial solution in the absence of source data for alignment. The GAN objective is based on the mean-squared loss function [Mao *et al.* (2017)],

$$\begin{aligned}
&\min_{\phi} \frac{1}{2}\mathbb{E}_{\substack{y\sim Y \\ z\in\mathcal{N}(0,I)}}[(D(G_c(z,y;\theta_c);\phi)-1)^2] + \frac{1}{2}\mathbb{E}_{x\sim T}[(D(F(x;\theta);\phi))^2], \\
&\min_{\theta} \frac{1}{2}\mathbb{E}_{x\sim T}[(D(F(x;\theta);\phi)-1)^2]. \qquad (2.18)
\end{aligned}$$

where the input to the Source Replicator $y \in Y$ is a 1-of-K binary vector, where $p(y|\pi) = \prod_{k=1}^{K} \pi_k^{\bar{y}_k}$. Following [Liang *et al.* (2020)], I assume the mixing components in π to be a uniform prior with $\pi_k = K^{-1}\ \forall k$. The uniform prior can be replaced with the source prior distribution. However, I found empirically that the uniform prior performs better than the latter.

In Figure 2.14C, the feature extractor (E) and classifier (C) together form the image classifier F. To account for smoother posterior probabilities, I apply label smoothing with a constant $\epsilon = 0.1$ [Szegedy *et al.* (2016b)] while training the source network. Label smoothing results in better generalization accuracy and has been adopted in many source-free domain adaptation approaches [Liang *et al.* (2020); Ishii and Sugiyama (2021); Ahmed *et al.* (2021)]. The parameters of the image classifier $F(.;\theta)$ are initialized using a pre-trained source network. Following [Liang *et al.* (2020)], I only train the feature extractor (E) and fix the classifier (C) during the target adaptation phase.

2.3.2 Experiments

Datasets

I perform my experiments on the standard source-free domain adaptation settings. For digits experiments, I use SVHN $\rightarrow$ MNIST, MNIST $\rightarrow$ USPS and USPS $\rightarrow$ MNIST combinations [Netzer *et al.* (2011); LeCun *et al.* (1998); Hull (1994)]. For object recognition, I use Office-31 [Saenko *et al.* (2010)], Office-Home [Venkateswara *et al.* (2017)] and VisDa-C [Peng *et al.* (2017)] datasets.

Training Setup

I follow the training protocol and use the network architectures from [Liang *et al.* (2020)]. For the digits dataset, the networks are trained from scratch. I use ResNet-50 and ResNet-101 as the backbone network for Office datasets and VisDa datasets respectively. The networks G_c and D_c consist of four fully connected layers of size 500 and discriminator D is a vanilla discriminator composed of two hidden layers of size 100. All the networks are trained on a batch size of 64 using an SGD optimizer with a momentum of 0.9. I use $1e^{-2}$ learning rate for office and $1e^{-3}$ for the VisDa dataset and decay it as $\eta = \eta_0(1 + 10p)^{-0.75}$ where p is the training progress. The learning rate for pre-trained layers is reduced by a factor of 10.

Source	Ar			Cl			Pr			Rw			Avg
Target	Cl	Pr	Rw	Ar	Pr	Rw	Ar	Cl	Rw	Ar	Cl	Pr	
Source	34.9	50.0	58.0	37.4	41.9	46.2	38.5	31.2	60.4	53.9	41.2	59.9	46.1
Source + LS	44.6	67.3	74.8	52.7	62.7	64.8	53.0	40.6	73.2	65.3	45.4	78.0	60.2
Rob. Adapt	-	-	-	-	-	-	-	-	-	-	-	-	65.1
SFDA	48.4	73.4	76.9	64.3	69.8	71.7	62.7	45.3	76.6	69.8	50.5	79.0	65.7
SHOT-IM	$\underline{55.4}$	$\underline{76.6}$	80.4	66.9	74.3	75.4	$\underline{65.6}$	$\underline{54.8}$	80.7	$\underline{73.7}$	58.4	83.4	70.5
SHOT-full	**57.1**	**78.1**	**81.5**	**68.0**	**78.2**	$\underline{78.1}$	**67.4**	**54.9**	$\underline{82.2}$	73.3	$\underline{58.8}$	**84.3**	**71.8**
GAP	$\underline{55.4}$	73.4	$\underline{80.8}$	$\underline{67.2}$	$\underline{75.5}$	**78.3**	65.5	54.0	**82.4**	**74.3**	**59.4**	$\underline{84.0}$	$\underline{70.8}$

Table 2.8: Comparison of source-free classification accuracies on the Office-Home dataset (ResNet-50). Bold numbers represent the highest accuracy and the underline denotes the second highest. LS stands for label smoothing.

Method	Plane	Bcycl	Bus	Car	Horse	Knife	Mcycl	Person	Plant	Sktbrd	Train	Truck	Avg
Source	55.1	53.3	61.9	59.1	80.6	17.9	79.7	31.2	81.0	26.5	73.5	8.5	52.4
Source + LS	60.9	21.6	50.9	**67.6**	65.8	6.3	82.2	23.2	57.3	30.6	84.6	8.0	46.6
SFIT	-	-	-	-	-	-	-	-	-	-	-	-	63.5
Rob. Adapt	-	-	-	-	-	-	-	-	-	-	-	-	74.9
SFDA	86.9	81.7	**84.6**	63.9	93.1	91.4	**86.6**	71.9	84.5	58.2	74.5	42.7	76.7
SHOT-IM	93.7	86.4	78.7	50.7	91.0	93.5	79.0	78.3	89.2	85.4	**87.9**	51.1	80.4
SHOT-Full	**94.3**	**88.5**	80.1	57.3	93.1	94.9	80.7	**80.3**	**91.5**	**89.1**	86.3	**58.2**	**82.9**
GAP	94.2	84.8	82.5	57.2	**93.8**	**95.1**	86.5	78.2	83.1	87.8	86.3	53.5	81.9

Table 2.9: Comparison of source-free classification accuracies on the VisDA dataset (ResNet-101). Bold numbers represent the highest accuracy and the underline denotes the second highest. LS stands for label smoothing.

Method	A→D	A→W	D→A	D→W	W→A	W→D	Avg
Source	68.9	68.4	62.5	96.7	60.7	99.4	76.1
Source + LS	80.8	76.9	60.3	95.3	63.6	98.7	79.3
SDDA	85.3	82.5	66.4	**99.0**	67.7	<u>99.8</u>	83.5
Rob. Adapt	-	-	-	-	-	-	87.0
SFDA	<u>92.2</u>	<u>91.1</u>	71.0	98.2	71.2	99.5	87.2
SHOT-IM	90.6	**91.2**	72.5	98.3	71.4	**99.9**	87.3
SHOT-full	**94.0**	90.1	**74.7**	98.4	**74.3**	**99.9**	**88.6**
GAP	90.6	90.9	<u>74.5</u>	<u>98.7</u>	<u>73.9</u>	<u>99.8</u>	<u>88.1</u>

Table 2.10: Comparison of source-free classification accuracies on the Office-31 dataset (ResNet-50). Bold numbers represent the highest accuracy and the underline denotes the second highest. LS stands for label smoothing.

Results

The results for digits, Office-Home, VisDa and Office-31 are in Table 2.11, 2.8, 2.9 and 2.10 respectively. I compare my approach with source-free domain adaptation methods like SFDA [Kim *et al.* (2020)], SDDA [Kurmi *et al.* (2021)], and SHOT [Liang *et al.* (2020)]. My method achieves comparable performance against all the baseline methods. SHOT-full outperforms my approach but it uses pseudo labeling loss along with SHOT-IM loss functions- entropy minimization and diversity maximization. My method is much simpler, does not use any auxiliary loss, or requires hyper-parameter tuning.

Method	S→M	M→U	U→M	Avg
Source	67.1	82.2	69.6	73.0
Source + LS	70.2	79.7	88.0	79.3
SFIT	90.4	84.7	82.3	85.8
SDDA	76.3	88.5	-	-
SHOT-IM	<u>99.0</u>	97.6	97.7	98.2
SHOT-Full	98.9	**98.0**	<u>97.9</u>	98.3
GAP	**99.1**	<u>97.6</u>	**98.4**	**98.4**
Target-Supervised (Oracle)	99.4	98.0	99.4	98.8

Table 2.11: Comparison of source-free classification accuracies on the digits dataset. Bold numbers represent the highest accuracy and the underline denotes the second highest.

2.3.3 Analysis

Ablation Study

I perform an ablation study on my loss function to understand the contribution of its different components. First, I remove the label smoothing from source model training. In the second experiment, I replace the mean-squared error (MSE) loss with the binary cross-entropy (BCE) loss. I perform these experiments on Office-31, Office-home, and digits datasets and the results are in Table 2.12. In the table, 'LS' denotes label smoothing used for source-model training. 'Alignment' denotes the alignment loss used for target adaptation. NA: No alignment was performed/Source-only performance. Losses functions are abbreviated as BCE: Binary Cross-Entropy;

LS	Alignment	Office-31	Office-Home	Digits
✗	NA	76.1	46.1	73.0
✗	BCE	84.1	67.0	98.1
✗	MSE	86.3	69.8	98.1
✓	NA	79.3	60.2	79.3
✓	BCE	85.7	68.8	97.2
✓	MSE	**88.1**	**70.8**	**98.4**

Table 2.12: Ablation study of GAP on Office-31 and OfficeHome source-free setting. 'LS' denotes label smoothing used for source-model training. 'Alignment' denotes the alignment loss used for target adaptation. NA: No alignment was performed/Source-only performance. BCE: Binary Cross-Entropy; MSE: Mean-squared-error

MSE: Mean-squared-error. As per the results, label smoothing provides benefits in generalization during source training and target domain adaptation. The MSE loss function results in slightly superior performance, mainly due to its stability, and also observed that BCE loss experiences partial mode collapse frequently.

Impact of batch size

When there are a large number of categories, a mini-batch cannot contain samples from all the classes. I compare GAP with SHOT-IM (Entropy minimization + diversity maximization loss) for different batch sizes. Figure 2.15 shows the results of this experiment on Art $\rightarrow$ Clipart (Office-Home) source-free setting. When the batch size decreases to $\frac{1}{4}$ of its starting value, my approach has a significantly lesser decrease (approximately 10%) in performance compared to SHOT-IM (25%). GAP does not

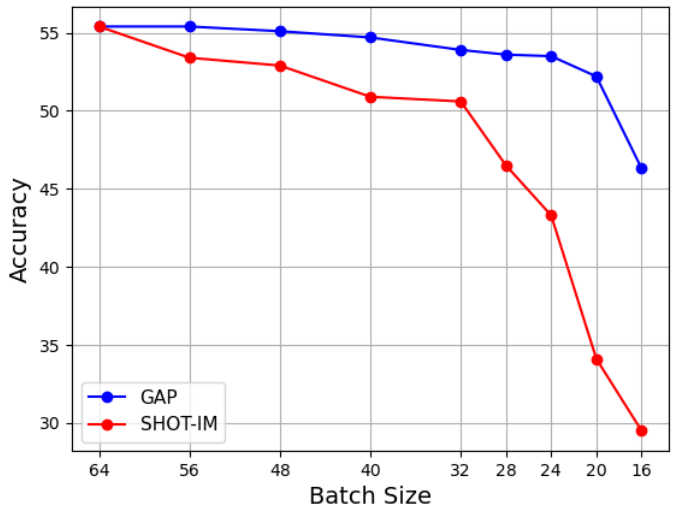


Figure 2.15: Comparison of GAP with SHOT-IM (Entropy minimization + Diversity maximization loss) for different batch sizes on Art → Clipart (Office-Home) source-free setting.

show fluctuations until the batch size reaches 20 whereas the performance of SHOT-IM drops significantly with the reduction of batch size. Due to this, all approaches using entropy minimization and diversity maximization losses are susceptible to small batch sizes. I overcome this issue by training the Discriminator multiple times before updating the Generator. This way the Discriminator is unaffected by the mini-batch bias.

TSNE Visualization

TSNE plots for visualizing the output probability space and the penultimate layer features for the VisDa dataset are in Figure 2.16. Different colors represents different domains: Blue: Source; Orange: Source Replicator; Green: Target. The probability distributions are showcased in the left section and the penultimate layer features on the right.

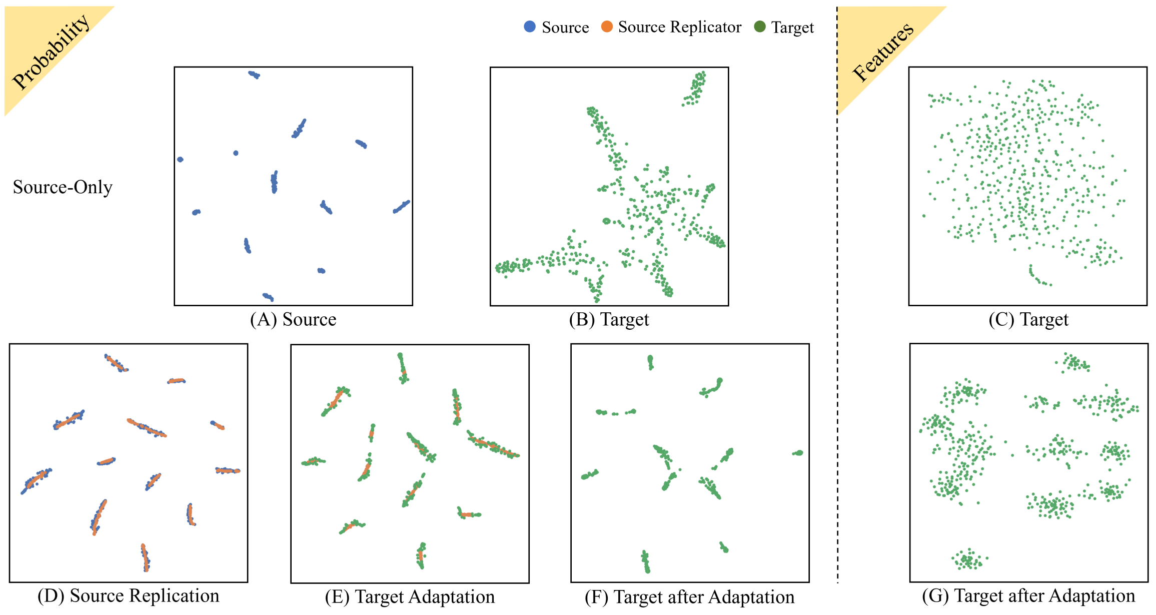


Figure 2.16: TSNE visualization for VisDa in a source-free setting. Different colors represents different domains: Blue: Source; Orange: Source Replicator; Green: Target. The probability distributions are showcased in the left section and the penultimate layer features on the right. The first row is before adaptation and the second row is GAP. (A) Source probability distribution on source trained model. (B) Target probability distribution on the source-trained model. (C) Target features using the source-trained model. (D) Source and Source Replicator probability distribution. (E) Source Replicator and Target probability distribution after adaptation. (F) Target probability distribution after adaptation. (G) Target features after adaptation.

The first row is before adaptation and the second row is my approach. (A) and (B) denote the output probabilities of the source and target data respectively using a model trained on the source. (C) shows the target features using the source-trained model. (D) exhibits GAP can replicate the source probability variations. (E) I train the model to have the same variations with target data. (F) and (G) are the target

Method	A→D	A→W	D→A	D→W	W→A	W→D	Mean
CDAN+E	92.9	94.1	71.0	98.6	69.3	**100.0**	87.7
ALDA	**94.0**	95.6	72.2	97.7	72.5	**100.0**	88.7
SymNets	93.9	90.8	74.6	**98.8**	72.5	**100.0**	88.4
TADA	91.6	94.3	72.9	98.7	73.0	99.8	88.4
MADA	87.8	90.0	70.3	97.4	66.4	99.6	85.2
MDD	93.5	94.5	74.6	98.4	72.2	**100.0**	88.9
CDAN+TransNorm	**94.0**	**95.7**	73.4	98.7	74.2	**100.0**	**89.3**
Source	68.9	68.4	62.5	96.7	60.7	99.4	76.1
DANN	79.7	82.0	68.2	96.9	67.4	99.1	82.2
GAP	84.7	89.3	72.9	97.2	72.8	99.2	86.0
GAP + KD	88.8	91.8	75.4	97.6	73.5	99.8	87.8
Source + LS	80.8	76.9	60.3	95.3	63.6	98.7	79.3
DANN + LS	78.7	86.3	69.0	94.7	71.5	99.4	83.3
GAP + LS	92.2	92.6	**78.0**	98.2	74.0	**100.0**	89.2
GAP + LS + KD	90.0	93.6	74.5	97.7	**74.4**	**100.0**	88.4

Table 2.13: Results on the Office-31 dataset with source present with ResNet-50.

outputs and features after adaptation. Based on these plots, it can be observed that GAP ends up clustering the penultimate layer features as well.

2.3.4 Domain Adaptation with Source Data

I evaluate GAP for regular unsupervised domain adaptation. In this scenario, I do not need to train a source replicator. Instead, I can align the target with actual source probabilities and train the network on source data as well. I use the popular Gradient Reversal layer and adapt it to my approach for these experiments. Specifically, the Discriminator is trained to discriminate between source and target output probabilities using the mean-squared loss function. The network is trained using reversed gradients from the Discriminator. In this scenario, I do not freeze the classification layer. I use Office-31 and Office-Home datasets for these experiments.

The experiment results are in Table 2.13 for Office-31 and Table 2.14 for Office-Home dataset. I compare GAP with domain adaptation methods like CDANN [Chen *et al.* (2019)], SymNets [Zhang *et al.* (2019)], and TADA [Wang *et al.* (2019a)] in the top section. In my approach, I consider both the cases of training the network - without label smoothing (midsection) and with label smoothing (bottom section). In the mid and bottom sections, the first row denotes the performance of source-only models. The second-row results are when I align the deep features using the vanilla DANN loss function for baseline comparison [Ganin *et al.* (2016b)]. The third row shows the performance of GAP. In the last row, I combine my approach with knowledge distillation [Hinton *et al.* (2015)]. I soften the probabilities using a temperature equal to 2 to align the inter-class relationships between the domains. I do not use temperature for the source-free experiments as the logit space between the source and target is not the same. Hence, using a temperature hurts the performance.

GAP outperforms all the listed baselines for source-present scenarios. The compared approaches use advanced techniques like attention [Wang *et al.* (2019a)] or complex architectures like SymNets [Zhang *et al.* (2019)], whereas the GAP is a simple model based on adversarial alignment. Knowledge distillation boosts the performance when no label smoothing is used. Using them together results in a negative transfer as label smoothing places deep features of the classes at equal distances from each other and disturbs the inter-class relationships [Müller *et al.* (2019)].

Source	Ar			Cl			Pr			Rw			Mean
Target	Cl	Pr	Rw	Ar	Pr	Rw	Ar	Cl	Rw	Ar	Cl	Pr	
CDAN+E	50.7	70.6	76.0	57.6	70.0	70.0	57.4	50.9	77.3	70.9	56.7	81.6	65.8
ALDA	53.7	70.1	76.4	60.2	72.6	71.5	56.8	51.9	77.1	70.2	56.3	82.1	66.6
SymNets	47.7	72.9	78.5	64.2	71.3	74.2	64.2	48.8	79.5	74.5	52.6	82.7	67.6
TADA	53.1	72.3	77.2	59.1	71.2	72.1	59.7	53.1	78.4	72.4	60.0	82.9	67.6
MDD	54.9	73.7	77.8	60.0	71.4	71.8	61.2	53.6	78.1	72.5	**60.2**	82.3	68.1
Source	34.9	50.0	58.0	37.4	41.9	46.2	38.5	31.2	60.4	53.9	41.2	59.9	46.1
DANN	45.6	59.3	70.1	47.0	58.5	60.9	46.1	43.7	68.5	63.2	51.8	76.8	57.6
GAP	54.5	71.1	78.5	61.6	73.7	71.7	61.6	54.2	80.4	72.5	57.4	83.8	68.4
GAP + KD	56.5	72.6	79.2	64.5	74.3	74.0	64.8	**57.7**	82.3	74.2	57.4	84.7	70.2
Source + LS	44.6	67.3	74.8	52.7	62.7	64.8	53.0	40.6	73.2	65.3	45.4	78.0	60.2
LS + DANN	50.9	59.3	73.0	54.6	67.2	69.8	55.2	53.7	76.0	68.4	58.7	79.3	63.8
GAP + LS	**57.5**	**74.3**	78.7	**65.6**	74.5	75.2	65.4	57.1	81.6	**75.4**	59.8	**85.6**	**70.9**
GAP + LS + KD	55.6	73.3	78.1	64.9	72.6	74.5	65.0	57.6	81.4	73.7	59.6	84.6	70.1

Table 2.14: Results on the Office-Home dataset with source present with ResNet-50.

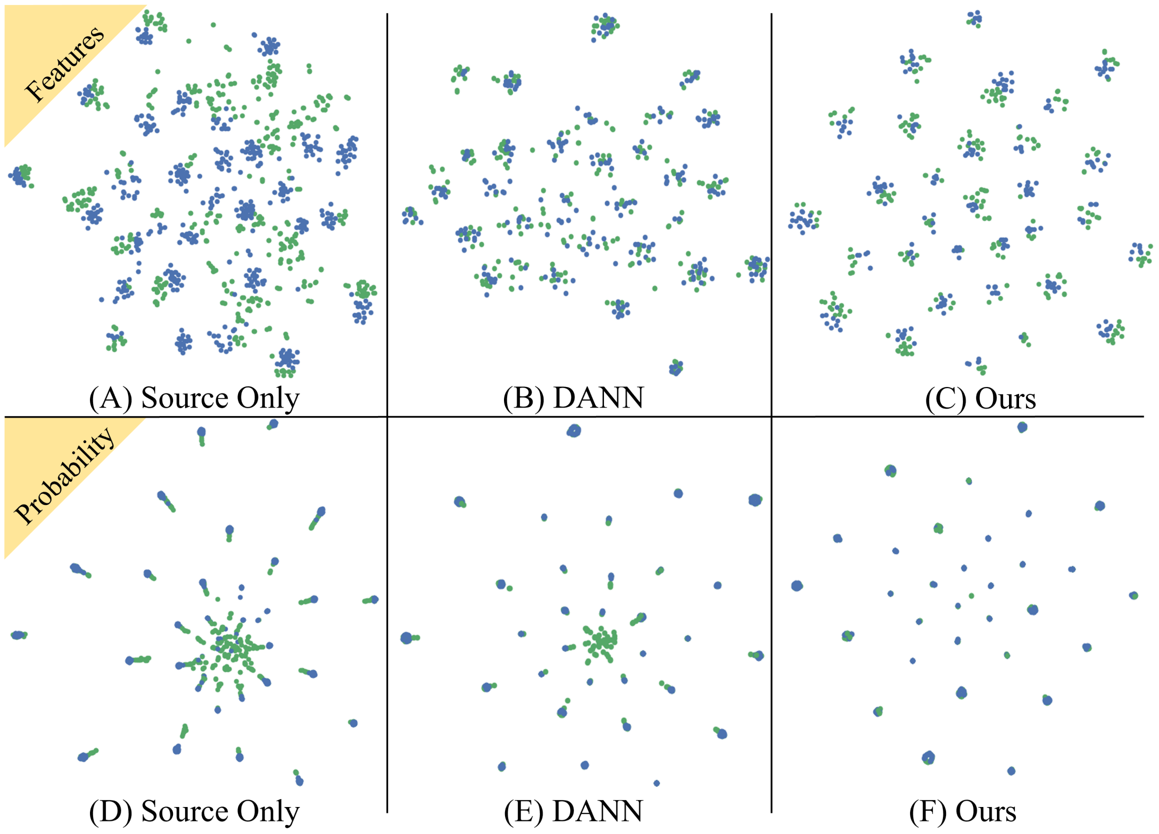


Figure 2.17: TSNE visualization of the penultimate layer features (top row) and the output probability space (bottom row) for Office31 - Amazon → Webcam with source present setting. Blue denotes Source domain (Amazon) and Green denotes Target domain (Webcam).

I also show the TSNE plots of the penultimate features and the probability outputs for Office-31 Amazon to Webcam in Figure 2.17. GAP aligns the probabilities better than DANN and results in similar feature alignment [Ganin *et al.* (2016b)] without explicitly aligning it.

2.3.5 Limitations

Although GAP has several advantages and yields good performance, I discuss the scenarios where it can underperform. GAP is based on $p_s(y) \approx p_t(y)$ assumption, and my experiments on multiple datasets that it is a reasonable assumption and works for the majority of the cases. However, this can affect the performance negatively when the target labels follow a long-tail distribution and would affect the baseline methods as well. On the other hand, if the distribution were known, it could be used as a prior to enhance the alignment. GAP assumes the source and target have identical label spaces. In its current form, GAP cannot be extended to *OpenSet* and *Partial* domain adaptation cases where the label spaces of the source and target are not identical. Exploring these settings for my method can be an interesting future direction.

Chapter 3

SELF-SUPERVISED LEARNING

Self-supervised learning approaches empower neural networks to extract rich features from data autonomously without relying on manual annotation or labeling. These learned features demonstrate remarkable transferability, proving valuable across a spectrum of downstream tasks, including classification. Consequently, the adoption of unsupervised training methods for networks is gaining a lot of traction.

Typically, these techniques introduce auxiliary tasks to the network, necessitating understanding the underlying data structure. For instance, [Noroozi and Favaro (2016)] partitioned images into patches and trained networks to solve jigsaw puzzles composed of these rearranged patches. Another approach involves predicting the rotation angle for both entire images and individual patches [Gidaris *et al.* (2018)].

While most existing self-supervised techniques were tailored for Convolutional Neural Networks (ConvNets), the recent rise of Vision Transformer (ViT) has outpaced ConvNets in various image processing domains such as object recognition and segmentation [Dosovitskiy *et al.* (2020)]. Initially devised for Natural Language Processing [Vaswani *et al.* (2017)], Transformers are now extending their prowess to domains like speech, image, and video processing [Li *et al.* (2019); Dosovitskiy *et al.* (2020); Arnab *et al.* (2021)]. However, their advancement primarily hinges on supervised training with extensive labeled data, making them data-hungry models. They excel over ConvNets only when abundant labeled data is available; otherwise, their performance diminishes due to lacking ConvNets' inherent biases like translation equivariance and locality [Dosovitskiy *et al.* (2020)]. Consequently, this underscores the importance of exploring unsupervised training methods for Vision Transformers.

While existing self-supervised techniques can be applied to Vision Transformers, it is crucial to note that Vision Transformers process images differently from ConvNets. ConvNets process images as grids, while Vision Transformers split images into fixed-size square patches and flatten them into vectors to create series-like input data. ConvNets rely on the strides of kernels to grasp the positioning of pixels, while Vision Transformers employ either a learnable or fixed (sinusoidal) positional embedding at the input level to furnish spatial information. ConvNets use kernels to extract features from an image, while Vision Transformers project embeddings into queries, keys, and values and compute self-attention between patches. ConvNets combine image features and classify images using fully connected layers, whereas Vision Transformers use a learnable classification token for classification.

Given these differences, I showcase a novel self-supervised technique called PatchRot for Vision Transformers [Chhabra *et al.* (2022a)]. PatchRot trains a Vision Transformer to predict the rotation of both patches and the image. Given an input image, PatchRot randomly selects either the image or its patches and applies a rotation operation. This design allows the Vision Transformer's classification head to capture global image information while the patch heads focus on local details. Consequently, the classification head predicts the rotation angle of the entire image, while the patch heads predict the rotation angles for their respective patches. This methodology enables PatchRot to learn and extract both global and local features simultaneously. PatchRot stands out as a straightforward yet effective self-supervised technique tailored specifically for Vision Transformers. It comprehensively trains both the image classification and patch heads, enhancing the Vision Transformer's ability to learn meaningful representations from images.

3.1 PatchRot

3.1.1 Problem Definition

Let $x \in [0,1]^{C \times H \times W}$ be the input image, where C, H, and W represent the number of channels, the height, and the width of the image, respectively. The goal here is to train a Vision Transformer T_ϕ with parameters ϕ in an unsupervised manner and extract high-quality features. Vision Transformer T_ϕ preprocess the image x into a sequence of image patches $s(x) = [x^s_1, x^s_2, ..., x^s_N]$, where N is the number of patches. The image patch dimensions are determined by patch size P where $x^s_i \in [0,1]^{C \times P \times P}$ represents the i-th patch of image x and $N = \frac{H}{P} \cdot \frac{W}{P}$. The image height H and width W need to be a multiple of patch size P to ensure that N is an integer.

3.1.2 Method

PatchRot is a novel technique to train a Vision Transformer where the rotation angle of an image and the rotation angle of individual patches of an image are predicted. RotNet [Gidaris *et al.* (2018)] showcased that a ConvNet trained to predict the rotation of an image learns features as good as those learned using supervised training. In this case, I train a Vision Transformer where the classification head (generally used for predicting the object category) is used to predict the rotation angle of the image. I use the last encoder output of the other heads to predict the rotation angles for the individual patches using new multilayer perceptron (MLP) heads. This way, the Vision Transformer can produce an output for every element in the input sequence, whereas ConvNets are limited to just one output. Therefore, PatchRot applies only to Vision Transformers.

I represent the rotation operation as $R(;\iota)$, where $\iota \in \{0,1,2,3\}$ that rotates the input by $\theta = 90°.\iota$ resulting in rotation of 0°, 90°, 180°, or 270°. A rotated image

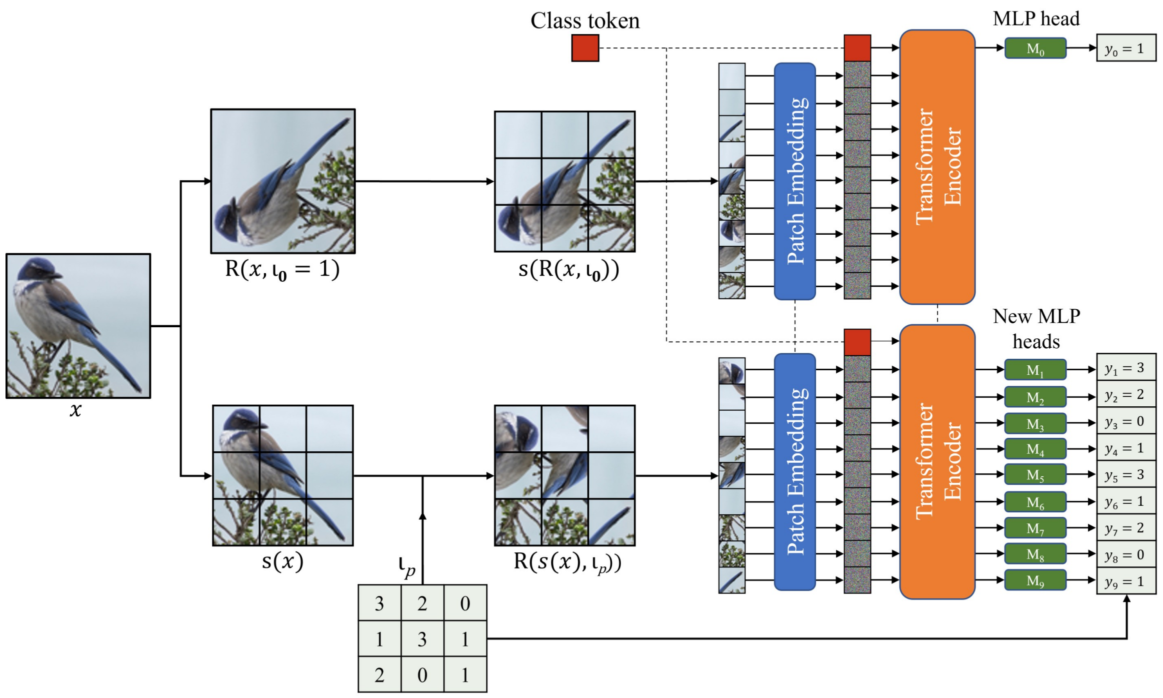


Figure 3.1: Model diagram of PatchRot. The input image is split into patches. The image and the patches are rotated by a random angle of $90°.\iota$ resulting in rotation by $0°, 90°, 180°$, or $270°$. The vision transformer (ViT) is trained to predict the rotation angle for the image and each patch using new additional MLP heads. Dotted lines denote the weights are shared.

$x_r = R(x; \iota_0)$ uses $y = \iota$ as the training label. Similar to [Gidaris *et al.* (2018)], I found that training using all four angles of rotations in the same minibatch yielded the best results. I overload the rotation operator to represent the PatchRot version of the image $x_{pr} = R(s(x); \iota_p)$, where I rotate all of the N patches of the input image. $\iota_p = \{\iota_1, \iota_2, ..., \iota_N\}$ denotes the sequence of rotation applied to the corresponding input sequence $s(x)$ and each $\iota_i \; \forall i = 1, 2, ..., N$ is sampled randomly using a discrete uniform distribution.

I train the Vision Transformer T_ϕ using x_r and x_{pr} images. The Vision Transformer

$T_\phi = PE \odot EB \odot M$ consists of a patch encoding block PE followed by encoder blocks EB_i where $i \in \{1, 2, ..., e\}$ and e is the total number of encoder blocks, and an MLP head M for classification. Let $E(s(x)) = \{E_0(s(x)), E_1(s(x)), ..., E_N(s(x))\}$ denote the output at the last encoder block EB_e where $E_i(s(x)) \in \mathbb{R}^h$ is the i-th output corresponding to the input sequence $s(x)$ and h is the embedding size. $E_0(s(x))$ represents the encoding of the classification head and $\{E_1(s(x))$, ..., $E_N(s(x))\}$ are the encoding for the patch heads. In the case of image rotation x_r, the encoding from the classification head is passed to the MLP head M_0 to predict the rotation category of the image as $M_0(E_0(s(x_r)))$. The encodings of the patch heads, in this case, are ignored. In the case of patch rotation, I introduce new MLPs $M_1, M_2, ..., M_N$ to classify the encodings of the individual patches. The MLPs predict the rotation angles of the individual patches as $M_i(E_i(s(x_{pr})))\ \forall i = 1, 2, ..., N$. PatchRot trains the Vision Transformer using,

$$\sum_{j=0}^{3} \mathcal{L}_{ce}(M_0(E_0(s(R(x, j)))), y = j) + \sum_{i=1}^{N} \mathcal{L}_{ce}(M_i(E_i(R(s(x), \iota_p))), y = \theta_i), \tag{3.1}$$

where the first term is the loss function penalizing rotation misclassification for each of the four angles of image rotation, and the second term is the loss function penalizing rotation misclassification for the image patches. $\mathcal{L}_{ce}$ is the standard cross-entropy loss function. Note that I do not rotate the image when rotating image patches, as doing both simultaneously has been demonstrated to hurt the downstream task performance.

3.1.3 Training Procedure

To avoid the possibility of the network learning to predict the angle of patch rotations using edge continuity, I use a buffer gap B between the patches, i.e., I initially partition the image using a larger patch size $P' = P + B$, where $P' > P$. Then, I randomly crop a patch of size P from each such patch. This results in a gap between patches of random size between 0 to $2B$ pixels. Due to the buffer between patches, the input size is reduced. Instead of scaling the image/patches to adjust to the original input size, I found it beneficial to perform the self-supervised training on reduced image size and use the original size for transferring knowledge to downstream tasks. To be precise, the PatchRot training image x_{pr} size is $C \times H_{pr} \times W_{pr}$ where $H_{pr} = P \cdot \lfloor \frac{H}{P+B} \rfloor$, and $W_{pr} = P \cdot \lfloor \frac{W}{P+B} \rfloor$ and $\lfloor . \rfloor$ denotes the floor operation. The number of patches is given by $N_{pr} = \lfloor \frac{H}{P+B} \rfloor \cdot \lfloor \frac{W}{P+B} \rfloor$. For creating a rotated image x_r, I produce a crop of this size instead of the original size for my algorithm and then rotate the cropped image. I was guided by the fact that training on smaller resolution images and then fine-tuning with higher resolution images has shown to yield performance gains [Touvron *et al.* (2019)].

Once the network is trained using PatchRot, I remove the newly added MLP heads - $M_1, M_2, ..., M_N$ and only use the classification head's encoder output and its MLP head (just like the original Vision Transformer) for downstream tasks. For adapting the network to the new classification task, I replace the last classification layer from MLP head M_0 with a new layer having an output size equal to the number of categories in the new task before retraining the network. PatchRot training is performed at a smaller input size, but I use the original image size for the downstream task. The larger image size results in an increase in the number of input patches: $\lfloor \frac{H}{P+B} \rfloor \cdot \lfloor \frac{W}{P+B} \rfloor \rightarrow \frac{H}{P} \cdot \frac{W}{P}$. Hence, for this case, I apply a linear interpolation of the positional embedding as designed in the original Vision Transformer [Dosovitskiy *et al.* (2020)].

3.2 Experiments

3.2.1 Datasets

I test my approach on the following object recognition datasets:

1. **CIFAR10** [Krizhevsky *et al.* (2009)] is a popular small-sized object recognition dataset with 32×32 image size and ten classes. I used a Random crop with zero padding of size four and random horizontal flip augmentations for training.

2. **CIFAR100** [Krizhevsky *et al.* (2009)] is another popular small-sized object recognition dataset with 32×32 image size but contains 100 classes. I used a Random crop with zero padding of size four and random horizontal flip augmentations for training.

3. **FashionMNIST** [Xiao *et al.* (2017a)] is a 32×32 grayscale dataset with 10 classes of clothing items. I did not use any augmentations for this dataset.

4. **Tiny-ImageNet** represents a medium-sized dataset formed using a subset of ImageNet. It contains images of size 64×64 and 200 classes. I used a Random crop with zero padding of size four and random horizontal flip augmentations for training.

5. **Animal-10N** [Song *et al.* (2019)] presents a fine-grain classification of 5 pairs of animals with noisy training labels, and I used an image size of 64×64. I used a Random crop with zero padding of size four and random horizontal flip augmentations for training.

6. **SVHN** [Netzer *et al.* (2011)] contains images of real digits extracted from House numbers. I use this dataset to test rotation invariant classes like 0, 1, and 8. Training a network to predict image rotation angles for rotation invariant classes

is not helpful [Feng *et al.* (2019)].

For all of the experiments, I used a scaled-down version of the original Vision Transformer (described in section 3.2.2) due to hardware limitations.

3.2.2 Training Details

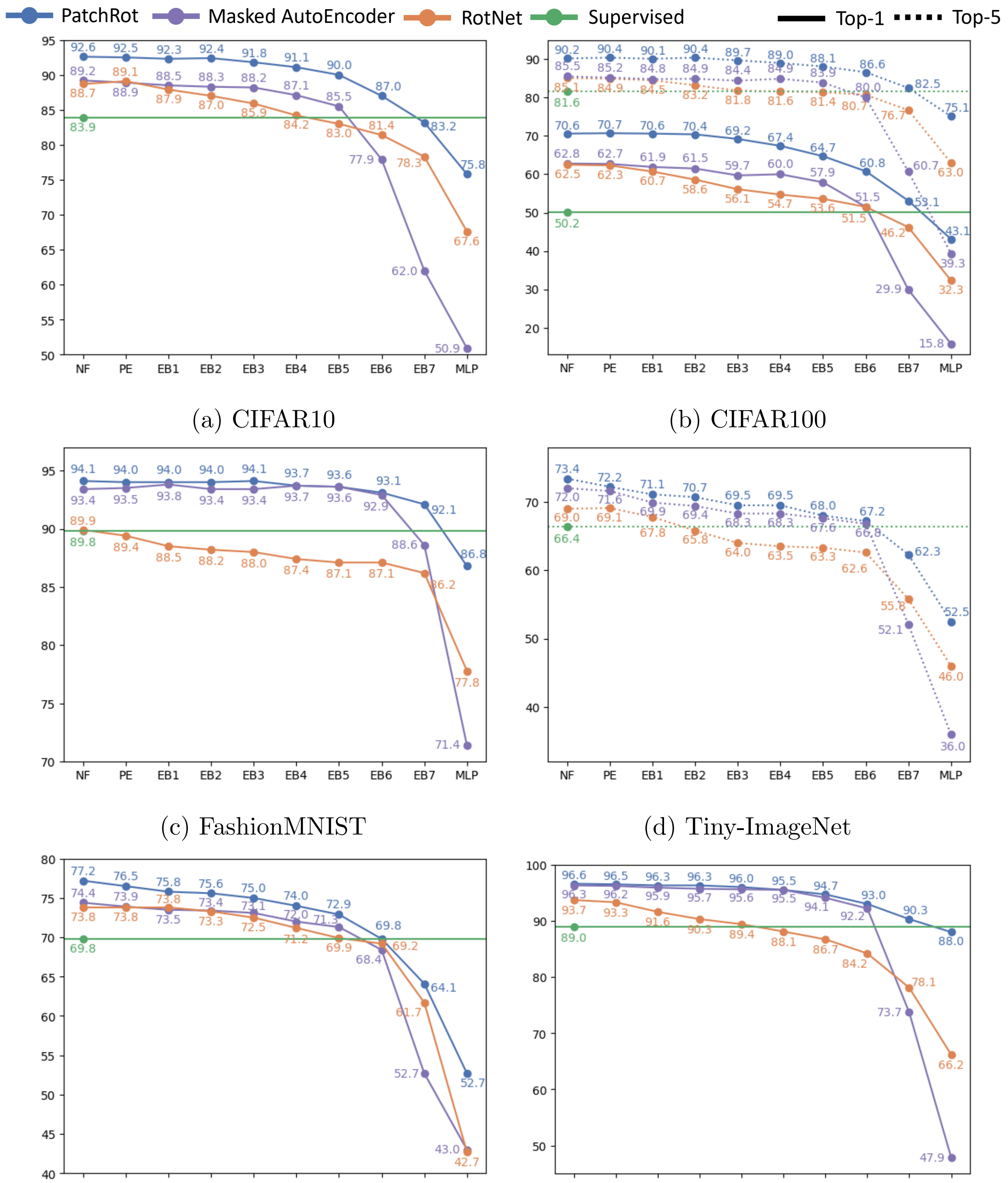
The experiments are performed using ViT-Lite from [Hassani *et al.* (2021a)] (a scaled-down version of the original Vision Transformer architecture [Dosovitskiy *et al.* (2020))], which is designed for smaller datasets due to limited hardware. Specifically, the number of encoder blocks is reduced to 6 with 256 embedding size and 512 expansion size. The number of attention heads is also reduced to 4, and the dropout is set to 0.1. The new MLP heads are identical to the original MLP head and consist of a 256×256 hidden layer and a classification layer of size 256×4. I train the network with the patch size of $P = 4$ for CIFAR10, CIFAR100, FashionMNIST, and SVHN and $P = 8$ for Tiny-ImageNet and Animal-10N. The buffer B is set to $\frac{1}{4}$ of the patch size, which results in PatchRot input size of 24×24 for 32×32 images and 48×48 for 64×64 images.

I used Adam Optimizer for training the Vision Transformer with a learning rate and a weight decay of 5×10^{-4} and 3×10^{-2}, respectively. The learning rate is warmed up for the first ten epochs and then decayed using a cosine schedule. The batch size is set to 128. These training hyper-parameters were taken from ViT-Lite [Hassani *et al.* (2021a)] and were kept the same across all experiments. However, due to multiple variations of the same image being presented in a mini-batch, the effective batch size for PatchRot is 128×5 samples. I follow the procedure of training the layers after the self-supervision training, as also implemented by [Noroozi and Favaro (2016)] for the experiments. The self-supervised training is performed for 300 epochs, and the supervised training is performed for 200 epochs.

Since the baselines are not available for my setting, I implemented the most suitable ones for comparison: RotNet [Gidaris *et al.* (2018)] and Masked Autoencoder (MAE)[He *et al.* (2022)]. RotNet [Gidaris *et al.* (2018)] was originally proposed for ConvNets, but I used it here by performing rotation prediction on the image using the Vision Transformer network. It has outperformed basic methods like Split-brain[Zhang *et al.* (2017)], Jigsaw [Noroozi and Favaro (2016)], etc., and also serves as my direct baseline. The Masked Autoencoder [He *et al.* (2022)] is a recent self-supervised approach for vision transformers. I used the code from their official repos and trained them with the same setting as PatchRot. The approach-specific hyper-parameters were taken from the baselines: 4 rotation angles for RotNet and 75% masking ratio for MAE.

3.2.3 Results

The results are in Figure 3.2 where Solid lines denote Top1 accuracy, and dashed lines denote Top5 accuracy. The X-axis shows the different layers of the transformer and signifies which layers are trained/frozen. NF represents no layers are frozen. PE represents only the patch embedding block is frozen. EB-i represents the encoder blocks. While training, all the layers before those layers are frozen (including the Patch Embedding block). MLP denotes The whole network is frozen except for the last classification layer. It can be observed that MAE and RotNet achieve comparable performance, but the performance of MAE drops significantly when training only the last few layers. This happens as those layers become generation task-specific. PatchRot, on the other hand, outperforms baselines for finetuning and linear probing (freezing the whole network denoted by MLP in my experiments) on all the datasets by a significant margin. Fine-tuning just the MLP head (one fully connected layer)



(a) CIFAR10 (b) CIFAR100

(c) FashionMNIST (d) Tiny-ImageNet

(e) Animal-10N (f) SVHN

Figure 3.2: Classification accuracies for **Supervised**, **RotNet**, **Masked AutoEncoder**, and **PatchRot**

of PatchRot achieves results close to the supervised learning, and fine-tuning one encoder block and MLP head outperforms the supervised training from scratch.

My results on the Animal-10N dataset showcase that the features extracted by PatchRot contain detailed information. These features can perform fine-grain classification and handle noise robustness better than the compared baselines. Similarly, in the case of the rotation invariant task (SVHN), it can be observed that the performance of RotNet and MAE drops quickly (f), but PatchRot is unaffected. Similar to objects, patches can be rotation-invariant too. However, such patches are generally part of the background and do not contain any object information. Training the network on them did not affect its final performance. The combination of global and local information captured by PatchRot helps overcome all such problems.

3.3 Analysis

3.3.1 Ablation Study

In this section, I test the importance of different components of PatchRot using CIFAR10 and present the results in Table 3.1. First, I train the Vision Transformer on the patch rotation image x_{pr} only and exclude the image rotation x_r versions (No ImageRot). As I do not train the classification head, the network learns only the local characteristics. As I freeze more layers, it can be noticed the drop in performance signifies the importance of global context. Similarly, I exclude the x_{pr} (No PatchRot) in the second experiment. This is similar to RotNet, with the only difference being that the self-supervised training is performed at a reduced image size. I also test removing the buffer between the patches (No Buffer). Without a buffer, networks tend to learn trivial shortcuts like edge continuity, and it can be observed that training the last few layers results in significantly lower performance. Next, I test the significance

Initialization	NF	PE	EB1	EB2	EB3	EB4	EB5	EB6	EB7	MLP
No ImageRot	91.8	91.9	91.9	91.4	91.0	90.0	88.8	82.2	69.4	54.9
No PatchRot	91.0	91.2	90.7	90.2	89.4	88.6	87.8	85.3	80.1	70.8
No Buffer	91.8	91.9	91.6	91.0	89.4	87.6	84.7	80.3	75.9	65.8
Original Size	92.1	92.2	91.6	91.1	90.7	89.9	89.0	86.2	81.6	73.9
Rotate Img & Patch	90.7	90.7	90.8	90.6	90.6	90.2	88.3	82.5	72.8	58.4
Reuse MLP head	91.1	91.0	90.9	90.4	89.9	89.6	87.7	84.0	76.6	65.7
PatchRot-full	**92.6**	**92.5**	**92.3**	**92.4**	**91.8**	**91.1**	**90.0**	**87.0**	**83.2**	**75.8**

Table 3.1: Top-1 classification accuracies on Ablation study of my method using CIFAR-10. Columns denote the parts of the Vision Transformer being fine-tuned. While finetuning on the downstream task, all the layers before that layer are frozen. NF: No layers are frozen; PE: Patch Embedding block; EB-1 to EB-7 are the encoder blocks; MLP: The whole network is frozen except for the new output classification layer.

of reduced-size training by comparing it with self-supervised training on the original image scale (Original Size). To test this, I divide the image using the original patch size P, and I randomly crop $P - B$ sized patches from it. These cropped patches are then resized to patch size P to maintain the original image size and still have a buffer.

I also test my hypothesis of rotating images and patches together (Section 3.1.2) in my next experiment (Rotate Img & Patch). Last, I experiment with the newly added MLP heads (Reuse MLP Head). Since adding new MLP heads temporarily increases the model size. Instead of adding new MLP heads, I use the same original MLP head to predict the image's rotation angle and all the patches. The results showcase that each component plays a vital role in the PatchRot training of the Vision transformer.

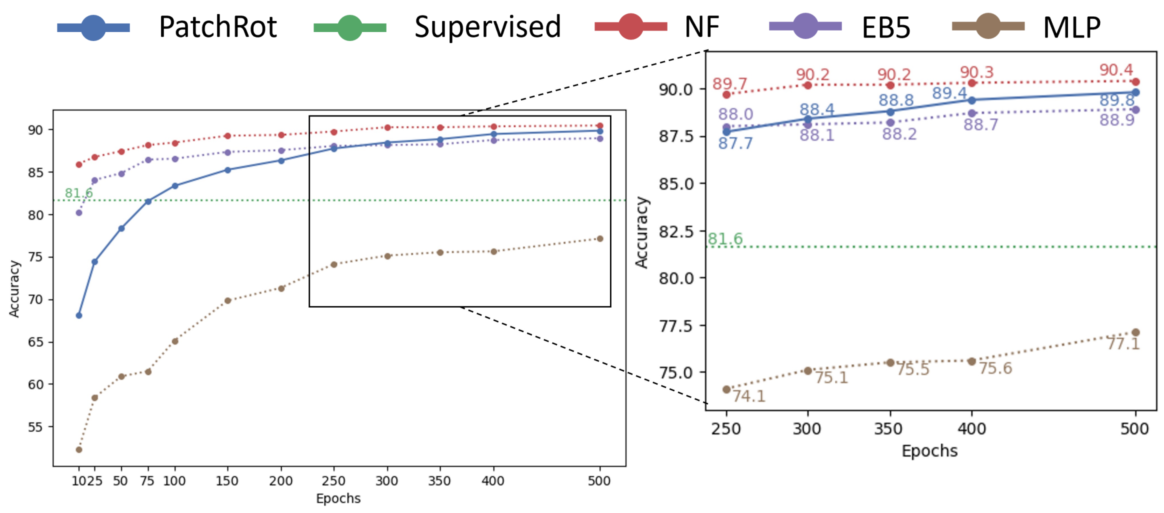


Figure 3.3: Top-5 classification accuracy vs. Pretraining epochs for CIFAR-100. Supervised denotes the supervised accuracy; PatchRot denotes the accuracy of PatchRot self-supervised task on test set; NF, EB5 and MLP denotes the accuracy on finetuning all the layers, freezing layers till EB5 and no fine-tuning of the network respectively.

3.3.2 Pre-training epochs vs. Classification Accuracy

Here, I study the impact of PatchRot's pretraining on the final classification performance. I train the Vision Transformer using my approach with a different number of epochs {10, 25, 50, 75, 100, 150, 200, 250, 300, 350, and 400} for the CIFAR100 dataset.

The results for this experiment are in Figure 3.3. The dotted lines capture the final object classification accuracy. I can observe that pretraining with just a few epochs can significantly improve final performance compared to training from random (Supervised). The blue curve is different from the other curves, and it represents the image and patch-level rotation accuracy of CIFAR100's test set. It shows that my self-supervised task doesn't overfit, and training it longer (350, 400, and 500) improves the accuracy of the self-supervised task test and the final classification.

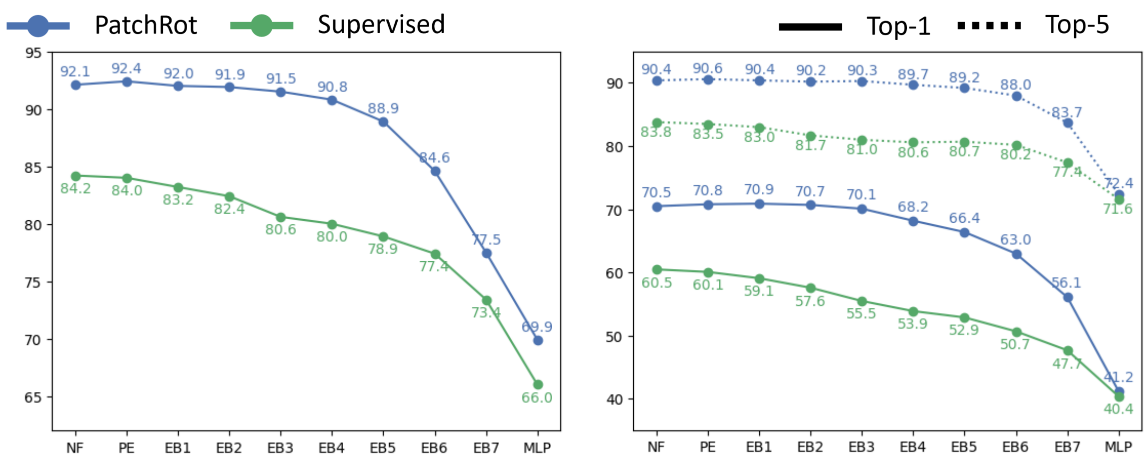
3.3.3 Application to Transfer Learning

PatchRot aims to learn rich features in an unsupervised manner, and this section shows its advantage for the transfer learning settings on CIFAR100 $\rightarrow$ CIFAR10 and CIFAR10 $\rightarrow$ CIFAR100 tasks. I first train two networks: one on source data using PatchRot and the other on its supervised object classification task. These networks are then used to initialize training on the target dataset.

The results for these experiments are in Figure 3.4. Solid lines denote Top1 accuracy, and dashed lines denote Top5 accuracy. The X-axis shows the different layers of the transformer and signifies which layers are trained/frozen. NF represents no layers are frozen. PE represents only the patch embedding block is frozen. EB-i represents the encoder blocks. While training, all the layers before those layers are frozen (including the Patch Embedding block). MLP denotes The whole network is frozen except for the last classification layer. It can be observed that PatchRot extracts better features than the supervised training, which can be used across datasets. Also, PatchRot has a significant margin advantage over supervised training, which drops only during the last few layers (where layers become domain/task-specific). Hence, pre-training the network with PatchRot has a significant advantage over supervised training.

3.3.4 Application to Semi-supervised learning

Self-supervised learning techniques are being popularly utilized for semi-supervised learning by training the network with labeled data and simultaneously training another output head of the network with the unlabeled data using the self-supervised loss [Gidaris *et al.* (2018); Zhai *et al.* (2019)]. In my experiments, instead of training the Vision Transformer on both the supervised and PatchRot loss together, I



(a) CIFAR100→CIFAR10 (b) CIFAR10→CIFAR100

Figure 3.4: PatchRot performance in a transfer learning setting. PatchRot denotes the ViT was trained on the source dataset self-supervised using PatchRot. Supervised denotes the ViT was trained on source using the supervised object classification task. Solid lines and the dashed lines show the Top-1 and Top-5 accuracy on the Y-axis, respectively. The X-axis contains different layers of the transformer, and while fine-tuning on the target dataset, all the layers before that layer are frozen. NF: No layers are frozen; PE: Patch Embedding block; EB-1 to EB-7 are the encoder blocks of the Vision Transformer. MLP: The whole network is frozen except for the new output linear layer.

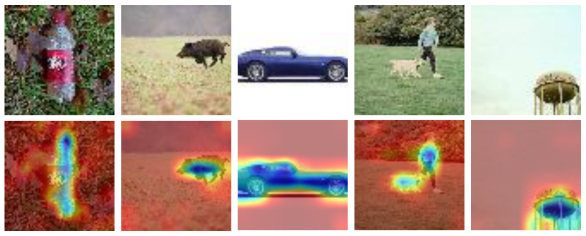
first train the network with PatchRot (self-supervised loss) on all the data and then fine-tune the network using the labeled data only. It is similar to self-supervised experiments, with the only difference being in data used for pertaining and fine-tuning tasks.

I use CIFAR10 for these experiments with {40, 250, 1000, 4000, 10000, 20000, 30000, and 40000} labeled samples, and the results are in Table 3.2. The second column (Sup) denotes the test accuracy on training the network with labeled samples

Labels	Sup	NF	PE	EB1	EB2	EB3	EB4	EB5	EB6	EB7	MLP
40	20.8	38.8	40.1	42.1	44.5	46.9	49.7	50.6	51.3	**55.2**	27.5
250	26.6	57.5	60.2	61.4	63.4	64.9	65.3	**65.8**	65.1	65.2	35.5
1000	38.6	70.1	71.4	71.9	72.6	72.8	**73.8**	73.5	72.0	72.2	58.4
4000	53.7	79.2	79.6	80.7	80.9	**81.1**	80.9	80.2	78.4	77.2	69.7
10000	67.1	84.7	85.2	85.5	**85.6**	85.5	84.8	84.3	82.2	79.6	73.0
20000	75.2	88.3	88.7	**88.9**	**88.9**	88.5	88.0	86.7	84.7	81.1	74.5
30000	79.8	90.0	90.5	**90.6**	90.3	89.9	89.2	88.1	85.9	82.0	75.0
40000	82.0	91.7	**92.0**	91.7	91.6	91.1	90.3	89.2	86.4	82.8	75.6
50000	83.9	**92.6**	92.5	92.3	92.4	91.8	91.1	90.0	87.0	83.2	75.8

Table 3.2: Results on CIFAR10 in a semi-supervised setting. PatchRot training is performed on all the samples, and then the network is fine-tuned with labeled samples only. The first column denotes the number of labeled samples, and the second column shows the supervised performance of labeled samples only. Other columns denote the parts of the Vision Transformer being fine-tuned, and all the layers before that layer are frozen. NF: No layers are frozen; PE: Patch Embedding block; EB-1 to EB-7 are the encoder blocks of the Vision Transformer. MLP: The whole network is frozen except for the new output linear layer.

only from scratch. The other columns show the different layers of the transformer and signify which layers are trained/frozen. NF represents no layers are frozen. PE represents only the patch embedding block is frozen. EB-i represents the encoder blocks. While training, all the layers before those layers are frozen (including the Patch Embedding block). MLP denotes The whole network is frozen except for the last classification layer. As expected, training more layers is beneficial with the


(a) Attention Maps

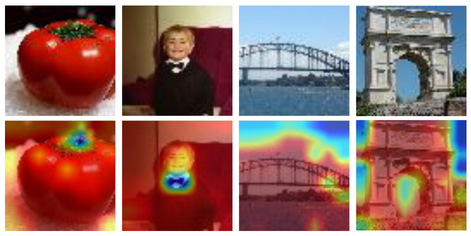

(b) Failure cases

Figure 3.5: Sample Attention Maps of ViT trained using PatchRot on the validation set of Tiny-ImageNet. Upper row: Input and Lower row: Attention Map.

increase in the number of labeled samples. PatchRot pre-training results in superior performance compared to supervised training, which showcases its application in semi-supervised learning.

3.3.5 Attention Maps

I show the attention maps of my Vision Transformer network trained using PatchRot on the validation set of Tiny-ImageNet in Figure 3.5. With just the unsupervised training, it can be observed that the model learns to attend to the main object in the image regardless of its position. I also show failure cases in Figure 3.5b, where the model learns to solve the problem by attending to a part of the object, like the top green part of the tomato, or using a background feature like sky positioning. Figure 3.6 shows more such attention maps.

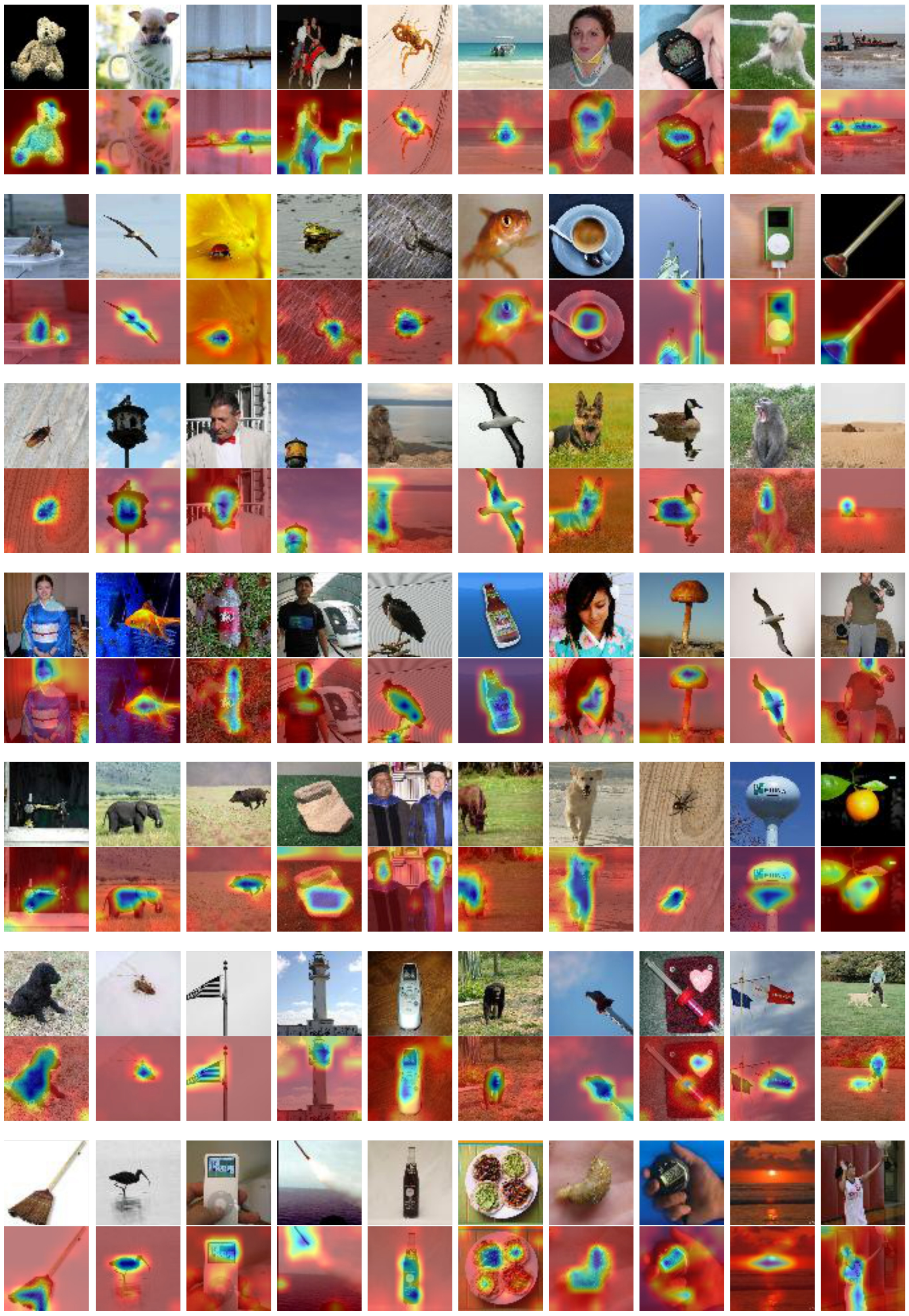


Figure 3.6: More Attention Maps of Vision Transformer trained using PatchRot on the validation set of Tiny-ImageNet.

Chapter 4

INPUT REGULARIZATION

The challenge of limited data presents a significant hurdle for neural networks, particularly impacting Vision Transformers due to their susceptibility to overfitting [Dosovitskiy *et al.* (2020)]. This concern is especially pronounced when training Vision Transformers on small-to-medium datasets, necessitating robust regularization techniques [Dosovitskiy *et al.* (2020)].

Standard regularization techniques like dropout [Srivastava *et al.* (2014)], weight decay[Hanson and Pratt (1988)], label smoothing [Szegedy *et al.* (2016a)], batch normalization [Ioffe and Szegedy (2015)], and data augmentations are widely employed for both Convolutional Neural Networks (ConvNets) and Vision Transformers. However, advanced methods such as Mixup [Zhang *et al.* (2018)] and Cutmix [Yun *et al.* (2019)], while effective, were primarily developed for ConvNets, prompting the need for novel augmentations tailored specifically for Vision Transformers.

While ConvNets and Vision Transformers may appear similar as they both process images to output class probabilities, their internal mechanisms differ significantly. ConvNets operate on the grid structure of an image using filters, whereas Vision Transformers transform images into patches and utilize self-attention among patches for feature extraction. This distinction allows Vision Transformers to maintain a global receptive field at any layer, whereas ConvNets increase their receptive field with depth [Dosovitskiy *et al.* (2020)].

In response to these notable disparities, I introduce a straightforward yet powerful data augmentation method called ‘PatchSwap’ tailored specifically for regularizing Vision Transformers [Chhabra *et al.* (2022b)]. Vision Transformers, as part of their

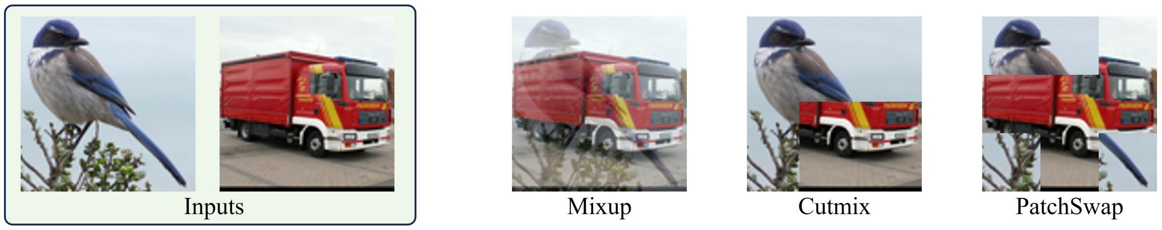


Figure 4.1: Comparison of images generated by Mixup, Cutmix, and PatchSwap.

processing, break down images into a sequence of patches. PatchSwap operates by exchanging some patches between two randomly chosen images, resulting in the creation of a new image (refer to Figure 4.2).

This approach shares certain similarities with Mixup and Cutmix: firstly, it combines two images and their respective labels to generate a fresh image with a new label; secondly, it trains the network to consistently infer this new label based on the combination of the two images. However, a notable divergence lies in PatchSwap's exploitation of Vision Transformers' global receptive field. This feature empowers Vision Transformers to establish connections between any pair of pixels across all network layers. Consequently, even if significant patches are dispersed throughout an image, a Vision Transformer can learn to extract their collective information at each network layer. PatchSwap fully leverages this attribute, distributing the object throughout the image, unlike Mixup and Cutmix. I show sample images generated by Mixup, Cutmix, and PatchSwap in Fig. 4.1.

While this approach benefits Vision Transformers, it does have limitations. Primarily, the number of patches that can be swapped between two images is limited to integer values between 0 and the total number of patches, restricting the mixing ratio compared to Mixup and Cutmix, which use continuous values. Additionally, patches in a PatchSwap image contain only one class, unlike Mixup and Cutmix, which can contain a mix of classes. These constraints limit the number of variations possible between two images. However, given the large number of patches in an image and the potential PatchSwap images, I did not find this to be a significant issue. Note: The total number of possible PatchSwap images between two images is 2^N, where N is the number of patches in an image. Despite these limitations, I found them to be advantageous, as they lead to extensions like Unlabeled-PatchSwap for semi-supervised settings and Inverse-PatchSwap for self-supervised settings.

Unlabeled-PatchSwap is based on the principle of consistency regularization, which has been fundamental to many semi-supervised techniques [Laine and Aila (2016); Tarvainen and Valpola (2017)]. It states that two different versions of an input should yield the same output from a network. Unlabeled PatchSwap operates on the same principle. A PatchSwap image is a combination of two images, and its label is determined by the labels of the individual images, based on the number of patches swapped. Multiple PatchSwap images can be formed by combining a fixed number of patches from the first image with a fixed number of patches from the second image. Consistency regularization ensures that these PatchSwap images have the same target label, even in the absence of label data, allowing the network to output consistent labels for PatchSwap images, regardless of the labels of the original images.

Inverse PatchSwap extends PatchSwap for self-supervised learning. It is based on the concept that when a Vision Transformer is trained to reconstruct an image from its partial view, as in a Masked Autoencoder, the network learns to extract generalizable features [He *et al.* (2022)]. The PatchSwap image is an image formed from partial views of two images. I train a decoder-encoder network to reconstruct the missing patches of both inputs. Since the reconstruction loss is only applied to the missing patches, I term my approach as Inverse PatchSwap. Inverse PatchSwap is a self-supervised extension that improves the quality of extracted features in a self-supervised manner by using a regularized input. The features extracted by such an encoder are of higher quality and contain information from both images at their respective patch locations.

Finally, I combine PatchSwap variants to further improve regularization. PatchSwap++ utilizes Inverse PatchSwap for pre-training before employing PatchSwap during supervised task training. Similarly, Unlabeled PatchSwap++ applies Inverse PatchSwap for pre-training, followed by using PatchSwap on labeled samples and Unlabeled PatchSwap on unlabeled data. Through extensive experiments across various datasets, I demonstrate that PatchSwap and its variants exhibit superior or comparable performance to state-of-the-art methods in diverse settings.

4.1 PatchSwap

4.1.1 Regularization

PatchSwap (abbreviated as PS) represents an innovative data augmentation method specifically designed to enhance the performance of Vision Transformers. In essence, PatchSwap involves the swapping of patches between two input images to produce a new composite image, referred to as the PatchSwap image. The Vision Transformer is trained to predict the category distribution of the PatchSwap image.

Let's consider x_1 and x_2 as two random images in $R^{C \times H \times W}$, accompanied by labels y_1 and y_2 respectively. Here, C represents the number of channels, while H and W denote the height and width of the images. The labels, y_1 and y_2 are represented as one-hot encoded vectors. A Vision Transformer T_θ with a parameters θ and patch size P, partitions an image x into $P \times P$ equally-sized patches denoted as $x^p = [x^1, x^2, ..., x^N]$. Here, each x^i signifies the i^{th} patch of the image x, with N representing the total number of patches. It's important to note that P must be a factor of H and W to ensure that the total number of patches created, $N = \frac{H}{P} \times \frac{W}{P}$,

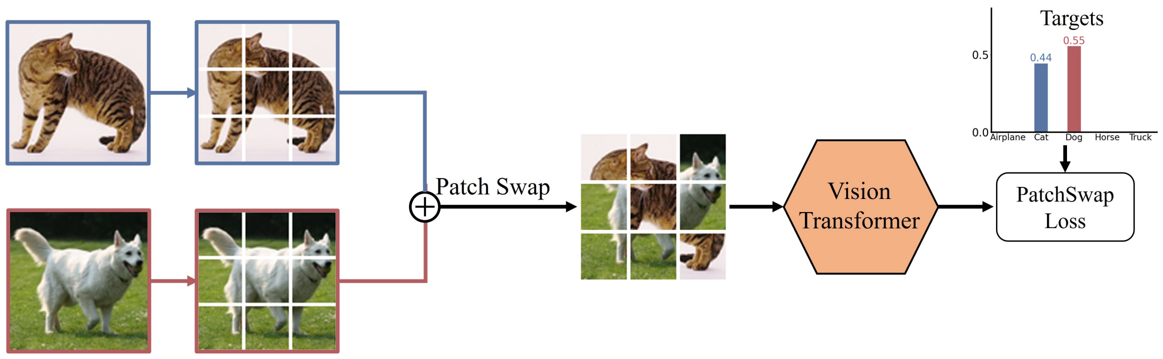


Figure 4.2: Overview of PatchSwap. Images are split into patches, and some of the patches are randomly swapped between two images to create a PatchSwap image. Vision Transformer is trained to predict mixed targets.

is an integer.

The creation of a PatchSwap image x_{ps} involves the exclusive selection of patches from either $x_1^p = [x_1^1, x_1^2, ..., x_1^N]$ or $x_2^p = [x_2^1, x_2^2, ..., x_2^N]$, facilitated by a binary mask $M = [M^1, M^2, \dots, M^N] \in 0, 1^N$. This PatchSwap image x_{ps} is formulated as:

$$x_{ps}(x_1; x_2; \lambda) = (1 - M) \cdot x_1^p + M \cdot x_2^p, \tag{4.1}$$

Here, '$\cdot$' denotes element-wise multiplication between a patch x^i and its corresponding mask element M^i, while '+' indicates the element-wise addition of patches. This means that the PatchSwap image x_{ps} includes the i^{th} patch from x_1 if $M^i = 0$, or from x_2 if $M^i = 1$. The binary mask M is generated based on the swapping ratio $\lambda \in [0, 1]$ which is sampled from a Beta distribution $\lambda \sim Beta(\alpha, \alpha)$. α is a hyperparameter that controls the shape of this distribution. To ensure a discrete number of patches to be swapped (an integer between 0 and N), I estimate the number of swapped patches to be $n = \texttt{round}(\lambda.N)$ where '.' is a multiplication operation and update the swapping coefficient to $\lambda' = \frac{n}{N}$. A random binary mask M is then generated, such that $n = \texttt{sum}(M)$, enabling the combination of x_1 and x_2 to create

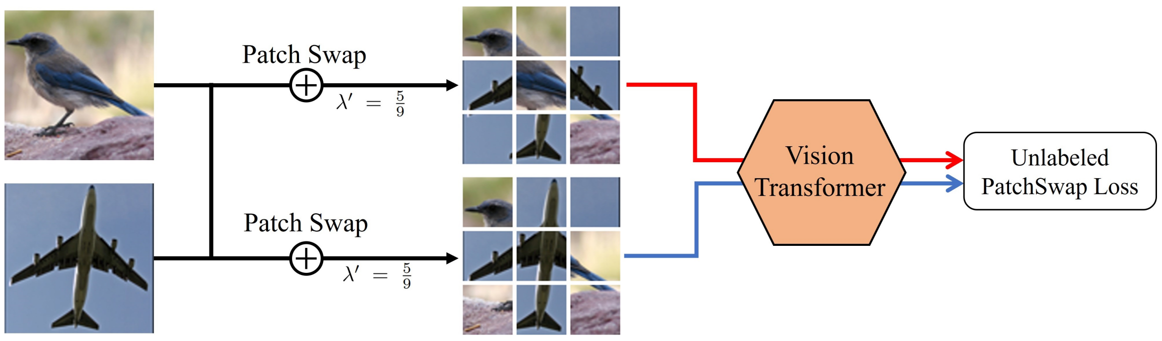


Figure 4.3: Overview of Unlabeled PatchSwap. Two distinct PatchSwap images are created such that they have the same swapping ratio λ' and the Vision Transformer is trained to produce consistent output for them.

the PatchSwap image x_{ps}. The process of creating a PatchSwap image with ith $\lambda' = \frac{5}{9}$ and $N = 9$ is demonstrated in Figure 4.2. The Vision Transformer T_θ is trained using the PatchSwap image x_{ps}, employing the following loss function:

$$\mathcal{L}_{ps}(x_1; y_1; x_2; y_2; \lambda) = -\,(1 - \lambda') \cdot y_1 \log T_\theta(x_{ps}) - \lambda' \cdot y_2 \log T_\theta(x_{ps}) \tag{4.2}$$

4.1.2 Unlabeled PatchSwap for Semi-supervised Learning

PatchSwap, an image transformation method, operates exclusively when image labels are accessible. However, to address scenarios where some labels are absent, I introduce Unlabeled PatchSwap (abbreviated as UPS) as a technique applied towards semi-supervised learning. Unlabeled PatchSwap is founded on the principle of consistency regularization, asserting that two altered versions of the same input should produce identical outputs when traversing through a neural network [Laine and Aila (2016); Tarvainen and Valpola (2017)]. The training mechanism incentivizes the network to generate similar outputs for these two perturbed inputs, often achieved through loss functions like Kullback-Leibler divergence or mean-squared error [Laine and Aila (2016); Tarvainen and Valpola (2017)].

In the context of Unlabeled PatchSwap, let x_{ps_1} and x_{ps_2} represent two PatchSwap images derived from x_1 and x_2, sharing a common swapping ratio λ' but constructed using distinct masks, M_1 and M_2, where $M_1 \neq M_2$. These masks select diverse sets of patches from x_1 and x_2 to form x_{ps_1} and x_{ps_2}, ensuring $x_{ps_1} \neq x_{ps_2}$. Nonetheless, the identical swapping ratio λ' implies that both images contain the same proportion of patches from x_1 and x_2. As the number of swapped patches defines the target label, I constraint x_{ps_1} and x_{ps_2} to exhibit an identical output label.

This concept is illustrated in Figure 4.3, demonstrating the creation of two distinct PatchSwap images characterized by $\lambda' = \frac{5}{9}$. Adhering to the consistency principle, the Vision Transformer T_θ is trained to generate the same outputs for x_{ps_1} and x_{ps_2}, governed by the equation:

$$\mathcal{L}_{ups}(x_1; x_2; \lambda) = ||T_\theta(x_{ps_1}) - T_\theta(x_{ps_2})||^2 \tag{4.3}$$

In the case of semi-supervised learning scenarios, where a subset of data D_l is labeled, and another subset D_u is unlabeled, I apply PatchSwap regularization loss on the labeled subset D_l and use unlabeled PatchSwap on the unlabeled data D_u. Hence, the comprehensive loss function for semi-supervised training is defined as:

$$\mathop{\mathbb{E}}_{\substack{(x_1,y_1)\sim D_l \\ (x_2,y_2)}} \mathcal{L}_{ps}(x_1; y_1; x_2; y_2, \lambda) + \quad \gamma \mathop{\mathbb{E}}_{x_1,x_2\sim D_u} \mathcal{L}_{ups}(x_1; x_2; \lambda) \tag{4.4}$$

Here, γ serves as a hyper-parameter regulating the significance of the unlabeled PatchSwap loss within the overall objective.

4.1.3 Inverse PatchSwap for Self-supervised Learning

If the training labels happen to be completely absent, training the network solely on Unlabeled PatchSwap and no supervised loss component tends to converge toward a suboptimal solution that does not learn anything. For such cases, I introduce

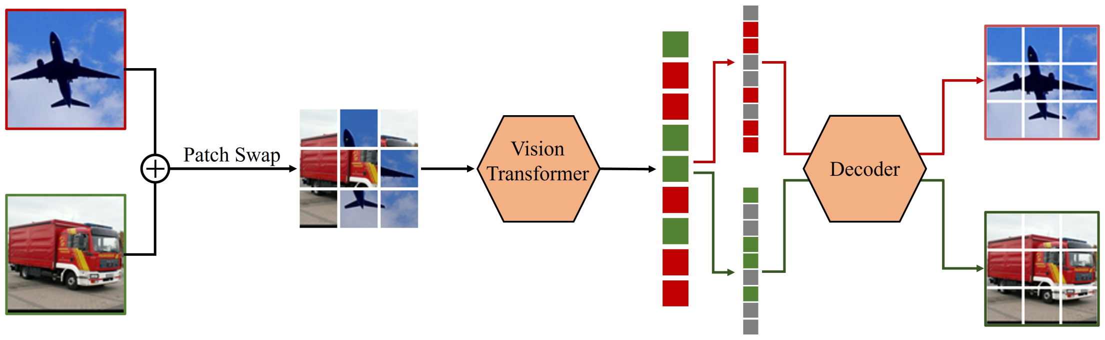


Figure 4.4: Overview of Inverse PatchSwap for Self-supervised learning. The Vision Transformer is trained to encode a PatchSwap Image. The patches originating from the other image are replaced by the Mask token (Gray) and the decoder is trained to reconstruct the original images.

Inverse PatchSwap (abbreviated as IPS), a self-supervised technique designed to extract generalizable features through an unsupervised process. It is based on the idea that reconstructing an image from its partial view trains the network to extract good features [He *et al.* (2022)]. In my methodology, PatchSwap images serve as input to the encoder, with the decoder tasked to reconstruct the original images. PatchSwap images act as regularization for training the encoder as it needs to learn to distinguish patches of different images and learn a joint representation. The features extracted by such an encoder contain information on both the images from the PatchSwap image and provide better generalization.

My approach involves utilizing a PatchSwap image, denoted as x_{ps}, derived from images x_1 and x_2, to be fed into a Vision Transformer T_θ that outputs features/encodings for all the patches. Next, I replace patch features belonging to one of the images (say x_2) with a learnable Mask token. These learnable Mask tokens are used in place of missing patches of x_1 and when passed through a decoder, I use them to reconstruct the missing patches in x_1. I generate two versions of each PatchSwap input: one replaces patches of x_2 with learnable Mask tokens, while the other replaces patches of x_1 with its learnable Mask tokens. The decoder is similar to the encoder but employs a projection layer to convert patch features or encodings into pixel values at the end. I used the mean-squared loss to train the pipeline but the reconstruction loss is applied only to the Mask tokens [He *et al.* (2022)].

In the construction of PatchSwap images, I employ the Beta distribution $\texttt{Beta}(\alpha, \alpha)$ for sampling λ as a regularization strategy. Extreme values of λ (i.e., 0 or 1) result in all input patches belonging to one image, forcing the decoder to generate the second image without any features or encodings. To prevent this scenario, I shift λ away from these extremes via the equation $\lambda := \tau + \lambda * (1 - 2 * \tau)$, setting a buffer τ to 0.2 based on prior experiments [He *et al.* (2022)]. Notably, [He *et al.* (2022)] used a fixed masking ratio of 0.75 but I found that using a random value for λ enhances the efficacy of Inverse PatchSwap. Upon training the encoder, I discard the decoder and attach a classifier to the encoder, leveraging the pre-trained features for classification tasks. Figure 4.4 depicts the schematic representation of Inverse PatchSwap.

4.1.4 PatchSwap++ and Unlabeled PatchSwap++

I have introduced three distinct variations of PatchSwap tailored for specific purposes: PatchSwap for regularizing supervised learning, Unlabeled PatchSwap for semi-supervised learning, and Inverse PatchSwap for self-supervised learning. To enhance supervised training, I combine Inverse PatchSwap with PatchSwap regularization. First, I employ Inverse PatchSwap for pre-training the Vision Transformer. Subsequently, I proceed to train the network in a supervised way, integrating PatchSwap regularization. In a similar way, for semi-supervised learning, I utilize the entire dataset to pre-train the Vision Transformer with Inverse PatchSwap. Then, leveraging this pre-trained model as a foundation, I apply Unlabeled PatchSwap. These innovative methodologies are denoted as PatchSwap++ and Unlabeled PatchSwap++ for supervised and semi-supervised tasks, respectively.

4.2 Experiments

4.2.1 Datasets

I use six datasets for my experiments: CIFAR10 [Krizhevsky *et al.* (2009)], CIFAR100 [Krizhevsky *et al.* (2009)], SVHN [Netzer *et al.* (2011)], FashionMNIST [Xiao *et al.* (2017b)], Tiny-ImageNet and ImageNet-100. These small and medium-sized datasets have a wide range of image types. For CIFAR datasets (CIFAR10 and CIFAR100), I randomly cropped images after zero-padding with $p = 4$ and also flipped horizontally with 0.5 probability. Tiny-ImageNet, a subset of ImageNet has images sized 64×64 pixels belonging to 200 classes. The augmentations applied to these images are the same as CIFAR datasets. In addition, I test RandAugment [Cubuk *et al.* (2020)] augmentation as the base augmentation on these images. For FashionMNIST, a dataset of grayscale images, I resize the images to 32×32 pixels. I use the same augmentation as CIFAR datasets but use zero-padding with $p = 2$. For the SVHN dataset, the images are resized to 32×32 followed by the same augmentations as FashionMNIST except for the horizontal flip. ImageNet-100 is another subset of the ImageNet dataset [Deng *et al.* (2009)] with samples from 100 classes. I used random horizontal flip and random crops of 128×128 image size from 136×136 as augmentations for this dataset.

4.2.2 Training Details

The experiments are performed using ViT-Lite [Hassani *et al.* (2021b)] architecture is a scaled-down version of the original Vision Transformer [Dosovitskiy *et al.*

(2020)]. The architectural details are as follows - 6 encoder blocks, dropout with 0.1 probability, hidden dimension of size 256, 4 attention heads, and the size of forward expansion layer is 512. The total number of parameters is 3.7 million as compared to 86 million of the original base version of the Vision Transformer, making ViT-Lite a light version of the original network. The networks are trained from scratch for 300 epochs (100 for ImageNet-100) with batch-size 128, weight decay 0.03, and Adam optimizer with learning rate 5×10^{-4}. The learning rate is warmed up for the first 10 epochs and then decayed per epoch with a cosine schedule for all the experiments.

4.2.3 Regularization

Setup

The state-of-the-art regularization techniques, namely Label Smoothing [Szegedy *et al.* (2016a)], Cutout [Devries and Taylor (2017)], Mixup [Zhang *et al.* (2018)], and Cutmix [Yun *et al.* (2019)], are used for comparison with PatchSwap and PatchSwap++. As the experiments for my settings are not available for the baselines, I use the official code released by the respective authors to report the results for comparison. For Cutout, I use the cutout size as follows: 8×8 cutout size for CIFAR100, 16×16 for CIFAR10 and FashionMNIST, 20×20 for SVHN, 32×32 for Tiny-ImageNet and 64×64 for Imagenet-100. I apply CutMix augmentation with a probability of 0.5. I always keep $\epsilon = 0.1$ for label smoothing, unless specified otherwise and $\alpha = 1.0$ for all the experiments. All the experiments for PatchSwap, PatchSwap++, and the baselines follow the same training protocol mentioned in 4.2.2.

Compared to the base version of the Vision Transformer [Dosovitskiy *et al.* (2020)], ViT-Lite uses a smaller patch size. I report results with varying patch sizes, specifically 4, 8, and 16 for all the datasets except Tiny-ImageNet and ImageNet-100.

Dataset	Tiny-ImageNet								ImageNet-100	
Augmentation	Standard				RandAugment				Standard	
Patch Size	8		16		8		16		16	
Method	Top1	Top5	Top1	Top5	Top1	Top5	Top1	Top5	Top1	Top5
Cross Entropy	41.9	65.2	34.4	57.7	46.2	70.4	39.1	63.4	54.6	79.9
Label Smoothing	42.8	63.0	34.6	56.5	47.0	69.7	39.3	62.7	54.7	80.3
Cutout	42.8	66.6	33.8	58.1	47.5	71.5	40.2	65.1	54.8	81.0
Mixup	46.6	69.0	38.5	62.4	49.9	73.5	43.2	67.5	60.2	83.4
Cutmix	48.4	71.6	39.5	63.5	48.4	74.9	44.0	68.0	60.3	84.3
PatchSwap	49.9	73.4	41.8	66.3	52.8	77.0	45.6	70.8	61.5	85.3
PatchSwap++	**53.6**	**76.6**	**46.9**	**72.0**	**58.5**	**80.9**	**51.6**	**76.3**	**67.0**	**88.3**

Table 4.1: Top1 and Top5 classification accuracies on TinyImagenet dataset with standard (Random crop with padding and horizontal flip) and RandAugment augmentations with different patch sizes and ImageNet-100 with Standard Augmentation (RandomResizedCrops and horizontal flip) and Patch size 16.

Tiny-ImageNet is tested with patch sizes 8 and 16 and ImageNet-100 is tested with 16 only to avoid the excessive computation overheads. Among the different patch sizes, the best results were obtained with the smallest patch size. As the patches decrease in size, the number of patches increases, which leads to a larger amount of virtual training data. However, it also leads to computational overhead which increases quadratically.

Patch Size	4		8		16	
Method	Top1	Top5	Top1	Top5	Top1	Top5
Cross Entropy	57.9	81.5	50.6	76.2	39.3	65.0
Label Smoothing	58.3	77.0	51.5	71.7	39.8	62.3
Cutout	57.0	81.1	50.2	76.1	39.1	64.7
Mixup	63.5	85.0	56.8	80.0	45.3	70.6
Cutmix	63.7	85.2	57.0	80.4	44.2	69.5
PatchSwap	64.9	86.4	58.5	82.5	45.7	71.6
PatchSwap++	**69.6**	**89.8**	**64.6**	**87.4**	**49.9**	**76.6**

Table 4.2: Top1 and Top5 classification accuracies on CIFAR-100 dataset with different patch sizes.

Results

I list my experiment results on Tiny-ImageNet and ImageNet-100 in Table 4.1. The results for CIFAR100 are in Table 4.2 and Table 4.3 contains results for CIFAR10, SVHN, and FashionMNIST. I reported Top1 classification accuracy for CIFAR10, SVHN, and FashionMNIST, and Top1 and Top5 classification accuracies for CIFAR100, Tiny-ImageNet, and ImageNet-100 datasets. As per the results, PatchSwap and PatchSwap++ outperform all of the compared regularization techniques, irrespective of the patch size. When PatchSwap is used, there is a boost of approximately 1.5% and 2.5% in performance over Cutmix and Mixup, respectively, and about 9% over the network trained with standard Cross-Entropy loss. PatchSwap++ provides an additional boost of about 4% over PatchSwap making it the best regularization among the compared approaches. Except in the case of SVHN, where the performance decreased a bit/stayed the same, I found Inverse PatchSwap

Dataset	CIFAR10			FashionMNIST			SVHN		
Patch Size	4	8	16	4	8	16	4	8	16
Cross Entropy	83.3	78.3	69.8	92.1	92.8	91.2	96.4	94.7	92.7
Label Smoothing	83.0	79.0	69.6	92.0	92.9	91.5	96.5	94.8	92.8
Cutout	84.0	79.2	70.1	94.2	93.5	91.4	96.8	96.2	94.5
Mixup	87.4	82.3	74.3	93.0	93.4	92.2	97.0	95.7	94.2
Cutmix	88.0	82.7	73.8	94.0	93.8	92.5	96.9	96.2	**94.8**
PatchSwap	88.3	84.7	74.9	94.4	93.9	92.6	**97.2**	96.8	**94.8**
PatchSwap++	**90.2**	**87.7**	**76.9**	**95.0**	**94.5**	**93.3**	97.1	**97.0**	94.6

Table 4.3: Top1 classification accuracies on CIFAR-10, FashionMNIST, and SVHN datasets with different patch sizes.

pre-training to be responsible. For the digits, I conclude that occlusions can have a significant impact and can result in reconstructing a different digit which might lead to the mixing of inter-class features. I display two such examples in Figure 4.6 (d) last two rows where digit 9 becomes a 5 and a 0 becomes a 6. For Tiny-ImageNet, PatchSwap and PatchSwap++ augmentation outperform RandAugment augmentation (with standard Cross-entropy). The results also show that RandAugment and PatchSwap are compatible, and using them together further boosts the performance.

4.2.4 Semi-supervised Learning

Setup

For the semi-supervised learning experiments, I use CIFAR10 [Krizhevsky *et al.* (2009)] and SVHN [Netzer *et al.* (2011)] datasets. For training, I use 4000 labeled samples, and the rest of the dataset is treated as unlabeled data. For pseudo-label

Dataset	CIFAR10			SVHN		
Patch Size	4	8	16	4	8	16
Cross Entropy	56.0	53.5	45.4	87.9	86.7	76.0
Pseudo Label	58.1	54.0	46.3	91.2	88.9	78.0
MeanTeacher	62.6	56.5	48.2	96.2	95.1	90.1
PatchSwap (Labeled)	63.2	60.6	51.3	89.7	87.2	81.0
PatchSwap++ (Labeled)	69.3	64.1	53.5	90.2	86.8	78.3
Unlabeled PatchSwap++	73.5	68.3	56.9	**96.3**	**95.9**	**90.7**
Unlabeled PatchSwap++ (F4*)	**75.4**	**69.6**	**57.8**	94.1	91.4	85.6

Table 4.4: Classification accuracies on CIFAR-10 and SVHN datasets in a semi-supervised setting. F4 denotes first four layers (Embedding layer and 3 encoder blocks) of the Vision Transformer are frozen.

training, a threshold of 0.9 probability is used [Lee (2013)]. For MeanTeacher, the Teacher network with an exponential moving average of the Student network is used for producing output labels for the unlabeled data [Tarvainen and Valpola (2017)]. Similar to [Tarvainen and Valpola (2017)], I use the exponential moving average for my experiments as well. My approach eliminates the need for numerous augmented versions of the image and instead, consistency regularization is applied to two PatchSwap versions of the images. For the first PatchSwap image, the Teacher network generates the targets and the Student network is then trained to match these generated outputs using the second PatchSwap image as input. For all my experiments, I used $\gamma = 100$. For the first 10 epochs I linearly increase the unlabeled loss. All the other experimental settings remain the same as the regularization experiments.

Results

The results for the semi-supervised experiments are displayed in Table 4.4. I include the results when only labeled data is used for training with PatchSwap regularization - PatchSwap (Labeled only) and PatchSwap++ (Labeled only). PatchSwap++ (Labeled only) performs Inverse PatchSwap pretraining on labeled samples only. Unlabeled PatchSwap is when PatchSwap and Unlabeled PatchSwap are used together. Unlabeled PatchSwap++ uses Inverse PatchSwap to pre-train the network with all the data and then uses Unlabeled PatchSwap. It is worth noting that my approach outperforms all the compared approaches. MeanTeacher is outperformed by PatchSwap with a labeled loss on CIFAR10 only. The Unlabeled PatchSwap provides an additional performance boost. Similar to regularization experiments (and the explanation), I found Inverse PatchSwap to be helpful for CIFAR10 but not for SVHN. I verify this by freezing the first four layers of the encoder (i.e. embedding layer and 3 encoder blocks) and only training the rest of the network. The results are indicated as Unlabeled PatchSwap++ (F4). As expected, freezing the layers provides an additional boost of 1% to CIFAR10 and reduces SVHN performance.

4.2.5 Self-supervised Learning

Setup

I test pretraining of Inverse PatchSwap on CIFAR10, CIFAR100, FashionMNIST, and Tiny-ImageNet, ImageNet-100 with patch sizes 8, 4, 16, 16 and 16, respectively. The decoder network for this task has 4 decoder blocks that resemble encoder blocks in architecture, followed by a projection layer that projects the features into the dimension of a patch. The patch is then reshaped and combined with the other patches to form the image. The loss is applied only to the missing patches. The

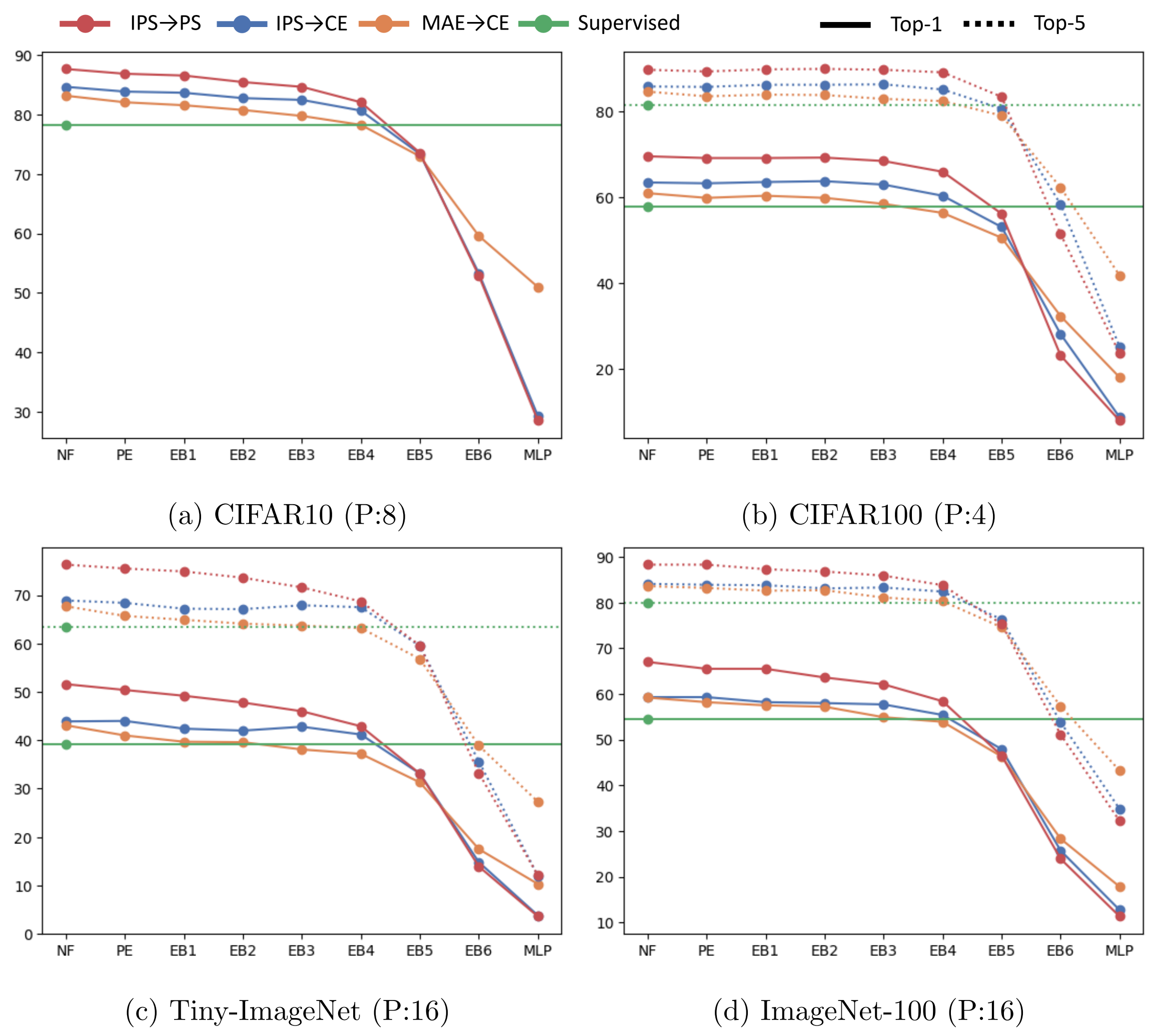
Dataset		FMNIST	CIFAR10	CIFAR100		TI-RA		IM100	
PatchSize		16	8	4		16		16	
Init	Method	Top1	Top1	Top1	Top5	Top1	Top5	Top1	Top5
Random	CE	91.2	78.3	57.9	81.5	39.1	63.4	54.6	79.9
MAE	CE	92.2	83.2	61.0	84.7	43.1	67.7	59.1	83.6
IPS	CE	92.5	84.7	63.5	85.9	43.9	68.9	59.4	84.1
Random	PS	92.6	84.7	64.9	86.4	45.6	70.8	61.5	85.3
MAE	PS	93.1	86.6	68.4	89.6	50.8	75.9	66.0	88.0
IPS	PS	**93.3**	**87.7**	**69.6**	**89.8**	**51.6**	**76.3**	**67.0**	**88.3**

Table 4.5: Comparison of different initialization (Init) and training methods (method) for FashionMNIST (FMNIST), CIFAR10, CIFAR100, TinyImageNet with RandAugment (TI-RA), and ImageNet-100 (IM100) datasets.

buffer τ is set to 0.2 for all the Inverse PatchSwap experiments. I used Masked AutoEncoder (MAE) [He *et al.* (2022)] as the baseline for my approach. The same setup is used for the Masked AutoEncoder baseline except the masking ratio is set to 0.75 as detailed in [He *et al.* (2022)]. I experimented with using Random, MAE, and Inverse PatchSwap as the initialization and considered Cross-entropy and PatchSwap for training the network on supervised tasks. I also test the quality of features learned by the layers by freezing different parts of the network and training the rest.

Results

The results for self-supervised experiments for different initialization and training methods are in Table 4.5. 'RA' denotes RandAugment was part of the base augmentation while training. It can be observed that Inverse PatchSwap provides the best

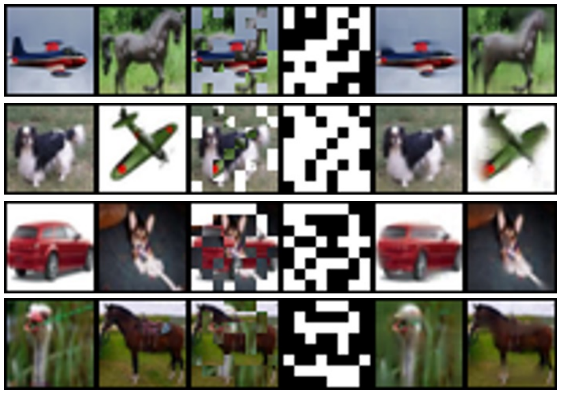


(a) CIFAR10 (P:8) (b) CIFAR100 (P:4)

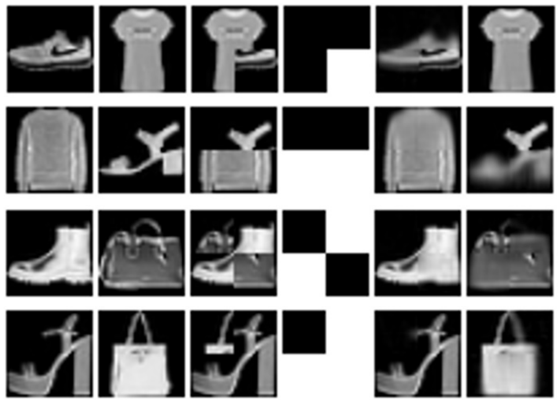
(c) Tiny-ImageNet (P:16) (d) ImageNet-100 (P:16)

Figure 4.5: Test Classification accuracies for **Supervised**, **MAE → CE** , **Inverse PatchSwap → CE** and **Inverse PatchSwap → PatchSwap**. 'P:x' denotes the 'x' patch size that was used. Top1 and Top5 accuracies are displayed by solid and dashed lines. The X-axis shows different layers of the transformer. While training, all the layers before those layers are frozen. NF: Layers are not frozen; PE: Embedding block; EB-i are the encoder blocks. MLP: The whole network is frozen except for the last classification layer.

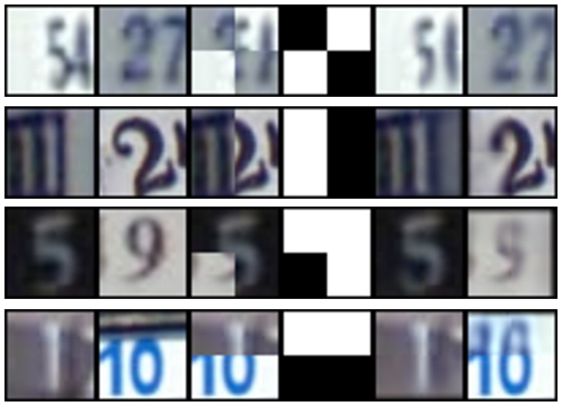
initialization regardless of the training method, and PatchSwap provides the best test performance for any initialization. The best performance is achieved when using



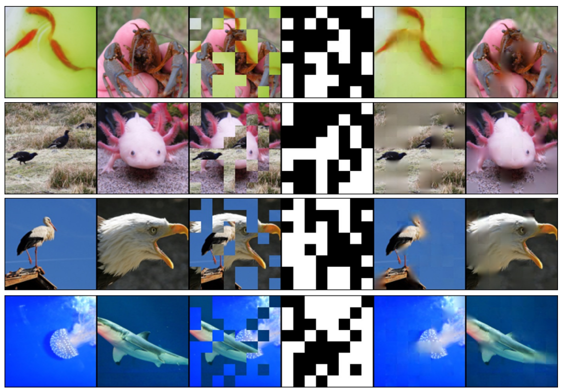
(a) CIFAR10 (P:4)



(b) FMNIST (P:16)



(c) SVHN (P:16)



(d) ImageNet-100 (P:16)

Figure 4.6: Reconstruction of Inverse PatchSwap on test sets of different datasets. The First and Second columns are the input images. The third column is the generated PatchSwap image and the fourth is the binary mask used for creating PatchSwap Image. The fifth and the sixth columns are the images produced by the decoder corresponding to the first and the second images. I overlay the output with the PatchSwap patches to improve the visual quality.

Inverse PatchSwap in conjunction with PatchSwap a.k.a. PatchSwap++.

I also test the quality of features for each layer by freezing the network before that layer and the results are displayed in Figure 4.5. In Figure 4.5, P:x signifies the 'x' patch size that was used for training the Vision Transformer. Solid lines denote

Top1 accuracy and dashed lines denote Top5 accuracy. The X-axis shows the different layers of the transformer and signifies which layers are trained/frozen. NF represents no layers are frozen. PE represents only the Patch Embedding block is frozen. EB-i represents the encoder blocks. While training, all the layers before those layers are frozen (including the Patch Embedding block). MLP denotes The whole network is frozen except for the last classification layer.

Inverse PatchSwap outperforms MAE when zero to five layers are frozen. However, as the number increases, MAE starts to outperform Inverse PatchSwap (IPS). MAE mentions that the last few layers of the encoder start to become generation task-specific. Hence, using a fully frozen network is not recommended. I observe an amplification of the same effect here and hence, IPS requires a minimum of 3 layers to be fine-tuned. I also observe that PatchSwap begins to underperform Cross-entropy loss in these cases. In Figure 4.6, I show the reconstruction of a few test samples from different datasets using Inverse PatchSwap. P:x denotes the 'x' patch size that was used for training the Vision Transformer in figure 4.6.

4.3 Analysis

4.3.1 Regularization Intensity

The regularization intensity in PatchSwap is controlled by the hyperparameter α, with $\lambda = \beta(\alpha, \alpha)$. When α takes small values ($\alpha << 1$), it leads to λ values close to 0 or 1 in Beta distribution. As a result of rounding, the swapping coefficient λ' then will end up being 0 or 1. In this setting, there is no PatchSwap regularization, and it is equivalent to using standard cross-entropy loss for training. Similarly, a swapping coefficient of $\lambda' = 0.5$ will occur as a result of large values of $\alpha >> 1$. This is the setting with the highest PatchSwap or maximum swapping. In this manner, α

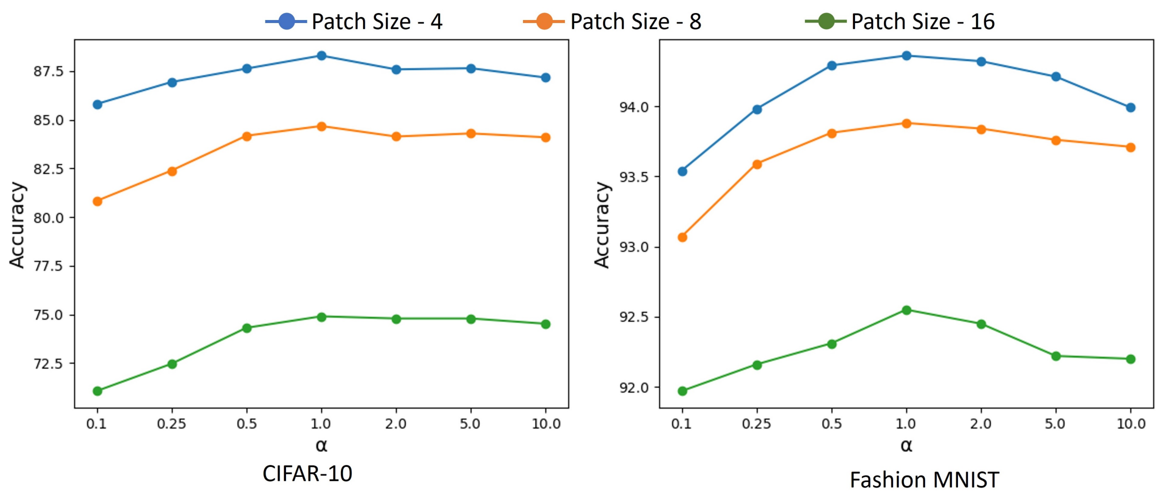


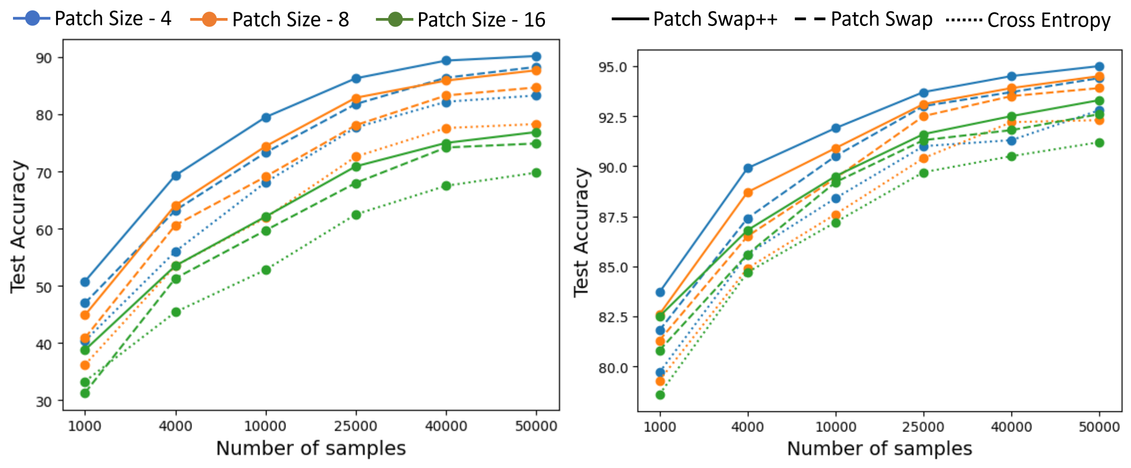
Figure 4.7: Impact of changing α on the CIFAR-10 (left) and FashionMNIST (right) datasets. Different colors denote different patch sizes used for training the Vision Transformer.

controls the balance between cross-entropy and regularization.

Figure 4.7 displays the accuracy-vs-α plots for CIFAR10 and FashionMNIST datasets. A small value of α yields significantly poor performance (similar to using only cross-entropy loss) due to reasons outlined earlier. Similarly, a large value of α also decreases the performance. When $\alpha = 1$ (as used in all my experiments), it results in a uniform distribution for the swapping coefficient with $\lambda' \in [0, 1]$. This setting results in the best performance.

4.3.2 Number of Samples

In this section, I evaluate PatchSwap with a reduced number of training samples to test its regularization capabilities. I use CIFAR10 and the FashionMNIST dataset for these experiments. Figure 4.8 displays my results. I report the test accuracies for a range of available training samples - 1000, 4000, $10,000$, $25,000$, $40,000$, and



(a) CIFAR10 (b) FashionMNIST

Figure 4.8: Comparison of Test accuracy vs Number of Samples used for training using Cross-Entropy, PatchSwap, and PatchSwap++ on CIFAR-10(a) and Fashion-MNIST(b) datasets.

the full set. I compare my results (for various patch sizes as well) with a network trained with standard Cross-Entropy loss (shown using dashed lines), PatchSwap, and PatchSwap++. From the plot, I observe that the performance of PatchSwap with $10,000$ CIFAR10 labeled training samples is the same as that when $25,000$ samples are used for supervised training and PatchSwap++ with $10,000$ outperforms Cross-entropy with $25,000$ samples. Similar trends can be observed for FashionMNIST where PatchSwap and PatchSwap++ require significantly less labeled data for training.

4.3.3 Training Progress

I showcase the training progression of PatchSwap vs Cross-Entropy for the CIFAR100 and SVHN datasets. The results are displayed in Figure 4.9. Cross-entropy achieves better test accuracy at the start of the training but PatchSwap results in

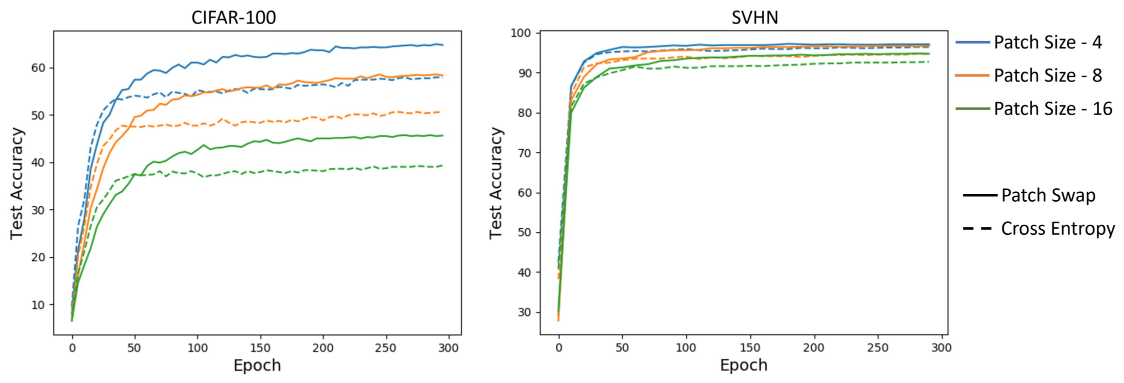


Figure 4.9: Test accuracy graph for training process on CIFAR-100 (left) and SVHN (right). Different patch sizes used for training the transformers are denoted by different colors. Solid lines represent PatchSwap and dashed lines represent Cross-entropy loss.

the best final accuracy for all the scenarios. The training accuracy was around 100 % for all the cases at the end of the training. However, PatchSwap results in much less overfitting.

4.3.4 *Training Losses*

Cross-entropy loss is often augmented with Mixup and Cutmix to help train networks that generalize better. Vision Transformers are trained using the same procedure too [He *et al.* (2022)]. In this section, I present results to show the impact of using PatchSwap in addition to Mixup-Cutmix. During every training step, one of the three techniques is chosen randomly. I use CIFAR100 and Tiny-ImageNet for these experiments and the results are summarized in Table 4.6. I also test the case of initializing the network with Inverse PatchSwap (last row) and training with all the losses. From the results in the Table, it is clear that standard cross-entropy suffers from high overfitting which is alleviated to an extent by PatchSwap. Even though

Dataset	CIFAR100			Tiny-ImageNet	
Patch Size	4	8	16	8	16
Cross Entropy	57.9	50.6	39.3	41.9	34.4
PatchSwap	64.9	58.5	45.7	49.9	41.8
MX+CX	66.4	60.3	47.7	51.2	43.8
MX+CX+PS	67.2	60.9	48.3	52.3	44.6
IPS→MX+CX+PS	**71.6**	**65.4**	**51.1**	**57.2**	**49.1**

Table 4.6: Top1 classification accuracies on CIFAR-100 and Tiny-ImageNet using different combinations of losses for training. MX: Mixup, CX: CutMix, PS: PatchSwap

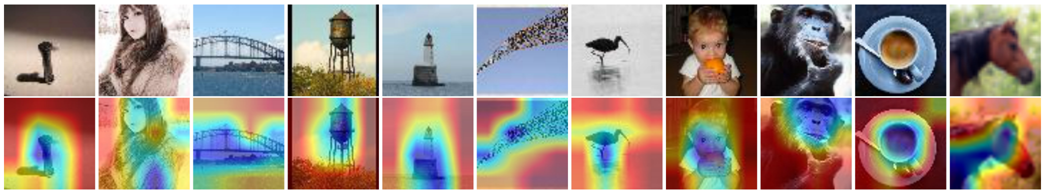


Figure 4.10: Attention Maps of PatchSwap++ on the validation set of Tiny-ImageNet.

Mixup+Cutmix performs better than PatchSwap alone, using all three of them together provides the best regularization. However, the best performance is achieved by using Inverse PatchSwap as initialization and all three losses for training.

4.3.5 Attention Maps

In Figure 4.10, I visualize the attention maps of PatchSwap++ on original (non-PatchSwap images) using Grad-CAM [Selvaraju *et al.* (2017)] on the validation set of Tiny-ImageNet with a patch size of 8. Additionally, I also show the class-specific attention maps of the model on PatchSwap images in Figure 4.11. It can observed that the network learns to pay attention to the appropriate patches for the classification

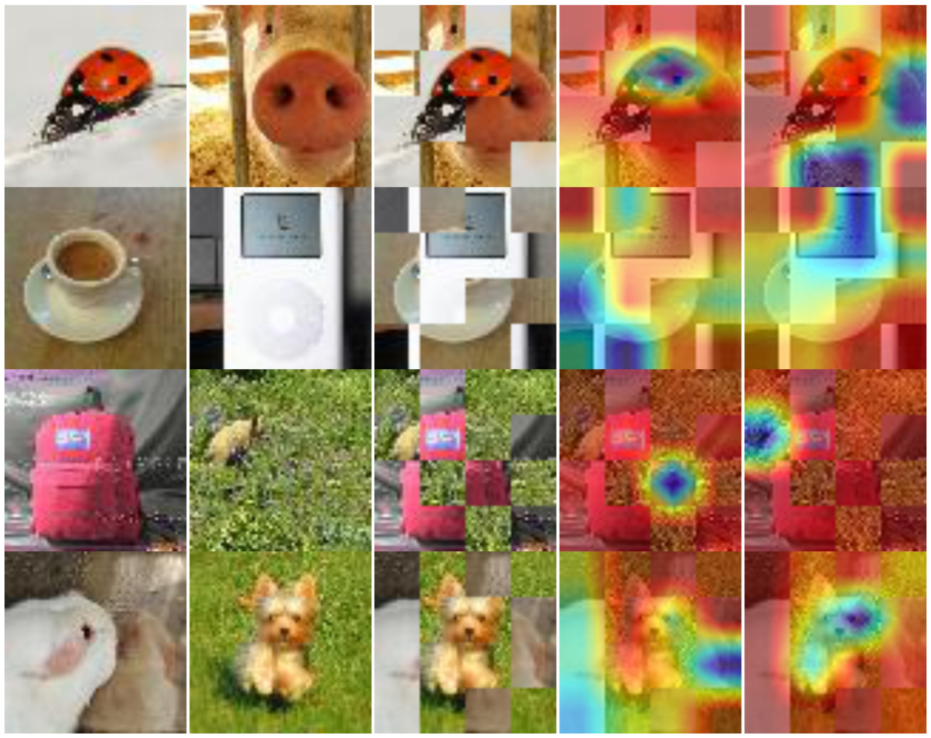


Figure 4.11: Class-specific Attention Maps of PatchSwap++ on PatchSwap images. The first and the second columns are the input images. The third column is the generated PatchSwap image. The fourth and fifth columns are class-specific attention maps for the first and the second objects. Example: In the first row, the objects are beetle and pig, and the third column shows the PatchSwap Image. I generate class-specific attention maps for Bettle on the PatchSwap image in Column 4 and it can be observed that the network focuses only on the patches from the Bettle image and ignores patches from the Pig image. Similarly, in the 5th column, the attention for Pig focuses on Pig patches and ignores others.

of the images. For example in the first row, the model is attending to the patch belonging to the 2nd row-3rd column patch for the ladybug class. Similarly, for the pig class, the model attends to the 3rd row-4th column and 4th row-3rd column patches which contain pig specific-patches. Similarly, I show a similar visualization for a model

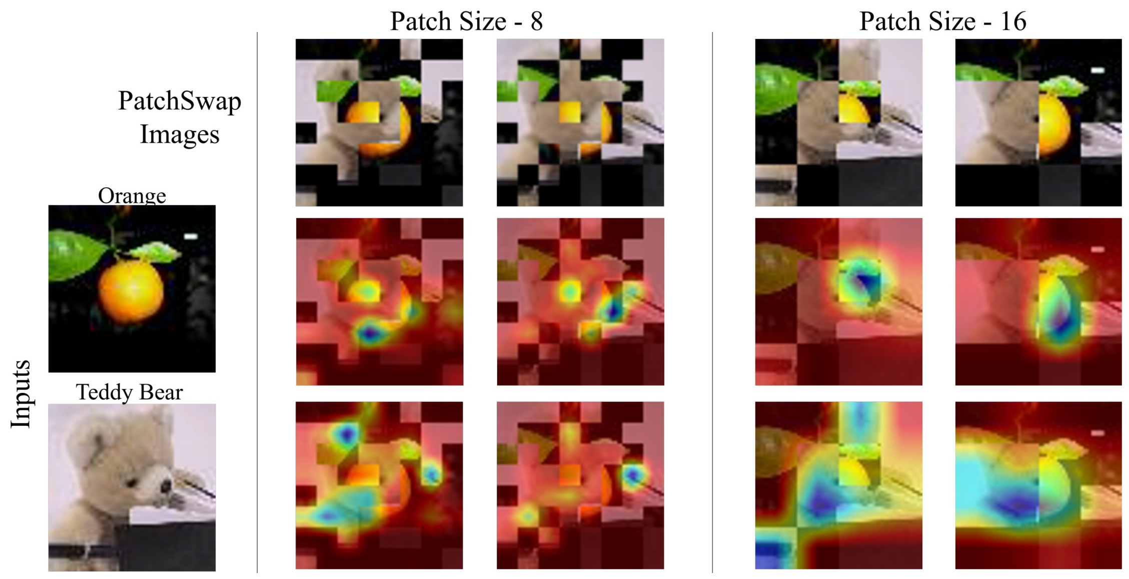


Figure 4.12: Class-specific Attention Maps for PatchSwap images. The first column shows the input images. I generate PatchSwap images for two different patch sizes - 8 (2^{nd} and 3^{rd} column) and 16 (4^{th} and 5^{th} column). The 2^{nd} row displays the attention map for Orange and the last row shows for Teddy Bear.

trained using vanilla PatchSwap with different patch sizes in Figure 4.12.

4.3.6 Transfer vs Self-supervised Learning

This experiment compares the impact of Self-supervised learning vs. Transfer learning. I use CIFAR10→CIFAR100 and CIFAR100→CIFAR10 with patch size 4 for this experiment. I consider different initializations for the model: (i) Random, (ii) Model trained on source with Cross-entropy, (iii) Model trained on source with PatchSwap, (iv) Model trained on source with Inverse PatchSwap, and (v) Model trained on target (self) with Inverse PatchSwap. The model is then trained on the target task with cross-entropy loss. The results are displayed in Table 4.7. I found both transfer and self-supervised learning to be helpful. Also, an initialization from

Init	CIFAR10 $\rightarrow$ CIFAR100	CIFAR100 $\rightarrow$ CIFAR10
Random	57.9	83.3
CE (Source)	61.0	85.1
PS (Source)	64.0	88.0
IPS (Source)	**64.2**	**88.2**
IPS (Target)	63.5	88.1

Table 4.7: Comparison of test accuracy for model initialized with Transfer learning vs. Self-supervised learning. I test this on CIFAR10 and CIFAR100 with patch size 4. CE: Cross-Entropy; PS: PatchSwap; IPS: Inverse PatchSwap

a regularized model (regardless of source/target) leads to better target test accuracy. The best performance was achieved by transfer learning for the model trained on source with self-supervision.

4.3.7 PatchSwap for ConvNets

Although I have designed PatchSwap for Vision Transformers, this regularization is compatible with ConvNets as well. One difference is that the patch size becomes a hyperparameter when PatchSwap is applied to ConvNets. I experiment with different patch sizes on the input - $\{1, 2, 4, 8, 16\}$. I show how different loss functions affect the training process using the ResNet-18 network trained on CIFAR10 and CIFAR100 datasets. I follow the same settings for running the baselines. The images are resized to be of size 32×32. A batch size of 128 is used to train the network for 300 epochs using an SGD optimizer with a learning rate and momentum of 0.1 and 0.9, respectively. The learning rate is warmed up for the first 10 epochs after which it is decayed by a factor of 0.1 at 150^{th} and 225^{th} epoch.

Method	CIFAR10	CIFAR100
Cross Entropy	95.1	76.0
Label smoothing	95.1	76.4
Cutout	95.9	76.8
Mixup	96.0	77.1
Cutmix	**96.2**	**78.2**
PatchSwap-1	94.1	73.4
PatchSwap-2	95.3	75.8
PatchSwap-4	95.9	77.6
PatchSwap-8	**96.2**	78.0
PatchSwap-16	96.0	77.8

Table 4.8: Top1 classification accuracies on CIFAR-10 and CIFAR-100 for ResNet-18 (ConvNet). PatchSwap-k denotes PatchSwap was performed using k patch size.

I summarize the results for these experiments in Table 4.8, where PatchSwap-k denotes the PatchSwap using patch size k. ResNet-18 outperforms my Vision Transformers due to its translation equivariance and locality inductive biases [Dosovitskiy *et al.* (2020)]. Vision Transformers demand abundant data to acquire these properties [Dosovitskiy *et al.* (2020)]. While ResNet-18 has 11 million in parameters, the number of parameters for my Vision Transformer is almost a third of it. From the results, I observe that PatchSwap produces good performance when applied to ConvNets too. However, the patch size hyperparameter, k needs to be tuned in this case. For CIFAR10 and CIFAR100 datasets, PatchSwap yields the best performance when $k = 8$.

Chapter 5

LABEL REGULARIZATION

The traditional method of training neural networks involves the utilization of one-hot targets and cross-entropy loss, a long-standing practice in the field. However, the use of one-hot targets has been recognized for its tendency to instigate overconfidence within the network, potentially hampering its generalization capabilities [Szegedy *et al.* (2016a)]. Over the years, various regularization techniques, such as Cutout [Devries and Taylor (2017)], Mixup [Zhang *et al.* (2018)], CutMix [Yun *et al.* (2019)], and others [Hendrycks *et al.* (2020); Gong *et al.* (2021)], have been introduced to address this issue, often involving modifications to the input data. However, these augmentations need to be done cautiously as they should not impact the main object in the input. An alternative strategy is Label Smoothing, which adjusts target labels during training by adding a uniform label distribution over the categories to the one-hot target [Szegedy *et al.* (2016a)]. Training with Label Smoothing has proven effective in enhancing generalization and has been widely adopted.

Despite the advantages of Label Smoothing, it is known to disrupt the relationships between categories [Müller *et al.* (2019)]. This problem arises from the use of a uniform probability vector in generating smoothed targets, assigning equal importance to all negative categories. Consequently, the network is instructed to treat all categories as equally distinct from each other, leading to compact and equidistant category clusters in the feature space [Müller *et al.* (2019)]. This outcome is undesirable; e.g., targets for the *Dog* class should have a relatively higher similarity with the *Cat* class, as compared to the *Truck* class. Enforcing uniform inter-category relationships limits the model's performance [Zhang *et al.* (2021). The inter-category relationship

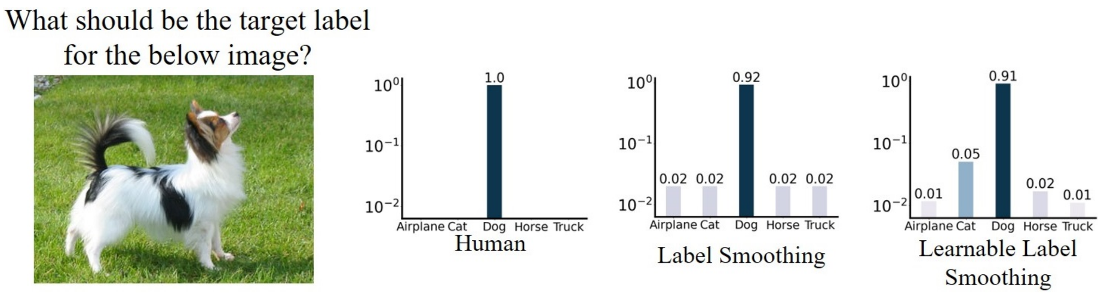


Figure 5.1: Illustration of targets generated via Human, Label Smoothing, and Learnable Smoothing.

is crucial for applications such as Knowledge Distillation, dealing with missing data, and learning from noisy labels [Hinton *et al.* (2015); Müller *et al.* (2019); Zhang *et al.* (2021)]. This prompts two fundamental questions: (1) Is it possible to regulate confidence while preserving the inter-category relationship? and, (2) What alternative should be employed in place of the uniform probability vector?

This dissertation introduces my novel solution, termed ***L**earnable **L**abel **S**moothing* (LLS), to address these questions. My approach aims to train the network to learn the optimal target vector, as illustrated in Figure 5.1. I present a category-wise learnable probability vector. By merging this probability vector with the one-hot label, similar to Label Smoothing, I create targets unique to each category. For a dataset with K categories, these category-wise learnable probability vectors together form the $K \times K$ Q-matrix, whose rows Q_k encode the inter-category similarities.

I demonstrate empirically that Learnable Label Smoothing outperforms Label Smoothing. Furthermore, networks trained with Learnable Label Smoothing prove to be more effective Teacher models in Knowledge Distillation scenarios. The learned Q-Matrices exhibit little variation notwithstanding randomized initialization and varying network architecture, enabling seamless knowledge transfer and distillation even in the absence of a Teacher network. While the Q-Matrix primarily depends on the training data, it is still useful for regularizing subsets (category-wise and sample-wise) of the data which reduces the necessity for frequent relearning. These characteristics enhance the matrix's versatility and widen the scope of its potential applications.

5.1 Learnable Label Smoothing

5.1.1 Preliminaries

Let D be a dataset with image label pairs $\{x, y\}$ where x represents an image, and $y \in \{1, \dots, K\}$ is the ground truth label. The ground truth labels are also represented as 1-hot vectors $p = [p_1, \dots, p_K]^\top$, where $p_i \in \{0, 1\}$. Correspondingly, $p_i = 1$ when index $i = y$, else it is 0. The neural network with parameters θ is represented as $f_\theta(.)$. For a sample x, the output probability vector is denoted by $\hat{p} = f_\theta(x)$. The standard cross-entropy objective $H(p, \hat{p})$ is minimized for network training, and is computed as,

$$H(p, \hat{p}) = -p \log \hat{p} = -\sum_{i=1}^{K} p_i \log \hat{p}_i = -\log \hat{p}_y. \tag{5.1}$$

However, the conventional training approach utilizing a 1-hot vector is known to induce overconfidence [Szegedy *et al.* (2016a)] and lead to poor calibration and overfitting [Mukhoti *et al.* (2020); Lin *et al.* (2017)]. To address this issue, Label Smoothing introduces a regularization technique by creating a modified target p^{ls} [Szegedy *et al.* (2016a)]. This is achieved by combing the 1-hot probability p with a uniform probability vector $u = [\frac{1}{K}, \dots, \frac{1}{K}]^\top$, resulting in,

$$p^{ls} = (1 - \alpha)p + \alpha u. \tag{5.2}$$

Here, α is the smoothing hyper-parameter, typically set to 0.1. The network trained using the cross-entropy with the modified targets (p^{ls}), mitigates the problem of overconfidence as,

$$\begin{aligned} H(p^{ls}, \hat{p}) &= -p^{ls} \log \hat{p} \\ &= (1-\alpha)H(p, \hat{p}) + \alpha H(u, \hat{p}) \\ &= (1-\alpha)H(p, \hat{p}) + \alpha KL(u||\hat{p}) + \alpha H(u). \end{aligned} \tag{5.3}$$

Here, the first term is the cross-entropy between $H(p, \hat{p})$ scaled by $(1-\alpha)$. The second term is the Kullback-Leibler Divergence between u and $\hat{p}$ driving the predictions to become more uniform and reducing the confidence of predictions. The last term is the entropy over u, where $H(u) = -\sum_i u_i \log u_i$, which is a constant.

5.1.2 Method

My approach replaces the uniform vector u in Label Smoothing with a learnable probability vector, granting the network the ability to select optimal targets. My learned target vector is of the form, $p^{lls} = (1-\alpha) * p + \alpha * q$ where, q is learned through network training. I argue that a 1-hot target vector is an overconfident and hard assignment of the image category. Label Smoothing ameliorates the effect of overconfidence by assuming a uniform prior label distribution. However, Label Smoothing could introduce unwanted biases when uniformly smoothing the probabilities [Lienen and Hüllermeier (2021)]. I aim to learn the distribution q and estimate the 'moving' target label p^{lls} even as the network trains to align the prediction $\hat{p}$ with p^{lls}. I share a probability vector q between all samples within a category, due to their shared relationships with other categories, and employ distinct q for each category. Hence, I learn a matrix Q of dimensions $K \times K$, where row i signifies a learnable probability vector $Q_i = [q_{i1}, q_{i2}, \ldots, q_{iK}]$ for category i. For a training sample (x, y), the modified label is given as p^{lls} where,

$$p^{lls} = (1-\alpha) * p + \alpha * Q_y, \tag{5.4}$$

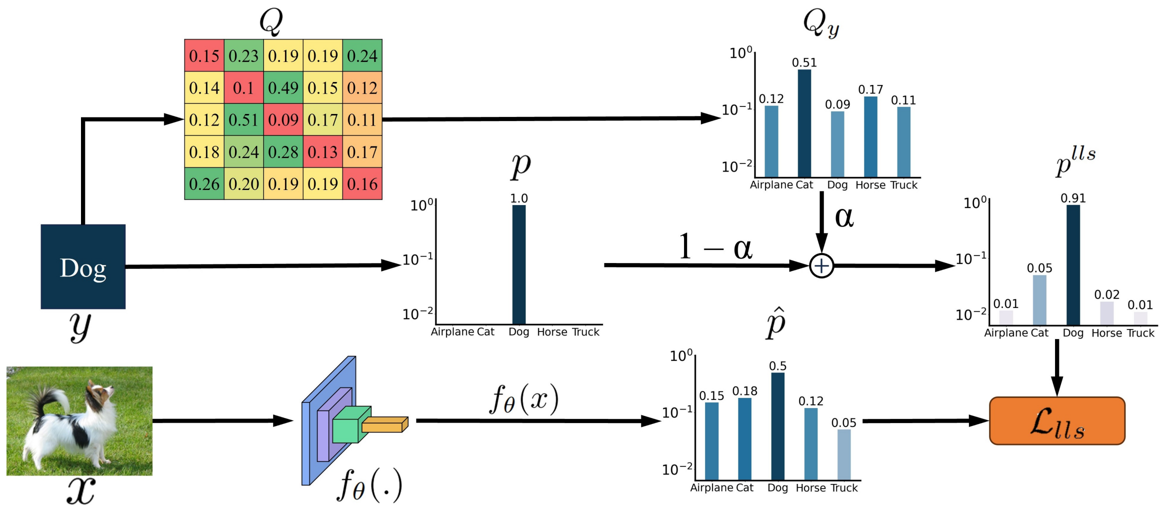


Figure 5.2: An overview of Learnable Label Smoothing. My approach utilizes a matrix Q with dimensions $K \times K$ that serves as the repository for learnable probability vectors for each category. A given 1-hot vector p of a category is mixed with its associated probability vector Q_y from matrix Q, governed by the hyperparameter α. This operation results in the target p^{lls} which is used for training with $\mathcal{L}_{lls}$ loss.

with p as the 1-hot vector corresponding to ground truth label y, and Q_y the y-th row of the learned Q matrix. α is the hyper-parameter similar to Label Smoothing. The purpose of the Q-Matrix is to facilitate the acquisition of the optimal mixing probability vectors during Label Smoothing. I refer to my framework as ***L**earnable **L**abel **S**moothing* (LLS).

5.1.3 Training Using Learnable Label Smoothing

Given the Learnable Label Smoothing (LLS) target probability vector p^{lls} and the network prediction $\hat{p}$, the standard training objective is the minimization of the cross-entropy loss $H(p^{lls}, \hat{p}) = -\sum_i p_i^{lls} \log \hat{p}_i$. The cross-entropy is an upper-bound on the KL-divergence between p^{lls} and $\hat{p}$, where $H(p^{lls}, \hat{p}) = KL(p^{lls}||\hat{p}) + H(p^{lls})$.

The second term $H(p^{lls})$ is the entropy of p^{lls} which is 0 when p^{lls} is 1-hot. When p^{lls} is not 1-hot, minimizing cross-entropy $H(p^{lls}, \hat{p})$ also minimizes the entropy of $H(p^{lls})$, making p^{lls} more 1-hot. This does not serve my purpose where I aim to retain the inter-category relationships in the target label. I aim to instead directly minimize the KL-divergence objective $KL(p^{lls}||\hat{p})$.

I term $KL(p^{lls}||\hat{p})$ as the Forward-KL. In standard Forward-KL divergence objectives, e.g., $KL(r||s)$, the distribution r is fixed and s is optimized to align with r. With $KL(p^{lls}||\hat{p})$ there is the challenge of a moving target where p^{lls} is being learned as $\hat{p}$ aligns with it. The forward $KL(p^{lls}||\hat{p})$ has smaller magnitude gradients w.r.t p^{lls} compared to the gradients w.r.t $\hat{p}$. This makes $\hat{p}$ align quickly with p^{lls} before the inter-category relationships are learned in p^{lls}. In other words, the network overfits before Learnable Label Smoothing can take effect. This is mitigated when I also have a Reverse-KL term $KL(\hat{p}||p^{lls})$ which ensures the target p^{lls} is updated quickly. Below I show the gradients of the two loss functions to showcase this property.

To facilitate the derivations, I employ specific notations: let $q = Q_y = [q_1, q_2, \ldots, q_K]$, where q_i represents the probability of the i-th category for the y-th row of the Q-Matrix. The entries in the Q-Matrix are generated from logits. For e.g., $[t_1, t_2, \ldots, t_K]$ are the logits that generate the y-th row in Q. Here, $q = softmax([t_1, t_2, \ldots, t_K])$, indicating that q is obtained by applying the Softmax activation function to the logits values. Likewise, I use $z = [z_1, z_2, \ldots, z_K]$ to represent the logits from the network f_θ which are then converted to predicted probabilities $\hat{p}$. Here, $\hat{p} = softmax([z_1, z_2, \ldots, z_K])$.

Firstly, I derive the gradient of the softmax probability $q_i = \frac{e^{t_i}}{\sum_k e^{t_k}}$ with respect to logits t_j, as this derivation will be utilized in subsequent derivative calculations,

$$\begin{aligned}
q_i &= \frac{e^{t_i}}{\sum_k e^{t_k}} \\
\frac{\partial q_i}{\partial t_j} &= \frac{e^{t_i} \cdot I\{i=j\}}{\sum_k e^{t_k}} \cdot \frac{\sum_k e^{t_k}}{\sum_k e^{t_k}} - \frac{e^{t_i}}{\sum_k e^{t_k}} \cdot \frac{e^{t_j}}{\sum_k e^{t_k}} \\
&= q_j I\{i=j\} - q_i q_j \\
&= q_j(I\{i=j\} - q_i) \qquad (5.5)
\end{aligned}$$

Similarly, I have the derivative,

$$\frac{\partial \hat{p}_i}{\partial z_j} = \hat{p}_j(I\{i=j\} - \hat{p}_i) \qquad (5.6)$$

I show the derivative of the forward KL loss $\mathcal{L}_{fkl}$ w.r.t. network logits z_j:

$$\begin{aligned}
\mathcal{L}_{fkl} &= KL(p^{lls}||\hat{p}) = H(p^{lls}, \hat{p}) - H(p^{lls}) \\
&= -\sum_i p_i^{lls} \log \hat{p}_i + \sum_i p_i^{lls} \log p_i^{lls} \qquad (5.7) \\
\frac{\partial \mathcal{L}_{fkl}}{\partial z_j} &= -\sum_i p_i^{lls} \frac{\partial \log \hat{p}_i}{\partial z_j} + 0 \\
&= -\sum_i \frac{p_i^{lls}}{\hat{p}_i} \frac{\partial \hat{p}_i}{\partial z_j} \quad \text{using } eq \text{ 5.6,} \\
&= -\sum_i \frac{p_i^{lls}}{\hat{p}_i} \hat{p}_j (I\{i=j\} - \hat{p}_i) \\
&= -\frac{p_j^{lls}}{\hat{p}_j} \hat{p}_j + \sum_i \frac{p_i^{lls}}{\hat{p}_i} \hat{p}_j \cdot \hat{p}_i \\
&= -p_j^{lls} + \hat{p}_j \sum_i p_i^{lls} \cdot \\
&= \hat{p}_j - p_j^{lls} \qquad (5.8)
\end{aligned}$$

Next, I derive the gradient of forward KL loss $\mathcal{L}_{fkl}$ w.r.t. Q-Matrix logits t_j:

$$\begin{aligned}
\mathcal{L}_{fkl} &= KL(p^{lls}||\hat{p}) = H(p^{lls}, \hat{p}) - H(p^{lls}) \\
&= -\sum_i p_i^{lls} \log \hat{p}_i + \sum_i p_i^{lls} \log p_i^{lls} \\
let \quad \mathcal{L}_{fce} &= -\sum_i p_i^{lls} \log \hat{p}_i \quad and \\
\mathcal{L}_{emt} &= \sum_i p_i^{lls} \log p_i^{lls} \qquad (5.9)
\end{aligned}$$

Next, I will solve derivatives of $\mathcal{L}_{fce}$ and $\mathcal{L}_{emt}$ separately and then combine them later to get the derivative of $\mathcal{L}_{fkl}$.

Derivative of Forward Cross-Entropy loss $\mathcal{L}_{fce}$ w.r.t. Q-Matrix logits t_j:

$$\begin{aligned}
\mathcal{L}_{fce} &= -\sum_i [(1-\alpha)p_i + \alpha q_i] \log \hat{p}_i \qquad (5.10) \\
\frac{\partial \mathcal{L}_{fce}}{\partial t_j} &= -\sum_i \alpha \frac{\partial q_i}{\partial t_j} \log \hat{p}_i \\
&= -\alpha \sum_i \frac{\partial q_i}{\partial t_j} \log \hat{p}_i \quad \text{using } eq \text{ 5.5}, \\
&= -\alpha \sum_i [q_j(I\{i=j\} - q_i)] \cdot \log \hat{p}_i \\
&= -\alpha q_j \sum_i [(I\{i=j\} - q_i)] \cdot \log \hat{p}_i \\
&= -\alpha q_j (\log \hat{p}_j - \sum_i q_i \cdot \log \hat{p}_i) \\
&= \alpha q_j \left(\sum_i q_i \cdot \log \hat{p}_i - \log \hat{p}_j \right) \qquad (5.11)
\end{aligned}$$

Derivative of Entropy Maximization loss on targets $\mathcal{L}_{emt}$ w.r.t. Q-Matrix logits t_j:

$$
\begin{aligned}
\mathcal{L}_{emt} &= \sum_i p_i^{lls} \log p_i^{lls} \\
&= \sum_i \left[(1-\alpha)p_i + \alpha q_i\right] \log[(1-\alpha)p_i + \alpha q_i] \qquad (5.12) \\
\frac{\partial \mathcal{L}_{emt}}{\partial t_j} &= \sum_i \alpha \frac{\partial q_i}{\partial t_j} \log[(1-\alpha)p_i + \alpha q_i] \\
&+ \sum_i [(1-\alpha)p_i + \alpha q_i] \cdot \frac{1}{[(1-\alpha)p_i + \alpha q_i]} \cdot \alpha \frac{\partial q_i}{\partial t_j} \\
&= \sum_i \alpha \frac{\partial q_i}{\partial t_j} \log[(1-\alpha)p_i + \alpha q_i] + \cdot \alpha \sum_i \frac{\partial q_i}{\partial t_j} \\
&= \alpha \sum_i \frac{\partial q_i}{\partial t_j} \{1 + \log[(1-\alpha)p_i + \alpha q_i]\} \\
&= \alpha \sum_i (1 + \log p_i^{lls}) \frac{\partial q_i}{\partial t_j} \quad \text{using } eq \text{ 5.5,} \\
&= \alpha \sum_i (1 + \log p_i^{lls}) \cdot q_j (I\{i = j\} - q_i) \\
&= \alpha q_j \sum_i (1 + \log p_i^{lls})(I\{i = j\} - q_i) \\
&= \alpha q_j \big[(1 + \log p_j^{lls}) - \sum_i q_i (1 + \log p_i^{lls})\big] \\
&= \alpha q_j \big[(1 + \log p_j^{lls}) - \sum_i q_i - \sum_i q_i \log p_i^{lls}\big] \\
&= \alpha q_j \big[1 + \log p_j^{lls} - 1 - \sum_i q_i \log p_i^{lls}\big] \\
&= \alpha q_j \big[\log p_j^{lls} - \sum_i q_i \log p_i^{lls}\big] \\
&= \alpha q_j \big(\log p_j^{lls} - \sum_i q_i \log p_i^{lls}\big) \qquad (5.13)
\end{aligned}
$$

The Final derivative of Forward KL $\mathcal{L}_{fkl}$ w.r.t. t_j can be obtained as:

$$\begin{aligned}
\frac{\partial \mathcal{L}_{fkl}}{\partial t_j} &= \frac{\partial \mathcal{L}_{fce}}{\partial t_j} + \frac{\partial \mathcal{L}_{emt}}{\partial t_j} \quad \text{using } eq \text{ } 5.11, \& \text{ } eq \text{ } 5.13, \\
&= \alpha q_j \big(\sum_i q_i \cdot \log \hat{p}_i - \log \hat{p}_j \big) \\
&+ \alpha q_j \big(\log p_j^{lls} - \sum_i q_i \log p_i^{lls} \big) \\
&= \alpha q_j \Bigg(\sum_i q_i \cdot \log \frac{\hat{p}_i}{p_i^{lls}} - \log \frac{\hat{p}_j}{p_j^{lls}} \Bigg) \qquad (5.14)
\end{aligned}$$

It can be observed that there is a notable disparity in the gradient of forward KL with respect to t_j as seen in eq 5.14, which is consistently one to two orders of magnitude smaller compared to its counterpart concerning z_j in eq 5.8. This discrepancy arises due to the logarithmic scaling effect on the gradients, resulting in a reduction in magnitude. To enhance the flow of gradients into the Q-Matrix without resorting to increasing the learning rate, I incorporate reverse KL $\mathcal{L}_{rkl}$ in the training loss function. Next, I show the gradient of the reverse KL $\mathcal{L}_{rkl}$ w.r.t. logits t_j of Q-Matrix to understand its impact.

The gradient of reverse KL $\mathcal{L}_{rkl}$ w.r.t. logits t_j of Q-Matrix can be derived as:

$$
\begin{aligned}
\mathcal{L}_{rkl} &= KL(\hat{p}||p^{lls}) = H(\hat{p}, p^{lls}) - H(\hat{p}) \\
&= -\sum_i \hat{p}_i \log p_i^{lls} + \sum_i \hat{p}_i \log \hat{p}_i \\
&= -\sum_i \hat{p}_i \log[(1-\alpha)p_i + \alpha q_i] + \sum_i \hat{p}_i \log \hat{p}_i \qquad (5.15) \\
\frac{\partial \mathcal{L}_{rkl}}{\partial t_j} &= -\sum_i \frac{\hat{p}_i}{(1-\alpha)p_i + \alpha q_i} \cdot \alpha \frac{\partial q_i}{\partial t_j} + 0 \\
&= -\alpha \sum_i \frac{\hat{p}_i}{(1-\alpha)p_i + \alpha q_i} \cdot \frac{\partial q_i}{\partial t_j} \quad \text{using } eq \text{ 5.5,} \\
&= -\alpha \sum_i \frac{\hat{p}_i}{(1-\alpha)p_i + \alpha q_i} \cdot q_j(I\{i=j\} - q_i) \\
&= -\alpha q_j \sum_i \frac{\hat{p}_i}{(1-\alpha)p_i + \alpha q_i} \cdot (I\{i=j\} - q_i) \\
&= -\alpha q_j \left[\frac{\hat{p}_j}{(1-\alpha)p_j + \alpha q_j} - \sum_i \frac{\hat{p}_i \cdot q_i}{(1-\alpha)p_i + \alpha q_i} \right] \\
&= -\alpha q_j \left[\frac{\hat{p}_j}{p_j^{lls}} - \sum_i \frac{\hat{p}_i \cdot q_i}{p_i^{lls}} \right] \\
&= \alpha q_j \left(\sum_i q_i \cdot \frac{\hat{p}_i}{p_i^{lls}} - \frac{\hat{p}_j}{p_j^{lls}} \right) \qquad (5.16)
\end{aligned}
$$

By examining eq 5.14 and 5.16, it becomes apparent that the gradients of forward and reverse KL exhibit strong similarities, differing primarily due to the presence of the *log* function in the $\frac{\hat{p}_i}{p_i^{lls}}$ terms. log function diminished the gradient values in the forward KL scenario, whereas, in reverse KL, the unscaled values are employed. Interestingly, the gradients derived from reverse KL align in magnitude order with those of forward KL concerning z in eq 5.8. This leads to a better convergence of the Q-Matrix.

Hence, I use objective for training using the LLS as the sum of Forward-KL and Reverse-KL, which is also termed the Jensen-Shannon Divergence, given by,

$$\begin{aligned}\mathcal{L}_{lls} &= JSD(p^{lls}, \hat{p}) = KL(p^{lls}, \hat{p}) + KL(\hat{p}, p^{lls}) \\ &= p^{lls} \log p^{lls} - p^{lls} \log \hat{p} + \hat{p} \log \hat{p} - \hat{p} \log p^{lls} \\ &= -H(p^{lls}) + H(p^{lls}, \hat{p}) - H(\hat{p}) + H(\hat{p}, p^{lls}).\end{aligned} \tag{5.17}$$

The first term $-H(p^{lls})$, is the negative entropy of the target, which when minimized drives the target p^{lls} towards a uniform distribution. The target is $p^{lls} = (1-\alpha)p + \alpha Q_y$, where only Q_y varies. Minimizing $-H(p^{lls})$ effectively drives Q_y to estimate inter-category relationships as Q_y becomes more uniform. Similarly, the third term $-H(\hat{p})$, is the negative entropy of the predictions, which when minimized drives the predictions $\hat{p}$ towards a uniform distribution. This plays the role of Label Smoothing which penalizes overconfidence in the predictions and delays overfitting. I name the second term $-H(p^{lls}, \hat{p})$, Forward Cross-Entropy, which aligns distributions p^{lls} and $\hat{p}$. Similarly, I name the 4th term $-H(\hat{p}, p^{lls})$ Reverse Cross-Entropy. Minimizing these terms aligns the target p^{lls} with the predictions $\hat{p}$.

5.1.4 The Q-Matrix

Minimizing $-H(p^{lls})$ maximizes the entropy of the target $p^{lls} = (1-\alpha)p + \alpha Q_y$, where only Q_y varies. The entropy of p^{lls} can be increased only by reducing Q_{yy} and increasing the other components of Q_y because the y-th component p_y^{lls} is greater than the other components by a fixed constant term $(1-\alpha)$. This propels the network to set $Q_{yy} \rightarrow 0$ and assign that probability to the other categories, thereby identifying inter-category relationships. Consequently, the Q matrix exhibits the lowest values at the diagonals and higher values for semantically closer categories.

```
# Define LLS
class LLS(nn.Module):
  def __init__(self, K, alpha=0.1):
    super().__init__()
    self.K = K
    self.alpha = alpha
    self.qmatrix = nn.Parameter(torch.zeros(K, K),
      requires_grad=True)

  def forward(self, logits, y):
    pred = F.softmax(logits, 1)

    y_tgt = (1- α) * F.one_hot(y, num_classes=self.K)
      + α * F.softmax(self.qmatrix[y], 1)

    forward_kl = KL(y_tgt, pred)
    backward_kl = KL(pred, y_tgt)
    loss = (forward_kl + backward_kl)/2

    return loss

# Define loss function
loss_fn = LLS(K, α)

# Define optimizer
# Set weight decay to 0 for Q-Matrix
params = [{'params': net.parameters(), 'weight_decay': wd},
       {'params': loss_fn.parameters(), 'weight_decay': 0}]
optimizer = SGD(params, lr, mom)
```

I learn a Q_y vector for every category. This results in a $K \times K$ Q-matrix where every row Q_y models the similarities of category y with the other categories. I observed a low variance in the Q-Matrix across randomized Q-Matrix initializations, network initializations, and network architectures. This allows for knowledge transfer between different networks. The Q-Matrix is primarily dependent on the training data but proves useful for transferring knowledge to category-wise and sample-wise subsets, as explored in Section 5.3.6. The LLS method is depicted in the model diagram in Figure 5.2.

5.1.5 PyTorch-like Pseudo Code

On the previous page, I showed the PyTorch-like Pseudo Code that can easily be implemented for training a neural network with Learnable Label Smoothing.

5.2 Experiments

5.2.1 Datasets and Setup

I scrutinized my methodology across diverse settings, encompassing small-scale objects, large-scale objects, and scenarios demanding fine-grained classification. In the realm of small-scale classification, I used FashionMNIST [Xiao *et al.* (2017a)], CIFAR10 [Krizhevsky *et al.* (2009)], and SVHN [Netzer *et al.* (2011)] datasets. These datasets, with images sized at 32×32, offer both diversity and challenges with 10-way classifications. SVHN presents an intriguing challenge as digits lack prominent inter-category relationships. For large-scale classification, my evaluation extended to CIFAR100 [Krizhevsky *et al.* (2009)], Tiny-ImageNet, and ImageNet-100. Due to hardware constraints, I leveraged Tiny-ImageNet and ImageNet-100 [URL], both subsets of the original ImageNet dataset [Deng *et al.* (2009)]. Tiny-ImageNet possesses 200 categories with 64×64 images, while ImageNet-100, featuring the original 224×224 image size, encompasses 100 categories. In the fine-grained classification domain, my experiments focused on distinguishing between various bird species using the CUB-200 dataset [Wah *et al.* (2011)] and different types of flowers using the Flowers-102 dataset [Nilsback and Zisserman (2008)]. Below is the training configuration used for these datasets.

CIFAR10 and CIFAR100

- Augmentations: Utilized padding of size 4, Random Crops, and random horizontal flips during training.
- Optimizer: Employed SGD optimizer with 0.9 momentum and weight decay of 5e-4.
- Training specifics: Networks were trained with a batch size of 128 for 300 epochs. The learning rate initiated at 0.1 and warmed up linearly for the first 10 epochs. Then, it decayed by a factor of 0.1 at the 150th and 225th epochs.

FashionMNIST

- Augmentation: Applied padding of size 2 with random crops as the sole augmentation.
- Network Configuration: Set the input channels to 1 for grayscale images.
- Optimizer: Employed SGD optimizer with 0.9 momentum and weight decay of 1e-4.
- Training specifics: Networks were trained with a batch size of 128 for 200 epochs. The learning rate began at 0.1, underwent a linear warm-up for the initial 5 epochs, and decayed by a factor of 0.1 at the 100th and 150th epochs.

SVHN

- No augmentation was used for this dataset.
- Optimizer: Used SGD optimizer with 0.9 momentum and weight decay of 1e-4.
- Training specifics: Networks were trained with a batch size of 128 for 200 epochs. The learning rate started at 0.1, had a linear warm-up for the first 5 epochs, and decayed by a factor of 0.1 at the 100th and 150th epochs.

Tiny-ImageNet

- Image size: Images in Tiny-ImageNet data were of size 64×64.
- Augmentations: Implemented padding of size 4, Random Crops, and random Horizontal flips.
- Optimizer: Used SGD optimizer with 0.9 momentum and weight decay of 1e-4.
- Training specifics: Networks were trained with a batch size of 64 for 100 epochs. The learning rate began at 0.1, underwent a linear warm-up for the first 5 epochs, and decayed by a factor of 0.1 at the 40th and 60th epochs.

ImageNet-100

- Image size: Training images were of the original ImageNet dataset size 224×224.
- Augmentations: Employed (1) Standard augmentation of random resized crops of 224 along with random Horizontal flips. (2) Standard augmentation with RandAugmentation.
- Optimizer: Utilized SGD optimizer with 0.9 momentum and weight decay of 1e-4.

Dataset	CUB-200				Flowers-102			
Network	MV2	R18	R50	R101	MV2	R18	R50	R101
1-Hot	77.76	78.08	80.81	81.71	91.03	90.37	90.69	91.74
LS	78.67	<u>78.56</u>	81.89	<u>82.62</u>	91.94	<u>90.50</u>	92.42	<u>92.73</u>
TF-KD$_{reg}$	77.64	-	80.96	-	91.95	-	91.30	-
OLS	**79.95**	-	<u>82.47</u>	-	<u>92.73</u>	-	<u>92.86</u>	-
LLS	<u>79.84</u>	**78.86**	**82.91**	**83.48**	**93.02**	**91.02**	**93.64**	**92.89**

Table 5.1: Comparison of classification accuracies on CUB-200 and Flowers-102 for fine-grain classification. MV2: MobileNetV2 and RX denote the ResNet network with X number of layers.

- Training specifics: Networks were trained with a batch size of 64 for 90 epochs. The learning rate began at 0.1, underwent a linear warm-up for the first 5 epochs, and decayed by a factor of 0.1 at the 30th, 60th, and 80th epochs.

CUB200 and Flowers102

- Approach: Utilized pretrained networks for these datasets, adapting the last classification layer based on the dataset's class count.
- Augmentation: Images were scaled to 256 and then randomly cropped to 224 for augmentation, along with random horizontal flips.
- Optimizer: Employed SGD optimizer with 0.9 momentum and weight decay of 1e-4.
- Training specifics: Networks were trained with a batch size of 64 for 100 epochs.

	CIFAR100				Tiny-ImageNet		
Method	R18	R34	R50	R101	R18	R50	R101
1-Hot	75.87	79.38	78.79	79.66	<u>63.20</u>	67.47	67.93
LS	77.26	79.06	78.80	79.88	63.13	67.63	<u>68.31</u>
FL-3	-	-	77.25	-	-	50.31	62.97
FLSD-53	-	-	76.78	-	-	50.94	62.96
TF-KD$_{self}$	77.10	-	-	-	-	68.18	-
TF-KD$_{reg}$	77.36	-	-	-	-	67.92	-
Zipf	<u>77.38</u>	77.38	-	-	59.25	-	-
OLS	-	**79.96**	<u>79.35</u>	<u>80.34</u>	-	-	-
LLS	**78.96**	<u>79.78</u>	**80.40**	**80.46**	**64.58**	**68.23**	**69.07**

Table 5.2: Comparison of classification accuracies on CIFAR100 and Tiny-ImageNet datasets. RX denotes the ResNet network with X number of layers.

The learning rate initiated at 0.01, underwent a linear warm-up for the first 5 epochs and decayed by a factor of 0.1 at the 45th and 80th epochs.

I evaluated my approach on these datasets using different networks that are mentioned in their respective tables. For ImageNet-100, I conducted tests with both standard augmentation and RandAug [Cubuk *et al.* (2020)]. I stored Q-matrix as logits which are then converted to probabilities using Softmax. The Q-matrix is initialized with zeros, leading to a uniform distribution as the starting point. The hyper-parameter α is set to 0.1 for all experiments.

Dataset	CIFAR10		SVHN	FMNIST	ImageNet-100			
Network	R18	R50	R18	R18	R18	R50	R18†	R50†
1-Hot	**95.60**	**95.92**	95.57	86.66	81.26	83.96	82.88	84.82
LS	<u>95.49</u>	95.31	<u>95.86</u>	<u>88.19</u>	<u>82.42</u>	<u>84.58</u>	<u>83.36</u>	<u>85.70</u>
LLS	<u>95.49</u>	<u>95.62</u>	**95.92**	**88.31**	**82.72**	**84.60**	**83.50**	**86.02**

Table 5.3: Comparison of classification accuracies on CIFAR10, SVHN, FashionMNIST and ImageNet-100 datasets. † denotes the network was trained with RandAug augmentation. RX denotes the ResNet network with X number of layers.

5.2.2 Results

I conduct a comprehensive comparison of my approach against prominent label regularization techniques, including Label Smoothing [Szegedy *et al.* (2016a)], Focal Loss [Mukhoti *et al.* (2020)], Teacher-Free Knowledge Distillation [Yuan *et al.* (2020)], and Online Label Smoothing (OLS) [Zhang *et al.* (2021)]. The results are detailed in Table 5.1, 5.2, and 5.3. When results were not available in the original paper I indicated them with '-'. My approach consistently outperforms the alternatives across the majority of the cases. Notably, OLS yields results comparable to my method; however, my approach entails significantly lower computational overhead. It is noteworthy that Label Smoothing tends to exhibit degenerative effects on the CIFAR10 dataset [Müller *et al.* (2019); Wang *et al.* (2019b)]. Nevertheless, my approach achieves comparatively higher performance even in those cases. I found Learnable label smoothing to be most helpful for tasks with ≥ 100 classes. My approach imposes minimal overhead while achieving superior performance.

Dataset	Tiny-ImageNet		ImageNet-100	
Augmentation	Standard	RandAugment	Standard	RandAugment
1-hot	39.48	44.45	62.16	70.36
LS	39.98	44.86	62.20	70.56
LLS	**40.36**	**45.46**	**63.86**	**70.72**

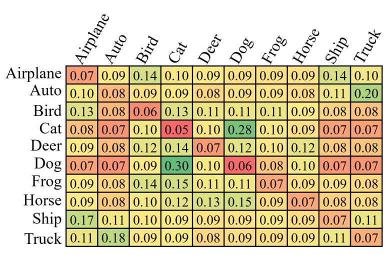
Table 5.4: Results on Tiny-ImageNet and ImageNet-100 with DEIT-Tiny using Standard and RandAugment as base augmentations.

5.2.3 Results with Vision Transformer

I show results on Tiny-ImageNet and ImageNet-100 with DEIT-Tiny [Touvron *et al.* (2021)] in Table 5.4. I used the 64×64 image size for Tiny-ImageNet and 224×224 for ImageNet-100 with patch sizes of 8 and 16 respectively. Vision Transformers are data-hungry and demand heavy regularization [Dosovitskiy *et al.* (2020)]. Hence, I also performed experiments with RandAugment [Cubuk *et al.* (2020)] as the base input augmentation.

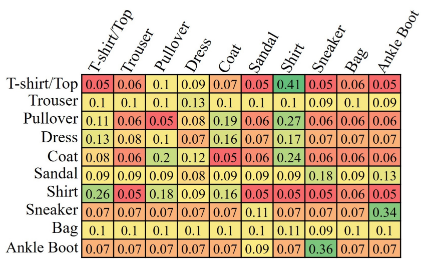
5.2.4 Q-Matrix

I present Q-Matrices for CIFAR10, FashionMNIST, and SVHN in Figure 5.3, showcasing their learned relationships. The Q-Matrix notably reveals distinct connections among the categories. For SVHN, where inter-category relationships are less pronounced, a majority of the values approach $\frac{1}{K}$ (0.1). Furthermore, the confusion matrix depicted in Figure 5.4a for the FashionMNIST reveals a pattern consistent with the Q-Matrix showcased in Figure 5.4b. For instance, Shirts frequently get misclassified as T-shirts, followed by pullovers and coats, owing to their close semantic



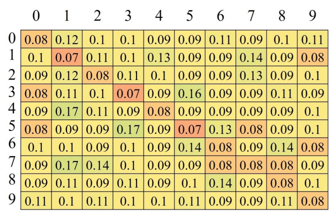
	Airplane	Auto	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck
Airplane	0.07	0.09	0.14	0.10	0.09	0.09	0.09	0.09	0.14	0.10
Auto	0.10	0.08	0.09	0.09	0.08	0.09	0.09	0.08	0.11	0.20
Bird	0.13	0.08	0.06	0.13	0.11	0.11	0.11	0.09	0.08	0.08
Cat	0.08	0.07	0.10	0.05	0.10	0.28	0.10	0.09	0.07	0.07
Deer	0.09	0.08	0.12	0.14	0.07	0.12	0.10	0.12	0.08	0.08
Dog	0.07	0.07	0.09	0.30	0.10	0.06	0.08	0.10	0.07	0.07
Frog	0.09	0.08	0.14	0.15	0.11	0.11	0.07	0.09	0.09	0.08
Horse	0.09	0.08	0.10	0.12	0.13	0.15	0.09	0.07	0.08	0.08
Ship	0.17	0.11	0.10	0.09	0.09	0.09	0.09	0.09	0.07	0.11
Truck	0.11	0.18	0.09	0.09	0.08	0.09	0.09	0.09	0.11	0.07

(a) CIFAR10



	T-shirt/Top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle Boot
T-shirt/Top	0.05	0.06	0.1	0.09	0.07	0.05	0.41	0.05	0.06	0.05
Trouser	0.1	0.1	0.1	0.13	0.1	0.1	0.1	0.09	0.1	0.09
Pullover	0.11	0.06	0.05	0.08	0.19	0.06	0.27	0.06	0.06	0.06
Dress	0.13	0.08	0.1	0.07	0.16	0.07	0.17	0.07	0.07	0.07
Coat	0.08	0.06	0.2	0.12	0.05	0.06	0.24	0.06	0.06	0.06
Sandal	0.09	0.09	0.09	0.08	0.09	0.09	0.09	0.18	0.09	0.13
Shirt	0.26	0.05	0.18	0.09	0.16	0.05	0.05	0.05	0.06	0.05
Sneaker	0.07	0.07	0.07	0.07	0.07	0.11	0.07	0.07	0.07	0.34
Bag	0.1	0.1	0.1	0.1	0.1	0.1	0.11	0.09	0.1	0.1
Ankle Boot	0.07	0.07	0.07	0.07	0.07	0.09	0.07	0.36	0.07	0.07

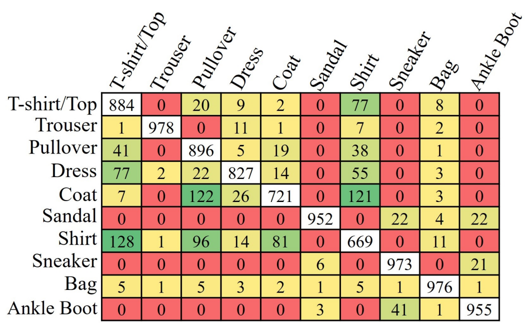
(b) FashionMNIST



	0	1	2	3	4	5	6	7	8	9
0	0.08	0.12	0.1	0.1	0.09	0.09	0.11	0.09	0.1	0.11
1	0.1	0.07	0.11	0.1	0.13	0.09	0.09	0.14	0.09	0.08
2	0.09	0.12	0.08	0.11	0.1	0.09	0.09	0.13	0.09	0.1
3	0.08	0.11	0.1	0.07	0.09	0.16	0.09	0.09	0.11	0.09
4	0.09	0.17	0.11	0.09	0.08	0.09	0.09	0.09	0.09	0.1
5	0.08	0.09	0.09	0.17	0.09	0.07	0.13	0.08	0.09	0.1
6	0.1	0.1	0.09	0.1	0.09	0.14	0.08	0.09	0.14	0.08
7	0.09	0.17	0.14	0.1	0.09	0.09	0.08	0.08	0.08	0.09
8	0.09	0.11	0.09	0.11	0.09	0.1	0.14	0.09	0.08	0.1
9	0.11	0.1	0.11	0.1	0.1	0.11	0.09	0.09	0.11	0.08

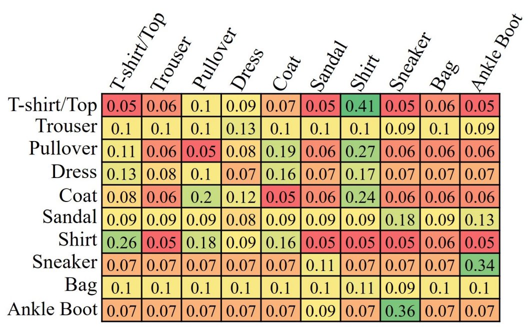
(c) SVHN

Figure 5.3: Sample learned Q Matrices. It can be observed that the Q-Matrix favors semantically closer categories. The final training label is obtained by mixing the Q-Matrix with the 1-hot vector of ground truth based on the α hyperparameter.



	T-shirt/Top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle Boot
T-shirt/Top	884	0	20	9	2	0	77	0	8	0
Trouser	1	978	0	11	1	0	7	0	2	0
Pullover	41	0	896	5	19	0	38	0	1	0
Dress	77	2	22	827	14	0	55	0	3	0
Coat	7	0	122	26	721	0	121	0	3	0
Sandal	0	0	0	0	0	952	0	22	4	22
Shirt	128	1	96	14	81	0	669	0	11	0
Sneaker	0	0	0	0	0	6	0	973	0	21
Bag	5	1	5	3	2	1	5	1	976	1
Ankle Boot	0	0	0	0	0	3	0	41	1	955

(a) Confusion Matrix on the test set.



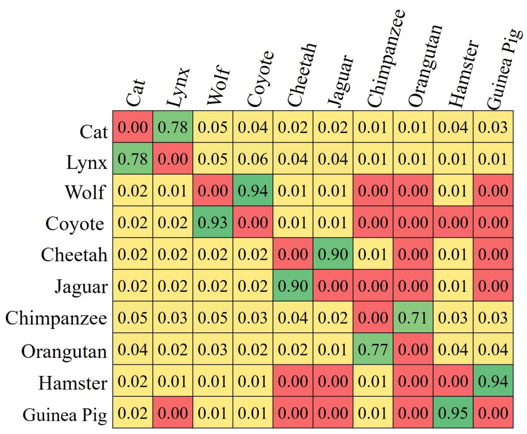
	T-shirt/Top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle Boot
T-shirt/Top	0.05	0.06	0.1	0.09	0.07	0.05	0.41	0.05	0.06	0.05
Trouser	0.1	0.1	0.1	0.13	0.1	0.1	0.1	0.09	0.1	0.09
Pullover	0.11	0.06	0.05	0.08	0.19	0.06	0.27	0.06	0.06	0.06
Dress	0.13	0.08	0.1	0.07	0.16	0.07	0.17	0.07	0.07	0.07
Coat	0.08	0.06	0.2	0.12	0.05	0.06	0.24	0.06	0.06	0.06
Sandal	0.09	0.09	0.09	0.08	0.09	0.09	0.09	0.18	0.09	0.13
Shirt	0.26	0.05	0.18	0.09	0.16	0.05	0.05	0.05	0.06	0.05
Sneaker	0.07	0.07	0.07	0.07	0.07	0.11	0.07	0.07	0.07	0.34
Bag	0.1	0.1	0.1	0.1	0.1	0.1	0.11	0.09	0.1	0.1
Ankle Boot	0.07	0.07	0.07	0.07	0.07	0.09	0.07	0.36	0.07	0.07

(b) Learned Q-Matrix from train set.

Figure 5.4: Confusion Matrix and Learned Q-Matrix of the FashionMNIST dataset. It can be observed that misclassifications in Figure 5.4a follow the same trend as the relationship learned by the Q-Matrix in Figure 5.4b.

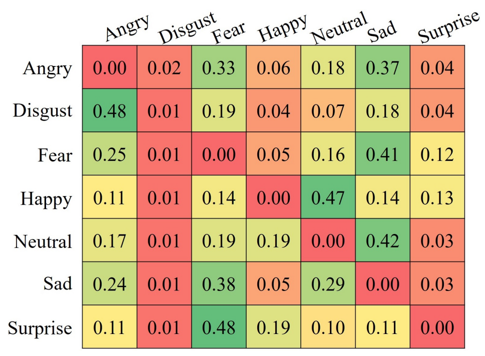
ties in that order. This correlation serves as a useful tool for estimating prediction uncertainty. For example, when an image is misclassified as a T-shirt, there is a higher likelihood of it being a Shirt and a significantly lower chance of being a Bag.

I show extra learned Q-Matrix on Animals-10N and FER2013 dataset in Figure 5.5. Animals-10N has a fine-grain classification task among 5 confusion pairs of animals. It can be observed that the network favors the other animal of the pairs



	Cat	Lynx	Wolf	Coyote	Cheetah	Jaguar	Chimpanzee	Orangutan	Hamster	Guinea Pig
Cat	0.00	0.78	0.05	0.04	0.02	0.02	0.01	0.01	0.04	0.03
Lynx	0.78	0.00	0.05	0.06	0.04	0.04	0.01	0.01	0.01	0.01
Wolf	0.02	0.01	0.00	0.94	0.01	0.01	0.00	0.00	0.01	0.00
Coyote	0.02	0.02	0.93	0.00	0.01	0.01	0.00	0.00	0.00	0.00
Cheetah	0.02	0.02	0.02	0.02	0.00	0.90	0.01	0.00	0.01	0.00
Jaguar	0.02	0.02	0.02	0.02	0.90	0.00	0.00	0.00	0.01	0.00
Chimpanzee	0.05	0.03	0.05	0.03	0.04	0.02	0.00	0.71	0.03	0.03
Orangutan	0.04	0.02	0.03	0.02	0.02	0.01	0.77	0.00	0.04	0.04
Hamster	0.02	0.01	0.01	0.01	0.00	0.00	0.01	0.00	0.00	0.94
Guinea Pig	0.02	0.00	0.01	0.01	0.00	0.00	0.01	0.00	0.95	0.00

(a) Animals-10N



	Angry	Disgust	Fear	Happy	Neutral	Sad	Surprise
Angry	0.00	0.02	0.33	0.06	0.18	0.37	0.04
Disgust	0.48	0.01	0.19	0.04	0.07	0.18	0.04
Fear	0.25	0.01	0.00	0.05	0.16	0.41	0.12
Happy	0.11	0.01	0.14	0.00	0.47	0.14	0.13
Neutral	0.17	0.01	0.19	0.19	0.00	0.42	0.03
Sad	0.24	0.01	0.38	0.05	0.29	0.00	0.03
Surprise	0.11	0.01	0.48	0.19	0.10	0.11	0.00

(b) FER2013

Figure 5.5: More Q-Matrices. Animals-10N has a fine-grain classification task among 5 confusion pairs of animals. It can be observed that the network favors the other animal of the pairs while assigning confidence. FER2013 has 7 basic facial expressions and the network favors the expressions that use similar facial movements

while assigning confidence. FER2013 has 7 basic facial expressions and the network favors the expressions that use similar facial movements.

5.3 Analysis

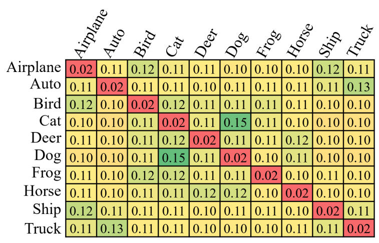
5.3.1 Ablation Study

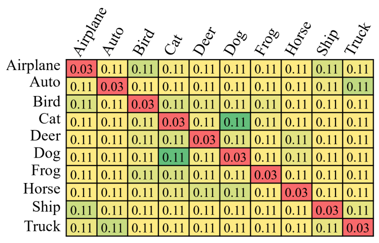
I conduct an ablation study on diverse loss components, as presented in Table 5.5, utilizing the Tiny-ImageNet and CUB-200 datasets. The initial four rows of the table demonstrate the outcomes obtained by excluding each individual component. The fifth and sixth rows correspond to the cross-entropy and symmetric cross-entropy loss, respectively. Subsequently, the sixth and seventh rows represent the forward and Reverse KL divergence losses. Based on the first and the last row, It can be observed that the cross-entropy loss is crucial and this component's absence results in

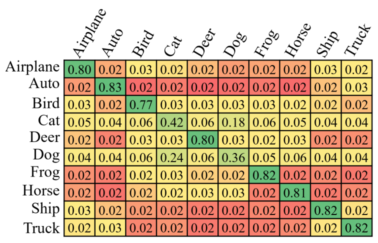
Description	Loss Terms				Tiny-ImageNet		CUB-200	
	$H(p^{lls}, \hat{p})$	$H(\hat{p}, p^{lls})$	$-H(\hat{p})$	$-H(p^{lls})$	R18	R50	R18	R50
No FCE	✗	✓	✓	✓	26.44	11.91	46.84	51.26
No RCE	✓	✗	✓	✓	64.14	68.04	78.84	82.88
No Pred EM	✓	✓	✗	✓	63.26	67.48	78.34	82.57
No Targets EM	✓	✓	✓	✗	63.80	67.51	78.10	82.78
FCE only	✓				63.40	66.83	78.46	82.66
Symmetric CE	✓	✓	✗	✗	62.91	66.80	78.13	82.07
Forward KL	✓	✗	✗	✓	63.03	67.87	78.22	82.52
Reverse KL	✗	✓	✓	✗	26.58	14.40	46.62	53.56
LLS	✓	✓	✓	✓	**64.58**	**68.23**	**78.86**	**82.91**

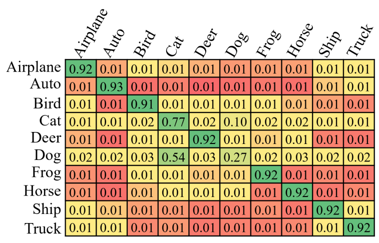
Table 5.5: Abalation Study experiments on Tiny-ImageNet and CUB-200 with ResNet-18 and ResNet-50. No FCE: No cross-entropy loss; No RCE: No reverse cross-entropy loss; No Pred EM: No entropy maximization loss on predictions; No Targets EM: No entropy maximization loss on targets; FCE only: Forward cross-entropy; Symmetric CE: Forward cross-entropy + Reverse cross-entropy.

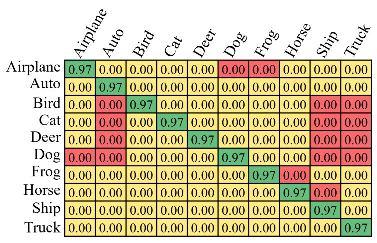
a failure of network convergence. Nevertheless, achieving the network's optimal performance necessitates the inclusion of all loss components. I also present the learned Q-Matrices on CIFAR10 for the same experiments. The outcomes are illustrated in Figure 5.6. Notably, it's apparent that the Q-Matrix achieves its optimal state exclusively with backward KL and JSD loss functions. Conversely, employing the Forward KL approach results in slower convergence, ultimately leading to suboptimal values.

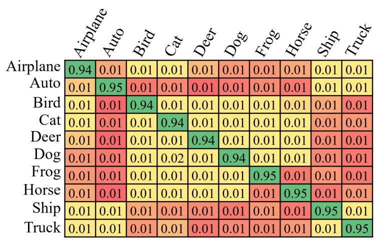

(a) No Cross-Entropy

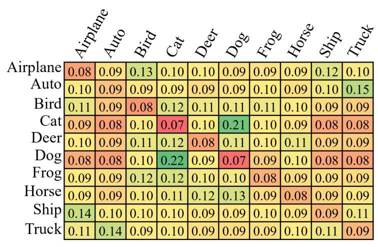

(b) No Reverse cross-entropy

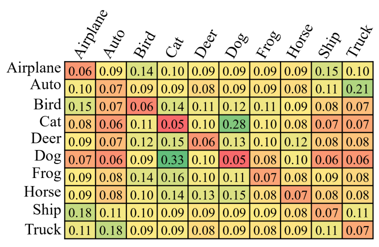

(c) No entropy max on $\hat{y}$


(d) No entropy max on $\bar{y}$


(e) Cross-entropy


(f) Symmetric cross-entropy


(g) Forward KL


(h) Backward KL

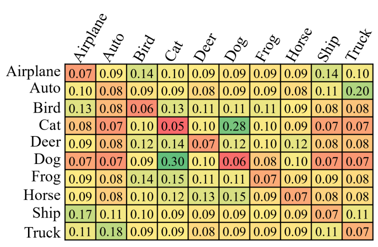

(i) JSD

Figure 5.6: Learned Q-Matrix on CIFAR10 as per the ablation study experiments.

5.3.2 Clusters Visualization

I present a visual analysis of clusters formed by 1-hot, Label Smoothing, and Learnable Label Smoothing targets using TSNE [Van der Maaten and Hinton (2008)]. Following the experimental framework detailed in [Müller *et al.* (2019)], I conducted my evaluations on both CIFAR10 and CIFAR100 datasets. Specifically, I trained AlexNet on CIFAR10 and ResNet-56 on CIFAR100, utilizing the penultimate features of the training set for visualization purposes. For these experiments, I concentrated on the *Dog*, *Cat*, and *Truck* classes from CIFAR10, and the *Beaver*, *Dolphin*, and *Otter* classes from CIFAR100.

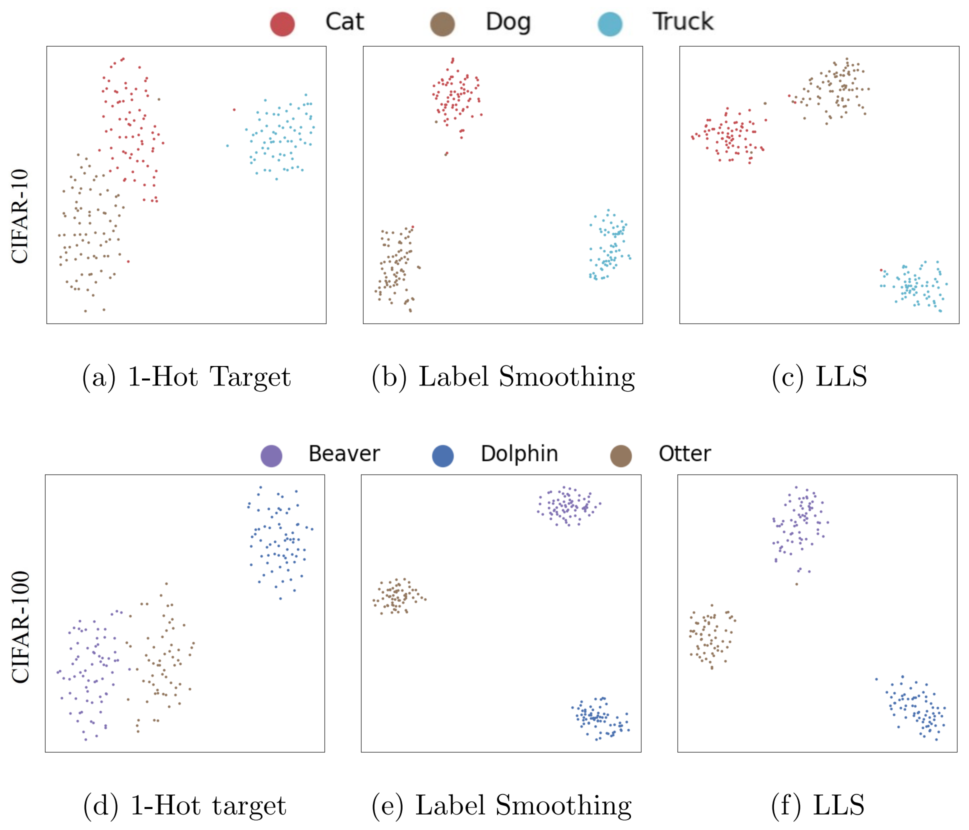
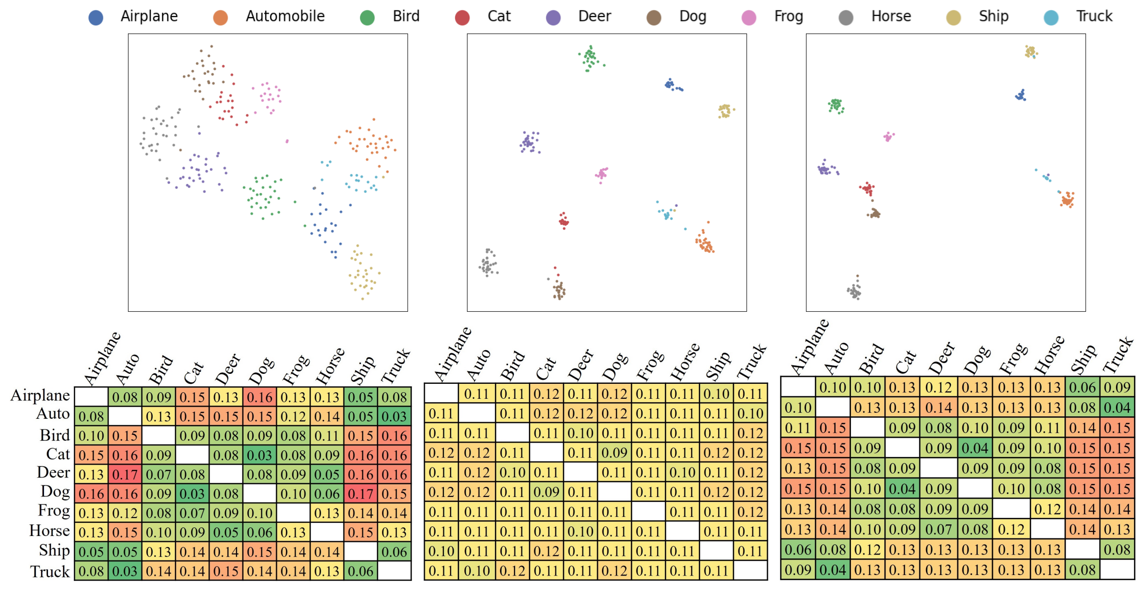


Figure 5.7: The TSNE visualizations illustrate three classes from CIFAR-10 (*Cat*, *Dog*, *Truck*) in the top row and CIFAR-100 (*Beaver*, *Dolphin*, *Otter*) in the bottom row. Notably, clusters formed by employing one-hot targets appear widely scattered, contrasting with the more tightly knit clusters generated by label smoothing and learnable label smoothing techniques. A noteworthy observation is the proximity of semantically related classes, such as *Dat* and *Dog* in CIFAR-10, or *Beaver* and *Otter* in CIFAR-100, when trained using 1-hot target and learnable label smoothing—these classes exhibit closer clusters in comparison to a third unrelated class. However, the use of label smoothing disrupts this inherent relationship by placing them at equal distances, effectively erasing the semantic closeness among them.

The results are showcased in Figure 5.7, significantly reinforcing my findings. Notably, clusters formed by employing one-hot targets appear widely scattered, contrast-



	Airplane	Auto	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck
Airplane		0.08	0.09	0.15	0.13	0.16	0.13	0.13	0.05	0.08
Auto	0.08		0.13	0.15	0.15	0.15	0.12	0.14	0.05	0.03
Bird	0.10	0.15		0.09	0.08	0.09	0.08	0.11	0.15	0.16
Cat	0.15	0.16	0.09		0.08	0.03	0.08	0.09	0.16	0.16
Deer	0.13	0.17	0.07	0.08		0.08	0.09	0.05	0.16	0.16
Dog	0.16	0.16	0.09	0.03	0.08		0.10	0.06	0.17	0.15
Frog	0.13	0.12	0.08	0.07	0.09	0.10		0.13	0.14	0.14
Horse	0.13	0.15	0.10	0.09	0.05	0.06	0.13		0.15	0.13
Ship	0.05	0.05	0.13	0.14	0.14	0.15	0.14	0.14		0.06
Truck	0.08	0.03	0.14	0.14	0.15	0.14	0.14	0.13	0.06	

(a) 1-hot Target

	Airplane	Auto	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck
Airplane		0.11	0.11	0.12	0.11	0.12	0.11	0.11	0.10	0.11
Auto	0.11		0.11	0.12	0.12	0.12	0.11	0.11	0.11	0.10
Bird	0.11	0.11		0.11	0.10	0.11	0.11	0.11	0.11	0.12
Cat	0.12	0.12	0.11		0.11	0.09	0.11	0.11	0.12	0.12
Deer	0.11	0.12	0.10	0.11		0.11	0.11	0.10	0.11	0.12
Dog	0.12	0.12	0.11	0.09	0.11		0.11	0.11	0.12	0.12
Frog	0.11	0.11	0.11	0.11	0.11	0.11		0.11	0.11	0.12
Horse	0.11	0.11	0.11	0.11	0.10	0.11	0.11		0.11	0.11
Ship	0.10	0.11	0.11	0.12	0.11	0.11	0.11	0.11		0.11
Truck	0.11	0.10	0.12	0.11	0.11	0.12	0.11	0.11	0.11	

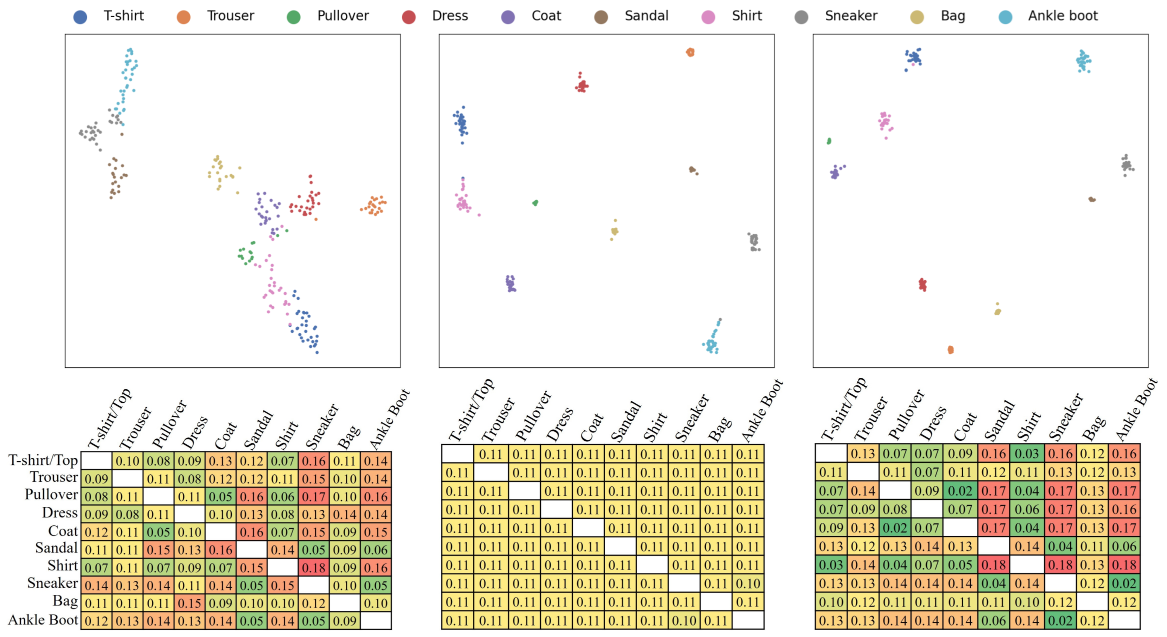
(b) Label Smoothing

	Airplane	Auto	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck
Airplane		0.10	0.10	0.13	0.12	0.13	0.13	0.13	0.06	0.09
Auto	0.10		0.13	0.13	0.14	0.13	0.13	0.13	0.08	0.04
Bird	0.11	0.15		0.09	0.08	0.10	0.09	0.11	0.14	0.15
Cat	0.15	0.15	0.09		0.09	0.04	0.09	0.10	0.15	0.15
Deer	0.13	0.15	0.08	0.09		0.09	0.09	0.08	0.15	0.15
Dog	0.15	0.15	0.10	0.04	0.09		0.10	0.08	0.15	0.15
Frog	0.13	0.14	0.08	0.08	0.09	0.09		0.12	0.14	0.14
Horse	0.13	0.14	0.10	0.09	0.07	0.08	0.12		0.14	0.13
Ship	0.06	0.08	0.12	0.13	0.13	0.13	0.13	0.13		0.08
Truck	0.09	0.04	0.13	0.13	0.13	0.13	0.13	0.13	0.08	

(c) LLS

Figure 5.8: Upper Row: TSNE visualization depicting penultimate features of CIFAR-10. Lower Row: L_1 normalized cosine distance between category centers to depict inter-category relationships. In the upper row, it is evident that the category clusters associated with 1-hot targets exhibit dispersion, while those of Label Smoothing and Learnable Label Smoothing appear more concentrated. In the lower row, it becomes apparent that Label Smoothing disrupts the inter-category relationships, resulting in equal distances between features of all categories. Conversely, 1-hot targets and Learnable Label Smoothing maintain and have similar inter-category relationships. Learnable Label Smoothing provides the advantages of both techniques.

ing with the more tightly knit clusters generated by label smoothing and learnable label smoothing techniques. A noteworthy observation is the proximity of semantically related classes, such as *Cat* and *Dog* in CIFAR10, or *Beaver* and *Otter* in CIFAR100, when trained using 1-hot target and learnable label smoothing—these classes exhibit closer clusters in comparison to a third unrelated class. However, the



	T-shirt/Top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle Boot
T-shirt/Top		0.10	0.08	0.09	0.13	0.12	0.07	0.16	0.11	0.14
Trouser	0.09		0.11	0.08	0.12	0.12	0.11	0.15	0.10	0.14
Pullover	0.08	0.11		0.11	0.05	0.16	0.06	0.17	0.10	0.16
Dress	0.09	0.08	0.11		0.10	0.13	0.08	0.13	0.14	0.14
Coat	0.12	0.11	0.05	0.10		0.16	0.07	0.15	0.09	0.15
Sandal	0.11	0.11	0.15	0.13	0.16		0.14	0.05	0.09	0.06
Shirt	0.07	0.11	0.07	0.09	0.07	0.15		0.18	0.09	0.16
Sneaker	0.14	0.13	0.14	0.11	0.14	0.05	0.15		0.10	0.05
Bag	0.11	0.11	0.11	0.15	0.09	0.10	0.10	0.12		0.10
Ankle Boot	0.12	0.13	0.14	0.13	0.14	0.05	0.14	0.05	0.09	

(a) One-Hot Target

	T-shirt/Top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle Boot
		0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11
	0.11		0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11
	0.11	0.11		0.11	0.11	0.11	0.11	0.11	0.11	0.11
	0.11	0.11	0.11		0.11	0.11	0.11	0.11	0.11	0.11
	0.11	0.11	0.11	0.11		0.11	0.11	0.11	0.11	0.11
	0.11	0.11	0.11	0.11	0.11		0.11	0.11	0.11	0.11
	0.11	0.11	0.11	0.11	0.11	0.11		0.11	0.11	0.11
	0.11	0.11	0.11	0.11	0.11	0.11	0.11		0.11	0.10
	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11		0.11
	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.10	0.11	

(b) Label Smoothing

	T-shirt/Top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle Boot
		0.13	0.07	0.07	0.09	0.16	0.03	0.16	0.12	0.16
	0.11		0.11	0.07	0.11	0.12	0.11	0.13	0.12	0.13
	0.07	0.14		0.09	0.02	0.17	0.04	0.17	0.13	0.17
	0.07	0.09	0.08		0.07	0.17	0.06	0.17	0.13	0.16
	0.09	0.13	0.02	0.07		0.17	0.04	0.17	0.13	0.17
	0.13	0.12	0.13	0.14	0.13		0.14	0.04	0.11	0.06
	0.03	0.14	0.04	0.07	0.05	0.18		0.18	0.13	0.18
	0.13	0.13	0.14	0.14	0.14	0.04	0.14		0.12	0.02
	0.10	0.12	0.11	0.11	0.11	0.11	0.10	0.12		0.12
	0.13	0.13	0.14	0.14	0.14	0.06	0.14	0.02	0.12	

(c) LLS

Figure 5.9: Upper Row: TSNE visualization depicting penultimate features of FashionMNIST. Lower Row: L1 normalized cosine distance between class cluster centers. In the upper row, it is evident that the class clusters associated with one-hot targets exhibit dispersion, while those of Label Smoothing and Learnable Label Smoothing appear more concentrated. Moving to the lower row, it becomes apparent that label smoothing disrupts the inter-class relationships, resulting in equal distances between all classes. Conversely, one-hot targets and Learnable Label Smoothing maintain and preserve these inter-class relationships. Notably, Learnable Label Smoothing combines the advantages of both techniques.

use of label smoothing disrupts this inherent relationship by placing them at equal distances, effectively erasing the semantic closeness among them.

I also display the penultimate layer features for all the categories of CIFAR10 in Figure 5.8 and of Fashion MNIST in Figure 5.9. In the upper row, it is evident that clusters formed by 1-hot targets are dispersed, while those generated by Label

Smoothing and Learnable Label Smoothing result in more cohesive and compact clusters. Moving to the second row, I delve into illustrating inter-category relationships by examining distances among cluster centers of the training data. I employed L_1-normalized cosine distances, defined as $\frac{cd(i,j)}{\sum_j cd(i,j)}$, where $cd(i,j) = 1 - \frac{c_i \cdot c_j}{||c_i|| \cdot ||c_j||}$, and c_i and c_j represent the cluster centers of categories i and j, respectively. Notably, Label Smoothing disrupts the inter-category relationship, rendering all categories equidistant from each other in feature space. In contrast, both 1-hot and Learnable Label Smoothing maintain the inter-category relationship.

These comparisons highlight that clusters formed by 1-Hot targets demonstrate dispersion, while those formed by label smoothing and learnable label smoothing exhibit a more cohesive and compact nature. Notably, label smoothing consistently disrupts inter-class relationships, equidistantly positioning classes within the feature space—a salient observation underscored in my findings. In contrast, both One-hot encoding and learnable label smoothing methods consistently uphold and sustain inter-class relationships effectively. Significantly, learnable label smoothing emerges clearly superior by showcasing the strengths of both methods.

5.3.3 Varying hyperparameter Alpha

I show outcomes obtained by varying the values of hyperparameter α (0.05, 0.1, 0.2, 0.3, 0.4, and 0.5) using a ResNet18 on CIFAR100, Flowers-102, and CUB-200 datasets in Figure 5.10. My results indicate that the range $\alpha \in (0.1, 0.4)$ consistently delivers the optimal performance across these datasets.

5.3.4 Coefficient of Variation

I calculated the average coefficient of variation for the Q-Matrix across various networks and initializations as $\frac{1}{K^2} \sum_{i,j} \frac{std_{ij}}{mean_{ij}} * 100$. Here, $mean_{ij}$ and std_{ij} represent

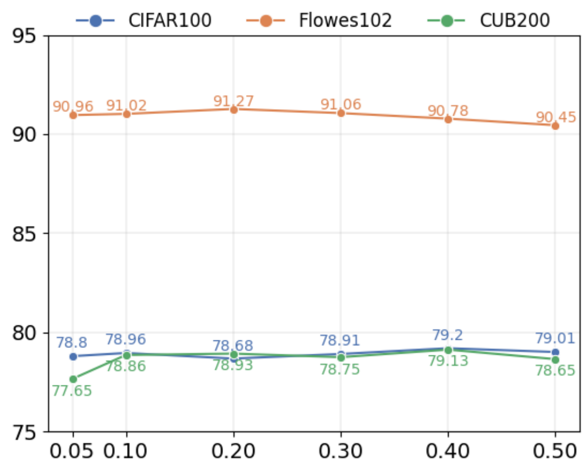


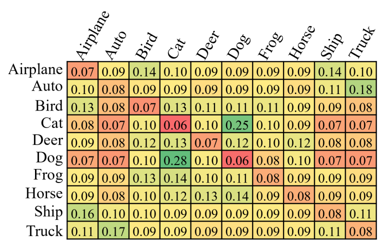
Figure 5.10: Results on varying α on CIFAR100, Flowers-102, and, CUB200 dataset with ResNet-18. It can be observed that $\alpha \in (0.1, 0.4)$ provides the best overall performance.

Dataset	CIFAR100	CIFAR10	SVHN	Flowers-102	CUB200
Initialization	0.18%	0.44%	1.24%	0.02%	0.04%
Network	0.79%	2.69%	6.39%	0.06%	0.08%

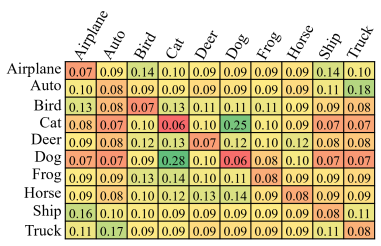
Table 5.6: Average Coefficient of Variation ($\frac{Std}{Mean} * 100$) across 6 initialization and more than 4 different networks.

the mean and the standard deviation of the i, j-th position across the Q-matrices respectively. This measure is independent of magnitude and illustrates the percentage change of a value relative to its mean.

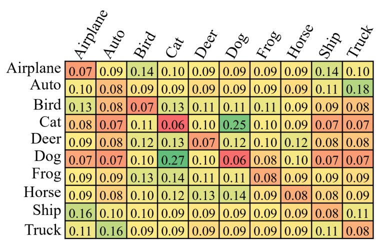
For my seed variation experiments, ResNet18 was employed across all datasets, utilizing initial seed values set to 0, 1, 2, 3, 4, and 5. In my network experiments, distinct architectures were utilized for various datasets that can be found below. () denotes the count of the networks used.



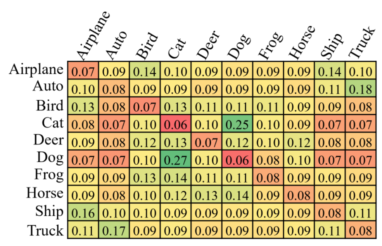
(a) Seed: 0



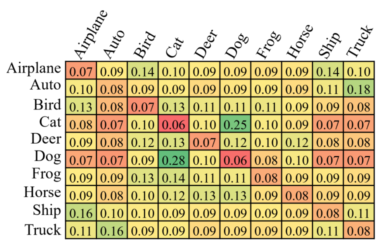
(b) Seed: 1



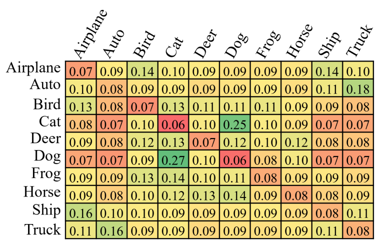
(c) Seed: 2



(d) Seed: 3

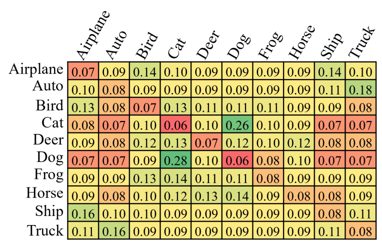


(e) Seed: 4

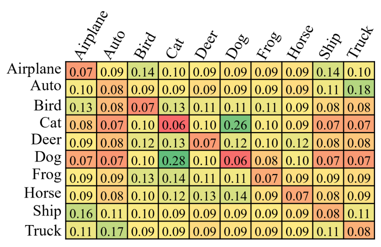


(f) Seed: 5

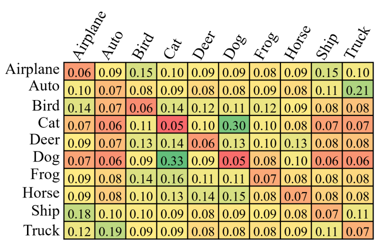
Figure 5.11: Learned Q-Matrices on CIFAR10 with different starting seeds.



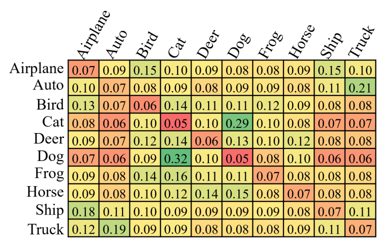
(a) ResNet18



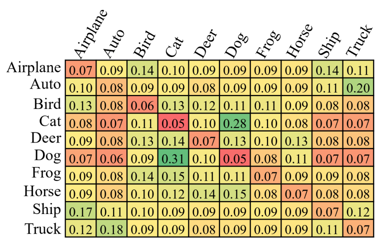
(b) ResNet34



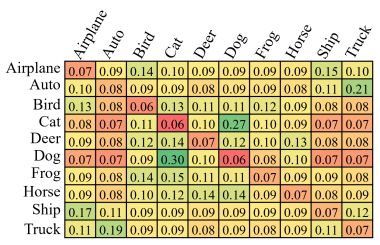
(c) ResNet50



(d) ResNet101



(e) DenseNet121



(f) GoogLeNet

Figure 5.12: Learned Q-Matrices on CIFAR10 with different network architectures.

- **CIFAR10** (6): ResNet18, ResNet34, ResNet50, ResNet101, DenseNet121, GoogleNet
- **SVHN** (4): ResNet18, ResNet50, DenseNet121, GoogleNet
- **CIFAR100** (7): ResNet18, ResNet34, ResNet50, ResNet56, ResNet101, DenseNet121, GoogleNet
- **Flowers-102** (4): ResNet18, ResNet50, ResNet101, MobileNetV2
- **CUB200** (4): ResNet18, ResNet50, ResNet101, MobileNetV2

Furthermore, in Figure 5.11 and 5.12, I present the learned Q-Matrices on CIFAR10 from different seeds and networks experiments respectively.

The findings are presented in Table 5.6, obtained by employing six different initializations and examining a minimum of four networks for each result. The results illustrate that the converged Q-Matrix demonstrates minimal variation across various initializations, displaying slight discrepancies among networks. This characteristic leads me to anticipate that the converged Q-Matrices of a new network will closely resemble the learned Q-matrix. Consequently, there arises an opportunity to employ the learned Q-Matrix in place of undertaking the process of learning a new one or transferring to a different network. Note that in the case of SVHN, there is a comparatively higher degree of changes in the Q-Matrix. I attribute this phenomenon to the absence of notable inter-category relationships in the dataset, resulting in some level of randomness.

Dataset	CIFAR100			Tiny-ImageNet		ImageNet-100
Teacher	R34→R18	R34→R34	R34→R50	R50 →R18	R101→R18	R50→R18
1-Hot	78.67	79.09	80.83	63.76	63.93	83.44
LS	79.40	80.15	81.15	64.31	64.02	83.32
LLS	**79.66**	**80.19**	**81.26**	**65.69**	**66.11**	**83.62**
LLS-ST	79.57	79.66	81.24	63.79	64.09	82.50

Table 5.7: Knowledge Distillation experiments on CIFAR100, Tiny ImageNet and ImageNet-100 datasets.. RX: ResNet-X. $Y \rightarrow Z$ denotes distillation from Y (Teacher) to Z (Student). The rows labeled 1-Hot, LS, and LLS correspond to scenarios where the Teacher network was trained using 1-hot encoding, Label Smoothing, and Learnable Label Smoothing, respectively. For LLS-ST (Learnable Label Smoothing-Substitute Teacher), only the learned Q-Matrix from the LLS Teacher network is used for distillation.

5.3.5 *Substitute Teacher for Knowledge Distillation*

Knowledge distillation employs a pre-trained teacher model f_t on the dataset to instruct the student model f_s. The teacher model generates targets for each sample, which the student model then uses to learn. The training loss for the student network is defined as:

$$\mathcal{L}_{KD} = \beta H(f_s(x), y) + (1 - \beta) H(f_s(x)/T, f_t(x)/T) \tag{5.18}$$

Here, H represents the cross-entropy loss, β is a parameter balancing the use of one-hot labels and teacher targets, and T is the temperature-regulating knowledge transfer from teacher to student. Across all datasets, I adopted their original training setup,

Dataset	CUB200		Flowers-102	
Teacher	R101→MV2	R101→R50	R101→MV2	R101→R50
1-Hot	78.82	81.57	91.64	92.00
LS	79.70	82.91	92.44	92.86
LLS	**80.15**	**83.38**	**92.63**	**93.40**
LLS-ST	79.62	83.02	92.49	93.14

Table 5.8: Knowledge Distillation experiments on fine-grain classification datasets: CUB200 and Flowers-102. RX: ResNet-X and M2: MobileNetV2. $Y \rightarrow Z$ denotes distillation from Y (Teacher) to Z (Student). The rows labeled 1-Hot, LS, and LLS correspond to scenarios where the Teacher network was trained using 1-hot encoding, Label Smoothing, and Learnable Label Smoothing, respectively. For LLS-ST (Learnable Label Smoothing-Substitute Teacher), only the learned Q-Matrix from the LLS Teacher network is used for distillation.

as described in Section 5.2.1, for knowledge distillation but altered the training loss function. Following the recent setup of knowledge distillation experiments, I set $\beta = 0$, implying that student networks are exclusively trained using teacher predictions, and used a temperature of 1 for all experiments.

However, the availability of a Teacher network can be constrained by computational or privacy considerations. In such scenarios, the Q-matrix of the Teacher network can serve as a substitute Teacher for Knowledge Distillation, denoted as LLS-ST. While a Teacher model furnishes targets on a per-sample basis, LLS-ST exclusively offers category-wise targets only.

The results for these experiments are presented in Table 5.7 and 5.8. The outcomes indicate that networks trained with LLS exhibit superior teaching capabilities during

	ImageNet-100	Tiny-ImageNet	Fashion MNIST	CIFAR100
1-Hot	77.22	54.26	86.49	73.70
LS	78.62	54.70	87.01	74.56
LLS	78.66	54.85	87.13	74.75
LLS-ST	**79.0**2	**55.21**	**87.56**	**74.92**

Table 5.9: The comparison between applying the learned Q-Matrix from the full data vs. employing 1-hot encoding, Label Smoothing, and Learnable Label Smoothing on sample-wise subsets. The results demonstrate a significant boost in generalization when the learned Q-Matrix is applied to the sample-wise subsets.

the distillation process. Remarkably, LLS-ST, despite imparting limited knowledge, imparts a performance boost comparable to employing a fully trained Teacher network.

5.3.6 Effectiveness on Subsets of Data

It's expected that the Q-Matrix is predominantly shaped by the characteristics of the training data, and any alterations to the training dataset consequently influence the learned Q-Matrix. However, once the Q-Matrix has been acquired, it remains applicable to both its category-wise and sample-wise subsets.

In this experiment, I meticulously examine these two types of subsets: (1) selecting the first 50% of categories and (2) randomly choosing 50% of samples from ImageNet-100, Tiny-ImageNet, FashionMNIST, and CIFAR100 datasets. Employing these data subsets, I train a ResNet-18 with 1-hot targets, Label Smoothing, and Learnable Label Smoothing as baselines. Subsequently, I delve into the impact of applying the learned Q-Matrix from the entire dataset, similar to the substitute Teacher experiments (LLS-

	ImageNet-100	Tiny-ImageNet	Fashion MNIST	CIFAR100
1-Hot	76.68	66.90	89.86	83.24
LS	77.88	66.98	90.56	83.56
LLS	**78.56**	**67.48**	**91.12**	**83.66**
LLS-ST	78.20	67.08	90.96	83.64

Table 5.10: The comparison between applying the learned Q-Matrix from the full data vs. employing 1-hot encoding, Label Smoothing, and Learnable Label Smoothing on class-wise subsets. When working with a restricted set of categories, acquiring a new Q-Matrix results in superior performance. However, the learned matrix demonstrates a close alignment with the performance of a freshly trained Q-matrix.

ST). For category subsets, I extract the logits corresponding to the selected categories from the Q-Matrix and exclusively apply Softmax to these chosen values.

The results of these experiments are detailed in Table 5.9 and 5.10. Notably, the learned Q-Matrix exhibits superior performance when applied to a subset of samples. When dealing with a subset of categories, learning a new Q-Matrix enhances generalization, with the learned matrix closely approaching the performance of a newly trained matrix, outperforming 1-hot targets and Label Smoothing.

5.3.7 Computation overhead of LLS

I examine the computation overhead of learnable label smoothing. LLS adds K^2 extra parameters which scales quadrically with the number of classes. Hence, I use the Tiny-ImageNet dataset as it has the highest number of classes (200) in my experiments. The total number of trainable parameters and training time with ResNet-18 and ResNet-101 are in table 5.11. As per the results, Learnable Label

	ResNet-18		ResNet-101	
	Parameters	Time (mins)	Parameters	Time (mins)
1-hot	11,578,632	142	44,131,080	674
LS	11,578,632	142	44,131,080	674
LLS	11,618,632 (0.3%↑)	146 (1.4%↑)	44,171,080 (0.1%↑)	680 (0.89%↑)

Table 5.11: Comparison of the number of training parameters and training time on Tiny-ImageNet with ResNet-18 and ResNet-101. Learnable Label Smoothing adds less than 0.3% parameters and increases training time by about 1%.

Smoothing adds less than 0.3% parameters in both cases and increases training time by about 1%.

Chapter 6

CONCLUSION

Labels are important for training a neural network. However, due to the cost of labeling the data, approaches that can use train a neural network with limited available labels are preferred. In this dissertation, I presented the 4 ways to reduce the cost of labeling: (1). Domain Adaptation which transfers knowledge from a similar dataset. (2). Self-supervised learning learns a rich set of features with an auxiliary task. (3). Regularizing neural network using input regularization (4). Regularizing neural network by label regularization. These methods can be employed across various label availability scenarios for `Making the Best of What We Have`.

6.1 Domain Adaptation

I presented three types of novel domain adaptation methods that align the domains at pixel, feature, or output levels.

6.1.1 Feature-level Adaptation

I discussed the Glocal domain alignment technique that does a salient modification to global alignment loss functions. Glocal alignment uses the most confident target pseudo labels and aligns individual categories which in turn improves global alignment. Through extensive experiments on various small and large datasets, I showcased the strength of the Glocal alignment. In all the comparisons, Glocal alignment results in superior performance compared to Global adversarial alignment.

6.1.2 Pixel-level Adaptation

Pixel-based unsupervised domain adaptation approaches are fascinating as their domain adaptation process can be visualized. Most of these approaches require two sets of generators, classifiers, and discriminators. In this dissertation, I present an iterative approach that trains a classifier and image generator in tandem and keeps the size of the model to a minimum. My approach can accurately translate a source image to a target domain while keeping the content preserved and classifies the source and target domain using a single classifier. Its three-way feature alignment ensures that the deep features are domain-invariant. The combined pixel and feature-level alignment establish a successful adaptation to the target domain. The translated images are of superior quality and my approach outperforms existing pixel and feature-level adaptation approaches for digits and traffic signs datasets.

6.1.3 Output-level Adaptation

I present a model for the Generative Alignment of Posterior probabilities (GAP) for source-free domain adaptation. GAP uses a Source Replicator (probability generator) that mimics the variations in the posterior probabilities of the source classes and then aligns target posterior probabilities to it through adversarial alignment. Through extensive experiments, I showcased that the approach is robust to smaller batch sizes, does not introduce any new hyper-parameters, and yields comparable performance to the compared baselines for source-free and source-present domain adaptation scenarios.

6.2 Self-supervised learning

I discussed PatchRot which is an easy-to-understand and implement self-supervised technique. It trains a Vision transformer to predict rotation angles $\in \{0\,°, 90°, 180°, 270°\}$ of image and image patches. With extensive experiments on multiple datasets, I showcased that a Vision transformer pretrained with PatchRot achieves superior results on downstream supervised learning. My approach is robust to overfitting and benefits from longer pretraining. It also works on rotation-invariant objects, can handle noise in the labels of the supervised task, and performs fine-grained classification. The features learned by PatchRot are generalizable and effective for various settings like supervised learning, transfer learning, and semi-supervised learning.

6.3 Input Regularization

I presented the PatchSwap technique for regularizing Vision Transformers. My approach swaps image patches between two images to create a regularized input for training. Also, it can be further extended to Unsupervised PatchSwap for semi-supervised applications by applying consistency regularization on two PatchSwap images. I also presented Inverse PatchSwap which can be used for complete self-supervised training. Different versions of PatchSwap can be used for different scenarios. Through extensive experiments, I showcase the strength of PatchSwap and its variants over existing techniques on various datasets.

6.4 Label Regularization

I introduce an innovative label regularization technique named ***L****earnable* ***L****abel* ***S****moothing* (LLS). This approach focuses on empowering networks to learn optimal target labels for regularization. Consequently, my method effectively produces compact feature clusters while preserving the inter-category relationships. Furthermore, the acquired understanding of these inter-category relationships is transferable, aiding in Knowledge Distillation even in scenarios where a Teacher network is unavailable. I believe Learnable Label Smoothing will play a transformative role in knowledge transfer paradigms for neural networks.

REFERENCES

Ahmed, S. M., D. S. Raychaudhuri, S. Paul, S. Oymak and A. K. Roy-Chowdhury, "Unsupervised multi-source domain adaptation without access to source data", in "Proceedings of the IEEE/CVF conference on computer vision and pattern recognition", pp. 10103–10112 (2021).

Arnab, A., M. Dehghani, G. Heigold, C. Sun, M. Lucic and C. Schmid, "Vivit: A video vision transformer", in "Proceedings of the IEEE/CVF International Conference on Computer Vision", pp. 6836–6846 (2021).

Ben-David, S., J. Blitzer, K. Crammer, A. Kulesza, F. Pereira and J. W. Vaughan, "A theory of learning from different domains", Machine learning **79**, 1-2, 151–175 (2010).

Bousmalis, K., N. Silberman, D. Dohan, D. Erhan and D. Krishnan, "Unsupervised pixel-level domain adaptation with generative adversarial networks", in "CVPR", pp. 3722–3731 (2017).

Chapelle, O., B. Scholkopf and A. Zien, "Semi-supervised learning. 2006", Cambridge, Massachusettes: The MIT Press View Article (2006).

Chen, X., S. Wang, M. Long and J. Wang, "Transferability vs. discriminability: Batch spectral penalization for adversarial domain adaptation", in "International Conference on Machine Learning", pp. 1081–1090 (2019).

Chhabra, S., P. B. Dutta, B. Li and H. Venkateswara, "Glocal alignment for unsupervised domain adaptation", in "Multimedia Understanding with Less Labeling on Multimedia Understanding with Less Labeling", pp. 45–51 (2021a).

Chhabra, S., P. B. Dutta, H. Venkateswara and B. Li, "Patchrot: A self-supervised technique for training vision transformers", arXiv preprint arXiv:2210.15722 (2022a).

Chhabra, S., H. Venkateswara and B. Li, "Iterative image translation for unsupervised domain adaptation", in "1st Workshop on Multimedia Understanding with Less Labeling, MULL 2021, co-located with ACM MM 2021", pp. 37–44 (Association for Computing Machinery, Inc, 2021b).

Chhabra, S., H. Venkateswara and B. Li, "Patchswap: A regularization technique for vision transformers.", in "BMVC", p. 996 (2022b).

Chhabra, S., H. Venkateswara and B. Li, "Generative alignment of posterior probabilities for source-free domain adaptation", in "Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision", pp. 4125–4134 (2023).

Cubuk, E. D., B. Zoph, J. Shlens and Q. V. Le, "Randaugment: Practical automated data augmentation with a reduced search space", in "Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops", pp. 702–703 (2020).

Deng, J., W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database”, in “2009 IEEE conference on computer vision and pattern recognition”, pp. 248–255 (Ieee, 2009).

Devries, T. and G. W. Taylor, “Improved regularization of convolutional neural networks with cutout”, CoRR **abs/1708.04552**, URL `http://arxiv.org/abs/1708.04552` (2017).

Dosovitskiy, A., L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, “An image is worth 16x16 words: Transformers for image recognition at scale”, in “International Conference on Learning Representations”, (2020).

Feng, Z., C. Xu and D. Tao, “Self-supervised representation learning by rotation feature decoupling”, in “Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition”, pp. 10364–10374 (2019).

Ganin, Y., E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand and V. Lempitsky, “Domain-adversarial training of neural networks”, The journal of machine learning research **17**, 1, 2096–2030 (2016a).

Ganin, Y., E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand and V. Lempitsky, “Domain-adversarial training of neural networks”, The Journal of Machine Learning Research **17**, 1, 2096–2030 (2016b).

Gidaris, S., P. Singh and N. Komodakis, “Unsupervised representation learning by predicting image rotations”, in “International Conference on Learning Representations”, (2018).

Gong, C., D. Wang, M. Li, V. Chandra and Q. Liu, “Keepaugment: A simple information-preserving data augmentation approach”, in “Proceedings of the IEEE/CVF conference on computer vision and pattern recognition”, pp. 1055–1064 (2021).

Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, “Generative adversarial nets”, in “Advances in neural information processing systems”, pp. 2672–2680 (2014).

Hanson, S. and L. Pratt, “Comparing biases for minimal network construction with back-propagation”, Advances in neural information processing systems **1** (1988).

Hassani, A., S. Walton, N. Shah, A. Abuduweili, J. Li and H. Shi, “Escaping the big data paradigm with compact transformers”, arXiv preprint arXiv:2104.05704 (2021a).

Hassani, A., S. Walton, N. Shah, A. Abuduweili, J. Li and H. Shi, “Escaping the big data paradigm with compact transformers”, CoRR **abs/2104.05704**, URL `https://arxiv.org/abs/2104.05704` (2021b).

He, K., X. Chen, S. Xie, Y. Li, P. Dollár and R. Girshick, "Masked autoencoders are scalable vision learners", in "Proceedings of the IEEE/CVF conference on computer vision and pattern recognition", pp. 16000–16009 (2022).

Hendrycks, D., N. Mu, E. D. Cubuk, B. Zoph, J. Gilmer and B. Lakshminarayanan, "Augmix: A simple data processing method to improve robustness and uncertainty", in "8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020", (OpenReview.net, 2020), URL `https://openreview.net/forum?id=S1gmrxHFvB`.

Hinton, G. E., O. Vinyals and J. Dean, "Distilling the knowledge in a neural network", CoRR **abs/1503.02531** (2015).

Hoffman, J., E. Tzeng, T. Park, J.-Y. Zhu, P. Isola, K. Saenko, A. Efros and T. Darrell, "Cycada: Cycle-consistent adversarial domain adaptation", in "International conference on machine learning", pp. 1989–1998 (Pmlr, 2018).

Houben, S., J. Stallkamp, J. Salmen, M. Schlipsing and C. Igel, "Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark", in "International Joint Conference on Neural Networks", (2013).

Hull, J. J., "A database for handwritten text recognition research", IEEE Transactions on pattern analysis and machine intelligence **16**, 5, 550–554 (1994).

Ioffe, S. and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", in "International conference on machine learning", pp. 448–456 (PMLR, 2015).

Ishii, M. and M. Sugiyama, "Source-free domain adaptation via distributional alignment by matching batch normalization statistics", CoRR **abs/2101.10842**, URL `https://arxiv.org/abs/2101.10842` (2021).

Isola, P., J.-Y. Zhu, T. Zhou and A. A. Efros, "Image-to-image translation with conditional adversarial networks", in "CVPR", pp. 1125–1134 (2017).

Kim, Y., D. Cho, P. Panda and S. Hong, "Progressive domain adaptation from a source pre-trained model", arXiv preprint arXiv:2007.01524 (2020).

Krizhevsky, A., G. Hinton *et al.*, "Learning multiple layers of features from tiny images", (2009).

Kurmi, V. K., V. K. Subramanian and V. P. Namboodiri, "Domain impression: A source data free domain adaptation method", in "Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision", pp. 615–625 (2021).

Laine, S. and T. Aila, "Temporal ensembling for semi-supervised learning", arXiv preprint arXiv:1610.02242 (2016).

LeCun, Y., L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition", Proceedings of the IEEE **86**, 11, 2278–2324 (1998).

Lee, D.-H., “Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks”, in “Workshop on challenges in representation learning, ICML”, vol. 3, p. 2 (2013).

Li, N., S. Liu, Y. Liu, S. Zhao and M. Liu, “Neural speech synthesis with transformer network”, in “Proceedings of the AAAI Conference on Artificial Intelligence”, vol. 33, pp. 6706–6713 (2019).

Liang, J., D. Hu and J. Feng, “Do we really need to access the source data? source hypothesis transfer for unsupervised domain adaptation”, in “International conference on machine learning”, pp. 6028–6039 (PMLR, 2020).

Lienen, J. and E. Hüllermeier, “From label smoothing to label relaxation”, in “Proceedings of the AAAI conference on artificial intelligence”, vol. 35, pp. 8583–8591 (2021).

Lin, T.-Y., P. Goyal, R. Girshick, K. He and P. Dollár, “Focal loss for dense object detection”, in “Proceedings of the IEEE international conference on computer vision”, pp. 2980–2988 (2017).

Long, M., Y. Cao, J. Wang and M. Jordan, “Learning transferable features with deep adaptation networks”, in “International conference on machine learning”, pp. 97–105 (2015).

Maaten, L. v. d. and G. Hinton, “Visualizing data using t-sne”, Journal of machine learning research **9**, Nov, 2579–2605 (2008).

Mao, X., Q. Li, H. Xie, R. Y. Lau, Z. Wang and S. Paul Smolley, “Least squares generative adversarial networks”, in “ICCV”, pp. 2794–2802 (2017).

Mirza, M. and S. Osindero, “Conditional generative adversarial nets”, CoRR **abs/1411.1784**, URL `http://arxiv.org/abs/1411.1784` (2014).

Miyato, T., S.-i. Maeda, M. Koyama and S. Ishii, “Virtual adversarial training: a regularization method for supervised and semi-supervised learning”, IEEE transactions on pattern analysis and machine intelligence **41**, 8, 1979–1993 (2018).

Mukhoti, J., V. Kulharia, A. Sanyal, S. Golodetz, P. Torr and P. Dokania, “Calibrating deep neural networks using focal loss”, Advances in Neural Information Processing Systems **33**, 15288–15299 (2020).

Müller, R., S. Kornblith and G. E. Hinton, “When does label smoothing help?”, Advances in neural information processing systems **32** (2019).

Murez, Z., S. Kolouri, D. Kriegman, R. Ramamoorthi and K. Kim, “Image to image translation for domain adaptation”, in “CVPR”, pp. 4500–4509 (2018).

Netzer, Y., T. Wang, A. Coates, A. Bissacco, B. Wu and A. Y. Ng, “Reading digits in natural images with unsupervised feature learning”, (2011).

Nilsback, M.-E. and A. Zisserman, "Automated flower classification over a large number of classes", in "Indian Conference on Computer Vision, Graphics and Image Processing", (2008).

Noroozi, M. and P. Favaro, "Unsupervised learning of visual representations by solving jigsaw puzzles", in "European conference on computer vision", pp. 69–84 (Springer, 2016).

Pei, Z., Z. Cao, M. Long and J. Wang, "Multi-adversarial domain adaptation", in "Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018", edited by S. A. McIlraith and K. Q. Weinberger, pp. 3934–3941 (AAAI Press, 2018).

Peng, X., B. Usman, N. Kaushik, J. Hoffman, D. Wang and K. Saenko, "Visda: The visual domain adaptation challenge", CoRR **abs/1710.06924** (2017).

Popovič, A., "The concept "shift of expression" in translation analysis", in "The nature of translation", pp. 78–88 (De Gruyter Mouton, 2011).

Ruder, S., "An overview of multi-task learning in deep neural networks", CoRR **abs/1706.05098**, URL `http://arxiv.org/abs/1706.05098` (2017).

Russo, P., F. M. Carlucci, T. Tommasi and B. Caputo, "From source to target and back: Symmetric bi-directional adaptive GAN", in "CVPR", pp. 8099–8108 (2018).

Saenko, K., B. Kulis, M. Fritz and T. Darrell, "Adapting visual category models to new domains", in "European conference on computer vision", pp. 213–226 (Springer, 2010).

Sajjadi, M., M. Javanmardi and T. Tasdizen, "Regularization with stochastic transformations and perturbations for deep semi-supervised learning", NeurIPS **29**, 1163–1171 (2016).

Selvaraju, R. R., M. Cogswell, A. Das, R. Vedantam, D. Parikh and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization", in "Proceedings of the IEEE international conference on computer vision", pp. 618–626 (2017).

Shen, J., Y. Qu, W. Zhang and Y. Yu, "Wasserstein distance guided representation learning for domain adaptation", in "AAAI", (2018).

Shu, R., H. Bui, H. Narui and S. Ermon, "A dirt-t approach to unsupervised domain adaptation", in "International Conference on Learning Representations", (2018).

Song, H., M. Kim and J.-G. Lee, "Selfie: Refurbishing unclean samples for robust deep learning", in "International Conference on Machine Learning", pp. 5907–5915 (PMLR, 2019).

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting", The journal of machine learning research **15**, 1, 1929–1958 (2014).

Szegedy, C., V. Vanhoucke, S. Ioffe, J. Shlens and Z. Wojna, "Rethinking the inception architecture for computer vision", in "Proceedings of the IEEE conference on computer vision and pattern recognition", pp. 2818–2826 (2016a).

Szegedy, C., V. Vanhoucke, S. Ioffe, J. Shlens and Z. Wojna, "Rethinking the inception architecture for computer vision", in "Proceedings of the IEEE conference on computer vision and pattern recognition", pp. 2818–2826 (2016b).

Taigman, Y., A. Polyak and L. Wolf, "Unsupervised cross-domain image generation", arXiv preprint arXiv:1611.02200 (2016).

Tarvainen, A. and H. Valpola, "Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results", Advances in neural information processing systems **30** (2017).

Touvron, H., M. Cord, M. Douze, F. Massa, A. Sablayrolles and H. Jégou, "Training data-efficient image transformers & distillation through attention", in "International conference on machine learning", pp. 10347–10357 (PMLR, 2021).

Touvron, H., A. Vedaldi, M. Douze and H. Jégou, "Fixing the train-test resolution discrepancy", Advances in neural information processing systems **32** (2019).

Tzeng, E., J. Hoffman, T. Darrell and K. Saenko, "Simultaneous deep transfer across domains and tasks", in "Proceedings of the IEEE International Conference on Computer Vision", pp. 4068–4076 (2015).

Tzeng, E., J. Hoffman, K. Saenko and T. Darrell, "Adversarial discriminative domain adaptation", in "CVPR", pp. 7167–7176 (2017).

Tzeng, E., J. Hoffman, N. Zhang, K. Saenko and T. Darrell, "Deep domain confusion: Maximizing for domain invariance", CoRR **abs/1412.3474** (2014).

Van der Maaten, L. and G. Hinton, "Visualizing data using t-sne.", Journal of machine learning research **9**, 11 (2008).

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser and I. Polosukhin, "Attention is all you need", Advances in neural information processing systems **30** (2017).

Venkateswara, H., J. Eusebio, S. Chakraborty and S. Panchanathan, "Deep hashing network for unsupervised domain adaptation", in "Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition", pp. 5018–5027 (2017).

Vorburger, P. and A. Bernstein, "Entropy-based concept shift detection", in "Sixth International Conference on Data Mining (ICDM'06)", pp. 1113–1118 (IEEE, 2006).

Wah, C., S. Branson, P. Welinder, P. Perona and S. Belongie, "The caltech-ucsd birds-200-2011 dataset", (2011).

Wang, X., L. Li, W. Ye, M. Long and J. Wang, "Transferable attention for domain adaptation", in "Proceedings of the AAAI Conference on Artificial Intelligence", vol. 33, pp. 5345–5352 (2019a).

Wang, Y., X. Ma, Z. Chen, Y. Luo, J. Yi and J. Bailey, "Symmetric cross entropy for robust learning with noisy labels", in "Proceedings of the IEEE/CVF international conference on computer vision", pp. 322–330 (2019b).

Xiao, H., K. Rasul and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms", (2017a).

Xiao, H., K. Rasul and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms", CoRR **abs/1708.07747**, URL `http://arxiv.org/abs/1708.07747` (2017b).

Yuan, L., F. E. Tay, G. Li, T. Wang and J. Feng, "Revisiting knowledge distillation via label smoothing regularization", in "Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition", pp. 3903–3911 (2020).

Yun, S., D. Han, S. Chun, S. J. Oh, Y. Yoo and J. Choe, "Cutmix: Regularization strategy to train strong classifiers with localizable features", in "2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019", pp. 6022–6031 (IEEE, 2019), URL `https://doi.org/10.1109/ICCV.2019.00612`.

Zhai, X., A. Oliver, A. Kolesnikov and L. Beyer, "S4l: Self-supervised semi-supervised learning", in "Proceedings of the IEEE/CVF International Conference on Computer Vision", pp. 1476–1485 (2019).

Zhang, C.-B., P.-T. Jiang, Q. Hou, Y. Wei, Q. Han, Z. Li and M.-M. Cheng, "Delving deep into label smoothing", IEEE Transactions on Image Processing **30**, 5984–5996 (2021).

Zhang, H., M. Cissé, Y. N. Dauphin and D. Lopez-Paz, "mixup: Beyond empirical risk minimization", in "6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings", (OpenReview.net, 2018), URL `https://openreview.net/forum?id=r1Ddp1-Rb`.

Zhang, R., P. Isola and A. A. Efros, "Split-brain autoencoders: Unsupervised learning by cross-channel prediction", in "Proceedings of the IEEE conference on computer vision and pattern recognition", pp. 1058–1067 (2017).

Zhang, Y., H. Tang, K. Jia and M. Tan, "Domain-symmetric networks for adversarial domain adaptation", in "Proceedings of the IEEE/CVF conference on computer vision and pattern recognition", pp. 5031–5040 (2019).

Zhu, J.-Y., T. Park, P. Isola and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks", in "ICCV", pp. 2223–2232 (2017).

APPENDIX A

PERMISSION STATEMENTS FROM CO-AUTHORS

The inclusion of the content presented in this dissertation did not raise any objections from any of the co-authors: Dr. Baoxin Li, Dr. Hemanth Kumar Demakethepalli Venkateswara, and Prabal Bijoy Dutta.