Low-Resource Neural Adaptation:
A Unified Data Adaptation Framework for Neural Networks

Beibin Li

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2022

Reading Committee:

Linda Shapiro, Chair

Frederick Shic

Yao Lu

Program Authorized to Offer Degree:
Computer Science and Engineering

University of Washington

Abstract

Low-Resource Neural Adaptation:
A Unified Data Adaptation Framework for Neural Networks

Beibin Li

Chair of the Supervisory Committee:
Professor Linda Shapiro
Paul G. Allen School of Computer Science and Engineering

Many machine learning (ML) models are trained on specific datasets for specific tasks. While traditional transfer learning can adapt to new datasets when labeled data are adequate, adapting to small datasets is still a challenging task. Researchers have applied multi-task learning, meta-learning, weakly-supervised learning, self-supervision, generative adversarial training, and active learning for various data adaptation applications. However, a unified data adaptation framework has yet to be developed. This study proposes a unified framework that can adapt to small datasets in a dynamic environment. Our framework, with a versatile encoder and various decoders, can simultaneously learn from source datasets and estimate confidence for novel data samples. We apply the framework to real-world medical imaging, affective computing, eye-tracking analysis, and database management applications.

TABLE OF CONTENTS

LIST OF FIGURES

Figure Number Page

LIST OF TABLES

Table Number Page

LIST OF ALGORITHMS

GLOSSARY

NEURAL NETWORK: an artificial neural network with artificial neurons or nodes.

ADAPTATION: of a machine learning model aims to use a pre-trained model for a target dataset, target task, or target domain. For instance, probing, fine-tuning, re-training, and prompt-tuning are common adaptation methods.

DATA SHIFT: occurs when there is a change in the data distribution.

DOMAIN SHIFT: occurs when the data domains changes, e.g., from wildlife images to histopathological images.

LOW-RESOURCE: occurs when the amount of training data or training labels is scarce.

SUPERVISED LEARNING: a machine learning task of learning a function that maps an input to an output based on example input-output pairs.

UNSUPERVISED LEARNING: a machine learning task of learning patterns from unlabeled data.

SELF-SUPERVISED LEARNING: a special mixture of supervised and unsupervised learning, where the data provides the supervision. It is a machine learning task that combines a small amount of labeled data with a large amount of unlabeled data during learning.

SELF-ADAPTATION: a machine learning model that can automatically adapt to distribution changes through self-supervised learning.

ACKNOWLEDGMENTS

The author wishes to express sincere appreciation to the University of Washington, where he has had the opportunity to work with Linda Shapiro, Frederick Shic, Yao Lu, and Sheng Wang. Without their guidance, the author could not finish his Ph.D. journey. Thanks to Professor John Kramlich for serving as a Graduate School Representative.

The collaborations with Nicholas Nuechterlein, Sachin Mehta, Deepali Aneja, Kechun Liu, Wenjun Wu, Shima Nofallah, Ezgi Mercan, Fatemeh Ghezloo, Bindita Chaudhuri, Mehmet Saygin Seyfioglu, Shu Liang, Jie Gao, Kalyani Sunil Marathe, Ananditha Raghunath, Zixuan Liu, Wisdom Ikezogwo, Rajesh Rao, and other talented researchers enriched the author's academic studies.

The author would also like to thank Pamela Ventola, Katarzyna Chawarska, Quan Wang, Li Feng, Laura Boccanfuso, Brian Scassellati, Adam Naples, Carla Wall, Mary Best, Sarah Corrigan, Tanya John, Marilena Mademtzi, James C. McPartland, Raphael A. Bernier, Fred Volkmar, Stephanie Valencia, Erin Barney, Claire Foster, Minah Kim, Yeojin Amy Ahn, Madeline Aubertine, Kelsey Dommer, Monique Mahony, Caitlin Hudac, Adham Atyabi, James (JC) Snider, Logan Hart, Emily Hilton, Anna Milgramm, Max Rolison, Lauren Dinicola, Finola Kane-Grade, Iolanda Leite, Colette Torres, Lisa Chen, Patricia Pérez-Fuster, Nicole Salomons, and other members of Yale University and Seattle Children's Research Institute.

Medical guidance from Joann Elmore, Donald Weaver, Stevan Knezevich, Jamen Bartlett, Corey Arnold, Annie Lee, Jason Wang, Mojgan Mokhtari, Caitlin May, Oliver Chang, and Megan Eguchi was essential for the author's success in cancer research.

Collaborations with Yao Lu, Srikanth Kandula, Chi Wang, and Surajit Chaudhuri at

Microsoft Research accelerated the author's study in database systems.

In addition, we would like to thank Irati Saez de Urabain and Timothy J. Smith from the University of London.

Cancer studies reported in this document were supported by grants R01-CA172343, R01-CA140560, U01-CA231782, and R01-CA200690 from the National Cancer Institute of the National Institutes of Health.

Autism studies reported in this document were supported by National Institutes of Health K01-MH104739, R21-MH103550, R03-MH092618, the National Science Foundation Expedition in Socially Assistive Robotics #1139078, the Simons Award #383661, and the Autism Biomarkers Consortium for Clinical Trial.

We extend gratitude to all of the families and participants who participated in our studies.

The author also thanks Shun-Tak Leung and his support through the Faithful Steward Endowed Fellowship.

Without the support from my parents, Peng Li and Donghong Pei, I could not accomplish my Ph.D. journey.

Chapter 1
INTRODUCTION

Inspired by human brains and neuroscience, neural network-based machine learning (ML) models can tackle many challenging real-world problems, including machine translation, medical imaging analyses, screening for developmental risk, and database optimization. However, many neural networks are dataset-specific and struggle to generalize to novel data. Moreover, researchers often have limited resources for acquiring labeled data in medical diagnosis, signal processing, and other machine learning applications. This study proposes a unified framework to address the common low-resource learning and data shift problems when only limited data or labels are available.

1.1 Challenge: Data Scarcity

Deep learning models show a potential to outperform human beings in many tasks, but training a deep neural network requires abundant data. Pedro Domingos, a machine learning expert, once argued that "the closest thing there is to a free lunch in ML is more data" [1]; Sambasivan et al. [2] from Google also complained, "everyone wants to do the model work, not the data work." While we agree that adding more data is usually more effective than designing a better ML model, the reality of data acquisition can be cruel. Unfortunately, capturing enough labeled data for deep neural networks is unrealistic in some applications because of privacy concerns or high labeling costs. Training a melanoma biopsy diagnosis system would require researchers to collect and label protected health information (PHI) data from various hospitals. Developing a mobile screening application for children with autism spectrum disorder (ASD) would require clinicians to label thousands of video streams from children. Deploying a neural network-based query optimization model would require the database management system (DBMS) to label hundreds of queries in real-time. Labeling all data samples is infeasible in these scenarios. When data or labels are scarce, the machine learning system faces a "low-resource" challenge.

Fortunately, scientists have discovered this data-hunger limitation of deep learning models and have investigated this problem extensively in the past few years. To address the scarcity problem of training data and learn from small datasets, researchers have invented various algorithms, including meta-learning, few-shot learning, self-supervised learning, and generative methods. Recently, scientists found that "language models are few-shot learners" in the Generative Pre-trained Transformer v3 (aka GPT-3) [3]. Similarly, bidirectional encoder representations from Transformers (BERT) can transfer knowledge in language understanding to downstream applications [4].

While these promising methods could help machine learning models adapt to small datasets, these models still need to learn from an enormous corpus in the pre-training stage. So, data acquisition and model improvement should run in parallel, and researchers need an active pipeline.

1.2 Challenge: Data Shifting

On the other hand, if models need to run parallel with data collection, distribution and data shifts might happen: deploying face recognition models to a different population group or applying a hematoxylin and eosin (H&E) stained analysis method to another hospital with a different brand of scanner. The data shifting problem is particularly severe for small datasets because the lack of data would lead to poor generalizability of the machine learning model.

As we agree with Domingos' and Sambasivan's opinions above on the importance of data and labels, an intuitive way to resolve the data shift problem is to acquire more labels after deploying deep learning models to the real-world. When a handful of new data become available to the system, researchers should decide which data to label to maximize the labeling utility. For instance, labeling a "fear" or "surprise" face is more valuable than labeling a "happy" face in facial expression recognition applications, because current algorithms and models often fail to identify fear and surprise in facial images. Labeling "atypia" or "in situ" diagnoses is often more important than labeling invasive cancer because the diagnosis of "atypia" v.s. "in situ" could affect treatment plans for patients. Machine learning researchers have designed various algorithms for this active learning scenario to decide which data should be labeled first, but these methods are often independent of traditional data-shift applications.

Challenged by the severe distribution shift problem, many machine learning models deteriorate after deployment for real-world applications. Active learning, uncertainty sampling, and many other exploration-versus-exploitation algorithms step in to mitigate this problem. While these techniques are promising in simulations, more challenges exist in real-world applications. Topol [5] pointed out that most deep learning studies were published with *in silicon* validation, which uses computer-based validation and testing sets. These validation sets are usually from the same distribution as the training set, and this "Silicon Valley-dation" method [5] might be too optimistic for many real-world problems, such as medical diagnosis and behavior analyses. Hence, adapting trained models to novel data would be necessary for robust deep learning uses. Researchers still face the unsolved problem of limited annotation resources.

1.3 Definition: Low-Resource Learning

Although machine learning applications suffer from data scarcity and data shifting problems, the biological counterpart of artificial neural networks, actual human brains, can address these problems flexibly and efficiently. When facing new data in a specific problem, humans are usually aware of the data's novelty, can adjust their confidence, and actively learn from the new data later on. Moreover, human beings can plan, organize, and employ strategies to solve problems in new domains with only a few labeled data samples. These different cognitive shifting abilities are all related to the human prefrontal cortex, which inspired us to create a unified framework to solve different data-adaptation problems.

Based on the two challenges discussed above, we define the ***low-resource*** data adaptation problem as:

- how to train a neural network-based model from a set of partially labeled source datasets so that the model can quickly adapt to novel and small target datasets.

- how to determine the confidence of a model to a novel data sample, how to detect data shifts, and how to rank the labeling priority for unlabelled data so that higher-priority data can provide more utility if labeled.

The low-resource adaptation problem is similar to many meta-learning, few-shot learning, and semi-supervised learning problems. However, datasets are only partially labeled in our setting, and we assume that researchers can request a few additional labels if more labeling resources are available. This interactive setting makes the data adaptation process more challenging.

1.4 Solution: A Unified Framework – UDA

We propose a unified data adaptation (UDA) framework that addresses data shift in the low-resource learning setting. The framework can encode a data sample into an embedding vector, where its decoders can use this vector for self-adaptation, synthetic data generation, and confidence estimation. This framework will also utilize confidences to rank uncertain target data samples by their utility, so that human annotators can choose to label the highest utility samples when they have more labeling resources.

Previous studies have explored active learning, dataset shift, meta-learning, and transfer learning separately. Inspired by the biological prefrontal cortex in human brains, which can simultaneously perform these adaptive tasks, we connect these machine learning components with a single versatile encoder and a group of decoders. The main contribution of this study is not to create brand new methods for machine learning, but to unify existing yet separated machine learning tasks together so that future researchers can apply the proposed framework for their low-resource learning and adaptation applications.

Bridging the background knowledge in human brains and artificial neural networks, we first introduce information encoding in Chapter 2 and data adaptation in Chapter 3. Then, in Chapter 4, we propose our unified framework, UDA. With this framework, we tackle real-world problems. In the medical imaging application (Chapter 5), UDA adapts to a new type of label with only a few weakly-supervised annotations, where it outperforms previous diagnostic classification approaches. In the facial expression recognition application (Chapter 6), UDA learns from several large public datasets and then infers on unlabelled data from children. In eye-tracking applications (Chapter 7), in addition to adapting from several public datasets, UDA can predict its confidence on new participants and perform gaze calibration. In the database optimization application (Chapter 8), UDA detects data shifts, generates synthetic data samples, and quickly adapts to new data distributions. At last, after discussing limitations and future opportunities in Chapter 9, we conclude this work in Chapter 10.

Chapter 2

INFORMATION ENCODING

Many neural network-based models, from the 1950s Perceptron algorithm [6] to the 2020s large pre-trained foundation models [7], draw inspirations from biological neurons and human behaviors.

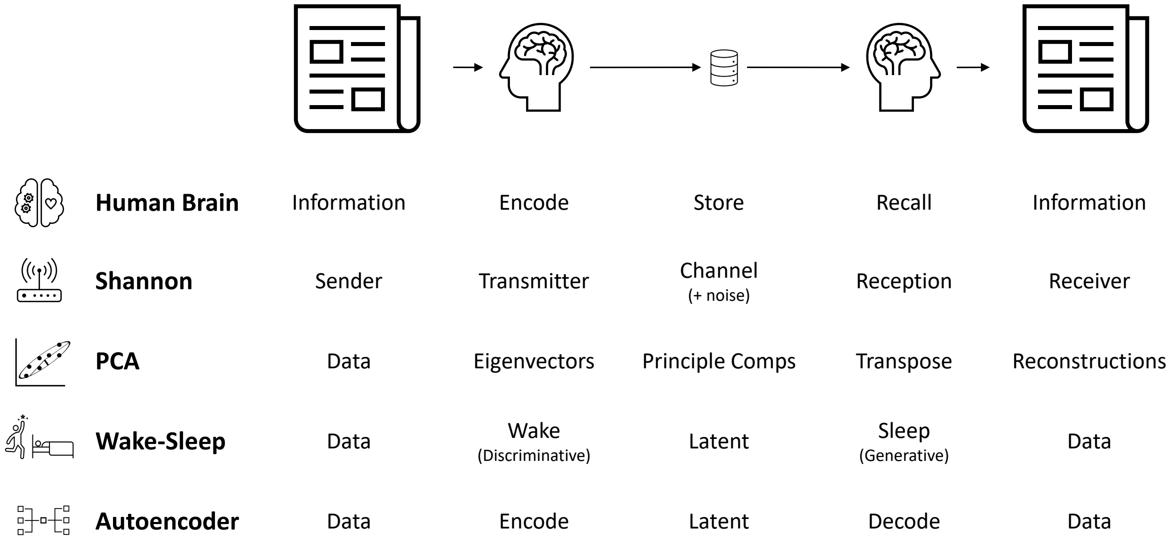
2.1 Information Encoding in the Human Brain

The academic concept of information encoding first appeared in the psychology community in the 1880s [8]. Modern cognitive psychology studies, such as [9], have shown that human brains can: (1) encode, store, and recall information; (2) learn and adapt from previous experiences; and (3) build representations of data relationships. In summary, the information encoding ability of *Homo sapiens* can convert perceived data to a construct and then restore the data within the brain later, as shown in Figure 2.1.

Numerous unknown mysteries exist in neuroscience research and deep learning studies. Nevertheless, current knowledge about human brains has already influenced the design of intelligent machines and algorithms. For instance, this memory encoding ability inspired the long-short term memory (LSTM) and the gated recurrent unit (GRU) for temporal data analyses. In the following sections, we focus on the impact of encoding theory in machines - signal processing and machine learning.

2.2 Information Encoding in Signal Processing

Defining the concept of information encoding for communications systems, Claude Shannon created mathematical tools for data compression and founded information theory [10]. As



Human Brain	Information	Encode	Store	Recall	Information
Shannon	Sender	Transmitter	Channel (+ noise)	Reception	Receiver
PCA	Data	Eigenvectors	Principle Comps	Transpose	Reconstructions
Wake-Sleep	Data	Wake (Discriminative)	Latent	Sleep (Generative)	Data
Autoencoder	Data	Encode	Latent	Decode	Data

Figure 2.1: Encoder-decoder design in different research domains. The first row shows memory encoding theory for human brain. The second row shows Shannon's general communication system [10]. The third rows shows equations for principal component analyses (PCA) for data dimensionality reduction [11, 12]. The fourth rows illustrates the Wake-Sleep algorithm [13] and the last rows shows the autoencoder design for unsupervised learning [14].

shown in the second row of Figure 2.1, Shannon's general communication model[1], which transmits information between two agents, follows the human brains' activities to encode, store, and recall memory.

Based on the communication model, Shannon further analyzed information entropy, $\sum_{i=1}^{n} p_i \log_2(p_i)$, to characterize the amount of information carried inside a data sample. In the formula, p_i is the probability that event i would occur from n possible outcomes. As Shannon claimed that "Information is the resolution of uncertainty," this information entropy value can measure uncertainties for a given sequence of data. Since the 1950s, researchers and experts have explored the concept of information entropy, both in the time domain and the

[1] The Shannon-Weaver model of communication, another variant of the Shannon's general communication model, is widely used in social science.

frequency domain, to analyze time-series data. These concepts still thrive in modern deep learning. For instance, as a non-linear loss function, cross-entropy can help train classifiers; by comparing two probability distributions, Jensen-Shannon divergence or Kullback-Leibler divergence can train generative models; by utilizing randomized algorithms, entropy-based sampling is a fundamental active learning tool.

Decades before Shannon's information theory, many researchers focused on the statistical perspective of dimension reduction. After inventing the famous product-moment correlation coefficient [15], Pearson created a method to find the closet fit for points in a space in 1901 [11]. Without knowing Pearson's work, Hotelling invented a similar mathematical tool, principal component analysis, to reduce data dimensionality in 1933 [12]. Although not inspired by neuroscience, this principal component analysis (PCA) strategy and the singular value decomposition (SVD) [16] algorithm are still similar to the brain's encoding process. These methods inspired researchers to combine neuroscience inspirations and mathematical tools for information encoding in complex neural networks.

2.3 Information Encoding in Artificial Neurons

The Perceptron algorithm [6], inspired by biological neurons, creates a sigmoid function with a simple neuron. Rosenblatt further developed the multi-layer perceptron (MLP) model [17] in 1961, which connects different layers of neurons together to form a neural network. However, the Perceptron and MLP methods failed to fulfill scientists' ambitions. At that time, nobody can efficiently train complex neural networks even if the MLP model can represent a complex decision process. More than two decades later, based on the chain rule and partial derivatives, Rumelhart, Hinton, and Williams [18, 19] created the backpropagation algorithm to train MLPs. With this effective algorithm, researchers investigated more complex models, such as convolutional neural networks (CNNs) and Transformers.

A decade later, the Helmholtz Machine and the Wake-Sleep algorithm [13], connecting human behavior and machines, use two neural networks to encode and decode information: during the wake phase, the discriminative network encodes data to a low-dimensional representation; during the sleep phase, the generative network restores the data with the compressed representation. Another decade later, Hinton et al. continued this idea and invented the autoencoder (aka, encoder-decoder) [14] for unsupervised learning. While the architecture is similar to the previous wake-sleep algorithm, its mathematical goal is to extend the PCA dimension reduction technique with non-linear higher-order functions.

Also inspired by the brain's encoding capability, Rao and Ballard [20] invented a predictive coding system for natural images. Their hierarchical predictive coding algorithm, which resembles "simple-cell-like" receptive fields, is consistent with the biological visual cortex. The predictive coding behavior of the human brain and neural network continuously influences modern researchers. Lotter, Kreiman, and Cox invented PredNet [21] in 2017 to perform predictive coding in videos. This strategy later inspired self-supervised learning and other unsupervised learning approaches in computer vision (e.g., [22, 23, 24, 25]). Recently, [26, 27] extended this self-supervised learning concept and improved vision transformers by a large margin.

2.4 Summary

Information encoding exists in both the human brain and artificial neural networks. The internal data representation in latent space could be essential for low-resource data adaptation. Inspired by the human brain and human behavior, self-supervised learning, foundation models, and other recent learning algorithms could mitigate the low-resource learning problems. The next chapter will discuss these methods in low-resource adaptation.

Chapter 3

LEARNING AND ADAPTING

With the information encoding knowledge introduced from the last chapter, we discuss learning and adapting skills in humans and machines here.

3.1 Adaptation in Humans: Executive Functioning Skills

Human executive function skills, including task shifting, inhibition of prepotent responses, and short-term memory, have inspired some recent deep learning approaches to address adaptation problems. For instance, educators often expose children to new environments and tasks to train children's cognitive flexibility [28], which inspired recent meta-learning [29] and few-shot learning [30] studies. Similarly, learning from multimodal information can help children adapting to new challenges [31], and this multimodal learning approach has been recently applied to analyze brain tumors [32], recognize activities [33], and understand visual scenes [34]. Studying humans' task shifting ability could benefit engineers designing flexible frameworks for data adaptation in machine learning applications.

In the last few years, we developed a mobile video game [35] to explore and quantify executive functioning skills in children with autism spectrum disorder (ASD), separately considering the social and the nonsocial performance to disentangle broader patterns of cognitive deficit specific to the disorder. We found that children with autism have different patterns in short-term memory and inhibitory response to angry facial expressions. We also modified and applied this mobile game to help patients with temporal lobe epilepsy from developing regions [36], and our results suggest that nocturnal temporal lobe interictal epilepsy is closely associated with short-term memory performance and visual attention. We used both standard psychological approaches and machine learning methods to analyze participants' behavior, and we found that executive functioning skills are closely related to developmental level and brain activity.

Researchers usually associate executive functioning skills with the prefrontal cortex, the front region of the frontal lobe in the human brain [37]. While the human prefrontal cortex has higher activation during creative problem-solving [38], this region is also related to memory, attention, and language generation. Moreover, the prefrontal cortex has implications for the generation of slow-wave sleep [39], which corresponds to the "sleep phase" (i.e., a generative pass) in the Helmholtz Machine [13] as mentioned in the previous chapter.

These facts inspire us to include memory, attention, and generative methods in our adaptation framework, because the prefrontal cortex can perform all these tasks. In the last decades, machine learning researchers have created artificial neural networks for these cognitive tasks (i.e., memory [40], attention [41], generation [42], and wake-sleep [13]), which are the cornerstones for our adaptation framework. The following sections focus on learning paradigms that involve deep learning models.

3.2 Adaptation in Machines:

The pervasive data shifting problem in machine learning applications could be intentional or unintentional. For instance, researchers can intentionally use and adapt an existing machine learning model for a novel problem (e.g., fine-tune a pneumonia model for COVID-19 diagnosis). A machine learning model trained on unintentionally biased data could fail during testing. Previous studies usually regard these data adaptation problems with different methodologies, including transfer learning, multi-task learning, domain adaptation, and active learning.

3.2.1 Transfer Learning: the Foundation of Adaptation

As the fundamental method for data adaptation, transfer learning assumes the feature extractor to be a stochastic learner, such as a logistic regressor, a support vector machine, or a neural network. During transfer learning, stochastic optimization algorithms, such as stochastic gradient descent [43, 44] or the Adam optimizer [45], would fine tune the pre-trained model on the target dataset. This learning strategy can transfer learned knowledge from massive source datasets to smaller target datasets, where the source and target dataset usually have relevant but still distinct data distributions. The word "transfer" is an umbrella term for various types of transferring methods, including supervised, unsupervised, inductive, and transductive. We refer readers to a comprehensive survey on transfer learning [46] for more examples.

The following sections introduce some well-known extensions of the transfer learning method. We note that these algorithms are usually closely related to each other, because their goals are similar – data adaptation. However, subtle differences among them drive researchers to perform radically different analyses in these subareas.

3.2.2 Multi-Task Learning and Foundation Models

Multi-task learning (MTL) applies transfer learning on multiple tasks, which can diversify source datasets and benefit the feature extractor. Researchers can train the machine learning model from one dataset, where each datum contains several labels, or from multiple datasets, where different datasets might contain different types of labels.

In recent years, researchers have applied the multi-task learning method from huge datasets to adapt the pre-trained models to various downstream applications. Nowadays, researchers call this type of models a "foundation model" [7], such as BERT [4] and GPT-3 [3]. A few months ago, researchers even trained foundation models with over a hundred tasks [47]. This multi-task learning method is also related to meta-learning, another method to learn from multiple datasets.

3.2.3 Meta-Learning and Few-shot learning

Inspired by the learning process of children, the meta-learning (aka, learning-to-learn) algorithm observes how other learners can learn novel tasks. As shown in Figure 3.1, unlike MTL that learns directly from multiple datasets, the meta learner needs to learn from different learners. Mathematically, this learning process yields a second-order optimization problem. Model-agnostic meta learner (MAML) [48] is a well-known model that can perform second-order optimization by splitting training datasets into training and validation, where the first order of learning is from the sub-training set, and the second-order of learning is from the sub-testing set. Besides MAML, researchers invented many other efficient algorithms in the past five years, and we will discuss them in the following chapter.

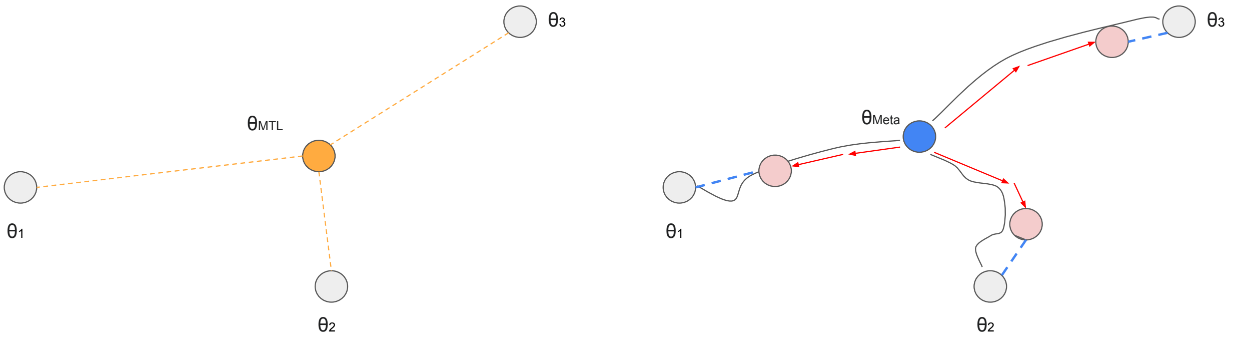


Figure 3.1: Differences between multi-task learning (MTL) and meta-learning (Meta). The optimal MTL model θ_{MTL} is visualized on the left-hand side, and the optimal model for meta-learning θ_{Meta} is on the right-hand side. Denote θ_i as the optimal model for training task τ_i. We provide this visualization with three meta-training tasks, and assume the model will perform two gradient updates (red arrows) during inference time.

As the name suggests, few-shot learning only allows a model to learn from a few data samples for a new task. So, many different algorithms, for instance, the multi-task learning, the foundation models, and the meta-learning algorithm, all belong to the few-shot learning paradigm. One of the most significant challenges in few-shot learning is the high-dimension, low-sample-size problem (i.e., $n < p$) [49], where the model could struggle to learn generalizable features.

3.2.4 Unsupervised and Self-Supervised Learning

While the above meta and few-shot models usually learn from supervised tasks, unsupervised learning is a mechanism to learn from abundant unlabelled data. The reconstruction task and autoencoder model [14] discussed in the previous chapter are examples of unsupervised methods used to pre-train deep learning models. Also inspired by neuroscience, predictive coding [20] is another common strategy for unsupervised learning. Recently, Kaiming He and many other researchers applied data augmentation for unsupervised learning, which includes masking [27] and frame predicting [21].

As a subset of unsupervised learning, self-supervised learning can learn from unlabelled and partially labeled data. For instance, in the teacher-student models (e.g., [50]), the "teacher" model would learn from labeled samples and infer unlabelled samples; then, the "student" model would learn from the teacher's predictions. In this scenario, the teacher distills valuable knowledge from the partially labeled source dataset for students to learn. In another setting, a self-supervised learner, such as Word2Vec [51] and the foundation model BERT [4], can use data augmentation or contrastive learning to gain knowledge.

3.2.5 Domain Adaptation

Researchers define "domain adaptation" as the ability to adapt from some source "domains" to a different target "domain." While this definition is very similar to the definition of transfer learning, domain adaptation usually assumes the source and target datasets are from different domains or even modalities. Researchers have designed several domain adaptation approaches, including low-resource learning [52], representation learning [53], and self-supervision [54].

3.2.6 Weakly Supervised learning

While data are from different domains in the previous scenario, labels are only weakly relevant in this weakly-supervised learning setting. For instance, researchers might only have coarse annotation for each image, but the downstream application needs detailed prediction on each pixel for segmentation. This type of task is usually more complex than simple domain adaptation, and we refer readers to the extensive survey paper [55].

3.2.7 Distribution Shifts and Active Learning

While the above methods are post-hoc algorithms, distribution shift needs real-time adaptation. Machine learning researchers have studied several shifts, including covariate, prior, concept, and domain shifts. After deploying a trained model for production, researchers might find that the incoming data samples might drift from the training set. For instance, after training a deep learning model on data from adults, researchers found that the major users are children, who have entirely different usage patterns. The standard and most effective strategy to adapt shifted data is to collect additional labels. However, when numerous data samples are available, the main question is how to select a limited amount of data samples to maximize the utility of labeling.

So, computer scientists have also designed several methods to address these problems. Balancing exploration and exploitation, multi-arm bandit machines [56] can detect meaningful data samples for labeling, where the upper confidence bounds algorithm [57], exponential-weight algorithm for exploration and exploitation (EXP3) [58], and similar algorithms have shown promising results in this setting. Similarly, active learning studies have applied margin-based [59], entropy-based [60], query-by-committee [61], expected model change [62], uncertainty sampling [63], expected error reduction [64], diverse subspace incorporation [65], and variance reduction [66] to choose unlabeled data.

3.3 Summary

As discussed, these advanced adaptation methods are closely related to each other, but they are usually used separately in different applications. The similarity of these data adaptation tasks and the distinctness of machine learning literature motivated us to unify them into a single framework.

Chapter 4
METHODOLOGY

Bringing inspiration from the prefrontal cortex to adapt to low-resource learning problems requires rigorous scientific explorations and non-trivial engineering efforts. We propose a unified data adaptation (UDA) framework to fulfill machine learning and engineering goals. The proposed framework can utilize self-supervision, meta-learning, domain adaptation, active learning, generation, and other relevant tasks to train generalizable models for data adaptation.

UDA aims to learn and transfer representations from source datasets to the small target dataset and future shifted data with self-adaptation. During training (the development thread), UDA can learn a common data representation technique from multiple external datasets and diverse tasks. This learning problem is similar to a model initialization problem, where multi-task learning Algorithm 1 and meta learning-based Algorithm 2 help UDA for training. During inference (the production thread), UDA can estimate confidence towards incoming data, evaluate variance on its decisions, detect the severity of data shifts, and evaluate trained models. If data shifts are small, UDA can automatically adapt to the slight change; when catastrophic shifts occur, UDA will require further data annotation, training, and re-evaluation from researchers. To facilitate these model adaptation goals, UDA can also estimate models' confidence toward new data samples.

UDA is not the first framework that applies multi-task learning, pretraining, active learning, or adaptation technique for neural networks. However, only a few studies combine these tasks into research and production. Inspired by the biological counterparts of human brains, UDA unifies these seemingly separated machine learning tasks together in a simple framework.

In this chapter, we first describe the design of UDA (Section 4.1) and two important threads and processes. Then, each neural network component (Section 4.2) is discussed, and a few toy examples for some components are simulated in Section 4.3. Algorithms and theoretical analyses are presented in Section 4.4 to develop the UDA system.

4.1 Overview

To accomplish these machine learning and engineering aims for low-resource adaptation, the UDA framework contains a versatile encoder and several decoders.The versatile encoder, $\mathcal{E}$, can encode a datum x (either from dataset S, dataset T, or other sources) to a fixed-length d-dimensional embedding vector in $(-1, 1)^d$. Taking an embedding as input, each decoder $\mathcal{D}$ has different functionalities for either supervised, unsupervised, or semi-supervised learning. This section illustrates how these components would interact in different stages.

The development and production stages are typical in machine learning cycles. However, the word "stage" might be inaccurate to describe UDA 's behavior. These two stages are usually interdependent, and they often happen concurrently. So, in our discussion, we refer to UDA as using two "threads": the development thread and the production thread. The development thread runs across a machine learning pipeline's whole life cycle. On the other hand, the production thread deploys the trained models for users (e.g., consumers, doctors, a computer system) after the initial training process converges. In reality, data and machine learning systems are dynamic; these two threads collaborate to make better predictions, estimate confidence, detect data shifts, and tune the model.

The ultimate goal of the development thread is to train a model that can perform accurately on the target task without overfitting to the small training set. To achieve this goal, UDA includes several decoding tasks to escape local optimum and regularize weights for small target datasets. UDA uses the idea of college curriculum design to learn from all courses (decoders) first. Then, it fine-tunes the target tasks. Machine learning literature has explored this idea extensively, and Section 4.4 discusses this training process.

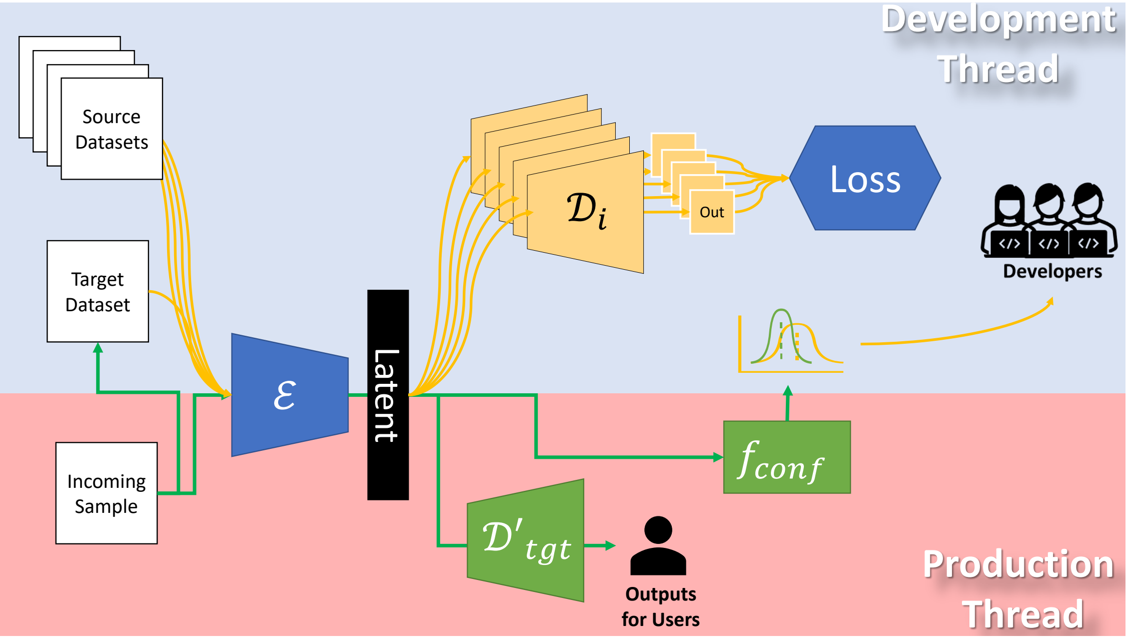


Figure 4.1: Illustration of the versatile encoder (blue), auxiliary decoders (yellow), and the target decoder (green) of the unified framework: in the development thread (with light blue background), all data and all decoders, including the target decoder, participate in the training process. In the production thread (with light red background), the encoder takes a datum as input and produces an embedding vector (black), while a copy of the target decoder ($\mathcal{D}'_{\text{tgt}}$) takes an embedding vector as input and produce outputs for users. The system will further use these new incoming data for developing and fine-tuning. When the confidence is low or irresolvable data shifts happen, the system would inform developers/maintainers of this system. Yellow arrows represent data flow in the development thread, while green arrows represent data flow in the production thread. For visual conciseness, we excluded details about different decoders (e.g., generative decoders and segmentation decoders) in this figure.

4.1.1 Development Thread

Training the encoder and decoder models mentioned above is the first step of machine learning model production. Developing a generalizable model could foster better adaptation to small datasets and faster adaptations to data shifts. Moreover, the development thread is not a one-time process in UDA. Because of the unpredictable nature of data shifts, UDA would continue to train, tune, and develop models concurrently during the production phase.

In low-resource learning settings, researchers usually have limited access to the actual data. So, learning from unlabelled data and external data could mitigate the data shortage problem. Supervised and unsupervised learning are included in the development thread to help the encoder learn from various data.

Based on multi-task learning and meta-learning, Algorithm 1 and 2 can help the encoder learn from various tasks. If the performance and convergence of these tasks satisfy minimum viable requirements, UDA would yield a production thread for deployment. This development thread is still ongoing to learn from newly-arrived data during the production and inference phase.

Training in the development thread could help adapt to novel data because of the training disparities from multiple decoding tasks and source datasets. However, the trained models are still vulnerable to bias in the training datasets. So, the production thread would continue adapting to actual data.

4.1.2 Production Thread

After deployment, the production thread must monitor the performance and degree of data shift in non-stationary environments. As many prior studies found, models' performance would deteriorate when data shifts from the training distribution. Hence, in addition to producing outputs for users, detecting performance drops and data shifts is another goal of the production thread. When data shifts are detected, UDA will fine-tune, re-train, request additional labels, or ask researchers for further help.

In most production pipelines, labels for newly-arrived data are not easily accessible until further annotation. In this condition, the confidence estimation function f_{conf} can evaluate UDA's confidence or variation on its predictions. When UDA shows low confidence or high variance, the system is unable to deal with the data shift and would request additional data annotation from researchers. In the meanwhile, unsupervised decoders in UDA would continue to study feature representation from these newly arrived data samples.

When labels for newly-arrived data are available (e.g., in a cancer diagnosis pipeline, when doctors confirmed/corrected the models' predictions), the production thread would invoke the development thread to perform additional training with these labels. For instance, fine-tuning, re-training, or re-evaluation could be performed concurrently in the development thread.

The development thread and the production thread depend on the encoders and decoders design; the following section discusses these components.

4.2 Components of the UDA

Modulization can allow researchers to freely choose needed components from UDA. However, if models are modularized too thin, more coding effort would be necessary to create a machine learning pipeline. To balance the flexibility and usability of modules, we split machine learning models into a versatile encoder and several decoders for each task.

For instance, the target decoder $\mathcal{D}_{tgt}$ can take an embedding as input and perform supervised learning to predict the labels in the target dataset. The generative decoder $\mathcal{D}_{gen}$ can take a noise vector to produce a synthetic data sample. The discriminative decoder $\mathcal{D}_{dis}$ can decide whether data are from the source datasets, the target dataset, or the generator. The reconstruction decoder $\mathcal{D}_{recon}$ can reconstruct the original datum by using embeddings from a corrupted input. The confidence estimation function f_{conf} can calculate how confident the system is of its predictions. We also provide a list of additional decoders for the encoder to learn data representations from diverse tasks, e.g., contrastive learning, reconstruction, and noise reduction.

The biological counterparts of human brains also inspire the separation of decoders. While the prefrontal cortex is the main power for executive function skills to detect and handle shifts, different brain parts are responsible for different information processing. The goal of UDA is not to fully emulate a biological brain or embody human brains to artificial neural network-based decoders. Instead, UDA should balance machine learning tasks and engineering considerations to adapt to small datasets and detect data shifts with limited resources. Separating the encoder from the decoders can help researchers fine-tune underlying machine learning models. It also creates a specific mathematical problem, as shown in Section 4.4, where the encoder's learning burden is alleviated by collaborating with the decoders.

4.2.1 Versatile Encoder

The encoder, as a crucial module in UDA, lives from the beginning of model pretraining to production. The encoder's primary responsibility is to find a suitable feature representation so that (1) data information can be encoded efficiently and (2) essential features can be extracted for future tasks. The intuition of encoders and decoders comes from neuroscience, which tells us that that human memory can encode, store, recall, and decode information [8]. Recently, unsupervised learning models, the "autoencoder" (aka, "encoder-decoder"), further applied the encoding theorem into neural networks [14].

Similar to [14], the UDA uses neural networks as versatile encoders, which can contain linear layers, convolutional layers, recurrent layers, and attention layers. More than one encoder might exist in multi-modality fusions and some other scenarios. For conciseness, we only discuss adaptation scenarios when only one encoder is available, and we will discuss multi-encoder cases briefly in Section 9. The versatile encoder also has several options to extend its capabilities.

Besides standard activation layers, normalization layers, and dropout layers, optional Monte Carlo (MC) dropout layers can help the encoder. While standard dropout layers could avoid overfitting, the Monte Carlo dropout layer can help calculate confidence scores and perform variational inference for input data samples. This optional MC dropout layer could help UDA to detect data shifts in the production thread. Similar to the MC dropout layers, variational Bayes [67] can calculate variance for input data. Kingma's variational autoencoder design is consistent with our encoder-decoder structure, which is another option for efficient variational inference in UDA.

Ronneberger et al. [68] found skip connections (aka, residual connections) between the encoder and decoder could help the model perform better on complex tasks. For instance, the semantic segmentation task usually requires low-, mid-, and high- levels of information, where skip connections can propagate these features. So, UDA also provide auxiliary skip-connection outputs where some decoders can utilize this information for more complex tasks. If the decoders use skip connections, corresponding decoders should have a symmetric architecture with the encoder.

These various options (MC dropout layer, variational Bayes layer, skip-connection outputs) can accommodate different applications. For instance, traditional classification and regression tasks (e.g., facial expression [69]) only need a standard encoder, and these options can all be turned off. With the variational Bayes layer turned on, a variational encoder can be helpful for real-time active learning systems (e.g., adaptation in DBMS [70]). An MC encoder, with the MC dropout layer, can help build calibration systems for eye-tracking, facial expression analyses, and other personalized models [71]. The optional skip connections from the encoder to other decoders can help instance segmentation for medical image analyses (e.g., in cancer analysis [72]).

4.2.2 Classification and Regression Decoders

With the encoder module, UDA can accomplish many different tasks by adding different decoders. Classification decoders use cross entropy-based losses to train the model, and regression decoders use mean squared errors (MSE) or mean absolute errors (MAE) during the training process. UDA also provides hinge loss, negative log-likelihood, Kullback-Leibler divergence, and other functions to train the model. The classification and regression decoders have the same structure, usually containing a sequence of fully connected layers, dropout layers, and activation layers. These decoders are the foundations of other types of decoders.

4.2.3 Reconstruction Decoders

While classification and regression are standard tasks in supervised learning, reconstruction is the most common task in unsupervised learning. Many previous studies (e.g., [14]) have explored the reconstruction task, and other unsupervised learning studies adapted similar structures for predictive coding, de-noising, data recovering, and other tasks. In these tasks, the input for and output from the neural network are in the same data space (e.g., $\mathbb{R}^{height \times width \times 3}$ for RGB images), and most of these algorithms adopt the encoder-decoder architecture.

The UDA framework combines the original reconstruction task with these novel unsupervised learning architectures to form a unified reconstruction decoder $\mathcal{D}_{\text{recon}}$. The architecture of reconstruction decoders can be either standard fashion or skip-connection fashion. The traditional reconstruction decoder only uses the latent feature vector as input with the standard autoencoder design. On the other hand, the skip-connection reconstruction decoder employs information from all encoder's layers as inputs (e.g., U-Net). The skip-connection reconstruction decoder is also a base class for generative decoders, segmentation decoders, and detection decoders below.

4.2.4 Discriminative Decoders and Generative Decoders

In the machine learning literature, discriminative models usually draw boundaries in the input space, and generative models aim to learn the distribution of data in the space [73]. Combined with a generator and a discriminator to construct the generative adversarial networks (GAN), Goodfellow et al. [42] created a novel unsupervised learning and data generation method through an mini-max training process.

The GAN architecture can help a machine learning pipeline learn an input space and feature representation. Moreover, there is sizable prior work in the machine learning literature on the GAN to generate realistic synthetic image, text and audio examples [74, 75, 76] which in turn have helped to train object detectors and image segmenters [77, 78]. This design also enables data augmentation in medical images, modality transfer with cycle consistency loss, and interpretable models. These data synthetic and data augmentation algorithms from the GAN can further help UDA to adapt data shifts more efficiently.

In the UDA, the generative decoder has the same architecture as the reconstruction decoder, because both decoders need to reconstruct the input space. The discriminative decoder is similar to the classification decoder, because both decoders need to distinguish between different data types. Unlike the previous decoders, discriminative and generative decoders require mini-max loss during the training process. Because this mini-max loss is a non-convex function, the system might not have a Nash equilibrium in the zero-sum game [79]. So, training generative and discriminative decoders could be a challenging task. However, the UDA has a flexible encoder design (e.g., with a variational Bayes layer), where other decoders (e.g., the reconstruction decoders) can collaborate to facilitate the encoder's learning process. So, these decoders and tasks could improve the training process for the GAN. Many recent studies, such as VAE-GAN [80] and Wass-GAN [81], use similar strategies to train the GAN system. During adaptation, generative decoders could help data augmentation methods during the production thread, and discriminative decoders could learn to distinguish data before and after data shifts.

4.2.5 Segmentation and Detection Decoders

While reconstruction decoders and generative decoders produce outputs from the input data space, segmentation and detection decoders produce outputs from slightly different spaces.

Semantic segmentation and object detection are two routine tasks in computer vision. Combining these two tasks, researchers defined *instance segmentation* as detecting and segmenting each distinct object from an image. Researchers also proposed a panoptic segmentation task to semantically segment background pixels and individually segment each distinct object of interest based on semantic and instance segmentation.

The segmentation and detection decoders in the UDA ground on the U-Net architecture and the Mask-RCNN design. Feature vectors and skip connections from all the encoder's layers flow into these decoders, where a region-proposal subnetwork (RPN) will propose regions-of-interest (ROIs) first, and a segmentation network will perform instance segmentation for these ROIs.

From an engineering point of view, segmentation decoders and detection decoders are the most complex ones in the UDA, because they contain different subnetworks. However, users can choose a subset of these decoders for desired tasks: detection, semantic segmentation, instance segmentation, or panoptic segmentation.

4.2.6 Meta-Learning Decoders

Unlike previous decoders, which usually learn from a single dataset, meta-learning decoders can learn the initialization of features from multiple datasets or sub-datasets. The UDA incorporates the prototypical network (ProtoNet) [82] as $\mathcal{D}_{\text{Proto}}$ to learn a metric space by computing distances on feature representations for each class. The $\mathcal{D}_{\text{proto}}$ can be a parametric network that learns a projection from the encoder's feature space to a novel space, and it can also be a non-parametric method that calculates feature distances with pre-defined functions.

4.2.7 Contrastive-Learning Decoders

While $\mathcal{D}_{\text{proto}}$ aims for meta-learning and few-shot learning, the SimCLR [83] decoder $\mathcal{D}_{\text{sim}}$ learns contrastively without supervision from labels. $\mathcal{D}_{\text{sim}}$ would first apply random data augmentation for input data and then learn to group augmented data from the source in the latent space. With a Siamese network design [84], the SimCLR method can learn visual representations without using extra labels or computer memory. The UDA further extended this SimCLR design to signals, videos, and feature vectors.

4.2.8 Confidence Estimation Function

For all the above decoders, confidence estimation abilities are added as a non-parametric function f_{conf}. As discussed in Section 4.1, this function will continuously monitor the machine learning pipeline's confidence toward incoming data in the production thread. Supporting this vital task flexibly, the UDA includes several different confidence estimation designs.

f_{conf} can utilize the discriminative decoder $\mathcal{D}_{\text{dis}}$ to predict data before and after the data shift. It can also utilize the MC dropout layer with any decoders to predict the model's confidence toward an output. This mechanism does not require additional neural layers, but it would run the inference function multiple times to produce a standard deviation. f_{conf} can also use embeddings from the variational Bayes layer from $\mathcal{E}$ to calculate variance directly.

Using all the components, we can construct complex pipelines to solve many different problems. The following section simulates three simple scenarios to showcase how different decoders collaborate to learn, predict, and adapt.

4.3 Simulations

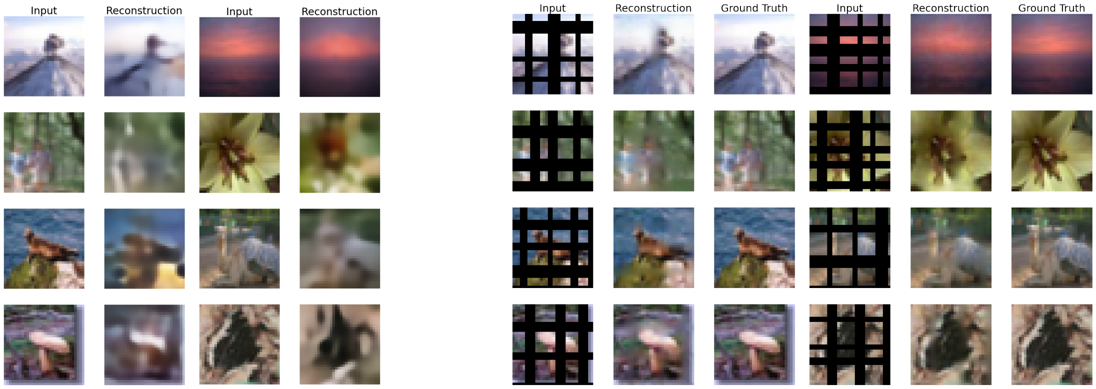
To study the rationalities and effectiveness of each proposed component, we show simple simulations on reconstruction, generation, and active learning in this section. We will thoroughly study the joint training and inference from these components in challenging real-world problems in later chapters.

4.3.1 Reconstructions Simulation

Self-supervised learning is a common strategy for low-resource data with or without labels, and simulating a reconstruction decoder could be helpful for all kinds of future studies. Two experiments are discussed on the CIFAR-100 dataset to reconstruct input data samples.

The first experiment utilizes a classic convolutional autoencoder with a $\mathcal{E}$ and a $\mathcal{D}_{\text{recon}}$: the encoder compresses a 32x32x3 image into 64 values in $\mathcal{E}$, which is only 2.1% of the original size. Then, the $\mathcal{D}_{\text{recon}}$ reconstructs the full image. As expected in Figure 4.2a, even if reconstructed images are blurry due to the low compression size, the UDA can still robustly reconstruct information from the latent space.

In one of our previous studies [85], we found neural networks can learn to remove unnecessary rows from the MNIST dataset. Inspired by this interesting behavior, we place random interlacing effects on the CIFAR-100 dataset. Then, a U-Net structure $\mathcal{E}$ and $\mathcal{D}_{\text{recon}}$ are applied to fill missing pixels for input images. As shown in Figure 4.2b, the UDA can successfully predict and reconstruct original images from corrupted inputs.



(a) Reconstructing

(b) Recovering with U-Net

Figure 4.2: Illustration for reconstruction tasks in the testing set. (Left) Encoding input image from 3072 values to 64 values, and then decode it back into the original image format. (Right) Random interlacing effects, such as blank vertical and horizontal stripes, are applied to images. The encoder would need to encode known information to a latent space, and the decoder needs to guess and reconstruct actual images from partial observations. A UNet-based encoder-decoder structure is used for recovering and denoising here.

4.3.2 Interpretation of Attention from GAN

We use cycle-consistency loss to interpret a CNN's decision and attention on the MNIST dataset. In this experiment, the UDA trains an encoder $\mathcal{E}$, two discriminative decoders $\mathcal{D}_{\text{dis}}$, and two generative decoders ($\mathcal{D}^a_{\text{gen}}$ and $\mathcal{D}^b_{\text{gen}}$). In this simulation, we define domain A as the class for "5" and domain B as the class for "2". These networks should learn how to modify one handwritten digit to another digit. This naive simulation shows the possibility of using a GAN to interpret the model's decisions on classification tasks, such as medical diagnosis.

Unlike the CycleGAN and StarGAN, the architecture used in this experiment utilizes the versatile encoder $\mathcal{E}$ for all structures (i.e., generator A, generator B, discriminator A, discriminator B), as shown in Figure 4.3a. The versatile encoder acts as a fair judge to only encode data representations, and decoders ($\mathcal{D}_{\text{gen}}$ and $\mathcal{D}_{\text{dis}}$) are responsible for generating and discriminating from latent vectors. Results are shown in Figure 4.3b, where UDA learns useful information between the two domains (classes).

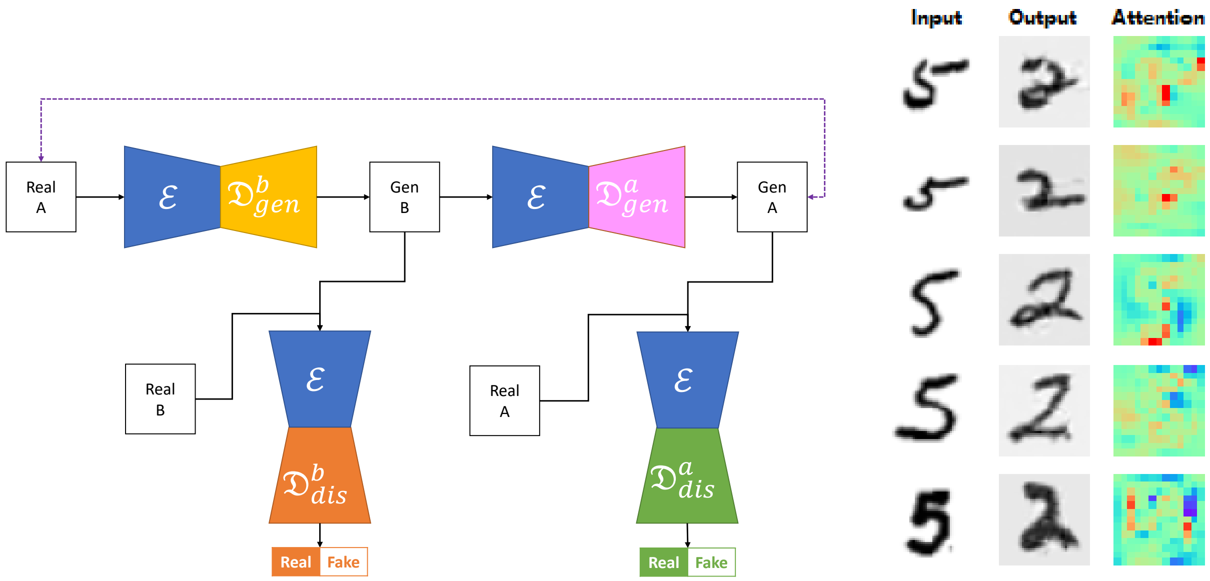
4.3.3 Detecting Shifts

Both simulations above are in the development thread, and we simulate a variance estimation scenario here for shift detection in the production thread. The encoder $\mathcal{E}$ and a classification decoder $\mathcal{D}_{\text{cls}}$ collaborate to estimate confidence in this experiment.

We use the original MNIST dataset as the training data, where the pipeline achieved 98% accuracy in the hold-out testing set. After deploying these models to production, unexpected data shift (i.e., the interlacing effects discussed earlier) occurs gradually and worsens over time. The confidence function f_{conf} monitors the pipeline's confidence in incoming data. Figure 4.4 shows the severity of incoming data, testing performance, and predicted variance. We can see a clear correlation among these metrics, which illustrates the feasibility of using the confidence estimation function in the UDA for future data shifts detection.

4.4 Algorithms and Theoretical Analyses

While the previous sections discuss various components and three simulations, we discuss two learning algorithms to unify the versatile encoder with the heterogeneous decoders in this section. Based on recent discoveries in multi-task and meta learning, UDA's training algorithms can efficiently and robustly learn from multiple tasks.



(a) Model

(b) Results

Figure 4.3: CycleGAN Simulation in UDA. (Left): Illustration of the CycleGAN architecture in the UDA. Each trapezoid represents a model (an encoder or a decoder), where trapezoids in the same color have the same weights (parameters). (Right):the first column is the input, the second column is the synthesized output, and the last column is the output attention from the discriminative decoder for the input, where high-attention values are near the top or bottom horizontal strikes. The generative and discriminative decoders can help synthesize more data and be used for decision interpretation.

When only limited low-resource data are available, machine learning models need to learn from diverse tasks, obtain generalizable parameters, and avoid overfitting small datasets. We assume all decoding tasks τ_i are from the same distribution, $p(\tau)$. Below, we propose two distinct methods: the multi-task learning approach is fast and convenient but sometimes cannot escape from local optimum; the meta-learning approach is slower but is less vulnerable to saddle points.

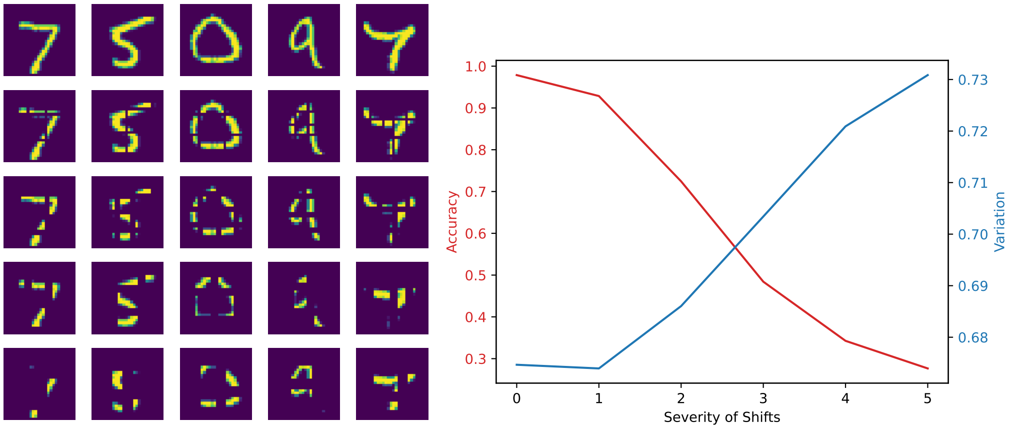


Figure 4.4: Severity of Data Shift and UDA Variation Prediction. **Left:** the severity of data shifts (corruption) increases row-by-row. **Right:** the machine learning pipeline's accuracy (colored in red) over shifted distribution, and UDA variation prediction (colored in blue). Their correlation is around -0.97.

4.4.1 Multi-Task Learning Approach

The UDA can apply multi-task learning (MTL), as many prior studies have done, to learn from various decoders and tasks. The MTL approach can combine losses and learn jointly for different tasks. In Equation 4.1, let $\mathcal{L}$ represents the loss, and let $\mathcal{L}_i$ represents the training loss for training task τ_i. The goal is to minimize the training loss, that is $\min \mathcal{L}$. We denote $\mathcal{E}_{\phi_i}$ as a trained encoder for task τ_i. The phrase "$\tau_i \sim p(\tau)$" means selecting the task τ_i from the population of all decoding tasks, $p(\tau)$. Let ϕ represents the weights (parameter values) of a neural network, and $\phi_\mathcal{E}$ represents the weights of the versatile encoder. Equation 4.2 shows the update rule for optimizing the encoder's parameters. Table 4.1 shows the notation used in these analyses.

$$\min_{\phi_\mathcal{E}} \mathcal{L}(\phi_\mathcal{E}) = \min_{\phi_\mathcal{E}} \sum_{\tau_i \sim p(\tau)} \mathcal{L}_{\mathcal{T}_i}(\mathcal{E}_{\phi_i}) \tag{4.1}$$

With a learning rate of α, the optimization goal is:

Set	Notation	Meaning
Model	$\mathcal{E}$	The versatile encoder
	$\mathcal{D}_\text{i}$	The decoder for task i
	$\widetilde{\mathcal{E}}$	A temporary deep copy of the encoder
	ϕ	Parameters for a model
	$\tilde{\phi}$	Empirical parameters
	$\mathcal{E}_\phi$	The encoder with parameters ϕ
	$\phi_\mathcal{E}$	Parameters for the encoder
	$\phi_{\mathcal{D}_\text{i}}$	Parameters for the decoder i
	$U(\phi, \tau_i)$	Updates for parameters ϕ on task τ_i
Data	S	A collection of source datasets
	T	The target dataset
	N	New incoming data for the target dataset
	$x_i, y_i \sim S$	A data sample and its label from dataset S
Task	τ_i	The i-th decoding task
	$p(\tau_i)$	Population of all decoding tasks
	$\tau_i \sim p(\tau_i)$	Select i-th task from the population of all decoding tasks
	τ_{tgt}	The target task
Training	$\mathcal{L}$	Loss for the UDA system
	$\mathcal{L}_{\mathcal{T}_i}$	Training loss for the i-th decoding task
	$\hat{\mathcal{L}_{\mathcal{T}_i}}$	Testing loss for the i-th decoding task
	g_i	Gradient obtained from training on task i

Table 4.1: List of common notations used for equations and analyses.

$$\phi_{\mathcal{E}} \leftarrow \phi_{\mathcal{E}} - \alpha \nabla_{\phi_{\mathcal{E}}} \sum_{\tau_i \sim p(\tau)} \mathcal{L}_{\mathcal{T}_i}(\phi_{\mathcal{E}_i}) \tag{4.2}$$

Based on this straightforward loss and optimization equations, we list the simple learning algorithm in Algorithm 1, where descriptions for each line are colored in blue.

Algorithm 1: Training with Multi-Task Learning in UDA.

```
Input:  $\mathcal{E}$ (the encoder), $\{\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_\mathrm{t}\}$ (list of decoders), $\{\tau_1, \tau_2, ..., \tau_t\}$ (list of
        decoding tasks), $\alpha$ (learning rate).
while Not Done do
    $x, y \sim S$ ;                                   // Sample data (and labels) randomly from datasets.
    $\mathcal{L} \leftarrow 0$
    for $\tau_i \in p(\tau)$ do
        $\mathcal{L}_{\mathcal{T}_i} \leftarrow Loss(\mathcal{E}, \mathcal{D}_\mathrm{i} | x, y)$ ;          // Calculate the loss for task $i$.
        $\mathcal{D}_\mathrm{i} \leftarrow Update(\mathcal{L}_{\mathcal{T}_i}, \mathcal{D}_\mathrm{i}, \alpha)$ ;                    // Update $i$-th decoder.
        $\mathcal{L} \leftarrow \mathcal{L} + \mathcal{L}_{\mathcal{T}_i}$
    end
    $\mathcal{E} \leftarrow Update(\mathcal{L}, \mathcal{E}, \alpha)$ ;                         // Update the encoder.
end
Fine-tune $\mathcal{E}, \mathcal{D}_\mathrm{tgt}$ for target task $\tau_\mathrm{tgt} \in p(\tau)$ ;      // Fine-tune for the target task
```

4.4.2 Meta Learning Approach

Besides using multi-task learning, the UDA can also phrase the data-adaptation problem as an encoder initialization quest. The UDA needs first to learn a common encoding representation from diverse decoding tasks; then, it can adapt to the target small dataset. So, similar to meta-learning problems, learning the initial encoding representation is the ultimate goal for the development thread. Based on model-agnostic meta learners (MAML) [48], the learning goal for the versatile encoder can be phrased as Equation 4.4.

Following the notations used above, we additionally denote $\hat{\mathcal{L}_{\mathcal{T}_i}}$ as the testing loss for task τ_i. The meta learning goal follows:

$$\min_{\phi_{\mathcal{E}}} \mathcal{L}(\phi_{\mathcal{E}}) = \min_{\phi_{\mathcal{E}}} \sum_{\tau_i \sim p(\tau)} \hat{\mathcal{L}_{\mathcal{T}_i}}(\mathcal{E}_{\phi_i}) \tag{4.3}$$

$$= \min_{\phi_{\mathcal{E}}} \sum_{\tau_i \sim p(\tau)} \hat{\mathcal{L}_{\mathcal{T}_i}}(\mathcal{E}_{\phi} - \alpha \nabla_{\phi_{\mathcal{E}}} \mathcal{L}_{\mathcal{T}_i}(\mathcal{E}_{\phi})) \tag{4.4}$$

Here, α is the learning rate for each decoding task. Let β be the learning rate for the versatile encoder. $\nabla_{\phi_{\mathcal{E}}}$ represents the partial derivative for parameters $\phi_{\mathcal{E}}$, and ∇^2 represents the second-order derivative (i.e., Hessian matrix).

Then, we can update the weight for $\mathcal{E}$, $\phi_{\mathcal{E}}$, as follows:

$$\phi_{\mathcal{E}} \leftarrow \phi_{\mathcal{E}} - \beta \nabla_{\phi_{\mathcal{E}}} \sum_{\tau_i \sim p(\tau)} \hat{\mathcal{L}_{\mathcal{T}_i}}(\phi_{\mathcal{E}_i}) \tag{4.5}$$

$$= \phi_{\mathcal{E}} - \beta \nabla_{\phi_{\mathcal{E}}} \sum_{\tau_i \sim p(\tau)} \hat{\mathcal{L}_{\mathcal{T}_i}}(\mathcal{E}_{\phi} - \alpha \nabla_{\phi_{\mathcal{E}}} \mathcal{L}_{\mathcal{T}_i}(\mathcal{E}_{\phi})) \tag{4.6}$$

$$= \phi_{\mathcal{E}} - \beta \sum_{\tau_i \sim p(\tau)} (I - \alpha \nabla^2_{\phi_{\mathcal{E}}} \mathcal{L}_{\mathcal{T}_i}(\mathcal{E}_{\phi})) \nabla_{\phi_{\mathcal{E}}} \hat{\mathcal{L}_{\mathcal{T}_i}}(\mathcal{E}_{\phi}) \tag{4.7}$$

The main distinction between UDA and MAML is the existence of various decoders. In MAML, the model aims for a single stationary point $\phi^{\star}$ where the model can quickly adapt to all training tasks. However, UDA only needs to reach a stationary subspace rather than a single point. The decoders can project from the subspace to other points, simplifying the training process.

The optimization goal with second-order gradients requires additional memory and computation time, which could be burdensome in the already slow development thread. In previous meta-learning studies, Hessian-free optimization methods can learn more efficiently: for instance, using Newton's method to estimate the Hessian (e.g., HF-MAML); using a constant to replace the inner gradient (e.g., FO-MAML); or applying implicit Jacobian to compute the meta gradient (e.g., iMAML). In this study, we adopt ideas from the Reptile [86] with first-order gradients to train the UDA framework.

Due to the existence of decoders, we modify Reptile and create the algorithm below (Algorithm 2). Instead of performing one gradient update to model parameters (shown in Equation 4.4), Reptile and Algorithm 2 use k gradient updates for each decoding task.

Algorithm 2: Training with Meta-Learning (Reptile) in UDA.

```
Input: $\mathcal{E}$ (the encoder), $\{\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_\mathrm{t}\}$ (list of decoders), $\{\tau_1, \tau_2, ..., \tau_t\}$ (list of
       decoding tasks), $k$ (number of inner steps), $\alpha$ (learning rate for inner model
       update) $\beta$ (learning rate for outer encoder update)
while Not Done do
    $\tau_i \sim p(\tau)$ ;                                  // Randomly sample a task
    $D_i \leftarrow$ decoder for task $\tau_i$ ;             // Fetch the corresponding decoder
    $\widetilde{\mathcal{E}} \leftarrow \mathcal{E}$ ;       // Get a temporary deep copy of encoder
    for $step \in [1, 2, ..., k]$ do
        $x, y \sim S$ ;                                      // Sample data (and labels) from the source datasets
        $\mathcal{L}_{\mathcal{T}_i} \leftarrow Loss(\widetilde{\mathcal{E}}, \mathcal{D}_\mathrm{i} | x, y)$ ;   // Calculate the loss
        $\widetilde{\mathcal{E}} \leftarrow Update(\mathcal{L}_{\mathcal{T}_i}, \widetilde{\mathcal{E}}, \alpha)$ ;   // Update the temporary encoder
        $\mathcal{D}_\mathrm{i} \leftarrow Update(\mathcal{L}, \mathcal{D}_\mathrm{i}, \alpha)$ ;   // Update $i$-th decoder with learning rate $\alpha$
    end
    $\mathcal{E} \leftarrow \mathcal{E} + \beta(\widetilde{\mathcal{E}} - \mathcal{E})$ ;   // Update the actual encoder with learning rate $\alpha$
end
Fine-tune $\mathcal{E}, \mathcal{D}_\mathrm{tgt}$ for target task $\tau_\mathrm{tgt} \in p(\tau)$ ;   // Fine-tune for the target task
```

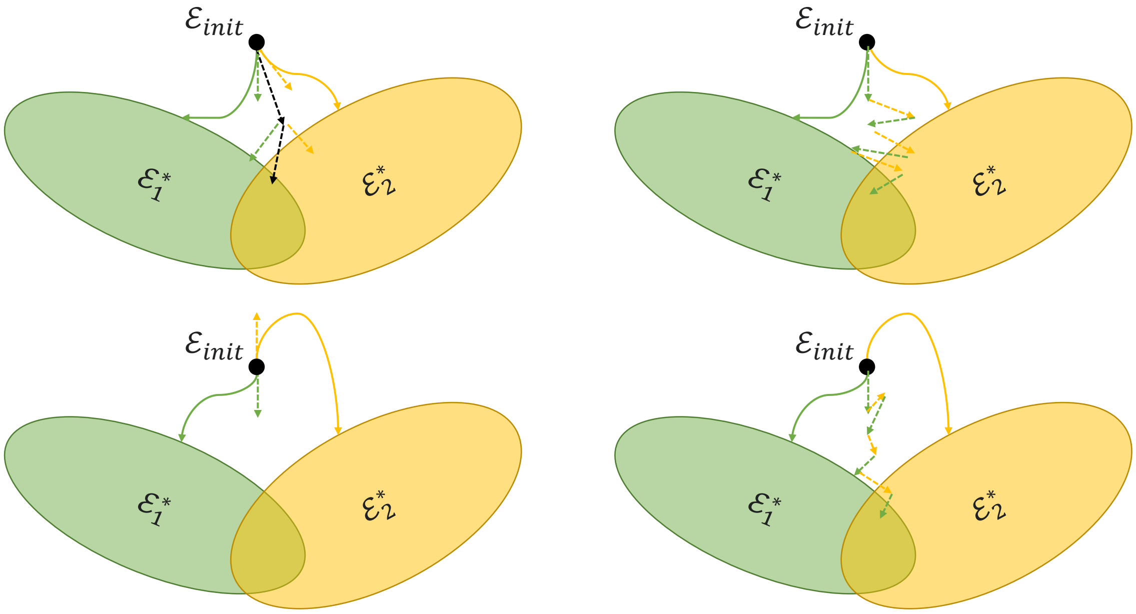
Note that the inner loop in Algorithm 2 can be parallelized and performed distributedly on different machines or data centers (e.g., hospitals and universities), which can preserve privacy and enable federated learning. Chapter 9 will discuss the privacy issues.

4.4.3 Analyses

Understanding the theoretical convergence of optimization algorithms can facilitate machine learning development, but modern mathematical tools are not powerful enough to analyze complex neural networks. Many questions are still unanswered even for simple supervised image classification tasks [7]. Some recent literature analyzed multi-task learning and meta-learning algorithms by imposing unrealistic assumptions (e.g., smoothness, Lipschitz continuity, bounds of loss functions, and bounds of data variances) about the dataset and tasks, which usually do not hold in low-resource settings. So, we only briefly discuss intuitions and rationalities behind the UDA. More rigorous analyses on optimization algorithms are needed in the future when mathematical tools are mature, and we only evaluate the UDA empirically in this study. We also raise important questions about low-resource adaptation for future studies in Chapter 9.

Unlike many other studies, the UDA separates models into an encoder and several decoders. Parametric decoders can quickly learn a non-linear projection from one latent space to another manifold. So, the optimal solution for the encoder is a subspace rather than a single point. For instance, as shown in Figure 4.5, the green region represents the optimal subspace for decoding task τ_1 and the yellow region represents the optimal subspace for decoding task τ_2; once the encoder reaches the intersection region, both decoders can find a projection to minimize training losses. In the sense of reaching an optimal space, the UDA is very similar to transfer learning and foundation methods.

MTL has two drawbacks. First, the MTL loss aims to learn the encoder's parameters $\phi_{\mathcal{E}}$ as the average of parameters for all decoding tasks, which is not necessarily the best parameter for adapting the small target dataset (discussed in Section 3). Second, averaging gradients from all decoding tasks might smooth the learning to zero, as shown in Figure 4.5a. Nevertheless, MTL is highly efficient and can learn from multiple tasks quickly even if the source datasets are huge. Thus, MTL is usually used to pre-train foundation models by using abundant training corpus. Because MTL is already widely studied in the literature,



(a) MTL

(b) Reptile

Figure 4.5: Comparison of the MTL and the Reptile learning algorithms: for task i, the optimal space for encoder parameters is in $\mathcal{E}_i^\star$. So, the overlapped region of the green (task 1) and the yellow (task 2) is the optimal region for the encoder. Dashed lines represent gradients for each task, and the black dashed line represents the sum of the two gradients. With the same learning rate, MTL learns more efficiently if the tasks are correlated (as in the first row), but the encoder might get stuck at a point where the averaged gradient is close to zero (as in the second row). On the other hand, although it runs slower than MTL, reptile could escape the local optimum by moving around.

we refer readers to [87] for multi-objective optimization within multi-task learning.

In the meta-learning approach, the encoder is not updated directly with decoding losses, but it is updated to the direction of $\widetilde{\phi} - \phi$, where $\widetilde{\phi}$ is trained by SGD and ϕ is the original model. Despite Algorithm 2 only using first-order gradients for each update, it contains second-order gradients learning and looking ahead by k steps. Let us denote G_j as the gradient at step j during the inner loop optimization (in Algorithm 2), where $j \in [1, k]$. For simplicity, denote θ as a shorthand for $\widetilde{\mathcal{E}_{\phi}}$, the weight for the encoder during the inner loop optimization, and let θ_j be the weight after the j-th update. Following conventions in [86] and expanding G_j with the Taylor theorem, the following equation illustrates the outer-loop update rule, where $O(||\theta_j - \theta_0||^2)$ is a small irresolvable rounding error.

$$
\begin{aligned}
\theta &\leftarrow \theta + \beta \sum_{j=1}^{k} G_j \\
&= \theta - \alpha\beta \sum_{j=1}^{k} \Big(\nabla \mathcal{L}_{\mathcal{T}_i}(\theta_0) + \nabla^2 \mathcal{L}_{\mathcal{T}_i}(\theta_j)(\theta_j - \theta_0) + O(||\theta_j - \theta_0||^2) \Big) \\
&\approx \theta - \alpha\beta \sum_{j=1}^{k} \Big(\nabla \mathcal{L}_{\mathcal{T}_i}(\theta_0) + \nabla^2 \mathcal{L}_{\mathcal{T}_i}(\theta_j)(\theta_j - \theta_0) \Big)
\end{aligned}
$$

When more than one decoding task (corresponds to $\mathcal{L}_{\mathcal{T}_i}$) is included, Algorithm 2 can approximate the second-order gradients in Equation 4.7 to find the best initial parameters for data adaptation. Geometrically, Figure 4.5b shows how the UDA can escape from a local optimum point, and [86] shows gradient update in detail. From a mathematics perspective, the second-order optimization provides additional curvature information for the loss function to adaptively estimate the optimization trajectory [88], which can help models adapt to new data and tasks more rapidly. Hence, Algorithm 2 can efficiently train a generalizable encoder $\mathcal{E}$ in the outer while loop.

In summary, researchers can choose a suitable optimization algorithm (Algorithm 1 or 2) based on the nature of the source datasets. When the source datasets contain many samples, multi-task learning can optimize the encoder quickly; when both the source datasets and the target dataset are small, the meta-learning method could learn better by avoiding local optima. Researchers can also combine these two approaches; for instance, the UDA can first apply the multi-task learning approach to quickly cold start the model and then use the meta-learning approach to fine-tune the encoder further.

4.5 Implementation and Engineering Details

While machine learning ideas and theoretical analyses are helpful, putting them together with an engineering effort is necessary to concretize ideas into real applications. We facilitate other open-source repositories to augment the scope of the UDA: for instance, Detectron-2 can help instance segmentation, TensorBoard helps training visualization, and Proto adds support for the prototypical network. We also included many of our previous study repositories into UDA: SGIN adds support for L1 regularization, OBF adds functionality for signal processing, DIOP helps medical imaging analyses.

Data handling, model, learning paradigm, and inference technique are the most critical parts of machine learning pipelines. We separate these parts into smaller modules for easier code readability, maintenance, and re-factorization.

4.5.1 Data

All machine learning processes start from data loading and processing. Especially in low-resource settings where data or labels are small, handling data is the key to training generalizable neural networks. The UDA supports loading, normalizing, and augmenting images, signals, and structured attributes.

Image: The UDA supports grayscale, RGB, RGBA, and other custom format of images. Random cropping, jittering, nosing, affine transformation, and flipping can be applied to augment input images.

Signal: Unlike images, temporal information is hard to extract inside signals. The UDA supports loading multi-channel signal data. Random offset, scale, rotation, shear, and noise can augment signal datasets in the UDA.

Structured Data: images, signals, and time-series data are usually unstructured data. However, many applications can use structured data (e.g., electronic health records and database optimization features). The UDA supports loading these structured attributes from SQL, CSV, and Excel files. Random noise and value removing can be applied to augment structured data.

Besides these data format and data augmentation methods, other customized features can also be supported.

4.5.2 Model

Similar to the flexible data handling implementations, model implementation in the UDA also supports more than ten architectures. Most model architectures inherit from PyTorch or Torchvision libraries for easier maintenance. The UDA includes ConvNeXt, ResNet, MobileNet, and other backbones for CNN-based encoders and decoders. While U-Net and RCNN structures can support segmentation and detection, single-direction and bidirectional RNNs (RNN, GRU, LSTM) can analyze time-series data. MLP-based and Transformer-based models are also suitable for all input data types.

4.5.3 Learning

Learning paradigms can connect data handling and model architecture in the development thread. Each learning paradigm is associated with each decoder mentioned in Section 4.2. While PyTorch includes implementations for standard classification and regression learning, UDA mainly focus on more advanced learning paradigms.

Self-supervised learning: reconstruction decoders $\mathcal{D}_{\text{recon}}$, SimCLR decoders $\mathcal{D}_{\text{sim}}$ are included in the UDA's self-learning paradigm. These learning paradigms allow neural networks to learn from unlabelled data, which can alleviate researchers' annotation burden when limited resources are available.

Adversarial learning: the mini-max loss in discriminative decoders $\mathcal{D}_{\text{dis}}$ and generative decoders $\mathcal{D}_{\text{gen}}$ is the only non-convex loss used in UDA. However, knowledge from other tasks could help training GANs. Inclusion of skip connections from $\mathcal{E}$ to $\mathcal{D}_{\text{gen}}$ and the variational Bayes layer can also help the adversarial learning process.

Meta-learning: prototypical decoders $\mathcal{D}_{\text{proto}}$ learns from metrics for meta learning and metric-based learning. Algorithm 2 can combine all tasks together within the UDA system.

4.5.4 Inference

While learning paradigms correspond to the development thread, inference tasks are related to the production thread. $\mathcal{D}_{\text{cls}}$, $\mathcal{D}_{\text{reg}}$, $\mathcal{D}_{\text{seg}}$, $\mathcal{D}_{\text{det}}$, and other decoders are implemented as object-oriented classes to produce outputs for users. The confidence estimator f_{conf} runs as a non-blocking thread to detect drops in confidence and increases of data shifts inside the production thread.

4.6 Summary

In this chapter, we introduced the UDA framework that unifies different low-resource adaptation scenarios and tasks together. The UDA contains a development thread to train machine learning models and a production thread to serve end user applications. We modularize neural networks into a versatile encoder and several tasks-specific decoders, which perform well in simulations. In the following chapters, we will show how the UDA solves challenging real-world problems.

Chapter 5
APPLICATIONS FOR MEDICAL IMAGING

In this chapter, we introduce the first application – classifying breast histopathological images [72] with the UDA framework. Pixel-wise labels are usually scarce in medical images because of high image resolutions and expensive annotation costs. To resolve the label scarcity issue in this low-resource setting, the UDA absorbs an off-the-shelf semantic segmentation model as the versatile encoder, requests additional annotation through weak supervision, creates an instance segmentation pipeline on top of the newly acquired labels, and performs breast cancer classification with extracted features.

Suggestions from doctors and clinicians are often helpful while developing automated cancer diagnostic tools. In this medical imaging application, we integrated experts' domain knowledge with the UDA system, where we acquired additional labels to adapt complex breast biopsies with only 100 labeled images. This application shows the flexibility of using the UDA for expert-machine collaboration.

5.1 Dilemmas in Breast Cancer

Breast cancer is one of the most common cancers for females: about 13% of women will develop breast cancer over their lifetimes, and 2.6% of women will die from breast cancer in the United States [89]. The recent development of computer vision screening tools for mammography [90] could reduce the second reader's workload, but diagnosing breast cancer is still a time-consuming and challenging task. Physicians usually recommend breast biopsies for diagnosis and treatment plans after finding suspicious areas in mammograms, ultrasound, or magnetic resonance imaging (MRI). Analyzing breast biopsies is essential for breast cancer diagnosis, but even pathologists only agree with each other in 70% of cases.

When analyzing breast biopsies, pathologists usually focus on ducts because most breast cancers begin in the terminal ducts or lobules of the breast [89]. In ductal carcinoma in situ (DCIS), cells inside ducts undergo malignant transformation to cancer cells; in invasive breast cancer, these abnormal cells have escaped from the duct and are growing in the surrounding tissue [89]. Hence, tissues surrounding duct borders are the most relevant regions for pathologists and machine learning models.

Based on the importance of ductal regions, Mercan et al. designed structure features [91] to summarize the architectural characteristics in duct-based structures. Their method can emulate pathologists' behaviors to interpret diagnostic decisions and has been shown to outperform pathologists on the difficult task of categorically differentiating DCIS from Atypia. They identified duct instances (i.e., all pixels inside an individual breast duct or lobule) by applying a union-find algorithm to split semantic segmentation predictions of specific tissue classes into smaller duct regions. However, many ducts entangle with each other in breast biopsies, and this approach cannot distinguish a duct instance that is adjacent to other ducts, as shown in Figure 5.1. Moreover, extracting their structure features from biopsies can take hours on a computer because this algorithm is not suitable for parallel processing on multiple computer cores. Tissue-level semantic segmentation labels are also expensive and hard to acquire. To create an interactive and real-time tool for clinical use, researchers need to improve the accuracy and reduce the computation time by adapting to the limited amount of biopsies and labels.

On the other hand, many research groups have designed end-to-end deep learning systems to classify breast histopathology images, including fully-convolutional networks (FCNs) [92], patch-to-ROI-level feature representation [93], and graph convolutional networks [94]. Mehta et al. also designed Y-Net [95], which can perform image segmentation and diagnostic classification at the same time. These systems are usually accurate and fast because of the recent advancement of Graphical Processing Units (GPUs) and parallelism for matrix manipulation. Nevertheless, due to the lack of ductal labels, these methods are somewhat blind to the underlying pathological and structural features that led to the clinical diagnosis.

They do not guarantee a focus on ductal regions and cannot offer a decisive interpretation as [91] does for pathologists. Even if the training data is limited, knowing why a deep learning algorithm makes a certain diagnostic decision is crucial in clinical practice.

While researchers often have to balance the trade-offs between human labeling time, computation speed, accuracy, and interpretability within these computer-aided diagnoses (CAD) tools, label scarcity is the leading cause of these dilemmas. Acquiring pixel-level annotation for breast biopsies costs a vast amount of experts' time, and labeling all biopsies is impractical in a real-world scenario. As a typical low-resource learning scenario, the lack of labels hindered the deployment of deep learning models to solve medical imaging problems. Hence, we bring the UDA framework to help mitigate this dilemma. Without any instance segmentation labels in the breast biopsy dataset, we adapt the UDA with weakly-supervised annotation to acquire additional labels based on the previously trained versatile encoder and semantic segmentation decoder.

In this study, we applied the UDA to identify individual duct structures in breast biopsies, extract features, and classify breast cancer diagnoses. By adding an instance segmentation model in the development thread, the UDA can request additional weakly-supervised labels from human annotators. The proposed pipeline improves upon previous approaches on all diagnosis tasks, outperforms general human pathologists in two out of three binary classification tasks, and achieves comparable performance to general pathologists in a four-way diagnostic classification task, distinguishing among Benign, Atypia, Ductal Carcinoma in Situ, and Invasive Cancer category examples.

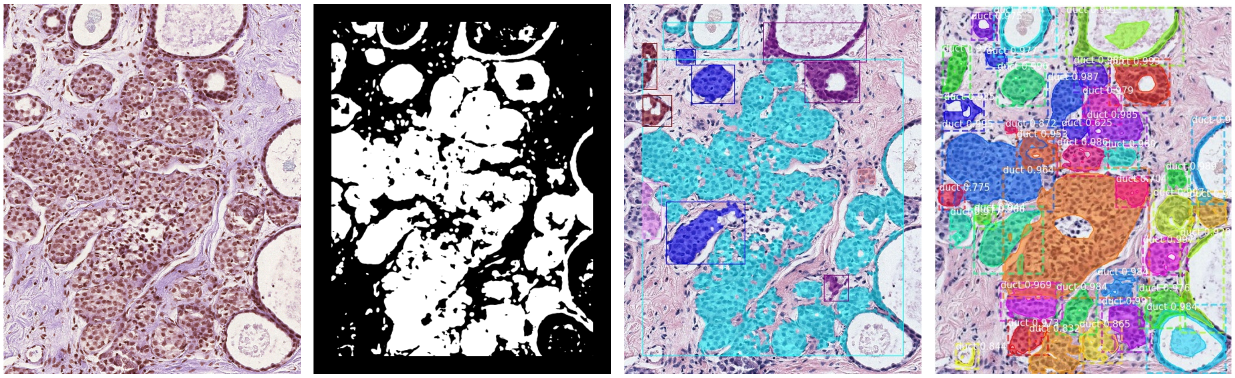
5.2 Related CAD Studies

Recent developments in breast cancer assessment, semantic segmentation, instance segmentation, and weakly supervised learning for medical imaging provide the groundwork for our study. Semantic segmentation is a common task in medical imaging; it partitions an image into multiple tissues, grades, or classes by classifying each pixel inside the image. For example, LSBB [96], Y-Net [95], and ESPNet [97] were designed for breast biopsy semantic segmentation; multi-scale U-Net [98], vanilla FCNs [99], EM-based models [100], and attention models [101] were created for prostate cancer; specialized auto-encoder [102], FCNs [103], and U-Net [104] were used for melanoma segmentation tasks. Unfortunately, these methods could not identify duct instances inside a region of interest (ROI) because semantic segmentation could not differentiate these instances from the semantic labels. Instead, instance segmentation labels, which contain all pixels within each breast duct or lobule, are needed.

While semantic segmentation has been widely applied to cancer diagnosis, instance segmentation is rarely used. Li et al. designed Path R-CNN (regions with convolutional neural network features) [105] based on Mask R-CNN [106] to classify glands and grade prostate cancer, which pioneered instance segmentation for medical imaging. In the recent two years, instance segmentation for nuclei [107], cluttered cells [108], polyps [109], and other tissues have been developed. In breast biopsies, each duct might contain different structural information about tissues, which could not be provided by tissue-level semantic segmentation. Hence, performing instance segmentation for ducts could be valuable for classifying histopathology images.

Acquiring instance segmentation labels for ducts is a tedious and time-consuming process, and no public datasets are available for instance segmentation on ducts to the best of our knowledge. For images with simple structures, semantic segmentation labels can be easily converted to instance segmentation labels using union-find, connected components, or other rule-based algorithms. Unfortunately, ducts inside breast biopsies are too complex for these rule-based label conversion algorithms, and instance annotations are needed to train an instance segmenter.

Geometrically, the shape of a duct is similar to the shape of a “doughnut”: it is usually ring-shaped with a thick circular border and central space, but often it does not have any holes because microscopic histopathology images are two-dimensional cross sections through three-dimensional structures. Breast ducts are analogous to pipes or tubes, and breast lobules are analogous to a hollow ball such as a tennis ball. Pre-cancerous conditions can



(a) ROI image (b) Binary (c) Previous (d) Ours

Figure 5.1: **Duct instances:** (a) the input ROI image in RGB color space; (b) the binary image inferred from tissue-level semantic segmentation, where the white pixels are ducts; (c) duct instances found by mathematical morphology and connected component algorithm; (d) the ducts inferred from our system. In (c) and (d), each color represents one duct instance. The connected component method (c) could not distinguish duct instance from the conglomerated region, even if it has been used to solve similar problems (e.g., in [91, 112]).

also cause the lining of the ducts and lobules to proliferate and fill the holes. All these features complicate rule-based algorithms. The difficulties compound when many ducts are adjacent to each other so that the borders of these ducts are difficult to distinguish. In DCIS and invasive cases, cancerous cells first begin to distort then escape from ducts, respectively, and thus duct cross sections develop into various and complex shapes.

Recent studies show that weak annotation [110], imperfect annotation [55], active learning, and human-in-the-loop methods [111] can be used to alleviate this problem. These methods encouraged us to design an efficient annotation plan to find ducts inside breast biopsies.

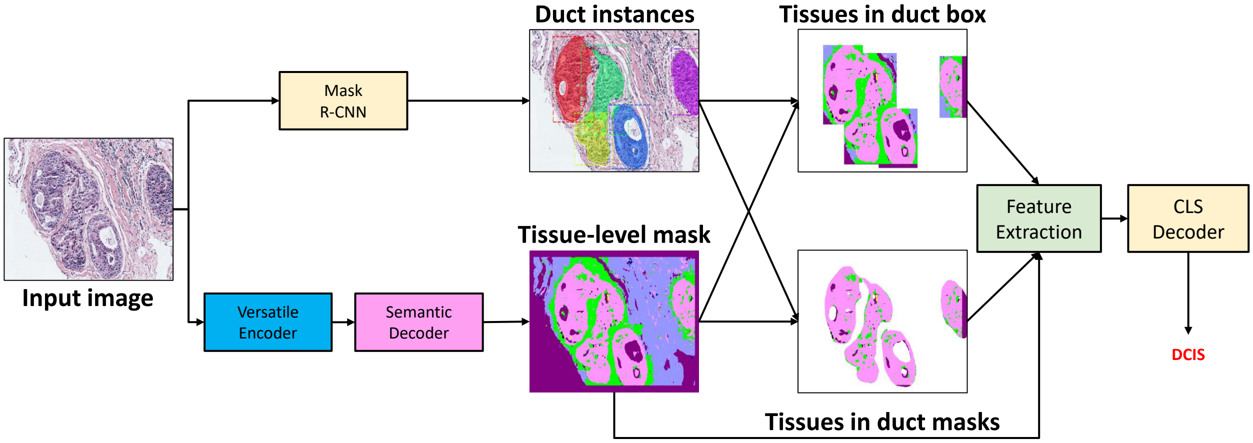


Figure 5.2: Ductal Instance-oriented Pipeline: this pipeline leverages existing versatile encoder, tissue-level semantic segmentation decoder, duct-level instance segmentation decoder, features extraction functions, and a classifier for diagnosis. The feature extractors calculate the tissue histogram frequency and co-occurrence matrix for ROI, box, and mask levels.

5.3 *System for Breast Cancer Diagnosis*

Several clinical studies have shown that stromal tissues and ducts are essential biomarkers for diagnosing breast cancer [89, 113]. Motivated by these studies, we apply UDA and create a machine learning-based framework that accounts for these important biomarkers in cancer diagnosis.

Based on the UDA framework, our system consists of three components: (1) a tissue-level semantic segmentation model with a versatile encoder and a segmentation decoder (Section 5.3.1), (2) a duct-level instance segmentation model (Section 5.3.2), and (3) a classifier with three-levels of extracted features (Section 5.3.3). The input ROI is fed to both duct-level and tissue-level segmentation modules simultaneously to produce instances of ducts and tissue-level segmentation masks. Histogram frequency and co-occurrence features are extracted at three levels: the duct, the bounding boxes, and the entire region of interest to predict the diagnosis. Our experimental results show that the proposed framework outperforms the previous state-of-the-art methods and matches pathologists' performance.

5.3.1 Semantic segmentation

An off-the-shelf segmentation network [96] is applied to generate tissue-level semantic segmentation. Similar to autoencoder structures mentioned in Chapter 4, this network contains an encoder (with multiple resolutions) and a segmentation decoder. The UDA framework absorbs the trained encoder in [96] as the versatile encoder in the diagnostic pipeline; then, it acquires instance-level annotation and creates a duct-level instance segmentation based on the segmentation network.

5.3.2 Duct-level instance segmentation

The instance segmentation network adopts the same structure as Mask R-CNN [106] and Path R-CNN [105]. The network consists of two stages. The first stage takes an ROI as input and produces duct candidates. In the second stage, these candidates are classified as duct or not. In addition to this classification, the second stage also produces bounding box coordinates and a pixel-wise mask of the duct.

To reuse the same architecture from our collaborators in [105], we did not utilize the versatile encoder directly for the instance segmentation task. However, the encoder implicitly impacted the instance segmentation task, because it participated in the weakly-supervised annotation in the paragraph below. Future researchers can redirect the data flow from the versatile encoder to the feature pyramid network and the region proposal network in the R-CNN structure for a more straightforward design.

Ductal regions are essential biomarkers in diagnosing breast cancer. However, collecting duct-level instance segmentation masks is difficult because pathologists must annotate the instances. We created a weakly supervised annotation tool to collect duct-level instance segmentation masks. Figure B.1 shows our annotation tool in the Appendix. We first run the UDA to perform semantic segmentation on ROI images in breast cancer biopsies. Benign epithelium (BE), malignant epithelium (ME), secretion (SC), and necrosis (NC) are tissues that surround ducts. Therefore, the UDA created binary masks by assigning pixels in these tissues as foreground with the remaining pixels as background, as shown in Figure 5.1b. These masks were then combined with the bounding box annotations[1] to produce duct-level instance segmentations. Overall, 4,347 duct instances were marked in 100 ROIs. Note that one bounding box might contain pixels from several ducts, and this annotation strategy can still mistakenly mark pixels from other ducts to the main duct in the bounding box. So, our annotations are only "silver standard" rather than "ground truth" because they are inexact.

5.3.3 Feature Extraction and Classification

Creating a classification decoder for the UDA is the last step for this diagnostic classification pipeline. Several methods (e.g., histogram features, co-occurrence features, and structural features) can extract features from tissue-level segmentation masks. Features from superpixels (regions of similar color in an area) contain encoded information about tissues (e.g., stromal tissue) and structures (e.g., ducts) present in biopsy images, which can improve the diagnostic classification compared to multi-instance learning-based methods [93]. Histogram frequency features can convey the distribution of tissues in an image, and the co-occurrence features can encode uncomplicated spatial relationships. With extracted frequencies from five layers inside and five layers outside of a duct instance, the structure features can capture the changes in the shape of the epithelial structures.

Though the structure features used in [91] would allow capturing architectural information around the ducts, they are computationally expensive as compared to the histogram and co-occurrence features. While computing structure features takes over an hour for each image, computing the proposed features takes only one second. The proposed method leverages the duct-, box-, and tissue-level information based on tissue histogram frequency and co-occurrence frequency. This design allows us to replace these computationally expensive features with simple features at three levels. These features are then fed to a classification

[1] The bounding boxes around ducts were marked by Beibin Li under the supervision of an expert pathologist, Stevan Knezevich.

network (e.g., a random forest or a multi-layer perceptron) to predict the diagnostic category. The ability of our system to aggregate clinically relevant information at different levels allows our framework to outperform existing methods by a significant margin (Section 5.4.4).

5.4 *Experiments and Results*

5.4.1 Dataset

Diagnostic labels: The Ductal Instance-Oriented Pipeline was developed by using digital whole slide images created from residual breast biopsy material [114, 115, 116]. The dataset consists of 428 ROI images, which were extracted from 240 breast biopsies and categorized by three expert pathologists, who selected the ROIs for each whole slide image and agreed on a consensus diagnosis for each slide. Similar to many previous studies on this dataset [95, 91, 117], we performed diagnosis classification for each of the 428 ROIs into 4 classes: Benign, Atypia, Ductal Carcinoma in Situ, or Invasive Cancer. The dataset is unique and is enriched with additional cases in the challenging Atypia and DCIS categories to establish statistical confidence in accuracy predictions for categorical classification. The enriched image dataset provides additional ROI input data on lower prevalence disease categories for developing the deep learning pipeline. The comparison of participants' diagnoses performance with expert pathologists was previously reported [118].

Tissue-level semantic annotations: The dataset also provides pixel-wise tissue-level labels for 58 ROIs. An expert pathologist annotated these ROIs into eight tissue classes: background (BG), benign epithelium (BE), malignant epithelium (ME), normal stroma (NS), desmoplastic stroma (DS), secretion (SC), blood (BL), and necrosis (NC). These ROIs were used to train a tissue-level segmentation model. See [96] for more details.

5.4.2 Implementation details

Duct-level instance segmentation decoder: In the development thread of the UDA, we fine-tuned Mask R-CNN [120] (pretrained on the MS-COCO dataset) with ResNet-50

Task	**Features**	**Sensitivity**	**Specificity**	**Accuracy**	**F_1**
Invasive vs Non-invasive	*Pathologists* [118]	*0.84*	*0.99*	*0.98*	*0.86*
	Superpixel Features [91]	**0.70**	0.95	0.94	0.62
	Structure Features [91]	0.49	0.96	0.91	0.51
	Ours	0.62	**0.98**	**0.95**	**0.73**
Atypia and DCIS vs Benign	*Pathologists*	*0.72*	*0.62*	*0.81*	*0.51*
	Superpixel Features	0.79	0.41	0.70	0.46
	Structure Features	**0.85**	0.45	0.70	0.50
	Ours	**0.85**	**0.63**	**0.79**	**0.59**
DCIS vs Atypia	*Pathologists*	*0.70*	*0.82*	*0.80*	*0.76*
	Superpixel Features	0.88	0.78	0.83	0.86
	Structure Features	0.89	0.80	0.85	0.87
	Ours	**0.91**	**0.89**	**0.90**	**0.92**

Table 5.1: **Diagnosis results for binary classification:** we show sensitivity, specificity, accuracy, and F_1 score for all models. We highlight the best machine performances in this table, and pathologists' performances are provided for comparison. The results for superpixel features and structure features are from [119, 91], where the standard deviations (STD) are not reported.

as a backbone network for 30 epochs using SGD with an initial learning rate of 0.01 and a momentum of 0.9. We used duct-level instance segmentation masks produced using our weakly-supervised annotation tool for fine-tuning Mask R-CNN. Compared to cellular entities, ductal regions are larger to detect with lower resolution images. Therefore, we resized all ROI images to a fixed spatial dimension of 512×512. We split the dataset into an 80:20 ratio: 80 ROIs for training and 20 for validation. On the validation set, Mask R-CNN achieved a mean intersection over union (mIOU) of 72% and a mean average precision (mAP) of 32%. Figure B.2 in the Appendix visualizes duct-level instance segmentation masks produced by

Mask R-CNN.

Tissue-level semantic segmentation decoder: Ductal regions can be identified at lower image resolutions, because the shape and texture can help recognition. However, tissue-level segmentation methods do not perform well at lower resolutions, because low-resolution images may lose information about cellular entities, which help differentiate between different tissues. Similar to [91], the UDA applied the semantic segmentation method [96] to produce tissue-level semantic masks at x40 magnification.

Diagnostic classification decoder: we tried a random forest model, a 3-degree polynomial support vector machine (SVM), an SVM with radial basis function (RBF) kernel, and a multi-layer perceptron (MLP) with four hidden layers (256, 128, 64, and 32 neurons for each layer, similar to [95]) for comparison.

5.4.3 Classification methods

Diagnostic classification tasks and baseline methods are introduced below. We run each experiment 100 times and report the mean performance.

Binary classification: Emulating the successive decisions made by pathologists, [91] performed three binary classification tasks (i.e. invasive v.s. non-invasive, atypia & DCIS v.s. benign, and DCIS v.s. atypia) in their studies. We performed similar experiments, using leave-one-out cross-validation to evaluate each binary classification model. If the number of features was more than the number of ROIs in a classification task, we performed principal components analysis (PCA) to reduce the number of features to 20 dimensions. We applied a weighted random sampling approach to sampling balanced positive and negative samples before training these binary classifiers.

Multi-class classification: We also conducted a 4-way classification experiment to compare our results with previous studies [95, 93]. We used the same training/validation/testing split as Y-Net [95], an extension of U-net with a separate branch for diagnostic classification; their discriminative masks improved classification accuracy by 7% over previous feature-engineering methods. On the other hand, a multi-instance learning method [93], analyzing extracted features from a CNN instead of tissue-level semantic information, outperformed previous methods. We will compare the proposed method with these baseline methods.

5.4.4 Main Results

Binary classification: Table 5.1 compares the performance of our method with the super-pixel and structural features [91] in terms of sensitivity, specificity, accuracy, and F_1 score. Overall, our method outperforms both methods in all binary classification tasks. We observe that the super-pixel-feature-based method delivers the best performance for the invasive vs. non-invasive task. This is because cancer cells spread out from the ducts in invasive cancer (as shown in Figure B.2h). This limits both our method and the structure-feature-based method to aggregate information around ducts, resulting in lower performance. In contrast, the super-pixel-feature-based method only accounts for pixel-level information and not structure-level information. Therefore, such methods are resilient to structural changes.

Multi-class classification: Table 5.2 compares 4-way classification performance of our method with state-of-the-art methods. Compared to these methods, our method delivers significantly better performance. For example, our method is about 7% and 3% more accurate than Y-Net and the multiple instance learning (MIL)-based method. Importantly, our method matches the performance of pathologists on this dataset. These results indicate the applicability of the UDA framework to adapt partially labeled medical imaging datasets.

5.4.5 Ablations

To understand the components of UDA in detail, we perform the following experiments:

Impact of duct-level instance segmentation and tissue-level semantic segmentation: Following [119], we extracted L*a*b, hematoxylin and eosin (H&E), and local binary pattern (LBP) histogram features for the duct-only method. For the other two methods (tissue-only and tissue+duct), we extracted histogram and co-occurrence tissue-level features

Method	**Accuracy**
Pathologists [118]	0.70
MIL with max-pooling [121]	0.55
MIL with learned fusion [93]	0.67
Semantic Learning [96]	0.55
Y-Net [95]	0.63
UDA (Ours)	**0.70** ± 0.02

Table 5.2: Multi-class classification results on the breast biopsy dataset. Our model outperforms existing methods by a significant margin and also, matches the performance of pathologists.

(similar to [95, 96]). Table 5.3 shows that the method that uses both duct- and tissue-level information delivers the best performance.

Ducts	**Tissue**	**Accuracy**
✓		0.57
	✓	0.67
✓	✓	**0.70**

Table 5.3: Impact of duct-level instance segmentation and tissue-level semantic segmentation.

Impact of different features: By aggregating the information about different structures present in the breast biopsy images, previous methods have proposed different features, namely structure features, histogram features, and co-occurrence features. Table 5.4 compares the performance of our method with these different features. Using both histogram and co-occurrence features delivers the best performance. Our method performs well with co-occurrence features alone because the co-occurrence matrix encodes strong relationships between different tissues.

Histogram	**Co-occurrence**	**Accuracy**
✓		0.66
	✓	0.69
✓	✓	**0.70**

Table 5.4: This table studies the impact of different features extracted from duct-level and tissue-level masks (Section 5.3.3). We did not use superpixel and structural features [119], because (1) they are computationally expensive and (2) our method delivers better performance with these simple features (**0.70** vs. 0.66).

Impact of extracting features from different levels: Our framework in Section 5.3 encodes information from three different levels: (1) tissue-level segmentation mask for the whole image, (2) tissue-level mask for duct bounding boxes and (3) tissue-level mask for duct instances. Table 5.5 shows that extracted features from all levels help improve performance.
Impact of classifiers: We study the impact of different classifiers in Table 5.6. Compared to widely used MLP and SVM, the random forest delivered the best performance. This is likely because random forests reduce high-variance by ensembling many trees into one model. This reduces over-fitting and improves performance, especially on small datasets (like ours).

5.5 Discussion

Interpretation: While parameters inside a random forest can be hard to understand, we adapted SHAP [122], a game theory-based approach, for interpretation. After training our diagnosis model, we applied SHAP to interpret the diagnostic decision for each ROI and search for the most important features among all ROIs. Table 5.7 compares the 10 most

Method	**Accuracy**
Tissue in ROI	0.67
Tissue in Duct box	0.66
Tissue in Duct mask	0.69
Tissue in Duct mask + ROI	0.69
Tissue in Duct box + ROI	0.67
Tissue in Duct box + mask	0.69
Tissue (All)	**0.70**

Table 5.5: Impact of extracting features from different levels (segmentation ROI mask, duct mask, and duct boxes).

important features from the UDA and from the tissue-level machine learning model (with 0.67 accuracy) in the ablation experiments.

The BD (boundary of ducts) values in co-occurrence features occur when a pixel is adjacent to the border of a mask or bounding box, which matches the boundary of duct tissues. The UDA identifies two co-occurrence features related to BD as essential features, which is consistent with the intuition of structure features. Even if the tissue-level model can identify similar co-occurrence features, it is unable to use information inside ductal regions; on the other hand, the UDA primarily focuses on duct masks and also bounding boxes, and it only ranks one ROI-level feature among the top-10 most essential features. More clinical studies are needed to verify the consistency of our interpretations with pathologists in the future.

Diagnosis: The UDA outperforms existing end-to-end and feature engineering approaches. For the four-way classification task, the UDA achieves comparable performance to general pathologists. In Section 5.4.4, the general pathologists' diagnostic accuracy is 70% for this unique dataset, which over-sampled DCIS and Atypia cases. In the real-world setting, pathologists diagnosing accuracy is over 92% [123].

Method	Accuracy
SVM (polynomial)	0.62
SVM (RBF-kernel)	0.65
MLP	0.66
Random Forest (UDA)	**0.70**

Table 5.6: Impact of different classification algorithms.

Limitation: Note that our dataset contains only 240 biopsies, and additional studies are needed to examine our method thoroughly. With future improvement in semantic segmentation and instance segmentation approaches, our system has the potential to achieve higher accuracy. In this study, we explored clinically relevant features for cancer diagnosis, but we used two separate networks for duct-level and tissue-level segmentation; in the future, the UDA framework can combine them by directing data flows from the versatile encoder to the feature pyramid network in the R-CNN decoder.

Human-Machine Collaboration: In this study, machine-generated semantic segmentation masks helped the annotator understand breast biopsies and perform manageable annotation tasks. These annotations were then used to train instance segmentation models for diagnosis purposes. More comprehensive studies, such as controlled and counterbalanced human factor experiments, are needed to investigate the effectiveness of this human-in-the-loop design and other types of human-machine interaction, such as educating pathologist trainees.

Future directions: Many questions need to be answered before deploying these CAD systems, for example: the generalizability to different datasets, the interpretability of models, the robustness of application to diverse images sets, and the vulnerability to noise and adversarial attacks. Abnormal breast histopathology and breast cancer are complex and heterogeneous disease processes that still require human experts to supervise the critical

Rank	**UDA (ours)**	**Tissue-level model**
1	BD & BE in duct mask	ME & NC in ROI
2	ME & NC in duct mask	BG & NC in ROI
3	BD & NC in duct mask	SC freq in ROI
4	BE & NS in bounding box	BE freq in ROI
5	BG & NC in duct mask	BE & SC in ROI
6	BE & SC in ROI	ME & NS in ROI
7	ME & SC in bounding box	BE & NS in ROI
8	NC freq in bounding box	NC freq in ROI
9	BE & SC in bounding box	NS & NC in ROI
10	DS freq in duct mask	SC & NC in ROI

Table 5.7: The top-10 important features from SHAP interpretation. **Left:** results from the Ductal Instance-oriented Pipeline (UDA). **Right:** results from the classifier for tissue-level semantic features (aka ROI-level features). Background (BG), benign epithelium (BE), malignant epithelium (ME), normal stroma (NS), desmoplastic stroma (DS), secretion (SC), blood (BL), necrosis (NC). and boundary of ducts (BD) are included for this interpretability analyses.

diagnostic decision process.

5.6 Summary

In this chapter, we applied the UDA, which contains a duct-level instance segmentation decoder, a tissue-level semantic segmentation decoder, three levels of pixel-wise features, and a diagnostic classification decoder, for breast pathology and cancer diagnosis. To train the unique instance-level segmentation model, we adapted weak annotation and human-in-the-loop design to acquire training data with the UDA. The proposed method outperforms previous computer-aided approaches in all diagnostic tasks. It also outperforms general pathologists in 2 out of 3 binary classification tasks and almost matches their overall performance on our unique dataset. The medical application in this chapter only learns from a single breast cancer dataset; we will show how the UDA can adapt from several large datasets in its development thread in the following chapter.

Chapter 6

APPLICATIONS FOR FACIAL EXPRESSION RECOGNITION

The previous medical imaging application utilizes several segmentation decoders in the UDA for diagnostic classification, and this chapter combines regression and classification decoders for autism spectrum disorder (ASD) classification [69]. While researchers have collected several public datasets for affective computing in adults, only a few studies have examined the adaptation of adults' models to children. After collecting data from an iPad application, we utilize UDA and explore the data adaptation quality for children with autism.

During our experiments, informed consent was obtained from parents of all children, and all study procedures were designed in accordance with the World Medical Association Declaration of Helsinki - Ethical Principles for Medical Research Involving Human Subjects as well as in compliance with HIPAA[1] to preserve privacy. The Institutional Review Boards approved these studies of Yale University, Seattle Children's Research Institute, and the University of Washington.

6.1 Autism and Facial Expression

Autism spectrum disorder (ASD) is a neurodevelopment disorder that affects the social communication and behavior of children [124, 125]. According to the Centers for Disease Control and Prevention, one out of 59 children is diagnosed with ASD in the United States[2]. Diagnosing ASD can be difficult because (1) the type and severity of symptoms have a wide spectrum, and (2) the behavior of children with autism is dependent on non-autism-specific factors such as cognitive functioning and age [126]. Facial attributes including expressions

[1] Health Insurance Portability and Accountability Act

[2] `https://www.cdc.gov/ncbddd/autism/data.html`

have been suggested as effective markers in autism-related clinical studies [124, 125, 127, 128].

Convolutional neural networks (CNNs) produce state-of-the-art results for recognizing different facial attributes (e.g., expressions, gender, and action units (AUs)) in the wild [129, 130, 112, 131, 132]. The high accuracy achieved in these recognition tasks can be attributed to large-scale labeled datasets, such as AffectNet [129] and EmotioNet [132], that enable CNNs to learn rich and generalizable representations. However, datasets at such scale do not exist for ASD, making it difficult to apply CNNs directly in the autism field.

In this chapter, we apply the UDA for ASD classification using facial attributes. Along with two widely used categorical facial attributes (facial expressions and AUs) for natural images, our system also predicts two continuous facial *affect* attributes (arousal and valence) that are effective in previous autism-related clinical studies [133]. For simplicity, we use facial attributes to represent *facial expression*, *AUs*, *arousal*, and *valence*. Since there are no publicly available datasets for autism with *all* of these different attributes, we learn representations for these attributes by leveraging two large-scale facial datasets of natural images that are collected in a wide variety of settings, including age, gender, race, pose, and lighting variations. The contributions of this experiment are (1) describing an ASD classification system based on the UDA framework and facial attributes, (2) showing the importance of these facial attributes in improving the performance of our system through statistical analysis, and (3) analyzing single vs. multi-task learning (of the UDA) for facial attribute recognition.

6.2 Related Works in Affective Computing

Facial attribute recognition: With recently curated, large-scale datasets [129, 132, 134, 135, 136, 137], it has become possible to train CNNs for facial attribute recognition. These networks can learn facial representations either independently or simultaneously [130, 138, 139]. Most existing datasets contain annotations for one or two facial attributes. In this experiment, we combine two large-scale datasets [129, 132] and train a model to produce four facial attributes simultaneously for ASD classification.

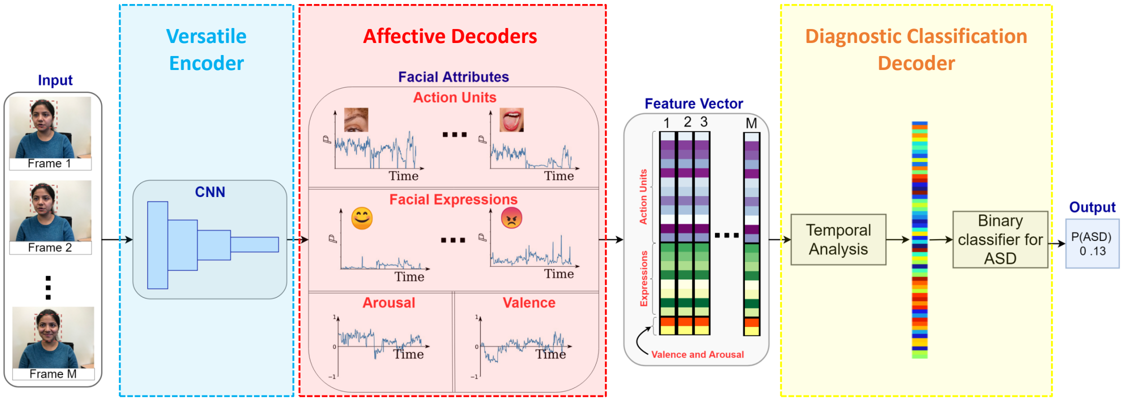


Figure 6.1: Overview of the UDA system for autism spectrum disorder (ASD) classification using facial attributes.

Facial attributes for autism: Various clinical studies have shown that facial attributes, including expressions [124, 125], emotions [127], and morphological features [128], are effective markers for autism. With recent developments in technology, including sensors and artificial intelligence, affective computing is gaining interest in the autism community. Egger et al. [140] use head orientation and expression to study autism-related behavior. Rudovic et al. [133] use facial landmarks and body pose along with captured audio and bio-signals for an automatic perception of children's affective state and engagement. We use representations of different facial attributes for ASD classification in this chapter.

6.3 UDA for Affective Computing and ASD Classification

As shown in Figure 6.1, our framework takes a video as an input and uses a CNN to extract four facial attributes per frame: facial expressions, AUs, arousal, and valence on the face[3]. We created another diagnostic classification decoder to classify children with and without ASD. Outputs corresponding to four facial attributes are concatenated to form a k-dimensional

[3] We use an HoG-based face detector on its excellent trade-off between speed and accuracy on an iPad.

feature vector per frame, represented as $\boldsymbol{f}^t = \{\boldsymbol{f}^t_{au}, \boldsymbol{f}^t_{expr}, f^t_{aro}, f^t_{val}\} \in \mathbb{R}^k$ where $\boldsymbol{f}_{au} \in \mathbb{R}^n$ and $\boldsymbol{f}_{expr} \in \mathbb{R}^m$ are feature vectors corresponding to n AUs and m expressions available in the dataset, while f_{aro} and f_{val} are scalar values between -1 and 1 that correspond to arousal and valence, respectively. We apply temporal feature extraction methods on each vector $\boldsymbol{f}^t \in \mathbb{R}^k$ to extract a single lower-dimensional temporal feature vector $\hat{\boldsymbol{f}} \in \mathbb{R}^l$ per video. Each temporal feature vector $\hat{\boldsymbol{f}}$ is fed to a binary classifier for ASD prediction.

Facial attribute recognition: we are interested in ASD classification. However, no large publicly available datasets provide annotated videos with facial attributes as labels for ASD. Therefore, we use publicly available large-scale datasets that provide some facial attributes for natural images in the wild. We use these datasets to train a CNN-based model that simultaneously predicts different facial attributes. Our network is a standard CNN that learns spatial representations by stacking convolution and down-sampling units, as shown in Table 6.1. Following the previously discussed Equation 4.1 and Algorithm 1, we use the multi-task learning strategy for this affective facial attributes analysis, because abundant pre-training data already exist in publicly available datasets.

ASD classification: After training our model on the publicly available facial datasets, we generate a k-dimensional feature vector $\boldsymbol{f}^t$ for each frame in the participant video by feeding the video into our trained CNN-model frame by frame. These vectors are concatenated to form a feature matrix $\boldsymbol{F} \in \mathbb{R}^{M \times k}$ per video, where M denotes the total number of frames in the video. Due to the temporal nature of the data, there may be redundancies in the feature matrix $\boldsymbol{F}$ that could hinder the analysis of the differences between ASD and non-ASD participants. Therefore, we project this high-dimensional feature matrix $\boldsymbol{F}$ to an l-dimensional vector $\hat{\boldsymbol{f}} \in \mathbb{R}^l$ using temporal analysis methods. In particular, we compute a mean vector $\boldsymbol{m} \in \mathbb{R}^k$ and a standard deviation vector $\boldsymbol{\sigma} \in \mathbb{R}^k$ that contain the mean and standard deviation values across our k features.

In addition, we compute an activation vector $\boldsymbol{a} \in \mathbb{R}^n$ that captures the mean activation time per action unit because of its significance in interpretability. We define $\boldsymbol{a}$ as: $\boldsymbol{a}(i) =$

$\frac{1}{M}\sum_{t=0}^{M}\mathbb{1}_{f^t_{au}(i)>\tau}$ where $\mathbb{1}$ is an indicator function and τ is a threshold. We use $\tau = 0.5$ in our experiments.

Similarly, because it has been shown that the percentages of positive arousal p_{aro} and positive valence p_{val} frames are meaningful for autism related studies [140, 133], we also compute these features. We concatenate the vectors and scalars obtained after temporal analysis to produce l-dimensional feature vector $\hat{\boldsymbol{f}} = \{\boldsymbol{m}, \boldsymbol{\sigma}, \boldsymbol{a}, p_{aro}, p_{val}\}$. We feed $\hat{\boldsymbol{f}}$ to a

	Layer/ Stride	**Repeat**	**Output**	
			Size	**Channels**
Versatile Encoder	Conv-3/2	1	112×112	32
	CU/2	1	56×56	32
	CU/1	1	56×56	32
	CU/2	1	28×28	64
	CU	3	28×28	64
	CU/2	1	14×14	128
	CU	7	14×14	128
	CU/2	1	7×7	256
	CU/1	3	7×7	256
	DWConv-3/1	1	7×7	512
	Avg. pool		1×1	512
Decoders	Linear ×4		$\mathcal{D}_{\text{expr}}, \mathcal{D}_{\text{au}}, \mathcal{D}_{\text{val}}, \mathcal{D}_{\text{aro}}$	

Table 6.1: Overview of the CNN-based UDA architecture. In the four decoders, four linear layers are used as last layer in parallel: $\mathcal{D}_{\text{expr}}$ for expressions, $\mathcal{D}_{\text{au}}$ for action units, $\mathcal{D}_{\text{aro}}$ for the arousal value, and $\mathcal{D}_{\text{val}}$ for the valence value. We note that $\mathcal{D}_{\text{aro}}$ and $\mathcal{D}_{\text{val}}$ are between -1 and 1. Here, Conv-3 and DWConv-3 represent 3×3 standard and depth-wise convolutional layers, and CU represents convolutional unit. We use three different CUs (BottleNeck [120], MobileNet [141], and EESP [142]) in our experiment.

binary classifier to predict if the participant is affected by ASD or not.

6.4 Experiments and Results

We first study the performance of our system on facial attribute recognition on different facial datasets. We then study the impact of each facial attribute on ASD classification along with their statistical significance.

6.4.1 Facial attribute recognition

Dataset: Most of the existing datasets provide annotations for one or two facial attributes. To train a network with all four facial attributes (expressions, AUs, arousal, and valence), we combine two publicly available datasets: (1) AffectNet [129] and (2) EmotioNet [132]. The resulting dataset contains about 1.2 million samples. For AffectNet, we split the training set into two subsets: training (285K) and validation (2.4K). Following [130], we use AffectNet's validation set as the test set (5.5K). For EmotioNet, we split the training set into three subsets: training (754K), validation (63K), and testing (126K).

Training details: We train the UDA for a total of 30 epochs using Stochastic Gradient Descent with a momentum of 0.9 and an initial learning rate of 0.01. We decrease the learning rate by 5% after every epoch for faster convergence. Annotations for facial attributes are different: some are continuous (arousal and valence), and some are discrete (AUs and expressions). Therefore, we use task-specific loss functions to learn representations for different facial attributes. In particular, we minimize cross-entropy loss for expression, binary cross-entropy loss for AUs, and the sum of L1 and L2 loss for arousal and valence, respectively. The multi-task learning follows Algorithm 1. We also use a weighted cross-entropy loss to address the class imbalance for each loss function. We use standard data augmentation strategies such as random flipping, cropping, rotation, and shearing while training our models.

Results: We use CNN-based versatile encoders in the UDA to predict facial attributes for a given input image in both single-task and multi-task settings. In the single-task set-up, the input image was fed to four *different* CNNs, where each CNN predicts a different facial attribute. In the multi-task set-up, as mentioned in previous chapters, the input image was fed to a *single* CNN that predicts all facial attributes at once. Researchers have proposed different CNN units (e.g., bottleneck block in ResNet [120]) in recent years to learn richer representations. To find a suitable trade-off between accuracy and a network's complexity, we study three different convolutional units: (1) the Bottleneck unit [120], (2) the EESP unit [142], and (3) the MobileNet unit [141]. Following the conventions in the literature, we use the following metrics to evaluate the performance of our model: (1) an average of F_1-score and accuracy for AUs [132], (2) F_1-score for expressions [129], and (3) correlation coefficient (CC) for arousal and valence [129].

We make the following observations from the results shown in Fig.6.2 : (1) multi-task learning delivers better performance than single-task learning for all different facial attributes except AUs. In particular, the multi-task learning-based system outperforms the single-task learning-based system for arousal by about 8%, and (2) the EESP units deliver similar performance to the Bottleneck, while the MobileNet units are smaller and more efficient in terms of floating-point operations (FLOPs). The second observation contrasts with other large-scale datasets, such as the ImageNet, where complex models often deliver better performance. This result suggests that facial expression datasets are not as complex as the ImageNet, where deeper models (e.g., [120]) would learn redundant parameters without giving significant performance gains. We note that the recognition performance of our method is on par with existing CNN-based methods [129, 130].

6.4.2 Application to ASD classification

Dataset: We collected a video dataset of 105 children (ASD: 62 and non-ASD: 43) with one video per participant using an iPad application; 88 of these children (ASD: 49 and non-ASD: 39) finished the experiment and then consented to use their data for our research. The diagnostic labels, *ASD* or *non-ASD*, are provided by clinicians based on the neuropsychological tests, which are done independently of these experiments.

CNN Unit	# Params	FLOPs	Expression	AU	Valence	Arousal
			(F_1)	(mF_1Acc)	(CC)	(CC)
Single-task						
Bottleneck	25.9 M	3.4 B	0.56	**0.78**	0.63	0.54
MobileNet	24.8 M	3.1 B	0.57	0.77	0.64	0.52
EESP	9.7 M	1.2 B	0.57	0.76	0.64	0.52
Multi-task with UDA						
Bottleneck	6.5 M	0.85 B	**0.58**	0.75	0.68	0.61
MobileNet	6.2 M	0.78 B	**0.58**	0.75	0.68	**0.62**
EESP	**2.4 M**	**0.29 B**	**0.58**	0.75	**0.69**	0.61
Literature						
SOTA	-	-	0.58	-	0.66	0.54
Human Performance	-	-	0.61	*	0.82	0.57

Figure 6.2: Qualitative performance of single and multi-task learning models in UDA. Here, Expr, AU, Val, Aro, and FLOPs indicate expression, AUs, arousal, valence, and floating points operations respectively.

Each participant watches an expert-designed video stimulus on an iPad during the experiment. The video stimulus is a compilation of short video clips that simultaneously display dynamic naturalistic scenes and social communication scenes on screen, as shown in Figure C.1. These clips are shown simultaneously, side-by-side, on a vertically split iPad screen. While the participant watches a video, our application captures and records the participant's facial response using the iPad's front camera. The video recorded using the iPad application is about 6 minutes and 35 seconds (9,575 valid frames) per participant.

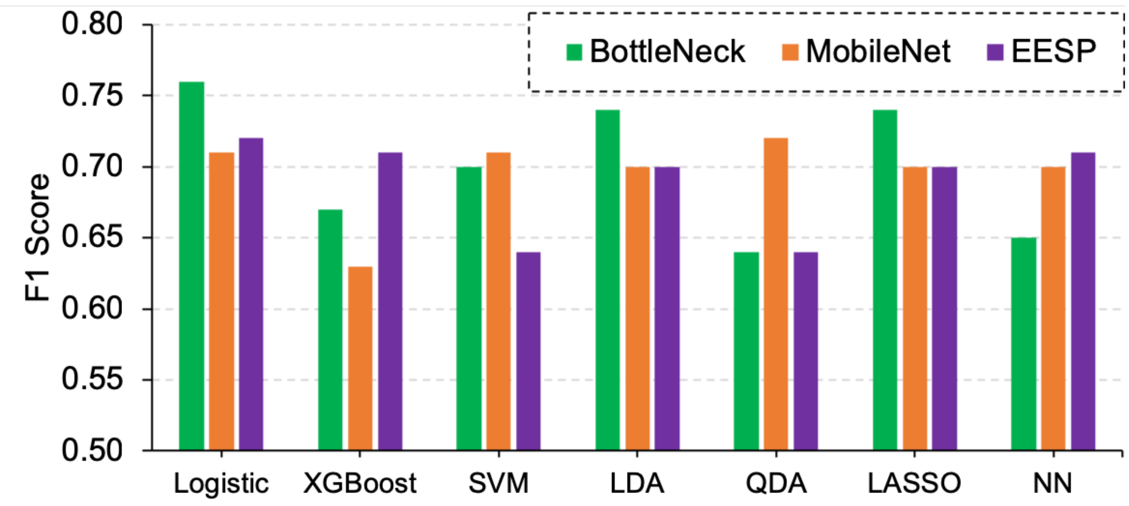
Methods: We constructed a 22-dimensional feature vector from four facial attributes produced by the CNN. The first twelve values in this vector represent the probability of each action unit, the following eight values represent the probability of each expression, and the last two values represent the arousal and valence attributes. The framewise analysis yields a 9575 × 22-dimensional matrix per participant. We then used temporal analysis methods (see Section 6.3) to construct a 58-dimensional feature vector per participant[4]. This feature vector comprises 44 values of means and standard deviations for each dimension (22 × 2), 12 values representing mean percentage activation time of action units, and two values representing the percentage of positive arousal and positive valence. We trained seven binary classifiers (logistic regression, LASSO, LDA, QDA, SVM with RBF kernel, XGBoost, and two-hidden-layer neural network (NN)) using these 58-dimensional feature vectors for ASD classification. Since the dataset is limited, we measured the classification performance (F_1 score, sensitivity, and specificity) using leave-one-out cross-validation.

Results: Fig. 6.3a compares the performances of seven ASD classifiers that use representations from different CNNs. Our system achieved the best F_1 score, sensitivity, and specificity with the Bottleneck as the base feature extractor. We also note that the ASD classification performance improved by 7% when adding features related to arousal, valence, and facial expressions. This result is consistent with our statistical analysis (Fig. 6.3c), where we found these three attributes are the most significant.

6.5 Summary

We applied the UDA for ASD classification using different facial attributes: facial expressions, AUs, arousal, and valence. The multi-task learning approach in the UDA is more effective to classify different facial attributes than the single-task approach. We also showed that representations of different facial attributes used in our experiment are statistically significant and improve the ASD classification performance by about 7% with an F_1 score of 76%.

[4] Temporal features can also be learned using methods such as RNNs and temporal CNNs. The following chapter will show some sample strategies



(a)

Facial attributes				F_1	**Sensitivity**	**Specificity**
AU	**Aro**	**Val**	**Expr**			
✓				0.69	0.69	0.62
✓	✓			0.72	0.71	0.67
✓	✓	✓		0.69	0.67	0.67
✓	✓	✓	✓	**0.76**	**0.76**	**0.69**

(b)

Facial attributes	**p-value**
Action Units (AUs)	0.223
Arousal (Aro)	0.007
Valence (Val)	0.001
Expression (Expr)	0.006

(c)

Figure 6.3: ASD classification results with affective facial attributes: (a) comparison of different binary classification methods, (b) impact of different facial attributes on the classification performance with BottleNeck as a CNN unit, and (c) statistical significance using Student's t-test of different facial attributes.

Chapter 7

APPLICATIONS FOR EYE-TRACKING

To create additional tools to help children with autism, we apply the UDA framework for eye-tracking estimation and attention analyses. Even if collecting eye-tracking data is straightforward for adults, collecting data from uncompliant children is challenging.

This chapter contains two different experiments. In the first experiment [71], we create a gaze estimation model where the underlying model takes a facial image as input and predicts where subjects are looking on the screen; this data collection protocol is similar to the facial expression experiments in the previous chapter. The second experiment [143] creates a deep learning model that takes a gaze scanpath as the signal input and then performs behavior analyses, such as autism classification. In both experiments, we pre-train deep learning models from public datasets and then adapt these models to small private datasets.

Similar to the previous chapter, we follow the World Medical Association Declaration of Helsinki - Ethical Principles for Medical Research Involving Human Subjects as well as in compliance with HIPAA. Our experiments and data collection procedures are approved by Yale University, Seattle Children's Research Institute, and the University of Washington.

7.1 Monitoring Machine Learning's Confidence Toward Novel Data

7.1.1 The Problem

Researchers have examined the feasibility and utility of visible-light cameras for mobile eye tracking for more than two decades [144, 145]. Recent advancements in deep learning have improved eye-tracking quality on portable devices [146, 147], including cellphones, tablets, and laptops.

However, as discussed in previous chapters, data-driven models are often vulnerable to outliers, sensitive to data shifts, and susceptible to increased error when deployed with challenging populations. Eye-tracking technologies should be accessible for people with disorders or disabilities, including individuals with motor neuron disease, dyslexia, epilepsy, and autism. Facial geometry, head movements, and facial expressions are different for special populations, which could hamper the generalizability of deep learning models. Gaze calibration and fine-tuning are common strategies used to combat these issues. However, only a few studies have examined when and how frequently gaze calibrations are needed. Furthermore, real-time gaze error measures, which can inform more advanced online adaptive calibration methods, are underexplored.

In this experiment, we created an iPad eye-tracking application (Section 7.1.2.2), collected 172 hours of data (Section 7.1.2.3) from 62 children (Section 7.1.2.1), applied the UDA framework (Section 7.1.2.4), designed two distinct strategies to predict calibration errors (Section 7.1.3), and provide suggestions (Section 7.1.4) on future eye-tracking studies. This experiment does not aim to outperform state-of-the-art deep learning models. Instead, our main contributions are: 1. showing the importance of creating accessible eye-tracking technology for challenging populations, 2. applying the UDA framework for gaze estimation, 3. illustrating the correlation between facial expression and gaze estimation quality, and 4. providing insights for future mobile eye-tracking designs.

7.1.2 Method

7.1.2.1 Participants

We recruited 62 children (34 with autism spectrum disorder (ASD), 28 with typical development (TD)) as participants for this experiment. Informed consent was obtained from the parents of all children in the experiment. The Institutional Review Board of Yale University approved this eye-tracking experiment.

7.1.2.2 Mobile App and Experimental Setup

To learn about the visual preferences of children with ASD and TD children, we designed an iPad mobile application to record participants' responses by using the frontal camera (30 Hz) while they were watching a series of stimuli on the iPad. Participants used an iPad Air 2 (2013-year model) in Landscape-"Right" mode, where the home button is on the right-hand side, and the camera is on the left-hand side. During an eye-tracking experiment session, parents placed the iPad on a stiff and immobile surface (such as a table or desk) in brightly lit conditions, while the children were seated in a chair with their faces 12-18 inches from the iPad screen.

Calibration stimuli (5 - 10 seconds) and visual preference stimuli (about 75 seconds) together formed a stimulus block for an experimental session. We showed participants four stimulus blocks (about 6 minutes total) in each lab and home session. As shown in Figure 7.3, visual preference stimuli depicted two short video clips (one social and one non-social) side-by-side simultaneously. Calibration stimuli included static 5-point calibrations and smooth pursuit (SP) calibrations. Figure 7.1 shows calibration locations. The five points in the static calibration were presented in one of 10 pseudo-randomized orders.

7.1.2.3 Data Collection

All participants attended lab sessions, and children with ASD also participated in home sessions. Home sessions were expected to occur at least 3x a week for 16 weeks, with manual constraints built into the application, so that sessions could be initiated only once every 12 hours.

7.1.2.4 Eye-Tracking Model

Based on the UDA framework and previous eye-tracking studies [146, 147], we created an Eye-Tracking Convolutional Neural Network (ET-CNN) for gaze prediction. Here, the versatile encoder contains a face convolutional neural network (CNN), an eye CNN, and a mask CNN,

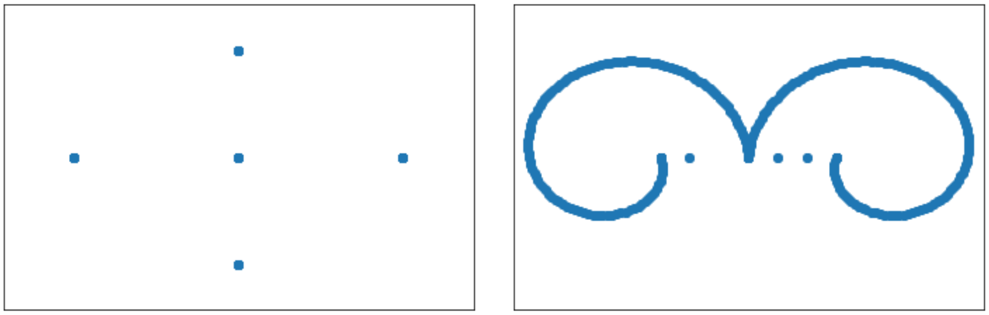


Figure 7.1: Calibration Locations. Top: 5-point calibration. Bottom: Smooth Pursuit calibration.

	ASD	TD	Total
Female	78.12 (14.80)	82.20 (16.38)	80.39 (15.38)
Male	73.12 (16.99)	76.93 (14.26)	74.51 (15.97)
Total	74.29 (16.43)	79.04 (15.04)	74.77 (15.98)

Table 7.1: Age (months) and demographic information for subjects: average and standard deviation (in parenthesis) are shown. There are a total of 62 participants, where 34 of them are diagnosed with autism spectrum disorder; 41 of them are male.

where each of these CNNs produce an embedding vector with 1000 neurons. Then, the ET-CNN concatenates these vectors with six other neurons[1]. The final two fully-connected layers apply the dropout mechanism with a 50% dropout rate, which can mitigate the overfitting problem and can be used for Monte Carlo approximation for error prediction (as discussed in Chapter 4).

For consistency, we resize the face to 200 x 200 pixels, the eye to 100 x 100 pixels, and the mask to 100 x 100 pixels. The Face CNN and the Eye CNN have the same structure: each

[1] a one-hot-vector encoding which represents the device (e.g., iPhone v.s. iPad) and device orientation.

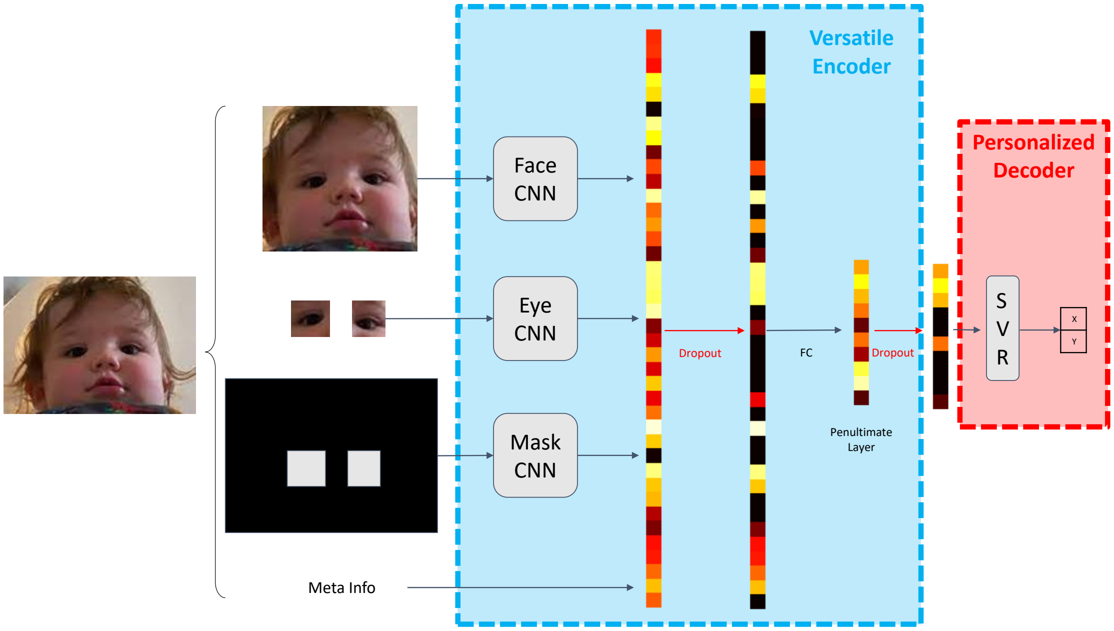


Figure 7.2: The ET-CNN based on the UDA: Deep-learning-based model for gaze estimation. The blue region is the versatile encoder. The red region is a personalized gaze-estimation decoder; it can be either a support vector regressor (SVR) or a fully-connected layer with hinge loss.

CNN contains one initial convolutional layer (with 16 output channels) and four additional residual blocks [120] (with 32, 64, 128, or 256 output channels accordingly). Each residual block contains three convolutional layers with a bottleneck design to reduce the number of parameters and floating-point operations. After the final convolutional layer, a fully-connected layer generates a vector with 1000 neurons. Creating an embedding vector with 256 neurons, the Mask CNN is a small network with only four convolutional layers. These four vectors (from the face, left eye, right eye, and mask) are concatenated together with six additional neurons, with a one-hot vector to encode the device information (e.g., iPhone v.s. iPad and device orientation).

Pre-training from the GazeCapture dataset, the UDA trains the encoder and the initial

Sessions	Number of Sessions	Length (Hours)
Lab	176	26
Home	895	146

Table 7.2: Number of sessions and collected experiment lengths. Each lab session is about 7 - 8 minutes each. The length of home session is based on the user.

regression decoder from the adults data. Table 7.3 shows the performance of the UDA (with the ET-CNN structure) on the GazeCapture dataset: in terms of spatial errors (X, Y, and Euclidian) and accuracy of determining whether gaze is on the left side or the right side of the screen. For assessing Left-vs-Right (L/R) accuracy, we only use ground-truth labels at least 1.5 cm away from the center horizontally for consistency with our iPad application (designed such that stimuli are 1.5 cm away from the center). With the bottleneck design and 1x1 convolutions [120], this ET-CNN performs 14x faster than the originally proposed iTracker model, but this computational improvement is not our main contribution. More efficient (e.g., MobileNet [141], EfficientNet [148]) and accurate (e.g., ViT [149], and other Transformer models [150]) networks could be used to replace the convolutional neural networks in ET-CNN, which is beyond the scope of this experiment. As shown in the following section, we mainly focus on variational inference and gaze calibration errors.

7.1.2.5 Monte Carlo Dropout Option in the UDA

The UDA framework enables us to use Monte Carlo (MC) dropout [151] to perform variance prediction for gaze estimation. The versatile encoder already integrates dropout layers, which operate the same way as traditional dropout [152] during training time. Half of the neurons in the dropout layer will be randomly zeroed during training time, which prevents neurons from adapting too much to the training data.

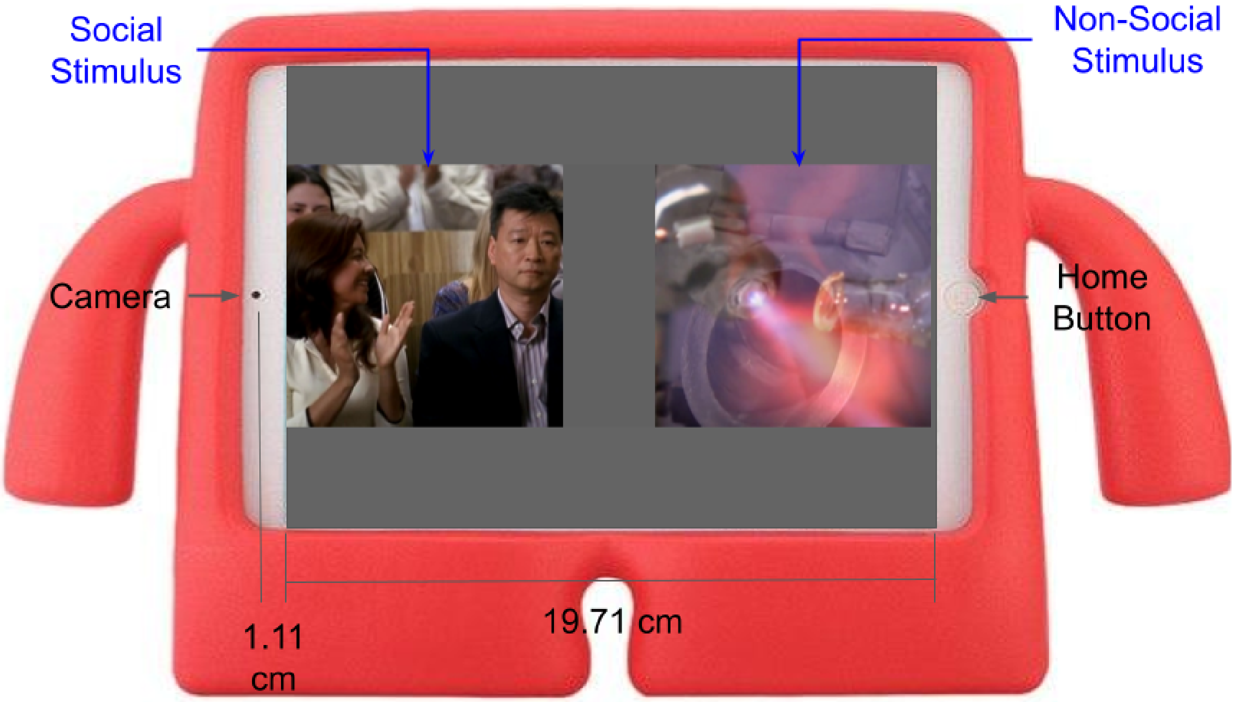


Figure 7.3: The iPad application: a visual preference stimulus is presented on the screen.

Computing Standard Deviations during Inference Time: unlike traditional dropout, the MC dropout does not calculate the expected value for each neuron during inference time. Instead, the MC dropout layer is similar to Gibbs sampling [153] and particle filter methods [154, 155], which perform several stochastic forward passes with different random dropouts. For simplicity, given an input image, we perform ten random stochastic passes and calculate standard deviations from these results.

Computation Overhead: the variational inference method mentioned above only results in an additional $10 \times (3262 \times 0.5 \times 128 + 128 \times 0.5 \times 2) \approx 2.1$ million floating point calculations. Compared to the whole ET-CNN model that costs 90 million floating-point operations to process one image, the computation overhead of this MC dropout mechanism is relatively small.

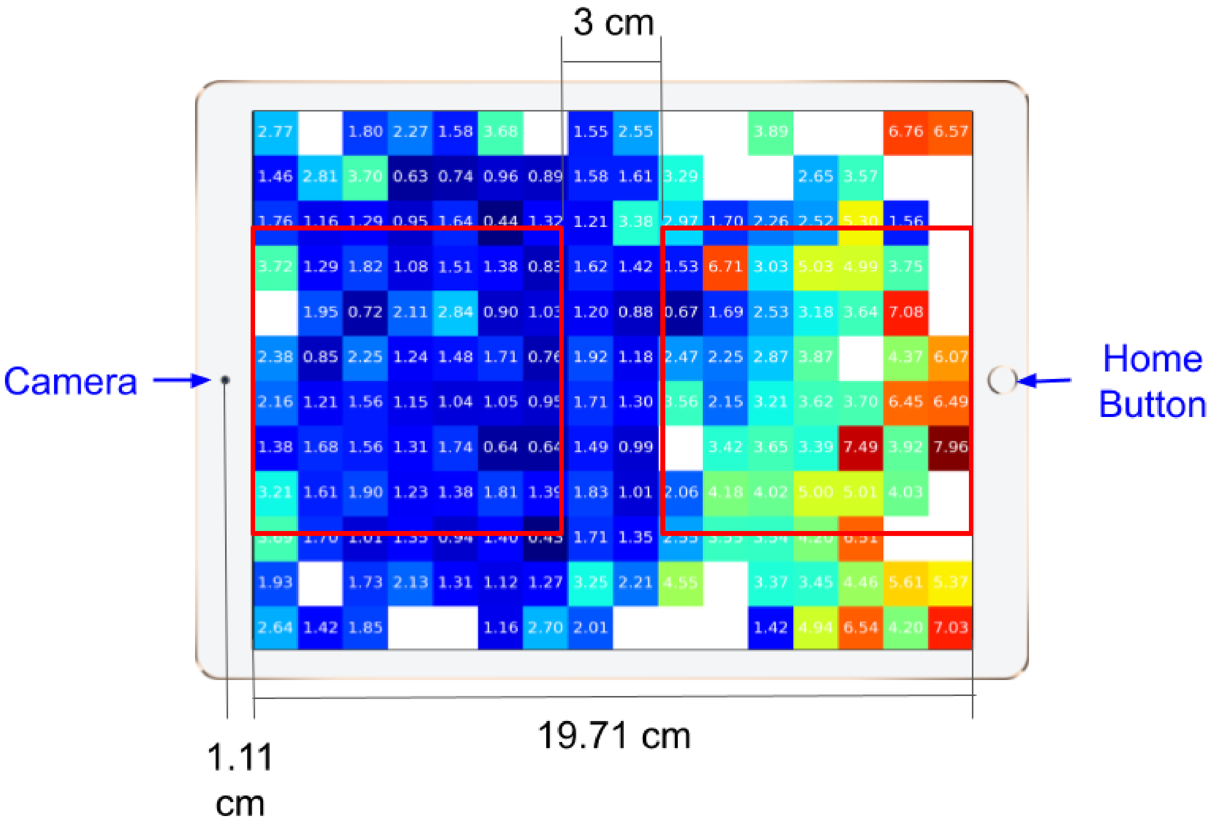


Figure 7.4: Testing errors from the GazeCapture dataset on iPad with Landscape Right mode.

7.1.2.6 Personalized Decoders

We use a linear-kernel support vector regression (SVR) model that learns from features in the last hidden layer (1000 neurons) to perform calibration and train personalized decoders for each subject. This method is similar to linear probing [156]. We examined several different machine learning models (including a mixture of Gaussians, decision trees, random forests, gradient boost trees, and SVR). The SVR performs the best among all models, which is consistent with prior work [147, 146].

Method	# Params	# FLOPs	X Error	Y Error	Euclidean Error	L/R Accuracy
iTracker [147]	6.29 M	1320 M	2.06	**2.12**	**3.27**	0.86
Ours	**2.48 M**	**90M**	**1.90**	2.546	3.53	0.86

Table 7.3: Testing Errors (calibration free) in the GazeCapture dataset on iPad Landscape Right mode, where the home button is on participant's right-hand side. The unit for X Error, Y Error, and Euclidean Error is in centimeters. While maintaining similar performance, the proposed method is about 2.5x smaller and 14x faster than the iTracker model, as shown in the number of model parameters (# Params) and the number of floating point operations (# FLOPs) columns. Both models achieved similar Left-versus-Right (L/R) classification accuracy. The iTracker model is more accurate on vertical predictions (Y-axis), and our proposed model is stronger in horizontal predictions (X-axis).

7.1.3 Analyses: Two Perspectives on Predicting Errors

This section evaluates the UDA's performance in predicting the versatile encoder's confidence on novel data. If we can estimate eye-tracking errors during experiments, researchers can bring interactive real-time calibrations to real-world eye-tracking applications. However, this non-trivial error prediction task, especially for mobile device eye tracking, remains an understudied area of eye-tracking research. Here, we present two different strategies to predict calibration error, and the UDA shows promising results achieving the error-prediction goal.

7.1.3.1 Facial Attribute Perspective

Many researchers have studied how gaze patterns change in response to emotional content [157, 158], but to our knowledge no groups have applied affective computing for eye-tracking quality control. Using the previsouly mentioned facial attributes analysis tool (Chapter 6) designed for children with ASD [69], we analyzed how facial action units (AU), emotion, valence, and arousal could influence eye-tracking errors (Table 7.4a).

The size of the detected face was, as expected, negatively correlated with eye-tracking error, which means larger faces are likely associated with larger eyes and, consequently, better gaze estimation. As shown in Table 7.4a, many facial action units [159, 160] correlated with eye-tracking error. A high proportion of frames with AU 43 (Eyes Closed) would result in higher eye-tracking errors due to increased instability of eye-tracking around moments of lost data. Facial action units 1, 5, and 2 are also associated with movements that result in larger and more visible eyes, which could explain how their increased presence is associated with lower eye-tracking errors. These findings are consistent with the fact that participants often open their eyes widely to improve data quality in eye-tracking experiments.

Neither valence nor arousal is correlated with eye-tracking errors. However, increased presence of the "joy" facial expression correlated with increased eye-tracking error (r=0.10, p=0.019). Joy is associated with facial action unit 12 (Lip Corner Puller) and facial action unit 6 (Cheek Raiser). Raising cheeks has the side effect of diminishing the visible portion of the eye, making gaze estimation more difficult.

7.1.3.2 Monte Carlo Perspective from UDA

We performed similar statistical analyses to characterize the predicted uncertainty with actual gaze estimation error. As shown in Table 7.4b, the predicted standard deviations (σ) from the MC mechanism in the x-axis (r=0.31), y-axis (r=0.35), and Euclidean distance (r=0.41) were highly correlated with actual errors. These associations with gaze error are stronger than those found with facial attributes. The larger the predicted σ, the larger the actual gaze estimation errors are.

7.1.3.3 Predicting Errors

Previous analyses illustrated significant correlations between gaze errors and our extracted features. While the facial attribute features are valuable and interpretable for psychological

Feature	r	Adj. p
Face Size	-0.29	$< 10^{-3}$
43 (Eyes Closed)	0.21	$< 10^{-3}$
1 (Inner Brow Raiser)	-0.20	$< 10^{-3}$
17 (Chin Raiser)	-0.15	$< 10^{-3}$
5 (Upper Lid Raiser)	-0.15	0.001
2 (Outer Brow Raiser)	-0.11	0.001
Joy	0.10	0.019

(a) Correlation for facial expression.

σ of MC Dropout	r	Adj. p
Euclidean	0.41	$< 10^{-3}$
x-axis	0.35	$< 10^{-3}$
y-axis	0.31	$< 10^{-3}$

(b) Correlation (r) from Monte Carlo (MC) Dropout, where the standard deviation (σ) is calculated through MC approximations per frame and then, averaged for a 5-second moving window.

Table 7.4: Two perspectives of calibration errors. In the lab setting, we performed correlation analyses from 24 features to the gaze estimation error, where 20 features are from affective computing, 1 is from face size, and 3 are from Monte Carlo dropout. Highly correlated features are listed in the tables and are ranked by absolute value of correlation. Correlation has a range of [-1, 1], where random guessing has a correlation of 0. Positive correlations mean the more present a feature, the higher the gaze calibration error. Correlations and associated p-values were computed using a repeated measures correlation method in order to avoid pseudoreplication.

Features	RMSE (cm)
Facial	1.03
MC Dropout	0.86
Both	**0.81**

Table 7.5: Predicting calibration error by using either facial attributes, Monte Carlo (MC) dropout, or both. Root mean squared errors (RMSE) in centimeters are reported for the testing participants.

analysis and affective computing, the Monte Carlo features follow an end-to-end design and have a higher correlation with calibration errors.

Then, we use these correlated facial attributes and MC standard deviations to predict eye-tracking data quality. The results are shown in Table 7.5. We used 49 participants to train a linear-kernel Support Vector Regressor (SVR) model to predict eye-tracking errors and used the rest for testing. The Monte Carlo dropout mechanism model outperforms the facial attributes-based model, but their combination performs better than both, suggesting potential contributions of both facial expression and ET-CNN estimation to estimating gaze error.

7.1.4 Adaptation with Gaze Calibration

The previous section suggests that the Monte Carlo dropout mechanism is highly correlated and can predict eye-tracking errors. Here, we further explore practical topics for real-world mobile eye-tracking applications. More analyses, including real-time simulation and rejection-vs-quality tradeoff, are attached in the Supplement material.

7.1.4.1 Error Analyses: Gaze Estimation Quality in Special Populations

In Section 7.1.2.4, we showed that the UDA achieved comparable performance with iTracker in the GazeCapture dataset [147]. However, only a few studies examined the generalizability of deep learning models to other populations. Here, we applied ET-CNN to the collected dataset from children.

As shown in Table 7.5, testing errors are significantly higher in children with autism than in typically developing adults. While the average Euclidean error for adults is 3.53 centimeters on iPad in the Landscape-Right mode, this error for children with ASD rises to 4.96 centimeters in the same mode. Eye-tracking technologies should be robust for different populations, where participants' skin color, age, gender, cultural background, cognitive conditions, pose, and movements could affect the performance of deep learning models. In the future, to robustly deploy deep learning-based eye-tracking models for practical use, researchers might need to evaluate testing errors separately for different populations.

7.1.4.2 Calibration Stimuli: 5 Points versus Smooth Pursuit

Because of high gaze estimation errors for children with ASD, we performed eye-tracking calibrations (Section 7.1.2.6) to the dataset. As discussed in Section 7.1.2.2, there are two types of eye-tracking calibration tasks, and we evaluated them separately with four experiments: no calibrations (None), with only 5-point calibrations, with only smooth pursuit (SP) calibrations, and with both types of calibrations. We evaluated the testing errors with leave-one-out cross-validation.

As shown in Table 7.6, smooth pursuit calibrations are better than 5-point calibrations in terms of Euclidean errors. Using a hybrid calibration stimulus with 5-point and smooth pursuit, we reduced the Euclidean error by about 0.5 to 4.32 centimeters. This result suggests that adding additional calibration content, such as the SP calibration, could increase calibration variety and improve calibration performance.

However, we note that in terms of gauging whether a child is looking left-vs-right (L/R) or

Population	Data	Euclidean	L/R	U/D
	Source	Error	Accuracy	Accuracy
Adults	[147]	3.53	0.86	0.82
TD Children	Lab	4.65	0.82	0.74
Children with ASD	Lab	4.86	0.81	0.68
Children with ASD	Home	4.96	0.74	0.65

Figure 7.5: Testing errors (without personalized calibration) of gaze estimation on different population: we evaluate our deep learning model on different populations: adults, typical developing children (TD), children with autism (ASD). Note that testing subjects never appeared in the training set, and only iPad data with Landscape-Right mode are evaluated for consistency. The performance degrade significantly from adults to children and from lab session to home session.

Calibrate	Euclidean	L/R	U/D
On	Error	Accuracy	Accuracy
None	4.80	0.81	0.71
5-point	4.59	0.81	0.74
Smooth Pursuit	4.47	0.82	0.69
Both	4.32	0.82	0.72

Figure 7.6: Calibrating on different calibration stimuli: 5-points versus Smooth Pursuit (SP). Here, we included all data we collected. We show testing errors by using None (no calibrations), 5-point calibrations, SP calibrations, and both 5-point and SP calibrations.

up-vs-down (U/D), the use of calibration did not noticeably appear to impact performance. The one exception is decreased U/D accuracy when using SP calibrations, potentially due to our specific implementation of SP calibration, which contained limited vertical movement. One reason for the lack of practical impact on dichotomous classification compared to Euclidean improvements is that we discarded more complex cases at the center of the screen to reflect areas of heightened L/R or U/D classification uncertainty. However, these findings also suggest that the utility of calibration methods on mobile device eye tracking may depend on assumptions regarding regions of specific interest and application requirements.

Demonstrating the limitations of data-driven models, we aim to evaluate calibration stimuli' effectiveness rather than create a state-of-art calibration algorithm. This improvement (0.5 centimeters) might be slight for typical adults, but it is still significant for children with disorders.

7.1.5 Discussions and Conclusion

We extended calibration strategies to adapt data shift in gaze estimation. We used the Monte Carlo option in UDA to predict gaze estimation errors and calibrate users' gaze data. The calibration overhead is relatively tiny: gaze estimation error is reduced by 10% with one calibration. We can apply the proposed systems for various eye-tracking tasks in the future.

In this experiment, we collected eye-tracking data for children with autism in both laboratory and home environments. This challenging real-world dataset illustrates the limitation of prior deep learning-based gaze estimation strategies. Based on the UDA, we created a facial attribute framework and a Monte Carlo framework to predict the standard deviation of a deep learning model's estimation, which highly correlates with actual errors. This analysis also suggested gaze error relationships with facial action units and emotional expression, pointing to the need to consider how child affect impacts the eye-tracking data quality more deeply and nuanced. For populations that may react atypically in terms of their affective conveyances, such considerations will be necessary for democratizing deep learning-based methods for eye-tracking studies.

7.2 Learning Oculomotor Behaviors from Self-Supervision

Human oculomotor behaviors are strongly associated with internal mental states. For example, pupil sizes change with cognitive load [161], oculomotor gaze properties change with emotional state [162], and patterns of visual exploration associate with skills such as executive functioning [163]. Because of this diversity of both measure and application, eye tracking has been used as a multi-modal technology to study human cognition in many psychological investigations, including human-computer interaction (HCI) studies. Prior work has spanned from studies of drivers' distractions in transportation [164], to pathologists' mental processes in medical imaging interpretation [165], to consumers' attention towards marketing materials [166]. Most eye-tracking studies are usually task-specific, and their analysis and interpretation typically require human expert knowledge.

The scarcity of labeled data is the primary challenge for researchers seeking to employ more domain-agnostic machine learning approaches in eye-tracking studies. While unlabeled eye-tracking data is plentiful, acquiring human subject data and assigning appropriate labels to this data has a high overhead cost. Annotating eye-tracking data with labels such as cognitive states, psychiatric conditions, or purchasing behaviors requires human reporting and expertise. Moreover, only a few large public eye-tracking datasets are available for researchers, with most eye-tracking studies including fewer than 30 participants [167]. To avoid the burdensome process of manually specifying relevant oculomotor features, data scientists need a more scalable and generalizable approach such that broader and more diverse eye-tracking applications may be enabled.

To address the scarcity of annotated data in eye-tracking studies, we apply the UDA framework with an oculomotor behavior encoder to automate eye-tracking scanpath analysis. As the design of UDA, this model includes a neural network-based encoder to encode properties of human gaze behaviors from scanpath data of arbitrary length and thus can facilitate many eye-tracking data mining applications. In this experiment, UDA leverages four pre-training tasks (pre-tasks) and four corresponding decoders to learn from unlabelled data. We pre-train the UDA on these datasets and conduct proof-of-concept experiments that demonstrate it is robust to data label scarcity. In the future, researchers can use the pre-trained UDA for many different downstream eye-tracking applications, including cognitive analysis, stimuli prediction, and participant classification.

7.2.1 Background: Scanpath Analyses

In desktop-mounted eye-tracking studies, researchers usually set up stimuli on a computer screen and use an eye-tracking device to capture participants' eye movements and the focal location of their gaze. The eye-tracking scanpath is a time series signal discretely recorded as (x, y) screen coordinates of a user's gaze.

Region-of-interest (ROI), saliency, and scanpath analyses are popular ways to analyze eye-tracking data. ROI analysis utilizes detailed knowledge of the presented stimuli; for instance, many eye-tracking studies of ASD have found that children with ASD look less at human faces [168, 169]. Saliency predictions, which also depend on stimuli, try to predict where humans look by analyzing the shown stimulus. On the other hand, scanpath analysis usually does not have access to the stimulus, thus treating the stimulus as a "black box." In this section, we focus on stimulus-agnostic scanpath methods so that the UDA could be generalized to more diverse eye-tracking stimuli and datasets.

Traditional scanpath analyses usually first apply fixation identification algorithms to reduce the scanpath time series to a series of discrete spatially- and temporally-constrained gaze fixations. Then, experts extract various features, such as fixation convex hull shape, fixation speed, fixation duration, and cohesion [170] from the identified fixations. These approaches are usually effective for specific applications but often not generalizable to other eye-tracking studies.

Previous studies have used Hidden Markov models [171], convolutional neural networks [172], recurrent neural networks (RNNs) [173], autoencoders [174] and other machine learning methods to extract features from scanpaths, but most of these studies only focus on a specific machine learning task. Similarly, random forests, SVMs, MLPs, and other machine learning

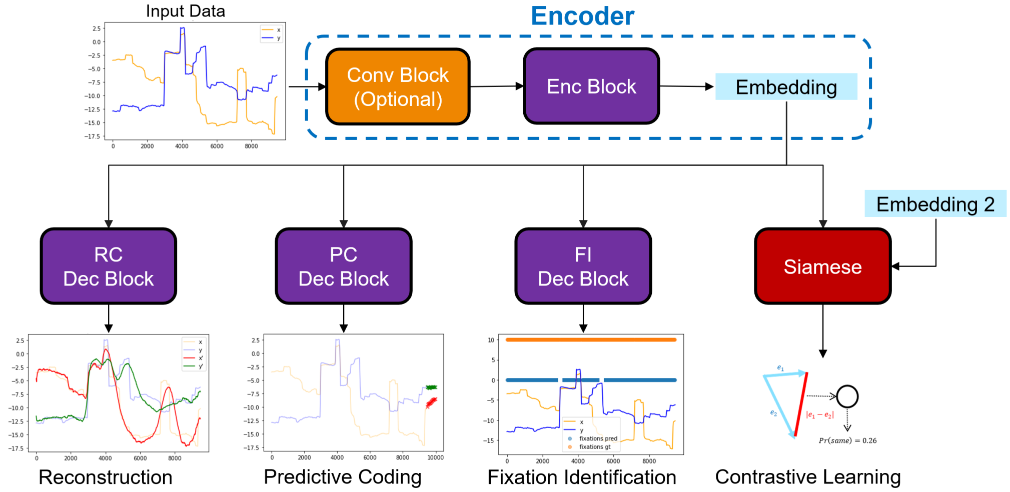


Figure 7.7: Applying the UDA to Learn Oculomotor Behavior from Scanpath. The model is constructed from the blocks inside the blue dashed region. The purple encoder (enc) and decoder (dec) blocks have the same sequential architecture (i.e., RNN, GRU, LSTM, Transformer, etc.).

approaches have been used on expert-designed features. By contrast, we focus on automatic methods to learn useful features for various eye-tracking applications in this work.

7.2.2 UDA for Eye-Tracking Scanpath

Here, the UDA framework includes an encoder and four decoders for four auxiliary pre-training tasks. The pre-task selection process involved eye-tracking experts' suggestions and evaluative machine learning experiments. The four pre-training tasks are reconstruction (RC), predictive coding (PC), fixation identification (FI), and contrastive learning (CL). The RC, PC, and CL tasks use classic self-supervised learning approaches, and the FI task applies supervised learning by automatically acquiring ground truth labels from I-VT [175], a common fixation identification algorithm.

7.2.2.1 The Encoder Network

The versatile encoder in UDA contains an optional convolutional block and a sequential encoder block. These blocks can have arbitrary sizes, dimensions, and architectures. The sequential encoder block can be any recurrent unit (e.g., RNN [18], GRU [176], LSTM [40]) or a Transformer block [41]. If the sequential encoder block is a recurrent unit, the versatile encoder in UDA will concatenate hidden states (and also cell states for LSTM) from all recurrent layers to create the embedding. If it is a Transformer, latent vectors from all timepoints will be used in the sequential decoders, and the last latent vector (for the <End-of-Sequence> token) will be used as the embedding. Modern deep learning applications have widely adopted these structures, and we will not describe them here.

7.2.2.2 Pre-Training Tasks Overview

The decoders for RC, PC, and FI tasks are sequential, meaning that their outputs are sequences of predictions. These decoders have the same architecture as the sequential encoder (i.e., RNN, GRU, LSTM, Transformer). For recurrent encoders and decoders, the forward and backward pass for each task follows the seq2seq design [177]. For Transformer blocks, each task follows the original Transformer autoencoder design [41]. Unlike the above pre-tasks, the CL task does not require sequential prediction. The CL decoder, which deploys the Siamese network design [84], uses the last latent vector as input for classification or clustering purposes.

7.2.2.3 Pre-Task: Reconstruction

Input reconstruction (RC) has been widely used to train autoencoders since the Helmholtz machine [13]. The encoder and the RC decoder learn from recovering 5 - 10 seconds signals. As discussed in previous chapters, the reconstruction loss is the standard mean squared error (MSE) between input and output reconstruction. The RC task forces the encoder to compress important information into a low-dimensional embedding, and then the RC decoder can recover the original input signal. Encoding and decoding signals usually result in smooth signals because the noise in signals only has a negligible impact on the reconstruction loss, as shown in Figure 7.7. This result is the desired behavior because it maintains essential information while neglecting trivial noise.

7.2.2.4 Pre-Task: Predictive Coding

Human brains tend to predict what will happen in the immediate future [20]. Previous studies also indicate that machines may have similar predictive coding (PC) abilities [178] and that learning from predictions of the future can give insight into input data [179]. Similar to the PredNet [21] that performs next-frame prediction in video sequences, the PC decoder use self-supervised learning to predict the oculomotor scanpath in the next 500 milliseconds by analyzing the last 5-10 seconds of the existing scanpath. Predicting future sequences requires the encoder to learn trends and patterns from the input, which is more challenging than simply reconstructing the input signal.

7.2.2.5 Pre-Task: Fixation Identification

Extracting information from fixations and filtering out noise from saccades are thus essential steps in eye-tracking data analyses. Researchers have invented various fixation identification (FI) algorithms to extract fixations, a process specific to gaze studies and not commonly employed in other signal processing application domains. UDA approaches FI as a semantic segmentation task without human annotation. The ground-truth labels are approximated from the I-VT (identification with velocity threshold) algorithm [175], which uses gaze velocity with expert-defined thresholds (i.e., 100 visual degrees per second, 200 ms minimum fixation length) to separate fixations and saccades. UDA performs binary classification (identify “fixation” or “saccade”) for each timepoint in the scanpath sequence. This FI task can help the UDA learn the concept of fixations and extract relevant features from the input scanpath sequence. This information is usually crucial for eye-tracking studies to understand human cognitive states.

7.2.2.6 Pre-Task: Contrastive Learning

Inspired by BERT [4] and related studies, we designed the CL decoder to decide if two scanpath segments come from the same scanpath. In a minibatch of scanpaths, we randomly cut small segments from the original scanpath, where each segment is about 20% - 40% of the length of the original scanpath. The CL decoder thus provides access to oculomotor behaviors that distinguish one scanpath from another.

As discussed, the CL decoder uses Siamese network architecture to encode scanpath segments x_1 and x_2 to two embeddings. Then, the CL decoder calculates the absolute distances for each dimension between the two segment embeddings, which are fed into a multi-layer perceptron (MLP) for classification.

7.2.2.7 Pre-Training Dataset

Merging datasets is a standard strategy in modern deep learning studies for expanding and diversifying training datasets. Following this strategy, we assemble a multi-source eye-tracking dataset with different types of presentation stimuli, experiment length, eye trackers, monitors, hardware setups (e.g., with/without chin rest), and participants. We pre-train UDA on three datasets (Table 7.6): the Lomonosov Moscow State University (MSU) study collected eye-tracking data from 48 participants, each of whom watched 41 video sequences; the Coutrot & Guyader (C&G)-1 study recruited 72 participants to watch 60 stimuli in four different audio-visual conditions, where 18 participants watched each audio-visual condition; the C&G-2 study recruited 40 participants in 2 conditions, where 20 subjects watched each condition. We chose these datasets because they recruited many participants from various setups, which could enrich the diversity for our pre-training stage.

To the best of our knowledge, no participants in the pre-training data appeared in the downstream application. During batch processing in the pre-training phase, a batch only contains signals from one database so that the CL task could be more challenging for the UDA to learn.

Set	Dataset	Eye-Tracker	Resolution	Monitor	Scanpath Length	# Data
Pre-Train	MSU [180]	SMI iView 1250	1920x1080	-	16 - 38 sec	2,377
	C&G-1 [181]	Eye-Link 1000	1024x768	21 inch	8 - 30 sec	4,283
	C&G-2 [182]	Eye-Link 1000	1280x1024	22 inch	20 - 80 sec	592
Downstream	MIT-1003 [183]	ISCAN RK-464	1280x1024	19 inch	3 sec	15,045
	Autism	Eye-Link 1000+	1920x1080	22 inch	15 - 23 sec	931

Table 7.6: The datasets used in the pre-training stage and downstream application stage. The monitor resolution size, the length of each scanpath, and the number of scanpaths are listed.

7.2.2.8 Data Representation

To create a general method for all types of eye-tracking data, we pre-process the data by using the following criteria. All signals are resampled to 60 Hz with bilinear interpretation because previous studies suggest many oculomotor behaviors could be analyzed within 60 Hz [184, 185]. If the gaze data is binocular, the left and right eye signals are averaged. The screen center coordinates are assigned to (0, 0). Coordinates (x, y) represent horizontal and vertical locations, respectively. The coordinate unit is normalized to visual degrees because pixels are not meaningful across different eye-tracker setups. Missing data due to equipment failure or blinks are filled with bilinear interpolation. However, when more than 50% of data are missing in a scanpath, the whole scanpath is discarded. Gaze points more than ten visual degrees off-screen are marked as an extreme value (i.e., -180 degrees); this usually indicates the participant is distracted during the experiment.

7.2.2.9 Pre-Training Engineering Details

Unless otherwise specified, we use the following UDA encoder, which we tuned to balance runtime and performance:

- The UDA encoder contains a convolutional layer (with kernel size 7 and 30 output channels), a leaky ReLU activation layer, a residual connection to the input, and an average pooling layer (with kernel size 2).

- The UDA encoder uses a 2-Layer GRU unit with 128 hidden neurons in each layer.

- The RC, PC, and FI decoders in the UDA have the same architecture as the UDA encoder's sequential block (i.e., 2-Layer GRU unit with 128 hidden neurons in each layer).

- The CL decoder has a hidden layer with 128 neurons, a sigmoid activation layer, and a batch normalization layer.

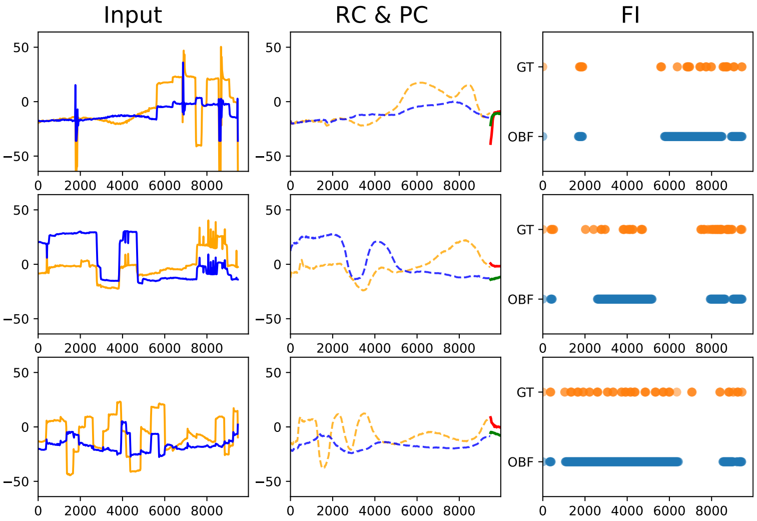


Figure 7.8: Results from the Pre-Train Stage: the proposed encoder encodes the input signal (with size $\sim$ 8 KB) to an embedding vector (with size 0.5 KB). We show the input signal (x signal in orange and y signal in blue) in the first column, the reconstructed signal in the second column, and the predicted future signal in green and red lines in the second column. The third column shows the saccades from the fixation identification (FI) task.

We pre-train the UDA for 500 epochs with a learning rate of 0.001. The learning rate is halved every 100 epochs. The gradient for each neuron is clipped to 0.5 to avoid the gradient exploding. We set the mini-batch size to 64 based on computational efficiency considerations.

We use 80% of the scanpaths for training, and the rest for validation for the standard supervised learning experiment. The following metrics are calculated for the validation set: for the RC and PC tasks, we use mean Euclidean distance as the evaluation metric; for the FI task, we use Area-Under-ROC-curve (AUC) score because the testing labels are unbalanced; and for the CL task, we use accuracy;

Figure 7.8 shows an example of inputs and outputs from the pre-training stage. A detailed analysis of UDA is given in Section 7.2.5.UDA's pre-task results are shown in Table D.3 and Table D.4 where they are compared to results of educated guessing (mean for regression tasks, or majority for classification tasks) and state-of-the-art methods. We re-implement the fully convolutional variational autoencoder (C-VAE) described in [172] and use it with the proposed UDA pre-training tasks. UDA performs well on all four pre-tasks even with a small embedding vector.

7.2.3 Results

Our novel pre-training methodology combines both self-supervised and supervised approaches so that the UDA can learn to encode important oculomotor behaviors from unlabeled scanpath data. The UDA can predict which stimulus a user watched, and it is 1.1 to 2 times more effective with the proposed pre-training tasks. The extracted features from the UDA can also be used directly to classify children with and without autism. In the future, data analysts can use the pre-trained UDA directly in their cognition, psychology, marketing, and other studies that involve eye-tracking technologies. Moreover, this section showed the potential autoencoders have to address data shift problems.

7.2.4 Adapting for Deployment: Example Applications

To illustrate the low-resource data adaptation in UDA, we conduct experiments on two downstream applications - stimulus prediction and autism classification. The pre-trained UDA achieved promising results during supervised learning and transfer learning in both downstream applications. We also conduct meta-learning experiments for the stimulus prediction application, where the UDA can work with more advanced learning techniques to boost classification performance.

7.2.4.1 Stimulus Prediction

We use the MIT-1003 dataset for this stimulus prediction experiment, where we would like to predict which stimulus the user was watching based on her/his gaze scanpaths. Originally designed to study saliency maps of images, the MIT-1003 dataset has 15,045 scanpaths. The large number of scanpaths in this dataset can help us evaluate the pre-training effects in detail. In total, 15 users participated the experiment, with each participant watching 1003 stimuli (i.e., images) for 3 seconds on each stimulus. This experiment illustrates how pre-training and fine-tuning the UDA could improve downstream application performance when the number of users in an eye-tracking experiment is low.

We use a classification decoder (i.e., a multi-layer perceptron) for the standard supervised learning experiment. The input signal first feeds into the UDA (corresponding to the dashed blue box in Figure 7.7); then, the MLP predicts which stimulus the user was watching based on the embedding calculated by the UDA. The MLP contains 2 hidden layers (with 256 and 512 neurons each): for each hidden layer, we add a dropout layer (with 0.5 probability), sigmoid activation, and batch normalization.

We also performed a metric-based meta-learning experiment with a prototypical decoder [82] for this downstream application. We reserved 200 stimuli for meta-training because most eye-tracking studies involve fewer than 200 stimuli (as shown in [167]). We train all models for 100 epochs with 100 iterations per epoch. The model used in this experiment has the same structure as the one in the supervised learning experiment, and we use 128 neurons as the embedding space for the metric-based meta-learning.

For both the supervised and metric-based experiments, we compare the UDA model with the C-VAE model (with the same structure as proposed in [172]) and a baseline method that is not pre-trained in the UDA framework. To better understand the effect of pre-training, we perform c-way k-shot classification experiments to learn from c stimuli and k users, where $c \in [10, 100, 1003]$ and $k \in [1, 3, 5, 10]$. The pre-trained UDA model performs 1.1 - 2 times better than the baseline in all the settings. This result suggests that UDA may be of benefit to the training of future eye-tracking applications, despite those applications being developed from data from only a few participants.

7.2.4.2 Autism Classification

To further illustrate the usefulness of UDA 's learned representations, we apply UDA to a challenging real-world eye-tracking dataset acquired from children with and without autism spectrum disorder (ASD). Researchers can extract embeddings from the UDA and then apply traditional machine learning approaches directly without fine-tuning theUDA.

In total, 49 children from this experiment were included in the examined dataset, with thirty-eight of the children diagnosed with ASD and the rest typically developing (TD) children. These children watched 19 stimuli on a 22-inch screen monitor in a room together with their parents and an experimenter. The monitor-to-head distance is about 65 centimeters, and the whole experiment takes about 11 minutes. The medical ethics Institutional Review Boards approved the data acquisition protocol from Yale University and the Seattle Children's Research Institute. As appropriate, eye-tracking data was collected under parents' and participants' consent and assent. Data handling was compliant with Health Insurance Portability and Accountability Act (HIPAA) and institutional data security guidelines. The experiment protocol follows Good Clinical Research Practice (GCP).

We want to classify the participants into two groups (ASD v.s. TD) by analyzing their scanpaths. For a given participant's data, the UDA first extracts the embedding for each scanpath and then concatenates all the embeddings together to form a final long embedding. We use the lasso for the classification task because the lasso's L1 regularization encourages the model to ignore redundant features from the long embedding vector.

We compare the features extracted from the pre-trained UDA with expert-guided features. The expert-guided scanpath features include the number of fixations, total fixation duration, saccade speeds, and average fixation speed, all of which have been used in previous autism research (e.g., discussed in [186]). We also compare these results with our previous state-of-the-art SGIN [187] method, which employs deep neural networks on features extracted from traditional ROI and scanpath analyses to provide state-of-the-art performance benchmarks in the autism classification task.

We use 5-fold cross-validation to evaluate these features, as shown in Table 7.7. The representations learned by the UDA outperform the expert-guided features by a large margin. The UDA shows promising results even though it employs fewer stimuli and participants compared to [187]. Table D.1 shows more detailed analyses on how the pre-tasks could influence UDA's performance on autism classification.

Method	Accuracy	AUC	F_1
Expert Features from [186]	0.74	0.68	0.82
SGIN (with more data) [187]	0.78	**0.83**	0.83
Ours (UDA)	**0.80**	**0.83**	**0.88**

Table 7.7: Results for Autism Classification

7.2.5 Discussions and Ablations

Here, we perform ablation studies on different backbone units, model sizes, pre-tasks, and pre-training datasets. For the following analysis, we evaluate the inference runtime on an

Nvidia GTX 1080 for 600-length (10-second) scanpaths. We evaluated 1450 scanpaths without batch processing and reported the average runtime in milliseconds. Future studies can trade off the runtime and performance for their specific needs.

7.2.5.1 Effects from UDA Backbone

We validate the RNN, the GRU, the LSTM, and the Transformer for the sequential encoder and decoder blocks for the UDA (Table D.3). The GRU and the LSTM perform well on all pre-tasks, and the GRU is more memory-efficient than the LSTM. Standard RNN units perform the worst among all tested backbones. In our pre-train experiments, the input sequence can contain over 600 timepoints (10 seconds of data), which might cause the vanishing of gradient problems inside the standard RNN unit. The memory gate inside the GRU and LSTM can help mitigate this problem.

Surprisingly, the Transformer model performs worse than recurrent methods in the pre-tasks. We re-evaluated the Transformer with the MIT-1003 downstream application, and its performance is similar to GRU UDA 's performance. The positional encoding might smooth valuable signals when the hidden dimension is not high enough, while the RC, PC, and FI pre-tasks rely heavily on the position information.

While the UDA method performs similarly on the PC and FI tasks regardless of whether the optional convolution block is included, adding the convolution block is still advantageous because it makes the recurrent block more efficient. The convolution block halves the number of time-steps required and thus significantly reduces the UDA's runtime.

While the C-VAE [172] has the fastest inference runtime, its performance is slightly lower than the GRU and the LSTM units even after we control for model size (Table D.4, first row). Additionally, the C-VAE could not encode various-length scanpaths into fixed-length embeddings, which is a potential limitation given that scanpaths can have radically different lengths. Nevertheless, researchers could use the fully-convolutional design for applications with similar-length scanpaths.

7.2.5.2 Effects from the UDA Model Size

The above table shows that small model sizes (e.g., 2 layers x 32 hidden units) can also yield satisfactory performance. Thus, the UDA has potential for future mobile-efficient real-time inference on cellphones and portable devices. Still, larger models can perform better than relatively more minor models. Increasing the number of layers significantly impacts runtime, while increasing the number of neurons has a significant impact on model size. With a two-layer GRU, the number of neurons in each hidden layer has minimal effects on the inference runtime. The marginal improvement from model size diminishes with more than 4 layers or 256 neurons, which could be a limitation from the pre-train dataset size.

7.2.5.3 Effects from Pre-Tasks

To evaluate how each pre-task could influenceUDA 's performance in downstream applications, we conduct the following ablation experiments by removing one pre-task in each experiment. Results from Appendix Table D.1 indicate that all four of these pre-tasks positively contribute to the UDA's downstream performance. For the MIT-1003 stimulus prediction application, the CL and RC tasks contribute the most, while the FI task contributes the least. This phenomenon might be caused by the fact that fewer fixations are presented in the short period (3 seconds) of time, but the locations (predicted in the reconstruction task) and similarities (implicitly learned in the contrastive learning task) are more valuable for this stimulus prediction task. On the other hand, the FI task contributes the most to the autism classification application, which might be caused by the fact that children with and without ASD have distinct fixation behavior and cognitive loads [188].

While all four of the pre-tasks are helpful for future downstream applications, eye-tracking and machine learning researchers could design other relevant pre-tasks to improve the UDA 's performance. We also examined the following pre-training tasks: "data source prediction", "prediction of the number of fixations/saccades", and many others. We found that the data source prediction task was too easy because participants engaged in very different tasks for different data sets (e.g., visual search, video watching), and these tasks were highly distinguishable based on trivial features (e.g., average gaze position). The number of fixations and saccades are highly correlated with the length of the signal, and the UDA framework failed to develop generalized representations of fixations and saccades based on fixation/saccade count prediction. We replaced these tasks with an explicit fixation identification task, which allowed for much richer physiologically-meaningful representations.

7.2.5.4 Effects from Diverse Datasets

Adding more diverse datasets in the pre-train stage could improve downstream applications' performance (details in Appendix). As giant billion-parameter pre-trained models have been released for computer vision and natural language processing in recent years, larger pre-trained models for eye-tracking data could be possible in the future when more public datasets are fed into UDA.

7.2.5.5 Broader Impact

The UDA has promising potential to aid human experts in cognitive analysis, autism classification, healthcare applications, and human-computer interactions with eye-tracking technology. The UDA method could also compress eye-tracking data into a lower-dimension embedding, which can help database systems to store valuable data more efficiently.

However, the UDA could have a malignant impact on privacy if it is used without users' permission or acknowledgment. In the future, privacy-preserving deep learning techniques, such as federated learning, could help alleviate privacy vulnerability. We advocate responsible and human-centered deployments of these deep learning technologies. To the best of our knowledge, all eye-tracking data used in this experiment are agreed upon by the participants, and all these data are anonymized.

7.2.5.6 Limitations and Future Directions

This section uses datasets from desktop-mounted eye-trackers because most physiology studies use similar eye-trackers. However, we believe our approach would be easily applied to head-mounted eye-trackers (e.g., eye-trackers in Google Glass, Oculus).

Focusing on the pre-training methodologies, we show that UDA can perform well even with traditional supervised learning, transfer learning, and meta-learning in downstream applications. In the future, more learning techniques can be tested to improve the performance in these downstream applications further.

7.3 Summary

By using the UDA framework, we created two promising eye-tracking applications in this chapter. While the first application illustrates the effectiveness on detecting data shift with confidence estimation functions, the second application demonstrates the ability in few-shot learning. Even if the medical imaging and behavior analyses applications (such as facial expression and gaze estimation) are post-hoc analyses, they can run in real-time if data streams are available. With an application in database management systems, the next chapter shows a real-time application to detect and adapt data shifts online.

Chapter 8

APPLICATIONS FOR DATABASE OPTIMIZATION

Researchers have explored different methods to estimate the cardinality of a given query inside a table, which is the most fundamental step for database optimization. Even if random sampling performs well for high-cardinality queries, as shown in our previous analyses [189, 190], machine learning-based approaches still have advantages over low-cardinality workloads. However, machine learning-based cardinality estimators are vulnerable to data shifts after deployment.

In this chapter, we apply the UDA framework to adapt learned cardinality estimators in real-time when only limited queries are available [70]. To distinguish this tailored version of the query adaptation system with the general UDA framework, we call this system *Warper*. *Warper* shows the complete life cycle of the development and production threads: the underlying UDA system needs to learn data representation during training and to adapt novel samples during inference. We evaluated *Warper* with four different low-resource data shifts.

We introduce the cardinality estimation (CE) task in Section 8.1, describe the architecture design of *Warper* in Section 8.2, and show experimental results in Section 8.3. All experiments in this chapter perform real-time inference for newly incoming data samples.

8.1 Machine Learning-based Cardinality Estimation

Learned Models (LM) [191] and MSCN [192] are popular machine learning-based cardinality estimators. These models train over a corpus of (predicate, cardinality) pairs for each relation and aim to estimate the cardinality for a given, possibly unseen, predicate. These models suffer the same low-resource problem and data shift problem as previous applications – even if ML models show promising accuracy on predicates that are similar to those used in training, changes to the underlying data or predicates can cause sizable drops in accuracy [191, 192, 193, 194].

Responding to changes in data or predicate workload, prior solutions suggest updating the models by re-training or fine-tuning using newly drawn (predicate, cardinality) pairs [191, 192]. Since these models require training sets of several thousands of queries, re-obtaining the updated training corpus is costly and has high latency. If only the data changes, the ground-truth cardinality labels will have to be recomputed [195, 192]. Worse, when the predicate workload changes, the models may have to wait until enough new queries appear. The net result is that model adaptation is slow and expensive; poor cardinality estimates lead to imperfect plan choices and query performance degradation. Thus, these cardinality estimators encountered the low-resource data adaptation problem.

An ideal adaptation technique should satisfy the following requirements. First, the costs to adapt models should be small, especially relative to the benefits. Next, different scenarios require different adaptations. For example, when only the dataset changes, i.e., the query workload is stable, a key concern is which ground truth cardinality labels to re-obtain to keep costs small when getting fresh training pairs. When the predicate workload changes and only limited examples are available, a key concern is how to generate more examples that mimic the new workload. With the UDA, we take an initial stab at two important questions: (1) when or how often should the model be updated; (2) how to use examples that have possibly inaccurate ground truth and are possibly no longer representative of the arrival predicate workload.

With the various UDA decoders mentioned in Chapter 4, we choose to use decoders for Generative Adversarial Network (GAN) [42, 196, 197]: a generative decoder aims to synthesize examples that are indistinguishable from the observed new predicates, and an adversarial discriminative decoder aims to distinguish the synthetic samples from actually observed samples. We also include a picker (i.e., a special regression decoder) that predicts which input queries are more valuable to update the CE model.

Warper takes a trained CE model and a new set of queries as input. It identifies the nature of the underway drift, learns to generate additional queries when necessary, and updates the CE model. The output of *Warper* is an improved CE model for the new data distribution. Notably, *Warper* is agnostic to and uses the underlying CE model as a black box (e.g., the model can be LM [191], or MSCN [192]). *Warper* does not modify the model's structure and hyper-parameters directly, but it generates predicates for the model to learn. Improvements accrue from generating and labeling useful examples to adapt the CE model quickly.

We are unaware of any prior work that effectively adapts learned database models to data and workload drifts. In our experiments on a typical database server without GPU, *Warper* speeds up model adaptation by many times over solutions from the ML literature while using $< 1\%$ extra CPU utilization (details in 8.3). *Warper* incorporates different learned components to handle different drifts and generates synthetic queries only when necessary. Finally, we show that faster adaptation translates query performance gains by injecting cardinality estimates into a production query optimizer.

Contributions of this chapter are as follows:

- We demonstrate how low-resource data and data drifts can affect trained ML models.
- Based on the UDA framework, we built the *Warper* system, which collaborates with an existing CE model in a non-invasive manner and accelerates its adaptation to new data and workload.
- Experiments show that *Warper* offers a worthwhile cost-speedup tradeoff; at a small cost, *Warper* adapts much faster than fine-tuning and other baselines. *Warper* also generalizes to different CE models and handles a broad scope of data.

8.2 Architecture Design for Adaptation

Warper adapts to drifts periodically. At each period where zero, one, or more kinds of drifts may happen, *Warper* can detect the type of drift (as shown in Table 8.1) that is underway and adapts using the following strategies. To extend data adaptation ability to any kinds of black-box machine learning model, *Warper* takes a CE model $\mathbb{M}$ as an additional component in the UDA system.

- `gen`: When there are inadequate new queries to update $\mathbb{M}$, *Warper* synthesizes additional queries that mimic the new workload.
- `pick`: When labeling cannot keep up or has high costs, *Warper* conserves budget by carefully picking useful queries to annotate.
- `update`: *Warper* updates[1] $\mathbb{M}$ with labeled queries.

Examples. (1) When workload drifts, new queries are inadequate and annotation is slow (`c2+c3` combined), *Warper* generates additional queries and picks which among them to annotate. (2) With adequate labeled queries (`c4`), *Warper* directly updates $\mathbb{M}$.

	Drift	Mitigations in *Warper*			Note
		`gen`?	`pick`?	`update`?	
`c1`	Data	×	✓	✓	Unchanged workload, slow labeling.
`c2`	Wkld	✓	✓	✓	Inadequate incoming queries.
`c3`	Wkld	×	✓	✓	Slow labeling, can happen with `c2`.
`c4`	Wkld	×	×	✓	Adequate queries with labels.

Table 8.1: Individual data and workload (wkld) drifts and low-resource data adaptation strategies. Complex cases are combinations of individual drifts.

System architecture, shown in Figure 8.1, has four key components. Here we denote $\mathbb{I}_{\texttt{train}}$ as the original training workload (i.e., a fixed set of queries) that was used to build $\mathbb{M}$.

- *The query pool* maintains tuples of (`q`, `gt`, z, l, l', s'), in which `q` is a predicate with ground truth cardinality `gt`, l denotes the source of the predicate which can be a prior

[1] Fine-tune or re-train, depending on $\mathbb{M}$; see 8.2.2.

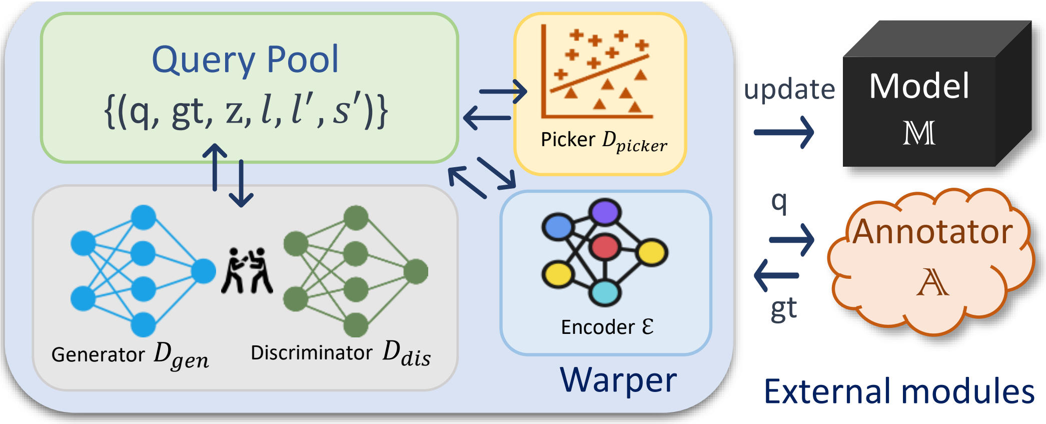


Figure 8.1: Architecture of the *Warper* system.

training workload, the new workload or synthesized (l = `train`, `new`, `gen`) and the other parameters are defined next.

- *GAN decoders* contains the generative decoder $\mathcal{D}_{\text{gen}}$ and the discriminative decoder $\mathcal{D}_{\text{dis}}$; together, they emit synthetic queries. l' is $\mathcal{D}_{\text{dis}}$'s prediction of the predicate source, and s' is the confidence score.

- *The picker* $\mathcal{D}_{\text{picker}}$, as a special regression decoder, borrows ideas from active learning [198] and performs a weighted sampling over a given set of predicates.

- $\mathcal{E}$, as the versatile encoder, can embed predicates `q` into a different space `z` which other components rely on.

- There are two external modules here that *Warper* is agnostic to: (1) the *CE model* $\mathbb{M}$, short for $\mathtt{M}_{X,D,y}$, is the previously trained model that *Warper* aims to improve and *Warper* need not know its structure. (2) The *annotator* $\mathbb{A}$ computes ground truth `gt` for a query predicate `q` and can be a DBMS query or custom code.

Unlike previous chapters' medical imaging and behavioral analyses applications, where only human experts can annotate newly acquired data samples, the DBMS system can run brute force searches to annotate ground truths autonomously for novel data.

8.2.1 Detect, identify, and adapt to drifts

Detect drifts: Prior learned database components use evaluation feedback [199] or blindly update periodically whenever enough new queries are available [191]. *Warper* uses evaluation feedback as follows: `det_drft` triggers when the evaluation error of the current CE model on the newly arrived queries exceeds by more than a threshold π beyond the error observed during training. Thus, *Warper* adapts when drifts cause model accuracy to degrade. We discuss further details, caveats, and corner cases in 8.2.4. If no drift is detected, *Warper* simply uses the current model. The `det_drft` trigger is simple and easy to implement efficiently.

Identify drift modes: Each `det_drft` call also characterizes the drift that is underway as one of the cases in Table 8.1 which we call the `mode` flag; thus `mode` can indicate a data drift `c1` and/or a workload drift {`c2,c3,c4`}. Note that the flag has multiple bits, and more than one kind of drift may occur at a time.

Data drifts. In data drifts, the cardinality labels for all queries (including those from $\mathbb{I}_{\texttt{train}}$) may be outdated. To check for data drift from D to D', we use different measures, including (1) counting the fraction of rows that are new or have changed since the model was last trained and (2) measuring the change in ground truth cardinality for a few canary predicates. A data drift sets the `c1` bit in the `mode` flag. For data drifts, we must re-obtain cardinality labels.

Workload drifts. We identify workload drifts as follows: (1) `c2` denotes the case when newly arrived queries are inadequate; that is, the number of new queries available (n_t) is below γ the number of annotated queries necessary to train a robust CE model. We estimate γ offline based on the training size when the accuracy of $\mathbb{M}$ stabilizes and tune γ online based on evaluation feedback; that is, we use the number of examples at which the error converges in offline training and adjust γ based on the evaluation error during adaptation. (2) `c3`

Algorithm 3: Adapting Data Shifts in Database System: One Invocation

Input: Newly arrived $\{\langle \texttt{q, gt} \rangle\}$, drift `mode` from `det_drft`, $\mathbb{M}, \mathcal{D}_{\text{gen}}, \mathcal{D}_{\text{dis}}, \mathcal{E}$ from the previous invocation.

Output: Updated CE model $\mathbb{M}$ and internal models $\mathcal{D}_{\text{gen}}, \mathcal{D}_{\text{dis}}, \mathcal{E}$ in UDA.

```
pool.append({⟨q, gt, $l$=new⟩})
if c1, c2, c3 ∈ mode then
    if c2 ∈ mode then
        while $n_i$ − − do
            $\mathrm{q_{gen}} \leftarrow \{\mathrm{gen}(\mathcal{D}_{\text{gen}}, \mathcal{E}, \mathrm{pool}, n_s)\}$ ;      // generate $n_s$ synthetic queries
            update_MultiTask( $\mathcal{D}_{\text{gen}}, \mathcal{D}_{\text{dis}}, \mathcal{E}, \mathrm{pool}, \mathrm{q_{gen}}$ ) ;   // see subsection 8.2.3
        end
        pool.append({⟨$\mathrm{q_{gen}}$, $l$=gen⟩});
    else
        update_AutoEncoder($\mathcal{D}_{\text{gen}}, \mathcal{E}$, pool) ;   // see subsection 8.2.3
    end
    anno(pool.pick($\mathcal{E}$, mode, $\mathrm{n_p}$)) ;   // update gt for $n_p$ queries
end
$\mathbb{M} \leftarrow$ update($\mathbb{M}$, pool) ;   // update the underlying CE model $\mathbb{M}$
.
```

denotes the case when not enough queries have ground truth labels: $n_a < \gamma$; this can happen because computing the labels is too slow or too expensive or when a query optimizer has what-if queries but lacks ground truth labels [200]. (3) `c4` occurs when both queries and ground truth are adequate: $n_t > \gamma, n_a > \gamma$. *Warper* is robust to inaccuracies in estimating γ since the drift type detection repeats in each period; *Warper* also uses a form of early stopping to reduce unnecessary resource usage. When there are limited queries to test the CE model, the workload may have already drifted, or the queries may be outliers from the previous distribution. Conservatively, our implementation flags this case to be a workload drift since that lets *Warper* react quickly, and false positives can be identified and addressed in subsequent periods.

Adapting to individual drifts. Algorithm 3 further fleshes out the data adaptation actions. *Warper* injects newly arrived predicates into the query pool, discerns the kind of drift, if any, updates the generative decoder and discriminative decoder if synthetic queries are needed (`c2`). For `c1`, `c2` and `c3`, *Warper* updates the encoder and picks the queries to use for training and annotation. Finally, *Warper* updates the CE model using predicates and labels from the pool for all four cases. We describe details of the modular calls, e.g., $\mathcal{E}$ for `embed`() and $\mathcal{D}_{\text{gen}}$ for `gen`() in subsection 8.2.2 and 8.2.3. It is easy to see that many calls can be parallelized.

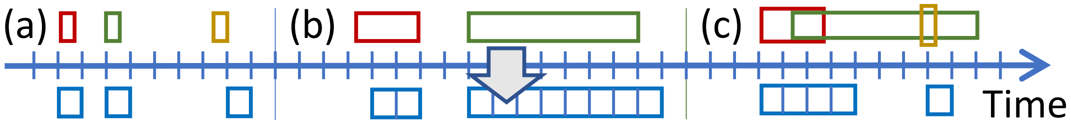


Figure 8.2: Top: different colored boxes show different kinds of drifts. Bottom: we show boxes whenever Warper adapts the CE model; note that Warper periodically evaluates if the model needs to be adapted and handles different kinds of drifts in a unified manner.

Adapting to complex drifts. When a drift discerns multiple modes, i.e., `mode` $=$ `c1|c2`, `c2|c3` and so on, *Warper* combines the mitigation strategies described here; Alg. 3

is capable of handling any combination of drifts. For continuous drifts, as discussed earlier, *Warper* repeats the drift detection and adapts periodically (e.g., in each x-tick in Figure 8.2). The same adaptation strategy is used since Alg. 3 already consolidates joint cases.

8.2.2 Using Warper Components

We describe in detail the design and implementation of individual modules used in *Warper* as shown in Figure 8.1 and Alg. 3.

The query pool is an in-memory data structure that maintains queries and their labels in $\{(\texttt{q}, \texttt{gt}, z, l, l', s'\})$ tuples. *Warper* initializes the query pool with $\mathbb{I}_{\texttt{train}}$ from the original training workload. Each $(\texttt{q}, \texttt{gt})$ tuple from $\mathbb{I}_{\texttt{train}}$ creates a record in the query pool with empty z, l', s' and $l = \texttt{train}$. For each adaptation step that calls Alg. 3, queries from the new workload along with their cardinality labels (if available) are injected into the query pool with $l = \texttt{new}$. The generative decoder $\mathcal{D}_{\text{gen}}$, when necessary, generates and injects new records $\texttt{q}_{\texttt{gen}}$ into the query pool with $l = \texttt{gen}$ and $\texttt{gt}$=-1. *Warper* fills empty fields using different components ($\mathbb{A}$, $\mathcal{E}$ and $\mathcal{D}_{\text{dis}}$).

The CE Model $\mathbb{M}$ is used as a black box and predicts cardinality given a query predicate: $\texttt{q} \rightarrow \mathbb{M} \rightarrow \texttt{card}$. The model exposes the `update()` API, which either re-trains or fine-tunes the model given a set of training examples $\{\texttt{q}, \texttt{gt}\}$ from the query pool. *Warper* aims to improve the CE model without needing to know the model design[2]. Re-trains or fine-tunes depend on individual models. For example, neural network models are iteratively trained and can be fine-tuned, while tree-based models must be re-trained. The initial $\mathbb{M}$ input to *Warper* is trained offline using $\{(\texttt{q}, \texttt{gt})\} \in \mathbb{I}_{\texttt{train}}$. Featurization of $\texttt{q}$ also depends on the specific model; for example in LM [191], $\texttt{q} = \{\texttt{low}_1, .., \texttt{low}_d, \texttt{high}_1, .., \texttt{high}_d\}$ is a vector containing low and high range checks of the query predicate. Such featurization is used among different *Warper* modules and hence *Warper* simply reuses the featurization of $\mathbb{M}$.

[2] To update the CE model requires knowing the input featurization and `update`() even without *Warper*, so we do not consider these as the model design. Instead, we refer to model design to model structures and hyperparameters

Layer	Versatile Encoder $\mathcal{E}$	Generative Decoder $\mathcal{D}_{\text{gen}}$	Discriminative Decoder $\mathcal{D}_{\text{dis}}$
1	Fully Conn. 128	Fully Conn. 128	Fully Conn. 3
2	Leaky ReLU activation	Leaky ReLU activation	
3	Fully Conn. 128	Fully Conn. 128	
4	Leaky ReLU activation	Leaky ReLU activation	
5	Fully Conn. 128	Fully Conn. 128	
6	Leaky ReLU activation	Leaky ReLU activation	
7	Fully Conn. $\lvert z \rvert$	Fully Conn. m	

Table 8.2: Specifications of the learned UDA modules. $|z|$: the embedding size. m: input size to $\mathbb{M}$.

The versatile encoder $\mathcal{E}$, which can calculates embedding (aka, latent vector) from input data sample, is learned to transform a query predicate into a compact representation:

$$\texttt{embed(q)} : q \rightarrow \mathcal{E} \rightarrow z.$$

Recall that *Warper* is agnostic to the CE model and each may use a different featurization, *Warper* leverages learned encoder of query predicates to decouple different components (i.e., $\mathcal{D}_{\text{gen}}, \mathcal{D}_{\text{dis}}, \mathcal{D}_{\text{picker}}$) from the featurization used by $\mathbb{M}$. We found that using predicate embeddings improves the performance of subsequent *Warper* modules. In our implementation, `embed`() uses the ground truth labels (i.e., concatenate to `q`) whenever they are available and up-to-date. Table 8.2 illustrates the model; embeddings need to be updated in each invocation, and the existing one needs to be recomputed.

The Generative decoder $\mathcal{D}_{\text{gen}}$ synthesizes new query predicates using the predicate embeddings in the pool:

$$\texttt{gen(q)} : \texttt{z} + \epsilon \rightarrow \mathcal{D}_{\text{gen}} \rightarrow \texttt{q}_{\texttt{gen}},$$

where $\epsilon \sim \mathbb{N}(0, \sigma^2)$ is a random Gaussian noise with σ that equals to the standard deviation of `z`, the embedding of the predicate previously seen. $\mathcal{D}_{\text{gen}}$ is a simple NN as shown in Table 8.2. Whereas prior generative methods often use random seeds ϵ as input [42], we find that generating from $z + \epsilon$ is more likely to resemble the new workload. $\mathcal{D}_{\text{gen}}$ is trained together with the discriminative decoder $\mathcal{D}_{\text{dis}}$ to formulate a GAN – details follow in 8.2.3.

The discriminative decoder $\mathcal{D}_{\text{dis}}$ is another NN that takes a predicate embedding as input and predicts whether a predicate resembles the training, new or generated workload:

$$\texttt{desc(q)} : \texttt{z} \rightarrow \mathcal{D}_{\text{dis}} \rightarrow l' \in \{\texttt{gen}, \texttt{new}, \texttt{train}\}, s'.$$

The Picker $\mathcal{D}_{\text{picker}}$ uses $\texttt{pick}(n)$ to select n queries from the pool; it finds records in the pool that can be more useful to the model $\mathbb{M}$ and thus reduces the annotation cost. There are two use cases of $\mathcal{D}_{\text{picker}}$ as shown in Alg. 3 and both diversify the selection based on the available information in different drift cases.

- For drift `c2` when $\mathcal{D}_{\text{dis}}$ is used, $\mathcal{D}_{\text{picker}}$ performs a weighted sampling with replacement from all the records in the pool with $l' = \texttt{new}$ based on their confidence score s'; more adversarial queries can be picked hence.

- For drift `c1` and `c3` in which only the picker is used, we apply a sampling stratified by the CE error. Specifically, we first cluster all records in the pool with cardinality labels annotated in the previous invocation into k buckets; the clustering is based on the evaluation error when running query predicates through $\mathbb{M}$. Next, for each new query without cardinality labels, we assign it to one of the buckets based on k-NN upon z. Finally, a stratified sampling with replacement picks records among different cluster buckets. Our simple solutions diversify the choices and pick more high-error queries.

8.2.3 Training Warper Components

There are three components in *Warper* that are learned: the versatile encoder $\mathcal{E}$, the generative decoder $\mathcal{D}_{\text{gen}}$, and the discriminative decoder $\mathcal{D}_{\text{dis}}$. As discussed in Chapter 4, the training process is an real-time and end-to-end process, which is referred as the `update_AutoEncoder()` and the `update_MultiTask()` function in Algorithm 3.

The `update_AutoEncoder()` function follows the training strategy in reconstruction decoders. The `update_MultiTask()` function follows Algorithm 1 to update the versatile encoder, the generative decoder, the discriminative decoder, and the picker (i.e., regression decoder).

8.2.4 Robustness in Warper

In addition to the discussions in Chapter 4, we note a few design considerations, caveats, and corner cases here.

Early stop in *Warper*. We use the accuracy gain of $\mathbb{M}$ after each adaptation step as the stopping criteria; once the gain is less than a small threshold, *Warper* directly uses the previous CE model unless a larger drift is observed in the future. This strategy saves computation resources because possible improvements are already minor at such a moment. Early stopping also adds robustness to inaccurate drift type identification.

Robustness and fall back options in *Warper* are as follows.

Drift detections: (1) False negatives - in a drift when `det_drft` does not trigger (i.e., the accuracy gap of the drift is minor or even negative), *Warper* uses the existing CE model, and no action is needed because empirically there is small accuracy degrade already. (2) False positives, i.e., when there is a large accuracy gap but no drift, is practically impossible.

Drift type identifications. For data drifts, the underlying database system should provide reliable signals. Still, in the event of faulty telemetry: (1) False positives - the bottom line is to recompute ground truth and *Warper* falls back to prior CE solutions [191], which has no negative impact on the model accuracy. (2) False negatives - this is the same as FN in drift detection.

Adaptation intervals and outliers from the new workload. Since *Warper* runs independently to n_t – the number of incoming queries available for each invocation, it is also robust to the adaptation intervals chosen by the users. Indeed, when n_t is small, using a mean error estimation for $\mathbb{M}$ brings in uncertainty for a real workload drift or outliers in the incoming queries, which in turn result in inaccurate control decisions. However, by the early stop and other mechanisms discussed above, *Warper* corrects itself when arrived queries are adequate.

Choosing a large adaptation interval, as shown by the yellow box in Figure 8.2 (a), may cause delayed drift detection. Since `det_drft` has a small overhead, we use frequent *Warper* invocations in practice. Besides, Figure 8.2 (c) shows an example of early stop.

8.3 Experiments and Results

We evaluate *Warper* against state-of-the-art adaptation solutions on a wide scope of drifts. Recall from 8.2.1 that *Warper* adapts periodically to complex and continuous drifts. We also show that faster adaptation translates to query plan improvements. The goals of this section is to show: When adapting different CE models to different types of *individual* drifts, *Warper* outperforms various baselines; *Warper* has a small CPU cost (e.g., about 1% on a typical database server) and adapts faster, reaching a similar accuracy in shorter time. For more join queries and ablation studies, we refer readers to the full work [70].

8.3.1 Can Warper help to mitigate drifts?

Datasets. In our experiments, we use the datasets shown in Table 8.3. These datasets have a wide variety in terms of row and column counts, column types, and distinctness and have been used in prior CE solutions [191, 201]. Further, we leverage the IMDB dataset [202] to evaluate adapting a join CE model.

Table Name	Cols		Rows	Distinct count
	Real	Cat.	n	Min/Medium/Max
Higgs	28	0	11M	3/6.7K/290K
PRSA	16	2	430K	5/645/35K
Poker	0	11	1M	4/10/13

Table 8.3: Datasets for evaluation in experiments. Cat.: categorical.

	Method to generate {`low`, `high`} predicates for column C.
`w1`	Draw from $r(C)$ uniformly at random.
`w2`	Draw from a logarithmic transform of $r(C)$.
`w3`	Equal to a sampled row plus a random width in $r(C)$
`w4`	Equal to $\min(C), \max(C)$ from a sample of k rows.
`w5`	Equal to a stratified sample row by frequency plus a random width in $r(C)$

Table 8.4: Methods to generate workloads. $r(C)$ denote the value range in column C.

Workloads. We have not identified public datasets with realistic drifts. As shown in Table 8.4, we use five methods to generate query predicates which have been applied in [191, 203, 204] to evaluate CE solutions or are simple modifications to existing methods. For example, LM [191] used a mixture of `w1+3` in their studies. Using a large set of workload distributions enables us to better evaluate the generalizability of various adpatation methods. Figure 8.5 demonstrates some workload visualizations using the PCA method [11, 12].

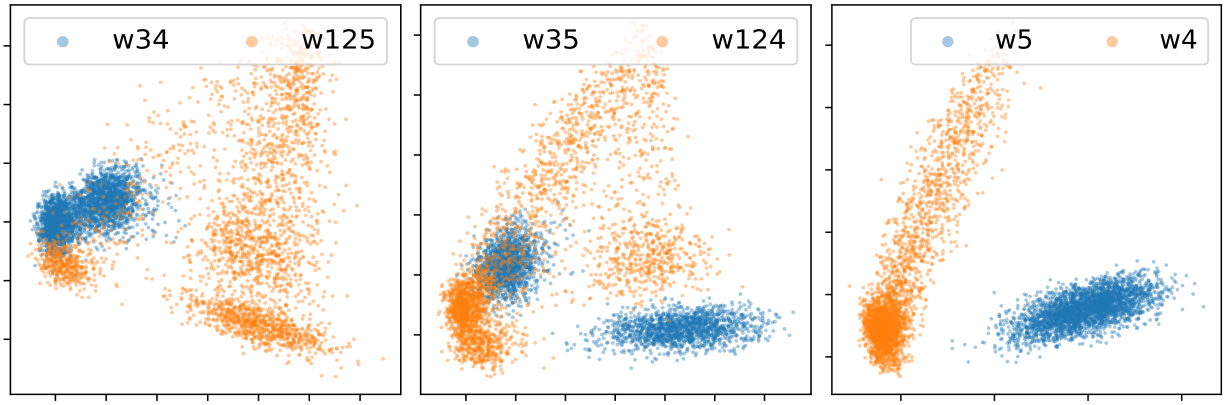


Figure 8.3: Visualizing some workloads on PRSA in our experiments.

Metrics. We measure the following key metrics:

Accuracy: Let $g, \widehat{g}$ be the estimated and actual cardinalities, for each predicate, we measure the q-error: $q_\theta(g, \widehat{g}) = \max(\frac{\max(g,\theta)}{\max(\widehat{g},\theta)}, \frac{\max(\widehat{g},\theta)}{\max(g,\theta)})$. This is a widely used metric [191, 192, 193] so that lower q-error indicates better accuracy and 1 is perfect accuracy; To prevent numeric error, we use $\theta = 10$ to follow [191]. For each test relation, we measure the geometric mean of q_θ over all predicates (GMQ) also to follow these prior work.

Test period and query arrival rate. How fast an adaptation solution mitigates a drift crucially depends on the drift period and the query arrival rate being tested. Hence, we leverage a fixed test time period of 30 mins in our experiments.

Computational overhead. We measure the cost of *Warper* to build and apply different learned components and report the latency aggregated in one thread. For *Warper* and various baselines, the costs consist of the time of data generation, data annotation, and model update. We report average CPU utilization on a desktop-grade database server with 12 cores.

Relative adaptation speedup, agnostic to the test period and query arrival rate, evaluates the effectiveness of a model adaptation solution. For a CE model, accuracy improvements with different numbers of training examples are near monotonic, as shown in Figure E.1. Let α, β, respectively, be the GMQ before and after the drift; we define $\Delta(A, \lambda)$ as the number of queries required for method A to reach an accuracy of $\beta + \lambda(\alpha - \beta)$.

We use a relative speedup $\Delta(FT, \lambda)/\Delta(A, \lambda)$ by comparing the numbers of queries required from the new workload for method A relative to that for fine-tuning (FT).We denote $\Delta_{.5}$, $\Delta_{.8}$ and Δ_1 for short of $\Delta(FT, \lambda)/\Delta(A, \lambda)$ where $\lambda \in [.5, .8, 1]$.

Drift metrics: To measure the severity of a drift, we leverage *blind* and *intrinsic* metrics from the active learning literature [205].

$\delta_{\mathtt{m}}$: Agnostic to the underlying data or workload drifts, $\delta_{\mathtt{m}}$ captures the accuracy gap between an unmodified model and the model trained exclusively on the new data and workload. Such metric is used in `det_drft` (8.2.1) and early stop 8.2.4.

$\delta_{\mathtt{js}}$: We leverage a discrete Jensen-Shannon Divergence [206] to measure the intrinsic distance between two workloads. The $\delta_{\mathtt{js}}$ metric produces $[0, 1]$ in which 0 indicates identical distributions. We use $k = 10$ and $m = 3$ in our experiments.

We also compare UDA (*Warper*) with other baseline methods: data augmentation (AUG), hard example mining (HEM), and mixture model (MIX).

CE models used in *Warper*. We show results with three ML-based estimators in our experiments.

LM [191] models take a range predicate as input and predict its cardinality. The input $\texttt{q} = \{\texttt{low}_1, .., \texttt{low}_d, \texttt{high}_1, .., \texttt{high}_d\}$ represents a conjunction of range predicates on d columns. We use the min and max value of a column to represent one sided predicates. The model is about 64KB in size. In our experiments, we use two variants of LM with MLPs and GBTs, namely `LM-mlp` and `LM-gbt`. These two models have the same input and output; however, `LM-mlp` updates the model by fine-tuning while `LM-gbt` uses re-training. MLP updates with a batch size of 32 and a learning rate of $1\mathrm{e}^{-3}$, while GBT uses a learning rate of $1\mathrm{e}^{-2}$.

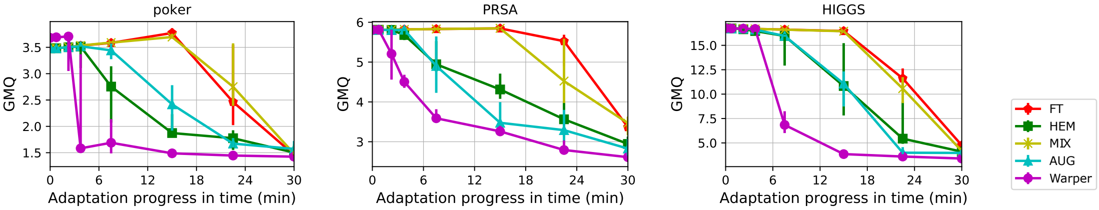


Figure 8.4: Comparison of handling workload drifts (`c2`) using `LM-MLP`. We show GMQ on the hold-out test set at different adaption steps with one new query arriving every five seconds. Table E.1a shows the relative speedups Δ. We plot first and third quarters on the error bar.

MSCN [192] learns a more complex model which uses query predicates, join conditions, and bitmaps as input. The model consists of a pooling layer on each input and an MLP, which produces the cardinality estimates. We use a simplified version here for single-table CE by removing the join condition and bitmap inputs. We use the same predicates and ground truth as LM. MSCN models are 64KB in size and updated using fine-tuning. We use PyTorch for implementation with a batch size of 32 and a learning rate of $1\mathrm{e}^{-3}$.

Evaluation method. We evaluate *Warper* and the baselines with different data and

Dataset		PRSA			Poker			HIGGS		
Method		AUG	HEM	UDA	AUG	HEM	UDA	AUG	HEM	UDA
Annotation cost*		0.01s/query			0.03s/query			0.39s/query		
Model building cost*		-	1s	52.1s	-	1s	60.5s	-	1s	58.5s
Avg	10 min @ 10 q/s	0.27%	0.27%	1.0%	0.74%	0.75%	1.58%	9.95%	9.96%	10.77%
CPU	10 min @ 1 q/s	0.03%	0.03%	0.75%	0.07%	0.07%	0.90%	0.95%	0.96%	1.77%
Usage	30 min @ .2 q/s	0.005%	0.01%	0.25%	0.015%	0.019%	0.29%	0.20%	0.20%	0.47%

Table 8.5: We show additional costs to adapt a CE model. *: Costs in a single thread. Given different arrival rates of new queries, FT requires a minimum of 10-30 mins to fully adapt. Using the same amount of time and newly arrived queries and small CPU in extra, UDA achieves better accuracy than the baselines.

workload drifts `c1-3` in Table 8.1, and the same CE model is used in all methods. For `c4`, *Warper* falls back to FT (8.2.1) and we do not evaluate explicitly. We run each experiment 10 times and report aggregated metrics, including error and adaptation efficiency. We evaluate each adaptation method at 0,20%,..,100% of our test period. n_t is then computed relative to time spent and query arrival rate. *Warper* uses a fixed $n_p = 1K$ in the picker; AUG and HEM randomly sample the same number of queries from different distributions to match *Warper*.

8.3.1.1 Results and discussions

We first examine workload drift `c2` in which all *Warper* components are used, whereas other cases use only a subset (see 8.2). Evaluations of other CE models, types of drift and workload changes will be discussed in 8.3.1.2.

Adaptation speedups. Table E.1a demonstrates model adaptation on three datasets with `LM-mlp`, while Figure 8.4 shows the progress at different adaptation steps. Our observations are two-fold.

First, we note that Δ shown here is already agnostic to and normalized by the query arrival rate. When more queries from the new workload arrive, all adaptation methods improve the accuracy and *Warper* adapt faster than other baselines. We observe large speedups provided by *Warper* on all test datasets. With more accuracy required, the gains are less in general; e.g., Δ_1 decreases to 3.1× on the PRSA dataset. The last bit of accuracy improvements may have to come from the real incoming queries.

Next, MIX occasionally outperforms FT slightly without additional queries used in the model update; HEM and AUG perform better with additional queries but are not as good as *Warper*. These adaptation solutions are ad-hoc and often require careful heuristics design to generate new queries. In practical systems, the use cases of these baselines remain unclear.

Qualitative results. Figure 8.5 demonstrates different sets of queries using the PCA visualization. As the adaptation proceeds over time, we find that the generated (in green) and picked queries (in red) in general follow the incoming query distribution (in orange). A small portion of generated queries near the middle of the diagonal do not follow either old or new distributions; we found that these queries help the adaptation.

Adaptation costs. Here, we measure the costs incurred by *Warper*. Table 8.5 breaks down the costs of *Warper* and alternatives at various query arrival rates. Note that all of these methods update the CE model, which takes a few seconds. FT and MIX do not incur any additional cost, unlike the other methods.

At each adaptation step, *Warper*'s costs contain updates to internal components, generating synthetic queries when needed which takes less than 1 second, and computing the ground truth. To adapt a CE model to the workload drift, *Warper* incurs a compute overhead of about 0.25% to 10.8% CPU usage depending on the query arrival rate. For any arrival rate, a system can choose to (1) adapt immediately, which has the highest instantaneous CPU usage, or (2) spread the adaptation cost over a longer duration. Furthermore, these numbers are from an unoptimized prototype in python; cost reduction optimizations could be helpful in future work. Nevertheless, the costs are already small and insignificant in comparison to the alternatives and do not hold back *Warper* from keeping up with all the

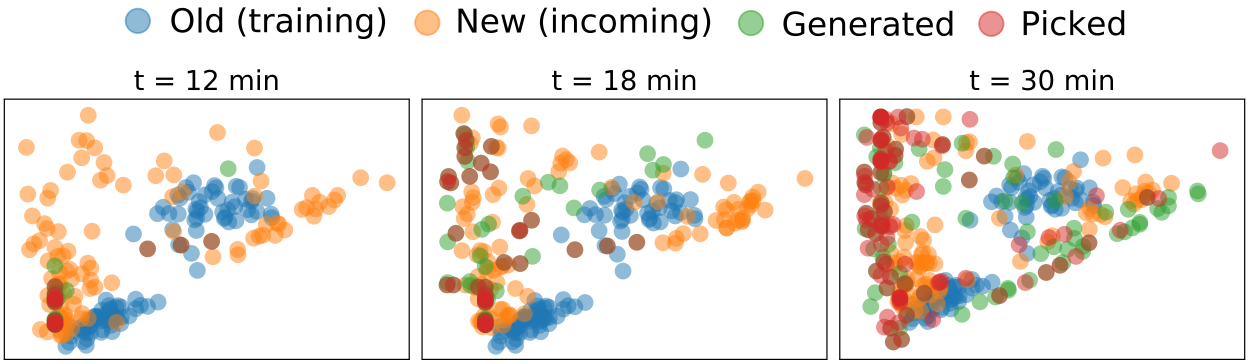


Figure 8.5: Visualization of adaption on the PRSA dataset with `c2` drift and workload `w12/345` at different time (t).

cases in Table 8.5.

Note that when new queries arrive at a higher rate (e.g., 1K q/s in 10 mins) than the cases shown in Table 8.5, *Warper* cannot keep up (CPU usage > 100%). *Warper* either spreads the adaptation in a longer period uese `c3` or `c4` mode by `det_drft` (8.2).

AUG and HEM annotate additional queries and are cheaper than *Warper* but do not adapt as fast as *Warper*. FT and MIX do not use additional queries and are the most efficient solutions; however, they do not offer fast adaptation as *Warper* and other baselines.

8.3.1.2 Generalization in Warper

Beyond the `c2` data shift case shown above, we are interested in how *Warper* can generalize to other types of drifts and workload changes.

Adapting to `c1` Drift: In a data drift `c1`, all labels from the training set $\mathbb{I}_{\texttt{train}}$ become outdated. In this data shifting scenario, *Warper* picks useful queries to label (recall the picker described in 8.2.2) and updates the CE model. Our experiments similarly run periodically; we compare *Warper* with FT that fine-tunes $\mathbb{M}$ by randomly picking the same numbers of queries to annotate from the training set. Table E.1b in the Appendix demonstrates the results. At different accuracy targets, we observe *Warper* has various speedups due to saved annotations.

Adapting to `c3` Drift: In a workload drift `c3`, the newly arrived queries are not companied with `gt` labels. The *Warper* picker works similarly as in `c1`: it picks and annotates queries from the newly arrived queries. At each adaptation step, we compare *Warper* against FT, which uniformly picks the same amount of queries at random for annotation. Table E.1c in the Appendix demonstrates the results, and *Warper* adapts faster with a fixed annotation budget compared with FT on randomly annotated samples.

In both cases above, *Warper* only uses the picker; the overhead caused by *Warper* is only updating the learned modules and is small.

Adapting for different workload changes. We consider the case of `c2` but vary the training and new workloads as illustrated in Table E.2. As shown in the results, different workload distributions exhibit different adaptation speedups with median $\Delta_{0.5}, \Delta_{0.8}, \Delta_1$ being 4.7, 4.6 and 3.7. Notably, speedups are less significant when the accuracy gap is already small (e.g., $\delta_{\mathtt{m}} \leq 0.2$). The accuracy gap in drifts $\delta_{\mathtt{m}}$ can be uncorrelated with the intrinsic distribution difference $\delta_{\mathtt{js}}$, but both metrics are useful to reveal the nature of observed data shifts. CE models that are explainable remain an open question among learned database components. Figure E.2 further demonstrates the adaptation progress on various datasets and query distributions. To this end, we consider *Warper* useful and robust to different distributions of workload drift.

Adapting for join CE. We also examined join cardinality estimation with the MSCN model on the IMDB dataset with an arrival rate of one new query per minute; other settings remain unchanged as above. We randomly generated 16K join queries to pre-train the MSCN estimator with a 128K storage budget. *Warper* achieved $\Delta_{0.5}, \Delta_{0.8}$ and $\Delta_{1.0}$ at 2.1x, 2.8x, 1.1x. That said, we consider *Warper* generic and agnostic to various CE models that need to be adapted.

Remark. It is clear that *Warper* offers a reasonable cost-speed tradeoff and can be a valuable alternative to FT when the cost overhead is affordable. In all test cases and datasets (Table E.1 and Table E.2 in the Appendix), we observe that *Warper* performs no worse than

FT ($\Delta \geq 1$), and applying *Warper* is less likely to cause degradation.

8.4 Summary

In this chapter, we forked the UDA framework and created the *Warper* system to improve previously-trained cardinality estimators in the context of data and workload drifts. The key ideas include generating queries from a generative decoder and picking among the available data samples to annotate with a regression decoder. We show that with small computational costs, UDA has good applicability and accelerates model adaptation to data shifts with only a little computation resource.

Chapter 9
DISCUSSION

As an emerging field, deep learning faces novel and challenging problems every day. So, adapting to the constantly changing world is critical in machine learning deployment. This study does not aim to solve all adaptation problems simultaneously, but we advocate unifying separated data adaptation subareas into a coherent scheme. Our applications of the UDA framework, with various modalities, tasks, backbone models, show promising results to data adaptation in different adaption scenarios. This chapter discusses the limitations and future directions for data adaptation in neural networks.

9.1 Limitations

We admit that this study has several limitations: (**1**) the framework is not fully automated and still requires customization, (**2**) we did not cover other popular domains, such as natural language processing and multi-modality learning, (**3**) this work does not provide strong theoretical guarantees on adaptation performance. Although out of the scope of this study, these critical issues are noteworthy for future researchers.

Our framework can already facilitate many adaptation applications autonomously, and we customized and showcased it for challenging real-world problems. For instance, we included an R-CNN model without delegating the versatile encoder for medical imaging analyses; we connected an autonomous ground truth annotator to a database optimization application – nevertheless, the UDA framework can quickly adapt to standard images and signals without modification. Customizations in Chapter 5, 6, 7, and 8 showed the UDA's flexibility for novel problems.

As an exploratory work to unify data adaptation, the UDA framework does not aim to compete with carefully developed industrial services. This project is not as influential as other open-source projects[1] either, because we focus on solving novel problems for underrepresented datasets rather than well-known public benchmarks. On the other hand, the UDA absorbed many practical designs from other projects so that researchers can adopt UDA quickly. Engineers can also integrate the UDA's design into AutoML in the future.

We showed that the UDA could adapt from source datasets and newly incoming data in computer vision, signal processing, medical analyses, behavior analyses, and database optimization. Many other domains, such as natural language processing and recommendation systems, are not fully covered in this study. However, data adaptation concepts, as discussed in Chapter 3, are universal for different domains. The UDA framework is also capable of handling multiple encoders where each encoder can learn representations from different modalities, and a multi-encoder design can solve more complex problems. These applications are not discussed here because of time and space constraints.

From a theoretical perspective, we introduced two different optimization algorithms to learn various tasks jointly in Chapter 4. We have defined the training loss as $\mathcal{L}$, but the model's generalizability to the target task is understudied in this work. Some recent theoretical studies explored linear probing, fine-tuning, and prompt-tuning for data adaptation, and we refer readers to the more comprehensive survey on adapting foundation models [7]. Nevertheless, strong theoretical guarantees on the adaptation loss $\mathcal{L}_{adapt}$ are still missing. Future work is needed to advance theoretical analyses so that scientists can create better models and algorithms.

Even if some readers might criticize task unification as a hodgepodge of various learning algorithms, integrating different adaptation methodologies is necessary for the robustness and deployment of deep learning models. While researchers have more challenges than solutions for neural network adaptation nowadays, the UDA can be a small but non-trivial step in this extensive unification process.

[1] For instance, Detectron2, fastText

9.2 Federated Learning and Decentralized Learning: Data Privacy

Many machine learning applications rely on collected human subject data, and the UDA framework could reduce the burden to acquire necessary data and labels for new deep learning applications in the future. Collecting fewer data could also help protect privacy and increase data security. We advocate responsible and human-centered deployments of these deep learning technologies.

While most organizations have strict policies for their datasets, adding more data could help build a more generalizable deep learning model. To promote the privacy and enlargement of training datasets at the same time, researchers can extend the UDA framework with federated learning [207, 208] and decentralized learning [209]. The training Algorithm 1 and Algorithm 2 can calculate loss from different datasets, and the UDA can learn distributedly from different machines and organizations.

For instance, in the federated learning scenario, a centralized server only stores a versatile encoder rather than any data. Each member organization can download the encoder, train on its private dataset, and then upload the updated gradient (rather than the data) to the central server for loss aggregation. Moreover, each organization has a different decoder tailored to their small datasets, which can resolve the discrepancies in data collection protocol (e.g., different machine brands, different staining habits). This method is compliant with HIPAA requirements, and many cellphone applications already use this method for spell correction, speech recognition, and face recognition. In the future, researchers can perform more theoretical analyses and scrutinize the underlying encoder to ensure the high preservation of data privacy.

In the decentralized learning scenario, each organization stores the versatile encoder with consensuses parameters. After calculating losses from its dataset, each organization can notify other organizations about the encoder's gradient to update the versatile model consistently across the globe. Compared to the federated learning process, this decentralized method can help avoid monopolistic degeneracies and man-in-the-middle attacks.

9.3 Diversity, Equality, Fairness, and Bias

Deep learning models usually perform well on unseen data but might extrapolate unreasonably in some scenarios. When the data size is small, models would have low confidence and perform poorly for underrepresented groups because of the lack of statistical power. As machine learning practitioners, we should create fair platforms for people from different backgrounds, regardless of skin color, age, gender, sex, sexual orientation, income, occupation, abilities, and other traits.

Recently, some researchers [210] found convolutional neural networks can identify patients' race from chest X-ray images, whereas human doctors could not accomplish such a task. Later, the authors concluded that deep learning models might worsen racial disparities: if a model "secretly used its knowledge of self-reported race to misclassify all black patients, radiologists would not be able to tell." We agree with [210] that future studies should report deep learning models' performances for each demographic subgroup, and we acknowledge that the concern for malicious deep learning models is valid. However, some significant distinctions exist between correlation and causation. The extracted features could indirectly correlate with race and gender when the model learns about diagnostic classification. These robust feature encoding and data disentangling capabilities are the foundations of transfer learning and adaptation methods. Similarly, based on Bayes' theorem, human experts – like unbalanced machine learning strategies – would use biological markers and socio-demographical traits when making diagnostic decisions [5]. Equal data acquisition protocols and model training processes could mitigate the inequality and bias dilemma.

Believing diversity could promote scientific advancements, we created deep learning models for minority groups when only a few resources (e.g., data and labels) were available. For instance, the proposed adaptive affective computation tool could help identify facial expressions for children with disabilities and disorders, removing technology barriers for special populations and other underrepresented groups. The proposed adaptive eye-tracking tool and adaptive cancer analysis tool could help clinicians and patients from developing regions with limited access to healthcare resources. Nevertheless, because of the severity of bias in data-driven models, more comprehensive studies are still necessary to explore alternative approaches for diversity, equality, and fairness.

Given theoretical limitations in current computer science technology, advocating for bringing fairness to deep learning models is an urgent and challenging task. Improving predictive power and increasing fairness could have a symbiotic relationship, but more studies are needed to create novel and practical strategies to solve these dilemmas.

Chapter 10
CONCLUSIONS

Aiming to create a unified data adaptation framework for low-resource learning, we proposed a unified framework, the UDA, that can perform supervised, unsupervised, self-supervised, generative, and active learning to adapt to low-resource data. The versatile encoder in the framework can project a data sample to an embedding vector for adaptation and confidence estimation. Inspired by the prefrontal cortex, we gave the framework flexible abilities to memorize prior source tasks, generate synthetic data, execute target tasks, and determine confidence. These abilities, fulfilled by auxiliary decoders in the framework, allow the encoder to learn data distributions and adapt data shifts more efficiently in low-resource settings. The framework can also help researchers evaluate their datasets and annotate novel data samples. We applied this framework to solve several challenging real-world problems, including medical imaging, facial expression recognition, gaze estimation, signal analyses, and database optimization.

BIBLIOGRAPHY

[1] Pedro Domingos. *The closest thing there is to a free lunch in ML is more data.* en. Tweet. Oct. 2021.

[2] Nithya Sambasivan et al. "“Everyone wants to do the model work, not the data work”: Data Cascades in High-Stakes AI". en. In: *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. Yokohama Japan: ACM, May 2021, pp. 1–15. ISBN: 978-1-4503-8096-6. DOI: `10.1145/3411764.3445518`.

[3] Tom B. Brown et al. "Language Models are Few-Shot Learners". In: *arXiv:2005.14165 [cs]* (July 2020). arXiv: 2005.14165.

[4] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186. DOI: `10.18653/v1/N19-1423`.

[5] Eric Topol. *Deep Medicine: How Artificial Intelligence Can Make Healthcare Human Again*. en. Google-Books-ID: 7iFlDwAAQBAJ. Basic Books, Mar. 2019. ISBN: 978-1-5416-4464-9.

[6] F. Rosenblatt. "The perceptron: A probabilistic model for information storage and organization in the brain". In: *Psychological Review* 65.6 (1958). Place: US Publisher: American Psychological Association, pp. 386–408. ISSN: 1939-1471. DOI: `10.1037/h0042519`.

[7] Rishi Bommasani et al. "On the Opportunities and Risks of Foundation Models". en. In: *arXiv:2108.07258 [cs]* (Aug. 2021). arXiv: 2108.07258.

[8] Hermann Ebbinghaus. “Memory: A Contribution to Experimental Psychology”. In: *Annals of Neurosciences* 20.4 (1885), pp. 155–156. ISSN: 0972-7531. DOI: 10.5214/ans.0972.7531.200408.

[9] E. Bruce Goldstein. *Cognitive Psychology: Connecting Mind, Research and Everyday Experience*. en. Google-Books-ID: 4LI8AwAAQBAJ. Cengage Learning, June 2014. ISBN: 978-1-305-17699-7.

[10] C. E. Shannon. “A mathematical theory of communication”. In: *The Bell System Technical Journal* 27.3 (July 1948). Conference Name: The Bell System Technical Journal, pp. 379–423. ISSN: 0005-8580. DOI: 10.1002/j.1538-7305.1948.tb01338.x.

[11] Karl Pearson. “On lines and planes of closest fit to systems of points in space”. In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (Nov. 1901). Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/14786440109462720, pp. 559–572. ISSN: 1941-5982. DOI: 10.1080/14786440109462720.

[12] H. Hotelling. “Analysis of a complex of statistical variables into principal components”. In: *Journal of Educational Psychology* 24.6 (1933). Place: US Publisher: Warwick & York, pp. 417–441. ISSN: 1939-2176. DOI: 10.1037/h0071325.

[13] Geoffrey E Hinton et al. “The ”wake-sleep” algorithm for unsupervised neural networks”. In: *Science* 268.5214 (1995). Publisher: American Association for the Advancement of Science, pp. 1158–1161.

[14] G. E. Hinton. “Reducing the Dimensionality of Data with Neural Networks”. en. In: *Science* 313.5786 (July 2006), pp. 504–507. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.1127647.

[15] Karl Pearson. “Note on Regression and Inheritance in the Case of Two Parents”. In: *Proceedings of the Royal Society of London Series I* 58 (Jan. 1895). ADS Bibcode: 1895RSPS...58..240P, pp. 240–242.

[16] G.W. Stewart. "On the Early History of the Singular Value Decomposition — SIAM Review". In: *SIAM Review*. Vol. 35. 4. 1993, pp. 551–556.

[17] Frank Rosenblatt. *PRINCIPLES OF NEURODYNAMICS. PERCEPTRONS AND THE THEORY OF BRAIN MECHANISMS*. en. Tech. rep. Section: Technical Reports. CORNELL AERONAUTICAL LAB INC BUFFALO NY, Mar. 1961.

[18] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. *Learning Internal Representations by Error Propagation*. en. Tech. rep. Section: Technical Reports. CALIFORNIA UNIV SAN DIEGO LA JOLLA INST FOR COGNITIVE SCIENCE, Sept. 1985.

[19] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". en. In: *Nature* 323.6088 (Oct. 1986). Number: 6088 Publisher: Nature Publishing Group, pp. 533–536. ISSN: 1476-4687. DOI: 10.1038/323533a0.

[20] Rajesh P. N. Rao and Dana H. Ballard. "Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects". en. In: *Nature Neuroscience* 2.1 (Jan. 1999), pp. 79–87. ISSN: 1097-6256, 1546-1726. DOI: 10.1038/4580.

[21] William Lotter, Gabriel Kreiman, and David Cox. "DEEP PREDICTIVE CODING NETWORKS FOR VIDEO PREDICTION AND UNSUPERVISED LEARNING". en. In: *International Conference on Learning Representations (ICLR)* (2017), p. 18.

[22] Haiguang Wen et al. "Deep Predictive Coding Network for Object Recognition". en. In: *Proceedings of the 35th International Conference on Machine Learning*. ISSN: 2640-3498. PMLR, July 2018, pp. 5266–5275.

[23] Chelsea Finn, Ian Goodfellow, and Sergey Levine. "Unsupervised Learning for Physical Interaction through Video Prediction". In: *Advances in Neural Information Processing Systems*. Vol. 29. Curran Associates, Inc., 2016.

[24] Huazhe Xu et al. “End-To-End Learning of Driving Models From Large-Scale Video Datasets”. In: 2017, pp. 2174–2182.

[25] Roshan Rane et al. “PredNet and Predictive Coding: A Critical Review”. In: *Proceedings of the 2020 International Conference on Multimedia Retrieval* (June 2020). arXiv: 1906.11902, pp. 233–241. DOI: 10.1145/3372278.3390694.

[26] Xinlei Chen, Saining Xie, and Kaiming He. “An Empirical Study of Training Self-Supervised Vision Transformers”. In: *arXiv:2104.02057 [cs]* (Aug. 2021). arXiv: 2104.02057.

[27] Kaiming He and Xinlei Chen. “Masked Autoencoders Are Scalable Vision Learners”. en. In: (2021), p. 14.

[28] Richard J Murnane and Frank Levy. *Teaching the New Basic Skills. Principles for Educating Children To Thrive in a Changing Economy.* ERIC, 1996.

[29] Ricardo Vilalta and Youssef Drissi. “A perspective view and survey of meta-learning”. In: *Artificial intelligence review* 18.2 (2002). Publisher: Springer, pp. 77–95.

[30] Yaqing Wang et al. “Generalizing from a few examples: A survey on few-shot learning”. In: *ACM Computing Surveys (CSUR)* 53.3 (2020). Publisher: ACM New York, NY, USA, pp. 1–34.

[31] Marjorie Siegel. “Rereading the signs: Multimodal transformations in the field of literacy education”. In: *Language arts* 84.1 (2006). Publisher: NCTE NATIONAL COUNCIL OF TEACHERS OF ENGLISH, p. 65.

[32] Nicholas Nuechterlein et al. “Leveraging Unlabeled Data for Glioma Molecular Subtype and Survival Prediction”. In: *2020 IEEE International Conference on Pattern Recognition (ICPR)*. IEEE, 2020.

[33] Francisco Javier Ordóñez and Daniel Roggen. “Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition”. In: *Sensors* 16.1 (2016). Publisher: Multidisciplinary Digital Publishing Institute, p. 115.

[34] Holger Caesar et al. “nuscenes: A multimodal dataset for autonomous driving”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition.* 2020, pp. 11621–11631.

[35] Beibin Li et al. “Social Influences on Executive Functioning in Autism: Design of a Mobile Gaming Platform”. In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems.* 2018, pp. 1–13.

[36] Guangpu Zhu et al. “Memory Deficit in Patients With Temporal Lobe Epilepsy: Evidence From Eye Tracking Technology”. In: *Frontiers in Neuroscience* 15 (2021). ISSN: 1662-453X.

[37] Shintaro Funahashi and Jorge Mario Andreau. “Prefrontal cortex and neural mechanisms of executive function”. In: *Journal of Physiology-Paris* 107.6 (2013). Publisher: Elsevier, pp. 471–482.

[38] Sietske W Kleibeuker et al. “Prefrontal cortex involvement in creative problem solving in middle adolescence and adulthood”. In: *Developmental cognitive neuroscience* 5 (2013). Publisher: Elsevier, pp. 197–206.

[39] Athanassios G Siapas and Matthew A Wilson. “Coordinated interactions between hippocampal ripples and cortical spindles during slow-wave sleep”. In: *Neuron* 21.5 (1998). Publisher: Elsevier, pp. 1123–1128.

[40] Sepp Hochreiter, J urgen Schmidhuber, and Corso Elvezia. “LONG SHORT-TERM MEMORY”. In: *Neural Computation* 9.8 (1997), pp. 1735–1780.

[41] Ashish Vaswani et al. “Attention Is All You Need”. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems.* arXiv: 1706.03762. Dec. 2017.

[42] Ian J. Goodfellow et al. “Generative Adversarial Networks”. In: *Advances in neural information processing systems.* Vol. Advances in neural information processing systems. arXiv: 1406.2661. June 2014.

[43] H. Robbins. “A Stochastic Approximation Method”. In: (1951). DOI: 10.1214/AOMS/1177729586.

[44] J. Kiefer and J. Wolfowitz. “Stochastic Estimation of the Maximum of a Regression Function”. In: *The Annals of Mathematical Statistics* 23.3 (Sept. 1952). Publisher: Institute of Mathematical Statistics, pp. 462–466. ISSN: 0003-4851, 2168-8990. DOI: 10.1214/aoms/1177729392.

[45] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. en. In: *arXiv:1412.6980 [cs]* (Jan. 2017). arXiv: 1412.6980.

[46] Sinno Jialin Pan and Qiang Yang. “A Survey on Transfer Learning”. In: *IEEE Transactions on Knowledge and Data Engineering* 22.10 (Oct. 2010). Conference Name: IEEE Transactions on Knowledge and Data Engineering, pp. 1345–1359. ISSN: 1558-2191. DOI: 10.1109/TKDE.2009.191.

[47] Vamsi Aribandi et al. “ExT5: Towards Extreme Multi-Task Scaling for Transfer Learning”. In: *arXiv:2111.10952 [cs]* (Nov. 2021). arXiv: 2111.10952.

[48] Chelsea Finn, Pieter Abbeel, and Sergey Levine. “Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks”. In: *arXiv:1703.03400 [cs]* (July 2017). arXiv: 1703.03400.

[49] Peter Hall, J. S. Marron, and Amnon Neeman. “Geometric Representation of High Dimension, Low Sample Size Data”. In: *Journal of the Royal Statistical Society. Series B (Statistical Methodology)* 67.3 (2005). Publisher: [Royal Statistical Society, Wiley], pp. 427–444. ISSN: 1369-7412.

[50] Tommaso Furlanello et al. “Born Again Neural Networks”. In: *arXiv:1805.04770 [cs, stat]* (June 2018). arXiv: 1805.04770.

[51] Tomas Mikolov et al. “Efficient Estimation of Word Representations in Vector Space”. In: *arXiv:1301.3781 [cs]* (Sept. 2013). arXiv: 1301.3781.

[52] Gabriela Csurka. *Domain adaptation in computer vision applications*. Springer, 2017.

[53] Yoshua Bengio, Aaron Courville, and Pascal Vincent. “Representation learning: A review and new perspectives”. In: *IEEE transactions on pattern analysis and machine intelligence* 35.8 (2013). Publisher: IEEE, pp. 1798–1828.

[54] Lerrel Pinto and Abhinav Gupta. “Supersizing Self-supervision: Learning to Grasp from 50K Tries and 700 Robot Hours”. en. In: *arXiv:1509.06825 [cs]* (Sept. 2015). arXiv: 1509.06825.

[55] Nima Tajbakhsh et al. “Embracing imperfect datasets: A review of deep learning solutions for medical image segmentation”. In: *Medical Image Analysis* (2020). Publisher: Elsevier, p. 101693.

[56] Michael N Katehakis and Arthur F Veinott Jr. “The multi-armed bandit problem: decomposition and computation”. In: *Mathematics of Operations Research* 12.2 (1987). Publisher: INFORMS, pp. 262–268.

[57] Peter Auer. “Using confidence bounds for exploitation-exploration trade-offs”. In: *Journal of Machine Learning Research* 3.Nov (2002), pp. 397–422.

[58] Peter Auer et al. “Gambling in a rigged casino: The adversarial multi-armed bandit problem”. In: *Proceedings of IEEE 36th Annual Foundations of Computer Science.* IEEE, 1995, pp. 322–331.

[59] Maria-Florina Balcan, Avrim Blum, and Santosh Vempala. “Efficient Representations for Life-Long Learning and Autoencoding”. In: *arXiv:1411.1490 [cs]* (Dec. 2014). arXiv: 1411.1490.

[60] Alex Holub, Pietro Perona, and Michael C. Burl. “Entropy-based active learning for object recognition”. en. In: *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops.* Anchorage, AK, USA: IEEE, June 2008, pp. 1–8. ISBN: 978-1-4244-2339-2. DOI: 10.1109/CVPRW.2008.4563068.

[61] Yue Zhao, Ciwen Xu, and Yongcun Cao. "Research on Query-by-Committee Method of Active Learning and Application". en. In: *Advanced Data Mining and Applications*. Ed. by Xue Li, Osmar R. Zaïane, and Zhanhuai Li. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2006, pp. 985–991. ISBN: 978-3-540-37026-0. DOI: 10.1007/11811305_107.

[62] Wenbin Cai, Ya Zhang, and Jun Zhou. "Maximizing Expected Model Change for Active Learning in Regression". en. In: *2013 IEEE 13th International Conference on Data Mining*. Dallas, TX, USA: IEEE, Dec. 2013, pp. 51–60. ISBN: 978-0-7695-5108-1. DOI: 10.1109/ICDM.2013.104.

[63] Yi Yang et al. "Multi-class active learning by uncertainty sampling with diversity maximization". In: *International Journal of Computer Vision* 113.2 (2015). Publisher: Springer, pp. 113–127.

[64] Nicholas Roy and Andrew McCallum. "Toward optimal active learning through monte carlo estimation of error reduction". In: *ICML, Williamstown* (2001), pp. 441–448.

[65] Klaus Brinker. "Incorporating diversity in active learning with support vector machines". In: *Proceedings of the 20th international conference on machine learning (ICML-03)*. 2003, pp. 59–66.

[66] David A Cohn, Zoubin Ghahramani, and Michael I Jordan. "Active learning with statistical models". In: *Journal of artificial intelligence research* 4 (1996), pp. 129–145.

[67] Diederik P. Kingma and Max Welling. "Auto-Encoding Variational Bayes". In: *arXiv:1312.6114 [cs, stat]* (May 2014). arXiv: 1312.6114.

[68] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *arXiv:1505.04597 [cs]* (May 2015). arXiv: 1505.04597.

[69] Beibin Li et al. “A Facial Affect Analysis System for Autism Spectrum Disorder”. In: *arXiv:1904.03616 [cs]* (Apr. 2019). arXiv: 1904.03616.

[70] Beibin Li et al. “Warper: Efficiently Adapting Learned Cardinality Estimators to Data and Workload Drifts”. In: *Proceedings of the 2022 International Conference on Management of Data*. 2022.

[71] Beibin Li et al. “Calibration Error Prediction: Ensuring High-Quality Mobile Eye-Tracking”. en. In: (2022), p. 14.

[72] Beibin Li et al. “Classifying Breast Histopathology Images with a Ductal Instance-Oriented Pipeline”. In: *IEEE International Conference on Pattern Recognition (ICPR)* (Dec. 2020). arXiv: 2012.06136.

[73] Christopher M Bishop and Julia Lasserre. “Generative or Discriminative? Getting the Best of Both Worlds”. en. In: (2007), p. 22.

[74] Tero Karras, Samuli Laine, and Timo Aila. “A style-based generator architecture for generative adversarial networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2019, pp. 4401–4410.

[75] Jiaxian Guo et al. “Long text generation via adversarial training with leaked information”. In: *arXiv preprint arXiv:1709.08624* (2017).

[76] Kundan Kumar et al. “Melgan: Generative adversarial networks for conditional waveform synthesis”. In: *Advances in Neural Information Processing Systems*. 2019, pp. 14910–14921.

[77] Judy Hoffman et al. “Cycada: Cycle-consistent adversarial domain adaptation”. In: *International conference on machine learning*. PMLR, 2018, pp. 1989–1998.

[78] Weixiang Hong et al. “Conditional generative adversarial network for structured domain adaptation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 1335–1344.

[79] Farzan Farnia and Asuman Ozdaglar. “Do GANs always have Nash equilibria?” en. In: *Proceedings of the 37th International Conference on Machine Learning.* ISSN: 2640-3498. PMLR, Nov. 2020, pp. 3029–3039.

[80] Anders Boesen Lindbo Larsen et al. “Autoencoding beyond pixels using a learned similarity metric”. In: *arXiv:1512.09300 [cs, stat]* (Feb. 2016). arXiv: 1512.09300.

[81] Martin Arjovsky, Soumith Chintala, and Léon Bottou. “Wasserstein gan”. In: *arXiv preprint arXiv:1701.07875* (2017).

[82] Jake Snell, Kevin Swersky, and Richard S. Zemel. “Prototypical Networks for Few-shot Learning”. In: *arXiv:1703.05175 [cs, stat]* (June 2017). arXiv: 1703.05175.

[83] Ting Chen et al. “A Simple Framework for Contrastive Learning of Visual Representations”. In: *arXiv:2002.05709 [cs, stat]* (June 2020). arXiv: 2002.05709.

[84] Jane Bromley et al. “Signature Verification using a ”Siamese” Time Delay Neural Network”. In: *Advances in Neural Information Processing Systems.* Vol. 6. Morgan-Kaufmann, 1994.

[85] Beibin Li et al. “Sparsely Grouped Input Variables for Neural Networks”. In: *arXiv:1911.13068 [cs, stat]* (Nov. 2019). arXiv: 1911.13068.

[86] Alex Nichol, Joshua Achiam, and John Schulman. “On First-Order Meta-Learning Algorithms”. In: *arXiv:1803.02999 [cs]* (Oct. 2018). arXiv: 1803.02999.

[87] Ozan Sener and Vladlen Koltun. “Multi-Task Learning as Multi-Objective Optimization”. In: *Advances in Neural Information Processing Systems.* Vol. 31. Curran Associates, Inc., 2018.

[88] Hong Hui Tan and King Hann Lim. “Review of second-order optimization techniques in artificial neural networks backpropagation”. en. In: *IOP Conference Series: Materials Science and Engineering* 495 (June 2019), p. 012003. ISSN: 1757-899X. DOI: 10.1088/1757-899X/495/1/012003.

[89] American Cancer Society. *Breast Cancer Facts & Figures 2019-2020*. en. Tech. rep. 2019, p. 44.

[90] Scott Mayer McKinney et al. "International evaluation of an AI system for breast cancer screening". In: *Nature* 577.7788 (2020). Publisher: Nature Publishing Group, pp. 89–94.

[91] Ezgi Mercan et al. "Assessment of Machine Learning of Breast Pathology Structures for Automated Differentiation of Breast Cancer and High-Risk Proliferative Lesions". en. In: *JAMA Network Open* 2.8 (Aug. 2019), e198777. ISSN: 2574-3805. DOI: `10.1001/jamanetworkopen.2019.8777`.

[92] Baris Gecer et al. "Detection and classification of cancer in whole slide breast histopathology images using deep convolutional networks". In: *Pattern recognition* 84 (2018). Publisher: Elsevier, pp. 345–356.

[93] Caner Mercan et al. "From patch-level to ROI-level deep feature representations for breast histopathology classification". In: *Medical Imaging 2019: Digital Pathology*. Vol. 10956. International Society for Optics and Photonics, 2019, 109560H.

[94] Bulut Aygüneş et al. "Graph convolutional networks for region of interest classification in breast histopathology". In: *Medical Imaging 2020: Digital Pathology*. Vol. 11320. International Society for Optics and Photonics, 2020, 113200K.

[95] Sachin Mehta et al. "Y-Net: joint segmentation and classification for diagnosis of breast biopsy images". In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2018, pp. 893–901.

[96] Sachin Mehta et al. "Learning to segment breast biopsy whole slide images". In: *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2018, pp. 663–672.

[97] Sachin Mehta et al. “Espnet: Efficient spatial pyramid of dilated convolutions for semantic segmentation”. In: *Proceedings of the european conference on computer vision (ECCV)*. 2018, pp. 552–568.

[98] Jiayun Li et al. “A multi-scale u-net for semantic segmentation of histological images from radical prostatectomies”. In: *AMIA Annual Symposium Proceedings*. Vol. 2017. American Medical Informatics Association, 2017, p. 1140.

[99] Nathan Ing et al. “Semantic segmentation for prostate cancer grading by convolutional neural networks”. In: *Medical Imaging 2018: Digital Pathology*. Vol. 10581. International Society for Optics and Photonics, 2018, 105811B.

[100] Jiayun Li et al. “An EM-based semi-supervised deep learning approach for semantic segmentation of histopathological images from radical prostatectomies”. In: *Computerized Medical Imaging and Graphics* 69 (2018). Publisher: Elsevier, pp. 125–133.

[101] Jiayun Li et al. “An attention-based multi-resolution model for prostate whole slide imageclassification and localization”. In: *arXiv preprint arXiv:1905.13208* (2019).

[102] Mohamed Attia et al. “Skin melanoma segmentation using recurrent and convolutional neural networks”. In: *2017 IEEE 14th International Symposium on Biomedical Imaging (ISBI 2017)*. IEEE, 2017, pp. 292–296.

[103] Manu Goyal and Moi Hoon Yap. “Multi-class semantic segmentation of skin lesions via fully convolutional networks”. In: *arXiv preprint arXiv:1711.10449* (2017).

[104] Mike van Zon et al. “Segmentation and Classification of Melanoma and Nevus in Whole Slide Images”. In: *2020 IEEE 17th International Symposium on Biomedical Imaging (ISBI)*. IEEE, 2020, pp. 263–266.

[105] Wenyuan Li et al. “Path R-CNN for prostate cancer diagnosis and gleason grading of histological images”. In: *IEEE transactions on medical imaging* 38.4 (2018). Publisher: IEEE, pp. 945–954.

[106] Kaiming He et al. "Mask R-CNN". en. In: *arXiv:1703.06870 [cs]* (Mar. 2017). arXiv: 1703.06870.

[107] Donghao Zhang et al. "Nuclei instance segmentation with dual contour-enhanced adversarial network". In: *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)*. IEEE, 2018, pp. 409–412.

[108] Fidel A Guerrero-Pena et al. "Multiclass weighted loss for instance segmentation of cluttered cells". In: *2018 25th IEEE International Conference on Image Processing (ICIP)*. IEEE, 2018, pp. 2451–2455.

[109] Jaeyong Kang and Jeonghwan Gwak. "Ensemble of instance segmentation models for polyp segmentation in colonoscopy images". In: *IEEE Access* 7 (2019). Publisher: IEEE, pp. 26440–26447.

[110] Zhi-Hua Zhou. "A brief introduction to weakly supervised learning". In: *National Science Review* 5.1 (2018). Publisher: Oxford University Press, pp. 44–53.

[111] Samuel Budd, Emma C Robinson, and Bernhard Kainz. "A Survey on Active Learning and Human-in-the-Loop Deep Learning for Medical Image Analysis". In: *arXiv preprint arXiv:1910.02923* (2019).

[112] Yong Li et al. "Patch-Gated CNN for Occlusion-aware Facial Expression Recognition". In: *2018 24th International Conference on Pattern Recognition (ICPR)*. IEEE, 2018, pp. 2209–2214.

[113] Therese B Bevers et al. "Breast cancer screening and diagnosis". In: *Journal of the National Comprehensive Cancer Network* 7.10 (2009). Publisher: Harborside Press, LLC, pp. 1060–1096.

[114] Natalia V Oster et al. "Development of a diagnostic test set to assess agreement in breast pathology: practical application of the Guidelines for Reporting Reliability and Agreement Studies (GRRAS)". In: *BMC Women's Health* 13.1 (2013). Publisher: Springer, p. 3.

[115] Donald L Weaver et al. “Predicting biopsy outcome after mammography: what is the likelihood the patient has invasive or in situ breast cancer?” In: *Annals of surgical oncology* 12.8 (2005). Publisher: Springer, pp. 660–673.

[116] Patricia A Carney et al. “The New Hampshire Mammography Network: the development and design of a population-based registry.” In: *AJR. American journal of roentgenology* 167.2 (1996). Publisher: Am Roentgen Ray Soc, pp. 367–372.

[117] Wenjun Wu et al. “MLCD: A Unified Software Package for Cancer Diagnosis”. en. In: *JCO Clinical Cancer Informatics* (2020), p. 9.

[118] Joann G Elmore et al. “Diagnostic concordance among pathologists interpreting breast biopsy specimens”. In: *Jama* 313.11 (2015). Publisher: American Medical Association, pp. 1122–1132.

[119] Ezgi Mercan. “Digital Pathology: Diagnostic Errors, Viewing Behavior and Image Characteristics”. en. In: (2017), p. 166.

[120] Kaiming He et al. “Deep Residual Learning for Image Recognition”. en. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, June 2016, pp. 770–778. ISBN: 978-1-4673-8851-1. DOI: 10.1109/CVPR.2016.90.

[121] Caner Mercan et al. “Multi-instance multi-label learning for multi-class classification of whole slide breast histopathology images”. In: *IEEE transactions on medical imaging* 37.1 (2017). Publisher: IEEE, pp. 316–325.

[122] Scott M Lundberg and Su-In Lee. “A unified approach to interpreting model predictions”. In: *Advances in neural information processing systems*. 2017, pp. 4765–4774.

[123] Joann G Elmore et al. “Variability in pathologists’ interpretations of individual breast biopsy slides: a population perspective”. In: *Annals of internal medicine* 164.10 (2016). Publisher: American College of Physicians, pp. 649–655.

[124] A Ting Wang et al. "Neural correlates of facial affect processing in children and adolescents with autism spectrum disorder". In: *Journal of the American Academy of Child & Adolescent Psychiatry* 43.4 (2004). Publisher: Elsevier, pp. 481–490.

[125] E. Loth et al. "Facial expression recognition as a candidate marker for autism spectrum disorder: how frequent and severe are deficits?" en. In: *Molecular Autism* 9.1 (Dec. 2018), p. 7. ISSN: 2040-2392. DOI: 10.1186/s13229-018-0187-7.

[126] Tony Charman. "Variability in neurodevelopmental disorders: evidence from Autism Spectrum Disorders". In: *Neurodevelopmental Disorders*. 2014.

[127] Mirella Dapretto et al. "Understanding emotions in others: mirror neuron dysfunction in children with autism spectrum disorders". In: *Nature neuroscience* (2006).

[128] H Ozgen et al. "Predictive value of morphological features in patients with autism versus normal controls". In: *Journal of autism and developmental disorders* (2013).

[129] A. Mollahosseini, B. Hasani, and M. H. Mahoor. "AffectNet: A Database for Facial Expression, Valence, and Arousal Computing in the Wild". In: *IEEE Transactions on Affective Computing* (2018).

[130] Jiabei Zeng, Shiguang Shan, and Xilin Chen. "Facial Expression Recognition with Inconsistently Annotated Datasets". en. In: *Computer Vision – ECCV 2018*. Vol. 11217. Cham: Springer International Publishing, 2018, pp. 227–243. ISBN: 978-3-030-01260-1 978-3-030-01261-8. DOI: 10.1007/978-3-030-01261-8_14.

[131] Corentin Kervadec et al. "CAKE: Compact and Accurate K-dimensional representation of Emotion". In: *Image Analysis for Human Facial and Activity Recognition (BMVC Workshop)*. 2018.

[132] C. Fabian Benitez-Quiroz, Ramprakash Srinivasan, and Aleix M. Martinez. "EmotioNet: An Accurate, Real-Time Algorithm for the Automatic Annotation of a Million Facial Expressions in the Wild". en. In: *2016 IEEE Conference on Computer Vision*

and Pattern Recognition (CVPR). Las Vegas, NV, USA: IEEE, June 2016, pp. 5562–5570. ISBN: 978-1-4673-8851-1. DOI: 10.1109/CVPR.2016.600.

[133] Ognjen Rudovic et al. "Personalized machine learning for robot perception of affect and engagement in autism therapy". en. In: *Science Robotics* 3.19 (June 2018), eaao6760. ISSN: 2470-9476. DOI: 10.1126/scirobotics.aao6760.

[134] Emad Barsoum et al. "Training Deep Networks for Facial Expression Recognition with Crowd-Sourced Label Distribution". en. In: *arXiv:1608.01041 [cs]* (Aug. 2016). arXiv: 1608.01041.

[135] Manuel G Calvo and Daniel Lundqvist. "Facial expressions of emotion (KDEF): Identification under different display-duration conditions". In: *Behavior research methods* 40.1 (2008). Publisher: Springer, pp. 109–115.

[136] Zhanpeng Zhang et al. "Facial landmark detection by deep multi-task learning". In: *European Conference on Computer Vision*. Springer, 2014, pp. 94–108.

[137] Xing Zhang et al. "Bp4d-spontaneous: a high-resolution spontaneous 3d dynamic facial expression database". In: *Image and Vision Computing* 32.10 (2014). Publisher: Elsevier, pp. 692–706.

[138] Olivia Wiles, A. Sophia Koepke, and Andrew Zisserman. "Self-supervised learning of a facial attribute embedding from video". en. In: *arXiv:1808.06882 [cs]* (Aug. 2018). arXiv: 1808.06882.

[139] Shan Li and Weihong Deng. "Deep Facial Expression Recognition: A Survey". en. In: *arXiv:1804.08348 [cs]* (Apr. 2018). arXiv: 1804.08348.

[140] Helen L. Egger et al. "Automatic emotion and attention analysis of young children at home: a ResearchKit autism feasibility study". en. In: *npj Digital Medicine* 1.1 (Dec. 2018), p. 20. ISSN: 2398-6352. DOI: 10.1038/s41746-018-0024-6.

[141] Andrew G. Howard et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". In: *arXiv:1704.04861 [cs]* (Apr. 2017). arXiv: 1704.04861.

[142] Sachin Mehta et al. “ESPNetv2: A Light-weight, Power Efficient, and General Purpose Convolutional Neural Network”. en. In: *arXiv:1811.11431 [cs]* (Nov. 2018). arXiv: 1811.11431.

[143] Beibin Li et al. “Learning Oculomotor Behaviors from Scanpath”. In: *Proceedings of the 2021 International Conference on Multimodal Interaction* (Oct. 2021). arXiv: 2108.05025, pp. 407–415. DOI: 10.1145/3462244.3479923.

[144] Ken Hinckley et al. “Sensing techniques for mobile interaction”. en. In: *Proceedings of the 13th annual ACM symposium on User interface software and technology - UIST ’00*. San Diego, California, United States: ACM Press, 2000, pp. 91–100. ISBN: 978-1-58113-212-0. DOI: 10.1145/354401.354417.

[145] Chul Woo Cho et al. “Robust gaze-tracking method by using frontal-viewing and eye-tracking cameras”. In: *Optical Engineering* 48.12 (Dec. 2009). Publisher: SPIE, p. 127202. ISSN: 0091-3286, 1560-2303. DOI: 10.1117/1.3275453.

[146] Nachiappan Valliappan et al. “Accelerating eye movement research via accurate and affordable smartphone eye tracking”. en. In: *Nature Communications* 11.1 (Dec. 2020), p. 4553. ISSN: 2041-1723. DOI: 10.1038/s41467-020-18360-5.

[147] Kyle Krafka et al. “Eye Tracking for Everyone”. en. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, June 2016, pp. 2176–2184. ISBN: 978-1-4673-8851-1. DOI: 10.1109/CVPR.2016.239.

[148] Mingxing Tan and Quoc Le. “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks”. en. In: *Proceedings of the 36th International Conference on Machine Learning*. ISSN: 2640-3498. PMLR, May 2019, pp. 6105–6114.

[149] Alexey Dosovitskiy et al. “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”. In: *arXiv:2010.11929 [cs]* (June 2021). arXiv: 2010.11929.

[150] Salman Khan et al. “Transformers in Vision: A Survey”. In: *arXiv:2101.01169 [cs]* (Sept. 2021). arXiv: 2101.01169.

[151] Yarin Gal and Zoubin Ghahramani. "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning". In: *arXiv:1506.02142 [cs, stat]* (Oct. 2016). arXiv: 1506.02142.

[152] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[153] Stuart Geman and Donald Geman. "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-6.6 (Nov. 1984). Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 721–741. ISSN: 1939-3539. DOI: 10.1109/TPAMI.1984.4767596.

[154] Pierre Del Moral. "Nonlinear filtering: Interacting particle resolution". en. In: *Comptes Rendus de l'Académie des Sciences - Series I - Mathematics* 325.6 (Sept. 1997), pp. 653–658. ISSN: 07644442. DOI: 10.1016/S0764-4442(97)84778-7.

[155] Jun S. Liu and Rong Chen. "Sequential Monte Carlo Methods for Dynamic Systems". en. In: *Journal of the American Statistical Association* 93.443 (Sept. 1998), pp. 1032–1044. ISSN: 0162-1459, 1537-274X. DOI: 10.1080/01621459.1998.10473765.

[156] Guillaume Alain and Yoshua Bengio. "Understanding intermediate layers using linear classifier probes". In: *arXiv:1610.01644 [cs, stat]* (Nov. 2018). arXiv: 1610.01644.

[157] Claudio Aracena et al. "Neural Networks for Emotion Recognition Based on Eye Tracking Data". In: *2015 IEEE International Conference on Systems, Man, and Cybernetics*. Oct. 2015, pp. 2632–2637. DOI: 10.1109/SMC.2015.460.

[158] Jia Zheng Lim, James Mountstephens, and Jason Teo. "Emotion Recognition Using Eye-Tracking: Taxonomy, Review and Current Challenges". en. In: *Sensors* 20.8 (Apr. 2020), p. 2384. ISSN: 1424-8220. DOI: 10.3390/s20082384.

[159] Paul Ekman. "An argument for basic emotions". en. In: *Cognition and Emotion* 6.3-4 (May 1992), pp. 169–200. ISSN: 0269-9931, 1464-0600. DOI: 10.1080/02699939208411068.

[160] Paul Ekman and Erika L. Rosenberg. *What the Face Reveals: Basic and Applied Studies of Spontaneous Expression Using the Facial Action Coding System (FACS)*. en. Google-Books-ID: KVmZKGZfmfEC. Oxford University Press, 1997. ISBN: 978-0-19-510447-9.

[161] Jackson Beatty. "Task-evoked pupillary responses, processing load, and the structure of processing resources." In: *Psychological bulletin* 91.2 (1982). Publisher: American Psychological Association, p. 276.

[162] Jakob De Lemos et al. "Measuring emotions using eye tracking". In: *Proceedings of measuring behavior*. Vol. 226. 2008, pp. 225–226.

[163] Jean-Pierre Thibaut and Robert M French. "Analogical reasoning, control and executive functions: a developmental investigation with eye-tracking". In: *Cognitive Development* 38 (2016). Publisher: Elsevier, pp. 10–26.

[164] Harry Zhang, Matthew RH Smith, and Gerald J Witt. "Identification of real-time diagnostic measures of visual distraction with an automatic eye-tracking system". In: *Human factors* 48.4 (2006). Publisher: SAGE Publications Sage CA: Los Angeles, CA, pp. 805–821.

[165] Tad T Brunyé et al. "Accuracy is in the eyes of the pathologist: the visual interpretive process and diagnostic accuracy with digital whole slide images". In: *Journal of biomedical informatics* 66 (2017). Publisher: Elsevier, pp. 171–179.

[166] JooWon Lee and Jae-Hyeon Ahn. "Attention to banner ads and their effectiveness: An eye-tracking approach". In: *International Journal of Electronic Commerce* 17.1 (2012). Publisher: Taylor & Francis, pp. 119–137.

[167] Zoya Bylinskii et al. *MIT Saliency benchmark*.

[168] Katarzyna Chawarska, Suzanne Macari, and Frederick Shic. "Decreased spontaneous attention to social scenes in 6-month-old infants later diagnosed with autism spectrum disorders". In: *Biological psychiatry* 74.3 (2013). Publisher: Elsevier, pp. 195–203.

[169] Thomas W. Frazier et al. "A Meta-Analysis of Gaze Differences to Social and Nonsocial Information Between Individuals With and Without Autism". In: *Journal of the American Academy of Child & Adolescent Psychiatry* 0.0 (May 2017). ISSN: 0890-8567, 1527-5418. DOI: 10.1016/j.jaac.2017.05.005.

[170] Quan Wang et al. "Operationalizing atypical gaze in toddlers with autism spectrum disorders: a cohesion-based approach". In: *Molecular autism* 9.1 (2018). Publisher: BioMed Central, p. 25.

[171] Roberto Pierdicca et al. "User-centered predictive model for improving cultural heritage augmented reality applications: An HMM-based approach for eye-tracking data". In: *Journal of Imaging* 4.8 (2018). Publisher: Multidisciplinary Digital Publishing Institute, p. 101.

[172] Wolfgang Fuhl. "Fully convolutional neural networks for raw eye tracking data segmentation, generation, and reconstruction". In: *arXiv preprint arXiv:2002.10905* (2020).

[173] Shane D Sims and Cristina Conati. "A neural architecture for detecting user confusion in eye-tracking data". In: *Proceedings of the 2020 International Conference on Multimodal Interaction*. 2020, pp. 15–23.

[174] Mahmoud Elbattah et al. "Learning clusters in autism spectrum disorder: Image-based clustering of eye-tracking scanpaths with deep autoencoder". In: *2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. IEEE, 2019, pp. 1417–1420.

[175] Dario D Salvucci and Joseph H Goldberg. "Identifying fixations and saccades in eye-tracking protocols". In: *Proceedings of the 2000 symposium on Eye tracking research & applications*. 2000, pp. 71–78.

[176] Kyunghyun Cho et al. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". en. In: *arXiv:1406.1078 [cs, stat]* (Sept. 2014). arXiv: 1406.1078.

[177] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. “Sequence to Sequence Learning with Neural Networks”. en. In: (2014), p. 9.

[178] Yanping Huang and Rajesh PN Rao. “Predictive coding”. In: *Wiley Interdisciplinary Reviews: Cognitive Science* 2.5 (2011). Publisher: Wiley Online Library, pp. 580–593.

[179] Michael W Spratling. “A review of predictive coding algorithms”. In: *Brain and cognition* 112 (2017). Publisher: Elsevier, pp. 92–97.

[180] Yury Gitman et al. “Semiautomatic visual-attention modeling and its application to video compression”. In: *Image Processing (ICIP), 2014 IEEE International Conference on.* IEEE, 2014, pp. 1105–1109.

[181] Antoine Coutrot and Nathalie Guyader. “How saliency, faces, and sound influence gaze in dynamic social scenes”. In: *Journal of vision* 14.8 (2014). Publisher: The Association for Research in Vision and Ophthalmology, pp. 5–5.

[182] Antoine Coutrot and Nathalie Guyader. “An efficient audiovisual saliency model to predict eye positions when looking at conversations”. In: *2015 23rd European Signal Processing Conference (EUSIPCO).* IEEE, 2015, pp. 1531–1535.

[183] Tilke Judd et al. “Learning to predict where humans look”. In: *2009 IEEE 12th international conference on computer vision.* IEEE, 2009, pp. 2106–2113.

[184] Alexander Leube, Katharina Rifai, and Siegfried Wahl. “Sampling rate influences saccade detection in mobile eye tracking of a reading task”. In: *J. Eye Mov. Res.* 10 (2017), p. 3.

[185] Ralf Kredel et al. “Eye-tracking technology and the dynamics of natural gaze behavior in sports: A systematic review of 40 years of research”. In: *Frontiers in psychology* 8 (2017). Publisher: Frontiers, p. 1845.

[186] N. J Sasson et al. “Brief Report: Circumscribed Attention in Young Children with Autism”. In: *Journal of autism and developmental disorders* (2010), pp. 1–6.

[187] Beibin Li et al. “Selection of Eye-Tracking Stimuli for Prediction by Sparsely Grouped Input Variables for Neural Networks: towards Biomarker Refinement for Autism”. en. In: *ACM Symposium on Eye Tracking Research and Applications*. Stuttgart Germany: ACM, June 2020, pp. 1–8. ISBN: 978-1-4503-7133-9. DOI: 10.1145/3379155.3391334.

[188] Frederick Shic, Katarzyna Chawarska, and Brian Scassellati. “The amorphous fixation measure revisited: With applications to autism”. In: *30th Annual Meeting of the Cognitive Science Society*. 2008, pp. 1–6.

[189] Beibin Li et al. “Q-error Bounds of Random Uniform Sampling for Cardinality Estimation”. In: *arXiv:2108.02715 [cs, math, stat]* (Sept. 2021). arXiv: 2108.02715.

[190] Beibin Li et al. “Cardinality Estimation: Is Machine Learning a Silver Bullet?” en. In: *The 3rd International Workshop on Applied AI for Database Systems and Applications (AIDB)*. Sydney, Australia, 2021, p. 5.

[191] Anshuman Dutt et al. “Selectivity estimation for range predicates using lightweight models”. en. In: *Proceedings of the VLDB Endowment* 12.9 (May 2019), pp. 1044–1057. ISSN: 2150-8097. DOI: 10.14778/3329772.3329780.

[192] Andreas Kipf et al. “Learned cardinalities: Estimating correlated joins with deep learning”. In: *CIDR* (2018).

[193] Zongheng Yang et al. “Deep unsupervised cardinality estimation”. In: *Proceedings of the VLDB Endowment* 13.3 (2019). Publisher: VLDB Endowment, pp. 279–292.

[194] Benjamin Hilprecht et al. “DeepDB: Learn from Data, not from Queries!” In: *arXiv preprint arXiv:1909.00607* (2019).

[195] Anshuman Dutt et al. “Efficiently approximating selectivity functions using low overhead regression models”. en. In: *Proceedings of the VLDB Endowment* 13.12 (Aug. 2020), pp. 2215–2228. ISSN: 2150-8097. DOI: 10.14778/3407790.3407820.

[196] Maayan Frid-Adar et al. “GAN-based synthetic medical image augmentation for increased CNN performance in liver lesion classification”. In: *Neurocomputing* 321 (2018). Publisher: Elsevier, pp. 321–331.

[197] Veit Sandfort et al. “Data augmentation using generative adversarial networks (CycleGAN) to improve generalizability in CT segmentation tasks”. In: *Scientific reports* 9.1 (2019). Publisher: Nature Publishing Group, pp. 1–9.

[198] Burr Settles. “Active learning literature survey”. In: (2009). Publisher: University of Wisconsin-Madison Department of Computer Sciences.

[199] Jialin Ding et al. “ALEX: an updatable adaptive learned index”. In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2020, pp. 969–984.

[200] Lin Ma et al. “Active learning for ML enhanced database systems”. In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2020, pp. 175–191.

[201] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository. http://archive.ics.uci.edu/ml*. 2017.

[202] Viktor Leis et al. “How good are query optimizers, really?” In: *Proceedings of the VLDB Endowment* 9.3 (2015). Publisher: VLDB Endowment, pp. 204–215.

[203] Nicolas Bruno, Surajit Chaudhuri, and Luis Gravano. “STHoles: A multidimensional workload-aware histogram”. In: *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*. 2001, pp. 211–222.

[204] Magnus Müller, Guido Moerkotte, and Oliver Kolb. “Improved selectivity estimation by combining knowledge from sampling and synopses”. In: *Proceedings of the VLDB Endowment* 11.9 (2018). Publisher: VLDB Endowment, pp. 1016–1028.

[205] João Gama et al. “A survey on concept drift adaptation”. In: *ACM computing surveys (CSUR)* 46.4 (2014). Publisher: ACM New York, NY, USA, pp. 1–37.

[206] Christopher Manning and Hinrich Schutze. *Foundations of statistical natural language processing.* MIT press, 1999.

[207] Keith Bonawitz et al. "Towards Federated Learning at Scale: System Design". en. In: *Proceedings of Machine Learning and Systems* 1 (Apr. 2019), pp. 374–388.

[208] Tian Li et al. "Federated Learning: Challenges, Methods, and Future Directions". In: *IEEE Signal Processing Magazine* 37.3 (May 2020). Conference Name: IEEE Signal Processing Magazine, pp. 50–60. ISSN: 1558-0792. DOI: 10.1109/MSP.2020.2975749.

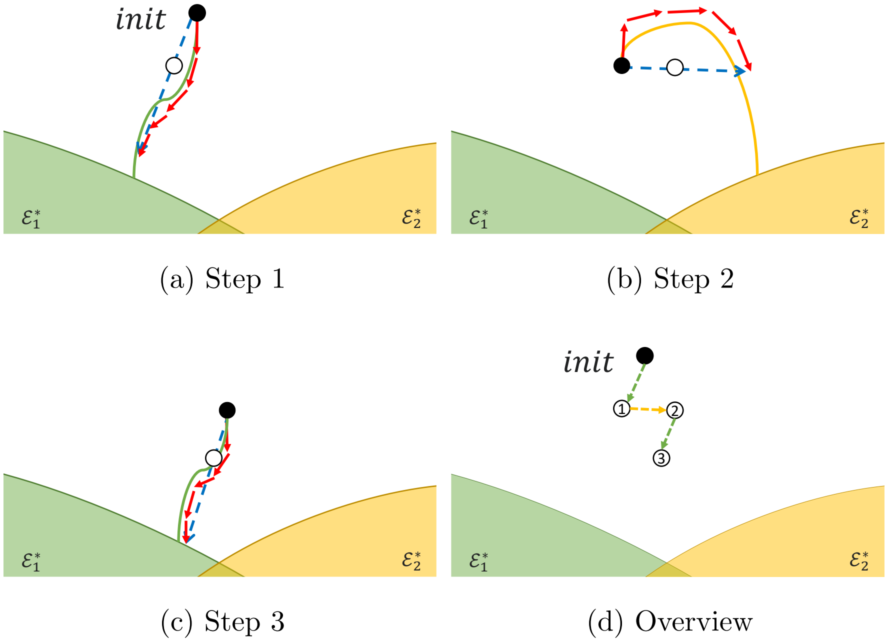
[209] István Hegedűs, Gábor Danner, and Márk Jelasity. "Decentralized learning works: An empirical comparison of gossip learning and federated learning". en. In: *Journal of Parallel and Distributed Computing* 148 (Feb. 2021), pp. 109–124. ISSN: 0743-7315. DOI: 10.1016/j.jpdc.2020.10.006.

[210] Imon Banerjee et al. "Reading Race: AI Recognises Patient's Racial Identity In Medical Images". In: *arXiv:2107.10356 [cs, eess]* (July 2021). arXiv: 2107.10356.

[211] *TPC-H Benchmark. "http://www.tpc.org/tpch/.*

Appendix A

SUPPLEMENT FOR METHODS



(a) Step 1

(b) Step 2

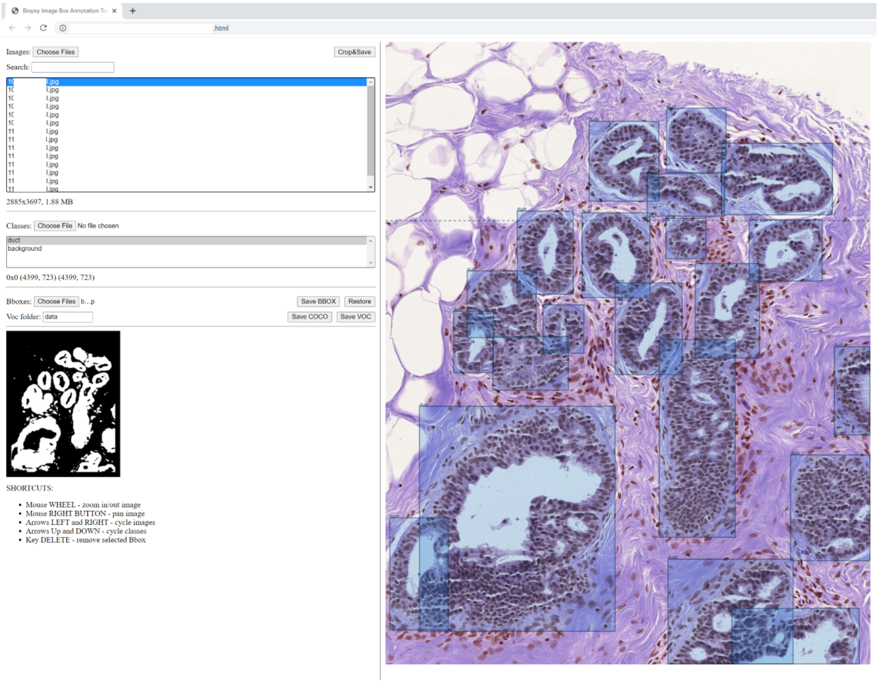
(c) Step 3

(d) Overview

Figure A.1: A visualization for the optimization path in Algorithm 2: suppose $k = 5$ for the inner loop steps, and solid red lines represent the gradient calculated for each inner-loop update. Green and yellow lines represent the optimization path for decoding task 1 and decoding task 2. Dashed blue lines represent the difference between $\widetilde{\phi}$ and ϕ for each iteration inside the outer loop. Solid and empty dots represent the weights before and after the outer-loop update. The last figure overviews three outer-loop iterations. One limitation of this meta-learning approach is the speed: the encoder needs to perform k inner updates for each outer-loop update; hence, it is $O(k)$ times slower than the MTL in the same condition.

Appendix B

SUPPLEMENT FOR MEDICAL IMAGING SYSTEM



10 l.jpg
10 l.jpg
10 l.jpg
10 l.jpg
10 l.jpg
10 l.jpg
11 l.jpg
11 l.jpg
11 l.jpg
11 l.jpg
11 l.jpg
11 l.jpg
11 l.jpg
11 l.jpg

2885x3697, 1.88 MB

Classes: No file chosen

duct
background

0x0 (4399, 723) (4399, 723)

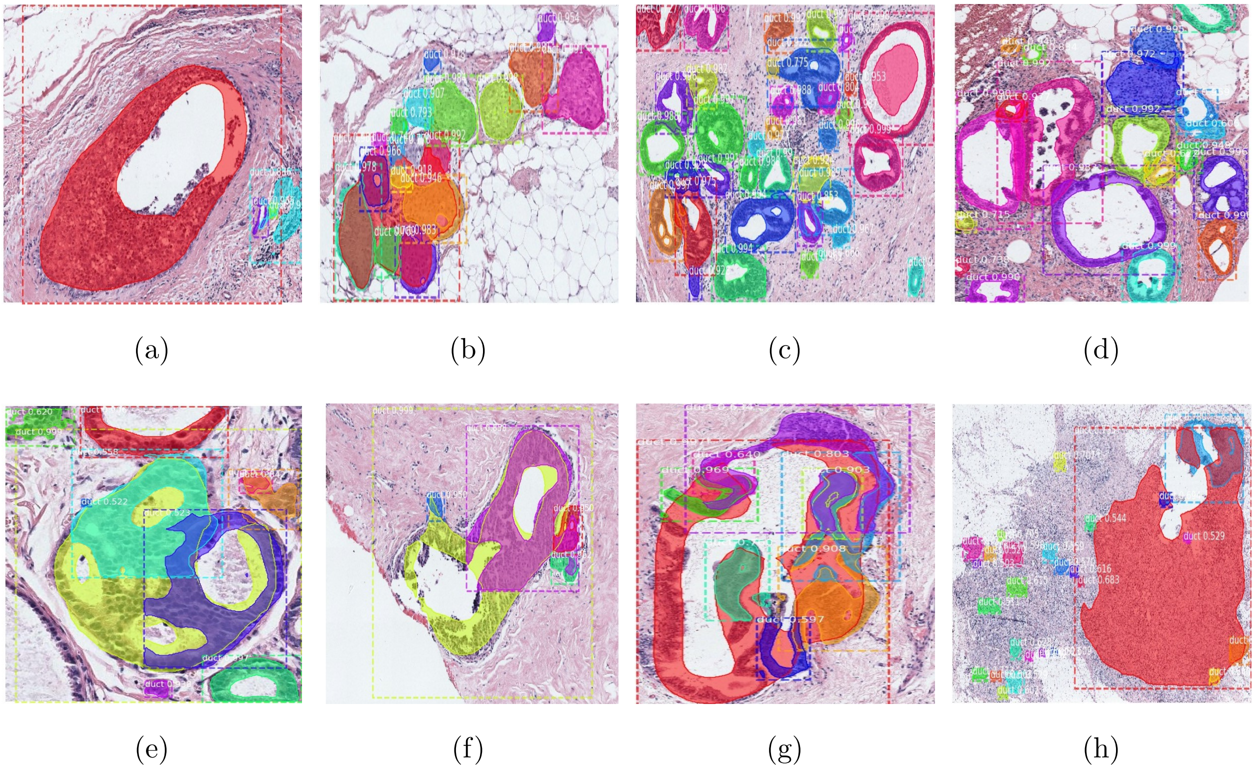
Bboxes: b...p

Voc folder: data

SHORTCUTS:

- Mouse WHEEL - zoom in/out image
- Mouse RIGHT BUTTON - pan image
- Arrows LEFT and RIGHT - cycle images
- Arrows Up and DOWN - cycle classes
- Key DELETE - remove selected Bbox

Figure B.1: **The graphical user interface (GUI) to annotate duct instances:** the main panel (right-hand side) shows the input image and allows users to create bounding box annotations. The bottom left side shows the binary tissue mask generated by the UDA framework to guide the annotator. The top left section allows the annotator to load, select, and save images and annotations. Filenames are removed in this visualization for privacy concerns. This GUI is developed based on HTML and YOLO BBox Annotation Tool[1].



(a)

(b)

(c)

(d)

(e)

(f)

(g)

(h)

Figure B.2: **Ductal segmentation testing result:** each color represents one instance of a duct in the biopsy. The top row shows four examples with satisfiable duct identification results, and the bottom row shows four imperfect examples. Examples (e), (f), and (g) have taken a single irregular and expanded duct and split it into multiple duct structures. Cancerous cells have escaped from ducts in (h), and our system mistakenly marks a big region (in red) as one duct.

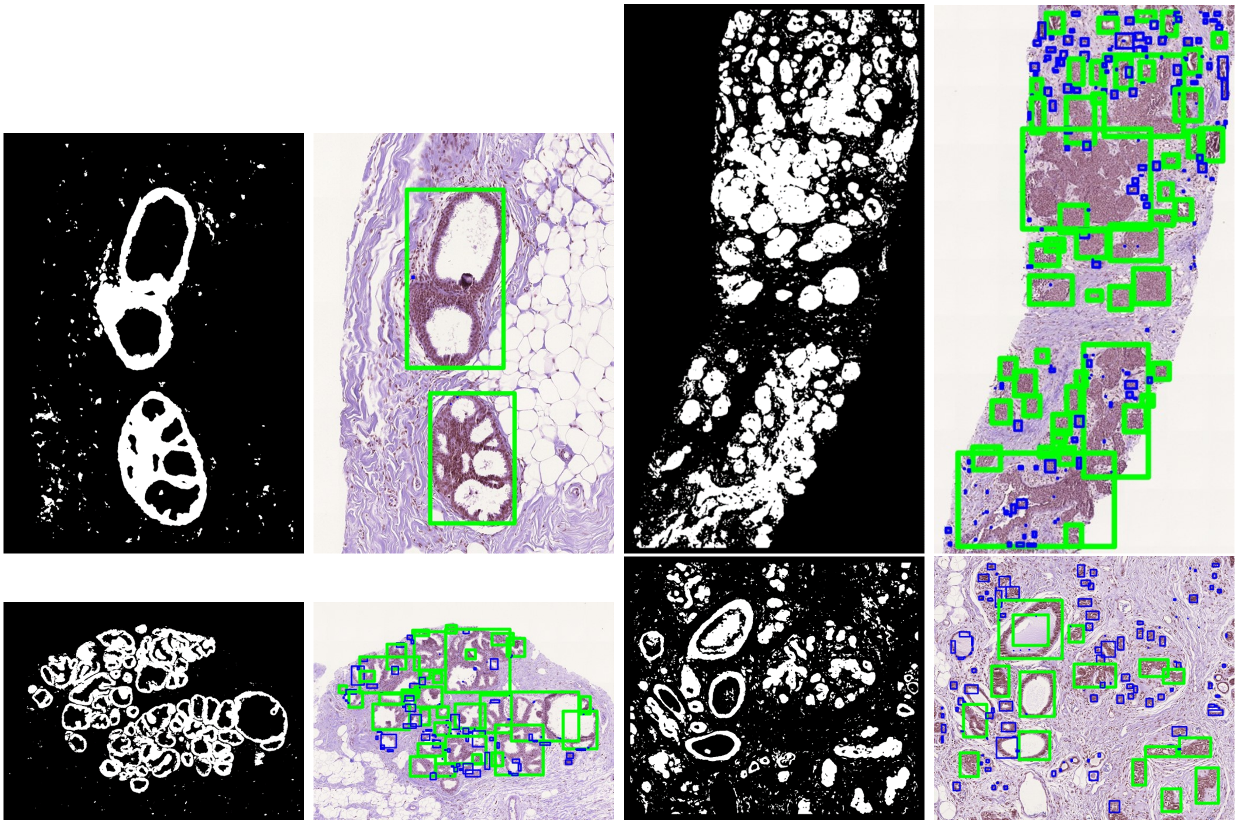


Figure B.3: Detection Results for Duct Instances: binary images are result from the semantic segmentation decoder. Instance detection results are from Mask R-CNN. Green boxes are true positives, and blue boxes are false positives.

Appendix C
SUPPLEMENT FOR FACIAL EXPRESSION SYSTEM

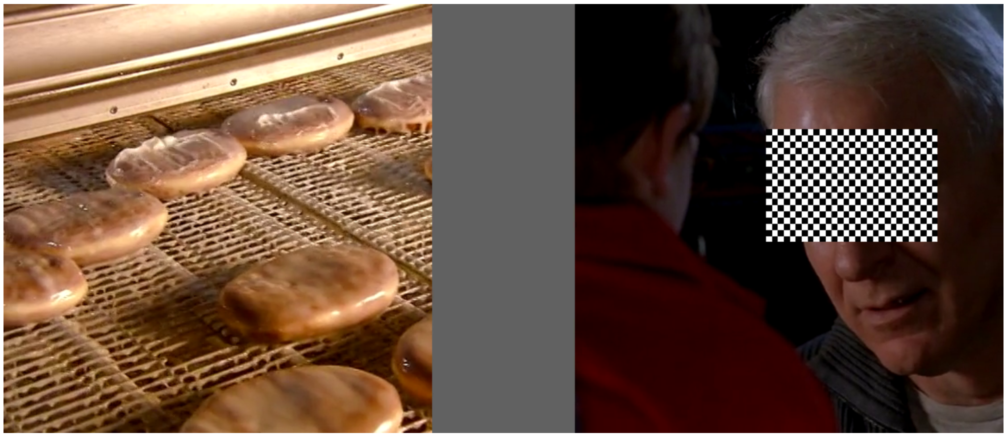


Figure C.1: Sample Frame of a Visual Preference Stimulus in the iPad application. In the image, we mosaic the actor's eyes for privacy issues.

Appendix D

SUPPLEMENT FOR EYE-TRACKING SYSTEM

	10-Shot Accuracy			Autism-Classification		
	10-w	100-w	1003-w	Acc	F_1	AUC
All Tasks	**0.65**	**0.48**	**0.31**	**0.80**	**0.88**	**0.83**
No-RC	0.4	0.43	0.29	**0.80**	0.87	0.71
No-PC	0.56	0.47	**0.31**	0.76	0.85	0.83
No-FI	0.58	0.45	0.30	0.71	0.82	0.7
No-CL	0.33	0.37	0.26	0.76	0.84	0.80

Table D.1: Ablation for pre-train tasks. For the 10-shot supervised learning experiment with MIT-1003 dataset, we evaluate the performance in 10-, 100-, 1003-way classification tasks. For the Autism classification experiment, we evaluate the accuracy, F_1, and AUC scores.

Experiment	Method	1-shot	3-shot	5-shot	10-shot
Supervised	C-VAE	0.23	0.18	0.19	0.34
	No Pre-Train	0.21	0.22	0.28	0.30
	Ours (UDA)	**0.25**	**0.28**	**0.41**	**0.63**
Metric-based	C-VAE	0.40	0.60	0.65	0.71
	No Pre-Train	0.50	0.65	0.72	0.77
	Ours (UDA)	**0.52**	**0.70**	**0.73**	**0.79**

(a) 10-way Classification

Experiment	Method	1-shot	3-shot	5-shot	10-shot
Supervised	C-VAE	0.04	0.11	0.15	0.24
	No Pre-Train	0.04	0.08	0.14	0.36
	Ours (UDA)	**0.06**	**0.14**	**0.32**	**0.44**
Metric-based	C-VAE	0.16	0.36	0.41	0.47
	No Pre-Train	0.12	0.28	0.31	0.39
	Ours (UDA)	**0.18**	**0.37**	**0.44**	**0.51**

(b) 100-way Classification

Experiment	Method	1-shot	3-shot	5-shot	10-shot
Supervised	C-VAE	0.01	0.04	0.08	0.14
	No Pre-Train	0.01	0.05	0.15	0.30
	Ours (UDA)	**0.02**	**0.08**	**0.17**	**0.31**

(c) 1003-way Classification with Supervised Learning. Metric-based meta-learning is not available for this experiment because the meta-testing set would be empty.

Table D.2: Stimulus Prediction Experiment Testing Results for the MIT-1003 Experiment: Learning from k-shot examples for each stimulus. We performed standard supervised learning and metric-based meta-learning for different scenarios.

Enc Block	Para	T	RC Dist	PC Dist	FI AUC	CL ACC
GRU	163k	4.62	**4.06**	**6.38**	**0.80**	**0.85**
RNN	55k	4.61	12.0	7.68	0.52	0.63
LSTM	217k	4.71	4.55	6.45	0.72	0.83
Transformer	343k	2.59	8.93	7.12	0.61	0.76
GRU (no Conv)	150k	7.68	4.35	**6.38**	**0.80**	0.84
Educated Guess			11.8	11.8	0.5	0.5
C-VAE with UDA	**15k**	**1.08**	8.18	6.54	0.65	0.79

Table D.3: Effects from different model backbones, where we use a 2-layer unit with 128 hidden neurons in each layer for these models. The number of trainable parameters (para) in the encoder and the inference runtime (T) in milliseconds is reported. The Euclidean distance (Dist), AUC score, and accuracy (ACC) are evaluated for the pre-tasks.

Enc Block	Para	T	RC Dist	PC Dist	FI AUC	CL ACC
2 x 32	15k	4.65	5.46	6.50	0.71	0.81
2 x 64	46k	4.67	4.52	6.53	0.71	0.81
2 x 128	163k	4.62	4.06	6.38	0.80	**0.85**
2 x 256	620k	4.72	**3.77**	**6.28**	**0.82**	**0.85**
1 x 128	64k	2.50	4.89	6.39	0.75	0.83
2 x 128	163k	4.62	4.06	6.38	0.80	0.85
3 x 128	262k	7.02	**3.52**	**6.30**	0.84	**0.86**
4 x 128	361k	11.89	3.67	6.42	**0.89**	0.85

Table D.4: Effects from different model size. A GRU unit with a convolutional layer are used across these experiments. The first column represents (the number of layers) x (the number of neurons in each layer) for the GRU unit. The number of trainable parameters (param) and inference run-time (T) in milliseconds are also reported.

Appendix E

SUPPLEMENT FOR DATABASE MANAGEMENT SYSTEM

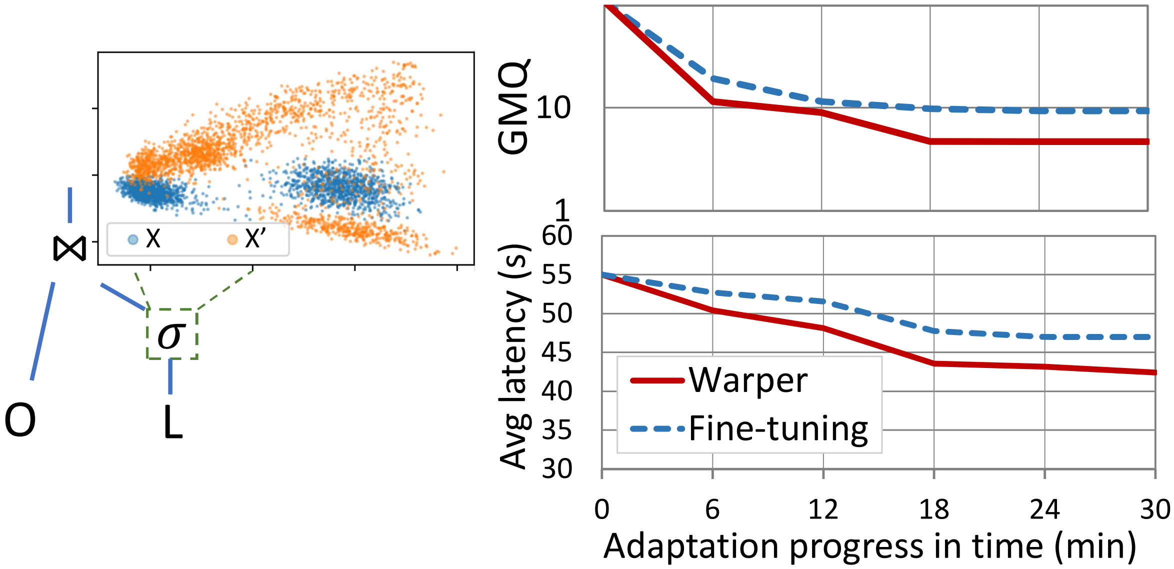


Figure E.1: Visualizing the effect of workload drift on the performance of a simple query using tables from TPC-H [211]; the predicate shown is drawn from distribution X which changes to X'[1]; on the right, we see how the accuracy of cardinality estimates from [191] and the query latency vary at different adaptation steps when new queries from X' arrive periodically.

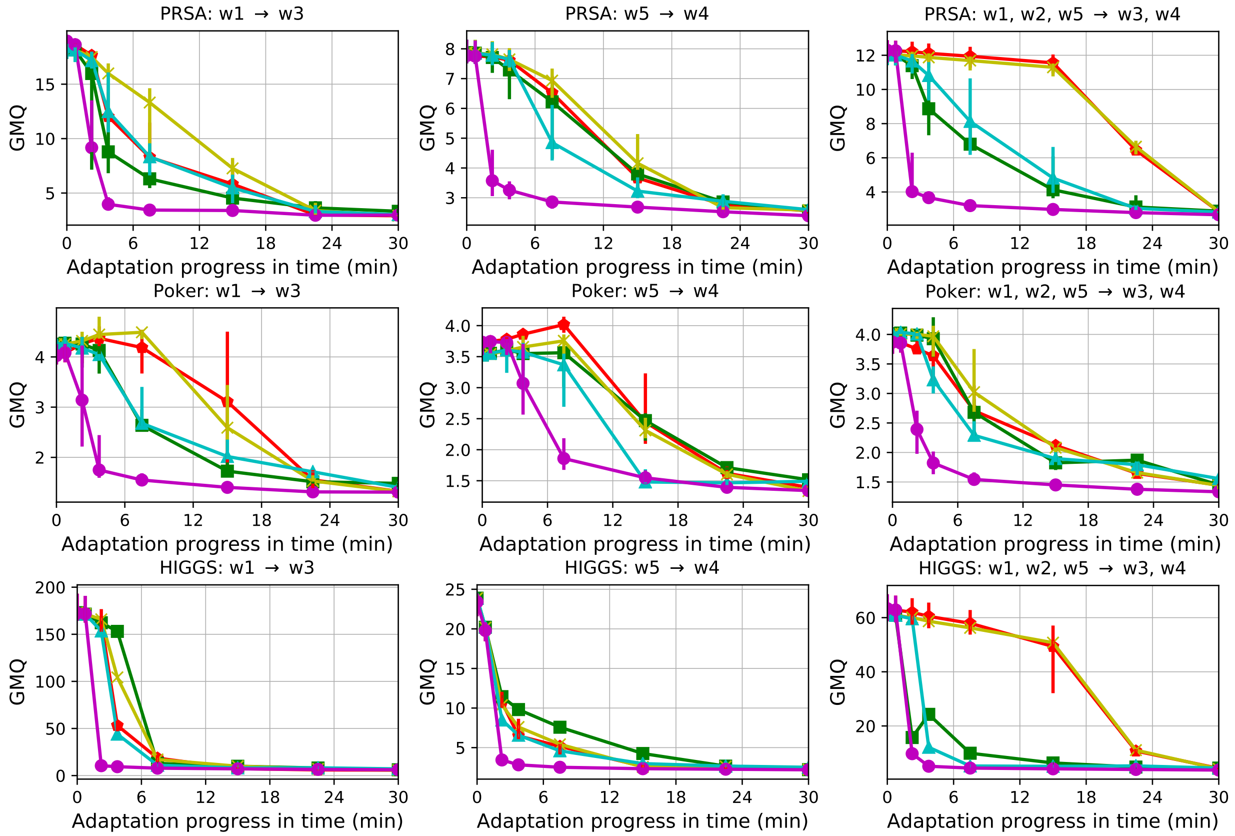


Figure E.2: We demonstrate adaptation in workload drift c2 using LM-MLP with training and newly arrived workloads described in figure title, e.g., w1 → w3 indicates that the model is trained on workload w1 and is tested when the workload drifts to w3. Red: Fine-tune. Green: hard-example mining. Yellow: MIX approach. Blue: Ad-hoc AUG method. Purple: Proposed UDA framework.

Exp.	Dataset	Cs	Wkld	Model	$\delta_{\mathtt{m}}$	$\delta_{\mathtt{js}}$	$\Delta_{.5}$	$\Delta_{.8}$	Δ_{1}
a.Wkld drift	PRSA	c2	w12/345	LM-mlp	2.4	0.31	7.4	4.8	3.1
(Figure 8.4)	Poker	c2	w12/345	LM-mlp	2.0	0.27	7.1	7.3	7.7
	Higgs	c2	w12/345	LM-mlp	12	0.60	3.8	3.7	3.5
b.Different	PRSA	c2	w12/345	LM-gbt	0.8	0.31	1.1	1.0	1.0
models	Poker	c2	w12/345	LM-gbt	1.3	0.27	1.0	3.5	6.8
	Higgs	c2	w12/345	LM-gbt	9.9	0.60	1.0	1.0	1.2
	PRSA	c2	w12/345	LM-ply	0.5	0.3	1.9	1.1	1.1
	Poker	c2	w12/345	LM-ply	1.7	0.3	1.0	1.0	1.1
	Higgs	c2	w12/345	LM-ply	6.1	0.6	1.0	4.0	1.5
	PRSA	c2	w12/345	LM-rbf	2.6	0.3	1.5	1.5	1.3
	Poker	c2	w12/345	LM-rbf	1.6	0.3	2.2	4.3	5.8
	Higgs	c2	w12/345	LM-rbf	11.5	0.6	1.2	1.3	1.2
	PRSA	c2	w12/345	MSCN	1.8	0.31	6.2	3.6	3.9
	Poker	c2	w12/345	MSCN	1.4	0.27	6.0	8.1	3.3
	Higgs	c2	w12/345	MSCN	9.6	0.6	2.5	5.2	3.2
c.Different	PRSA	c1	w1-5	LM-mlp	0.4	-	3.0	7.6	1.0
drifts	Poker	c1	w1-5	LM-mlp	0.9	-	1.3	1.1	1.5
	Higgs	c1	w1-5	LM-mlp	12	-	1.5	1.0	1.0
	PRSA	c3	w12/345	LM-mlp	2.1	0.31	1.1	1.1	1.0
	Poker	c3	w12/345	LM-mlp	1.9	0.27	1.4	1.4	1.2
	Higgs	c3	w12/345	LM-mlp	0.5	0.60	1.0	1.0	1.0
d.Join CE	IMDB	c2	w4/w1	MSCN	72	0.52	2.1	2.8	1.1

Table E.1: *Warper* with different CE models, drifts and workload distributions; results are aggregated over 10 runs.

Exp.	Wkld	$\delta_{\mathtt{m}}$	$\delta_{\mathtt{js}}$	$\Delta_{.5}$	$\Delta_{.8}$	Δ_{1}
d. Different workload	`w1/2`	1.1	0.35	3.8	3.8	4.0
	`w1/3`	16.0	0.43	3.2	4.4	6.2
	`w1/4`	5.2	0.41	5.2	4.8	6.2
	`w2/3`	13.7	0.32	7.9	5.5	4.2
	`w2/4`	5.2	0.34	8.9	8.6	8.0
	`w5/3`	14.7	0.30	4.7	5.7	3.4
	`w5/4`	4.4	0.22	4.6	3.8	1.6
	`w34/125`	0.1	0.28	1.3	1.5	1.5
	`w35/124`	0.2	0.26	4.5	1.2	1.1
	`w125/34`	9.2	0.25	12.7	12.1	1.4

Table E.2: *Warper* with different workload distributions on PRSA.

Query setting	Executed as:	Predicate on	Latency gap
`S1` - Buffer spill	Single thread	L	2.1×
`S2` - Join type	Single thread	L, O	306×
`S3` - Bitmap distr.	Multi-thread	L, O	5.3×

Table E.3: Queries used in this section. Latency gap indicates the max latency difference between plans with accurate and inaccurate CE.

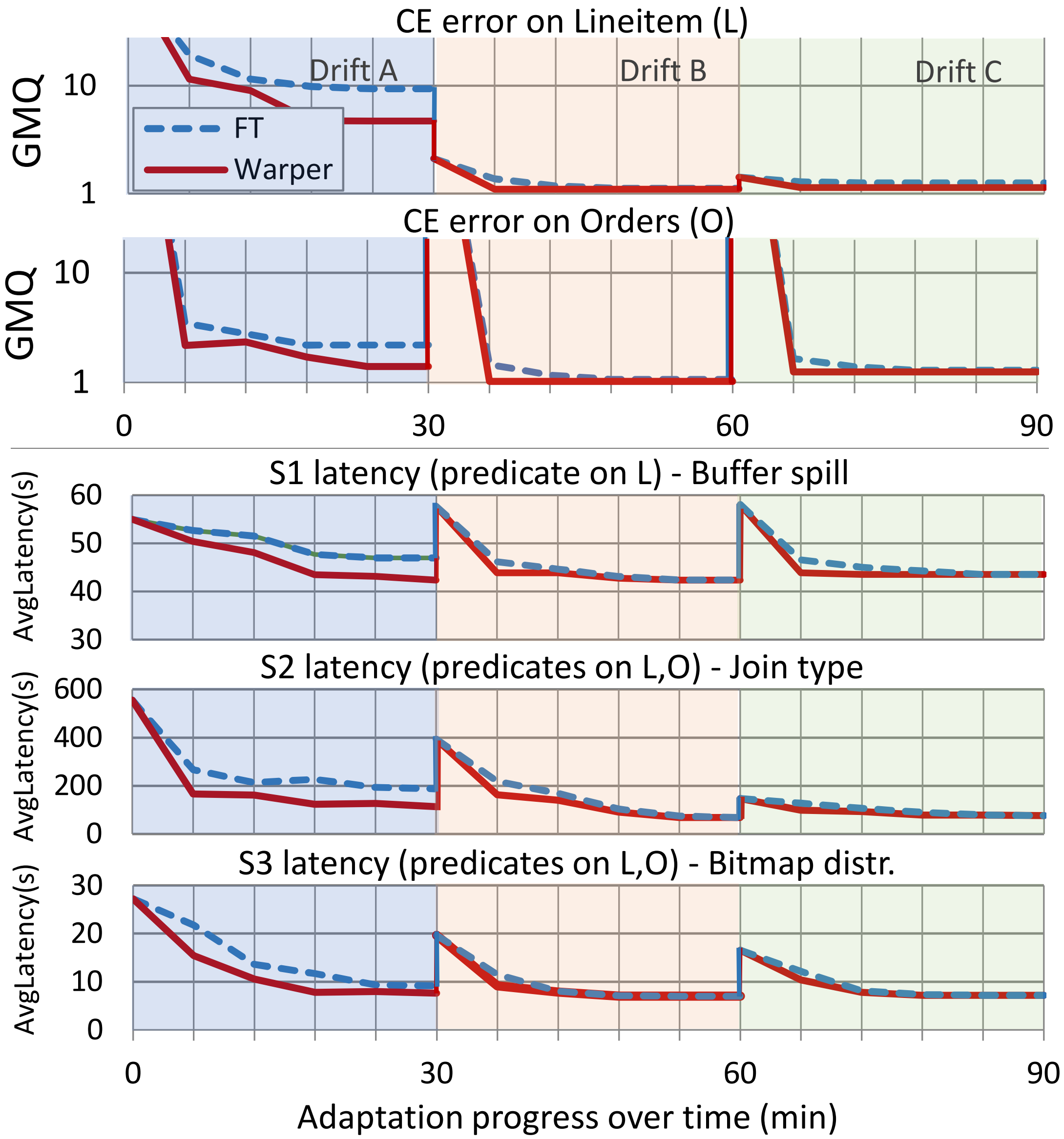


Figure E.3: We show that faster model adaptation results in query performance gains in three cases and also in continuous drifts.

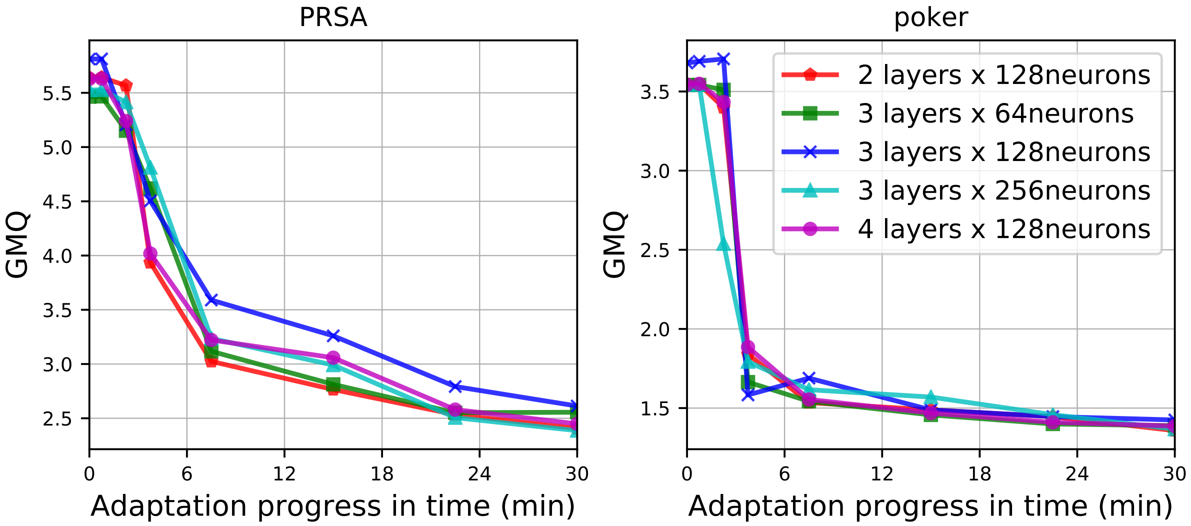


Figure E.4: Varying the NN hyperparameters in the encoder and generative decoder.

	Dataset	*Warper*	$\mathbb{P}\rightarrow$rnd pick	$\mathbb{P}\rightarrow$entropy	$\mathbb{G}\rightarrow$`AUG`
$\Delta_{0.8}$	PRSA	4.8	3.3	3.8	4.6
	Poker	7.3	1.3	6.7	6.9
Δ_1	PRSA	3.1	2.0	3.2	3.2
	Poker	7.7	1.0	1.6	6.9

Table E.4: Replacing learned *Warper* components with alternatives.

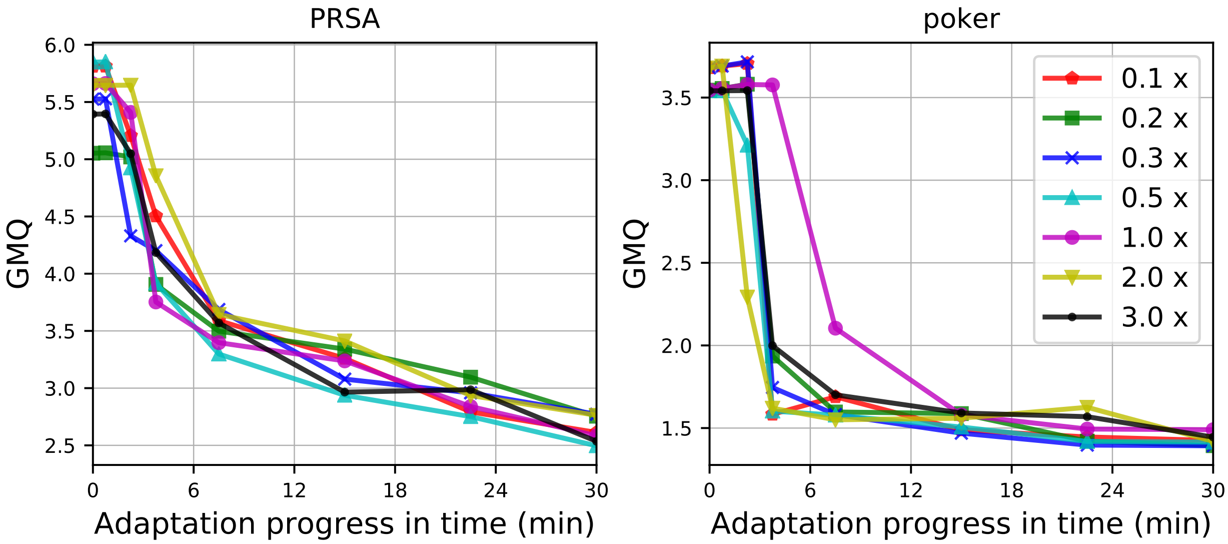


Figure E.5: Trading compute for adaption speedup. We vary the number of generated queries n_g and show the speedups. $n\times$ indicates $n_g = n \times n_t$ generated.

Dataset	PRSA				Poker			
n_g	0.1×	0.3×	1×	3×	0.1×	0.3×	1×	3×
Anno.	1.2s	3.6s	12.1s	36.3s	3.2s	9.6s	31.9s	95.7s
Model	const 52.2s (0.24% CPU usage)				const 60.6s (0.28% CPU usage)			
CPU	0.25%	0.26%	0.30%	0.41%	0.29%	0.33%	0.43%	0.72%

Table E.5: Trading compute for adaption speedup. We show the additional CPU utilization when n_g varies as multiples of n_t with an adaptation period 30min and one query arriving per five seconds.