Few-shot Learning in Changing Domains

Ian Mason

Doctor of Philosophy

Institute of Perception, Action and Behaviour

School of Informatics

University of Edinburgh

2021

Abstract

This thesis will present a number of investigations into how machine learning systems, in particular artificial neural networks, function in changing domains. In the standard machine learning paradigm a model is evaluated on held out (test) data from the same dataset the model is trained on. That is, across training and test splits, the data is independent and identically distributed. However, in many situations, after a model is trained we encounter related but different data (out of distribution) on which we hope to use our model. This thesis is concerned with methods for how we can adapt our model to work well on such new data and additionally considers the restriction that very often there may not be large amounts of data to adapt on.

In the first part of the thesis we examine a challenging application situation in real-time animation systems. We focus on changing styles of human locomotion, building systems that can model multiple styles at once and rapidly adapt to new styles. By augmenting state of the art systems with style modulating residual adaptation or feature-wise linear modulation we are able to model large numbers of styles at high levels of detail. We also make contributions to the general modelling of locomotion with the creation of contact-free local phase labelling.

In the second part of the thesis we examine the problem in more controlled settings. We present a technique for general adaptation in situations where the change in domain is caused by a change in measurement system and the original training data is not available (source-free). By aligning activation distributions, and training in a bottom-up manner, we achieve improvements in accuracy, calibration and data efficiency over existing techniques. We additionally analyse the effects of changing domains at the unit (or neuron) level, showing how changes in individual unit's activation distributions can reveal network structure and may be able to give us cues for faster adaptation via improved credit assignment.

In summary we highlight the importance of few-shot adaptation in multiple different settings and show how different techniques can be used productively to solve problems in these areas and provide inspiration for the next generation of machine learning models that should be able to learn continually from small amounts of data.

Acknowledgements

I would like firstly to thank my supervisor Taku Komura for guiding me through the PhD journey, providing research insight and assistance, and allowing me to follow my own interests. Secondly Cian Eastwood who, through his diligent commitment to our collaborative projects, helped me to become a better writer, programmer and thinker. I would also like to thank all other co-authors and colleagues who assisted me with the formulation and presentation of the ideas herein and provided motivation when it was most needed. Specific thanks go to Sebastian Starke, He Zhang, Julian Habekost, Levi Fussell, Alex Bird, Kunkun Pang, Chris Williams, and Floyd Chitalu.

I must also thank the people who guided this work through providing feedback and advice. Specifically Hakan Bilen and Steve Tonneau who provided invaluable feedback on how to complete a PhD well. Thanks too, for improvements and corrections of this thesis, go to the examiners Dr. Siddharth Narayanaswamy and Professor Shigeru Kuriyama. Additionally I acknowledge all the anonymous people: reviewers who helped improve published manuscripts; authors whose publicly released code and datasets made the research process smoother; the administrative staff who allowed me to focus fully on the work, and the taxpayers who indirectly funded this work through EPSRC.

Last but not least, thank you to my family who have always provided me their unconditional support and to my friends who made my time in Edinburgh immensely enjoyable. Those who have provided much cherished relief from the day to day pressures this life presents are too many to name, but special thanks go to Sasha Smith, Peter Morrison, Liam Douglas, Laura Jiménez Hernández, Abdul Baza and Federico Venturini.

Declaration

I declare that this thesis is an original report of my research, has been written by me and has not been submitted for any previous degree. For experimental work performed collaboratively, contributions have been indicated clearly and acknowledged in the Statement of Contributions. Due references have been provided on all supporting literatures and resources.

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Ian Mason*)

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

There remains a large gap between the performance and behaviours of current deep learning systems and the abilities shown by biological systems. In particular, in the ability to continually learn rapidly from small amounts of data. This thesis details a number of investigations in adapting learned artificial neural network models to new data distributions where only a small number of samples may be available.

1.1 Thesis Overview

The standard machine learning paradigm assumes one fixed data distribution which is split into training, validation and testing sets (Murphy, 2012, Chapter 1). In particular we assume that data is independent and identically distributed (i.i.d.) across sets. In real applications a machine learning model will often be developed using some collected and curated dataset and then deployed into production where the task remains the same but the data distribution may change. In this case, the model will receive samples that are out of distribution (o.o.d.).

After model deployment a change in data distribution can occur for any number of reasons (Quionero-Candela et al., 2009) and will, in all likelihood, reduce model performance. In this situation we have three options open to us: do nothing; retrain the model from scratch, or adapt on data from the new distribution. Doing nothing is equivalent to assuming the data is i.i.d. and hoping the drop in performance is not too severe. Retraining the model from scratch can be a valid solution although this may require expensive collection of data, long training times and fails to make use of the

learning performed on the training data distribution. Therefore, the most elegant solution, and the solution we focus on in this thesis, is to adapt on data from the new data distribution. By adapting the trained model we aim to transfer learned knowledge and use this to increase data efficiency and reduce training times.

So, we know we want to adapt trained models, but what does good adaptation look like? Given that we may not have a lot of data from the new data distribution, i.e. the few-shot scenario, we work under the assumption that good adaptation will often involve changing few model parameters. When we have limited data we must take great care to avoid overfitting so aim to reduce the model's capacity by reducing the number of trainable parameters. This is motivated by the intuition that models with high capacity are more prone to overfitting (Goodfellow et al., 2016, Chapter 5). Furthermore, taking a connectionist viewpoint, all learned knowledge in a neural network is stored in the connection weights. If we reduce the number of parameters that need to be adapted, we increase the number of connection weights that remain constant, maintaining more learned structure about the task we wish to solve.

This thesis is titled *Few-Shot Learning in Changing Domains* and throughout we investigate problem scenarios where we wish for models to well handle data from different probability distributions. Very often we will additionally consider the low-data scenario. Whilst biological systems demonstrate abilities to adapt quickly to unseen data samples (Hiratani and Latham, 2020), deep neural networks do not generalise in the same way (Geirhos et al., 2019a). However, the gap in behavioural performance has been dropping during the years of this thesis' development (Geirhos et al., 2021).

1.2 Thesis Outline

All work herein concerns changing data distributions. Still, the content of this thesis can be broadly divided into two parts. We begin the thesis by outlining a theoretical background that justifies our approaches in Chapter 2. We give a formal definition of domain shift and demonstrate how this links with notions of style and content.

Part I (Chapter 3). We concern ourselves first with practical applications of models that can handle data from multiple domains. In particular, style modelling in neural animation synthesis systems. Such systems are designed for generating realistic user controlled animations and have applications in various areas of computer graphics. In Part I we create large datasets containing multiple styles of humanoid locomotion

data. By considering this data in terms of style and content factors we aim to learn content invariant parameters, shared across all styles, and separate style (or domain) specific parameters. We show that this approach allows us to efficiently model arbitrary numbers of locomotion styles in high quality. By reducing the capacity of style specific modelling, by reducing the number of style specific parameters and increasing other forms of regularisation, we can examine the learning of new styles with limited data. Finally, we consider existing approaches for animation style modelling applications and provide engineering adjustments to achieve improved qualitative performance.

Part II (Chapters 4 & 5). In the second part of the thesis we turn to more fundamental approaches. In particular, we focus on adapting to a tightly defined set of possible changes in data distribution, and analyse approaches for adapting to such changes using carefully designed datasets. We begin in the source-free domain adaptation scenario where the original data used to train a model is not accessible but we still wish to adapt to a change in data distribution. We compare methods that work to make predictions more confident on the new data with methods that attempt to align the latent feature distributions between the original training data and the new deployment data. We show that by flexibly parameterising marginal feature distributions we can achieve improvements over existing methods in terms of accuracy, calibration, and data efficiency. Part II also contains an investigation into how we may be able to automatically select which few model parameters to update for an unknown change in data distribution. By working with a controlled setup, we show that differences in unit-level activation distributions align with our intuitions about which parameters should update and use this to motivate an improved method for automatic data-dependent credit assignment.

We conclude the thesis with a recap of our finding in Chapter 6 where we draw links between our methods and the broader picture of learning intelligent systems. We additionally provide several appendices for completeness.

1.3 Statement of Contributions

The majority of the original material, results and figures in this thesis are derived from the following works:

Chapter 3 - Style Modelling in Neural Animation Systems:

- *Few-shot Learning of Homogeneous Human Locomotion Styles* (2018). **Ian Ma-**

son, Sebastian Starke, He Zhang, Hakan Bilen, Taku Komura. Published in: *Computer Graphics Forum, 37(7).* Mason et al. (2018).
This work lays out a framework for modelling multiple styles of human locomotion in a neural network based animation system, with particular focus on dealing with the low-data regime through the bias-variance tradeoff. The implementation and development of this work was completed by Ian Mason with minor technical advice from Sebastian Starke and He Zhang. Supervision and project development guidance was received from Hakan Bilen and Taku Komura.

- *Real-time Style Modelling of Human Locomotion Using Feature-Wise Transformations and Contact-Free Local Phases* (2021). **Ian Mason**, Sebastian Starke, Taku Komura. Published in: *Proceedings of the ACM on Computer Graphics and Interactive Techniques, 2022.* Mason et al. (2022)
  Extends the ideas from *Few-shot Learning of Homogeneous Human Locomotion Styles*, offering improvements to both the animation synthesis network and the style modelling methodology. We additionally create a large dataset of stylised locomotion to better evaluate the differences between models. The implementation and development of this work was completed by Ian Mason, with implementation advice and project development input from Sebastian Starke, and supervision and project development from Taku Komura.

Chapter 4 - Feature Restoration for Source-Free Domain Adaptation:

- *Source-Free Adaptation to Measurement Shift via Bottom-Up Feature Restoration* (2021). Cian Eastwood*, **Ian Mason***, Christopher K. I. Williams, Bernhard Schölkopf. Published in: *International Conference on Learning Representations, 2022.* Eastwood et al. (2022).
  In specific situations, we can expect that learning to extract features with the same semantic meaning, across new out of distribution data and the original training data, is sufficient to achieve good model performance on new o.o.d. data. This work describes such situations and presents a feature restoration method which has substantial benefits for model calibration and data efficiency. The implementation and development of this work was completed jointly by Ian Mason (IM) and Cian Eastwood (CE) with supervision and project development from Chris Williams and Bernhard Schölkopf.

  Disentangling exact contributions to this work is a challenge. This project was

completed as a joint effort over a two year period of meetings, failed ideas, brainstormed solutions, joint writing sessions, results evaluations and more. This work is published as an equal contribution from CE and IM but is presented in full in this thesis in order to tell a complete story. As it is a requirement to try and disentangle collaborative contributions, I have tried to provide some clarity by highlighting key points in the journey to publication. I maintain however, that the assigning of granular intellectual credit is not a fruitful activity due to the intangible contributions of participants in collaborative projects. That is, the whole is greater than the sum of the parts.

The initial exploration and validation of the ideas behind our approach, and an initial code framework including the creation of EMNIST-DA were done by CE. From here IM and CE worked jointly on the project. MNIST datasets and CAMELYON17 were added by IM, CIFAR datasets by CE. The final experiments were refactored and run by IM, with CE creating the paper outline, developing the background theory (with BS and CW) and outlining the feature restoration framework. IM drafted the experiment explanations and limitations to slot into the story created by CE. The writing, experimental setups, theoretical background, and explanations were carefully discussed and refined jointly for the final submission.

Chapter 5 - Novel Approaches with Unit-level Surprise:

- *Unit-level Surprise in Neural Networks* (2021). Cian Eastwood*, **Ian Mason***, Christopher K. I. Williams. Published in: *NeurIPS 2021 Workshop: I (Still) Can't Believe It's Not Better.* Eastwood et al. (2021).
  One avenue for improving data efficiency in neural networks is a better understanding of credit assignment, that is, which neurons should be responsible for updating to adapt to new data. This work presents one possible way to think about this problem by asking which units in an artificial neural network are most 'surprised' by the new data. We show that patterns of surprise closely match our intuitions about credit assignment and discuss the potential of, and the issues with, this approach going forward. The implementation and development of this work was completed jointly by Ian Mason and Cian Eastwood with supervision and project development from Chris Williams.

  This work came out of ideas discussed and developed while creating *Source-*

Free Adaptation to Measurement Shift via Bottom-Up Feature Restoration. The contribution story remains much the same with the initial exploration being used for both projects. CE developed surprise-based update rules with IM creating FlexTune comparisons. Writing was performed in a similar manner with CE outlining the story and developing the background and IM running and explaining the experimental section. Again, choices for what analyses and experiments to run, along with how to present the ideas in the final submission, were made jointly.

*Denotes equal contribution.

Chapter 2

Theoretical Background

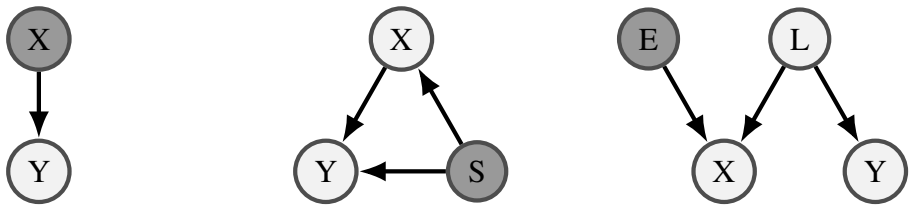
In this thesis, each chapter will outline the related work specific to the topics therein. However, before we begin the main content, we will briefly outline certain themes that underpin all the work that will be presented, namely domain adaptation, style & content, and few-shot learning. In this chapter we will give an overview of these high level areas and touch upon the related literature.

2.1 Domain Shift and Domain Adaptation

In the machine learning literature domain adaptation, domain shift and transfer learning are extremely related, often overlapping, concepts used differently by different authors. In this section we will outline how these concepts relate to one another and explain how they should be viewed in the remainder of this thesis.

2.1.1 Domain Shift

Contrary to the common independent and identically distributed assumption, very often when building machine learning systems the data which a deployed model receives (test/deployment data) is somehow different from the data on which it was trained (training/development data). For example, an image classification system may be trained on photorealistic synthetic images and during deployment have to classify images from cameras that process photographs differently (Liu et al., 2020). Alternatively different hospitals may use different methods or equipment for imaging, so data from a new hospital may differ from the model's training data (Guan and Liu, 2021).



(a) Simple Covariate Shift (b) Mixture Component Shift (c) Domain Shift

Figure 2.1: Different data generating processes lead to different types of dataset shift as described in Storkey (2009). Darker circles highlight the variables that change under the dataset shift in question.

This phenomenon is known as dataset shift (Quionero-Candela et al., 2009; Moreno-Torres et al., 2012; Wiles et al., 2021). We can formalise this as taking two datasets, $\mathcal{D}_{tr} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N_{tr}}$ and $\mathcal{D}_{te} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N_{te}}$, the training and testing data respectively, consisting of predictor data points $x^{(i)}$ and associated target labels $y^{(i)}$, where the data distributions, P_{tr} and P_{te}, are different. That is, $P_{tr} \neq P_{te}$. The data from P_{te} is said to be out of distribution (o.o.d.) and whilst this could be taken to mean any type of change in data distribution (e.g. classifying images to classifying audio), more commonly the development and deployment distributions are closely related. Dealing with dataset shift is a large area of research, as almost all machine learning models are vulnerable to changes in data distribution (Torralba and Efros, 2011; Kurakin et al., 2017; Recht et al., 2019).

One useful way to think about dataset shift is to examine the different ways in which it can occur. This is important as different types of dataset shift may be best dealt with in different ways. A non-exhaustive list of the ways in which dataset shift can occur is given by Storkey (2009), from which we highlight three examples in Figure 2.1.

Figure 2.1a shows simple covariate shift. In this case, the data generating process assumes that prediction (or output) variables, y, are caused by covariates (or input variables), x. That is, the covariates have some distribution, $P(x)$, from which the output variables are generated according to some other distribution, $P(y|x)$, giving the data distribution, $P(x,y) = P(y|x)P(x)$. The dataset shift itself (simple covariate shift) occurs due to a change in the distribution of covariates, $P(x)$. If we have learned a model for $P(y|x)$ that accurately captures the distribution, a change in covariate distribution should not require any change in the learned model. However, in practical situations there may some benefits to accounting for simple covariate shift (Shimodaira, 2000;

Sugiyama et al., 2007; Storkey, 2009).

Figure 2.1b shows mixture component shift. For this type of dataset shift, samples (x,y) are assumed to come from multiple different sources, with the ratio of samples from each of the sources changing between model training and testing. For example, we may think of samples as being noisy observations drawn from one of three underlying functions (i.e. $y = f_i(x) + \varepsilon$ for $i \in \{1,2,3\}$), with a change in the frequency with which we select to draw from each of the functions causing the dataset shift. In Figure 2.1b the change in source component proportions is represented by changing variables S.

Figure 2.1c shows domain shift. In this case we suppose some underlying domain invariant latent representation of the data, $L = l$, that causes the observations (x,y). However, covariates x may be changed by a change in domain specific, or environment, variables E. We can think about this concretely by considering the observed covariates to be a domain dependent mapping, m_E, of the underlying latent representation, so $x = m_e(l)$. A change in E changes the mapping. As an example, we could think of L as a variable for the 'true' underlying colour of an object in a scene which we wish to classify as being a primary colour or not. A change in E could be a change in rendering pipeline or scene lighting that changes the appearance of the colour in the scene, X. When E changes our observation x will change, but the underlying colour l and label y remain fixed.

This thesis largely focuses on these *domain shift* data generating assumptions, and variations upon them. In particular we will consider data which we presume to have some underlying latent representation (L) unchanging across changing domains. Much of our focus will be on how we can extract or recover this representation in the presence of nuisance variables (E) that change model behaviour when a change in domain occurs (the dataset shifts).

2.1.2 Domain Adaptation

Domain adaptation is commonly framed as taking a model trained on one or more source domains, with data distribution P_s, and adapting the model to work well on one or more target domains, with target domain t having data distribution P_t (Zhang et al., 2013; Kouw and Loog, 2018). In particular, source and target domains have different data distributions, $P_s \neq P_t$. Note that this is exactly the dataset shift scenario described

in Section 2.1.1, that is, domain adaptation is the task of adapting to dataset shift using the shifted (test) data. Domain adaptation on domain shifts is the task of adapting a model to dataset shifts of the form shown in Figure 2.1c (i.e. E changes).

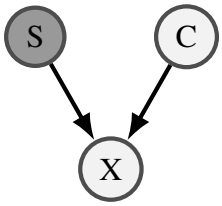
There are many different approaches to domain adaptation and many different areas where domain adaptation methods are applicable (Ramponi and Plank, 2020; Toldo et al., 2020; Guan and Liu, 2021). For our purposes we make a distinction between model-free approaches, which make no assumptions about the distributional form of, and do not explicitly parameterise, $P_s, P_t, P(L)$, $P(X)$ or $P(Y)$ (Ganin and Lempitsky, 2015; Long et al., 2018), and model-based approaches, which assume certain distributional forms for, or explicitly parameterise, $P_s, P_t, P(L)$, $P(X)$ or $P(Y)$ (Sun and Saenko, 2016; Zellinger et al., 2017). In some sense, even classic fine-tuning approaches (Girshick et al., 2014; Zeiler and Fergus, 2014) can be seen as domain adaptation under our definition of adapting a model to some dataset shift. However, commonly domain adaptation is necessary without access to labelled target domain data (unsupervised domain adaptation).

Domain adaptation is commonly associated with, and sometimes used interchangeably with, transfer learning. It is more accurate to consider domain adaptation to be a subset of transfer learning. In general we consider transfer learning to be any type of modelling task where learned information can be transferred from some already pretrained model(s). An example of transfer learning that is not domain adaptation is when the dataset does not shift but the task being performed with the data changes (Goodfellow et al., 2016, Section 15.2) (e.g. image classification to visual question answering).

Another closely related area is domain generalisation, where the aim is to learn the source model in such a way that it is able to perform well on unseen target domains without adapting on the test data. This can be achieved through data augmentation or inductive biases about the dataset shifts that may occur upon deployment (Lake et al., 2015, 2017; Raissi et al., 2019; Michaelis et al., 2019; Djolonga et al., 2021).

2.2 Style and Content

One way that machine learning practitioners have chosen to view certain data, X, is as being generated by style factors, S, and content factors, C (Tenenbaum and Freeman, 1996; Jing et al., 2019; Jin et al., 2021), see Figure 2.2a. This has allowed for the creation of neural style transfer algorithms (Gatys et al., 2015) and, with this view,



(a) Style and Content

(b) Style Image

(c) Content Image

(d) Stylised Image

Figure 2.2: Style and Content. (a) The data generating graph is the same as domain shift (Figure 2.1c) with no label. (b), (c), (d) Stylising an image with neural style transfer. A style image (b, Picasso's Mediterranean Landscape) is combined with a content image (c, The Shore, Leith), to create a stylised image (d) using an online neural style transfer tool (Nakano, 2018).

combining the content from one datapoint with the style of another has been used for many creative applications (Johnson et al., 2016; Holden et al., 2017a; Shen et al., 2017). We show an example of combining the content of one image with the style of another in Figure 2.2d.

2.2.1 The Content Invariance Assumption

From a semantic or creative viewpoint 'style' can have many meanings (Jin et al., 2021). For example, are the famous swirled stars of Van Gogh's Starry Night part of the content of the image or the style? We take the view that the content is the concepts and objects that would be consistent in the image were another painter to paint the same scene. That is, we use a data-driven definition of style that we refer to as the content invariance assumption (von Kügelgen et al., 2021). For a dataset that contains data in multiple styles, style should capture the factors that vary and content the factors that do not.

If we now return to Figure 2.2a, this assumption is captured by the changing S variables coloured in dark grey. If we compare this figure to the domain shift graph (Figure 2.1c) we see that these are in fact *the same data generating assumptions* as domain shift if we drop the label Y. So if we care only about generative models modelling $P(X)$, modelling style and content under the content invariance assumption is the same problem as handling domain shift. Changing environment variables or nuisance variables, E, in Figure 2.1c corresponds to changing style, S, in Figure 2.2a. The underlying domain invariant latent representation L corresponds to the content C.

2.3 Few-shot Learning

Whilst the most successful modern deep learning models leverage vast quantities of data (Brown et al., 2020; Radford et al., 2021), humans and animals are able to learn new tasks and concepts quickly with limited numbers of examples (Lake et al., 2019; Hiratani and Latham, 2020). Few-shot learning (Wang et al., 2020) is the task of learning from small amount of data. One-shot learning is the even more restricted task of learning from a single sample (Fei-Fei et al., 2006), in general we can use N-shot learning to refer to learning from N samples.

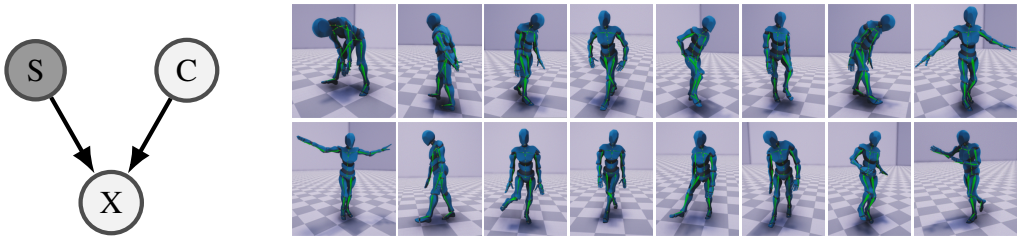
Few-shot learning is closely related to domain adaptation. Firstly, after a dataset shift there is no guarantee that we will be able to collect large amounts of data from the new data distribution; it may be prohibitively expensive or time consuming to do so. Therefore adapting to a new domain often requires additionally considering that the adaptation may need to be performed few-shot. This partnering of few-shot learning and domain adaptation provides the area in which this thesis, *Few-Shot Learning in Changing Domains*, is situated.

Few-shot learning is also closely related to domain adaptation as it almost always requires some form of transfer learning. Following the taxonomy of Wang et al. (2020) three major approaches to few-shot learning are: using prior knowledge to augment the few-shot data in order to train a more accurate model (Shyam et al., 2017; Benaim and Wolf, 2018); using inductive biases to restrict the model capacity allowing better performance with small amounts of data (Lake et al., 2015; Motiian et al., 2017; Zhang et al., 2018b; Finlayson, 2020, Chapter 0), and providing good parameter initialisations (Caelles et al., 2017; Finn et al., 2017; Arik et al., 2018; Nichol et al., 2018) or learned optimisers (Li et al., 2017b; Ravi and Larochelle, 2017) to give the few-shot data a better chance to achieve a good solution.

Throughout this thesis our approach for dealing with few-shot data will be to combine suitably setting a model's capacity (inductive biases) with learning from good parameter initialisations (transfer learning).

Chapter 3

Style Modelling in Neural Animation Systems



(a) Style and content (b) Modelling large numbers of stylised motions

Figure 3.1: Modelling changing styles for humanoid animation. (a) Data generating assumptions. (b) Examples of styles our local phase based system can model.

We begin our exploration of changing domains by examining style and content as it applies to human motion. Figure 3.1a recaps our data generating assumptions, that is, we observe some motion data, X, which is assumed to be created from underlying style factors, S, and content factors, C. Given a large dataset of stylised human motion, labelled with styles, we can use the content invariance assumption (Section 2.2.1) to assume that those (latent) factors that are consistent across styles are the content factors and those (latent) factors that differ across styles are the style factors.

That is, we assume, for a given motion, the existence of some latent motion representation shared and invariant across all styles. This is equivalent to the latent representation L shared across domains in the general domain shift case (Figure 2.1c). Similarly a change in style specific factors is equivalent to a change in environment variables

E. With this view, style and domain may be used interchangeably and having labelled styles is equivalent to knowing which domain a datapoint comes from.

In this chapter we assume we have access to data from multiple domains/styles and use this to tackle two tasks. First, we try to learn the latent content factors, C, and separate them from style factors, S, to allow us to switch between styles of motion, which has many practical uses in computer animation (that is, we change domains/styles by changing the learned environment/style factors). After doing this, we then aim to few-shot learn the domain/style specific factors for a new style. The main idea being that given a large amount of initial data, with sufficient styles, we can learn a content extractor general across styles. Therefore, for a new style we need only learn a few style specific parameters and hence reduce the amount of data required to learn to model the new style.

The work presented in this chapter is concerned with this practical application of domain adaptation; changing domains in order to qualitatively change the style of human locomotion animations. We develop specific engineering techniques and inductive biases in order to achieve high quality results, and examine the learning of new domains with low amounts of data.

As mentioned above, understanding style as it applies to human motion has many practical applications in computer graphics, such as showing character personalisation or emotional expression through walking gait differences, or editing animation systems based on a video game character's interaction with the world (e.g. injury or disguise). Indeed, *style transfer* has a long history in computer animation research (Hsu et al., 2005; Liu et al., 2005; Taylor and Hinton, 2009; Aberman et al., 2020). Such work has largely tackled offline style transfer where the aim is to explicitly replace the style factors of one motion clip with those of a different motion clip while maintaining the content (the same approach as done for images in Gatys et al. (2015) and in Figure 2.2). In order to do this, methods have been developed that make use of explicit timing or foot contact adjustments to create realistic motions (Xia et al., 2015).

A second important recent development in animation research is the creation of neural *animation synthesis* systems (Fragkiadaki et al., 2015; Holden et al., 2017b; Starke et al., 2020). These neural network based approaches avoid the need for explicit timing adjustments by generating successive frames of motion autoregressively, automatically capturing how poses, stepping patterns, and global timing change over time. Addi-

tionally, many such systems are able to synthesise controllable animation purely from previous frames and some high level user input such as a path or trajectory to follow (Holden et al., 2016).

So, whilst style transfer methods cannot easily be applied for user controlled animation synthesis where future timing and contact information is unknown and animations may be arbitrarily long, most animation synthesis systems fail to account for multiple modalities (styles) of movements and offer limited ability for editability of the type of motion being generated once they are trained. By combining ideas from both these approaches, we can create user controlled animation synthesis that can switch between a wide variety of motion styles. We refer to this task as neural *style modelling*. Qualities that we look for in good style modelling systems include: the ability to model arbitrary numbers of styles; the ability to easily add new styles to the model; high quality generation of stylised motion conditioned on high level user input, and the ability to interpolate or transition well between styles.

Our general framework for real-time style modelling involves learning two neural networks. Firstly, an *animation synthesis network* that predicts the next frame of motion conditioned on the past frames, user control and style information. Crucially, we share animation synthesis parameters across styles to encourage content factors to be captured by this network, making the content invariance assumption over motion capture datasets containing multiple styles. To model style we create a *style modulation network* that transforms the (hidden representations of the) animation synthesis network (Bird, 2020, Chapter 5). We use a learned style representation which allows us to efficiently model the style space whilst being able to generate motions in multiple styles.

This chapter begins with an overview of related work in animation synthesis, style transfer and style modelling. Afterwards, we examine style modelling in the low data regime. One of the challenges for neural style modelling is a lack of availability of large amounts of stylised motion capture data, so we design our approach with an ability to learn from small amounts of data in mind. To end the chapter we turn towards engineering improvements for style modelling, creating a dataset with a large number of stylised motion capture frames and zooming in on the design choices for animation synthesis and style modulation networks.

3.1 Related Work

During the development of this thesis the number of works using machine learning approaches for skeleton-based human animation and style transfer has increased dramatically (Mourot et al., 2021), this section gives an overview of those most related to the work herein.

3.1.1 Data-Driven Animation Synthesis

Classic methods for modelling motion made use of various approaches such as blending radial basis functions (Rose et al., 1998), principal component analysis (Safonova et al., 2004), and Gaussian processes (Mukai and Kuriyama, 2005; Ikemoto et al., 2009), but, as with many areas (LeCun et al., 2015), has come to be dominated by neural network based learning approaches. An early model (Fragkiadaki et al., 2015) used a sequence learning approach to build an encoder-recurrent-decoder architecture making use of LSTM cells (Hochreiter and Schmidhuber, 1997) to predict short motion sequences from given input frames. This approach was extended by Li et al. (2018b) who used scheduled sampling (Bengio et al., 2015) to allow for the generation of longer animation sequences. Holden et al. (2016) created a general framework motion synthesis by learning a manifold of human motion, Bütepage et al. (2017) compared several different architectures for motion generation, and Martinez et al. (2017) demonstrated that low quantitative error does not always correspond to high qualitative performance in generative models for motion by using a baseline model that predicts a constant pose.

The most closely related works to our approach learn to synthesise animation in a controllable manner given high level user input. The Phase-Functioned Neural Network (PFNN) (Holden et al., 2017b) was designed to implicitly align motion capture data using a global cyclic phase to modulate the weights of a neural network, this allows for the generation of high quality user controlled motion. However, the formulation of a global phase does not apply for all animation synthesis tasks so Zhang et al. (2018a) and Starke et al. (2019) extended this approach to quadruped motion and character scene interaction using a learned blending of expert weights. Starke et al. (2020) additionally experimented with the use of local motion phases for generating high quality interactive basketball animations. Ling et al. (2020) opt to use variational autoencoding (Kingma and Welling, 2014) over standard feed-forward architectures, and Henter

et al. (2020) use normalising flows (Kingma and Dhariwal, 2018). However, none of these approaches explicitly consider the modelling of multiple styles and make no modifications to encourage the learning of style and content.

Whilst this chapter deals with kinematic approaches to animation synthesis it would be remiss to not briefly mention that a number of authors have instead taken a physics based approach, commonly using reinforcement learning. Mimicking motion capture data using imitation learning whilst accounting for physics with a physics engine has allowed for the creation of controllable animations in simulation (Merel et al., 2017; Wang et al., 2017; Peng et al., 2018b). An additional use case involves the use of world models (Ha and Schmidhuber, 2018) for motion tracking (Fussell et al., 2021). Whilst making use of physics engines can help prevent impossible poses, the animation quality of such approaches has not yet reached that of competing kinematic methods.

3.1.2 Style Transfer and Style Modelling

Similarly to animation synthesis, multiple different angles for tackling style transfer were developed before neural network based approaches became increasingly popular in recent years. For example, Hsu et al. (2005) developed a correspondence algorithm to align motions and model style differences, and Liu et al. (2005) used a physics based optimisation. An early learning approach used factored conditional restricted Boltzmann machines (Taylor and Hinton, 2009) and conditioned on a style label, although, as a more theoretical work, lacked the high quality demos common to other methods. Other notable approaches include those of Xia et al. (2015) who constructed a mixture of regression models combined with a nearest neighbours database search to map style onto content, and Yumer and Mitra (2016) who model motion in spectral space to hand design a frequency domain based algorithm for the same task.

Many modern methods for style transfer in animation take inspiration from methods first developed for images. For example, Gatys et al. (2015) optimise to match Gram matrices in the latent space of an image classification network to created stylised images (similar to those seen in Figure 2.2) and Holden et al. (2016) optimise using the same matrices to transfer style between clips of motion. Johnson et al. (2016) take inspiration from Gatys et al. (2015) to design a loss function to train a neural network to perform style transfer on images and Holden et al. (2017a) do similarly for motion. Multiple works show that applying affine transformation to hidden representations of convolutional neural networks can also create aesthetically pleasing style transfer on

images (Dumoulin et al., 2017; Huang and Belongie, 2017; Huang et al., 2018; Perez et al., 2018); this approach has been used by Aberman et al. (2020) to transfer style between $2D$ video and $3D$ skeletal motions. Finally cycle consistency losses have been utilised to alter image style (Zhu et al., 2017a,b; Huang et al., 2018), this too has inspired works for motion style transfer (Dong et al., 2020) and retargeting (Villegas et al., 2018). As all these methods take inspiration from preceding image style transfer techniques they are designed for situations where both style and content are given *before* performing style transfer. While these approaches are useful, for example for augmenting datasets, they are not designed for use with animation synthesis systems where future content rapidly changes as user input changes and motion may be arbitrarily long. Other approaches that are also designed to stylise a clip of content motion include those of Du et al. (2019), Smith et al. (2019), Wen et al. (2021) and Park et al. (2021).

There has been some limited work for style modelling in animation systems, that is, changing style on the fly as user controlled content changes. Holden et al. (2017b) add a one-hot vector to the PFNN which allows them to model a small number of styles, although this representation is undesirable as it requires the total style count to be known before training and with large numbers of styles many of the network's parameters are given over to handling the label. More recent motion matching systems (Clavet, 2016; Holden et al., 2020) produce high quality stylised motions but require storing a database of motion capture clips (or some compressed representation) and learn no explicit style representation, leading to increasing costs with increasing database size.

3.2 Style Modelling with Residual Adaptation

To create an initial system for style modelling we use a Phase-Functioned Neural Network (Holden et al. (2017b), Figure 3.2) for animation synthesis; shared across all styles, it aims to capture the content factors in the data. To model the style factors we modulate the hidden units of the PFNN using residual adapters (Rebuffi et al., 2017, 2018).

3.2.1 The Phase-Functioned Neural Network

We use the PFNN because it encodes a strong inductive bias that human locomotion is cyclic which allows for high quality modelling of motion with fairly cheap (shallow,

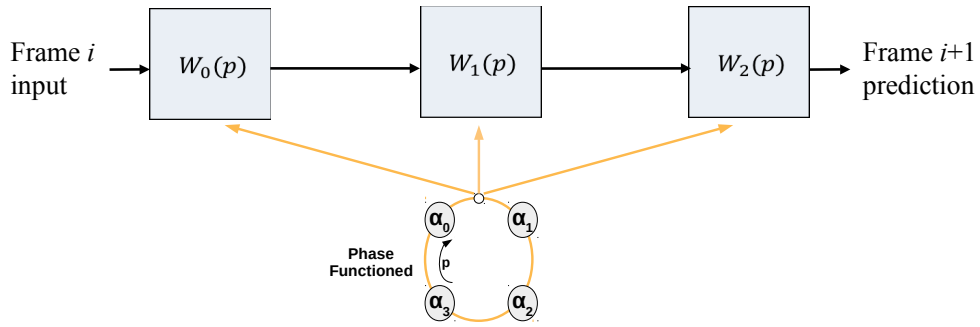


Figure 3.2: A Phase-Functioned Neural Network. The weights used for prediction depend on a phase label and are set by blending parameters using Equation 3.1.

feed-forward) architectures. Specifically we use a 3 layer architecture where the layer parameters are themselves a function of the locomotion phase, p, which is a number between 0 and 2π defined to be 0 when the right foot makes contact with the floor plane and π when the left foot makes contact with the floor plane, with the phase values for all other frames of animation data found via interpolation.

Specifically, we follow Holden et al. (2017b), using a cubic Catmull-Rom Spline to blend four different 'control points', $\boldsymbol{\beta} = \{\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \boldsymbol{\alpha}_3, \boldsymbol{\alpha}_4\}$, each themselves one possible setting of the neural network parameters. That is,

$$\begin{aligned}\mathbf{W}(p;\boldsymbol{\beta}) =& \boldsymbol{\alpha}_{k_1} + d(\frac{1}{2}\boldsymbol{\alpha}_{k_2} - \frac{1}{2}\boldsymbol{\alpha}_{k_0}) \\ &+ d^2(\boldsymbol{\alpha}_{k_0} - \frac{5}{2}\boldsymbol{\alpha}_{k_1} + 2\boldsymbol{\alpha}_{k_2} - \frac{1}{2}\boldsymbol{\alpha}_{k_3}) \\ &+ d^3(\frac{3}{2}\boldsymbol{\alpha}_{k_1} - \frac{3}{2}\boldsymbol{\alpha}_{k_2} + \frac{1}{2}\boldsymbol{\alpha}_{k_3} - \frac{1}{2}\boldsymbol{\alpha}_{k_0})\end{aligned} \tag{3.1}$$

$$d = \frac{4p}{2\pi} \quad (mod\ 1) \tag{3.2}$$

$$k_n = \left\lfloor \frac{4p}{2\pi} \right\rfloor + n - 1 \quad (mod\ 4), \tag{3.3}$$

where $\mathbf{W}(p) = \{W_0(p), W_1(p), W_2(p)\}$ is the three layers of network parameters of the PFNN for phase value p.

3.2.2 Residual Adaptors

The PFNN originally made use of one-hot labels to model different modalities of locomotion (i.e. styles such as crouching). However, for our initial style modelling we make use of residual adapters (Rebuffi et al., 2017), that are essentially skip connections with a weight matrix. That is, if a standard feed-forward layer is written as

$h^{(i+1)} = \sigma(Wh^{(i)} + b)$, with $h^{(i)}$ the input, $h^{(i+1)}$ the output, σ some non-linearity and W & b the layer parameters. Then a parallel residual adapter has the form $h^{(i+1)} = \sigma(Wh^{(i)} + b + W_{res}h^{(i)})$, where W_{res} is the weight matrix for the residual adapter (seen in Figure 3.3).

Our aim with residual adaptation is to use the PFNN to learn style agnostic (content-modelling) parameters and to learn a new residual adapter for each style we wish to model. This is directly inspired by Rebuffi et al. (2017) who learn a set of domain agnostic parameters with separate domain specific residual adapters for each domain they wish to model[1]. As with Rebuffi et al., no explicit direction is provided in order to separate out style and content factors of the data, instead we find empirically that this behaviour naturally emerges by jointly training with multiple styles. One of the main benefits of this approach is the ability to sequentially add new residual adapters once the animation synthesis network has been trained, allowing us to model new unseen styles without having to retrain the whole system as would be required with a labelling approach. Additionally, the capacity of the residual adapters can easily be changed depending on the problem they are used for by restricting, or increasing, the number of available parameters. We will make use of this property when working in the low data regime in Section 3.3.

3.2.3 Combining Animation Synthesis and Style Modelling

So, given a dataset containing a large number of motion capture frames in S styles of locomotion, we aim to learn a set of style agnostic parameters for the PFNN $\boldsymbol{\beta}_{ag}$ and S sets of style specific residual adapter parameters $\{\boldsymbol{\beta}^{(s)} | s \in \{1,\ldots,S\}\}$. Note that, in order to increase the representational power of the residual adapters and to capture phase dependent style information, we make the residual adapters themselves phase-functioned. That is, for a given style, s, at phase p, the residual adapter weights are given by the function of the phase given in Equation 3.1, with control points $\boldsymbol{\beta}^{(s)} = \{\boldsymbol{\alpha}_1^{(s)}, \boldsymbol{\alpha}_2^{(s)}, \boldsymbol{\alpha}_3^{(s)}, \boldsymbol{\alpha}_4^{(s)}\}$. The style agnostic parameters, $\boldsymbol{\beta}_{ag}$, are shared between all styles and have control points $\boldsymbol{\beta}_{ag} = \{\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \boldsymbol{\alpha}_3, \boldsymbol{\alpha}_4\}$.

For a datapoint $(x^{(s)}, y^{(s)})$, where $x^{(s)}$ is one frame of animation data and $y^{(s)}$ is the subsequent frame we wish to predict (in style s), $x^{(s)}$ is fed into the PFNN which uses style

[1] This approach of learning some domain agnostic parameters and some domain specific parameters is common in domain adaptation. For example, when learning a shared encoder and domain specific decoders (Mor et al., 2018).

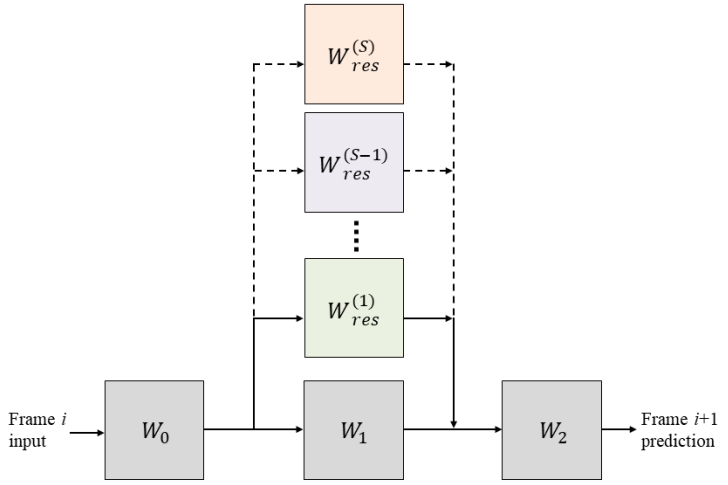


Figure 3.3: Style Modelling with Residual Adaptation. PFNN weights, shared across styles, are shown in grey. Style specific, phase-functioned, residual adapter weights are shown in other colours. Weights (W_0, W_1, W_2, $W_{res}^{(s)}$) are calculated for a given phase value using Equation 3.1. Phase omitted from figure for brevity.

agnostic weights, $\{W_0(p), W_1(p), W_2(p)\}$, calculated from $\boldsymbol{\beta}_{ag}$ using Equation 3.1. The first layer of the PFNN, $W_0(p)$, processes the input frame $x^{(s)}$ into a hidden representation. This hidden representation is then modulated by style specific residual adapter weights, $W^{(s)}$, calculated from $\boldsymbol{\beta}_s$ using Equation 3.1. $W^{(s)}$ aims to model style specific factors in the data, whilst the second layer of the PFNN, $W_1(p)$, aims to model content specific (style invariant) factors. The final layer, $W_2(p)$, combines the style specific and style invariant representations to predict the new frame $y^{(s)}$. Recall, that the aim of this system is not to optimally model the training data, but rather to split out the style (domain specific) and content (domain invariant) factors in order to be able to replace the style factors of $x^{(s)}$ with those of a new style. Figure 3.3 shows this architecture, with the shared style agnostic parameters shown in grey and the style specific parameters shown in different colours.

3.3 Few-shot Development

One challenge with style modelling is data availability. Neural networks are notoriously data hungry and we can never capture all possible styles of motion. If we wish to learn to model a new (unseen) style, capturing sufficient data may be expensive and time consuming or not possible due to the need to access specialist motion capture

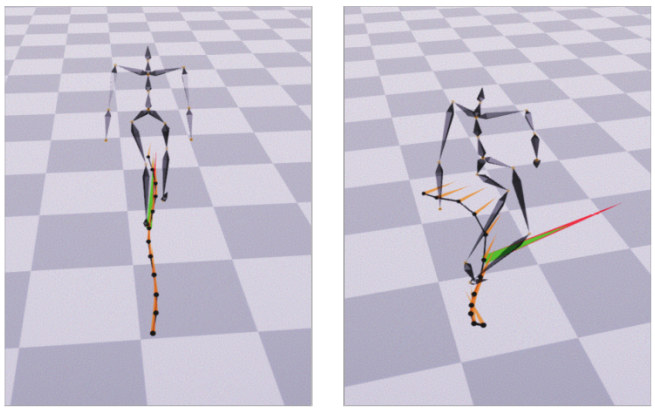


Figure 3.4: Overfitting. When only a small amount of training data is available it is easy to overfit, here a character walks reasonably in a straight line but creates unrealistic poses when following turning trajectories unseen in the training data. To create this visualisation a PFNN was trained on neutral locomotion data and fine-tuned with limited stylised (*bent forward*) data.

equipment. For this reason it is desirable to have models that can learn new styles from small amounts of data (few-shot). Training on smaller amounts of data also has the advantage of requiring lower training times.

When we have only a small amount of data this may not capture an extensive range of movements, perhaps containing only a short (1-5s) clip of stylised walking in a straight line. In this case it is easy to overfit, simply mimicking the training data, and it is challenging to generalise to new walking directions and speeds. Figure 3.4 shows an example of this, where a character walking with the *bent forward* style is reasonably modelled when moving in a straight line but becomes unrealistic when turning.

If we can learn a reasonable set of style agnostic parameters, which we attempt to do using a large dataset of stylised locomotion, then our aim is to transfer this learned knowledge to model changes in walking speeds and turning directions for new styles with low amounts of data. With our fixed set of style agnostic parameters, $\boldsymbol{\beta}_{ag}$, we can simply learn a new residual adapter to model a new style. Our basic approach to working with small amounts of data is to change the residual adapter's capacity to qualitatively balance overfitting and underfitting (Goodfellow et al., 2016, Section 5.2). We do this by using a canonical polyadic (CP) decomposition of the residual adapters along with variable regularisation strength.

3.3.1 CP Decomposition

To flexibly change the capacity of phase-functioned residual adapters we perform a CP decomposition (Kolda and Bader, 2009) to reduce the number of style specific parameters. The CP decomposition can be seen as a generalisation of singular value decomposition to higher dimensions. For our case we consider a $3D$ tensor $W \in \mathbb{R}^{I \times J \times K}$ where each matrix slice of the tensor, W_k, can be written as

$$W_k = AD^{(k)}B^T \tag{3.4}$$

with $A \in \mathbb{R}^{I \times R}$, $B \in \mathbb{R}^{J \times R}$, $D^{(k)} \in \mathbb{R}^{R \times R}$ and $D^{(k)}$ a diagonal matrix for $k = 1, \ldots, K$ and some positive integer R. Note that A and B are the same $2D$ matrices across all slices whereas $D^{(k)}$ changes.

We need to use a tensor decomposition because of the phase-functioned nature of the residual adapter parameters, in particular we consider the phase as the third dimension. To understand this we highlight that Holden et al. (2017b) discretised the phase at runtime in order to improve inference speed. Specifically, Holden et al. saved 50 weight matrices, calculated using Equation 3.1, for phases $[0, 2\pi/50, \ldots, 2\pi)$ and used the weights associated with the closest phase value for a frame to make the next frame prediction. By stacking these 512×512 weight matrices, we can create a $3D$ tensor of size $512 \times 512 \times 50$. Using Equation 3.4, we can decompose each of the 50 slices (which are the 50 saved weight matrices) into 3 matrices of sizes $512 \times R$, $R \times R$ and $R \times 512$. By changing R we can change the capacity of the residual adapter.

Since, for the CP decomposition, only the $R \times R$ matrix varies over the phase dimension with the rectangular matrices fixed, this means that only this square diagonal matrix has phase-functioned weights (weights that change as the phase changes). This means that with the CP decomposition we are able to maintain phase dependence in the style representations whilst also varying their capacity.

As in Zhao et al. (2017), we perform the CP decomposition *before training*, that is, we learn factors A, B & D from Equation 3.4 directly. This approach allows us to learn the best parameters for these factors during training. Doing this means that, for a new style, we learn two (non phase-functioned) rectangular matrices with $512 \times R$ weights and one phase-functioned diagonal square matrix, with R weights. Note that this subtly changes $\boldsymbol{\beta}^{(s)}$, as now only the central matrix requires control points $\{\boldsymbol{\alpha}_1^{(s)}, \boldsymbol{\alpha}_2^{(s)}, \boldsymbol{\alpha}_3^{(s)}, \boldsymbol{\alpha}_4^{(s)}\}$.

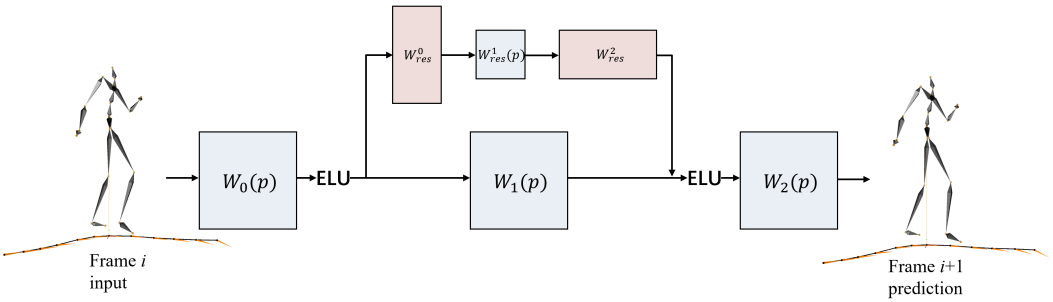


Figure 3.5: System Overview. Style agnostic parameters, shared across all styles, are learned in the animation synthesis PFNN network (bottom row). Style specific parameters are learned in a CP decomposed residual adapter (top row) which can change capacity as the size of the central matrix changes. Phase-functioned weight matrices are shown in blue, non phase-functioned weight matrices are shown in red. We use the exponential linear unit (ELU) as our non-linearity Clevert et al. (2015).

An overview of our system, showing a decomposed residual adapter for a single style, is shown in Figure 3.5. This figure highlights in blue those weight matrices which are phase-functioned and in red those which are not.

3.3.2 Variable Dropout

Another benefit of each learned style having a corresponding residual adapter is that this allows us to vary the regularisation strength depending on the data. A simple way to achieve this is to vary the dropout rate when training with low amounts of data. This variable dropping of connections in the residual adapter can intuitively be seen as varying how much the network can rely on the style specific parameters versus the style agnostic parameters. Whilst we could use alternative regularisation methods ($L1$, $L2$), initial experiments found these to be roughly equally effective. Therefore, we use dropout due to its simplicity and the intuitive trade off between style agnostic and style specific parameters.

The ability of our system to generalise to unseen motions in a new style is primarily determined by two factors. The most important factor is the data quantity available, with more data less regularisation is needed as overfitting becomes harder. The second factor is how similar a new style is to the styles used to learn the style agnostic parameters. Without a huge number of styles the style agnostic parameters will not be

truly agnostic but rather a useful shared representation of the content shared between pretraining styles, so a new style that is very different from the pretraining styles may require lower regularisation. Since this similarity between styles is hard to measure, we run a grid search over dropout rates and select the value which gives best qualitative performance.

3.4 Datasets, Training & Implementation

We have now outlined an approach for the few-shot learning of locomotion styles. Before we evaluate this system we detail the technical specifics of our method. This includes the dataset we use for learning style agnostic parameters, the dataset we use for few-shot learning, loss functions, training procedures and implementations details.

3.4.1 Datasets

To learn shared style agnostic parameters we capture a large number of frames of stylised locomotion and extract useful features to create neural network inputs and outputs. For learning new styles we rely on existing datasets (Gross and Shi, 2001) which contain many different motions, albeit with limited amounts of data.

3.4.1.1 Large Style Locomotion Dataset

To train the PFNN weights we capture a dataset containing 8 representative styles of locomotion inspired by the styles used by Xia et al. (2015): *angry, childlike, depressed, neutral, old, proud, sexy and strutting*. These styles are performed by a single actor of height 182cm and captured at 120 fps (frames per second) using the Vicon Nexus optical motion capture system.

Each of the 8 captured styles are globally cyclic, that is, similar poses can be aligned using a single phase variable. We capture running and walking motions moving on a flat plane as these are the most common motions for interactive applications. The result is a large dataset of around 1 hour of motion capture data (Table 3.1). Note that since there is no fixed correspondence between captures in different styles the specific style components must be learned from this unaligned, unstructured data.

Table 3.1: Unmirrored frame counts and number of locomotion cycles for each of the styles in our large captured dataset.

Style	No. Frames	No. Cycles
Angry	11552	337
Childlike	11552	335
Depressed	11552	208
Neutral	11552	242
Old	11552	262
Proud	11552	180
Sexy	11552	252
Strutting	11552	216

3.4.1.2 Data Processing

After capturing the data we extract motion features to create data in the same format as used by Holden et al. (2017b). The data is first mapped to the same skeletal structure as used by Gross and Shi (2001) and Holden et al. (2017b), to enable us to learn new styles without needing to worry about changes in character skeletons. The data is additionally mirrored to double the number of frames available and ensure a balance of turning directions. To make sure the dataset is balanced over all styles we use an equal number of motion capture frames for each style (23104 after mirroring, see Table 3.1).

The motion features we extract are the 31 skeletal joint positions, velocities and rotations. These are calculated for every frame relative to the character's root joint (hips) position, this create a local co-ordinate frame which allows us to ignore global translation. Each frame is additionally labelled with a phase value as defined in Section 3.2.1. To do this labelling a semi-automatic process is used which extracts approximate foot contacts with the floor plane by thresholding foot joint velocities (which are 0 when the foot is planted on the floor). These foot contacts are manually cleaned and right foot contacts labelled with phase 0, left foot contacts with phase π and linear interpolation finding all other phase values. We refer to one full cycle of the phase label (a change of 2π) as a locomotion cycle and use this as a useful indicator of the amount of data available for a given style as the PFNN aligns motions using the phase (Table 3.1 also shows locomotion cycles per style).

Recalling that one of the purposes of a style modelling approach is to allow a user to control the motion content via high level input we additionally extract the same

user controllable information as in Holden et al. (2017b). This takes the form of the character trajectory found by projecting the root joint position onto the floor plane. For a single frame this trajectory consists of 12 points found by sampling the projection of the root joint every 10 frames for 60 frames in the past, the current frame, and 50 frames in the future. At each point the position and facing direction of the root joint relative to the current root joint position and facing direction is calculated. This trajectory representation, projected onto the floor plane, can be seen in Figure 3.4 (left) along with the skeletal structure of the character. Technically, each frame of motion is also given a style label although this is not used as input to the network, instead it is used to allow us to select which residual adapter to use.

After this processing the training data consists of inputs, $x_i \in \mathbb{R}^{234}$, and outputs, $y_i \in \mathbb{R}^{400}$, that we aim to predict. x_i is frame of motion capture data and consists of: 2×12, x and z planar trajectory positions; 2×12, x and z planar trajectory directions; 3×31, x, y and z joint positions, and 3×31, x, y and z joint velocities. y_i is the next frame of motion after x_i and consists of: 2×6, x and z current and future planar trajectory positions; 2×6, x and z current and future planar trajectory directions; 3×31, x, y and z joint positions; 3×31, x, y and z joint velocities; 3×31, x, y and z joint rotation forward components; 3×31, x, y and z joint rotation upward components; 1 predicted change in phase, and 1×3, x and z planar root translations with y root rotation around the vertical axis to perform a global character location update.

We standardise the data before training the neural network, importantly we standardise jointly over all styles. This is because the mean and standard deviation of (input and output) style specific representations captures a substantial amount of the style variation (Huang and Belongie, 2017; Li et al., 2017a), however, we wish to model the style factors with the style specific parameters. Standardising each style individually makes modelling the style specific variation with network parameters more difficult, so we standardise all styles together.

3.4.1.3 Few-shot Dataset

To evaluate learning in the low data regime we extract 50 different styles of motion from the open source CMU motion capture database (Gross and Shi, 2001). This database has a large number of motion styles but often very limited data for individual styles, meaning learning a controllable character that can create general locomotion in a given style is difficult. For example, for certain styles there may only be 1 locomotion

cycle in the database from which we aim to extract the style factors and use these with the pretrained content (style agnostic) representation to generalise to new turning directions and speeds. Appendix A.1 provides a table with frame counts and numbers of locomotion cycles for each of the styles extracted.

This few-shot data is processed to create the same features described above however, since some of the styles are asymmetric (see Appendix A.1) we do not mirror them. Additionally, when training with this data we normalise using the same mean and standard deviation calculated on the large, 8 style, dataset. If styles were standardised individually this would change the relative amounts of style and content information in the input features, and the standardisation would be limited to the specific speeds and turning directions in the data making generalisation harder.

3.4.2 Training & Implementation

The first step to learn our style modelling system is to train the agnostic parameters $\boldsymbol{\beta}_{ag}$ using the large balanced 8 style dataset. We additionally learn 8 style specific residual adapters setting R to be 30 in Equation 3.4[2]. Relative to the number of style agnostic parameters, this adds only 1.5% more parameters for each style we learn[3].

For style s with input frame $x_i^{(s)}$, corresponding phase value $p_i^{(s)}$, and output frame $y_i^{(s)}$ (which we predict), we train the network by minimising a mean squared error loss. This is given by

$$\mathcal{L}(x_i^{(s)}, y_i^{(s)}, p_i^{(s)}; s) = ||y_i^{(s)} - \Phi(x_i^{(s)}, p_i^{(s)}; \boldsymbol{\beta}^{(s)}, \boldsymbol{\beta}_{ag})||_2 + \lambda_{ag}||\boldsymbol{\beta}_{ag}||_1 + \lambda_s||\boldsymbol{\beta}^{(s)}||_1, \quad (3.5)$$

where Φ is the PFNN using the residual adapter for style s for style modulation and a small L1 loss is placed on the weights for (sparsity-encouraging) regularisation (Murphy, 2012, Section 13.3). We set λ_{ag} and all λ_s to 0.01.

As well as balancing the data, we encourage style invariance by balancing how styles are learned during pretraining. We split the data into minibatches, with each minibatch containing one style of motion. During training we cycle through the styles, meaning every 8^{th} minibatch contains data from the same style. Since the data is balanced we

[2] As with the dropout rate, R is chosen by performing a grid search over multiple values and checking for qualitative signs of overfitting or underfitting such as those shown in Figure 3.4. Other hyperparameters are similarly chosen by performing small grid searches and evaluating qualitative performance.

[3] If we discretise the phase saving 50 weight matrices for each phase-functioned layer, the extra number of parameters is only 0.03% more.

have an even number of minibatches for each style and our overall objective becomes

$$L(\mathbf{X},\mathbf{Y},\mathbf{P}) = \sum_{i=1}^{N}\sum_{s=1}^{S} \mathcal{L}(x_i^{(s)}, y_i^{(s)}, p_i^{(s)}; s), \tag{3.6}$$

where N is the number of frames per style, S is the number of styles, and $\mathbf{X},\mathbf{Y},\mathbf{P}$ are the sets of all input frame representations, phase values and output frame representations respectively ($\mathbf{X} = \bigcup_{i=1}^{N}\bigcup_{s=1}^{S} x_i^{(s)}$ etc.).

As in Holden et al. (2017b), both hidden layers of the PFNN have size 512. We pretrain our system for 25 epochs using Adam (Kingma and Ba, 2014) which takes approximately 6 hours on a single NVIDIA GTX 1080 Ti GPU. This number of epochs acts as an additional early stopping regularisation as we find training for longer can produce results overfitted to certain styles.

3.4.2.1 Few-Shot Implementation

If this training procedure learns a good set of style agnostic parameters, then we can freeze these parameters (bottom row in Figure 3.5) and learn only new residual adapters for new styles of motion. Given enough data, the style agnostic parameters should learn to model general motion content, that is, 'the things that are common to all styles' (content invariance assumption, Section 2.2.1). This means that for a new style, we need only learn how to modulate these general features rather than learning everything about locomotion from scratch. Together with the decomposed residual adapters ($R = 30$) and making use of variable dropout this allows us to learn with small amounts of data.

For each style extracted from short motion capture clips from the CMU dataset we initialise the system using the pretrained weights for the PFNN. A new residual adapter is then learned from scratch with the PFNN weights held fixed.

3.4.2.2 Building a Real-Time Demo

To create a real-time demo we implement our neural network in a game engine (Unity, (Haas, 2014)) and predict the pose for the next frame given the current pose and user controlled trajectory. We blend the trajectory predicted by our model with a keyboard controlled trajectory using the method of Zhang et al. (2018a). To increase speed at runtime we discretise phase functioned weights in the same way as Holden et al. (2017b), saving 50 matrices for different phase values.

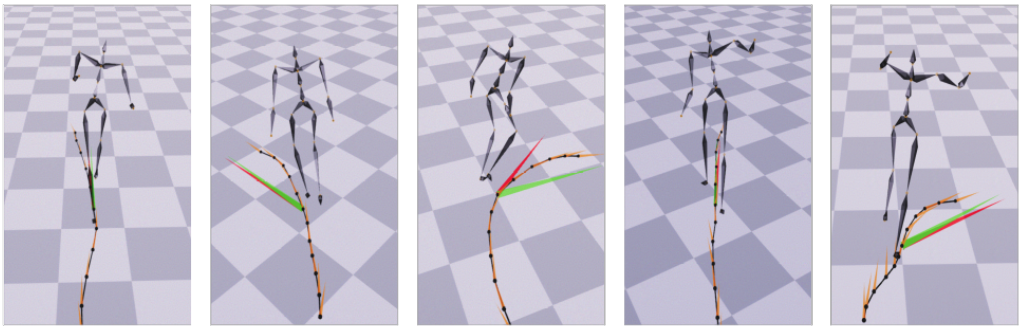


Figure 3.6: Modelling multiple styles learned from limited data. From left to right the styles shown are: *joy, on toes crouched, roadrunner, wild arms & zombie*.

3.5 Evaluating the Residual Adaptation Approach

Having pretrained the model and learned new residual adapters for each of the few-shot styles we can evaluate our approach and compare the results to other systems. As this system is an animation system designed for real-time applications the majority of the comparisons are based on qualitative performance. For this reason, a video demonstrating the results discussed in this section can be found at `https://www.youtube.com/watch?v=Q-agrHm-ztU`. Examples of modelling different styles are shown in Figure 3.6 and the accompanying video.

In general the quantitative evaluation of generative models is a challenging topic (Theis et al., 2016). Evaluations based on model error may be biased towards overfitted models and may not correlate with perceived perceptual quality (Martinez et al., 2017). Whilst there exist quantitative evaluation techniques for very specific aspects of animation systems, such as measuring foot sliding artifacts (Zhang et al., 2018a) or computational performance (Holden et al., 2020), the development of general quantitative techniques for evaluating animation quality is a largely unexplored area in which evaluation metrics for different tasks may be orthogonal (Theis et al., 2016). At the time of writing, we are not aware of any good quantitative metrics for evaluating the overall perceptual quality of an animation system and so must primarily rely on a subjective evaluation of animation quality.

3.5.1 Evaluating Our Model

We first consider how our approach compares to other models that we create in order to try and learn new styles from low amounts of data. The simplest approaches of

Table 3.2: Computational comparisons with changing residual adapter capacity. Full adapters, with high capacity, are the most expensive and tend to overfit. Diagonal adapters, with low capacity, are the cheapest but cannot model styles correctly due to underfitting. Memory column, the storage requirements for a new residual adapter, the two values show storing only the control points at runtime and storing 50 weight matrices with a discretised phase respectively. Training column, the mean training time for a new style. Runtime column, the time for a forwards pass of the network.

Method	Memory (MB)	Training (s)	Runtime (s)
Full Adapter	4.01 / 50.1	99	0.0013
Diagonal Adapter	0.016 / 0.20	44	0.0010
CP Decomposition	0.126 / 0.131	50	0.0011

training the whole PFNN from scratch using only the data from a new style, or fine-tuning a PFNN learned on a large dataset of locomotion, heavily overfit. This creates unrealistic poses even for small changes in the input trajectory unseen in the training data, see Figure 3.4.

We also compare our CP decomposition to residual adapters learned with different capacities. In particular, we examine 'full' residual adapters, where we use no decomposition, learning a 512×512 phase-functioned weight matrix for each style, and 'diagonal' residual adapters, where we restrict the weight matrix to be diagonal with 512 phase-functioned parameters. Table 3.2 shows the computational differences between these approaches. Note that the full residual adapter, which increases the original PFNN parameter count by 44.7% for each new style, is significantly more expensive in terms of storage than the other approaches. This table also highlights another benefit of learning with small amounts of data, that is, extremely fast training times (less than a minute compared to 6 hours for pretraining).

Qualitatively we find that, whilst the full residual adapters can well model most styles they have a tendency to overfit due to having too much capacity. This manifests as a character which does not respond well to user control, preferring to mimic the training data. For example, walking in a straight line when the user tries to turn left or right. With diagonal residual adapters we found the opposite problem of insufficient capacity to cause underfitting. This means that the system failed to learn to model styles well, outputting some type of average motion rather than stylised poses. In contrast, we find that using the CP decomposition to set a suitable capacity allows our model to

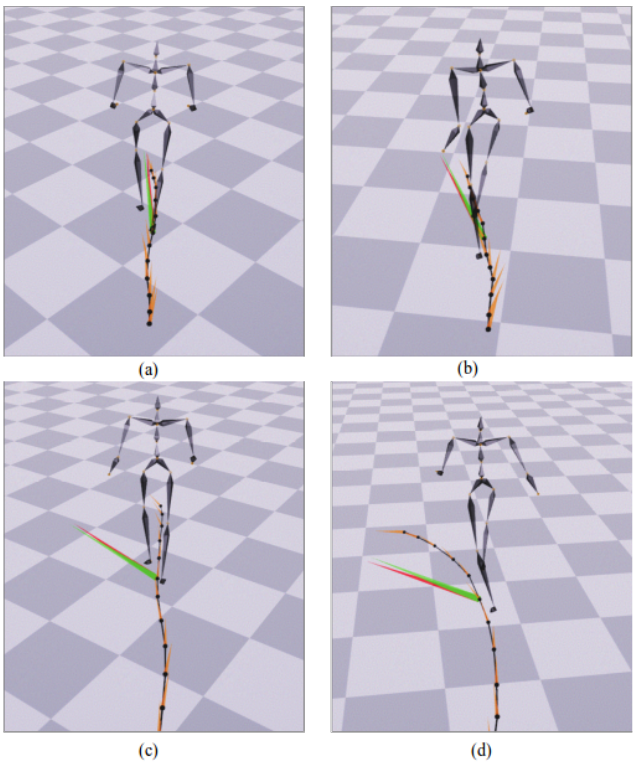


Figure 3.7: Underfitting and overfitting to different styles as residual adapter capacity changes. (a) *Dinosaur* style, CP decomposed residual adapter, reasonably capturing the style. (b) *Dinosaur* style, diagonal residual adapter, the style is captured poorly due to underfitting. (c) *Elated* style, full residual adapter, the style is well modelled but the character struggles to turn left. (d) *Elated* style, CP decomposed residual adapter, the style is still well modelled but the character can turn more easily.

generalise better. Examples of overfitting and underfitting are shown in Figure 3.7 and the accompanying video.

To some extent we can visualise the role of the style agnostic parameters by setting the residual adapter weights to 0. This visualisation, seen in the accompanying video, is fairly close to a neutral walk and represents a reasonable underlying agnostic motion for our style dataset. However, it may well be that the style agnostic parameters could be improved upon given larger amounts of data.

We also note the importance of training on multiple styles (domains) during pretraining (Arjovsky et al., 2019). If we train only on neutral motion the network cannot learn to separate style and content factors in the data. If we then try to learn a new style, this leads to difficulties with the network often outputting motions that are quite close to the neutral data. This happens because we have no changing domains to demonstrate

what is content ('the things that are the same *across domains*') and what is style ('the things that are different *across domains*').

3.5.2 Comparisons with a Style Transfer Approach

We additionally compare our style modelling approach with the style transfer approach of Holden et al. (2016). Holden et al. learn an autoencoder, to learn a manifold of human motion, and then optimise in latent space to transfer the style of one clip onto the content of another using a Gram matrix approach (Gatys et al., 2015). Although not designed for style modelling, in that a user cannot interactively control the motion content using this method, we can still compare outputs. To do this we select a content clip in the *neutral* style and use Holden et al.'s method to stylise this clip using the style from one of our clips for few-shot learning. We then take the trajectory of the stylised motion and use this as input to our system in order that both motions may follow a similar path.

Results of this process for the *balanced*, *crossover*, and *chicken* styles are shown in the accompanying video. These results show that our method usually produces results that are qualitatively at least as good as Holden et al.'s and often capture many more style details. However, if data is extremely limited, optimising using the Gram matrix can produce superior results, albeit for a single content clip without learning a general representation of the style (as we do with our residual adapter parameters).

3.5.3 Reflections on the Residual Adaptation Approach

Our system is able to learn to model new styles of locomotion given only a few frames of data (Appendix A.1). The majority of styles we evaluate on can be generalised to new turning directions and realistic changes in movement speeds. However, some styles present problems for our approach. Here we reflect on two challenges that arise when using our system, namely generalising beyond new turning directions to new motions (heterogeneous transfer) and modelling styles that have no clear global phase (non periodic motions).

3.5.3.1 Heterogeneous Transfer

Our results are strongest for homogeneous style modelling. That is, given a short clip of walking data, we can utilise the style agnostic parameters to help generalise to new

walking directions and speeds. However, some works (Yumer and Mitra, 2016) have shown an ability to perform heterogeneous style transfer, that is, to take the style of one motion (e.g. walking) and map it onto the content of a different motion (e.g. jumping, punching).

For our approach, heterogeneous transfer is difficult because, given motion capture data of a character walking in a straight line there is no deterministic mapping that tells us how this style manifests when running or jumping say. Learning to perform heterogeneous style modelling in animation synthesis systems is very desirable. Whilst we implemented a number of methods to try and achieve this including layer normalisation (Ba et al., 2016) and a Fourier based loss function inspired by Yumer and Mitra (2016), we were not able to achieve consistent high quality heterogeneous transfer.

For style modelling approaches, the easiest way to achieve some type of heterogeneous transfer would be to collect larger, more varied datasets of stylised locomotion to pretrain on. Our 8 style dataset contains only forwards walking and running movements, so larger datasets with more types of movements and more styles to learn how these movements relate across styles (domains) are desirable.

3.5.3.2 Non Periodic Motions

Whilst our pretraining data contained only globally cyclic styles, several of the new styles we try to learn do not have such a global phase cycle. Those styles which are globally cyclic tend to be well modelled with our phase-functioned system. However, those styles that have a poorly defined global phase, because of different limbs moving with different phases or stochasticity (e.g. *wild arms* or *drunk*.), are difficult to reproduce with our model.

Such styles likely require architecture changes as the PFNN's inductive bias required highly regular periodic motion, summarisable with a single phase value. Furthermore, our use of only cyclic motions and the PFNN to learn style agnostic parameters may prevent the learning of truly agnostic parameters as we are restricted to using a global phase. In the rest of this chapter we investigate potential solutions to these problems, creating a larger, more varied, dataset and relaxing the PFNN's strong globally cyclic assumption.

3.6 Harder Style Modelling - The 100Style Dataset

Having discovered some limitations of our 8 style dataset from this initial exploration of few-shot style learning, we iterate on this dataset to create a range of styles that are intentionally difficult for style modelling as well as collecting much more data. The dataset we create contains over 4 million frames of locomotion in 100 different styles. We call this dataset 100STYLE and provide a video of the styles captured: `https://youtu.be/ZPj_7Ewe3eU`.

3.6.1 The 100Style Dataset

Along with our 8 style dataset, several other datasets have been released for animation research. However, these datasets tend to either contain a large number of motion capture frames for a small number of styles (as in our dataset, Ionescu et al. (2014), and Xia et al. (2015)), or a large number of styles with a small number of frames per styles (as in Gross and Shi (2001)). It is this lack of available data which led us to focus on few-shot learning with the PFNN. Other authors have developed augmentation techniques (Lee et al., 2018), methods that utilise 2*D* video data (Aberman et al., 2020), and hand crafted solutions (Yumer and Mitra, 2016) for the same reasons. This lack of data also restricts the evaluation of style transfer and modelling systems, often to small numbers of globally cyclic styles (Holden et al., 2017a; Smith et al., 2019). To address these issues we create the 100STYLE dataset which contains more styles of locomotion than previous datasets with many more motion capture frames for each style.

The 100 styles we capture are designed to provide a range of challenges for style transfer and modelling systems. In particular we intentionally capture styles that contain stochastic movements, are asymmetric, and have no clear global phase cycle. Figure 3.8 shows examples of some of the styles captured in 100STYLE. Appendix A.2 gives a full list of styles and the number of frames of motion capture per style.

This dataset is captured using Xsens motion capture technology which captures joint positions and rotations for a 28 bone skeleton. Multiple gaits of motion are captured according to predetermined scripts, the scripts are given in Appendix A.2 and a breakdown of the dataset by number of frames captured per gait is shown in Table 3.3. The data is captured at 60fps with a single actor, 182cm tall, in a 4.5m $\times$ 4.5m capture area. The final dataset contains approximately 19 hours of motion capture data.

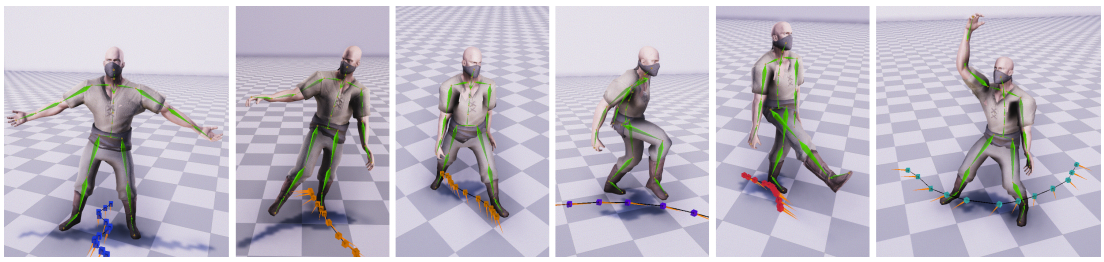


Figure 3.8: Examples of styles in the 100STYLE dataset. From left to right: *Star* - Backwards Walk; *Drunk* - Forwards Walk; *Drag Right Leg* - Forwards Walk; *Roadrunner* - Forwards Run; *Kick* - Backwards Walk, and *Wild Arms* - Sidestep Run. The *Drunk* style is stochastic, *Drag Right Leg* is asymmetric and *Wild Arms* has joints moving with different phase cycles (no global phase cycle).

Table 3.3: The 100STYLE dataset broken down by frames captured per gait.

Motion Gait	Frames	Time(m)	Ratio(%)
Backwards Run	434,329	120	10.7
Backwards Walk	768,574	213	19.0
Forwards Run	414,911	115	10.2
Forwards Walk	777,640	216	19.2
Sidestep Run	250,980	70	6.2
Sidestep Walk	541,041	150	13.3
Idling	81,685	23	2.0
Transitions	786,818	218	19.4
Total	4,055,978	1125	100.0

3.6.2 Data Processing

The data captured is processed in much the same way as in Section 3.4.1, with symmetric styles being mirrored and extracting input frame and output frame representations. We also extract short clips of stylised data.

Given this improved dataset, we do however make some improvements and changes to the data processing pipeline. Namely: we use a 25 bone skeleton instead of a 31 bone skeleton (discarding unneeded head top and toe joints from the motion capture); we extract foot contact predictions in the output; we add rotations to the input frame (as Zhang et al. (2018a) find it useful for gated expert systems), and we extract *local* phase variables as will be explained in Section 3.7. The upshot of this new processing is that we extract input frames, $x_i \in \mathbb{R}^{348}$, output frames, $z_i \in \mathbb{R}^{342}$ and clips of stylised

motion, $y \in \mathbb{R}^{240\times300}$.

The input frame, $x_i \in \mathbb{R}^{348}$, consists of: 2×12, x and z planar trajectory positions; 2×12, x and z planar trajectory directions; 3×25, x, y and z joint positions; 3×25, x, y and z joint velocities; 3×25, x, y and z joint rotation forward components; 3×25, x, y and z joint rotation upward components. All relevant values are calculated in the character's local co-ordinate frame. 8 local phase variables are also extracted from each input frame (see Section 3.7).

The output frame, $z_i \in \mathbb{R}^{342}$, consists of: 2×6, x and z current and future planar trajectory positions; 2×6, x and z current and future planar trajectory directions; 3×25, x, y and z joint positions; 3×25, x, y and z joint velocities; 3×25, x, y and z joint rotation forward components; 3×25, x, y and z joint rotation upward components; 8 predicted local phases; 8 predicted local phase updates, and 2 approximate foot contact labels. The output frame is calculated in the input frame's local co-ordinate frame, this allows us to implicitly model the global character translation.

The stylised clip, of motion $y \in \mathbb{R}^{240\times300}$, consists of: 3×25, x, y and z joint positions; 3×25, x, y and z joint velocities; 3×25, x, y and z joint rotation forward components, and 3×25, x, y and z joint rotation upward components calculated in the local co-ordinate frame for 240 consecutive frames. We do not add the trajectory to the style clips as we are not interested in the specific motion in the clips, only the spatio-temporal joint relationships.

3.7 Improved Animation Synthesis with Local Motion Phases

With this new large dataset we now ask how can we improve the quality of style modelling systems. In particular, we no longer focus on the low data regime, instead focusing on engineering and design decisions to better allow neural networks to model multiple styles of locomotion. We will first consider the animation synthesis network, in particular changing the PFNN architecture to better accommodate motions that are not globally cyclic. Afterwards we will look at learning a style representation from the clips of motion capture data we extract from the 100STYLE dataset.

To improve animation synthesis we take inspiration from works created during the development of this thesis that use gated experts systems to capture local movements

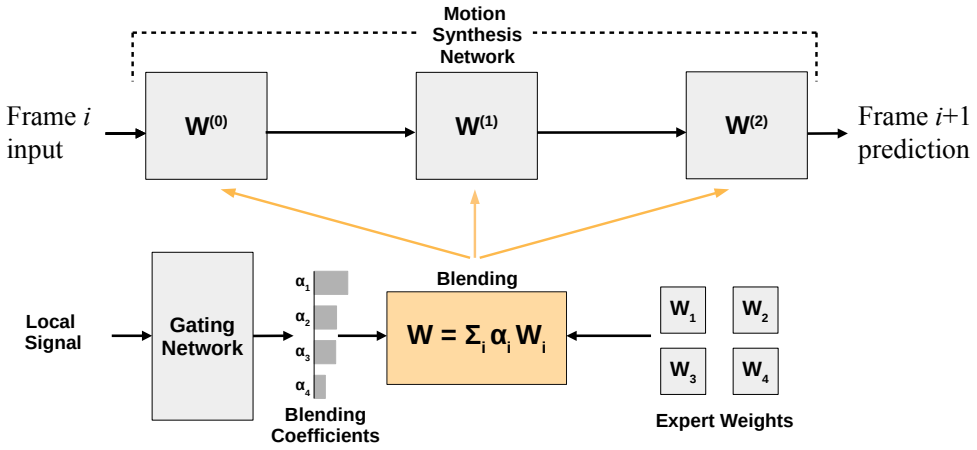


Figure 3.9: A Gated Experts System. As with the PFNN, weights used for prediction are set by blending parameters. Rather than being manually defined, the blending is learned dependent on some local signal.

in locomotion (Starke et al. (2019, 2020); Zhang et al. (2018a), Figure 3.9). A gated expert system consists of 2 sub-networks, firstly a *gating network*, $\mathbf{\Omega}$, that uses some local signal to predict blending coefficients $\{\alpha_i\}_{i=1}^{K}$. These coefficients are used to linearly blend expert weights, $\{\mathbf{W}_i\}_{i=1}^{K}$, (analogous to control points in the PFNN, these expert weights are possible settings of a neural network's parameters). This idea was originally developed in order to learn, rather than define manually, the blending of control points used in the PFNN in Equation 3.1 (Zhang et al., 2018a). Once blended, the output weights, $\mathbf{W} = \sum_i \alpha_i \mathbf{W}_i$, form the weights of a *motion synthesis network*, $\mathbf{\Phi}$, which is used to predict the next frame from the input frame.

3.7.1 A Hybrid Approach to Local Phases

Recently Starke et al. (2020) have proposed to use a local phase as input to the gating network of a gated experts system. The local phase aims to capture the phase cycles of individually specified joints, rather than one global phase value, and use these signals to condition how to blend expert weights. Since several of the styles in the 100STYLE dataset are not globally cyclic and have joints moving with different phase cycles we repurpose this approach to model locomotion styles. We call a gated experts system that uses local phases for gating a Local Phase Network (LPN).

However, the local phases used by Starke et al. are created using contacts of specific joints with the environment (the floor plane, objects, etc.). With stylised motion we do

not always have contacts with the environment, for example the arms may move freely without interacting with anything in the scene. Therefore we must additionally extend the local phase creation to the contact-free setting.

Our approach to creating local phases is to use the method of Starke et al. when contacts are available and to use our own contact-free method when they are not. To create local phases, we fit a sinusoidal function to some function of the joint we are interested in. We refer to the function we fit as a source function, $G(t)$ (Starke et al., 2020, Section 5.2). This fitting process is able to flexibly fit many functions, so in order to fit local phases we must simply design a reasonable source function.

3.7.2 Contact-Based Local Phases

Our contact-based source function is a step function defined by the presence of contacts (or lack thereof) with the floor plane. We extract approximate foot contact labels, for a given foot, by thresholding the foot bone velocities and distance from the ground. If both the ball centred at the end of the foot with radius d_{max} intersects the ground plane and the foot velocity v_{max} is below some threshold we label the frame with a contact (1) and if not with no contact (0). This creates the step function which aims to be 1 when the foot in question is touching the ground and 0 otherwise. d_{max} and v_{max} can be adjusted for different styles but we use default values of $d_{max} = 0.01$ and $v_{max} = 0.15$.

3.7.3 Contact-Free Local Phases

For the case where contacts are not available for a given bone, we aim to design a source function that cycles with the bone's movements. To do this, we separate the motion capture clips for a given style into the different gaits shown in Table 3.3. For a given style with a given gait we use PCA to calculate the 3 principal components of the bone's position, in the local co-ordinate frame, over the motion clip. Before fitting PCA we standardise by the mean position but not by the variance. This is because we do not care about the bone's average location relative to the root, but the variance in the bone's location is precisely what we aim to capture with our source function.

The source function is then created by projecting the (centred) bone's position onto the first principal component. Note that we must also direct this principal component. This is because if we calculate local phase cycles for two bones moving either in sync or out of sync by half a phase cycle, if the principal component is not directed there is no way

to tell the difference between these situations. To direct the principal component we check if it is positive along the character's forwards facing axis and flip the direction if not.

The resultant source function is approximately sinusoidal for locally cyclic movement and is large in absolute value when the bone is far away from its mean position along a specific axis (the first principal component). This can be seen in the top row of Figure 3.10, where the uppermost 1D source function (white curve) is sinusoidal in shape.

3.7.4 Processing Source Functions and Calculating Local Phases

The source functions described above for contact-based and contact-free situations are post-processed before local phase values are calculated for each frame. We use the same post-processing as Starke et al. (2020), normalising the function over a one second window and applying a Butterworth filter to the source functions. The filter smooths the steps in the contact-based situation and filters out some noise in the contact-free case.

The evolutionary fitting process of Starke et al. fits a function of the form $a_i \cdot \sin(f_i \cdot t - s_i) + b_i$, where i is the frame index, t the timestamp input to source function $G(t)$, and $\{a_i, f_i, s_i, b_i\}$ parameters optimised during fitting. Phase values, for frame i with timestamp t, are then extracted from this fitted function as $\phi_i = f_i \cdot t - s_i \pmod{2\pi}$. We then extract a $2D$ phase representation,

$$\mathbf{p}_i^{(b)} = ||\mathbf{v}_w|| \cdot a_i \cdot \begin{pmatrix} \sin\phi_i \\ \cos\phi_i \end{pmatrix}, \tag{3.7}$$

where $\mathbf{p}_i^{(b)}$ is a $2D$ phase vector for frame i and bone b. $||\mathbf{v}_w||$ is the maximum bone velocity over a one second window, w, centred on frame i. Scaling by this velocity significantly reduces the phase magnitude when bones are stationary and the local phase is not clearly defined.

We choose to extract local phases for the end effector bones, that is, the hands and the feet. We use the contact-based method whenever contacts are available (most of the time for the feet) and the contact-free method when contacts are unavailable (all of the time for the hands). Figure 3.10 shows examples of the local phase extraction for three different styles. In this figure we show the directed first principal components with magenta arrows and planes perpendicular to the arrows, centred at the mean of

the bone in question. Intuitively the local phase function is positive when the bone is in front of the plane and negative when it is behind it.

3.8 Style Modulation with Feature-Wise Linear Modulation

Given that we are now working with larger amounts of data we can also reconsider how style is modelled in our system. In particular, we investigate the use of FiLM parameters (feature-wise linear modulation, Perez et al. (2018)) for learning a compact representation of locomotion style. We learn our *style modulation network*, $\mathbf{\Psi}$, as a FiLM generator which outputs the FiLM parameters given an input clip of stylised locomotion. The FiLM parameters modulate the hidden units of the motion synthesis network, $\mathbf{\Phi}$ (the network with blended expert weights).

In the broader context of this thesis, the motion synthesis network has parameters shared across all styles and should capture the content factors in the data. The style modulation network should capture the style factors present in a given input clip of stylised motion, $\mathbf{y}$. By changing only the style factors, that is changing the input clip of stylised motion, we can change the style of generated motions without changing the content.

3.8.1 Style Modulation Network

FiLM parameters (Perez et al., 2018) are learned, element-wise, scale and shift parameters that are applied to the hidden unit of a neural network. The FiLM generator outputs FiLM parameters for each of the two hidden layer of the motion synthesis network,

$$\mathbf{\Psi}(\mathbf{y}) = \left\{ \boldsymbol{\gamma}^{(1)}, \boldsymbol{\beta}^{(1)}, \boldsymbol{\gamma}^{(2)}, \boldsymbol{\beta}^{(2)} \right\}, \tag{3.8}$$

where $\boldsymbol{\gamma}^{(i)}$ and $\boldsymbol{\beta}^{(i)}$ are the scale and shift vectors of FiLM parameters for layer i.

Similar to Aberman et al. (2020), we additionally apply layer normalisation (Ba et al., 2016) to the hidden representations before applying the affine FiLM transformation. This creates a form of conditional normalisation (Dumoulin et al., 2017). We can summarise the application of this conditional normalisation, for a single frame x_j with

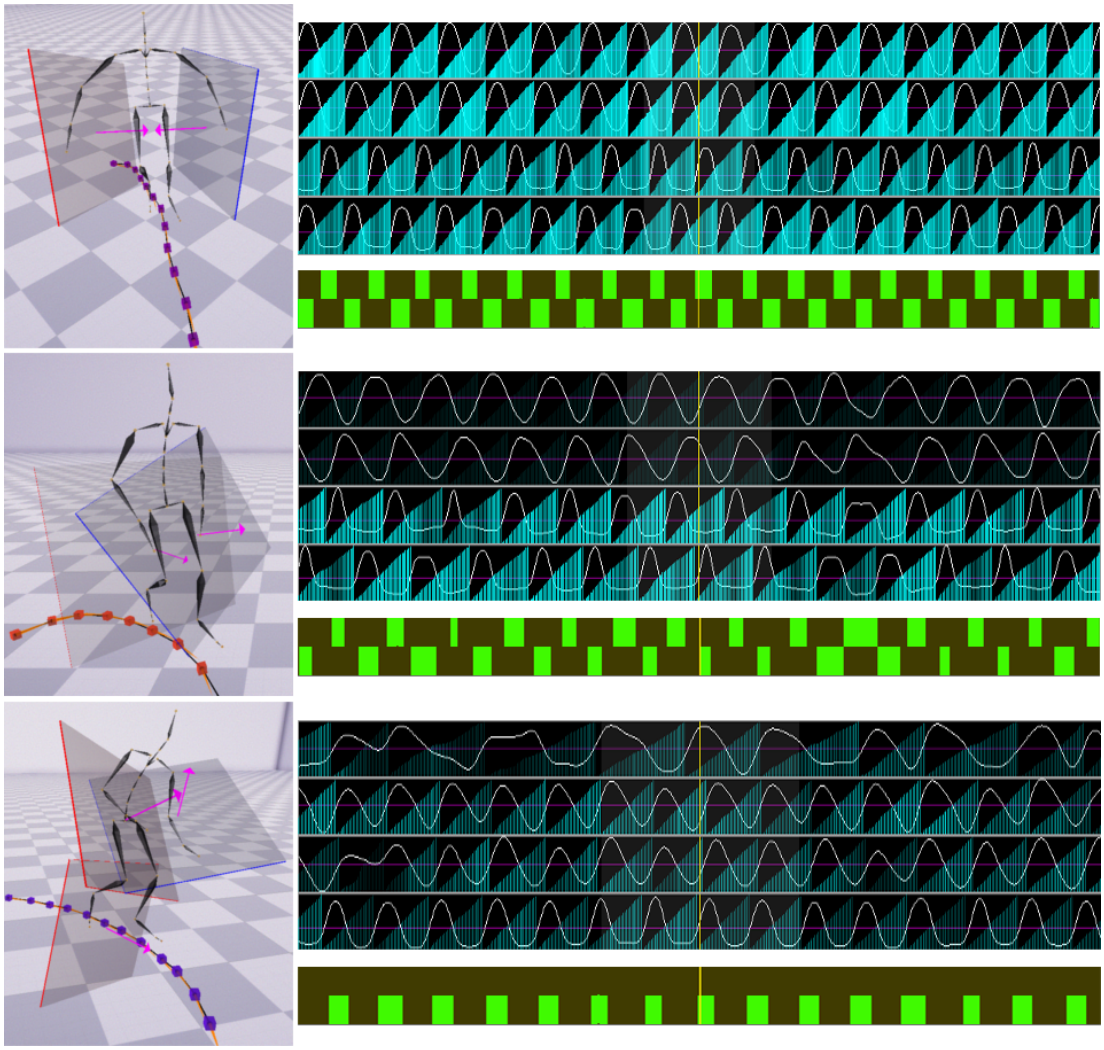


Figure 3.10: Local Phase Labelling. Top to bottom the styles are: *Pendulum Hands* (contact-free and contact-based local phases create clear cycles); *Hands In Pockets* (the phase has a negligible magnitude for the hands due to low bone velocity), and *Left Hopping* (we can use the contact-free method for feet if contacts are not available). In this figure the white curves show the source functions over many frames. From top to bottom the source functions are for the right hand, left hand, right foot and left foot. The blue bars show the extracted local phases with transparency representing the phase magnitude, $||\mathbf{v}_w|| \cdot a_i$. More transparency corresponds with lower magnitude. The green bars show foot contact detections. The frames visualised on the left correspond to the locations of the vertical yellow bar in the source functions and foot contacts.

index j, as

$$\mathbf{h}_j^{(i)} = ELU\left(\boldsymbol{\gamma}^{(i)} \circ \left[\frac{\boldsymbol{F}_j^{(i)} - \boldsymbol{\mu}_j}{\boldsymbol{\sigma}_j}\right] + \boldsymbol{\beta}^{(i)}\right), \tag{3.9}$$

where, for sample j, $\boldsymbol{F}_j^{(i)}$ is the output features of layer i in motion synthesis network, $\boldsymbol{\Phi}$, and $\mathbf{h}_j^{(i)}$ is the input to layer $i+1$. ELU is the exponential linear unit; $\boldsymbol{\gamma}^{(i)}$ & $\boldsymbol{\beta}^{(i)}$ are as described in Equation 3.8, and $\boldsymbol{\mu}_j$ & $\boldsymbol{\sigma}_j$ are the layer normalisation statistics, that is, the mean and standard deviation taken over $\boldsymbol{F}_j^{(i)}$. $\circ$ is the element-wise product.

3.8.2 Theoretical Analysis

Like the residual adaptation approach, FiLM parameters also modulate the learned hidden representations of an input frame[4]. In fact, a one-hot labelling of style (Holden et al., 2017b; Smith et al., 2019) effectively learns a bias term that can be viewed as a style specific learned offset applied to the first layer's hidden units. As discussed in the related work (Section 3.1), applying affine transformations to hidden representations is a relatively common approach for style transfer and modelling. What we note here is that these affine transformations can be viewed as a special case of the Multi-Task Dynamical Systems (MTDS) framework (Bird and Williams, 2019).

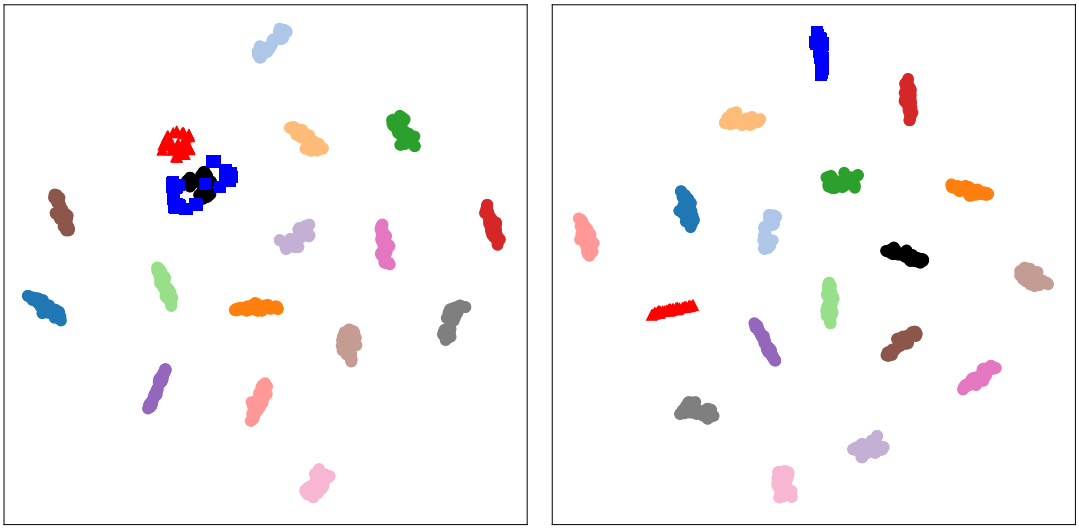
MTDS is a principled probabilistic framework for customising time series models to individualised sequences (or, in the language of this thesis, adapting a time series model to a new domain). For locomotion style, this results in modelling styles using whole layers of neural network weights (Bird, 2020, Figure 5.5). In our case, the affine transformations of hidden units discussed above can be seen as specific restricted ways in which layers of neural network weights can be modified.

To see why this is the case we write a feed-forward layer as $\boldsymbol{h}^{(i+1)} = \sigma(\boldsymbol{W}\boldsymbol{h}^{(i)} + \boldsymbol{b})$, where $\boldsymbol{h}^{(i)}$ is the hidden units for layer i, $\boldsymbol{W}$ & $\boldsymbol{b}$ the layer parameters and σ some non-linearity. We then write the element-wise scaling FiLM parameters, $\boldsymbol{\gamma}^{(i)}$, as a diagonal matrix, $\boldsymbol{W}^s$, with the scaling parameters on the diagonal. Keeping the shift parameters as vector $\boldsymbol{\beta}^{(i)}$, we write a layer modulated by FiLM parameters as

$$\boldsymbol{h}^{(i+1)} = \sigma\left(\boldsymbol{W}^s(\boldsymbol{W}\boldsymbol{h}^{(i)} + \boldsymbol{b}) + \boldsymbol{\beta}^s\right) = \sigma\left(\boldsymbol{W}'\boldsymbol{h}^{(i)} + \boldsymbol{b}'\right), \tag{3.10}$$

where $\boldsymbol{W}' = \boldsymbol{W}^s\boldsymbol{W}$ and $\boldsymbol{b}' = \boldsymbol{W}^s\boldsymbol{b} + \boldsymbol{\beta}^s$ are the new layer parameters. Similar analysis can be easily performed for one-hot vector and residual adapter modulation. This

[4]Residual adapters add an offset to the hidden units which changes as the network input changes.



(a) Before Fine-Tuning (b) After Fine-Tuning

Figure 3.11: t-SNE for FiLM parameters. We show embedding for 50 different stylised motion clips for 15 style that the FiLM generator is pretrained on (pastel colours). We also show 50 embeddings for 3 styles unseen during pretraining (black circles, blue squares, red triangles). (a) Before fine-tuning the style the FiLM generator is trained on are well separated but the unseen styles are not. (b) After fine-tuning all styles are well separated.

shows how MTDS and affine transformations of hidden units relate, in particular the transformations of units we perform are a specific manipulation of layer parameters as done more generally by Bird (2020).

We point out however, that despite the generality of learning style specific layers, this process is very expensive as it requires capturing style variation with whole layers of neural network parameters[5]. Our methods, which share style agnostic parameters across styles, allow us to learn much more compact representation of styles which are both cheaper to store and faster to calculate with. We also posit that, by forcing our model to learn to represent styles as an affine transformation of the hidden units, this encourages linear interpretability of the learned style manifold and hence more plausible linear interpolations between styles.

3.8.3 Learning New Styles

Whilst the 100STYLE dataset contains a large amount of variation in human locomotion, we do not have a dense enough sampling to expect to achieve generalised learning of FiLM parameters for unseen styles as we effectively have 100 samples from the 'style space of human locomotion' (Ji et al., 2021). However, we would still like to be able to model new styles of locomotion given new data.

Much like with the PFNN, we can view the parameters of the LPN as the style agnostic parameters and the FiLM parameters as the style specific parameters. Whilst the FiLM generator can have arbitrary capacity, the FiLM parameters themselves have size equal to the number of hidden units in the layer they transform. In particular, after training the FiLM generator, FiLM parameters can be calculated and saved offline for use with the LPN. This means we can approximately reconstruct our 100STYLE training data with 100 such sets of FiLM parameters and the LPN parameters. Therefore, given some new stylised locomotion data, we simply need to learn a reasonable set of FiLM parameters that can be used with the trained LPN parameters. To do this we can simply fine-tune the FiLM generator on the new style data while holding LPN parameters fixed.

Figure 3.11 visualises this argument using a t-SNE visualisation (Van der Maaten and Hinton, 2008). Figure 3.11a shows that, given new styles of locomotion the FiLM parameters extracted (without fine-tuning) are not well separated by style, that is, the FiLM generator does not provide robust style generalisation. Figure 3.11b shows that, by holding LPN parameters fixed and fine-tuning the FiLM generator, we are able to extract well separated FiLM parameters for new styles using this fine-tuning approach.

3.9 A Complete System for Style Modelling in Human Locomotion

We combine our improved, local phase based, animation synthesis network with FiLM style modelling to create a single architecture for modelling the 100STYLE dataset. This architecture can be trained end to end and is shown in Figure 3.12 where $\mathbf{W}^{(l)}$ denotes the l^{th} layer of the motion synthesis network. In this figure the dashed red

[5]This would amount to $512 \times 327 \times 4 \approx 670,000$ parameters for each style if we learned the last layer of a PFNN to be style specific.

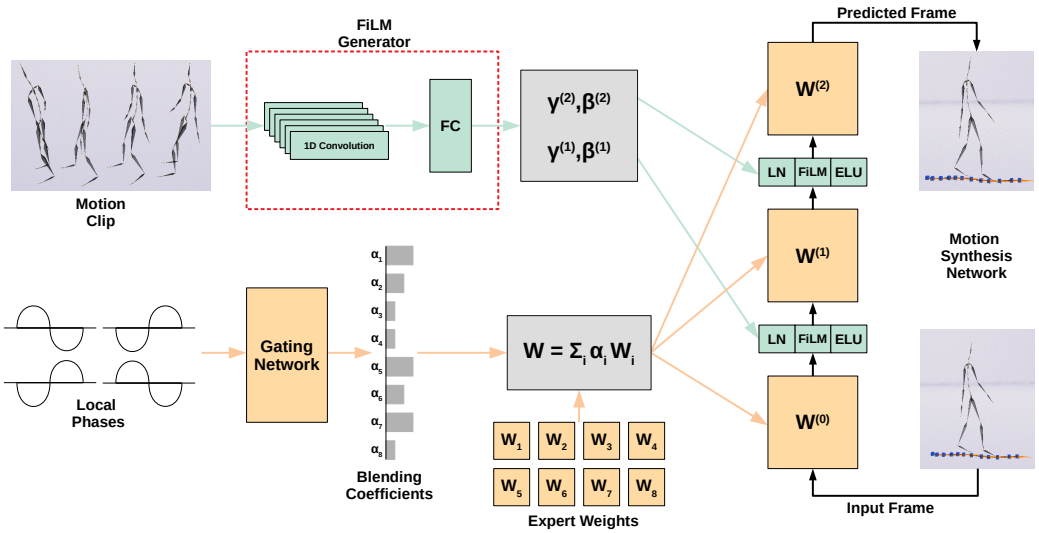


Figure 3.12: 100STYLE Style Modelling System. Animation synthesis (peach) is performed using a local phase network that uses a gating network to blend expert weights to create the parameters of a motion synthesis network. Style modulation (green) is done using FiLM parameters which modulate the hidden units of the motion synthesis network. LN is layer normalisation (Ba et al., 2016).

box represent the calculation of the FiLM parameters (Equation 3.8), it is this network that is fine-tuned to learn new styles. The green boxes represent applying the FiLM parameters to the motion synthesis network as in Equation 3.9.

3.9.1 Training & Implementation

We predict output frame $\mathbf{z}_j^s$ from input frame $\mathbf{x}_j^s$ for style s as

$$\hat{\mathbf{z}}_j^s = \boldsymbol{\Phi}\left(\mathbf{x}_j^s; \boldsymbol{\Omega}(\mathbf{p}_j^{(1)}, \mathbf{p}_j^{(2)}, \mathbf{p}_j^{(3)}, \mathbf{p}_j^{(4)}), \boldsymbol{\Psi}(\mathbf{y}^s)\right), \tag{3.11}$$

where $\hat{\mathbf{z}}_j^s$ is the predicted output frame, $\boldsymbol{\Phi}$ is the motion synthesis network whose parameters are created using the output of the gating network, $\boldsymbol{\Omega}$. $\mathbf{p}_j^{(i)}$ are the phase vectors for the 4 end effectors calculated with Equation 3.7. The hidden units of the motion synthesis network are transformed using the output of the FiLM generator, $\boldsymbol{\Psi}$, calculated using stylised clip of motion in style s, $\mathbf{y}^s$.

The motion synthesis network, $\boldsymbol{\Phi}$, is a 3 layer feed-forward neural network with both hidden layers being of size 512. Gating network, $\boldsymbol{\Omega}$, is also feed-forward and 3 layers with both hidden layers of size 32, the output is of size 8 for the 8 sets of expert weights we blend. The FiLM generator, $\boldsymbol{\Psi}$, uses $1D$ convolutional layers with 256 filters that

convolve over the time dimension of the motion clip with a window size of 25. 2 such layers are used with max-pooling after convolutions, 2 fully connected layers follow with hidden units of size 2048 and 2048 output units (512 element-wise scale and shift parameters for the 2 motion synthesis network hidden layers).

Before training the data is standardised using mean and variance values calculated over all training styles. We train using 95 of the styles in the 100STYLE dataset, with the 5 held out styles usable for evaluating fine-tuning (Section 3.8.3). To train we optimise the mean squared error objective, $\mathcal{L}^{mse}$, between actual and predicted output frame. We additionally find a bone length loss, $\mathcal{L}^{bll}$, useful to prevent bone stretching artifacts. We regularise training with a dropout rate of 0.3 everywhere Srivastava et al. (2014). Our overall objective function is $\mathcal{L} = \mathcal{L}^{mse} + \mathcal{L}^{bll}$,

$$\mathcal{L}^{mse} = \sum_{s=1}^{S} \frac{1}{N_s} \sum_{j=1}^{N_s} (\mathbf{z}_j^s - \hat{\mathbf{z}}_j^s)^2, \tag{3.12}$$

$$l^{bll}(\mathbf{z}, \hat{\mathbf{z}}) = \frac{1}{B} \sum_{b=1}^{B} \left| ||z_{b_p} - z_b||_2 - ||\hat{z}_{b_p} - \hat{z}_b||_2 \right|, \tag{3.13}$$

$$\mathcal{L}^{bll} = \sum_{s=1}^{S} \frac{1}{N_s} \sum_{j=1}^{N_s} l^{bll}(\mathbf{z}_j^s, \hat{\mathbf{z}}_j^s). \tag{3.14}$$

Here s is a style label, $s \in \{1, \ldots, S\}$, and N_s is the number of training data frames for style s. z_b is the $3D$ position vector for bone b in ground truth output frame $\mathbf{z}$. z_{b_p} is the position vector for the parent joint of bone b. $\hat{z}_b$ & $\hat{z}_{b_p}$ are the same joint positions but for the predicted output frame $\hat{\mathbf{z}}$. B is the number of bones in the character skeleton.

We hold out 10% of the data for quantitative test set evaluations and 10% for a validation set. The remaining data is used to train the network (Figure 3.12). We use minibatch training with each minibatch containing data from a single style and to ensure balanced training we again cycle through the training styles so every S^{th} minibatch contains the same style of motion. We define one epoch as one pass through the style data with the smallest number of motion capture frames and train for 90 epochs using Adam (Kingma and Ba, 2014) with a learning rate of 1×10^{-4}.

3.9.2 Improving the Real-Time Demo

As for the residual adaptation approach we create a real-time user-controllable demo where animation frames are predicted using our style modelling system. As our network predicts foot contacts we additionally add foot contact inverse kinematics to the

demo. This means that when a foot contact is predicted we lock the foot in place on the floor to reduce foot sliding artifacts. We also skin the character using the Mixamo (Adobe, 2015) Y bot skin. Finally, we note some small skinning artifacts in the demo so apply inverse kinematics along each arm to move arm joints to the predicted positions which helps prevent incorrect rotations.

As discussed, we can precalculate FiLM parameters offline to increase speed at runtime, and to reduce storage costs as we do not need to store the FiLM generator. We can save a single set of FiLM parameters for each style either using a single clip of data or averaging over multiple clips. To experiment with how well our model is able to interpolate between styles we add a 3-way interpolation system that blends FiLM parameters based on the barycentric co-ordinates at a triangle location selected by the user.

3.10 Evaluating the Local Phase Based Style Modelling System

With our new system for style modelling, we qualitatively evaluate by demonstrating failure cases of other systems that we are able resolve. As we have a large number of styles to work with we can evaluate the style modelling capacity of different systems and highlight specific styles that prove challenging to model. We again provide a video of animation results: `https://youtu.be/mnlnG8d5PM8`.

We compare several different models. *PFNN One-hot* (Holden et al., 2017b) uses a one-hot vector with the PFNN to model styles. *PFNN Resad* is our residual adaptation approach seen in Figure 3.5. We also combine our FiLM style modulation method with different animations synthesis networks, specifically the PFNN (*PFNN FiLM*), the MANN of Zhang et al. (2018a) (*MANN FiLM*), and the LPN (*LPN FiLM* - our system in Figure 3.12). We additionally combine the alternative style modulation approaches with our LPN network for comparison, creating *LPN One-hot* and *LPN Resad*.

3.10.1 Reducing Failure Cases

We first compare the qualitative results of different systems, demonstrating scenarios that are challenging for different approaches. Each of the comparisons can be seen in the accompanying video at full and half speed.

3.10.1.1 Motions Without a Global Phase

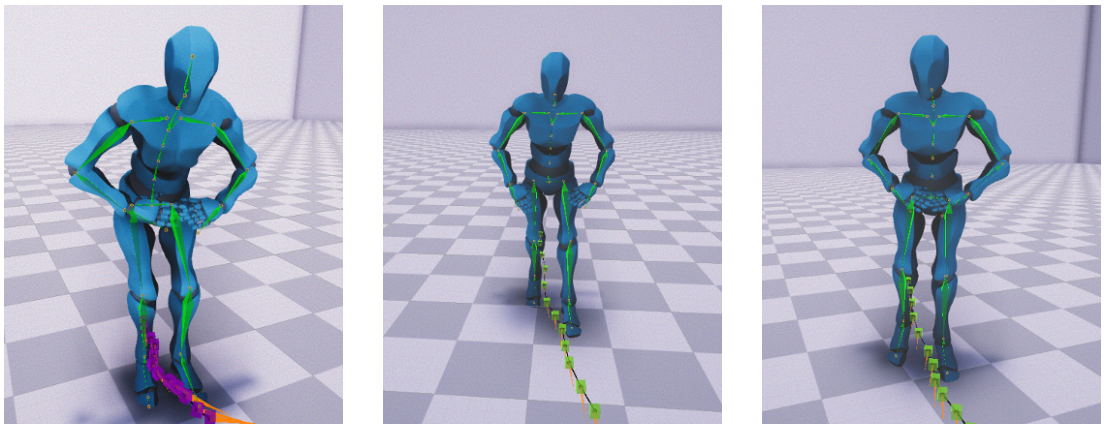


Figure 3.13: No Global Phase. Style: *Pendulum Hands*. Left to Right: Data, PFNN FiLM, LPN FiLM. The hands do not swing as they should using the PFNN.

Styles without a single global phase cannot easily be modelled with the PFNN as the inductive bias creates phase specific behaviour. Figure 3.13 demonstrates an example for a style where the hands move with a different phase cycle than the feet. Using the same style modulation technique (FiLM), with the PFNN the model does not capture all the style details with the hands performing a 'stuck' or 'average' motion whereas the LPN is able to well model the local movements in the style.

Contact-Free Modelling. We can also use our contact-free local phase extraction to generate local phases for foot bones when contacts are not available. Figure 3.14 shows our LPN FiLM model performing a hopping motion. This flexibility regarding available contact information, as well as the difficulty the PFNN has modelling styles with no coherent global phase, emphasises the importance of using a more flexible architecture for stylised locomotion synthesis (in our case the LPN).

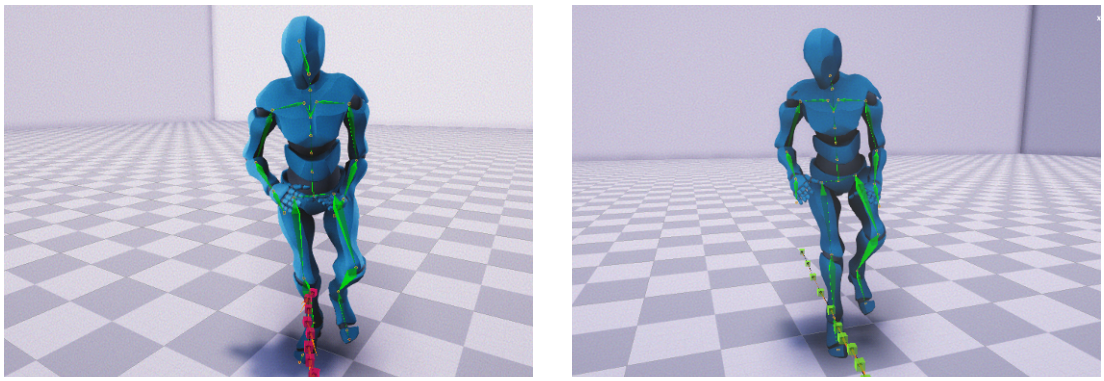


Figure 3.14: Contact-Free Modelling. Style: *Left Hop*. Left: Data. Right: LPN FiLM.

3.10.1.2 Dealing With Large Numbers of Styles

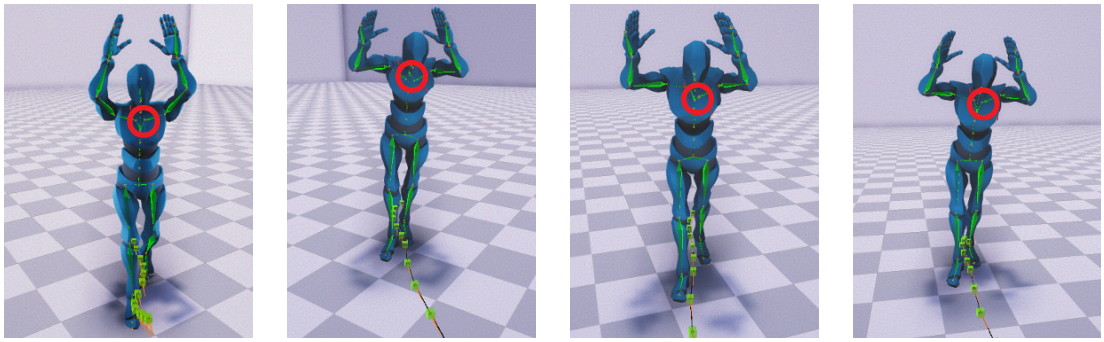


Figure 3.15: Capacity Challenges. Style: *Arms Above Head*. Left to Right: Data, LPN One-hot, LPN Resad, LPN Film. Lower capacity style modulation approaches can create artifacts as highlighted in the clavicle bones.

The large number of styles in the 100STYLE dataset allows us to test the limits of model capacity as the number of styles increases. In particular, we find that when trained on 95 styles both one-hot and residual adaptation methods display some unnatural artifacts for certain styles (Figure 3.15). Note that by capacity we refer to the ability of the style specific parameters to modulate the motion synthesis network. That is, both the LPN one-hot and LPN Resad models add *one* shift vector to *one* layer of hidden units wheres LPN FiLM *both* scales and shifts *all* hidden units. For LPN one-hot specifically, this problem can also be seen in the fact that styles are sometimes mixed up or incorrectly modelled (see video).

Interpolation Differences. We also evaluate the effect of different methods on the ability to generate interpolations or transitions between styles. Figure 3.16 shows that for LPN one-hot interpolation is not smooth, instead the style only changes when the interpolation is completed. In contrast LPN FiLM learns a continuous interpretable style space where linear interpolation create smooth transitions. The presence of artifacts when using different style modulation approaches as well as better interpolation behaviour demonstrate the efficacy of modelling style with FiLM parameters.

3.10.1.3 No Explicit Phase Features

Our final comparison is to the MANN of Zhang et al. (2018a) where a gated experts system is used but the gating is based on end effector bone velocities. As shown in Figure 3.17, like with the PFNN, this method can struggle when the upper limbs move with a clear phase cycle that is different from the lower limbs. We attribute this to the

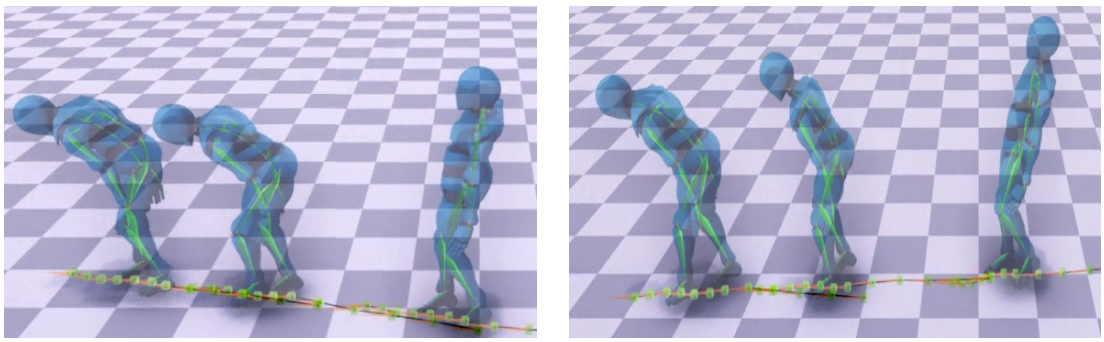


Figure 3.16: Interpolation. Style: *Bent Forward* to *Lean Back*. Left: LPN One-hot. Right: LPN FiLM. The one-hot method does not create a smoothly changing transition.

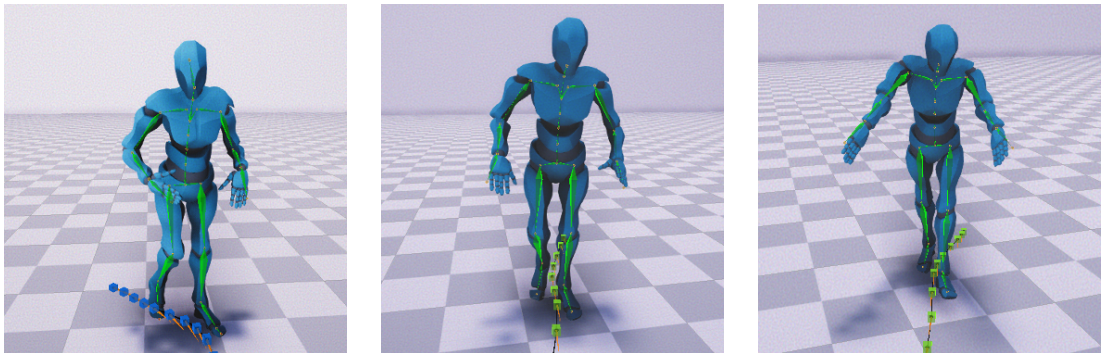


Figure 3.17: No Phase Features. Style: *Swimming*. Left to Right: Data, MANN FiLM, LPN FiLM. The arms do not move in a cycle when using the MANN.

fact that the MANN does not extract explicit phase features, instead having to learn the different cycles implicitly from the velocities. The differences between MANN FiLM and LPN FiLM demonstrate the efficacy of explicitly extracting local phase features.

3.10.2 Numerical Comparisons

Whilst qualitative performance is by far the most important factor for animation systems we are able to gain some insights from quantitative analysis.

3.10.2.1 Computational Costs

Our first numerical comparison is in terms of computational cost. Table 3.4 shows parameter counts for the different models we compare as well as the runtime performance. We first note that, whilst all systems can run comfortably at 60 fps, the PFNN is substantially faster than the gated expert systems. This is not surprising as the PFNN has fewer parameters to blend (4 control points vs. 8 experts). What's more Holden

Table 3.4: Storage and Runtime Comparisons. *A.S.N.* is the parameter count for the animation synthesis network, *S.M.N.* is the parameter count for the style modulation network, and *P.S.R.* is the parameters required per style at runtime. Runtime (ms) shows the average time (with one standard deviation) to predict a single frame.

Model	A.S.N.	S.M.N.	P.S.R.	Runtime (ms)
PFNN One-hot	2,436,380	194,560	2048	0.78 ± 0.01
LPN One-hot	4,935,928	389,120	4096	4.11 ± 0.02
PFNN Resad	2,436,380	2,978,440	31,352	0.92 ± 0.01
LPN Resad	4,935,928	24,952,320	262,656	4.78 ± 0.06
PFNN FiLM	2,436,380	35,668,530	2048	1.37 ± 0.02
MANN FiLM	4,870,392	35,668,530	2048	4.48 ± 0.04
LPN FiLM	4,935,928	35,668,530	2048	4.53 ± 0.04

et al. (2017b) provide a runtime optimisation for the PFNN (trading off storage for speed by saving 50 weight matrices that discretise the phase) that cannot easily be extend for systems with multiple phases. So, whilst there are serious qualitative challenges with the PFNN, is performance is critical and we only need to model a number of styles this architecture may be preferable.

Probably the most interesting column in Table 3.4 is the P.S.R. column. This shows the number of additional parameters that need to be saved in order to model one new style. We note first that our residual adaptation approach loses out. All residual adapter parameters must be saved because the input to the residual adapter (and hence the output style modulation) will change during runtime. On the other hand we see that both FiLM and one-hot methods are significantly cheaper. What is important to realise here is that FiLM is able to achieve much higher quality results with the same number of parameters. This demonstrates that how the parameters are learned (i.e. learning an interpolatable style space) and how the parameters are used to modulate the network (i.e. scale and shift all layers) are more important design choices than simply how many parameters to use.

3.10.2.2 Test Error

We also use our held out test set to compare the test error on unseen frames of motion, visualised in Figure 3.18. Our main analysis of these results is that, like Martinez et al. (2017), we find the quantitative test error does not necessarily align with the qualitative performance. As Martinez et al. (2017) showed, a low mean squared error

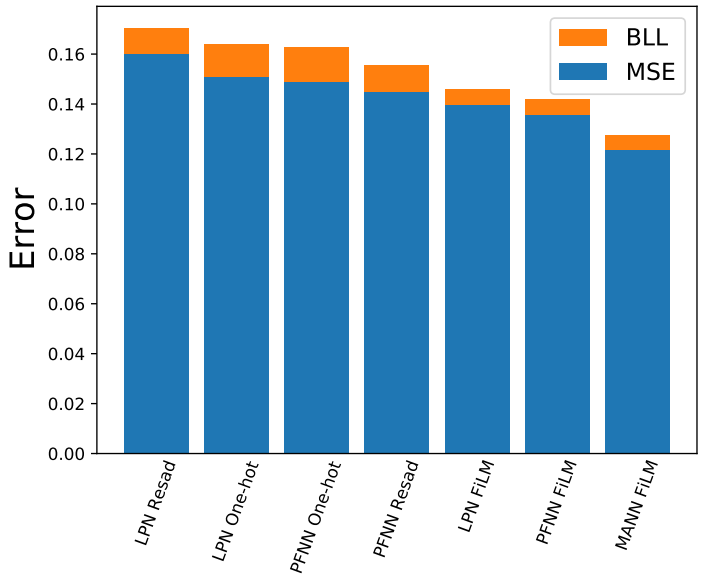


Figure 3.18: Test Error. We compare the error of different models on a held out test set. Each bar is divided into the blue mean squared error (MSE) and the orange bone length loss (BLL).

can be achieved by outputting an average pose (as opposed to say capturing all motion details but with poses being out of sync with the ground truth predictions).

Whilst we must not over-interpret Figure 3.18, there are still some interesting patterns to highlight. Firstly we note that the systems that use FiLM to model style achieve the lowest errors, this aligns with our observation that FiLM has much higher capacity for changing the hidden representations of an animation synthesis network. Also, the difference between PFNN Resad and LPN Resad suggests that the CP decomposition we perform when using the residual adapter with the PFNN provides more benefits than just a controllable number of parameters[6]. As seen from the qualitative results, the LPN really only improves over the MANN for the few styles when upper and lower body move with different phase cycles. Whilst the MANN may have better quantitative performance on periodic motions (the majority of the data), the LPN and MANN are qualitatively similar for such motions. Together, these points underline that the inductive biases used to select *how* to model style and content are more important than raw network capacity and that qualitative evaluation is key for model selection (at least when using the datasets available).

[6]We do not perform such a decomposition when using the LPN as we do not have a single phase variable to decompose over.

3.11 Discussions

In this chapter we have investigated the modelling of human locomotion style and explored methods for the low data regime alongside designing improved architectures when sufficient data is available. Regardless of the architecture and approach there are some areas that remain open challenges or directions for potential improvements.

3.11.1 What is Style Anyway?

Throughout this chapter we have defined style using the data generating assumptions of Figure 3.1a. That is, making a content invariance assumption to define content as 'the things that are the same over the styles in the data' and style as 'everything else that differs'. Whilst this is a principled approach that allows us to design systems that make use of large datasets, motion style is, at its core, a creative concept. For example our dataset contains styles such as *Left Hop* and *Spin Clockwise*, and one can reasonably ask if these are styles of motion or if they are actions that can be performed in any style? We could then expand this argument further, thinking about the less clear *Chicken* style similarly. If this is a young person doing a chicken impression it may look very different to the impression performed by an older person, in fact each person would probably do it differently. How well can this type of style be predicted without specific stylised data for each individual? All that is to emphasise that 'style' is a subjective and creative concept and we hope that our data and model will be useful for improving the design of creative systems.

3.11.2 Future Improvements

In designing our local phase approach we saw how the extraction of local phases is fundamentally a question of *source function design*. We designed a source function that made sense for our stylised data and was easy to compute. However, for different datasets, or different challenges we may wish to extract local phases from new source functions. As an example, our source function will not handle randomness well as it is designed for joints moving with a deterministic phase cycle. Extracting information that can be used to well model different types of movements through designing new source functions may allow us to achieve similar performance improvements in other animation tasks.

Additionally, all of our data and systems herein have been designed for kinematic

(non physics based) systems. This allows for artistic and creative freedom as well as avoiding costly physics simulation. However, our models are quite capable of producing physically implausible motions, for example when interpolating between styles the arms may clip through the torso. It seems highly likely that any general model of human motion will need to make use of some type of physics understanding as this is a fundamental constraint on realistic motion capture data. Creating systems that learn world models (Ha and Schmidhuber, 2018) that take physics into account (Fussell et al., 2021); that combine kinematic approaches with physical simulation to prevent self-collisions and enforce physically plausible motions; or that use some other method for combining physics-based and kinematic approaches, will be necessary to create characters capable of modelling the full general range of constrained human motions.

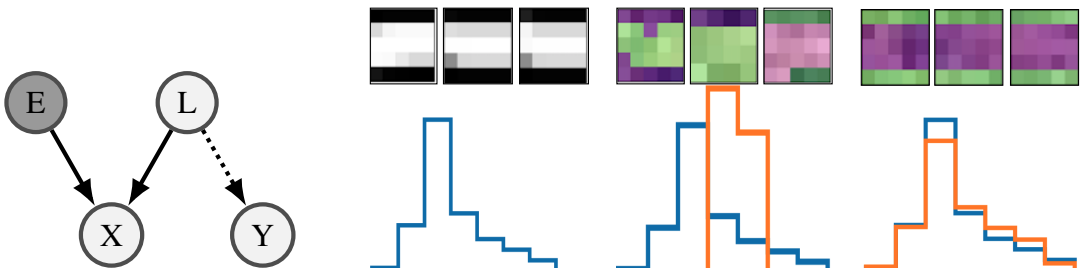
3.12 Summary

We began this chapter by distinguishing style modelling and style transfer methods that make use of stylised humanoid locomotion data. Initially, motivated by a lack of available data we designed a system that modelled style using residual adapters whose capacity could be varied depending on style complexity and data available. We took the lessons learned from analysing this approach, along with modern developments (Perez et al., 2018; Starke et al., 2020), to improve general style modelling systems. We designed a new dataset for style modelling and used this to evaluate an improved style modelling system created via feature engineering, animation synthesis network design and the learning of compact but powerful style representations.

By exploring neural animation synthesis, we have been looking at practical applications of systems that can handle changing domains. In the remainder of this thesis we turn to more general adaptation approaches. Rather than being designed for specific applications, these approaches attempt to understand how domain changes manifest in neural networks and how we might go about resolving some of the challenges caused by changing domains, particularly under the restriction of limited data.

Chapter 4

Feature Restoration for Source-Free Domain Adaptation



(a) Domain Shift. We may not have access to label Y.

(b) Feature Restoration Intuition. Aligning activation distributions restores semantic meaning and hence classifier performance.

Figure 4.1: The feature restoration approach to resolving domain shift. (a) Data generating assumptions for source-free domain adaptation. (b) Aligning unit activation distributions to restore semantic meaning to features.

So far we have been looking at the importance of developing techniques for rapid and few-shot learning for creative applications. However, few-shot learning remains a general challenge for modern machine learning systems (Wang et al., 2020). In the remainder of this thesis we will explore the more general domain shift scenario with controlled experimentation on specific datasets.

In particular, whereas in Chapter 3 we considered mainly transfer learning approaches for adaptation in generative models, in this chapter we return to the discriminative task. That is, we wish to continue to *classify* data well after a change of domain (Fig-

ure 4.1a). It is also important to restate that very often it may be that data in a new domain is unlabelled, for example perhaps a model has been trained on expensively annotated data and then deployed in the real world whereupon some change in conditions (E in Figure 4.1a) causes a degradation in performance. In such a situation relabelling new data may be prohibitively costly or time consuming but we still wish to be able to adapt our model to the new domain (without labels, dashed arrow in Figure 4.1a). This is the task of unsupervised domain adaptation (UDA) which has been well studied.

A further important restriction is that when a model is deployed we may lose access to the data on which the model was trained (source data). This may occur for example because of privacy regulations or storage constraints on mobile devices. Once again, we still wish to be able to adapt a model to changing domains in this situation. This is the task of source-free domain adaptation (SFDA), (Liang et al., 2020).

This chapter both presents a model-based method for unsupervised source-free domain adaptation and discusses situations in which we can expect it to work well (measurement shifts) particularly within the context of other confidence maximising SFDA methods (Lee et al., 2013; Liang et al., 2020).

4.1 Related Work

As discussed in Section 2.1.2, given labelled data in a new domain we can simply finetune a source domain model (Girshick et al., 2014). However, in the more challenging scenario where such labelled data we must attempt unsupervised domain adaptation or even source-free domain adaptation if the source data is unavailable.

4.1.1 Unsupervised Domain Adaptation

Ben-David et al.'s theoretical work (Ben-David et al., 2007, 2010a,b) created bounds on performance in a target domain using the source domain error, the variational distance between source and target distributions and a third term which is assumed to be small. Ben-David et al. additionally showed that learning features from which a low capacity classifier cannot tell the difference between source and target domains can reduce the variational distance term.

Inspired by these results, many model-free UDA methods have been proposed which

attempt to match source and target feature distributions (Ganin and Lempitsky, 2015; Long et al., 2015; Ganin et al., 2016; Tzeng et al., 2017; Long et al., 2018; Shu et al., 2018). Since these methods do not parameterise the feature distributions, they usually require access to the source and target data simultaneously in order to align them. That is, the source feature distribution cannot be stored in advance for later use if the source data becomes unavailable.

Whilst model-based methods do exist for UDA, they are often not designed with the source-free setting as a possibility. In particular, they may still use the source data during adaptation on the target domain but not for aligning the distributions (Sun and Saenko, 2016), or they do not design their parameterisations to be cheap to store offline (Zellinger et al., 2017).

4.1.2 Source-Free Domain Adaptation

In the source-free setting, which we focus on in this chapter, there are two basic approaches. The first approach is to ask 'what does the output of a well adapted classifier look like?' The second approach, and the one we take, asks 'what does the input to a classifier look like in a well adapted model?'

When considering the output of a well adapted classifier, since a good classifier should predict a single class with high probability (effectively a one-hot vector) many SFDA methods minimise the entropy of target domain predictions. This is equivalent to maximising prediction confidence and encourages one-hot outputs. One successful entropy minimising technique is SHOT (Liang et al., 2020) which as well as minimising entropy for single predictions additionally maximises the entropy of average predictions which prevents the adapted classifier from always predicting a single class. This combination of terms is a repurposing of the information maximisation (IM) loss of Krause et al. (2010). Liang et al. (2020) also include a pseudo-labelling loss which is itself entropy minimising (Lee et al., 2013). Whilst effective for improving classification accuracy, entropy minimising losses are classification specific and, by making predictions more confident, destroy model calibration.

Many more recent works train generative models to capture the source data distribution so it can be utilised to help adapt in the target domain (Li et al., 2020; Morerio et al., 2020; Kundu et al., 2020; Stan and Rostami, 2021). However, apart from being expensive, these methods still rely on entropy minimisation (Li et al., 2020; Kundu et al.,

2020) and pseudo-labelling (Morerio et al., 2020; Stan and Rostami, 2021). TENT (Wang et al., 2021) also minimises entropy but only trains batch normalisation parameters aiming to rapidly adapt for online learning.

When we instead consider the input to the classifier in a well adapted model, we ask how might we be able to align source and target feature distributions without access to the source data? Whilst we are not aware of this question being asked explicitly by other authors, there are a number of works that focus on aligning batch normalisation (BN) statistics which implicitly align source and target marginal feature distributions. Adaptive BN (AdaBN, Li et al. (2017a)) is a parameter free method that replaces the source domain batch normalisation statistics with statistics calculated on the target domain. Ishii and Sugiyama (2021) specifically retrain feature extractor parameters in order that the batch normalisation statistics on the target domain after adaptation are the same as the batch normalisation statistics were on the source domain before adaptation. Hou and Zheng (2020) stylise target domain images to look more like target domain images by matching batch normalisation statistics, implicitly linking domain adaptation and style transfer approaches (as we do explicitly in Section 2.2.1). The method we develop for SFDA also aligns marginal distributions but, by not being restricted to the first two moments, is much more flexible in its parameterisation.

4.1.3 Continual Learning

A natural and important extension of domain adaptation is continual, or lifelong, learning (Parisi et al., 2019). Continual learning systems aim to effectively reuse learned knowledge to adapt a model on a new task without significantly reducing performance on previously learned tasks (often without access to previous task data). Whilst this ability to sequentially learn new tasks is an important aspect of intelligence in biological systems (Power and Schlaggar, 2017), neural networks are vulnerable to catastrophic forgetting, where the learning of a new task destroys performance on previously learned tasks. Very generally, continual learning methods try to avoid significant changes in learned parameters in order to reduce the amount of forgetting. Different methods use different techniques to maintain learned information and operate at different levels of granularity, from regularisation on the whole network by matching outputs between tasks through to estimating the importance of individual parameters (Li and Hoiem, 2017; Kirkpatrick et al., 2017; Lee et al., 2017; Zenke et al., 2017).

Whilst continual learning is a useful property for deployed intelligent systems to have, the aim of the work in this chapter is to achieve the best performance on a new domain without concern for maintaining performance on the source domain.

4.2 Feature Restoration

In source-free domain adaptation we have some model (neural network), f_s, trained to classify data in the source domain. This network can be viewed as a feature extractor, g_s, and a classifier, h, so $f_s = h \circ g_s$. Whereas methods like SHOT (Liang et al., 2020) are motivated by asking what the output of the classifier h should look like, we instead consider the input to the classifier. In particular, the feature extractor's (g_s) learned feature space $Z = g_s(X_s)$.

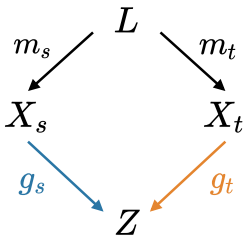
If we relate this to our data generating assumption in Figure 4.1a (using the m_E notation introduced in Section 2.1) where $E = s$ and $g_s(X_s) = g_s(m_s(L))$, in order to classify well Z must capture information in L about Y despite the nuisance variables from E (left path of Figure 4.2a). When the domain changes we still require that Z captures information in L about Y but, when the data shifts to the target domain $E = t$, the learned function g_s will often not be able to well extract this information due to a change in the nuisance variables[1]. Therefore, we learn a new feature extractor in the target domain, g_t.

Now consider the question, what does the output of the feature extractor (or the input to the classifier) look like in a 'good' model for a new target domain? In an idealised situation, where we have datapoints in source and target domains with identical L and different E, we would like that $g_s(m_s(L)) = g_t(m_t(L))$ (right path of Figure 4.2a). In realistic situations we cannot pair examples with identical L across different domains, so instead we aim to match the outputs of $g_s(m_s(\cdot))$ and $g_t(m_t(\cdot))$ in distribution[2]. That is, we relearn a feature extractor in the target domain, g_t, in order that $p(g_t(X_t)) \approx p(g_s(X_s))$.

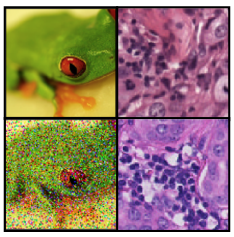
We call the process of relearning g_t in the target domain, with the aim to extract the

[1] By 'well extract' we mean extract latent Z in a way that allows the classifier, h_s, to achieve good performance.

[2] If our assumption in Figure 4.1a is reasonable, that E is all that changes across domains, then we expect L to have a similar distribution across domains given enough data. That is, matching distributions is a reasonable approach.



(a) Feature restoration



(b) Measurement shifts

(c) *Non* measurement shifts

Figure 4.2: Feature restoration diagram and measurement shift examples. *(b,c):* top=source, bottom=target. *(b):* CIFAR-10-C 'frog' & CAMELYON17 'tumor'. *(c):* OFFICE-31 'desk chair' & VISDA-C 'person'.

same Z as in the source domain, feature restoration. The idea is that restoring features in this way should be sufficient to restore classifier performance. In synthetic examples where we can generate data with identical L across domains we can quantitatively measure the degree of restoration with $|g_s(m_s(L)) - g_t(m_t(L))|$. If we consider each neuron or unit separately, this intuitively aligns with ideas of neuron invariance (Quiroga et al., 2005), that is, *if a neuron responds in the same way to a feature with the same semantic meaning this is sufficient to restore classifier performance*. This intuitive idea is seen in Figure 4.1b for a single neuron in the first layer of a convolutional network which responds to a black and white horizontal edge (left column, showing the maximum activating patches and the activation distribution for this neuron). Upon some change in domain (here black and white digits to green and purple digits) the neuron responds in a different way (middle column, showing the new activation distribution in orange and the original activation distribution in blue along with the max activating patches which have no clear semantic meaning). By matching the activation distributions so the neuron responds in the same way, semantic meaning is restored (right column, showing the activation distribution after feature restoration in orange and the original activation distribution in blue along with the max activating patches which now share the same 'horizontal edge' semantic meaning).

4.2.1 When Does Feature Restoration Make Sense?

As discussed, our approach is based on the idea that neurons should respond in the same way to features with the same semantic meaning across domains. In practice this approach will not work for all domain adaptation tasks where restoring performance may require learning features with *new* semantic meaning. This can happen because

of concept shift (Moreno-Torres et al., 2012, Sec. 4.3), where the features that define a class can change across domains (a shift in L when E changes) or because a source neural network can learn to exploit spurious correlations, or shortcuts, in the data (Arjovsky et al., 2019; Geirhos et al., 2020) that do not work in the target domain (i.e. it is possible to learn Y from X without relying solely on L).

However, for many domain adaptation tasks restoring the source features in the target domain will be a reasonable thing to do. We call the shifts where it is possible to restore features to restore performance *measurement shifts* as they are often caused by a change in measurement system (e.g. different cameras, lighting etc.). Figure 4.2b shows measurement shift examples, note the right hand column showing a change in colour and contrast caused by a change in machinery used to capture these cell images (Bandi et al., 2018). By making the assumption that we are working with measurement shifts we will see that we can gain benefits over other methods for SFDA in improved calibration and few-shot performance.

To further examine the problems of non measurement shifts examine Figure 4.2 which shows domain adaptation examples that are not measurement shifts (Saenko et al., 2010; Peng et al., 2018a). The right column of this figure shows examples of the person class in synthetic and real domain for the VISDA-C dataset, as a domain adaptation task this type of set up provides many difficulties (Ringwald and Stiefelhagen, 2021). For example, as the synthetic domain contains only two grey person models, rendered from different viewpoints with different lighting, in the real-to-synthetic task a source network can learn to exploit image features in the real domain that do not exist in the synthetic domain e.g. complex textures (Geirhos et al., 2019b). These difficulties cause existing methods to rely on models that are initially pretrained using a different large dataset (commonly IMAGENET (Russakovsky et al., 2015)) in order to extract more general image features first, in effect implicitly changing the domain adaptation task from SYNTHETIC $\rightarrow$ REAL to IMAGENET $\rightarrow$ SYNTHETIC $\rightarrow$ REAL. Some works even lower the learning rate of early network layers after IMAGENET pretraining in order to better maintain the low-level general features (Liang et al., 2020). Table 4.1, adapted from Eastwood et al. (2022), shows the reliance of existing methods on this IMAGENET pretraining step in order to achieve good performance on this type of domain adaptation task.

Due to these issues, we focus our efforts on measurement shifts, both real and synthetic, and evaluate our feature restoration approach in this setting.

Table 4.1: VISDA-C Results (ResNet-101). Without IMAGENET pretraining adaptation on the SYNTHETIC → REAL task achieves extremely low performance.

Model	ImageNet pretrain	Avg. Acc.
No corruption	✗	99.8
Source-only	✗	10.4
AdaBN (Li et al., 2017a)	✗	15.9
SHOT (Liang et al., 2020)	✗	17.1
No corruption	✓	99.6
Source-only	✓	47.0
AdaBN (Li et al., 2017a)	✓	65.2
SHOT (Liang et al., 2020)	✓	82.9

4.3 EMNIST-DA

To examine adaptation on measurement shifts we use several existing datasets explained and visualised in Appendix B.1. However, we find it useful to create an additional original dataset designed to highlight the benefits of feature restoration. This dataset is designed to be very simple to understand and interpret while providing a range of adaptation difficulties. To do this we base our dataset on the digit and character extended MNIST dataset, EMNIST, as MNIST based datasets tend to be too easy to resolve (see Section 4.5, Table 4.4). To get a range of adaptation difficulties we choose 13 conceptually simple shifts designed to give a range of initial accuracies on a model trained on the original (black and white) EMNIST data. We call this dataset EMNIST-DA, a sample from each shift is shown in Figure 4.3 with the 'identity' shift being the original data. To use this dataset we train models on the train split of EMNIST and adapt on shifts generated from data from the test split of EMNIST. This is done so we do not have exactly the same samples (with identical L) to adapt on, which avoids relying too heavily on such a requirement.

4.4 Restoring Features

As outlined, our aim is to relearn the feature extractor in the target domain such that $p(g_t(X_t)) \approx p(g_s(X_s))$. We break this process into two parts, steps to take during model development and the adaptation process after model deployment (SFDA). We will first outline the formal mathematical set up to allow us to rigorously explain the pipeline.

Figure 4.3: EMNIST-DA shifts.

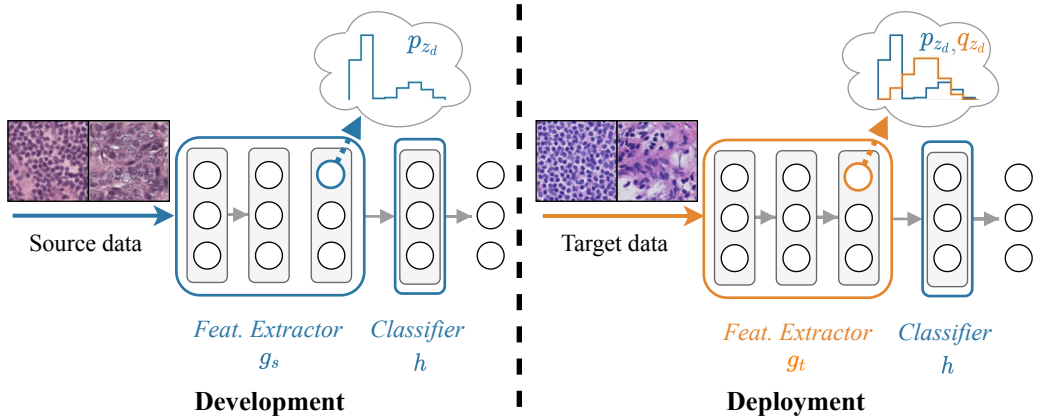


Figure 4.4: Feature Restoration Overview. *Left:* At development time, approximate marginal feature distributions, $\{p_{z_d}\}_{i=1}^{D}$, are saved using the source data. *Right:* At deployment time, the approximations of the marginal feature distributions under the target data, $\{q_{z_d}\}_{i=1}^{D}$, are aligned with those under the source data.

During development a source model, $f_s : \mathcal{X}_s \to \mathcal{Y}_s$, is trained on source data $\mathcal{D}_s = \{(\mathbf{x}_s^{(i)}, y_s^{(i)})\}_{i=1}^{n_s}$ with $\mathbf{x}_s^{(i)} \in \mathcal{X}_s$ and $y_s^{(i)} \in \mathcal{Y}_s$ and n_s the number of labelled source domain samples. After training, this model is deployed and receives some unlabelled target domain data $\mathcal{D}_t = \{\mathbf{x}_t^{(i)}\}_{i=1}^{n_t}$, with $\mathbf{x}_t^{(i)} \in \mathcal{X}_t$ with n_t samples. To adapt on this new data we learn a new target model $f_t : \mathcal{X}_t \to \mathcal{Y}_t$ which aims to predict the target domain labels $\{y_t^{(i)}\}_{i=1}^{n_t}$, where $y_t^{(i)} \in \mathcal{Y}_t$. As we are interested in the source-free setting, the source data, D_s, is not available during this adaptation process.

4.4.1 Development

During development our model is designed *with the assumption that a domain shift will occur after deployment*. Previous works have also made similar assumptions, us-

ing training techniques[3] to encourage clustering of representations (Liang et al., 2020), storing mean per-class feature representations (Chidlovskii et al., 2016) or generating artificial negative datasets (Kundu et al., 2020). For us, this assumption means storing lightweight approximations of the marginal feature distributions. We denote these statistics of the source model as $\mathcal{S}_s$.

So, our source model f_s is trained to classify $\mathcal{D}_s$ using some loss function, $\mathcal{L}_{src}$, in our case the standard multi-class cross-entropy loss. Unlike other methods, storing marginal feature distributions does not require any change to a standard training pipeline, instead occurring with a single additional forwards pass after training. This means the method can easily be used on off the shelf pretrained models without expensive or time consuming retraining. We can again think of this source model in terms of a feature extractor $g_s : \mathcal{X}_s \to \mathbb{R}^D$ and a classifier $h : \mathbb{R}^D \to \mathcal{Y}_s$, with D the dimensionality of the hidden feature space. If we have paired source and target measurement shift data, then our aim is to learn a new feature extractor, $g_t : \mathcal{X}_t \to \mathbb{R}^D$, such that $\mathbf{z}_s^{(i)} = g_s(\mathbf{x}_s^{(i)}) = \mathbf{z}_t^{(i)} = g_t(\mathbf{x}_t^{(i)})$, where $\mathbf{z}_s^{(i)}$ is the hidden feature vector for sample i from the source domain and $\mathbf{z}_t^{(i)}$ the same in the target domain. The classifier h therefore remains unchanged in order that the predicted labels, $\hat{y}_s^{(i)} = f_s(\mathbf{x}_s^{(i)}) = h(g_s(\mathbf{x}_s^{(i)}))$ and $\hat{y}_t^{(i)} = f_t(\mathbf{x}_t^{(i)}) = h(g_t(\mathbf{x}_t^{(i)}))$, for data from source and target domains are equal.

In order to achieve our aim that $p(g_t(X_t)) \approx p(g_s(X_s))$, during development we store some approximation to the feature distribution, $p(g_s(X_s))$, so that when data X_s in unavailable we can still align distributions. When D is large, dealing with the joint distribution over features can be prohibitively expensive[4], therefore, we choose to approximate (and store representations of) the marginal feature distributions. Noting that this amounts to storing approximations of the $1D$ activation distributions of each of the D units in the final layer of g_s, we see more clearly the links to invariance discussed in Section 4.2. That is, we will align the activation distributions of individual unit's in order to encourage them to respond in the same way to target data as they did to source data. Figure 4.4 shows an overview of our approach, with the development figure on the left showing the storing of approximate marginal distributions on the source data.

To capture the marginal distributions we use softly binned (Dougherty et al., 1995)

[3]Label smoothing (Szegedy et al., 2016; Müller et al., 2019) and learning rate design.

[4]If we assume feature distributions are jointly normal estimating the covariance matrix is $O(ND^2)$ per update, where N is the batch size. If we try a more general histogram binning approximation with B bins per dimension, memory complexity is $O(B^D)$.

Table 4.2: Storage Comparisons. The storage (MB) required is shown for different datasets, the parameters of the corresponding source model architectures, and the softly binned histogram parameterisations (bin-counts) of marginal feature distributions.

	MNIST	**CIFAR-100**	**VisDA-C**	**ImageNet**
Architecture	LeNet	ResNet-18	ResNet-101	ResNet-101
Source dataset	33	150	7885	138000
Source model	0.9	49	173	173
Source bin-counts	0.004	0.02	0.5	0.5

histograms. By using histograms we are able to flexibly capture the $1D$ distributions which may be skewed or multi-modal (see Section 4.5.1). The soft binning (rather than hard binning) makes the distribution parameterisation differentiable which allows us to backpropagate through this operation. Additionally, this representation is very lightweight and does not scale with dataset size as, with a fixed number of bins B and feature space dimensionality D, storage is constant $O(BD)$. This storage is inconsequential when compared to the storage required for the source model itself as shown in Table 4.2.

4.4.1.1 Calculating Softly Binned Histograms

To calculate the softly binned histograms, to approximately parameterise the marginal feature distributions, we use the soft binning function outlined by Yang et al. (2018, Section 3.1). We denote the marginal distribution of the d^{th} feature, z_d, in the feature space created by g_s, p_{z_d}. This gives us the D distributions $\{p_{z_d}\}_{i=1}^{D}$. For a single feature we approximately parameterise p_{z_d} with B normalised bin counts $\pi^s_{z_d} = [\pi^s_{z_d,1}, \ldots, \pi^s_{z_d,B}]$. Each normalised bin count, $\pi^s_{z_d,b}$, is the probability that a single sample $z_d^{(i)}$ will be in bin b under the source data, and thus $\sum_{b=1}^{B} \pi^s_{z_d,b} = 1$.

To calculate these normalised bin counts π_{z_d} (dropping the source/target domain notation for clarity), the first step is to normalise the range of values z_d can take on to be in $[0,1]$. This ensures overlapping supports for variables with different ranges, easier comparison between distributions for different features, and that the binning softness is comparable across variables (the degree to which mass is distributed into neighbouring bins). To normalise the range of z_d we need to calculate the maximum and minimum value of z_d over all samples, $\{z_d^{(i)}\}_{i=1}^{n}$. These are denoted $z_d^{max} = max_i\, z_d^{(i)}$ and $z_d^{min} = min_i\, z_d^{(i)}$ respectively.

The next step is to create B bins using $B-1$ uniformly-spaced and monotonically-increasing cut points, $\mathbf{c}$, over $[0,1]$: $\mathbf{c} = [c_1, c_2, \ldots, c_{B-1}] = \frac{1}{B-2}[0,1,2,\ldots,B-3,B-2]$. Using the cut points and the maximum and minimum values we can calculate the soft count for each bin for a sample $z_d^{(i)}$ using the vector soft binning function $\mathbf{u}$. This creates a B-dimensional vector of soft counts $\mathbf{u}(z_d^{(i)})$,

$$\mathbf{u}(z_d^{(i)}; z_d^{min}, z_d^{max}) = \sigma((\mathbf{w}\left(\frac{z_d^{(i)} - z_d^{min}}{z_d^{max} - z_d^{min}}\right) + \mathbf{w}_0)/\tau), \tag{4.1}$$

where $\mathbf{w} = [1,2,\ldots,B]$, $\mathbf{w}_0 = [0, -c_1, -c_1 - c_2, \ldots, -\sum_{j=1}^{B-1} c_j]$, $\tau > 0$ is a temperature factor and σ is the softmax function. The output $\mathbf{u}(z_d^{(i)})_b$ is the mass assigned to bin b, and $\sum_{b=1}^{B} \mathbf{u}(z_d^{(i)})_b = 1$ for single sample $z_d^{(i)}$. Both $\mathbf{w}$ and $\mathbf{w}_0$ are predefined constant vectors based on the number of bins as in Yang et al. (2018). The temperature parameter τ changes the behaviour of $\mathbf{u}$, with $\mathbf{u}(z_d^{(i)})$ tending to a one-hot vector as $\tau \to 0$. We set $\tau = 0.01$ in our experiments. If we get new values outside the range of z_d^{min} and z_d^{max} they will be handled sensibly by the soft binning function falling into the leftmost or rightmost bins respectively as $\tau \to 0$. Finally, we get the total bin counts over all samples, by summing $\mathbf{u}(z_d{}^{(i)})$ over i, before dividing by the total number of samples, n, to get normalised bin counts π_{z_d}, i.e., $\pi_{z_d} = \sum_{i=1}^{n} \frac{\mathbf{u}(z_d{}^{(i)}; z_d^{min}, z_d^{max})}{n}$.

As discussed, the reasoning for using this approach is so that the parameterisation is both flexible and differentiable, but, as seen in the left 'cloud' of Figure 4.4 which depicts an approximate marginal distribution created with this process, for intuitive understanding and reasoning we can think of these distributions as simple histograms.

To summarise, during development we can use the source data with this soft binning functionality to calculate normalised bin counts for the d^{th} feature as

$$\pi_{z_d}^s = \sum_{i=1}^{n_s} \frac{\mathbf{u}(z_d^{(i)})}{n_s} = \sum_{i=1}^{n_s} \frac{\mathbf{u}(g_s(\mathbf{x}_s^{(i)})_d \,; z_d^{min}.z_d^{max})}{n_s}. \tag{4.2}$$

We can calculate these counts for all D features to get $\pi_{\mathbf{z}}^s = [\pi_{z_1}^s, \pi_{z_2}^s, \ldots, \pi_{z_D}^s]$. We additionally find it useful to store parameterisations of the marginal logit distributions, $\pi_{\mathbf{a}}^s$, where $\mathbf{a}^{(i)}$ is the output of the classifier, h, pre-softmax for sample i, and $\pi_{\mathbf{a}}^s$ is defined analogously to $\pi_{\mathbf{z}}^s$. The idea behind additionally aligning logit distributions is to further constrain the ways in which the marginal feature distributions can be aligned.

The normalised bin counts for marginal feature and logit distributions along with the maximum and minimum values features and logits take on in the source data form the

lightweight statistics that can be packaged with the model for use during deployment. $\mathcal{S}_s = \{\pi^s_{\mathbf{z}}, \pi^s_{\mathbf{a}}, \mathbf{z}^{min}, \mathbf{z}^{max}, \mathbf{a}^{min}, \mathbf{a}^{max}\}$, where $\mathbf{z}^{min} = [z_1^{min}, z_2^{min}, \ldots, z_D^{min}]$ and $\mathbf{z}^{max}$, $\mathbf{a}^{min}$, and $\mathbf{a}^{max}$ are defined analogously..

4.4.2 During Deployment

After training the model, f_s, is deployed along with the source statistics $\mathcal{S}_s$. At this point some domain shift occurs (Figure 4.1a) and we receive n_t unlabelled examples from the target domain, $\mathcal{D}_t = \{\mathbf{x}_t^{(i)}\}_{i=1}^{n_t}$.

Given a batch of data from $\mathcal{D}_t$ we can approximate each of the D marginal feature distributions and K marginal logit distributions in the target domain using the same soft binning process described in Section 4.4.1.1. This gives a parameterisation of the target domain distributions using $\pi^t_{\mathbf{z}}$ and $\pi^t_{\mathbf{a}}$. In order to align the target distributions with the source distributions we retrain the feature extractor (g_s to g_t) such that the Kullback–Leibler (KL) divergence (Kullback, 1959) between the source and target marginal distributions is minimised[5]. To do this we minimise the following loss function using the target domain data,

$$\mathcal{L}_{tgt}(\pi^s_{\mathbf{z}}, \pi^t_{\mathbf{z}}, \pi^s_{\mathbf{a}}, \pi^t_{\mathbf{a}}) = \sum_{d=1}^{D} D_{SKL}(\pi^s_{z_d} || \pi^t_{z_d}) + \sum_{k=1}^{K} D_{SKL}(\pi^s_{a_k} || \pi^t_{a_k}), \tag{4.3}$$

where $D_{SKL}(p||q) = \frac{1}{2}D_{KL}(p||q) + \frac{1}{2}D_{KL}(q||p)$ is the symmetric KL divergence that can be easily calculated with our binning parameterisation using[6]

$$D_{KL}(\pi^s_{z_d} || \pi^t_{z_d}) = \sum_{b=1}^{B} \pi^s_{z_d,b} \log \frac{\pi^s_{z_d,b}}{\pi^t_{z_d,b}}. \tag{4.4}$$

Note that the source distributions are fixed during deployment (as we do not have access to the source data so they cannot be altered) and variable ranges are normalised using the source data minimum and maximum values ($\mathbf{z}^{min}, \mathbf{z}^{max}, \mathbf{a}^{min}, \mathbf{a}^{max}$) to ensure equivalent scaling. This means that $\pi^s_{z_d,b}$ is the probability of a source domain sample for feature d falling into bin b and $\pi^t_{z_d,b}$ similarly for a target domain sample. During adaptation $\pi^t_{\mathbf{z}}$ and $\pi^t_{\mathbf{a}}$ are approximated using a batch of data to calculate Equation 4.2 and then evaluate the loss. As the training of g_t progresses, $\pi^t_{\mathbf{z}}$ and $\pi^t_{\mathbf{a}}$ change as the

[5] Any measure of the difference between distributions that can be minimised should also suffice including other f-divergences or the use of optimal transport Courty et al. (2017).

[6] In a slight overloading of notation we write $D_{KL}(\pi^s_{z_d} || \pi^t_{z_d})$ for the KL divergence between the distributions parameterised by $\pi^s_{z_d}$ and $\pi^t_{z_d}$.

parameters of g_t change to align the marginal distributions, whereas $\pi^s_{\mathbf{z}}$ and $\pi^s_{\mathbf{a}}$ remain fixed.

4.4.2.1 Pseudo-Code Summary

The processes for feature restoration during development and deployment are summarised in pseudo-code in Algorithms 1 and 2. During development the source model is trained and source statistics, $\mathcal{S}_s$, calculated. During deployment the feature extractor, g_t, is learned in order to realign the target feature distributions with those saved in the source domain.

Algorithm 1: Feature Restoration at *development* time.

Input: Untrained source model f_s, labelled source data $\mathcal{D}_s = (\mathcal{X}_s, \mathcal{Y}_s)$, number of bins B, number of training iterations I.

`/* Train source model` $f_s = h \circ g_s$ `*/`

for *i in range(I)* **do**

  $L_i \leftarrow \mathcal{L}_{src}(f_s, D_s)$;

  $f_s \leftarrow \text{SGD}(f_s, L_i)$;

`/* Calculate feature and logit ranges */`

$\mathbf{z}^{min}, \mathbf{z}^{max} \leftarrow \text{CALC\_RANGE}(g_s, X_s)$;

$\mathbf{a}^{min}, \mathbf{a}^{max} \leftarrow \text{CALC\_RANGE}(f_s, X_s)$;

`/* Calculate feature and logit bin counts */`

$\pi^s_{\mathbf{z}} \leftarrow \text{CALC\_BC}(g_s, X_s; \mathbf{z}^{min}, \mathbf{z}^{max}, B)$;

$\pi^s_{\mathbf{a}} \leftarrow \text{CALC\_BC}(f_s, X_s; \mathbf{a}^{min}, \mathbf{a}^{max}, B)$;

`/* Gather source stats` $\mathcal{S}_s$ `*/`

$\mathcal{S}_s \leftarrow \{\pi^s_{\mathbf{z}}, \pi^s_{\mathbf{a}}, \mathbf{z}^{min}, \mathbf{z}^{max}, \mathbf{a}^{min}, \mathbf{a}^{max}\}$;

Output: $f_s, \mathcal{S}_s$

Algorithm 2: Feature Restoration at *deployment* time.

Input: Trained source model f_s, unlabelled target data $\mathcal{X}_t$, source data statistics $\mathcal{S}_s$, number of adaptation iterations I.

`/* Initialise target model` $f_t = h \circ g_t$ `*/`

$f_t \leftarrow f_s$;

`/* Adapt target feature extractor` g_t `*/`

for *i in range(I)* **do**

$\pi^t_{\mathbf{z}} \leftarrow \text{CALC\_BC}(g_t, X_t; \mathbf{z}^{min}, \mathbf{z}^{max}, B)$;

$\pi^t_{\mathbf{a}} \leftarrow \text{CALC\_BC}(f_t, X_t; \mathbf{a}^{min}, \mathbf{a}^{max}, B)$;

$L_i \leftarrow \mathcal{L}_{tgt}(\pi^s_{\mathbf{z}}, \pi^t_{\mathbf{z}}, \pi^s_{\mathbf{a}}, \pi^t_{\mathbf{a}})$;

$g_t \leftarrow \text{SGD}(g_t, L_i)$;

Output: g_t

4.4.2.2 The Bottom-Up Inductive Bias

Intuitively many of the measurement shifts we consider should be largely resolvable by changing parameters in the early layers of g_t. That is, if we can learn to extract features with the same semantic meaning across domains in the early layers then the combining of these features performed by later layers does not need to change allowing us to preserve learned structure. For example if we can learn to extract the same initial edges from an image after a change in brightness (or a change in colour as in Figure 4.1b), the remainder of g_t should still be valid.

However, updating with a standard gradient based update will adapt all parameters of g_t simultaneously. We can include the inductive bias outlined above by adapting g_t in a *bottom-up* manner where successive layers (or blocks of layers for ResNets (He et al., 2016)) are gradually unfrozen. This means the first layer (or block) is trained for several epochs before also allowing the second layer (or block) to update and continuing to add one layer (or block) at a time until all parameters of g_t are able to update.

With this additional inductive bias, we call our method *Bottom-Up Feature Restoration* (BUFR) and show in the results (Section 4.5) that this structure preserving regularisation significantly improves classification performance, data efficiency and model calibration. In the next chapter (Chapter 5) we will investigate this intuition further and explore which parameters are best to update for different types of change in domain.

4.4.3 Model Implementation Details

Before exploring the performance and behaviour of feature restoration for source-free domain adaptation we will briefly outline the technical implementation of our methods and the baseline methods with which we can compare.

We evaluate all our methods on multiple datasets described in Section 4.3 and Appendix B.1. For datasets using digits and characters, that is, when the source domain data is MNIST or EMNIST, we use a simple 5 layer convolutional architecture based on LeNet (LeCun et al., 1998). The specifics are shown in Table 4.3. For all other datasets (object recognition and medical diagnosis) we use a standard ResNet-18 (He et al., 2016).

Table 4.3: LeNet based network architecture used for digit and character datasets. For convolutions, the weights-shape entry is: *num. input channels* $\times$ *num. output channels* $\times$ *filter height* $\times$ *filter width*.

Block	**Weights-Shape**	**Stride**	**Padding**	**Activation**	**Dropout Prob.**
Conv + BN	$3 \times 64 \times 5 \times 5$	2	2	ReLU	0.1
Conv + BN	$64 \times 128 \times 3 \times 3$	2	2	ReLU	0.3
Conv + BN	$128 \times 256 \times 3 \times 3$	2	2	ReLU	0.5
Linear + BN	6400×128	N/A	N/A	ReLU	0.5
Linear	$128 \times$ Number of Classes	N/A	N/A	Softmax	0

During the development phase we use validation sets to select learning rates and number of training epochs for each dataset and architecture. Dropout is used for regularisation as shown in Table 4.3. After development the last linear layer is frozen as the classifier h and the remaining network parameters are fine-tuned to learn g_t. The dropout is turned off when gathering source statistics and during deployment as we do not want to randomly drop samples when parameterising the marginal distributions. During adaptation (deployment) all methods are trained using stochastic gradient descent (SGD) with a momentum of 0.9 and results reported over 5 random runs. Where applicable a batch size of 256 is used. We use the best performing learning rate from $\{0.0001, 0.001, 0.01, 0.1, 1\}$. It is worth noting that in order to select the best learning rate we must use a labelled target domain validation set as is common practice in UDA (Gulrajani and Lopez-Paz, 2021) although often not made explicit[7]. When

[7] A labelled target domain dataset for model selection may not actually be available in a real world UDA/SFDA situation.

training bottom-up we use 30 epochs per block and gradually decay the learning rate as a function of the number of unfrozen blocks, all other methods are trained with a constant learning rate for 150 epochs.

4.4.3.1 Baselines

After pretraining (development) we report the performance of the source model on the source domain data as *no corruption* and the performance before adaptation on the target data as *source-only*. We also create an approximate upper-bound on performance, *target-supervised*, which trains on the target domain using labelled data and reports test accuracy using an 80-10-10 training-validation-test split. We then compare our method with some UDA results reported in the literature and implement several baselines for direct comparison which we briefly summarise here.

AdaBN. Replace the batch normalisation statistics of the source data with those of the target data (Li et al., 2017a, 2018a). We also use this method to measure shift 'severity', that is, how hard it is to resolve a given change in domain, as some shifts with low source-only accuracy can be well resolved with this simple baseline.

PL. A basic pseudo-labelling approach (Lee et al., 2013) which selects the label with the highest predicted probability on the target data and uses this as if it were the real label to train with cross-entropy.

SHOT-IM. The information-maximisation loss used by Liang et al. (2020) which aims to minimise the entropy of single sample softmax predictions (make close to one-hot) while maximising the entropy of the average softmax prediction (make close to uniform across all samples)

SHOT. The full method of Liang et al. (2020) which uses the *SHOT-IM* loss along with a self-supervised pseudo-labelling loss. Additionally specific pretraining techniques are used (e.g. label smoothing).

BNM-IM. The method used by Ishii and Sugiyama (2021) which combines the *SHOT-IM* loss with a batch norm matching loss (*BNM*) that aims to align the means and variances of the target domain features with the means and variances captured by the source model batch normalisation statistics.

Marg. Gauss. An alternative parameterisation for aligning the marginal distributions where we approximate them with $1D$ Gaussian distributions. This can be done using

the *BNM* loss of Ishii and Sugiyama (2021).

Full Gauss. Rather than aligning marginal distributions we experiment with aligning a Gaussian approximation to the full D-dimensional joint feature distribution. An empirical mean vector and a covariance matrix are estimated over the source data after pretraining the model. To adapt to the target data, empirical mean and covariances are calculated for each batch and the distributions aligned using the analytical formula for the KL divergence between multivariate Gaussians (Duchi, 2007, Sec. 9). We use $D_{KL}(Q||P)$ where Q is the target domain distribution and P the source domain as, in this direction, we only need invert the covariance matrix for P once, which can be done offline during development, rather than for Q on every batch.

TENT. Minimise the prediction entropy (close to one hot) by updating only batch normalisation parameters (Wang et al., 2021). We note that *TENT* is designed for an 'online' setting, where only a single epoch of training is permitted. Whilst we primarily evaluate in the 'offline' situation we also evaluate some of our methods in this single epoch set-up. In the online scenario we find methods are very sensitive to the learning rate and thus search over learning rates $\{0.1, 0.01, 0.001, 0.0001\}$ and report the performance for the best learning rate for a specific method. Additionally, as learning quickly is very important in this setting, we increase the temperature parameter τ in Equation 4.1 to 0.05. Roughly speaking, this gives a 'lower resolution' approximation which encourages alignment in the most important areas.

4.4.3.2 Model Calibration

Alongside model accuracy we also evaluate model calibration with the Expected Calibration Error (ECE, Naeini et al. (2015)). ECE measures the expected difference between model confidence and model accuracy, and is better when it is lower. As an example, we would like that when a binary classification model predicts class A with probability 0.8 and class B with probability 0.2 then the true class label is class A 80% of the time and class B 20% of the time. If this is true over all probability values the model can predict, we say the model is well calibrated. A well calibrated model means that when our model predicts with high confidence we can trust its predictions.

To calculate ECE we bin predictions into 10 evenly-spaced bins based on confidence and take the absolute difference between average accuracy and average confidence of the samples in each bin. To get a single value, the absolute differences per bin are

averaged over all bins, weighted by the normalised bin counts. We can also consider the Maximum Calibration Error (MCE) which is the maximum absolute difference over the bins.

Model calibration can also be visualised using reliability diagrams (DeGroot and Fienberg, 1983; Niculescu-Mizil and Caruana, 2005) and confidence histograms (Zadrozny and Elkan, 2001). Reliability diagrams plot average accuracy for samples in a bin against the confidence bins, with a well calibrated model showing bars with heights approximating the $y = x$ graph. Since reliability diagrams do not show the underlying distribution of model prediction confidences, they can be paired with a histogram of the confidences to visualise the underlying frequencies with which samples occur in each bin (as in Guo et al. (2017, Figure 1)).

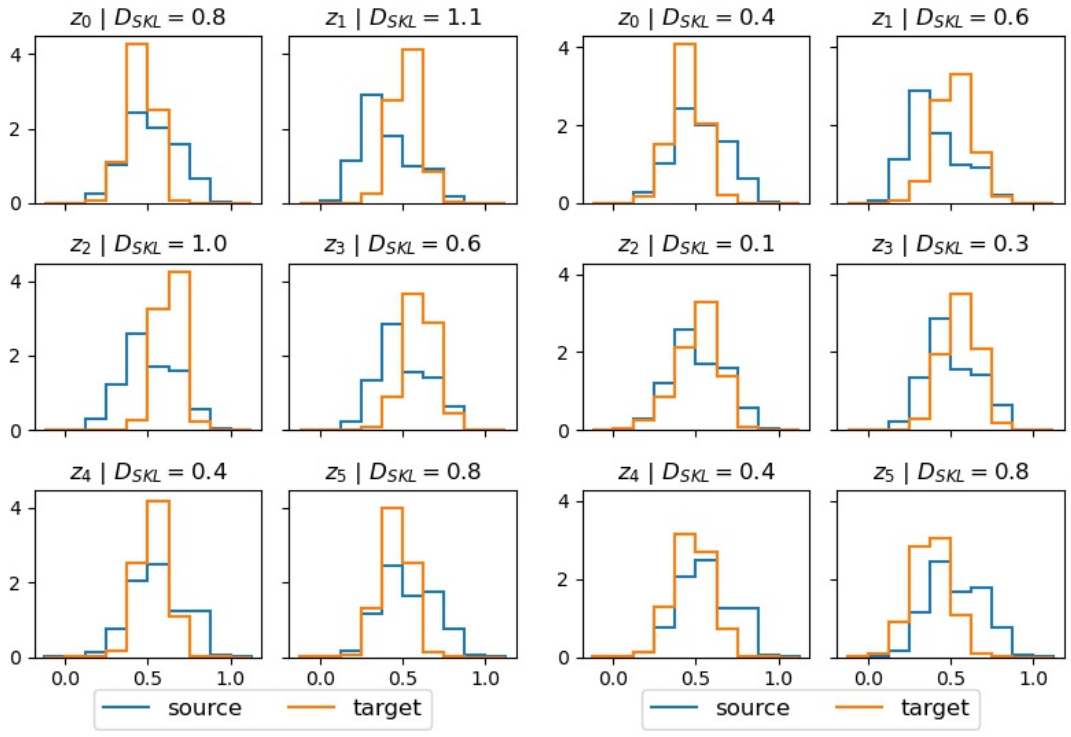
4.5 Performance and Analysis

We now have a full description of our approach to source-free domain adaptation for measurement shifts and can evaluate the pros and cons of this technique. We both compare models quantitatively and analyse model behaviour to try and gain a deeper understanding of observed phenomena.

4.5.1 Distribution Alignment

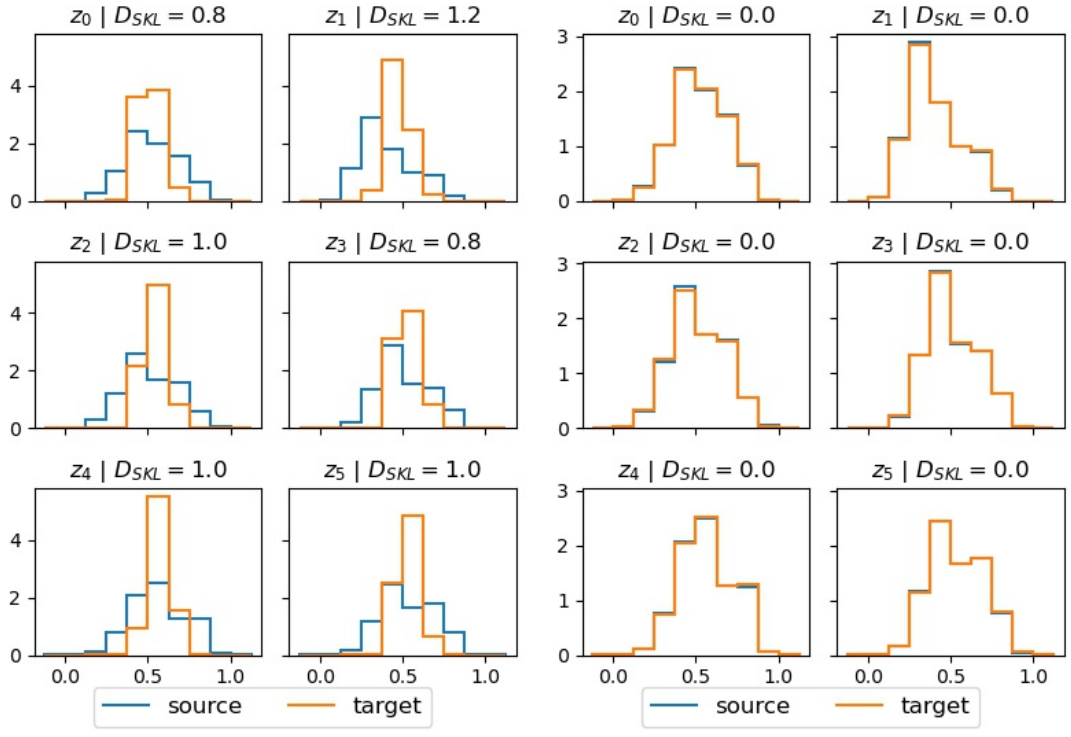
Figure 4.5a shows source and target histogram parameterisations for 6 marginal distributions when adapting on the EMNIST-DA stripe shift. We observe that the source distributions may be heavily skewed (i.e. not Gaussian) and that the target activation distributions before training g_t are substantially different from the source activation distributions.

Figures 4.5b, 4.5c & 4.5d show the same source distributions with the target distributions after using different adaptation methods. Figure 4.5d qualitatively shows that the loss function we have designed (Equation 4.3) does achieve our objective of closely aligning marginal feature distributions. Figures 4.5b & 4.5c show that other methods for SFDA achieve their results without achieving the same level of distribution alignment. In Appendix B.2 we show similar figures for CIFAR-10-C which show that marginal distributions may also be multimodal. Appendix B.2 additionally shows marginal logit distributions.



(a) Source-Only

(b) AdaBN



(c) SHOT-IM

(d) BUFR

Figure 4.5: Histograms showing distribution alignment on the EMNIST-DA stripe shift. The blue curves are the saved marginal distributions under the source data (identity). The orange curves are the marginal distributions under the target data (stripe). (a): Source-Only, before adaptation distributions are different. (b,c): AdaBN and SHOT-IM, after adapting the marginal distributions are not aligned despite achieving reasonable accuracy. (d) BUFR, after adaptation the activation distributions align very closely, making D_{SKL} very small.

Table 4.4: Digit Dataset Results. Shown are the mean and 1 standard deviation.

Model	**MNIST-C**		**MNIST-M**	
	ACC ↑	**ECE** ↓	**ACC** ↑	**ECE** ↓
No corruption	99.5 ± 0.1	0.3 ± 0.0	99.5 ± 0.1	0.3 ± 0.0
Source-only	86.2 ± 1.8	7.2 ± 1.4	42.7 ± 4.6	27.0 ± 7.1
AdaBN (Li et al., 2017a)	94.2 ± 0.2	4.1 ± 0.1	59.1 ± 1.9	24.4 ± 2.8
PL (Lee et al., 2013)	96.4 ± 0.4	3.1 ± 0.4	43.1 ± 2.1	56.1 ± 2.2
SHOT-IM (Liang et al., 2020)	97.3 ± 0.2	2.3 ± 0.2	66.9 ± 9.3	30.9 ± 9.0
SHOT (Liang et al., 2020)	$\mathbf{97.7 \pm 0.2}$	$\mathbf{2.0 \pm 0.2}$	94.4 ± 3.1	$\mathbf{2.8 \pm 2.9}$
FR (ours)	96.7 ± 0.1	2.5 ± 0.2	86.5 ± 0.6	9.8 ± 0.8
BUFR (ours)	96.4 ± 0.6	3.0 ± 0.6	$\mathbf{96.2 \pm 1.7}$	2.9 ± 1.5
Target-supervised	99.3 ± 0.0	0.5 ± 0.0	98.5 ± 0.0	1.1 ± 0.1

4.5.2 Quantitative Performance

We now know that our loss function creates the desired behaviour so can evaluate the quantitative performance of our feature restoration (FR) and bottom-up feature restoration (BUFR) methods for SFDA on measurement shifts. Initial experiments on MNIST-M and MNIST-C (Table 4.4) showed that, for these very simple datasets, all methods achieve good performance with accuracy greater than 95% and low calibration error. Full per-shift results for all datasets are given in Appendix B.3, these show that if we remove one assumption breaking corruption from MNIST-C, BUFR and SHOT (the strongest baseline) perform equivalently. That is, due to the low number of classes and the relatively mild corruptions, the MNIST based datasets are not challenging enough to highlight major differences between different SFDA approaches.

It is for this reason that we created the more challenging EMNIST-DA dataset for which results are shown in Table 4.5. This table reports classification accuracy and ECE averaged over all shifts as well as the most severe and mild individual domain shifts, as measured by AdaBN performance. Using this dataset, containing many more challenging shifts, BUFR convincingly outperforms all baseline methods in terms of both accuracy and calibration. This is particularly noticeable on severe shifts where initial feature-space class-separation, that is, the extent to which classes from the target domain cluster in feature space on the source model, is likely poor. We believe that this is important because methods that make use of some form of entropy minimisation (PL,

Table 4.5: Digit and Character Results. Shown are the mean and 1 standard deviation.

Model	**EMNIST-DA**		**EMNIST-DA-SEVERE**		**EMNIST-DA-MILD**	
	ACC ↑	**ECE** ↓	**ACC** ↑	**ECE** ↓	**ACC** ↑	**ECE** ↓
No corruption	89.4 ± 0.1	2.3 ± 0.1	89.4 ± 0.1	2.3 ± 0.1	89.4 ± 0.1	2.3 ± 0.1
Source-only	29.5 ± 0.5	30.8 ± 1.6	3.8 ± 0.4	42.6 ± 3.5	78.5 ± 0.7	4.8 ± 0.5
AdaBN (Li et al., 2017a)	46.2 ± 1.1	30.3 ± 1.1	3.7 ± 0.7	52.4 ± 4.9	84.9 ± 0.2	4.9 ± 0.3
Marg. Gauss. (Ishii and Sugiyama, 2021)	51.8 ± 1.1	26.7 ± 1.1	4.8 ± 0.5	51.6 ± 6.4	85.8 ± 0.3	4.5 ± 0.3
Full Gauss.	67.9 ± 0.7	17.4 ± 0.7	29.8 ± 9.8	45.8 ± 8.4	85.7 ± 0.2	4.9 ± 0.2
PL (Lee et al., 2013)	50.0 ± 0.6	49.9 ± 0.6	2.7 ± 0.4	97.2 ± 0.4	83.5 ± 0.1	16.4 ± 0.1
BNM-IM (Ishii and Sugiyama, 2021)	63.7 ± 2.2	35.6 ± 2.2	8.3 ± 1.3	90.2 ± 1.1	86.5 ± 0.1	13.0 ± 0.1
SHOT-IM (Liang et al., 2020)	70.3 ± 3.7	29.6 ± 3.7	24.0 ± 7.5	76.0 ± 7.5	86.3 ± 0.1	13.7 ± 0.1
SHOT (Liang et al., 2020)	80.0 ± 4.4	19.7 ± 4.4	55.1 ± 23.5	42.7 ± 23.0	86.1 ± 0.1	14.8 ± 0.1
FR (ours)	74.4 ± 0.8	12.9 ± 0.9	15.3 ± 6.8	58.0 ± 6.8	86.4 ± 0.1	4.6 ± 0.3
BUFR (ours)	**86.1 ± 0.1**	**4.7 ± 0.2**	**84.6 ± 0.2**	**5.6 ± 0.3**	**87.0 ± 0.2**	**4.2 ± 0.2**
Target-supervised	86.8 ± 0.6	7.3 ± 0.7	85.7 ± 0.6	7.0 ± 0.5	87.3 ± 0.7	8.4 ± 1.1

SHOT-IM, SHOT, BNM-IM) essentially work to make predictions more confident but, if initial predictions are poor due to a mixing of class representations in feature space, increasing confidence alone may not restore classification performance. A large variance across runs for severe shifts with SHOT-IM and SHOT provides further evidence for this reasoning; the large variability suggests the differences in the initial target domain feature space across runs have a big impact on how well the adapted model will perform.

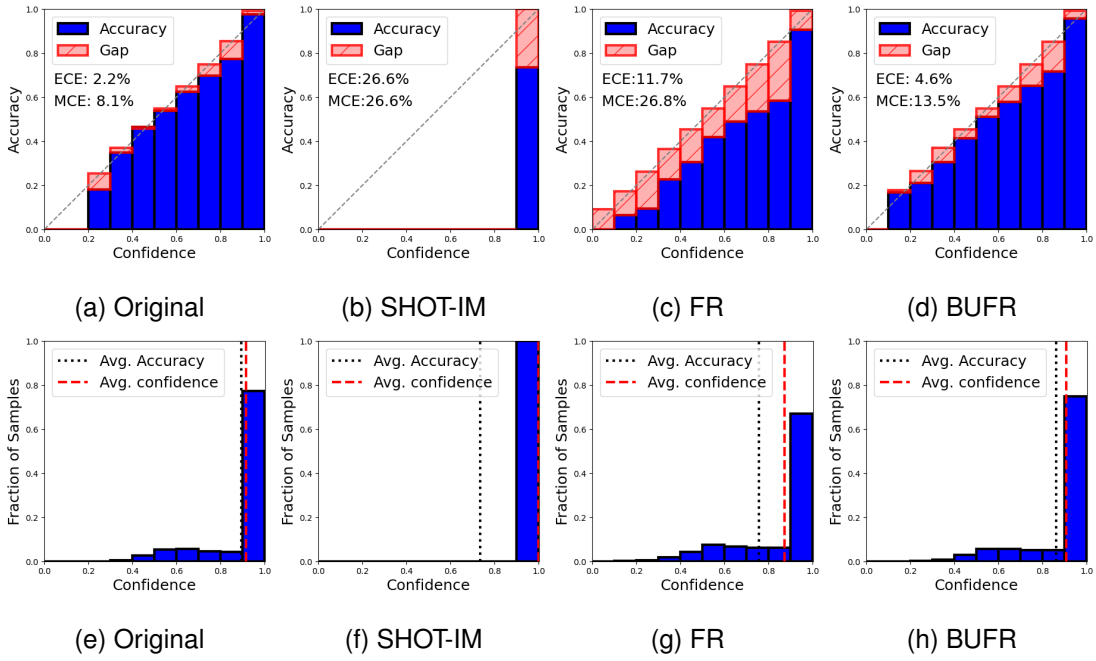
On the most severe shift only BUFR achieves good performance in the target domain after adaptation. For the mild shift all methods perform well, but we still see that BUFR performs the best and that the methods that make use of entropy minimisation are much more poorly calibrated than those that don't, due to maximising the confidence of predictions. Indeed, the calibration benefits of methods that lack entropy minimisation terms when optimising g_t (AdaBN, Marg. Gauss, Full Gauss., FR, BUFR) are seen most clearly in the mild shift where these methods are notably better calibrated than other methods since they do not work to make predictions more confident.

4.5.2.1 Visualising Model Calibration

Figures 4.6, 4.7 & 4.8 emphasise the differences in model calibration for our FR methods when compared to an entropy minimising method in SHOT-IM. These figures show reliability diagrams alongside *ECE* and *MCE* values for all EMNIST-DA shifts (Figure 4.6), the most severe EMNIST-DA shift (Figure 4.7), and the most mild EMNIST-DA shift (Figure 4.8). The reliability diagrams are paired with confidence histograms to

visualise the underlying frequencies with which predictions are made within a given confidence range.

As a general behaviour, compared to the original model pretrained and evaluated in the source domain (Figures 4.6a & 4.6e), the SFDA methods shown all tend towards overconfidence but our methods significantly less so. The size of the red 'Gap' bar below the line $y = x$ shows the extent of this overconfidence. Looking at the rightmost bin in Figures 4.6b, 4.6c & 4.6d we see that when FR methods predict with high confidence they are much more likely to be accurate than SHOT-IM which simply predicts everything with high confidence as seen in Figure 4.6f. We also see that BUFR remains well calibrated even on severe shifts whereas SHOT-IM confidently makes incorrect predictions (Figure 4.7). As seen in Table 4.5, our methods are also better calibrated on the mild shift (Figure 4.8).



(a) Original (b) SHOT-IM (c) FR (d) BUFR

(e) Original (f) SHOT-IM (g) FR (h) BUFR

Figure 4.6: Reliability diagrams and confidence histograms over *all* EMNIST-DA corruptions. *(a-d):* Reliability diagrams. *(e-h):* Corresponding confidence histograms. *(a & e):* The 'Original' source model is well-calibrated on the *source data*. *(b & f):* Minimising prediction entropy leads to overconfidence. *(c & g, d & h):* Our methods, FR and BUFR, which do not make use of entropy minimisation, are more well calibrated.

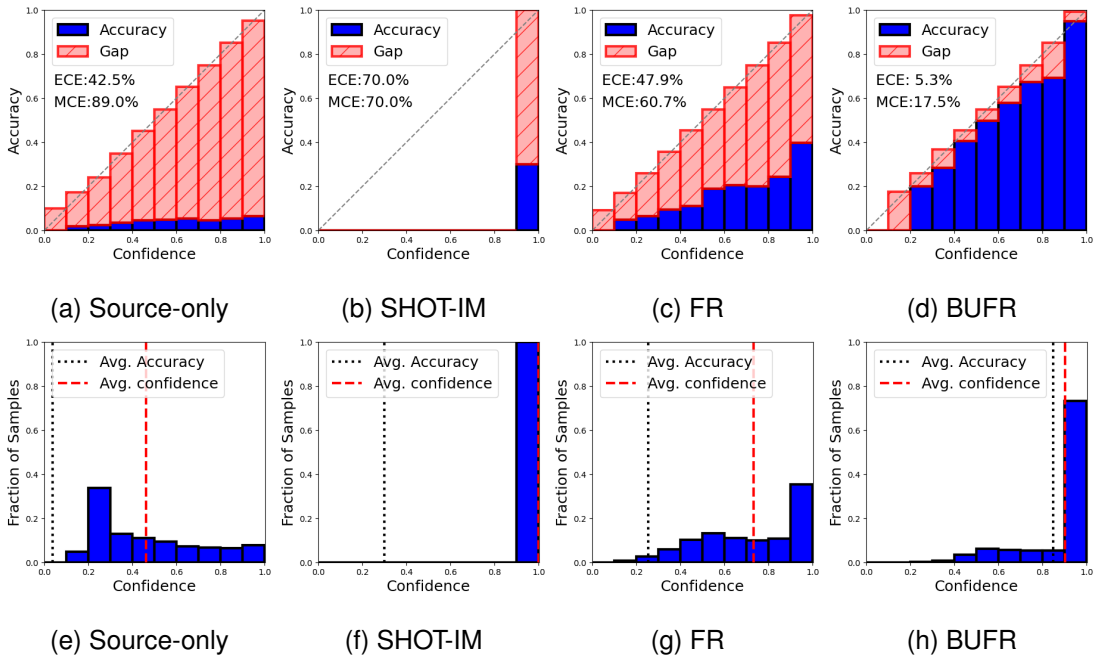


Figure 4.7: Reliability diagrams and confidence histograms for a severe EMNIST-DA shift (sky). *(a-d):* Reliability diagrams. *(e-h):* Corresponding confidence histograms. *(a & e)*: The source model on the *target data* achieves poor accuracy but generally predicts with low confidence. *(b & f)*: SHOT-IM confidently makes incorrect predictions. *(d & h)*: Our BUFR method is more well calibrated.

4.5.2.2 Object Recognition Results

We evaluate our method on CIFAR-10-C and CIFAR-100-C to see how performance changes when classifying images that are more complex than single handwritten characters. Table 4.6 gives the classification accuracies and expected calibration errors for different methods on these datasets. Note that this table contains the results reported for two UDA (i.e. not source-free) methods (Ganin et al., 2016; Sun et al., 2019).

For this classification task we again see that BUFR gives the best performance, providing the highest accuracies and lowest ECEs (except for ECE on CIFAR-100-C). More importantly we see similar trends to those seen when using EMNIST-DA. Firstly, we see that the bottom-up inductive bias significantly improves performance. Secondly, that whilst entropy minimisation techniques are the most competitive baselines in terms of classification accuracy these methods are poorly calibrated compared to methods that avoid maximising confidence. Finally, that the benefits of BUFR are most clearly realised for severe shifts, for example, in the CIFAR-10-C per-shift results of Appendix B.3 the impulse-noise shift achieves 75% accuracy with the best

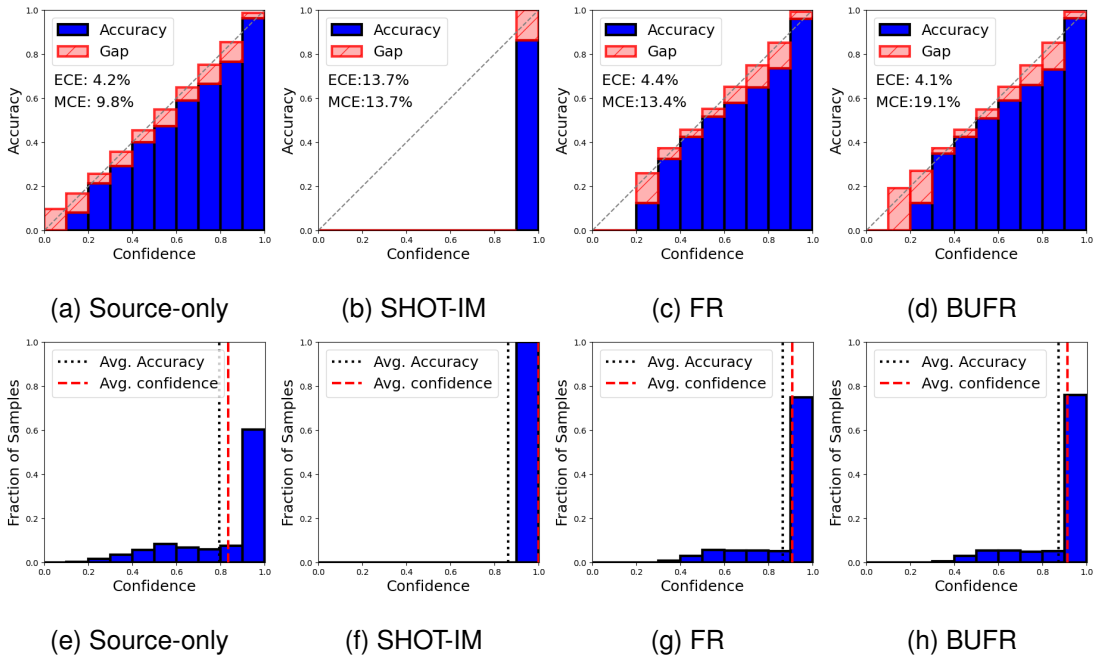


Figure 4.8: Reliability diagrams and confidence histograms for a mild EMNIST-DA shift (shot noise). *(a-d):* Reliability diagrams. *(e-h):* Corresponding confidence histograms. Examining the rightmost bin show that when FR and BUFR are confident *(c & g, d & h)*, predictions are more accurate than for SHOT-IM *(b & f)*.

performing baseline whereas BUFR achieves 89%.

Finally, we note that several recent SFDA methods outperform slightly older UDA methods suggesting that the use of heuristics, such as 'what does the output of a good classifier look like?' or 'gradually make predictions on the input you are seeing more and more confident,' is an extremely powerful approach to learning given a reasonable initial model to transfer from. Surprisingly, in Table 4.6 BUFR even outperforms the target-supervised baseline. We attribute this to the efficacy of the bottom-up inductive bias and the relatively small number of target domain samples which must be reduced further into training, validation and test splits for supervised learning.

4.5.2.3 Online Results

Although not our main focus, we also evaluate methods in the 'online' setting of Wang et al. (2021) where we view the target domain data as an online stream of incoming data so are restricted to one pass through the data and care primarily about how fast and how well the model can be made to work in the target domain. For this reason the performance is measured as the average accuracy over minibatches *during training*

Table 4.6: Object Recognition Results. Shown are the mean and 1 standard deviation.

Model	**CIFAR-10-C**		**CIFAR-100-C**	
	ACC ↑	**ECE** ↓	**ACC** ↑	**ECE** ↓
No corruption	95.3 ± 0.2	2.4 ± 0.1	76.4 ± 0.2	4.8 ± 0.1
DANN* (Ganin et al., 2016)	81.7	-	61.1	-
UDA-SS.* (Sun et al., 2019)	83.3	-	53	-
Source-only	57.8 ± 0.7	28.2 ± 0.4	36.4 ± 0.5	19.4 ± 0.9
AdaBN (Li et al., 2017a)	80.4 ± 0.1	11.2 ± 0.1	56.6 ± 0.3	$\mathbf{12.5 \pm 0.1}$
PL (Lee et al., 2013)	82.5 ± 0.3	17.5 ± 0.3	62.1 ± 0.2	37.7 ± 0.2
SHOT-IM (Liang et al., 2020)	85.4 ± 0.2	14.6 ± 0.2	67.0 ± 0.2	32.9 ± 0.2
TENT (Wang et al., 2021)	86.6 ± 0.3	12.8 ± 0.3	66.0 ± 0.4	25.7 ± 0.4
FR (ours)	87.2 ± 0.7	11.3 ± 0.3	65.5 ± 0.2	15.7 ± 0.1
BUFR (ours)	$\mathbf{89.4 \pm 0.2}$	$\mathbf{10.0 \pm 0.2}$	$\mathbf{68.5 \pm 0.2}$	14.5 ± 0.3
Target-supervised	88.4 ± 0.9	6.4 ± 0.6	68.1 ± 1.2	9.6 ± 0.7

rather than after training has finished. In such a situation BUFR is not easily applicable as we wish to achieve good performance with a few minibatches whereas when training bottom-up parameters are more slowly trained.

Table 4.7 therefore compares FR with relevant SFDA baselines. For CIFAR-10-C we note that FR achieves the best accuracy and calibration and is competitive on CIFAR-100-C even when compared to TENT, a method specifically designed for this setting. We again show full per-shift results in Appendix B.3.

Table 4.7: Online Results. Shown are the mean and 1 standard deviation.

Model	**CIFAR-10-C**		**CIFAR-100-C**	
	ACC ↑	**ECE** ↓	**ACC** ↑	**ECE** ↓
AdaBN (Li et al., 2017a)	80.3 ± 0.0	12.1 ± 0.0	56.6 ± 0.3	$\mathbf{10.0 \pm 0.1}$
SHOT-IM (Liang et al., 2020)	83.2 ± 0.2	10.9 ± 0.1	62.3 ± 0.3	13.8 ± 0.1
TENT (Wang et al., 2021)	81.8 ± 0.2	11.5 ± 0.1	$\mathbf{63.1 \pm 0.3}$	14.3 ± 0.1
FR (ours)	$\mathbf{85.9 \pm 0.3}$	$\mathbf{9.5 \pm 0.2}$	62.7 ± 0.3	13.6 ± 0.1

4.5.3 Few-shot Feature Restoration

So far we have largely examined synthetically generated corruptions or shifts applied to a dataset. We use the CAMELYON17 dataset as an example of a real world measurement

shift where histopathological images are taken using different scanners at different hospitals. A ResNet-18 is trained on images from a single hospital and achieves 99.3% accuracy, we then adapt models on the images from a different hospital. Table 4.8 shows results averaged over the 4 target domains (different hospitals).

Despite the fact that this dataset is an ideal candidate for entropy minimising SFDA methods due to a high AdaBN accuracy (most pseudo-labels will be correct after updating batch normalisation statistics) and only two possible classes (giving random pseudo-labels a 50% chance of being correct), our methods still achieve competitive accuracy when trained on all dataset samples.

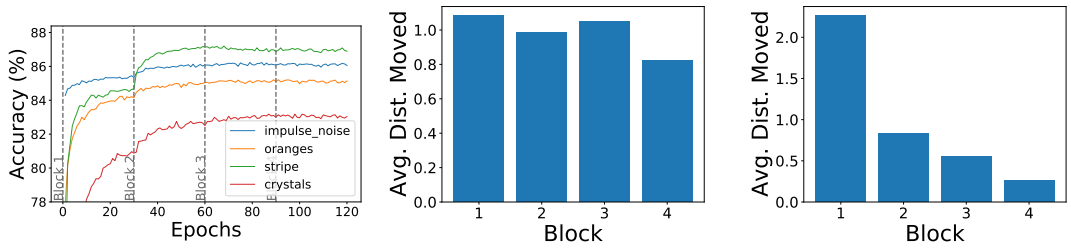
More interesting in this case is how different methods respond when only a small amount of data is available, as may be the case in a deployed system. Table 4.8 shows that when we have 50 samples or fewer, only the feature restoration methods significantly improve over the AdaBN baseline that simply uses target rather than source domain batch normalisation statistics (which happens for all methods during training). These results show that feature restoration methods perform well in practice on a real world measurement shift and that we achieve better data efficiency.

Table 4.8: CAMELYON17 accuracies for varying samples per class in the target domain.

Model	**5**	**10**	**50**	**500**	**All**($>$ **15k**)
Source-only	55.8 ± 1.6	55.8 ± 1.6	55.8 ± 1.6	55.8 ± 1.6	55.8 ± 1.6
AdaBN (Li et al., 2017a)	82.6 ± 2.2	83.3 ± 2.3	83.7 ± 1.0	83.9 ± 0.8	84.0 ± 0.5
PL (Lee et al., 2013)	82.5 ± 2.0	83.7 ± 1.7	83.6 ± 1.2	85.0 ± 0.8	$\mathbf{90.6 \pm 0.9}$
SHOT-IM (Liang et al., 2020)	82.6 ± 2.2	83.4 ± 2.5	83.7 ± 1.2	86.4 ± 0.7	89.9 ± 0.2
FR (ours)	$\mathbf{84.6 \pm 0.6}$	86.0 ± 0.7	86.0 ± 1.1	89.0 ± 0.6	89.5 ± 0.4
BUFR (ours)	84.5 ± 0.8	$\mathbf{86.1 \pm 0.2}$	$\mathbf{87.0 \pm 1.2}$	$\mathbf{89.1 \pm 0.8}$	89.7 ± 0.5

We also analyse the effect of the bottom-up inductive bias for improving data efficiency on EMNIST-DA. Figure 4.9a confirms the intuition of Section 4.4.2.2 that changing parameters in the early layers should be sufficient to resolve many measurement shifts. This figure shows that, when training bottom-up, training the first two blocks provides almost all of the accuracy improvements. BUFR exploits this behaviour by initially updating the early layers of a network which allows us to update fewer parameters in order preserve learned structure in the later layers. Indeed, Figure 4.9b shows that FR moves[8] parameters in all layers a similar amount whereas Figure 4.9c shows that for

[8]The amount a parameter is moved is calculated by the Euclidean distance between parameter values before and after adaptation.



(a) BUFR training curves (b) FR distance moved (c) BUFR distance moved

Figure 4.9: Bottom-Up Training Analysis. (a) BUFR accuracy during training on selected EMNIST-DA corruptions. Dashed-grey lines indicate when the next block is unfrozen. (b, c) Average Euclidean distance moved by parameters in each block when adapting g_s to g_t using the same constant learning rate. (b) FR moves all parameters a similar amount. (c) BUFR changes primarily the early layers.

BUFR it is primarily the early parameters that are altered.

The regularisation benefits of changing the 'right' few parameters using BUFR are shown in Table 4.9. As the number of training samples reduces, BUFR begins to outperform all baseline methods by a large margin ($> 20\%$). In general, without this bottom-up training, performance drops sharply as the amount of training data reduces. In fact, with only 5 samples per class BUFR outperforms some baselines when using all the data (400 samples per class). The more general question of which are the 'right' few parameters will motivate our investigations in Chapter 5.

Table 4.9: EMNIST-DA accuracy with varying numbers of samples per class.

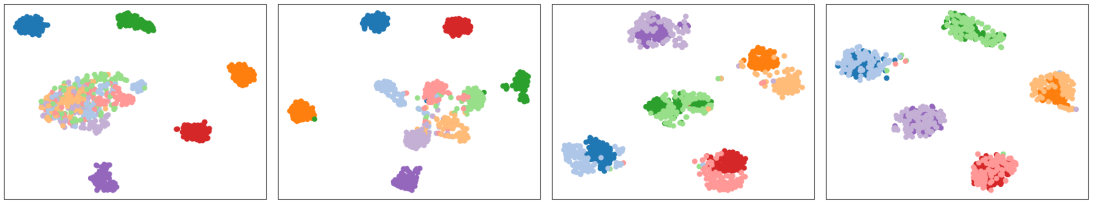
Model	**5**	**10**	**20**	**50**	**400**
Marg. Gauss.	49.3	49.9	50.4	50.7	50.6
Full Gauss.	55.4	59.7	61.0	63.3	68.3
PL	45.8	46.3	46.0	46.7	49.7
BNM-IM	50.5	51.5	53.0	54.7	61.4
SHOT-IM	48.3	51.7	51.2	54.7	73.4
FR (ours)	50.8	50.5	60.1	63.1	75.6
BUFR (ours)	**78.0**	**82.3**	**83.8**	**84.9**	**86.2**

Table 4.10: EMNIST-DA degree of restoration.

Model	**D**
Source-only	3.2 ± 0.0
AdaBN (Li et al., 2017a)	3.1 ± 0.1
Marg. Gauss. (Ishii and Sugiyama, 2021)	2.9 ± 0.0
Full Gauss.	2.0 ± 0.0
PL (Lee et al., 2013)	2.6 ± 0.0
BNM-IM (Ishii and Sugiyama, 2021)	2.5 ± 0.1
SHOT-IM (Liang et al., 2020)	2.9 ± 0.1
FR (ours)	1.8 ± 0.0
BUFR (ours)	$\mathbf{1.2 \pm 0.0}$

4.5.4 Further Analysis

To better understand how feature restoration works we perform several minor analyses consisting of: examining the feature-space class-separation; quantifying the degree of



(a) Source-only (19%) (b) SHOT-IM (67%) (c) FR (77%) (d) BUFR (83%)

Figure 4.10: t-SNE (Van der Maaten and Hinton, 2008) applied to the feature representations of 5 classes from the EMNIST-DA crystals shift. Dark colours show the source data, light the target. Model accuracies are shown in parentheses.

feature restoration; examining the semantic meaning of features, and a small ablation study.

4.5.4.1 Feature-Space Class-Separation

We use t-SNE (Van der Maaten and Hinton, 2008) to see how latent feature representations cluster in the feature space, see Figure 4.10. Figure 4.10a shows that measurement shifts can cause feature representations that were separately clustered in feature space of the source domain (dark colours) to become poorly separated in the target domain (light colours). The effect of adapting on this measurement shift with an entropy minimising technique (SHOT-IM) is shown in Figure 4.10b. Whilst minimising entropy does better separate the classes we see that the feature representations are neither fully separated nor returned to their original locations in the source domain feature space. In contrast Figures 4.10c & 4.10d show that our feature restoration methods both more homogeneously cluster the target domain representations and return them to their original locations.

4.5.4.2 Quantifying the Degree of Feature Restoration

In Table 4.10 we quantify the degree to which features are restored using EMNIST-DA. As this data is synthetically generated we can create paired source and target domain data (identical L in Figure 4.1a). With this paired data we can calculate the average pairwise distance over all domains,

$$D = \frac{1}{T}\sum_{t=1}^{T}\frac{1}{N}\sum_{i=1}^{N}|g_s(x_s^{(i)}) - g_t(x_t^{(i)})|, \tag{4.5}$$

where T is the number of target domains, $x_s^{(i)}$ is a source domain (identity) image and $x_t^{(i)}$ the same image shifted in the target domain (e.g. colour change, noise added etc.), and N the number of paired datapoints. Table 4.10 shows that our feature restoration methods do actually restore the features more than baseline methods. That is, after adaptation, neurons respond more similarly to data with the same underlying L when using our methods. This shows quantitatively what we saw in Section 4.5.1 showing the alignment of marginal distributions, but in this case for individual samples rather than entire batches. In general, pure alignment methods (e.g. Full Gauss.) tend to perform better on this metric than entropy minimising techniques which may find ways to restore performance without restoring features.

4.5.4.3 Semantic Meaning of Features

Used to give the intuition behind feature restoration in Section 4.2, Figure 4.1b shows a real experimental result using the EMNIST-DA grass shift. This figure is created by explicitly aligning the marginal feature distributions in the first layer (rather than the penultimate layer) for illustrative purposes. We see that a neuron that responds to white horizontal edges on a black background in the source domain (left column) continues responding to patterns of light and dark in the target domain before adaptation (middle column). After aligning the marginal distribution (right column) using BUFR, a sense of semantic meaning is restored to the feature so it again detects horizontal image edges (purple edges on a green background).

4.5.4.4 Ablation Study

Table 4.11 shows an ablation study where we drop parts of the loss from Equation 4.3 and see how this affects performance. We experiment with only aligning the feature distributions and not the logits (w/o logits), and with using the asymmetric KL divergence (w/o $D_{KL}(P||Q)$). We see that for the easier CIFAR-10-C task, these loss components are not hugely important. However, as the number of classes increases in CIFAR-100-C, both additionally constraining the feature space by aligning the logits and using the symmetric KL divergence improves performance.

Table 4.11: Ablation study on CIFAR-10-C and CIFAR-100-C.

Model	CFR-10-C	CFR-100-C
$\mathcal{L}_{tgt}$ w/o logits	86.7 ± 0.2	62.3 ± 1.3
$\mathcal{L}_{tgt}$ w/o $D_{KL}(P\|\|Q)$	86.5 ± 0.3	61.5 ± 0.2
$\mathcal{L}_{tgt}$	$\mathbf{87.2 \pm 0.7}$	$\mathbf{65.5 \pm 0.2}$

4.6 Discussions

We end this chapter with some discussions on potential pitfalls with the feature restoration approach and point to ideas from existing works that may inspire further exploration in this direction.

Aligning the marginals may be insufficient. By aligning marginal distributions we hope to restore the joint feature distribution, but this is not guaranteed unless features are independent (and they aren't). Whilst we found the alignment is often sufficient to achieve good performance, there is potential for improvement if features can be encouraged to be independent using disentanglement approaches (Bengio et al., 2013; Burgess et al., 2018; Eastwood and Williams, 2018).

An alignment of the feature space distribution may not be unique. As a general theoretical point, it is not always possible to guarantee that aligning distributions gives the individual sample alignment that we aim for. Two 2D standard Gaussian distributions can be aligned in infinitely many ways due to to rotational symmetry. Whilst we cannot provide theoretical guarantees that the alignment we learn is the optimal one, the pairwise metric we use for quantifying the degree of feature restoration demonstrates that, in practice, aligning the distributions does achieve better per sample alignment than competing methods.

Feature distributions may change in the target domain. Similarly, our aim with feature restoration is to restore the behaviour of feature detectors (units) in new domains. However, it may be that restoring this behaviour does not mean total marginal distribution alignment if the frequency with which a feature appears changes between source and target domains. For example. if a feature detects eyes in images and a new domain contains fewer faces that the source domain, we may still want to detect eyes even though the distribution has changed. Whilst technically this violates our domain shift assumptions in that the distribution of L changes (Figure 4.1a), this type

of dataset shift is not uncommon. This presents an opportunity for further exploration of other, non alignment based, methods to restore the behaviour of feature detectors across changing domains.

Model selection. Like other works in UDA and SFDA we used a target domain validation set to select model hyperparameters. In a real SFDA situation there would be no such validation set available. This flaw in these methods has been gaining attention with novel validation procedures (You et al., 2019) and new realistic benchmarks (Gulrajani and Lopez-Paz, 2021) being proposed.

4.7 Summary

In this chapter we investigated feature restoration as an approach to resolving domain shift, specifically measurement shifts. By aligning marginal feature activation distributions we restored model performance after a change in domain by encouraging the same features to be extracted on the target domain as were learned on the source domain. We showed that our approach improved adapted model behaviour in the target domain, particularly regarding improved calibration, and analysed the behaviour of adapted models to gain a better understanding of why this approach works.

So far, in Chapter 3 we selected which parameters to train to model changes in style (e.g. choosing to train style specific residual adapters) and in this chapter we used a bottom-up inductive bias to improve performance, selecting to primarily update parameters in the early layers of feature extractors. In the final chapter we will explore further how we might go about choosing which parameters to train as data changes domains.

Chapter 5

Novel Approaches with Unit-level Surprise

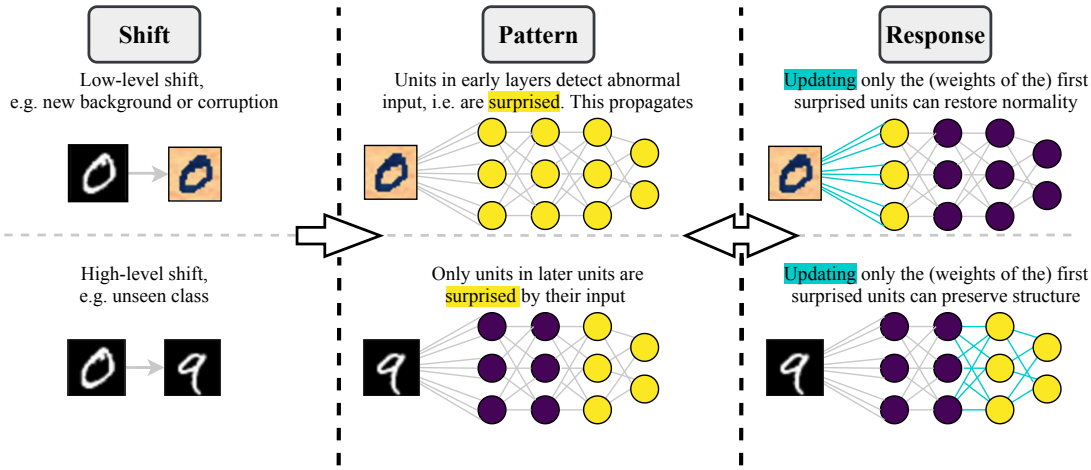


Figure 5.1: Patterns of unit-level surprise can be used to choose which few parameters to update based on a shift in data distribution. The update response may change the pattern which may in turn change the parameters to be updated by the response. Purple units are unsurprised, yellow units surprised, blue indicates the weights to update.

In Chapter 3, motivated by model capacity tradeoffs (Goodfellow et al., 2016, Section 5.2), we saw how assigning a small subset of network parameters to model a shift in style allowed us to improve performance in the low data regime. Similarly in Chapter 4, motivated by maintaining knowledge through maintaining structure, we saw how adapting some parameters much more than others is beneficial when data is limited (specifically training bottom-up). That is, we have been working with the idea that

good few-shot adaptation is achieved by updating some relatively small proportion of model parameters.

However, whilst so far we have presented different methods for adaptation, we have not considered the general question of *which few parameters are best to update in order to adapt quickly*. This question of which parameters to update to reduce some global error signal is a *credit assignment problem* - "who is responsible for this?" For neural networks most commonly credit assignment is achieved using backpropagation (Rumelhart et al., 1986) and then parameters are updated with stochastic gradient descent.

In this chapter we examine the general question of which parameters are best to update in a neural network depending on how the domain changes. We begin with an initial analysis of how changes in unit-level activation distributions can be used to create patterns of parameters that align with intuitions about who should update. We then use this to create shift dependent learning behaviour and give an extensive reflection on issues arising with this approach as motivation for next steps.

We consider specifically few-shot adaptation under domain shift as discussed in Chapter 2 (Figure 2.1c) where both predictor data points and target labels are given. Figure 5.1 gives an overview of our approach to this type of problem. In particular a model is trained on some data which then experiences some domain *shift* (E changes in Figure 2.1c), from this change we aim to detect which neurons are most 'surprised' by this change and examine the *patterns* these neurons form. Finally, based on how the patterns form, we develop and perform some *response* that selects a subset of parameters to update in order to adapt to the changing data.

5.1 Related Work

The desire to create systems that can rapidly adapt on out of distribution data has led to the development of many different techniques. As discussed, fast low data adaptation is often achieved by changing a small number of model parameters such as the batch normalisation statistics and parameters (Li et al., 2017a; Ishii and Sugiyama, 2021; Wang et al., 2021). The related work we discuss focuses on two ways we might be able to improve this adaptation, namely by learning or adapting neural network modules or by taking inspiration from how biological systems solve these problems.

5.1.1 Modularity, Modules, and Sparsely Changing Mechanisms

One approach to improving the generalisability of machine models as well as their ability to maintain knowledge and adapt quickly has been to take a causal view (Schölkopf et al., 2012). Relevant to our work is the assumption that observed variables are created by underlying modular and reusable causal mechanisms (Parascandolo et al., 2018). Moreover, that real world changes in data distribution tend be caused by sparse changes in these mechanisms (Bengio et al., 2020; Goyal et al., 2021). This is the Sparse Mechanism Shift hypothesis, defined by Schölkopf et al. (2021) as: "Small distribution changes tend to manifest themselves in a sparse or local way in the causal/disentangled factorization i.e., they should usually not affect all factors simultaneously". The key insight of this assumption is that if networks can be learned such that they themselves capture these mechanisms in a modular way, then, given some small distribution change, only a small subset of network parameters should need to update allowing for fast few-shot adaptation and continual learning.

Outside of causality there has been substantial interest in investigating to what extent neural networks are modular (Hod et al., 2021; Csordás et al., 2021), for interpretability (Filan et al., 2021), and for improved systematic generalisation (Andreas et al., 2016; D'Amario et al., 2021). The lottery ticket hypothesis (Frankle and Carbin, 2019; Chen et al., 2021) suggests that certain sparse subsets of neurons (i.e. modules) are the most important due to their initialisation. These can be trained in isolation to achieve close to the full neural network's performance. However, as the winning lottery tickets are determined by random initialisation, there is little reason to believe that these subsets will align with mechanism shifts in the real world.

Other methods such as FlexTune (Royer and Lampert, 2020) select different subsets of network parameters, specifically different layers or blocks, to train depending on how a dataset shifts. However, this approach requires searching over all layers or blocks of a network which is neither scalable nor able to automatically select which parameters to update based on the shift.

5.1.2 Biological Inspirations

Synaptic plasticity (Hebb, 1949; Citri and Malenka, 2008) refers to the ability of synapses in the brain to change their synaptic strength over time. Drawing the loose analogy between artificial and biological neurons this is analogous to the ability to

change connection strengths (weights) between artificial neurons (as happens during learning/training). Metaplasticity refers to the plasticity of synaptic plasticity (Abraham and Bear, 1996; Abraham, 2008), that is, changing how plastic a neuron is i.e. how much or how fast the synaptic strength can be modified. This allows adaptive learning behaviour (Hulme et al., 2013; Wang and Tao, 2009). An example of analogous metaplasticity-like behaviour in artificial networks would be changing the learning rates for specific neurons. Using this inspiration, if we are able to identify specific units (ideally a small proportion of all units) that are most relevant for a given domain shift, we can set these units to be more plastic (e.g. training these units more using a higher learning rate) when adapting to the shift. In the brain metaplasticity is affected by neuromodulators, that is, chemicals that regulate behaviour of populations of neurons (Katz, 1999).

5.2 Unit-level Surprise

The approach we take in order to select which few parameters to update is based on the intuition that if, after some domain shift, a unit sees features 'similar' to those seen during pretraining then there is no need for it to update. This is somewhat similar to the feature restoration idea from Chapter 4 where we aimed to update units in order to restore the 'same' features after adaptation. However, here we ask which units have 'good' behaviour *before* adapting and use this to select which units to update. In order to do this we focus specifically on the unit-level, that is, examining each unit individually rather than grouped together as layers or modules.

For each unit we calculate the (Bayesian) surprise (Itti and Baldi, 2009) by measuring the difference between its activation distributions on source and target domains. This quantity is designed to be high when the unit sees unusual activations (different from those seen on the source domain) and low when it seems activations similar to those seen during pretraining on the source domain. Henceforth we refer to this simply as the unit's surprise, which is a somewhat overloaded term examined further in Appendix C.1.

To calculate the surprise for a unit we store its $1D$ activation distribution under the training data, $P(A)$, and under the shifted data distribution, $Q(A)$. We parameterise these distributions exactly as in Section 4.4 using softly-binned histograms from which we get 10 normalised bin counts for each unit, $\pi^p_1, \ldots \pi^p_{10}$ for $P(A)$, and $\pi^q_1, \ldots \pi^q_{10}$ for

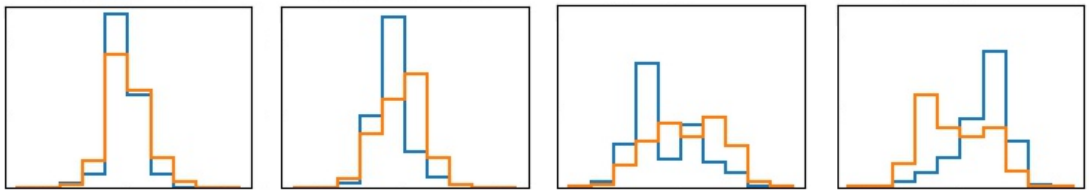


Figure 5.2: Examples of our histogram parameterisations of $P(A)$, in blue, and $Q(A)$, in orange. When new data is received, the activation distribution changes, the more different the distributions the higher the surprise. From left to right, the surprise values s(A) are approximately 0.1, 0.2, 0.3, 0.4.

$Q(A)$.[1][2] From these distributions we calculate the surprise,

$$s(A) = D_{KL}\left(Q(A;\{\pi_i^q\}_{i=1}^{10}) \,||\, P(A;\{\pi_i^p\}_{i=1}^{10})\right) = \sum_{i=1}^{10} \pi_i^q \log \frac{\pi_i^q}{\pi_i^p}. \tag{5.1}$$

This KL divergence measures the difference between $P(A)$ and $Q(A)$, it is 0 when the distributions are identical and is large when they are very different. In particular we can view this as a measure of how surprising the new activations are under a domain shift for a specific unit. Figure 5.2 gives examples of distributions $P(A)$ and $Q(A)$ for different units in a neural network and approximate surprise values.

5.2.1 Controlled Experimental Setup

To allow to us to understand the behaviour of unit-level surprise we use a simple and controlled experimental set up. We train a 5 layer neural network with 3 convolutional and 2 fully connected layers whose architecture is the same as shown in Table 4.3 (without batch normalisation layers). Initially this network is pretrained with 37 of the 47 classes in (standard) EMNIST (Cohen et al., 2017), after which we choose 1 of 10 possible data distribution shifts from which we calculate the unit-level surprise for all units in the network and from this design a response to adapt. Specifically we use seven 'low-level' shifts from EMNIST-DA (Section 4.3) where we expect the early layers of the network to need to update, and three 'high-level' shifts, where we expect it to be preferable to update the later layers. We make use of dropout during this initial

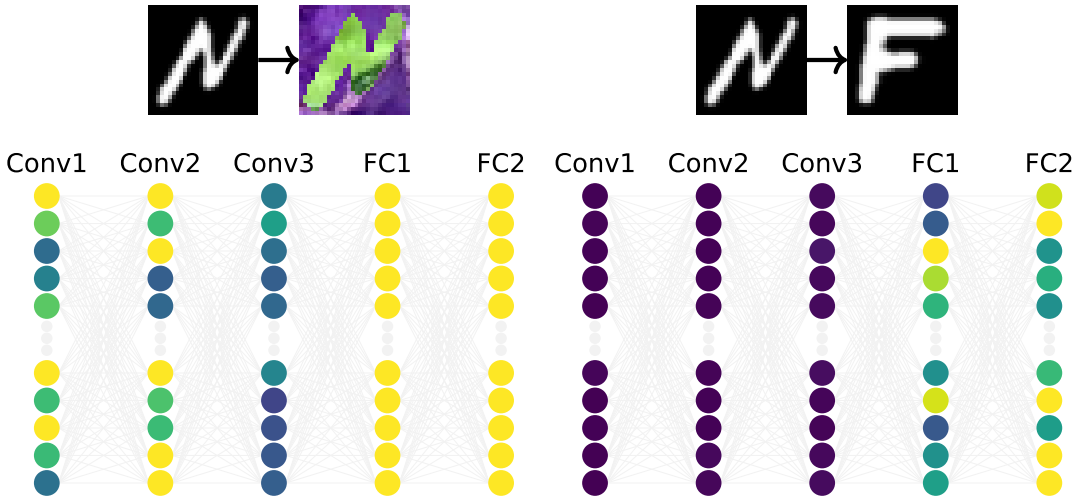
[1]Note that $P(A)$ is calculated once by running one forwards pass of the network after pretraining, whereas $Q(A)$ is calculated with each batch and changes during adaptation.

[2]For convolutions, $P(A)$ and $Q(A)$ are estimated using each spatial location of a feature map as one sample of the activation, a.

pretraining but not during adaptation as it complicates the propagation of surprise and has limited effect on the results.

The seven shifts we select from EMNIST-DA are crystals, fog, Gaussian blur, grass, impulse noise, sky and stripe. These are selected as they strongly affect the zero-shot accuracy and intuitively can be resolved by adapting the early convolutional filters (which can be roughly thought of as edge detectors). To create the high-level shifts we use new classes, unseen during training. After pretraining on the first thirty seven EMNIST classes we then choose 5 of the 10 unseen classes three times from the range $[38,47]$ to arrive at three high-level shifts: H1:$[38,39,40,41,42]$, H2:$[43,44,45,46,47]$, and H3:$[38,40,42,44,46]$.

5.2.2 Analysing Surprise Patterns



(a) Low-level shift (crystals background) (b) High-level shift (unseen classes)

Figure 5.3: Patterns of surprise for different changes in data distribution. Yellow units are highly surprised and dark blue units are unsurprised. (a) For low-level shifts surprise is noticed in the early layers and this effect propagates through the rest of the network. (b) For high-level shifts early layers are unsurprised (similar features) whereas the later fully connected layers show surprise.

Armed with this controlled setup and the calculation of unit-level surprise, described above, we can calculate the surprise values for each unit in our network and visualise this as in Figure 5.3. The key observation of this process is that different shifts in the data cause different patterns of surprise, in particular not all shifts are noticed at the same level of abstraction which in turn motivates our design of an update rule that

handles different shifts at different levels (Section 5.3).

On further analysis we can also offer some explanation of the patterns that arise. In Figure 5.3a after pretraining we expect the early convolutional filters to detect something like black and white edges, then after receiving new data where the edges are now purple and green these filters are surprised by this change. As the early filters have large changes in activation distribution this effect naturally propagates through to the later units. Similarly in Figure 5.3b when an unseen class is received, the black and white edges are still present in the data (giving a similar activation distribution) but when combining the image features in the fully connected layers the network sees some new combination and so surprise is only noticed later in the network.

5.3 Surprise Based Responses

Now that the units of our neural network are equipped with the ability to measure surprise and hence create data dependent effects, how might we use this to better update the network's parameters in the low data regime? We create an update rule where the incoming weights to a unit update only if the unit is sufficiently surprised (above some threshold). However, due to the propagation of surprise effect discussed in Section 5.2 we also add a condition that, in order for a unit to update, the parent units (units connected to the unit in question from the previous layer) must be sufficiently unsurprised (below some threshold). The intuition behind this is a credit assignment approach where a unit asks, *"Am I responsible for this? If the situation is the same as before, no need for me to update,"* and, *"Are there updates happening on the preceding units? In which case, hold tight this is being dealt with."*

To formally define this rule we consider a 'child' unit in layer l with index j and surprise c_j as defined in Equation 5.1, in the previous layer, $l-1$, we consider some 'parent' of this unit with index i and surprise p_i, the parent is connected to the child with some weight w_{ij}.

To get a single value representing the surprise of all parent units we aggregate the surprise of all parents of the the child unit as $\bar{p}_j$, which is a weighted sum of the parent surprises,

$$\bar{p}_j = \sum_i \frac{|w_{ij}|}{\sum_k |w_{kj}|} \cdot p_i \ , \tag{5.2}$$

where we normalise the weight values, $\frac{|w_{ij}|}{\sum_k |w_{kj}|}$, in order to get a comparable scale across all child units in a layer.

This then gives a single value for the (child) unit's own surprise, c_j, and the parent surprises, $\bar{p}_j$. We can then create our update rule by thresholding these values, specifically the update for weight w_{ij} is

$$w_{ij} := w_{ij} - \mathbb{I}[c_j > \alpha]\mathbb{I}[\bar{p}_j < \beta] \cdot \eta \nabla_{w_{ij}} \mathcal{L}, \tag{5.3}$$

where $\mathbb{I}[c_j > \alpha]$ and $\mathbb{I}[\bar{p}_j < \beta]$ are indicator functions that are 1 when the threshold condition is true and 0 otherwise, with α and β set experimentally. η is the learning rate, and $\mathcal{L}$ is the cross-entropy loss function. This hard gating ensures the weight only updates if the child unit is sufficiently surprised and the parent units are not and is diagrammed in Figure 5.4. We note analogies with metaplasticity in that only certain neurons will be trained at any one time based on how surprised they are. Additionally the $\mathbb{I}[\bar{p}_j < \beta]$ term can be linked to neuromodulation, sending a signal that prevents subsequent units from updating.

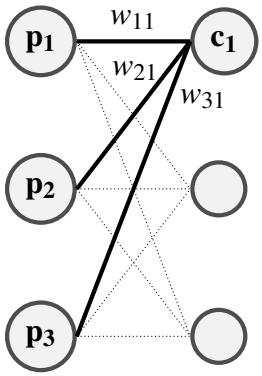


Figure 5.4: Update rule schematic. The thick weight connections update only if the aggregated parent surprise, $\bar{p}_1$, is low enough and the child surprise, c_1, is high enough.

To evaluate, we compare our update rule against two other possible responses to a change in data distribution to get three possible approaches:

1. **SGD.** Use stochastic gradient descent (SGD) to update all parameters.
2. **FlexTune (Royer and Lampert, 2020)**. Fine-tune every individual layer separately and select the best performing layer. This serves as an upper-bound on single layer performance.

Table 5.1: 5-shot accuracy when training: all layers (SGD); single layers using FlexTune (Royer and Lampert, 2020); using our surprise-based update rule. We average over low level, high level and all shifts to get the rows of the table.

	Conv1	Conv2	Conv3	FC1	FC2	SGD	FlexTune	Upd. Rule
Low	82.7 ± 0.4	70.2 ± 1.2	61.4 ± 0.2	56.3 ± 0.2	51.5 ± 0.3	71.3 ± 2.1	82.7 ± 0.4	$\mathbf{82.9 \pm 0.5}$
High	0.0 ± 0.0	0.3 ± 0.5	25.4 ± 3.8	80.6 ± 4.6	$\mathbf{94.5 \pm 0.9}$	74.4 ± 5.2	$\mathbf{94.5 \pm 0.9}$	79.2 ± 4.2
All	57.9 ± 0.3	49.3 ± 0.8	50.6 ± 1.2	63.6 ± 1.2	64.4 ± 0.3	72.2 ± 2.9	$\mathbf{86.3 \pm 0.5}$	81.8 ± 1.2

3. **Update rule based on unit-level surprise.** Use the update rule described above in Equation 5.3, summarised as: update only if you are surprised and your parents are not.

5.3.1 Implementation details

To implement the possible responses described above we pretrain using 2000 samples per class with held out validation and test sets of 400 samples per class each. We then evaluate the sample efficiency of the different responses by testing 2, 5, 10, 20 and 50 samples per class, or *shots*. We also experiment with using all 2000 samples as a baseline.

We optimise our network using stochastic gradient descent with a momentum of 0.9 and a batch size of 256. Initially the network is pretrained for 150 epochs on the identity (black and white) data with a learning rate of 0.01. During adaptation we make use of early stopping training for a maximum of 100 epochs with a patience of 10, stopping based on the network accuracy. Adaptation uses a learning rate of 0.1 except when training all network parameters with SGD, this situation requires a lower learning rate of 0.01 to avoid divergence as SGD updates more parameters than other responses. We find setting α and β in Equation 5.3 to 0.01 to be effective values without being overly specific; whilst one could set α and β differently for each layer and dataset shift the search space grows very quickly and it becomes easy to overfit these hyperparameters to a specific set up.

The full per-shift results of these experiments are given in Appendix C.2 where each experiment is run over 3 seeds to report a mean and standard deviation.

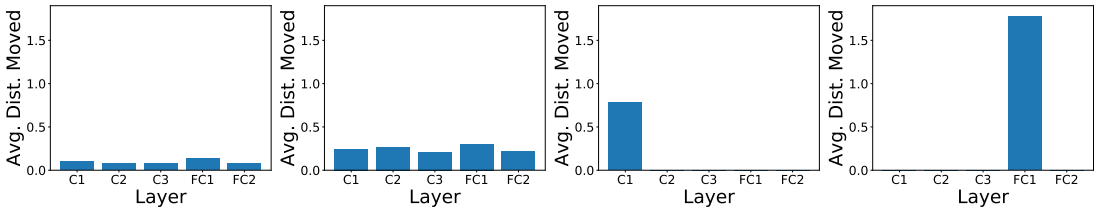
5.3.2 Analysing Different Responses

We now examine the quantitative performance of the different responses, firstly in the 5-shot setting with results given in Table 5.1. Looking at the first five columns which train individual layers we see clearly that low level shifts are best dealt with by adapting the early layers and high level shifts are best dealt with in the later layers. That is, like Royer and Lampert (2020) we find that fine-tuning the last layer is not always the optimal thing to do. What's more, this aligns with both the intuitions and the surprise patterns given in Section 5.2 (and Figure 5.3).

Looking at the last three columns of Table 5.1 which show each of the responses we consider, we see that FlexTune performs the best on average (86.3%). However, this method is clearly biologically implausible and scales extremely poorly with network depth; if we do not know in advance what type of shift will occur[3], one must train every layer individually and decide which is the best layer to use based on a validation set accuracy. So if we do not have a priori knowledge of the type of shift that will occur and do not want to search over every possibility using an oracle-like validation set our update rule will give the best performance (81.8%). That is, our surprise based rule can *automatically select* which *few parameters* to update in a *data dependent manner*.

To analyse why our update rule performs much better than SGD we visualise how much the parameters are moved by both SGD and our update rule. We calculate the Euclidean distance moved between parameter values before and after adaptation and average these distances over each layer to create the plots in Figure 5.5. Figures 5.5a and 5.5b show that SGD moves parameters in all layers approximately equal distances and changes very little between different shift types. This underlines that standard backpropagation is not a sufficiently effective credit assignment technique for selecting which few parameters to update for a specific shift. On the other hand, Figures 5.5c and 5.5d show that our method exhibits clear data dependent behaviour, selecting specific parameters to update which align with the preferred FlexTune layers for adapting (updating the early convolutional layers for low level shifts and the fully connected layers for high level shifts) . Whilst parameters are moved different amounts by different methods, from Table 5.1 it seems to be the case that selecting which few parameters to update is the more important behaviour over how far the parameters are moved.

[3]If we know in advance what type of shift will occur we can select to train the first convolutional layer for low level shifts and the last fully connected layer for high level shifts.



(a) Low, SGD (b) High, SGD (c) Low, Upd. Rule (d) High, Upd. Rule

Figure 5.5: Mean Euclidean distances moved by units in each layer. (a) using SGD for a low level shift (crystals background shift), (b) using SGD for a high level shift (H1, 5 unseen classes), (c) our update rule for the same low level shift and (d) our update rule for the same high level shift.

Table 5.2: Accuracy for different numbers of data shots. Results averaged over all shifts.

	2	5	10	20	50	2000
SGD	60.0 ± 1.9	72.2 ± 2.9	78.9 ± 0.5	82.6 ± 0.2	86.1 ± 0.3	92.1 ± 0.1
FlexTune	$\mathbf{82.6 \pm 0.7}$	$\mathbf{86.3 \pm 0.5}$	$\mathbf{87.8 \pm 0.4}$	$\mathbf{89.2 \pm 0.2}$	$\mathbf{90.4 \pm 0.2}$	$\mathbf{92.5 \pm 0.0}$
Upd. Rule	66.3 ± 2.5	81.8 ± 1.2	86.2 ± 0.6	88.2 ± 0.4	89.7 ± 0.1	92.4 ± 0.0

To complete our understanding, we examine the results for different numbers of shots (samples per class) in Table 5.2. Here we see that when we have all the data (2000 shots) the importance of which parameters to update fades away with all methods performing similarly, this is unsurprising as SGD is known to work well given sufficient data. However, as the number of shots reduces the importance of selecting the correct few parameters to update increases, with both our method and FlexTune significantly outperforming SGD for 5-50 shots. In the 2-shot scenario FlexTune is by far the best, Table C.2 (Appendix C.2) shows this is largely because our update rule is outperformed on high level shifts where it tends to update the penultimate layer (Figure 5.5d) rather than the last layer. This again shows that selecting which few parameters to update really matters when data is limited.

5.4 Challenges Arising and Directions for Further Investigation

We have seen that using unit-level surprise to update a small number of parameters, dependent on how data shifts, is able to provide large improvements for few-shot per-

formance. However, we have also seen that our method is not necessarily optimal in which parameters it selects. In this section we reflect on the main challenges encountered during the development of these techniques and difficulties in extending them to more complex scenarios.

5.4.1 Reflections on Network Modularity

Our initial intent with unit-level surprise was to discover different modules[4] that would adapt to different changes in the data (Hod et al., 2021). We find that with our set up such modules do not arise, with low level shifts surprising almost all units in the network (Fig. 5.3a) and high level shifts surprising almost all units in fully connected layers (Fig. 5.3b). This property additionally causes our update rule to update entire single layers rather than a more modular and more sparse set of units.

Upon reflection, it seems clear that expecting modules to arise that also align with the unseen domain shifts in our dataset is asking too much. For example, when pretraining only on black and white characters, why would modules emerge that separate colour from other features (as would be needed for background shifts such as *crystals* and *sky*)? Indeed, this leads us to ask if it is ever possible to learn an appropriate modularisation without explicitly optimising for this property while pretraining over datasets containing multiple different shifts/environments.

To learn modules that align with changes in data distribution we could look for inspiration in the ideas of sparsely changing causal mechanisms. Indeed, Bengio et al. (2020) explicitly use these ideas to motivate a meta-learning approach for training neural networks which optimise for adaptation speed as a proxy for few parameters needing to update. This presents a way in which unit-level surprise, or some similar quantity, could be used to learn more modular networks - by using the number[5] of surprised units as an alternate proxy score for correctly modularising knowledge that is a more direct measure of the number of parameters needing to be updated.

[4] We consider ideal modules to be sparse subsets of units that are not necessarily restricted to single layers and perform some subtask relevant to the overall task (in our case character classification).

[5] A continuous proxy can be used for differentiability.

5.4.2 Reflections on Unit-Level Surprise

To our minds, the most interesting part of our approach is the general use of unit-level information for improving credit assignment. The specificities of our surprise calculation and update rule design are less important and we now consider alternative methods that could be investigated.

5.4.2.1 Alternative Unit-Level Information

The fact that our surprise-based update rule does not always update the 'best' units (Table 5.1) motivates an exploration of alternative unit-level information or different measures of surprise. For example, we could use parameter uncertainty and select to optimise the parameters which have highest uncertainty (Jospin et al., 2020; Aitchison et al., 2021). Alternatively, if working in a continual learning situation, we could use task importance (Kirkpatrick et al., 2017; Zenke et al., 2017) to help select which parameters to update.

Many of the possible pitfalls when using marginal feature activation distributions, discussed in Section 4.6, remain when using unit-level surprise. In particular, changes in the frequencies with which certain features appear in source and target domains may cause changes in the marginal activation distributions that should not cause parameter updates. Alternative ways of measuring unit-level surprise that can account for some of these situations could help us to design a more general approach. For example, sample-wise calculations of surprisal, or a measure of surprise that allows for surprise decreases as well as surprise increases.

5.4.2.2 Making Better Use of Unit-Level Surprise

Figure 5.5 shows that our method tends to update parameters from a single network layer. Since we do not explicitly minimise unit-level surprise, surprised units tend to stay surprised during training meaning aggregated parent surprise never gets sufficiently small to allow child units to update. One possible solution to this problem would be to gradually update the source domain marginal distribution, $P(A)$, with samples from the target domain marginal distribution, $Q(A)$, for units that update. This allows that if a domain shift occurs and we receive large amounts of data in the new domain, units that update using our update rule (Equation 5.3) will gradually become unsurprised allowing later units to update.

One challenge with these hand designed tweaks to calculating and learning surprise is that designing only necessary changes in the approach is difficult. This is because we do not know (without a combinatorial search) which parameters are optimal to update in any given situation, meaning we can only compare approaches empirically (i.e. using few-shot accuracy rather than hypothesising new approaches based on evidence). A better understanding of the roles that individual units play (Olah et al., 2018; Goh et al., 2021) may also provides us with inspiration and intuition about which units should update. Another solution is to try and avoid hand designing update rules entirely. In particular, meta-learning optimisers that make use of unit-level information (Andrychowicz et al., 2016; Ravi and Larochelle, 2017).

5.5 Summary

The methods presented in this thesis have largely handled few-shot data by updating some few model parameters. In this chapter we turned to the question of *which* few parameters we should update when an unknown dataset shift occurs. We examined the use of unit-level information for this task, in particular unit-level surprise, measuring differences in source and target domain marginal activation distributions. We showed that patterns of unit-level surprise often align with our intuitions about which units should update and can detect changes in data distribution at different levels of abstraction. We proposed a simple surprise based update rule that provided more granular credit assignment than standard backpropagation allowing us to update fewer parameters to achieve better few-shot performance.

Chapter 6

Conclusion

This thesis has examined various aspects of neural network behaviour under changing domains. Initially, we demonstrated the practical value of modelling new domains (or styles) when only limited data is available for animation in computer graphics. We then explored specific domain shifts that can cause drastic reductions in the performance of a trained model and proposed a model-based approach for recovering original model behaviours by restoring features. We additionally investigated which parameters are most affected by a change in data distribution, and used this to analyse different credit assignment approaches and create dataset shift dependent adaptation.

Motivated by specific data generating assumptions (Chapter 2), all of our investigations have worked to alter learned latent features at varying levels of abstraction. The residual adapters and FiLM parameters of Chapter 3 create style specific behaviour by modulating latent style agnostic features learned by training over multiple styles. In Chapter 4, our feature restoration approach explicitly aligns marginal distributions of the latent features extracted by source and target domain feature extractors. We directly measure changes in latent feature distributions to select which parameters to update when using unit-level surprise in Chapter 5.

The alignment of feature distributions, the minimisation of the entropy of representations/predictions, and the reduction of surprise, are all examples of general heuristics that allow well initialised models to learn without the need for large amounts of new labelled data. We believe that the use of such heuristics will be required for the development of better artificially intelligent models that can learn continually and rapidly with limited supervisory input. It is our hope that our contributions on the applica-

tion, implementation and analysis of these heuristics will help with some small part of the journey towards the better understanding, and potentially creation, of intelligent systems.

6.1 Future Work

The ways in which we have been thinking about changing domains are also applicable to several other open problems in artificial intelligence and machine learning. To conclude the thesis, we link together these areas and suggest ways in which we might wish to think about them.

Using changing domains to understand the world. In Chapter 3, we used a data-driven definition for separating latent data generating variables, along with the availability of data in multiple domains (styles), to learn style/domain agnostic representations of the data. In Chapter 5 we discussed work on sparse mechanism shifts (Schölkopf et al., 2021) and again found the necessity for training over multiple domains to learn domain invariant representations (Arjovsky et al., 2019; Geirhos et al., 2020). This raises the broader question of can we use changing domains (environments) to extract some (potentially causal) representation of the world? An embodied agent acting in the real world experiences constant gradual environmental changes, e.g. changes in lighting, parallax/viewpoint and temperature. In particular, can we use these changes to extract invariant representations that represent useful concepts for adaptively acting in the world (learning L in Figure 2.1c)?

Continual and lifelong learning If we are to make use of such constantly changing domains we must create systems that can learn continually. In particular we both wish to learn good lifelong shared representations, and be able to modulate or utilise the representations for specific tasks by learning in an online manner. The learning of good representations that can be shared across domains is discussed in the previous paragraph, but choosing which representations to maintain, which to update, and which to use for a given task requires further methods and analysis.

The role of scale. Indubitably some of the most impressive machine learning results come from systems that have been designed to leverage compute at immense scale (Devlin et al., 2019; Brown et al., 2020; Radford et al., 2021). Learning at scale, particularly with multi-modal/multi-domain data (Ramesh et al., 2021) (Radford et al., 2021), is able to create remarkable behaviours and is beginning to show improvements

on out of distribution datasets (Geirhos et al., 2021). To what extent the intuitions and analyses explored in this thesis are relevant for such large scale models, and how to go about adapting these approaches to work with these systems, is an open question. On the other hand, perhaps Sutton's bitter lesson, that methods that leverage compute will outstrip carefully designed methods that leverage human knowledge and insight, will continue to hold water (Sutton, 2019).

How should we perform credit assignment in neural networks? Almost all of the neural network works cited in this thesis train their networks using backpropagation of some global error signal, along with some optimiser. This type of credit assignment (Minsky, 1961), as it applies to neural networks parameters, is not modular (Caporale and Dan, 2008) and thought to be biologically implausible (Lillicrap et al., 2020). Automatically selecting which parameters to update dependent on the task and the data, in order to update learned representations efficiently whilst maintaining as much relevant information as possible, may be achievable with backpropagation or may require new ideas. For example, the creation of backpropagation alternatives (Lansdell et al., 2020; Millidge et al., 2020) or the learning of local update rules (Löwe et al., 2019; Gregor, 2020).

Bibliography

Aberman, K., Weng, Y., Lischinski, D., Cohen-Or, D., and Chen, B. (2020). Unpaired motion style transfer from video to animation. *ACM Transactions on Graphics (TOG)*, 39(4):64.

Abraham, W. C. (2008). Metaplasticity: tuning synapses and networks for plasticity. *Nature Reviews Neuroscience*, 9(5):387–387.

Abraham, W. C. and Bear, M. F. (1996). Metaplasticity: the plasticity of synaptic plasticity. *Trends in Neurosciences*, 19(4):126–130.

Adobe (2015). Adobe's mixamo. `https://www.mixamo.com`. Accessed: 2020-10-04.

Aitchison, L., Jegminat, J., Menendez, J. A., Pfister, J.-P., Pouget, A., and Latham, P. E. (2021). Synaptic plasticity as bayesian inference. *Nature Neuroscience*, 24(4):565–571.

Andreas, J., Rohrbach, M., Darrell, T., and Klein, D. (2016). Neural module networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 39–48.

Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M. W., Pfau, D., Schaul, T., Shillingford, B., and De Freitas, N. (2016). Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, pages 3981–3989.

Arbelaez, P., Maire, M., Fowlkes, C., and Malik, J. (2011). Contour detection and hierarchical image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(5):898–916.

Arik, S., Chen, J., Peng, K., Ping, W., and Zhou, Y. (2018). Neural voice cloning with a few samples. In *Advances in Neural Information Processing Systems*, volume 31.

Arjovsky, M., Bottou, L., Gulrajani, I., and Lopez-Paz, D. (2019). Invariant risk minimization. *arXiv preprint arXiv:1907.02893*.

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

Bandi, P., Geessink, O., Manson, Q., Van Dijk, M., Balkenhol, M., Hermsen, M., Bejnordi, B. E., Lee, B., Paeng, K., Zhong, A., et al. (2018). From detection of individual metastases to classification of lymph node status at the patient level: the camelyon17 challenge. *IEEE Transactions on Medical Imaging*, 38(2):550–560.

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

Ben-David, S., Blitzer, J., Crammer, K., and Pereira, F. (2007). Analysis of representations for domain adaptation. In *Advances in Neural Information Processing Systems*, pages 137–144.

Ben-David, S., Lu, T., Luu, T., and Pál, D. (2010b). Impossibility theorems for domain adaptation. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 129–136.

Benaim, S. and Wolf, L. (2018). One-shot unsupervised cross domain translation. In *Advances in Neural Information Processing Systems*, volume 31.

Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. (2015). Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, volume 28.

Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828.

Bengio, Y., Deleu, T., Rahaman, N., Ke, N. R., Lachapelle, S., Bilaniuk, O., Goyal, A., and Pal, C. (2020). A meta-transfer objective for learning to disentangle causal mechanisms. In *International Conference on Learning Representations*.

Bird, A. (2020). *Multi-task dynamical systems: customising time series models*. PhD thesis, The University of Edinburgh.

Bird, A. and Williams, C. K. I. (2019). Customizing sequence generation with multi-task dynamical systems. *arXiv preprint arXiv:1910.05026*.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33.

Burgess, C. P., Higgins, I., Pal, A., Matthey, L., Watters, N., Desjardins, G., and Lerchner, A. (2018). Understanding disentangling in β-VAE. In *International Conference on Learning Representations*.

Bütepage, J., Black, M., Kragic, D., and Kjellström, H. (2017). Deep representation learning for human motion prediction and classification. *arXiv preprint arXiv:1702.07486*.

Caelles, S., Maninis, K.-K., Pont-Tuset, J., Leal-Taixé, L., Cremers, D., and Van Gool, L. (2017). One-shot video object segmentation. In *Proceedings of The IEEE Conference on Computer Vision and Pattern Recognition*, pages 221–230.

Caporale, N. and Dan, Y. (2008). Spike timing–dependent plasticity: A hebbian learning rule. *Annu. Rev. Neurosci.*, 31:25–46.

Chen, T., Frankle, J., Chang, S., Liu, S., Zhang, Y., Carbin, M., and Wang, Z. (2021). The lottery tickets hypothesis for supervised and self-supervised pre-training in computer vision models. In *Proceedings of The IEEE Conference on Computer Vision and Pattern Recognition*, pages 16306–16316.

Chidlovskii, B., Clinchant, S., and Csurka, G. (2016). Domain adaptation in the absence of source domain data. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 451–460.

Citri, A. and Malenka, R. C. (2008). Synaptic plasticity: multiple forms, functions, and mechanisms. *Neuropsychopharmacology*, 33(1):18–41.

Clavet, S. (2016). Motion matching and the road to next-gen animation. In *GDC*.

Clevert, D.-A., Unterthiner, T., and Hochreiter, S. (2015). Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*.

Cohen, G., Afshar, S., Tapson, J., and van Schaik, A. (2017). EMNIST: an extension of MNIST to handwritten letters. *arXiv preprint arXiv:1702.05373*.

Courty, N., Flamary, R., Habrard, A., and Rakotomamonjy, A. (2017). Joint distribution optimal transportation for domain adaptation. *Advances in Neural Information Processing Systems*, 30.

Csordás, R., van Steenkiste, S., and Schmidhuber, J. (2021). Are neural nets modular? inspecting functional modularity through differentiable weight masks. In *International Conference on Learning Representations*.

D'Amario, V., Sasaki, T., and Boix, X. (2021). How modular should neural module networks be for systematic generalization? In *Advances in Neural Information Processing Systems*.

DeGroot, M. H. and Fienberg, S. E. (1983). The comparison and evaluation of forecasters. *Journal of the Royal Statistical Society: Series D (The Statistician)*, 32(1-2):12–22.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). 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)*, pages 4171–4186.

Djolonga, J., Yung, J., Tschannen, M., Romijnders, R., Beyer, L., Kolesnikov, A., Puigcerver, J., Minderer, M., D'Amour, A., Moldovan, D., et al. (2021). On robustness and transferability of convolutional neural networks. In *Proceedings of The IEEE Conference on Computer Vision and Pattern Recognition*, pages 16458–16468.

Dong, Y., Aristidou, A., Shamir, A., Mahler, M., and Jain, E. (2020). Adult2child: Motion style transfer using cyclegans. In *Motion, Interaction and Games*, MIG '20.

Dougherty, J., Kohavi, R., and Sahami, M. (1995). Supervised and unsupervised discretization of continuous features. In *International Conference on Machine Learning*, pages 194–202.

Du, H., Herrmann, E., Sprenger, J., Fischer, K., and Slusallek, P. (2019). Stylistic locomotion modeling and synthesis using variational generative models. In *Motion, Interaction and Games*, MIG '19.

Duchi, J. (2007). Derivations for linear algebra and optimization. `https://web.stanford.edu/~jduchi/projects/general_notes.pdf`. Accessed: 5th October 2021.

Dumoulin, V., Shlens, J., and Kudlur, M. (2017). A learned representation for artistic style. In *International Conference on Learning Representations*.

Eastwood, C., Mason, I., and Williams, C. K. I. (2021). Unit-level surprise in neural networks. In *NeurIPS 2021 Workshop, I (Still) Can't Believe It's Not Better*.

Eastwood, C., Mason, I., Williams, C. K. I., and Schölkopf, B. (2022). Source-free adaptation to measurement shift via bottom-up feature restoration. In *International Conference on Learning Representations*.

Eastwood, C. and Williams, C. K. (2018). A framework for the quantitative evaluation of disentangled representations. In *International Conference on Learning Representations*.

Eslami, S. M. A., Rezende, D. J., Besse, F., Viola, F., Morcos, A. S., Garnelo, M., Ruderman, A., Rusu, A. A., Danihelka, I., Gregor, K., Reichert, D. P., Buesing, L., Weber, T., Vinyals, O., Rosenbaum, D., Rabinowitz, N., King, H., Hillier, C., Botvinick, M., Wierstra, D., Kavukcuoglu, K., and Hassabis, D. (2018). Neural scene representation and rendering. *Science*, 360(6394):1204–1210.

Fei-Fei, L., Fergus, R., and Perona, P. (2006). One-shot learning of object categories. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(4):594–611.

Filan, D., Casper, S., Hod, S., Wild, C., Critch, A., and Russell, S. (2021). Clusterability in neural networks. *arXiv preprint arXiv:2103.03386*.

Finlayson, S. G. (2020). *Learning inductive representations of biomedical data*. PhD thesis, Harvard University Graduate School of Arts and Sciences.

Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135.

Fragkiadaki, K., Levine, S., Felsen, P., and Malik, J. (2015). Recurrent network mod-

els for human dynamics. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4346–4354.

Frankle, J. and Carbin, M. (2019). The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*.

Fussell, L., Bergamin, K., and Holden, D. (2021). Supertrack: motion tracking for physically simulated characters using supervised learning. *ACM Transactions on Graphics (TOG)*, 40:1–13.

Ganin, Y. and Lempitsky, V. (2015). Unsupervised domain adaptation by backpropagation. In *International Conference on Machine Learning*, pages 1180–1189.

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

Gatys, L. A., Ecker, A. S., and Bethge, M. (2015). A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*.

Geirhos, R., Jacobsen, J.-H., Michaelis, C., Zemel, R., Brendel, W., Bethge, M., and Wichmann, F. A. (2020). Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2:665–673.

Geirhos, R., Medina Temme, C. R., Rauber, J., Schütt, H. H., Bethge, M., and Wichmann, F. A. (2019a). Generalisation in humans and deep neural networks. In *Advances in Neural Information Processing Systems*, pages 7549–7561.

Geirhos, R., Narayanappa, K., Mitzkus, B., Thieringer, T., Bethge, M., Wichmann, F. A., and Brendel, W. (2021). Partial success in closing the gap between human and machine vision. In *Advances in Neural Information Processing Systems*.

Geirhos, R., Rubisch, P., Michaelis, C., Bethge, M., Wichmann, F. A., and Brendel, W. (2019b). Imagenet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. In *International Conference on Learning Representations*.

Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587.

Goh, G., †, N. C., †, C. V., Carter, S., Petrov, M., Schubert, L., Radford, A., and Olah, C. (2021). Multimodal neurons in artificial neural networks. *Distill*.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

Goyal, A., Lamb, A., Hoffmann, J., Sodhani, S., Levine, S., Bengio, Y., and Schölkopf, B. (2021). Recurrent independent mechanisms. In *International Conference on Learning Representations*.

Gregor, K. (2020). Finding online neural update rules by learning to remember. *arXiv preprint arXiv:2003.03124*.

Gross, R. and Shi, J. (2001). The cmu motion of body (mobo) database. Technical report.

Guan, H. and Liu, M. (2021). Domain adaptation for medical image analysis: a survey. *IEEE Transactions on Biomedical Engineering*, PP.

Gulrajani, I. and Lopez-Paz, D. (2021). In search of lost domain generalization. In *International Conference on Learning Representations*.

Guo, C., Pleiss, G., Sun, Y., and Weinberger, K. Q. (2017). On calibration of modern neural networks. In *International Conference on Machine Learning*, pages 1321–1330.

Ha, D. and Schmidhuber, J. (2018). World models. *arXiv preprint arXiv:1803.10122*.

Haas, J. K. (2014). A history of the unity game engine.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of The IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778.

Hebb, D. O. (1949). *The organization of behavior: A neuropsychological theory*. Psychology Press.

Hendrycks, D. and Dietterich, T. (2019). Benchmarking neural network robustness to common corruptions and perturbations. In *International Conference on Learning Representations*.

Henter, G. E., Alexanderson, S., and Beskow, J. (2020). MoGlow: Probabilistic and controllable motion synthesis using normalising flows. *ACM Transactions on Graphics (TOG)*, 39(4):236:1–236:14.

Hiratani, N. and Latham, P. E. (2020). Rapid bayesian learning in the mammalian olfactory system. *Nature communications*, 11(1):1–15.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Hod, S., Casper, S., Filan, D., Wild, C., Critch, A., and Russell, S. (2021). Detecting modularity in deep neural networks. *arXiv preprint arXiv:2110.08058*.

Holden, D., Habibie, I., Kusajima, I., and Komura, T. (2017a). Fast neural style transfer for motion data. *IEEE Computer Graphics and Applications*, 37(4):42–49.

Holden, D., Kanoun, O., Perepichka, M., and Popa, T. (2020). Learned motion matching. *ACM Transactions on Graphics (TOG)*, 39(4):53–1.

Holden, D., Komura, T., and Saito, J. (2017b). Phase-functioned neural networks for character control. *ACM Transactions on Graphics (TOG)*, 36(4):42.

Holden, D., Saito, J., and Komura, T. (2016). A deep learning framework for character motion synthesis and editing. *ACM Transactions on Graphics (TOG)*, 35(4):138:1–138:11.

Hou, Y. and Zheng, L. (2020). Source free domain adaptation with image translation. *arXiv preprint arXiv:2008.07514*.

Hsu, E., Pulli, K., and Popović, J. (2005). Style translation for human motion. *ACM Transactions on Graphics (TOG)*, 24(3):1082–1089.

Huang, X. and Belongie, S. (2017). Arbitrary style transfer in real-time with adaptive instance normalization. In *Proceedings of the IEEE International Conference on Computer Vision*.

Huang, X., Liu, M.-Y., Belongie, S., and Kautz, J. (2018). Multimodal unsupervised image-to-image translation. *arXiv preprint arXiv:1804.04732*.

Hulme, S. R., Jones, O. D., and Abraham, W. C. (2013). Emerging roles of metaplasticity in behaviour and disease. *Trends in Neurosciences*, 36(6):353–362.

Ikemoto, L., Arikan, O., and Forsyth, D. (2009). Generalizing motion edits with gaussian processes. *ACM Transactions on Graphics (TOG)*, 28(1).

Ionescu, C., Papava, D., Olaru, V., and Sminchisescu, C. (2014). Human3.6m: Large

scale datasets and predictive methods for 3d human sensing in natural environments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(7):1325–1339.

Ishii, M. and Sugiyama, M. (2021). Source-free domain adaptation via distributional alignment by matching batch normalization statistics. *arXiv preprint arXiv:2101.10842*.

Itti, L. and Baldi, P. (2009). Bayesian surprise attracts human attention. *Vision Research*, 49(10):1295–1306.

Ji, X., Pascanu, R., Hjelm, D., Lakshminarayanan, B., and Vedaldi, A. (2021). Test sample accuracy scales with training sample density in neural networks. *arXiv preprint arXiv:2106.08365*.

Jin, D., Jin, Z., Hu, Z., Vechtomova, O., and Mihalcea, R. (2021). Deep Learning for Text Style Transfer: A Survey. *Computational Linguistics*, pages 1–51.

Jing, Y., Yang, Y., Feng, Z., Ye, J., Yu, Y., and Song, M. (2019). Neural style transfer: A review. *IEEE Transactions on Visualization and Computer Graphics*, 26(11):3365–3385.

Johnson, J., Alahi, A., and Fei-Fei, L. (2016). Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*, pages 694–711.

Jospin, L. V., Buntine, W., Boussaid, F., Laga, H., and Bennamoun, M. (2020). Hands-on bayesian neural networks–a tutorial for deep learning users. *arXiv preprint arXiv:2007.06823*.

Katz, P. S. (1999). *Beyond Neurotransmission: Neuromodulation and its Importance for Information Processing*. Oxford University Press New York:.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D. P. and Dhariwal, P. (2018). Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, volume 31.

Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes. In *International Conference on Learning Representations*.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A.,

Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., and Hadsell, R. (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences of the United States of America*, 114(13):3521–3526.

Koh, P. W., Sagawa, S., Marklund, H., Xie, S. M., Zhang, M., Balsubramani, A., Hu, W., Yasunaga, M., Phillips, R. L., Gao, I., Lee, T., David, E., Stavness, I., Guo, W., Earnshaw, B. A., Haque, I. S., Beery, S., Leskovec, J., Kundaje, A., Pierson, E., Levine, S., Finn, C., and Liang, P. (2021). WILDS: A benchmark of in-the-wild distribution shifts. In *International Conference on Machine Learning*.

Kolda, T. G. and Bader, B. W. (2009). Tensor decompositions and applications. *SIAM Review*, 51(3):455–500.

Kouw, W. M. and Loog, M. (2018). An introduction to domain adaptation and transfer learning. *arXiv preprint arXiv:1812.11806*.

Krause, A., Perona, P., and Gomes, R. (2010). Discriminative clustering by regularized information maximization. In *Advances in Neural Information Processing Systems*, pages 775–783.

Krizhevsky, A. (2009). Learning multiple layers of features from tiny images. Technical report.

Kullback, S. (1959). *Information theory and statistics*. Wiley.

Kundu, J. N., Venkat, N., Babu, R. V., et al. (2020). Universal source-free domain adaptation. In *Proceedings of The IEEE Conference on Computer Vision and Pattern Recognition*, pages 4544–4553.

Kurakin, A., Goodfellow, I., and Bengio, S. (2017). Adversarial examples in the physical world. *ICLR Workshops*.

Lake, B. M., Linzen, T., and Baroni, M. (2019). Human few-shot learning of compositional instructions. In *Proceedings of the 41st Annual Conference of the Cognitive Science Society*.

Lake, B. M., Salakhutdinov, R., and Tenenbaum, J. B. (2015). Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338.

Lake, B. M., Ullman, T. D., Tenenbaum, J. B., and Gershman, S. J. (2017). Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40.

Lansdell, B. J., Prakash, P. R., and Kording, K. P. (2020). Learning to solve the credit assignment problem. In *International Conference on Learning Representations*.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

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

Lee, D.-H. et al. (2013). Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *Workshop on Challenges in Representation Learning, ICML*, volume 3.

Lee, K., Lee, S., and Lee, J. (2018). Interactive character animation by learning multi-objective control. *ACM Transactions on Graphics (TOG)*, 37(6).

Lee, S.-W., Kim, J.-H., Jun, J., Ha, J.-W., and Zhang, B.-T. (2017). Overcoming catastrophic forgetting by incremental moment matching. *Advances in Neural Information Processing Systems*, 30.

Li, R., Jiao, Q., Cao, W., Wong, H.-S., and Wu, S. (2020). Model adaptation: Unsupervised domain adaptation without source data. In *Proceedings of The IEEE Conference on Computer Vision and Pattern Recognition*, pages 9641–9650.

Li, Y., Wang, N., Shi, J., Hou, X., and Liu, J. (2018a). Adaptive batch normalization for practical domain adaptation. *Pattern Recognition*, 80:109–117.

Li, Y., Wang, N., Shi, J., Liu, J., and Hou, X. (2017a). Revisiting batch normalization for practical domain adaptation. In *International Conference on Learning Representations Workshop*.

Li, Z. and Hoiem, D. (2017). Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(12):2935–2947.

Li, Z., Zhou, F., Chen, F., and Li, H. (2017b). Meta-SGD: Learning to learn quickly for few-shot learning. *arXiv preprint arXiv:1707.09835*.

Li, Z., Zhou, Y., Xiao, S., He, C., and Li, H. (2018b). Auto-conditioned lstm network for extended complex human motion synthesis. In *International Conference on Learning Representations*.

Liang, J., Hu, D., and Feng, J. (2020). Do we really need to access the source data?

Source hypothesis transfer for unsupervised domain adaptation. In *International Conference on Machine Learning*, pages 6028–6039.

Lillicrap, T. P., Santoro, A., Marris, L., Akerman, C. J., and Hinton, G. (2020). Backpropagation and the brain. *Nature Reviews Neuroscience*, 21(6):335–346.

Ling, H. Y., Zinno, F., Cheng, G., and van de Panne, M. (2020). Character controllers using motion vaes. *ACM Transactions on Graphics (TOG)*, 39(4).

Liu, C. K., Hertzmann, A., and Popović, Z. (2005). Learning physics-based motion style with nonlinear inverse optimization. *ACM Transactions on Graphics (TOG)*, 24(3):1071–1081.

Liu, Z., Lian, T., Farrell, J., and Wandell, B. A. (2020). Neural network generalization: The impact of camera parameters. *IEEE Access*, 8:10443–10454.

Long, M., Cao, Y., Wang, J., and Jordan, M. (2015). Learning transferable features with deep adaptation networks. In *International Conference on Machine Learning*, pages 97–105.

Long, M., Cao, Z., Wang, J., and Jordan, M. I. (2018). Conditional adversarial domain adaptation. In *Advances in Neural Information Processing Systems*.

Löwe, S., O' Connor, P., and Veeling, B. (2019). Putting an end to end-to-end: Gradient-isolated learning of representations. In *Advances in Neural Information Processing Systems*, volume 32.

MacKay, D. J. (1992). Information-based objective functions for active data selection. *Neural computation*, 4(4):590–604.

Martinez, J., Black, M. J., and Romero, J. (2017). On human motion prediction using recurrent neural networks. In *Proceedings of The IEEE Conference on Computer Vision and Pattern Recognition*.

Mason, I., Starke, S., and Komura, T. (2022). Real-time style modelling of human locomotion via feature-wise transformations and local motion phases. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 5(1).

Mason, I., Starke, S., Zhang, H., Bilen, H., and Komura, T. (2018). Few-shot learning of homogeneous human locomotion styles. *Computer Graphics Forum*, 37(7):143–153.

Merel, J., Tassa, Y., Srinivasan, S., Lemmon, J., Wang, Z., Wayne, G., and Heess, N. (2017). Learning human behaviors from motion capture by adversarial imitation. *arXiv preprint arXiv:1707.02201*.

Michaelis, C., Mitzkus, B., Geirhos, R., Rusak, E., Bringmann, O., Ecker, A. S., Bethge, M., and Brendel, W. (2019). Benchmarking robustness in object detection: Autonomous driving when winter is coming. In *Machine Learning for Autonomous Driving Workshop, NeurIPS 2019*.

Millidge, B., Tschantz, A., and Buckley, C. L. (2020). Predictive coding approximates backprop along arbitrary computation graphs. *arXiv preprint arXiv:2006.04182*.

Minsky, M. (1961). Steps toward artificial intelligence. *Proceedings of the IRE*, 49(1):8–30.

Mor, N., Wolf, L., Polyak, A., and Taigman, Y. (2018). A universal music translation network. *arXiv preprint arXiv:1805.07848*.

Moreno-Torres, J. G., Raeder, T., Alaiz-Rodríguez, R., Chawla, N. V., and Herrera, F. (2012). A unifying view on dataset shift in classification. *Pattern Recognition*, 45:521–530.

Morerio, P., Volpi, R., Ragonesi, R., and Murino, V. (2020). Generative pseudo-label refinement for unsupervised domain adaptation. In *Proceedings of the IEEE Winter Conference on Applications of Computer Vision*, pages 3130–3139.

Motiian, S., Jones, Q., Iranmanesh, S., and Doretto, G. (2017). Few-shot adversarial domain adaptation. In *Advances in Neural Information Processing Systems*, volume 30.

Mourot, L., Hoyet, L., Le Clerc, F., Schnitzler, F., and Hellier, P. (2021). A survey on deep learning for skeleton-based human animation. *Computer Graphics Forum*.

Mu, N. and Gilmer, J. (2019). MNIST-C: A robustness benchmark for computer vision. *arXiv preprint arXiv:1906.02337*.

Mukai, T. and Kuriyama, S. (2005). Geostatistical motion interpolation. *ACM Transactions on Graphics (TOG)*, 24(3):1062–1070.

Müller, R., Kornblith, S., and Hinton, G. E. (2019). When does label smoothing help? In *Advances in Neural Information Processing Systems*, volume 32.

Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press.

Naeini, M. P., Cooper, G., and Hauskrecht, M. (2015). Obtaining well calibrated probabilities using Bayesian binning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29.

Nakano, R. (2018). Porting arbitrary style transfer to the browser. `https://magenta.tensorflow.org/blog/2018/12/20/style-transfer-js/`. Accessed: 2021-12-16.

Nichol, A., Achiam, J., and Schulman, J. (2018). On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*.

Niculescu-Mizil, A. and Caruana, R. (2005). Predicting good probabilities with supervised learning. In *International Conference on Machine Learning*, pages 625–632.

Olah, C., Satyanarayan, A., Johnson, I., Carter, S., Schubert, L., Ye, K., and Mordvintsev, A. (2018). The building blocks of interpretability. *Distill*.

Parascandolo, G., Kilbertus, N., Rojas-Carulla, M., and Schölkopf, B. (2018). Learning independent causal mechanisms. In *International Conference on Machine Learning*, pages 4036–4044.

Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., and Wermter, S. (2019). Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71.

Park, S., Jang, D.-K., and Lee, S.-H. (2021). Diverse motion stylization for multiple style domains via spatial-temporal graph-based generative model. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 4(3):1–17.

Peng, X., Usman, B., Kaushik, N., Wang, D., Hoffman, J., and Saenko, K. (2018a). VISDA: A synthetic-to-real benchmark for visual domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 2021–2026.

Peng, X. B., Abbeel, P., Levine, S., and van de Panne, M. (2018b). Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics (TOG)*, 37(4).

Perez, E., Strub, F., De Vries, H., Dumoulin, V., and Courville, A. (2018). Film: Visual reasoning with a general conditioning layer. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Power, J. D. and Schlaggar, B. L. (2017). Neural plasticity across the lifespan. *Wiley Interdisciplinary Reviews: Developmental Biology*, 6(1):e216.

Quionero-Candela, J., Sugiyama, M., Schwaighofer, A., and Lawrence, N. D. (2009). *Dataset Shift in Machine Learning*. The MIT Press.

Quiroga, R. Q., Reddy, L., Kreiman, G., Koch, C., and Fried, I. (2005). Invariant visual representation by single neurons in the human brain. *Nature*, 435(7045):1102–1107.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al. (2021). Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*.

Raissi, M., Perdikaris, P., and Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707.

Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., Chen, M., and Sutskever, I. (2021). Zero-shot text-to-image generation. In *International Conference on Machine Learning*.

Ramponi, A. and Plank, B. (2020). Neural unsupervised domain adaptation in NLP—A survey. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6838–6855.

Ravi, S. and Larochelle, H. (2017). Optimization as a model for few-shot learning. In *International Conference on Learning Representations*.

Rebuffi, S.-A., Bilen, H., and Vedaldi, A. (2017). Learning multiple visual domains with residual adapters. In *Advances in Neural Information Processing Systems*, pages 506–516.

Rebuffi, S.-A., Bilen, H., and Vedaldi, A. (2018). Efficient parametrization of multi-domain deep neural networks. In *Proceedings of The IEEE Conference on Computer Vision and Pattern Recognition*.

Recht, B., Roelofs, R., Schmidt, L., and Shankar, V. (2019). Do imagenet classifiers generalize to imagenet? In *International Conference on Machine Learning*.

Ringwald, T. and Stiefelhagen, R. (2021). Adaptiope: A modern benchmark for un-

supervised domain adaptation. In *Proceedings of the IEEE Winter Conference on Applications of Computer Vision*, pages 101–110.

Rose, C., Cohen, M. F., and Bodenheimer, B. (1998). Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics and Applications*, 18(5):32–40.

Royer, A. and Lampert, C. (2020). A flexible selection scheme for minimum-effort transfer learning. In *Proceedings of the IEEE Winter Conference on Applications of Computer Vision*, pages 2191–2200.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015). Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252.

Saenko, K., Kulis, B., Fritz, M., and Darrell, T. (2010). Adapting visual category models to new domains. In *European Conference on Computer Vision*, pages 213–226.

Safonova, A., Hodgins, J. K., and Pollard, N. S. (2004). Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. *ACM Transactions on Graphics (TOG)*, 23(3):514–521.

Schölkopf, B., Janzing, D., Peters, J., Sgouritsa, E., Zhang, K., and Mooij, J. (2012). On causal and anticausal learning. *arXiv preprint arXiv:1206.6471*.

Schölkopf, B., Locatello, F., Bauer, S., Ke, N. R., Kalchbrenner, N., Goyal, A., and Bengio, Y. (2021). Towards causal representation learning. *Proceedings of the IEEE Advances in Machine Learning and Deep Neural Networks*, 109(5):612–634.

Shen, T., Lei, T., Barzilay, R., and Jaakkola, T. (2017). Style transfer from non-parallel text by cross-alignment. In *Advances in Neural Information Processing Systems*, volume 30.

Shimodaira, H. (2000). Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of Statistical Planning and Inference*, 90(2):227–244.

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

Shyam, P., Gupta, S., and Dukkipati, A. (2017). Attentive recurrent comparators. In *International Conference on Machine Learning*, pages 3173–3181.

Smith, H. J., Cao, C., Neff, M., and Wang, Y. (2019). Efficient neural networks for real-time motion style transfer. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 2(2).

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

Stan, S. and Rostami, M. (2021). Unsupervised model adaptation for continual semantic segmentation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 2593–2601.

Starke, S., Zhang, H., Komura, T., and Saito, J. (2019). Neural state machine for character-scene interactions. *ACM Transactions on Graphics (TOG)*, 38(6).

Starke, S., Zhao, Y., Komura, T., and Zaman, K. (2020). Local motion phases for learning multi-contact character movements. *ACM Transactions on Graphics (TOG)*, 39(4):54–1.

Storkey, A. J. (2009). When training and test sets are different: characterising learning transfer. In *Dataset Shift in Machine Learning*, pages 3–28. MIT Press.

Sugiyama, M., Krauledat, M., and Müller, K.-R. (2007). Covariate shift adaptation by importance weighted cross validation. *Journal of Machine Learning Research*, 8(5).

Sun, B. and Saenko, K. (2016). Deep CORAL: Correlation alignment for deep domain adaptation. In *European Conference on Computer Vision*, pages 443–450.

Sun, Y., Tzeng, E., Darrell, T., and Efros, A. A. (2019). Unsupervised domain adaptation through self-supervision. *arXiv preprint arXiv:1909.11825*.

Sutton, R. (2019). The bitter lesson. `http://www.incompleteideas.net/IncIdeas/BitterLesson.html`. Accessed: 7th January 2021.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the

inception architecture for computer vision. In *Proceedings of The IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826.

Taylor, G. W. and Hinton, G. E. (2009). Factored conditional restricted boltzmann machines for modeling motion style. In *International Conference on Machine Learning*, pages 1025–1032.

Tenenbaum, J. and Freeman, W. (1996). Separating style and content. In *Advances in Neural Information Processing Systems*, volume 9.

Theis, L., van den Oord, A., and Bethge, M. (2016). A note on the evaluation of generative models. In *International Conference on Learning Representations*.

Toldo, M., Maracani, A., Michieli, U., and Zanuttigh, P. (2020). Unsupervised domain adaptation in semantic segmentation: A review. *Technologies*, 8(2):35.

Torralba, A. and Efros, A. A. (2011). Unbiased look at dataset bias. In *Proceedings of The IEEE Conference on Computer Vision and Pattern Recognition*, pages 1521–1528.

Tzeng, E., Hoffman, J., Saenko, K., and Darrell, T. (2017). Adversarial discriminative domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7167–7176.

Van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(11).

Villegas, R., Yang, J., Ceylan, D., and Lee, H. (2018). Neural kinematic networks for unsupervised motion retargetting. In *Proceedings of The IEEE Conference on Computer Vision and Pattern Recognition*.

von Kügelgen, J., Sharma, Y., Gresele, L., Brendel, W., Schölkopf, B., Besserve, M., and Locatello, F. (2021). Self-supervised learning with data augmentations provably isolates content from style. In *Advances in Neural Information Processing Systems*.

Wang, D., Shelhamer, E., Liu, S., Olshausen, B., and Darrell, T. (2021). TENT: Fully test-time adaptation by entropy minimization. In *International Conference on Learning Representations*.

Wang, S.-Z. and Tao, H. W. (2009). History matters: illuminating metaplasticity in the developing brain. *Neuron*, 64(2):155–157.

Wang, Y., Yao, Q., Kwok, J. T., and Ni, L. M. (2020). Generalizing from a few examples: A survey on few-shot learning. *ACM Computing Surveys (CSUR)*, 53(3):1–34.

Wang, Z., Merel, J., Reed, S., Wayne, G., de Freitas, N., and Heess, N. (2017). Robust imitation of diverse behaviors. *arXiv preprint arXiv:1707.02747*.

Wen, Y.-H., Yang, Z., Fu, H., Gao, L., Sun, Y., and Liu, Y.-J. (2021). Autoregressive stylized motion synthesis with generative flow. In *Proceedings of The IEEE Conference on Computer Vision and Pattern Recognition*, pages 13612–13621.

Wiles, O., Gowal, S., Stimberg, F., Alvise-Rebuffi, S., Ktena, I., Cemgil, T., et al. (2021). A fine-grained analysis on distribution shift. *arXiv preprint arXiv:2110.11328*.

Xia, S., Wang, C., Chai, J., and Hodgins, J. (2015). Realtime style transfer for unlabeled heterogeneous human motion. *ACM Transactions on Graphics (TOG)*, 34(4):119:1–119:10.

Yang, Y., Morillo, I. G., and Hospedales, T. M. (2018). Deep neural decision trees. In *ICML Workshop on Human Interpretability in Machine Learning (WHI)*.

You, K., Wang, X., Long, M., and Jordan, M. (2019). Towards accurate model selection in deep unsupervised domain adaptation. In *International Conference on Machine Learning*, pages 7124–7133.

Yumer, M. E. and Mitra, N. J. (2016). Spectral style transfer for human motion between independent actions. *ACM Transactions on Graphics (TOG)*, 35(4):137.

Zadrozny, B. and Elkan, C. (2001). Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers. In *International Conference on Machine Learning*, pages 609–616.

Zeiler, M. D. and Fergus, R. (2014). Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*, pages 818–833.

Zellinger, W., Grubinger, T., Lughofer, E., Natschläger, T., and Saminger-Platz, S. (2017). Central moment discrepancy (CMD) for domain-invariant representation learning. In *International Conference on Learning Representations*.

Zenke, F., Poole, B., and Ganguli, S. (2017). Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, pages 3987–3995.

Zhang, H., Starke, S., Komura, T., and Saito, J. (2018a). Mode-adaptive neural networks for quadruped motion control. *ACM Transactions on Graphics (TOG)*, 37(4):145.

Zhang, K., Schölkopf, B., Muandet, K., and Wang, Z. (2013). Domain adaptation under target and conditional shift. In *International Conference on Machine Learning*, pages 819–827.

Zhang, Y., Tang, H., and Jia, K. (2018b). Fine-grained visual categorization using meta-learning optimization with sample selection of auxiliary data. In *European Conference on Computer Vision*, pages 233–248.

Zhao, C., Hospedales, T. M., Stulp, F., and Sigaud, O. (2017). Tensor based knowledge transfer across skill categories for robot control. In *International Joint Conference in Artificial Intelligence (IJCAI)*, volume 10, pages 1–4.

Zhu, J.-Y., Park, T., Isola, P., and Efros, A. A. (2017a). Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*.

Zhu, J.-Y., Zhang, R., Pathak, D., Darrell, T., Efros, A. A., Wang, O., and Shechtman, E. (2017b). Toward multimodal image-to-image translation. In *Advances in Neural Information Processing Systems*, pages 465–476.

Appendix A

Style Modelling for Animation

This appendix contains further information on the data we use for style modelling and how it was collected.

A.1 CMU Few-Shot Dataset

The first table in this appendix, Table A.1, shows the styles used for few-shot style modelling along with the number of training frames and locomotion styles available for each style.

A.2 The 100Style Dataset

The next table, split over two pages, contains frame counts and additional information for each of the styles in the 100STYLEdataset (Table A.2, Table A.3).

A.2.1 Motion Capture Scripts

We additionally provide the motion capture scripts used by the actor for each of the 100 styles collected. Note that the 'Backwards Walk' script is identical to the 'Forwards Walk' script but walking backwards. Similarly the 'Backwards Run' script is identical to the 'Forwards Run' script but running backwards. We do not provide a script for idling with the only instruction being to stand in the centre of the motion capture space for around 15 seconds idling in a posture of the style being captured. See Tables A.4, A.5, A.6, A.7 & A.8.

Table A.1: Unmirrored frame counts and locomotion cycles for few-shot styles.

Style	Number of Frames	Number of Locomotion Cycles	Symmetric
Balance	1491	12	Yes
Bent Forward	464	6	Yes
Bent Knees	182	2	Yes
Bouncy	304	4	No
Cat	197	2	Yes
Chicken	56	1	Yes
Cool	244	3	Yes
Crossover	230	2	Yes
Crouched	310	4	Yes
Dance	680	9	No
Dinosaur	527	5	Yes
Drag Leg	412	5	No
Dragon	125	1	Yes
Drunk	431	7	Yes
Duck Foot	258	3	Yes
Elated	67	1	Yes
Empi	234	1	Yes
Frankenstein	293	2	Yes
Gangly	493	7	Yes
Gedanbarai	294	2	Yes
Ghost	124	2	Yes
Graceful	323	6	Yes
Heavyset	417	4	Yes
Heiansyodan	95	1	Yes
Hobble	143	2	No
Hurt Leg	455	4	No
Jaunty	78	1	Yes
Joy	138	2	Yes
Lean Right	302	4	No
Left Hop	455	8	No
Legs Apart	186	2	Yes
Mantis	212	5	Yes
March	213	2	Yes
Mawashigeri	342	1	Yes
Monkey	103	2	Yes
Oiduki	228	1	Yes
On Toes Bent Forward	390	5	Yes
On Toes Crouched	591	8	Yes
Painful Left Knee	411	5	No
Penguin	183	6	Yes
Pigeon Toed	249	3	Yes
Prarie Dog	123	3	Yes
Quail	55	2	Yes
Roadrunner	665	17	Yes
Rushed	78	2	Yes
Sneaky	314	2	Yes
Squirrel	129	9	Yes
Stern	958	19	Yes
Stuff	97	2	Yes
Stumble	66	1	Yes
Swing Shoulders	259	3	Yes
Syutouuke	182	1	Yes
Wild Arms	161	2	Yes
Wild Legs	211	2	Yes
Wounded Leg	986	25	No
Yokogeri	135	1	No
Zombie	2241	25	Yes

Table A.2: Unmirrored frame counts for styles in the 100STYLE dataset, 1 of 2.

Style Name	Number of Frames	Stochastic	Symmetric
Aeroplane	39815	No	Yes
Akimbo	35919	No	Yes
Angry	25836	No	Yes
ArmsAboveHead	32734	No	Yes
ArmsBehindBack	34573	No	Yes
ArmsBySide	34605	No	Yes
ArmsFolded	35479	No	No
Balance	50215	No	No
BeatChest	35766	No	Yes
BentForward	34899	No	Yes
BentKnees	39058	No	Yes
BigSteps	32162	No	Yes
BouncyLeft	35347	No	No
BouncyRight	35779	No	No
Cat	45664	No	Yes
Chicken	37171	No	Yes
CrossOver	49445	No	Yes
Crouched	35961	No	Yes
CrowdAvoidance	45536	Yes	Yes
Depressed	46082	No	Yes
Dinosaur	44159	No	Yes
DragLeftLeg	47100	No	No
DragRightLeg	47079	No	No
Drunk	40813	Yes	Yes
DuckFoot	53656	No	Yes
Elated	30203	No	Yes
FairySteps	61085	No	Yes
Flapping	29954	No	Yes
FlickLegs	54801	No	Yes
Followed	33129	Yes	Yes
GracefulArms	35560	No	Yes
HandsBetweenLegs	34031	No	Yes
HandsInPockets	35352	No	Yes
Heavyset	36616	No	Yes
HighKnees	47754	No	Yes
InTheDark	51173	Yes	Yes
KarateChop	44645	No	Yes
Kick	49531	No	Yes
LawnMower	39812	No	Yes
LeanBack	38295	No	Yes
LeanLeft	42458	No	No
LeanRight	44261	No	No
LeftHop	19980	No	No
LegsApart	43386	No	Yes
LimpLeft	49085	No	No
LimpRight	43484	No	No
LookUp	41732	No	Yes
Lunge	40767	No	No
March	36955	No	Yes

Table A.3: Unmirrored frame counts for styles in the 100STYLE dataset, 2 of 2.

Style Name	Number of Frames	Stochastic	Symmetric
Monk	43483	No	No
Morris	35172	No	Yes
Neutral	39459	No	Yes
Old	68594	No	No
OnHeels	35905	No	Yes
OnPhoneLeft	34928	No	No
OnPhoneRight	37811	No	No
OnToesBentForward	35340	No	Yes
OnToesCrouched	38596	No	Yes
PendulumHands	37721	No	Yes
Penguin	67976	No	Yes
PigeonToed	56513	No	Yes
Proud	34784	No	Yes
Punch	35338	No	No
Quail	43698	No	Yes
RaisedLeftArm	30515	No	No
RaisedRightArm	32890	No	No
RightHop	19916	No	No
Roadrunner	29006	No	Yes
Robot	53326	No	Yes
Rocket	34272	No	Yes
Rushed	21238	No	Yes
ShieldedLeft	32339	No	No
ShieldedRight	33021	No	No
Skip	32118	No	Yes
SlideFeet	44216	No	Yes
SpinAntiClock	31379	No	No
SpinClock	30843	No	No
Star	38114	No	Yes
StartStop	81345	No	Yes
Stiff	46598	No	Yes
Strutting	43946	No	Yes
Superman	32435	No	No
Swat	29451	No	No
Sweep	43930	No	No
Swimming	34603	No	Yes
SwingArmsRound	45382	No	Yes
SwingShoulders	45826	No	Yes
Teapot	31942	No	No
Tiptoe	36052	No	Yes
TogetherStep	63544	No	Yes
TwoFootJump	28476	No	Yes
WalkingStickLeft	43619	No	No
WalkingStickRight	44112	No	No
Waving	39756	Yes	Yes
WhirlArms	39114	No	Yes
WideLegs	43191	No	Yes
WiggleHips	63231	No	Yes
WildArms	38409	No	Yes
WildLegs	51699	No	Yes
Zombie	41904	No	Yes

Table A.4: Forwards Walk Mocap Script.

Gait: Walk Forward
Begin T-Pose in centre
Walk straight forward to edge of mocap area
Turn 180° and walk straight to other edge
Walk in 1 clockwise diamond
Turn and walk 1 anticlockwise diamond
Walk one figure 8 – begin by turning to left
Walk one figure 8 – other direction
Walk one large circle clockwise
Walk one large circle anticlockwise
Walk straight back to centre and stop
Turn 90° left, walk to left side and stop
Turn 90° right, walk to corner and stop
Walk back to centre and stop
Walk one small circle clockwise
Walk one small circle anti-clockwise
Return to centre and T-Pose

Table A.5: Forwards Run Mocap Script.

Gait: Run Forward
Begin T-Pose in centre
Run straight forward to edge of mocap area
Turn 180° and run straight to other edge
Run in 1 clockwise diamond
Turn and run 1 anticlockwise diamond
Run one figure 8 – begin by turning to left
Run one figure 8 – other direction
Run one large circle clockwise
Run one large circle anticlockwise
Run straight back to centre and stop
Turn 90° left, run to left side and stop
Turn 90° right, run to corner and stop
Run back to centre and stop
Run one small circle clockwise
Run one small circle anti-clockwise
Return to centre and T-Pose

Table A.6: Sidestep Walk Mocap Script.

Gait: Sidestep Walk
Always face out from the centre
Begin T-Pose in the centre
Side walk to edge of mocap area
Side walk in clockwise diamond
Side walk in anti-clockwise diamond
Side walk in circle clockwise
Side walk in circle anticlockwise
Side walk left and stop
Side walk right and stop
Return to centre

Table A.7: Sidestep Run Mocap Script.

Gait: Sidestep Run
Always face out from the centre
Begin T-Pose in the centre
Side run to edge of mocap area
Side run in clockwise diamond
Side run in anti-clockwise diamond
Side run in circle clockwise
Side run in circle anticlockwise
Side run left and stop
Side run right and stop
Return to centre

Table A.8: Transitions Mocap Script.

Gait: Transitions
Aim to capture all transitions
Begin T-Pose in centre
While moving in random directions
Forwards walk to forwards run and back
Forwards walk to backwards walk and back
Forwards walk to backwards run and back
Forwards walk to side walk and back
Forwards walk to side run and back
Forwards run to back walk and back
Forwards run to back run and back
Forwards run to side walk and back
Forwards run to side run and back
Back walk to back run and back
Back walk to side walk and back
Back walk to side run and back
Back run to side walk and back
Back run to side run and back
Side walk to side run and back
Return to centre and T-pose

Appendix B

Feature Restoration

B.1 Datasets

Section 4.3 described the EMNIST-DA dataset created for feature restoration and unit-level surprise investigation. In this section we briefly describe the other preexisting datasets we use to evaluate feature restoration.

The simplest datasets used are based on MNIST. MNIST-M (Ganin et al. (2016), Figure B.1) combines MNIST digits with random background patches from BSDS500 (Arbelaez et al., 2011) to create a single target domain and MNIST-C (Mu and Gilmer (2019), Figure B.2) contains 15 corruptions of the standard MNIST data, creating 15 possible target domains. For both datasets the source data is standard MNIST with both pretraining and adaptation performed on data from the MNIST training split.

For more complex image recognition we use common corruptions of CIFAR datasets (Krizhevsky, 2009), CIFAR-10-C and CIFAR-100-C (Hendrycks and Dietterich (2019), Figure B.3). CIFAR-10-C has 10 possible object classes and CIFAR-100-C 100. The source domain is uncorrupted CIFAR-10 or CIFAR-100 with each of the 19 corruptions forming one possible target domain. As with EMNIST-DA, models are trained on the training set of CIFAR-10/CIFAR-100 and adapted to corruptions of the test set.

To demonstrate measurement shift in the real world we use CAMELYON17 (Bandi et al. (2018), Koh et al. (2021), Figure B.4), a dataset for classifying histopathological images taken from 5 different hospitals. The aim is to detect the presence (or lack) of tumorous tissue. A source model is trained on data from a single hospital (hospital 3) with each remaining hospital providing one target domain.

Figure B.1: *Top:* MNIST examples. *Bottom:* MNIST-M examples.

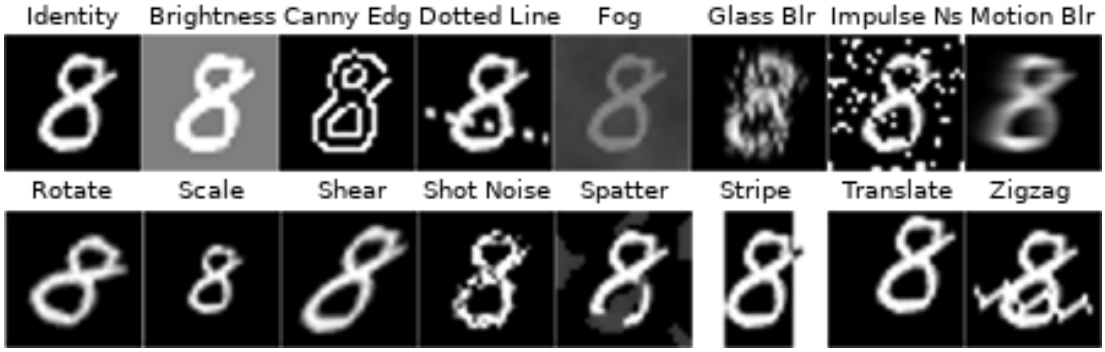


Figure B.2: 15 different MNIST-C corruptions.

Figure B.3: 19 different CIFAR corruptions. The same corruptions are used for CIFAR-10-C and CIFAR-100-C.

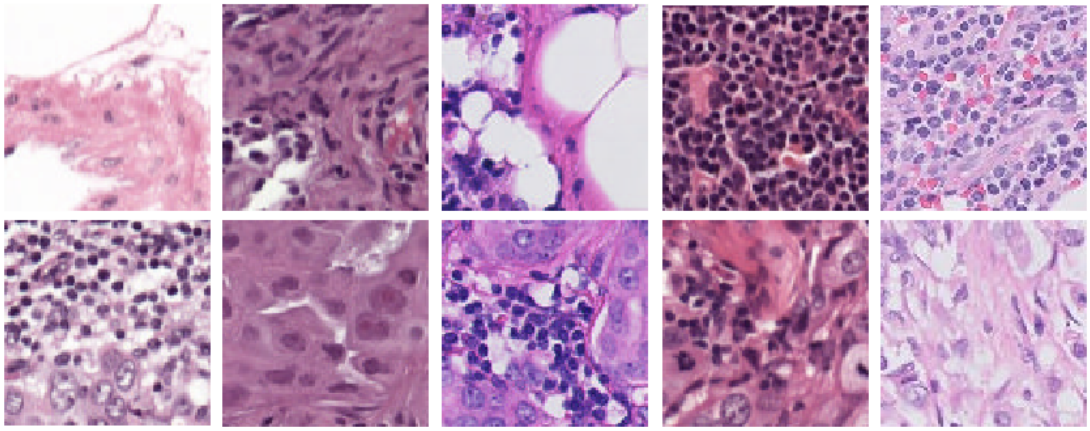
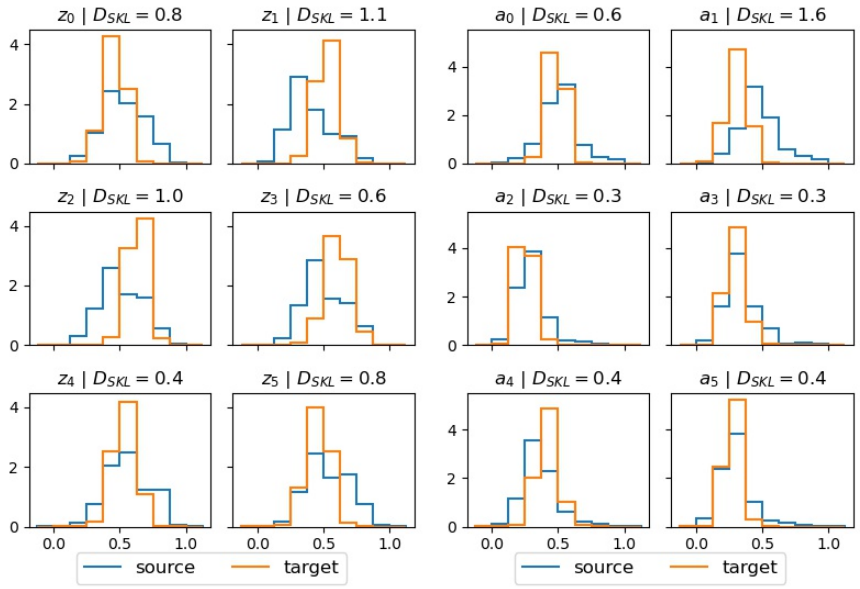


Figure B.4: CAMELYON17 examples. Each column is a different hospital. *Top row:* no tumour tissue. *Bottom row:* tumour tissue.

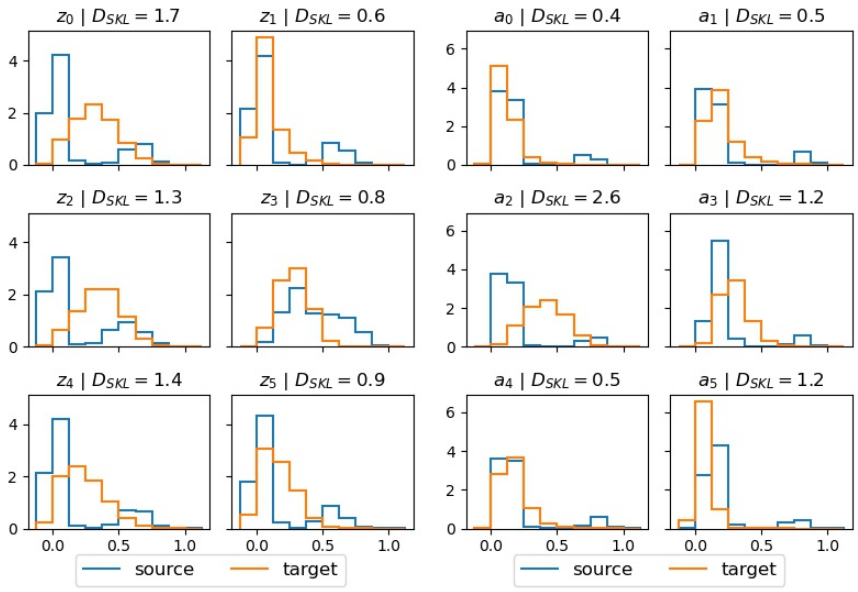
B.2 Activation Distributions

This appendix shows source and target marginal feature and logit distributions before adaptation for EMNIST-DA (Figure B.5) and CIFAR-10-C (Figure B.6). Note that for CIFAR-10-C source distributions tend to be bimodal whereas target distributions tend to be unimodal before adaptation. With a view of hidden units as feature detectors, the two modes of the source distributions can be intuitively understood as whether a feature is detected (the unit is 'on') or not (the unit is 'off'). Figure B.7 shows the alignment of the marginals after adaptation for different SFDA methods on CIFAR-10-C. Figure 4.5 in Section 4.5.1 showed the equivalent alignment for EMNIST-DA. Whilst Figure B.7 also shows that our *BUFR* method aligns distributions more closely than other methods, for CIFAR-10-C it seems that some distribution alignment is more important for achieving reasonable accuracy than for EMNIST-DA since the marginal distributions are more closely aligned even when this is not optimised for directly (as in *AdaBN* and *SHOT-IM*).



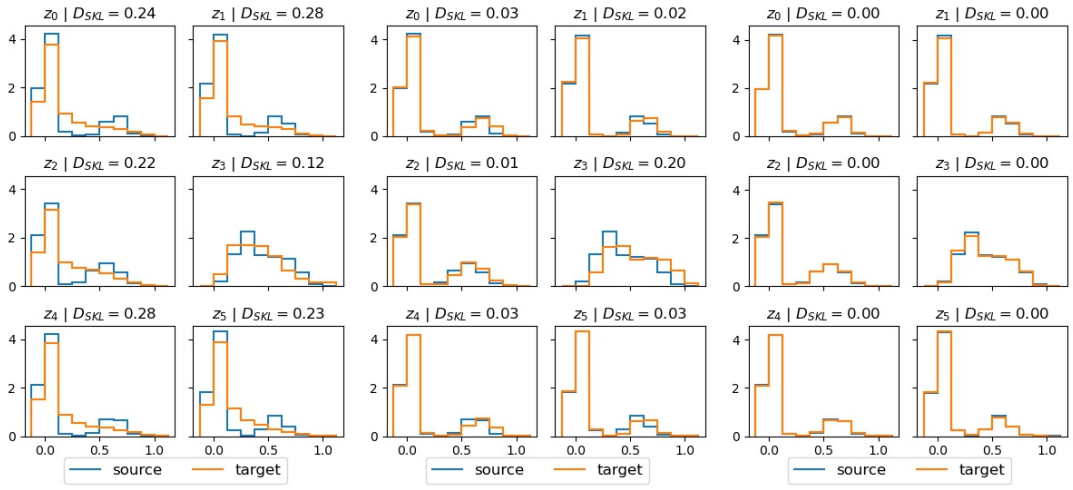
(a) Feature distributions (b) Logit distributions

Figure B.5: Histograms of 6 marginal distributions using the EMNIST-DA stripe shift. The blue curves are the saved marginal distributions under the source data (identity). The orange curves are the marginal distributions under the target data (stripe) *before adaptation*. (a) Marginal feature activation distributions. (b) Marginal logit activation distributions. D_{SKL} is the symmetric KL divergence.



(a) Feature distributions (b) Logit distributions

Figure B.6: Histograms of 6 marginal distributions using the CIFAR-10-C impulse-noise shift. The blue curves are the saved marginal distributions under the source data (identity). The orange curves are the marginal distributions under the target data (impulse-noise) *before adaptation*. (a) Marginal feature activation distributions. (b) Marginal logit activation distributions. D_{SKL} is the symmetric KL divergence.



(a) AdaBN (b) SHOT-IM (c) BUFR

Figure B.7: Histograms showing distribution alignment on the CIFAR-10-C impulse-noise shift. The blue curves are the saved marginal distributions under the source data (identity). The orange curves are the marginal distributions under the target data (impulse-noise) *after adaptation*. (a) AdaBN only slightly aligns the marginal distributions. (b) SHOT-IM partially aligns the marginal distributions despite not optimising for alignment. (c) BUFR matches the activation distributions very closely.

B.3 Extended Results

This appendix provides full, per-shift, results for each of the SFDA methods used for MNIST-C, MNIST-M, EMNIST-DA, CIFAR-10-C, and CIFAR-100-C.

B.3.1 MNIST-C full results

Tables B.1 and B.2 show the accuracy and ECE results for each individual corruption of the MNIST-C dataset. As the translate corruption violates the assumptions underlying the use of a fixed classifier h, we additionally show average results without this corruption. Without translate all methods achieve very high accuracy ($\geq 95\%$).

Table B.1: MNIST-C accuracy (%) results. Shown are the mean and 1 standard deviation.

	Src-only	**AdaBN**	**PL**	**SHOT-IM**	**SHOT**	**FR**	**BUFR**
Brightness	84.8 ± 11.4	99.4 ± 0.0	99.5 ± 0.0	99.4 ± 0.1	99.5 ± 0.1	98.7 ± 0.1	99.2 ± 0.1
Canny Edges	72.2 ± 0.8	91.0 ± 0.7	96.2 ± 1.0	98.1 ± 0.1	98.6 ± 0.1	97.8 ± 0.1	98.5 ± 0.0
Dotted Line	98.6 ± 0.2	98.8 ± 0.2	99.3 ± 0.1	99.4 ± 0.1	99.4 ± 0.1	98.6 ± 0.1	99.1 ± 0.0
Fog	30.1 ± 12.5	93.9 ± 1.9	99.3 ± 0.0	99.4 ± 0.1	99.5 ± 0.0	98.6 ± 0.1	99.2 ± 0.0
Glass Blur	88.9 ± 2.3	95.3 ± 0.3	97.8 ± 0.1	98.2 ± 0.1	98.3 ± 0.0	97.2 ± 0.1	97.9 ± 0.0
Impulse Noise	95.2 ± 0.6	97.9 ± 0.1	98.4 ± 0.1	98.7 ± 0.1	98.9 ± 0.1	97.8 ± 0.0	98.6 ± 0.0
Motion Blur	85.6 ± 3.9	97.3 ± 0.4	98.8 ± 0.1	99.1 ± 0.1	99.2 ± 0.1	98.1 ± 0.1	98.8 ± 0.1
Rotate	96.7 ± 0.1	96.7 ± 0.0	97.7 ± 0.1	98.4 ± 0.1	98.8 ± 0.0	97.5 ± 0.1	97.9 ± 0.1
Scale	97.2 ± 0.1	97.2 ± 0.1	98.7 ± 0.1	99.1 ± 0.0	99.2 ± 0.0	98.0 ± 0.0	98.7 ± 0.2
Shear	98.9 ± 0.1	98.9 ± 0.0	99.0 ± 0.0	99.1 ± 0.0	99.2 ± 0.0	98.3 ± 0.1	98.8 ± 0.1
Shot Noise	98.6 ± 0.0	99.0 ± 0.0	99.2 ± 0.0	99.2 ± 0.1	99.2 ± 0.0	98.3 ± 0.2	99.0 ± 0.1
Spatter	98.7 ± 0.1	98.8 ± 0.1	99.0 ± 0.1	99.0 ± 0.0	99.1 ± 0.1	98.4 ± 0.1	98.8 ± 0.0
Stripe	91.1 ± 1.2	90.9 ± 1.5	97.9 ± 1.0	99.2 ± 0.0	99.4 ± 0.1	98.3 ± 0.1	99.1 ± 0.1
Translate	64.6 ± 0.5	64.4 ± 0.6	69.5 ± 0.8	75.1 ± 4.1	78.7 ± 3.1	76.7 ± 2.1	64.5 ± 8.9
Zigzag	91.8 ± 0.6	93.0 ± 0.2	98.2 ± 0.2	98.9 ± 0.1	99.2 ± 0.1	98.2 ± 0.1	98.8 ± 0.1
Avg.	86.2 ± 1.8	94.2 ± 0.2	96.6 ± 0.1	97.3 ± 0.2	97.7 ± 0.2	96.7 ± 0.1	96.4 ± 0.6
Avg.\translate	87.7 ± 1.9	96.3 ± 0.2	98.5 ± 0.1	98.9 ± 0.0	99.1 ± 0.0	98.1 ± 0.1	98.7 ± 0.0

Table B.2: MNIST-C ECE (%) results. Shown are the mean and 1 standard deviation.

	Src-only	**AdaBN**	**PL**	**SHOT-IM**	**SHOT**	**FR**	**BUFR**
Brightness	2.4 ± 0.4	0.3 ± 0.1	0.3 ± 0.1	0.4 ± 0.1	0.9 ± 0.1	0.9 ± 0.1	0.5 ± 0.1
Canny Edges	22.2 ± 0.7	6.1 ± 0.7	4.0 ± 2.5	1.5 ± 0.1	0.6 ± 0.1	1.6 ± 0.1	1.0 ± 0.1
Dotted Line	0.7 ± 0.1	0.7 ± 0.1	0.5 ± 0.1	0.4 ± 0.1	0.9 ± 0.0	1.0 ± 0.1	0.6 ± 0.1
Fog	26.4 ± 18.0	3.5 ± 1.4	0.5 ± 0.0	0.4 ± 0.0	0.9 ± 0.1	1.0 ± 0.1	0.5 ± 0.1
Glass Blur	5.9 ± 1.6	3.1 ± 0.3	1.8 ± 0.1	1.4 ± 0.1	0.4 ± 0.2	2.1 ± 0.1	1.5 ± 0.1
Impulse Noise	1.2 ± 0.2	1.2 ± 0.1	1.2 ± 0.1	0.9 ± 0.1	0.7 ± 0.1	1.5 ± 0.1	1.0 ± 0.1
Motion Blur	9.1 ± 3.4	1.4 ± 0.2	1.0 ± 0.2	0.7 ± 0.1	0.8 ± 0.0	1.4 ± 0.1	0.8 ± 0.1
Rotate	2.0 ± 0.1	2.2 ± 0.1	1.9 ± 0.1	1.2 ± 0.1	0.6 ± 0.1	1.9 ± 0.1	1.5 ± 0.1
Scale	1.0 ± 0.1	1.7 ± 0.1	1.0 ± 0.1	0.7 ± 0.1	0.8 ± 0.1	1.5 ± 0.0	0.8 ± 0.1
Shear	0.7 ± 0.1	0.7 ± 0.0	0.8 ± 0.1	0.7 ± 0.1	0.8 ± 0.1	1.2 ± 0.1	0.9 ± 0.1
Shot Noise	0.7 ± 0.1	0.6 ± 0.1	0.5 ± 0.1	0.5 ± 0.1	0.8 ± 0.1	1.2 ± 0.1	0.7 ± 0.1
Spatter	0.6 ± 0.1	0.7 ± 0.1	0.7 ± 0.1	0.7 ± 0.1	0.9 ± 0.1	1.2 ± 0.1	0.8 ± 0.0
Stripe	4.3 ± 1.0	6.5 ± 1.4	2.8 ± 3.1	0.5 ± 0.1	0.9 ± 0.1	1.2 ± 0.2	0.6 ± 0.0
Translate	25.2 ± 0.3	28.6 ± 0.6	29.0 ± 0.7	24.2 ± 4.1	19.2 ± 3.0	18.8 ± 2.7	33.7 ± 8.9
Zigzag	5.4 ± 0.5	4.9 ± 0.3	1.3 ± 0.2	0.8 ± 0.0	0.8 ± 0.0	1.4 ± 0.1	0.8 ± 0.1
Avg.	7.2 ± 1.4	4.1 ± 0.1	3.1 ± 0.4	2.3 ± 0.2	2.0 ± 0.2	2.5 ± 0.2	3.0 ± 0.6
Avg.\translate	5.9 ± 1.5	2.4 ± 0.2	1.3 ± 0.4	0.8 ± 0.0	0.8 ± 0.0	1.4 ± 0.0	0.8 ± 0.0

B.3.2 EMNIST-DA full results

Tables B.3 and B.4 show the accuracy and ECE results for each individual shift of EMNIST-DA. To help the baseline models we also calculate the average accuracy without background colour shifts (bgs) as these tend to be the most severe shifts for which BUFR has the most benefits.

Table B.3: EMNIST-DA accuracy (%) results. Shown are the mean and 1 standard deviation.

	Src-only	**AdaBN**	**Marg. Gauss.**	**Full Gauss.**	**PL**	**BNM-IM**	**SHOT-IM**	**SHOT**	**FR**	**BUFR**
Bricks	4.2±0.5	5.9±1.0	9.1±1.2	22.6±2.1	6.8±1.2	14.2±1.4	20.5±4.8	76.0±0.2	32.4±4.8	83.8±0.3
Crystals	19.7±3.1	42.1±1.9	50.0±0.7	60.0±1.0	47.4±1.6	61.2±2.5	71.5±3.8	80.1±0.2	76.8±0.4	82.6±0.3
Dotted Line	76.2±0.7	80.8±0.4	82.3±0.4	82.6±0.5	80.7±0.5	86.5±0.4	87.1±0.1	87.5±0.1	87.6±0.1	88.3±0.0
Fog	4.5±0.9	69.0±2.6	77.1±0.6	85.1±0.6	77.4±3.3	86.3±0.3	86.2±0.1	87.0±0.1	87.0±0.1	88.3±0.1
Gaussian Blur	45.1±3.2	65.2±1.6	77.1±0.8	82.3±0.2	78.8±0.8	83.7±0.3	83.7±0.2	83.9±0.2	83.0±0.4	86.0±0.1
Grass	2.3±0.1	6.1±0.4	5.9±1.1	52.7±3.5	6.7±1.8	14.6±6.1	42.4±40.9	61.8±36.5	79.2±0.3	84.5±0.2
Impulse Noise	36.8±1.6	76.7±0.8	81.4±0.3	82.6±0.1	79.9±0.6	84.2±0.3	84.4±0.2	84.4±0.2	84.8±0.2	86.0±0.1
Inverse	5.6±0.5	8.1±2.1	14.1±6.2	64.4±4.3	11.3±2.0	60.4±23.4	83.2±0.4	85.1±0.2	83.1±0.7	88.3±0.1
Oranges	26.5±2.7	40.7±2.3	49.3±1.0	77.1±0.6	43.0±3.1	79.9±0.6	80.5±0.3	82.4±0.2	82.3±0.3	84.8±0.3
Shot Noise	78.5±0.7	84.9±0.2	85.8±0.3	85.7±0.2	85.0±0.3	86.5±0.1	86.3±0.1	86.1±0.1	86.4±0.1	87.0±0.2
Sky	3.8±0.4	3.7±0.7	4.8±0.4	29.8±9.8	3.3±0.6	8.3±1.3	24.0±7.5	55.1±23.5	15.3±6.8	84.6±0.2
Stripe	15.4±1.1	46.9±4.9	63.0±5.6	82.3±0.8	63.8±3.7	82.8±0.5	83.9±0.3	85.1±0.2	84.5±0.4	87.1±0.1
Zigzag	65.0±0.2	71.3±0.2	73.8±0.1	76.1±0.3	72.3±0.2	79.7±0.5	81.0±0.5	85.8±0.2	85.7±0.2	87.5±0.2
Avg.	29.5±0.5	46.2±1.1	51.8±1.1	67.9±0.7	50.5±0.6	63.7±2.2	70.3±3.7	80.0±4.4	74.4±0.8	86.1±0.1
Avg.\bgs	40.9±0.4	62.8±1.1	69.3±1.4	80.1±0.5	68.6±0.5	81.2±3.1	84.5±0.1	85.6±0.1	85.2±0.1	87.3±0.0

Table B.4: EMNIST-DA ECE (%) results. Shown are the mean and 1 standard deviation.

	Src-only	**AdaBN**	**Marg. Gauss.**	**Full Gauss.**	**PL**	**BNM-IM**	**SHOT-IM**	**SHOT**	**FR**	**BUFR**
Bricks	54.6±5.0	64.4±1.1	62.0±0.9	52.4±2.0	93.2±1.2	84.9±1.4	79.4±4.8	22.5±0.3	44.1±4.2	6.2±0.4
Crystals	27.0±3.0	29.0±1.2	24.0±0.3	22.0±0.6	52.7±1.7	38.1±2.5	28.5±3.8	19.3±0.1	10.5±0.3	6.9±0.3
Dotted Line	11.4±0.6	9.2±0.5	7.9±0.4	7.5±0.4	19.6±0.7	13.0±0.4	12.9±0.1	12.9±0.1	3.8±0.2	3.4±0.2
Fog	19.2±6.5	13.8±1.5	8.8±0.8	4.8±0.4	24.1±4.0	13.3±0.4	13.8±0.1	13.5±0.1	3.9±0.2	3.3±0.1
Gaussian Blur	15.0±3.9	15.3±0.9	8.7±0.5	6.8±0.3	21.2±0.8	15.8±0.4	16.4±0.2	16.0±0.3	6.4±0.7	4.9±0.1
Grass	21.6±5.4	61.3±0.8	61.0±1.0	27.2±2.7	93.6±1.5	84.6±6.2	57.5±40.9	37.5±36.1	8.7±0.6	5.7±0.2
Impulse Noise	32.0±1.8	9.9±0.6	7.1±0.3	6.6±0.1	20.1±0.6	15.3±0.3	15.6±0.2	15.8±0.2	5.3±0.1	4.7±0.1
Inverse	65.1±5.8	60.8±2.2	54.9±5.7	18.1±3.0	89.3±2.1	39.0±23.3	16.9±0.5	14.7±0.1	5.6±0.5	3.3±0.1
Oranges	23.8±2.7	25.3±2.0	22.5±2.3	10.1±0.6	57.6±2.7	19.6±0.6	19.6±0.4	17.4±0.2	6.9±0.5	5.5±0.3
Shot Noise	4.8±0.5	4.9±0.3	4.5±0.3	4.9±0.2	16.4±0.1	13.0±0.1	13.7±0.1	14.8±0.1	4.6±0.3	4.2±0.2
Sky	42.6±3.5	52.4±4.9	51.6±6.4	45.8±8.4	97.2±0.4	90.2±1.1	76.0±7.5	42.7±23.0	58.0±6.8	5.6±0.3
Stripe	63.8±3.0	31.6±4.4	20.2±4.8	6.8±0.4	36.2±3.8	16.8±0.5	16.1±0.3	15.0±0.3	5.4±0.2	4.1±0.1
Zigzag	19.9±0.3	16.7±0.2	14.6±0.1	12.7±0.3	27.6±0.2	19.7±0.5	19.0±0.5	14.4±0.1	4.9±0.1	3.8±0.2
Avg.	30.8±1.6	30.3±1.1	26.7±1.1	17.4±0.7	49.9±0.6	35.6±2.2	29.6±3.7	19.7±4.4	12.9±0.9	4.7±0.2
Avg.\bgs	28.9±1.3	20.3±0.8	15.8±1.1	8.5±0.4	31.8±0.5	18.2±3.1	15.6±0.1	14.6±0.1	5.0±0.2	4.0±0.1

B.3.3 CIFAR-10-C full results

Tables B.5 and B.6 show the accuracy and ECE results for each individual corruption of CIFAR-10-C. As with EMNIST-DA BUFR achieves the biggest wins on the most severe shifts (those with lowest AdaBN performance).

Table B.5: CIFAR-10-C accuracy (%) results. Shown are the mean and 1 standard deviation.

	Src-only	**AdaBN**	**PL**	**SHOT-IM**	**TENT**	**FR**	**BUFR**
Brightness	91.4 ± 0.4	91.5 ± 0.3	91.9 ± 0.2	92.7 ± 0.3	93.2 ± 0.3	93 ± 0.4	93.3 ± 0.3
Contrast	32.3 ± 1.3	87.1 ± 0.3	86.6 ± 3.2	90.8 ± 0.9	91.3 ± 1.7	90.9 ± 0.9	92.9 ± 0.7
Defocus blr	53.1 ± 6.4	88.8 ± 0.4	89.3 ± 0.5	90.5 ± 0.4	90.9 ± 0.5	90.9 ± 0.3	91.5 ± 0.5
Elastic	77.6 ± 0.6	78.2 ± 0.4	79.2 ± 0.8	81.4 ± 0.5	82.7 ± 0.5	82.7 ± 0.4	84.2 ± 0.3
Fog	72.9 ± 2.6	85.9 ± 0.9	86.5 ± 0.8	88.7 ± 0.4	89.5 ± 0.4	89.5 ± 0.5	91.5 ± 0.5
Frost	64.4 ± 2.4	80.7 ± 0.7	82.4 ± 1.2	85.4 ± 0.6	86.8 ± 0.7	87 ± 0.6	89.1 ± 0.9
Gauss. blr	35.9 ± 8	88.2 ± 0.6	89 ± 0.7	90.5 ± 0.5	91 ± 0.6	91.2 ± 0.6	92.3 ± 0.4
Gauss. nse	27.7 ± 5.1	69.2 ± 1	74.6 ± 0.6	79.2 ± 0.9	81.3 ± 0.5	81.9 ± 0.1	85.9 ± 0.4
Glass blr	51.3 ± 1.8	66.7 ± 0.4	69 ± 0.3	73.7 ± 1	74.7 ± 0.8	76.8 ± 0.8	80.3 ± 0.5
Impulse nse	25.9 ± 3.8	62.1 ± 1	67.2 ± 0.5	73.2 ± 0.8	75.3 ± 0.8	76.6 ± 0.4	89.3 ± 1.4
Jpeg compr.	74.9 ± 1	74.1 ± 1	77.3 ± 0.6	81 ± 0.3	82.9 ± 0.5	83.4 ± 0.5	85.8 ± 0.6
Motion blr	66.1 ± 1.7	87.2 ± 0.2	87.8 ± 0.2	89.1 ± 0.2	90 ± 0.3	89.8 ± 0.2	90.8 ± 0.2
Pixelate	48.2 ± 2.2	80.4 ± 0.5	82 ± 0.4	85.5 ± 0.7	87.6 ± 0.9	87.5 ± 0.8	89.9 ± 0.6
Saturate	89.9 ± 0.4	92 ± 0.1	92.5 ± 0.3	93.1 ± 0.1	93.3 ± 0.1	93.4 ± 0.4	93.5 ± 0.3
Shot nse	34.4 ± 4.9	71.2 ± 1.2	77.1 ± 0.9	81.6 ± 0.7	83.5 ± 0.6	85.6 ± 1.9	87 ± 0.2
Snow	76.6 ± 1.1	82.4 ± 0.6	83.8 ± 1.1	86.4 ± 0.6	87.8 ± 0.7	88.4 ± 1.7	89.7 ± 0.5
Spatter	75 ± 0.8	83.3 ± 0.5	85.5 ± 0.3	88 ± 0.2	88.5 ± 0.3	91 ± 2.4	92.6 ± 0.5
Speckle nse	40.7 ± 3.7	70.4 ± 0.8	76.1 ± 1.2	81.3 ± 1	83.2 ± 0.9	85.8 ± 1.7	87.4 ± 0.4
Zoom blr	60.5 ± 5.1	88.1 ± 0.3	89 ± 0.4	90.6 ± 0.2	91.3 ± 0.3	91.2 ± 0.7	91.6 ± 0.2
Avg.	57.8 ± 0.7	80.4 ± 0.1	82.5 ± 0.3	85.4 ± 0.2	86.6 ± 0.3	87.2 ± 0.7	89.4 ± 0.2

Table B.6: CIFAR-10-C ECE (%) results. Shown are the mean and 1 standard deviation.

	Src-only	**AdaBN**	**PL**	**SHOT-IM**	**TENT**	**FR**	**BUFR**
Brightness	4.7 ± 0.2	4 ± 0.1	8.1 ± 0.2	7.2 ± 0.4	6.4 ± 0.3	5.9 ± 0.3	6.2 ± 0.2
Contrast	43.5 ± 2.8	5.7 ± 0.4	13.2 ± 3.1	9.8 ± 0.9	8.4 ± 1.6	6.6 ± 0.4	6.6 ± 0.7
Defocus blr	28.2 ± 4	6.1 ± 0.4	10.7 ± 0.5	9.4 ± 0.4	8.6 ± 0.5	7.8 ± 0.3	7.9 ± 0.4
Elastic	12.4 ± 0.7	12.6 ± 0.4	20.8 ± 0.8	18.6 ± 0.5	16.5 ± 0.5	15.1 ± 0.5	15.2 ± 0.3
Fog	17.4 ± 2.1	7.5 ± 0.7	13.5 ± 0.8	11.3 ± 0.3	10.1 ± 0.4	8.9 ± 0.3	8 ± 0.5
Frost	22.7 ± 1.7	10.4 ± 0.6	17.5 ± 1.2	14.6 ± 0.6	12.7 ± 0.6	10.7 ± 0.8	10.3 ± 0.9
Gauss. blr	40.7 ± 6.2	6.1 ± 0.4	11 ± 0.7	9.5 ± 0.4	8.5 ± 0.5	7.5 ± 0.4	7.3 ± 0.4
Gauss. nse	57.6 ± 6.7	18.5 ± 0.7	25.3 ± 0.6	20.9 ± 0.9	18 ± 0.4	15.9 ± 0.3	13.3 ± 0.4
Glass blr	31.2 ± 1.3	20.8 ± 0.4	30.9 ± 0.3	26.3 ± 1	24.2 ± 0.8	20.9 ± 0.7	18.9 ± 0.5
Impulse nse	51.2 ± 4	23.3 ± 0.8	32.7 ± 0.5	26.8 ± 0.9	23.7 ± 0.8	20.6 ± 0.5	10.2 ± 1.3
Jpeg compr.	14.6 ± 0.8	15.5 ± 0.7	22.6 ± 0.6	18.9 ± 0.4	16.4 ± 0.5	14.5 ± 0.4	13.6 ± 0.7
Motion blr	21.1 ± 1.3	6.8 ± 0.3	12.1 ± 0.2	10.9 ± 0.2	9.5 ± 0.3	8.7 ± 0.3	8.6 ± 0.2
Pixelate	36.9 ± 2.5	11.1 ± 0.4	17.9 ± 0.4	14.5 ± 0.7	11.9 ± 0.9	10.7 ± 0.7	9.5 ± 0.6
Saturate	5.5 ± 0.3	4.2 ± 0.1	7.4 ± 0.3	6.9 ± 0.1	6.4 ± 0.1	5.9 ± 0.2	6 ± 0.3
Shot nse	50.2 ± 5.9	17 ± 0.9	22.8 ± 0.9	18.4 ± 0.7	15.9 ± 0.6	14 ± 0.3	12.3 ± 0.2
Snow	14.3 ± 0.5	9.8 ± 0.4	16.1 ± 1.1	13.6 ± 0.6	11.6 ± 0.7	10.5 ± 0.7	9.7 ± 0.4
Spatter	16.9 ± 0.8	9.3 ± 0.3	14.5 ± 0.3	12 ± 0.2	11 ± 0.2	9.3 ± 0.2	7 ± 0.6
Speckle nse	43.2 ± 4.5	17.9 ± 0.5	23.8 ± 1.1	18.7 ± 1	16.1 ± 0.9	13.9 ± 0.7	11.9 ± 0.4
Zoom blr	24.4 ± 3.5	6.2 ± 0.1	11 ± 0.4	9.4 ± 0.2	8.3 ± 0.3	7.6 ± 0.5	7.9 ± 0.2
Avg.	28.2 ± 0.4	11.2 ± 0.1	17.5 ± 0.3	14.6 ± 0.2	12.8 ± 0.3	11.3 ± 0.3	10 ± 0.2

B.3.4 CIFAR-100-C full results

Tables B.7 and B.8 show the accuracy and ECE results for each individual corruption of CIFAR-100-C. As with EMNIST-DA BUFR achieves the biggest wins on the most severe shifts (those with lowest AdaBN performance).

Table B.7: CIFAR-100-C accuracy (%) results. Shown are the mean and 1 standard deviation.

	Src-only	**AdaBN**	**PL**	**SHOT-IM**	**TENT**	**FR**	**BUFR**
Brightness	63.2 ± 1.1	66.1 ± 0.5	69.6 ± 0.7	72.6 ± 0.6	72.2 ± 0.5	71.8 ± 0.5	73.6 ± 0.2
Contrast	13.9 ± 0.6	61.4 ± 0.4	59.2 ± 3.5	70.1 ± 0.4	64 ± 3.1	68 ± 0.5	72.2 ± 0.5
Defocus blr	35.9 ± 0.7	65.6 ± 0.1	69.3 ± 0.1	71.8 ± 0.3	71 ± 0.5	71.2 ± 0.1	72.2 ± 0.2
Elastic	58.5 ± 0.7	60.4 ± 0.2	63.9 ± 0.4	66.9 ± 0.2	65.5 ± 0.2	65.9 ± 0.4	67.1 ± 0.5
Fog	36.9 ± 0.5	55.4 ± 0.6	60.4 ± 0.6	66.5 ± 0.6	67.1 ± 0.6	64.9 ± 0.4	70.1 ± 0.5
Frost	41.1 ± 0.9	55.3 ± 0.6	60.1 ± 0.8	65.2 ± 0.4	65.3 ± 0.9	63 ± 0.5	67.5 ± 0.7
Gauss. blr	28.2 ± 1	64.3 ± 0.3	68.9 ± 0.1	71.7 ± 0.2	71 ± 0.3	70.9 ± 0.3	72.9 ± 0.6
Gauss. nse	11.9 ± 1.2	43.8 ± 0.6	53.1 ± 0.7	60.3 ± 0.4	59.5 ± 0.6	57.7 ± 0.4	63 ± 0.3
Glass blr	45.1 ± 0.9	53.3 ± 0.6	57.3 ± 0.7	62.4 ± 0.3	61.4 ± 0.5	60.5 ± 0.3	63.2 ± 0.4
Impulse nse	7.2 ± 0.8	40.8 ± 0.4	50.6 ± 0.5	58.4 ± 0.6	56.3 ± 0.7	55.2 ± 0.9	66.9 ± 0.6
Jpeg compr.	48.6 ± 0.9	49.8 ± 0.7	55.8 ± 0.3	61.2 ± 0.5	60.8 ± 0.1	59.3 ± 0.5	62.6 ± 0.4
Motion blr	45.1 ± 0.5	63.4 ± 0.2	66.3 ± 0.6	69.7 ± 0.2	69 ± 0.5	68.6 ± 0.4	70.8 ± 0.2
Pixelate	22.3 ± 0.4	59.4 ± 0.6	64.9 ± 0.6	69.7 ± 0.4	69.8 ± 0.4	68.1 ± 0.3	71.4 ± 0.5
Saturate	55.8 ± 0.4	65.7 ± 0.4	70.2 ± 0.8	72.6 ± 0.2	71.4 ± 0.7	72.2 ± 0.5	72.4 ± 0.6
Shot nse	14.1 ± 1.2	44.6 ± 0.9	56.1 ± 0.8	61.9 ± 0.6	60.3 ± 0.4	59.8 ± 0.3	62.1 ± 2.8
Snow	49.4 ± 0.8	53.5 ± 0.4	59.8 ± 0.9	65 ± 0.6	65.6 ± 0.4	63.8 ± 0.6	65.9 ± 2.2
Spatter	54.8 ± 1.1	64.9 ± 0.6	72.1 ± 0.3	73.8 ± 0.4	72.9 ± 0.5	73.8 ± 0.5	74.3 ± 0.2
Speckle nse	15.6 ± 1.3	42.3 ± 1	54.2 ± 1.5	62.1 ± 0.6	59.8 ± 0.3	59.6 ± 0.8	62.1 ± 2.7
Zoom blr	45.1 ± 0.7	65.9 ± 0.3	69.1 ± 0.5	71.9 ± 0.3	71.1 ± 0.8	71 ± 0.6	71.2 ± 0.4
Avg.	36.4 ± 0.5	56.6 ± 0.3	62.1 ± 0.2	67 ± 0.2	66 ± 0.4	65.5 ± 0.2	68.5 ± 0.2

Table B.8: CIFAR-100-C ECE (%) results. Shown are the mean and 1 standard deviation.

	Src-only	**AdaBN**	**PL**	**SHOT-IM**	**TENT**	**FR**	**BUFR**
Brightness	6.3 ± 0.3	9.4 ± 0.3	30.2 ± 0.7	27.4 ± 0.4	20.7 ± 0.4	12.4 ± 0.3	12 ± 0.6
Contrast	37.8 ± 2.2	11.4 ± 0.3	40.5 ± 3.4	29.6 ± 0.8	29.5 ± 3.5	14 ± 0.2	12.8 ± 0.5
Defocus blr	16 ± 0.8	9.7 ± 0.3	30.6 ± 0.2	28.2 ± 0.4	21.6 ± 0.3	13.4 ± 0.3	12.7 ± 0.2
Elastic	8 ± 0.1	10.8 ± 0.2	35.9 ± 0.4	33 ± 0.3	25.8 ± 0.1	15.2 ± 0.2	15.3 ± 0.3
Fog	21 ± 0.6	12.2 ± 0.3	39.5 ± 0.6	33.3 ± 0.7	24.8 ± 0.5	15.9 ± 0.3	14 ± 0.6
Frost	14.1 ± 1.1	13.3 ± 0.4	39.7 ± 0.8	34.8 ± 0.4	26.1 ± 0.7	16.3 ± 0.3	15.3 ± 0.2
Gauss. blr	20.5 ± 1.4	10 ± 0.4	31 ± 0.1	28.4 ± 0.2	21.7 ± 0.2	13.5 ± 0.2	12.5 ± 0.3
Gauss. nse	39.3 ± 5.5	16.7 ± 0.2	46.8 ± 0.6	39.8 ± 0.5	30.8 ± 0.7	19.4 ± 0.5	17.5 ± 0.5
Glass blr	15.7 ± 1.1	13.4 ± 0.1	42.5 ± 0.7	37.6 ± 0.3	29.1 ± 0.4	17.9 ± 0.4	17.6 ± 0.6
Impulse nse	35.1 ± 2.6	17.4 ± 0.2	49.3 ± 0.6	41.5 ± 0.7	33.7 ± 0.8	20.5 ± 0.3	15.2 ± 0.2
Jpeg compr.	8.6 ± 0.2	15 ± 0.4	44.1 ± 0.4	38.8 ± 0.5	29.6 ± 0.2	19.1 ± 0.2	18.2 ± 0.5
Motion blr	12.2 ± 0.2	10.4 ± 0.3	33.6 ± 0.6	30.3 ± 0.2	23.2 ± 0.4	14.3 ± 0.3	13.6 ± 0.3
Pixelate	27.5 ± 1	11.6 ± 0.4	35 ± 0.6	30.3 ± 0.4	22.5 ± 0.3	14.2 ± 0.4	13.6 ± 0.4
Saturate	8.8 ± 0.2	9.5 ± 0.3	29.6 ± 0.8	27.4 ± 0.3	21.2 ± 0.6	12.7 ± 0.2	12.3 ± 0.7
Shot nse	37.2 ± 5.9	16 ± 0.2	43.7 ± 0.8	38.1 ± 0.5	30.2 ± 0.8	18.6 ± 0.4	17 ± 0.6
Snow	8.5 ± 0.3	14.4 ± 0.2	40.1 ± 0.9	34.9 ± 0.7	25.7 ± 0.5	17 ± 0.5	14.9 ± 0.1
Spatter	6.7 ± 0.3	9.3 ± 0.1	27.8 ± 0.3	26.2 ± 0.4	20 ± 0.6	12 ± 0.3	11 ± 0.4
Speckle nse	34.5 ± 5.4	17.2 ± 0.3	45.7 ± 1.6	37.9 ± 0.6	30.6 ± 0.2	18.7 ± 0.6	16.8 ± 0.8
Zoom blr	10.5 ± 0.3	9.1 ± 0.2	30.8 ± 0.5	28.2 ± 0.4	21.5 ± 0.7	13.1 ± 0.5	13.2 ± 0.7
Avg.	19.4 ± 0.9	12.5 ± 0.1	37.7 ± 0.2	32.9 ± 0.2	25.7 ± 0.4	15.7 ± 0.1	14.5 ± 0.3

B.3.5 CIFAR-10-C full online results

Tables B.9 and B.10 show the accuracy and ECE results for each individual corruption of CIFAR-10-C when in the online (single epoch) set-up. Again FR achieves the biggest wins on the most severe shifts.

Table B.9: CIFAR-10-C *online* accuracy (%) results. Shown are the mean and 1 standard deviation.

	Src-only	**AdaBN**	**SHOT-IM**	**TENT**	**FR**
Brightness	91.4 ± 0.4	91.6 ± 0.2	92.2 ± 0.4	91.8 ± 0.3	92.8 ± 0.3
Contrast	32.3 ± 1.3	87.1 ± 0.4	87.8 ± 0.5	87.8 ± 0.6	89.8 ± 0.6
Defocus blr	53.1 ± 6.4	88.7 ± 0.5	89.7 ± 0.5	89.1 ± 0.5	90.6 ± 0.5
Elastic	77.6 ± 0.6	78 ± 0.3	80.3 ± 0.6	79.2 ± 0.5	82 ± 0.4
Fog	72.9 ± 2.6	85.9 ± 1.1	87.2 ± 0.5	86.5 ± 0.8	89 ± 0.8
Frost	64.4 ± 2.4	80.7 ± 0.8	83 ± 0.8	81.8 ± 0.8	85.9 ± 0.7
Gauss. blr	35.9 ± 8	88.3 ± 0.7	89.5 ± 0.6	88.8 ± 0.5	90.8 ± 0.6
Gauss. nse	27.7 ± 5.1	68.8 ± 0.9	75.4 ± 0.8	72.3 ± 0.7	80.6 ± 0.6
Glass blr	51.3 ± 1.8	66.7 ± 0.5	70.6 ± 1	68.3 ± 0.6	74.7 ± 0.9
Impulse nse	25.9 ± 3.8	62 ± 1.2	68.8 ± 0.8	65.5 ± 0.7	74.5 ± 0.4
Jpeg compr.	74.9 ± 1	73.9 ± 1.2	78.4 ± 0.9	76.2 ± 0.9	82.2 ± 0.5
Motion blr	66.1 ± 1.7	87 ± 0.1	88.2 ± 0.3	87.6 ± 0.3	89.5 ± 0.2
Pixelate	48.2 ± 2.2	80.5 ± 0.4	83.2 ± 0.7	81.7 ± 0.5	86.7 ± 0.7
Saturate	89.9 ± 0.4	91.9 ± 0.1	92.4 ± 0.1	92.3 ± 0.2	92.8 ± 0.2
Shot nse	34.4 ± 4.9	70.9 ± 1.2	77.7 ± 1.6	74.6 ± 1.3	82.2 ± 0.6
Snow	76.6 ± 1.1	82.6 ± 0.7	84.5 ± 0.9	83.4 ± 0.9	86.8 ± 0.6
Spatter	75 ± 0.8	83.2 ± 0.5	86 ± 0.2	84.6 ± 0.2	88.6 ± 0.2
Speckle nse	40.7 ± 3.7	70.2 ± 0.7	77.2 ± 0.6	74.2 ± 0.6	82.4 ± 0.2
Zoom blr	60.5 ± 5.1	88 ± 0.4	89.4 ± 0.2	88.6 ± 0.3	90.7 ± 0.2
Avg.	57.8 ± 0.7	80.3 ± 0	83.2 ± 0.2	81.8 ± 0.2	85.9 ± 0.3

Table B.10: CIFAR-10-C *online* ECE (%) results. Shown are the mean and 1 standard deviation.

	Src-only	**AdaBN**	**SHOT-IM**	**TENT**	**FR**
Brightness	4.7 ± 0.2	5.4 ± 0.2	5.1 ± 0.3	5.1 ± 0.2	4.9 ± 0.3
Contrast	43.5 ± 2.8	6.8 ± 0.4	8.7 ± 0.5	7.6 ± 0.5	6.1 ± 0.4
Defocus blr	28.2 ± 4	7.1 ± 0.4	6.7 ± 0.3	7 ± 0.3	6.4 ± 0.3
Elastic	12.4 ± 0.7	13.5 ± 0.3	12.7 ± 0.4	12.9 ± 0.5	12.3 ± 0.4
Fog	17.4 ± 2.1	8.4 ± 0.6	8.3 ± 0.3	8.3 ± 0.4	7.3 ± 0.5
Frost	22.7 ± 1.7	11.2 ± 0.7	10.9 ± 0.5	11 ± 0.5	9 ± 0.6
Gauss. blr	40.7 ± 6.2	7.3 ± 0.4	6.8 ± 0.4	7 ± 0.3	6.3 ± 0.4
Gauss. nse	57.6 ± 6.7	19.2 ± 0.7	16 ± 0.6	17.6 ± 0.4	13.2 ± 0.6
Glass blr	31.2 ± 1.3	21.4 ± 0.6	19.6 ± 0.7	20.7 ± 0.5	17.9 ± 0.7
Impulse nse	51.2 ± 4	23.8 ± 0.9	20.6 ± 0.8	22.2 ± 0.4	17.9 ± 0.4
Jpeg compr.	14.6 ± 0.8	16.3 ± 0.9	14 ± 0.5	15.1 ± 0.6	12.2 ± 0.4
Motion blr	21.1 ± 1.3	7.8 ± 0.1	7.6 ± 0.3	7.7 ± 0.2	7 ± 0.2
Pixelate	36.9 ± 2.5	12 ± 0.4	10.8 ± 0.6	11.5 ± 0.5	8.9 ± 0.5
Saturate	5.5 ± 0.3	5.1 ± 0.1	5 ± 0.1	5.1 ± 0.1	4.9 ± 0.2
Shot nse	50.2 ± 5.9	17.9 ± 0.9	14.3 ± 1.1	16 ± 0.8	12 ± 0.5
Snow	14.3 ± 0.5	10.7 ± 0.3	9.9 ± 0.6	10.4 ± 0.6	8.9 ± 0.5
Spatter	16.9 ± 0.8	10.2 ± 0.4	9.1 ± 0.2	9.6 ± 0.2	7.6 ± 0.2
Speckle nse	43.2 ± 4.5	18.8 ± 0.5	14.8 ± 0.5	16.3 ± 0.5	11.9 ± 0.2
Zoom blr	24.4 ± 3.5	7.3 ± 0.3	6.8 ± 0.2	7.1 ± 0.2	6.3 ± 0.1
Avg.	28.2 ± 0.4	12.1 ± 0	10.9 ± 0.1	11.5 ± 0.1	9.5 ± 0.2

B.3.6 CIFAR-100-C full online results

Tables B.11 and B.12 show the accuracy and ECE results for each individual corruption of CIFAR-100-C when in the online (single epoch) set-up. Again FR achieves the biggest wins on the most severe shifts.

Table B.11: CIFAR-100-C *online* accuracy (%) results. Shown are the mean and 1 standard deviation.

	Src-only	**AdaBN**	**SHOT-IM**	**TENT**	**FR**
Brightness	63.2 ± 1.1	66.1 ± 0.4	69.3 ± 0.9	69.9 ± 0.7	69.4 ± 0.4
Contrast	13.9 ± 0.6	61.4 ± 0.5	64.8 ± 0.5	66.6 ± 1.1	64.5 ± 0.3
Defocus blr	35.9 ± 0.7	65.6 ± 0.1	69 ± 0.1	69.4 ± 0.3	68.6 ± 0.2
Elastic	58.5 ± 0.7	60.4 ± 0.2	63.3 ± 0.5	63.7 ± 0.1	63.4 ± 0.3
Fog	36.9 ± 0.5	55.4 ± 0.6	61 ± 0.5	62.5 ± 0.7	61.7 ± 0.5
Frost	41.1 ± 0.9	55.3 ± 0.6	60.5 ± 1	61.8 ± 0.6	60.8 ± 0.8
Gauss. blr	28.2 ± 1	64.3 ± 0.3	68.6 ± 0.2	69 ± 0.6	68.4 ± 0.5
Gauss. nse	11.9 ± 1.2	43.8 ± 0.6	53.5 ± 0.2	55.1 ± 0.5	54.7 ± 0.3
Glass blr	45.1 ± 0.9	53.3 ± 0.6	57.8 ± 0.4	58.2 ± 0.5	57.9 ± 0.5
Impulse nse	7.2 ± 0.8	40.8 ± 0.5	50.2 ± 0.4	50.9 ± 0.7	51.7 ± 0.8
Jpeg compr.	48.6 ± 0.9	49.8 ± 0.7	56 ± 0.2	57.2 ± 0.2	56.6 ± 0.6
Motion blr	45.1 ± 0.5	63.4 ± 0.2	66.4 ± 0.4	66.7 ± 0.6	66 ± 0.3
Pixelate	22.3 ± 0.4	59.4 ± 0.6	65.1 ± 0.7	67.1 ± 0.4	65.6 ± 0.6
Saturate	55.8 ± 0.4	65.7 ± 0.4	69.5 ± 0.6	69.5 ± 0.6	69.3 ± 0.4
Shot nse	14.1 ± 1.2	44.6 ± 0.9	54.9 ± 0.1	55.5 ± 0.4	56.4 ± 0.3
Snow	49.4 ± 0.8	53.5 ± 0.4	59.7 ± 1.1	61.6 ± 0.8	60.5 ± 0.5
Spatter	54.8 ± 1.1	64.9 ± 0.6	71.3 ± 0.5	70.6 ± 0.6	71.6 ± 0.7
Speckle nse	15.6 ± 1.3	42.3 ± 1	54.2 ± 0.3	54.9 ± 0.3	55.8 ± 0.6
Zoom blr	45.1 ± 0.7	65.9 ± 0.3	68.9 ± 0.6	69 ± 0.4	68.8 ± 0.3
Avg.	36.4 ± 0.5	56.6 ± 0.3	62.3 ± 0.3	63.1 ± 0.3	62.7 ± 0.3

Table B.12: CIFAR-100-C *online* ECE (%) results. Shown are the mean and 1 standard deviation.

	Src-only	**AdaBN**	**SHOT-IM**	**TENT**	**FR**
Brightness	6.3 ± 0.3	11.4 ± 0.1	11.6 ± 0.4	11.9 ± 0.2	10.9 ± 0.2
Contrast	37.8 ± 2.2	12.5 ± 0.2	14.6 ± 0.3	14.1 ± 0.6	12.5 ± 0.2
Defocus blr	16 ± 0.8	11.4 ± 0.2	11.3 ± 0.2	11.8 ± 0.2	11.4 ± 0.3
Elastic	8 ± 0.1	12.7 ± 0.2	12.9 ± 0.2	13.4 ± 0.3	13 ± 0.2
Fog	21 ± 0.6	13.8 ± 0.2	13.9 ± 0.2	14.4 ± 0.2	13.6 ± 0.3
Frost	14.1 ± 1.1	14.5 ± 0.2	14.5 ± 0.6	14.8 ± 0.3	13.9 ± 0.4
Gauss. blr	20.5 ± 1.4	11.9 ± 0.3	11.6 ± 0.4	11.9 ± 0.2	11.9 ± 0.4
Gauss. nse	39.3 ± 5.5	17.7 ± 0.3	16.7 ± 0.3	17.2 ± 0.7	16.5 ± 0.3
Glass blr	15.7 ± 1.1	15 ± 0.1	15.2 ± 0.5	16.1 ± 0.3	15.1 ± 0.1
Impulse nse	35.1 ± 2.6	18.4 ± 0.2	18.1 ± 0.2	19.4 ± 0.5	17.9 ± 0.3
Jpeg compr.	8.6 ± 0.2	16.2 ± 0.3	15.9 ± 0.1	16.4 ± 0.4	16.2 ± 0.2
Motion blr	12.2 ± 0.2	12.2 ± 0.2	12.2 ± 0.3	12.9 ± 0.2	12.5 ± 0.2
Pixelate	27.5 ± 1	13 ± 0.3	12.5 ± 0.3	12.4 ± 0.1	12.3 ± 0.2
Saturate	8.8 ± 0.2	11.4 ± 0.1	11.3 ± 0.3	11.7 ± 0.4	11.3 ± 0.4
Shot nse	37.2 ± 5.9	17.2 ± 0.3	16.4 ± 0.2	17.5 ± 0.7	16.2 ± 0.3
Snow	8.5 ± 0.3	15.6 ± 0.2	15 ± 0.4	14.7 ± 0.3	14.8 ± 0.1
Spatter	6.7 ± 0.3	11.4 ± 0.2	10.7 ± 0.2	11.3 ± 0.3	10.7 ± 0.3
Speckle nse	34.5 ± 5.4	18.2 ± 0.4	16.5 ± 0.4	17.5 ± 0.3	16.1 ± 0.4
Zoom blr	10.5 ± 0.3	11.2 ± 0.2	11.2 ± 0.3	11.6 ± 0.3	11.4 ± 0.1
Avg.	19.4 ± 0.9	14 ± 0.1	13.8 ± 0.1	14.3 ± 0.1	13.6 ± 0.1

Appendix C

Unit-Level Surprise

C.1 Surprise

The *surprisal*, also known as the information content, of an event $X = x$, with $X \sim P(X)$, is defined as $\log(1/P(x)) = -\log(P(x))$. This quantity is large when $P(x)$ is small and intuitively represents how surprised we are to see $X = x$. *Surprise* can be used to refer to different mathematical quantities. For example, the entropy, $H(X) = -\sum P(x) \log P(x)$, can also be called the surprise and mathematically is the expected surprisal. To emphasise our intuitive reasoning, throughout Chapter 5 we refer to the KL divergence $D_{KL}(Q||P) = H(Q,P) - H(Q)$ as surprise. This quantity can also be called the Bayesian surprise, information gain, or asymmetric surprise (Eslami et al., 2018; Itti and Baldi, 2009). We measure unit-level surprise, $s(A)$, by calculating the KL divergence between source activation distribution $P(A)$ and target activation distribution $Q(A)$, i.e. $s(A) = D_{KL}(Q(A)||P(A))$ (Itti and Baldi, 2009; MacKay, 1992).

To relate the KL divergence to surprisal we consider a second random variable $Y = y$, $Y \sim Q(Y)$. Sometimes we may assume that we will receive samples from $P(X)$ but actually receive samples from $Q(Y)$, for example if an unexpected domain shift occurs[1]. The amount of *additional* surprisal we receive on account of our assumption when we observe sample y is $\log(1/P(X = y)) - \log(1/Q(Y = y))$. The KL divergence $D_{KL}(Q||P)$ is the expected value of this quantity over Q, that is, the expected additional surprisal we receive due to incorrectly assuming that we will receive samples from P.

[1] Although in this situation only the distribution changes with the random variable staying the same, the mathematical explanation is clearer using different random variables.

C.2 Extended Results

This section provides full per-shift results comparing SGD, FlexTune and our surprise based update rule for different numbers of sample per class (shots).

Table C.1: 2-shot accuracy across shifts: training different layers of a CNN.

	Conv1	Conv2	Conv3	FC1	FC2	FC1 + FC2
H1	0.0 ± 0.0	0.0 ± 0.1	7.2 ± 4.9	65.3 ± 14.1	95.2 ± 1.4	77.8 ± 1.4
H2	0.0 ± 0.0	0.0 ± 0.0	5.0 ± 0.8	45.7 ± 8.8	80.5 ± 5.4	71.4 ± 4.2
H3	0.0 ± 0.0	0.0 ± 0.0	10.8 ± 7.3	49.4 ± 4.3	92.9 ± 2.3	81.4 ± 5.7
Crystals	69.9 ± 4.9	52.1 ± 0.9	51.0 ± 0.9	50.0 ± 0.6	47.8 ± 0.7	49.6 ± 1.1
Fog	86.5 ± 3.2	84.5 ± 0.4	84.0 ± 0.5	82.3 ± 0.8	78.8 ± 1.2	81.6 ± 0.4
Gauss. Blur	82.7 ± 0.3	80.4 ± 2.0	79.0 ± 0.3	74.9 ± 0.6	71.5 ± 1.0	74.1 ± 1.9
Grass	81.4 ± 0.8	21.6 ± 7.2	8.1 ± 0.7	7.3 ± 0.6	7.2 ± 0.5	7.7 ± 0.5
Imp. Noise	87.4 ± 0.9	86.0 ± 0.2	83.4 ± 1.4	83.2 ± 0.9	80.7 ± 0.6	81.9 ± 1.4
Sky	74.4 ± 3.9	54.4 ± 0.5	33.2 ± 2.8	18.4 ± 0.7	13.7 ± 2.7	18.4 ± 3.3
Stripe	74.3 ± 0.9	58.2 ± 4.8	58.2 ± 1.4	45.7 ± 2.1	36.7 ± 1.0	47.9 ± 2.6
Avg High	0.0 ± 0.0	0.0 ± 0.0	7.7 ± 4.0	53.5 ± 3.4	89.6 ± 1.5	76.9 ± 2.9
Avg Low	79.5 ± 1.3	62.4 ± 2.1	56.7 ± 0.5	51.7 ± 0.6	48.1 ± 0.5	51.6 ± 0.7
Avg All	55.7 ± 0.9	43.7 ± 1.5	42.0 ± 1.4	52.2 ± 0.8	60.5 ± 0.3	59.2 ± 0.9

Table C.2: 2-shot accuracy across shifts: comparison of different responses. Zero-shot is the accuracy before any updates are performed.

	Zero-shot	SGD	FlexTune	Upd. Rule
H1	0.0 ± 0.0	56.7 ± 4.2	95.2 ± 1.4	33.8 ± 21.4
H2	0.0 ± 0.0	51.9 ± 5.7	80.5 ± 5.4	34.3 ± 11.2
H3	0.0 ± 0.0	57.6 ± 5.3	92.9 ± 2.3	41.4 ± 9.4
Crystals	46.3 ± 0.4	55.3 ± 1.3	69.9 ± 4.9	68.5 ± 5.4
Fog	78.4 ± 0.4	83.6 ± 1.2	87.1 ± 2.3	86.6 ± 3.2
Gauss. Blur	60.4 ± 2.1	81.2 ± 0.6	82.7 ± 0.3	81.8 ± 1.8
Grass	5.8 ± 0.2	20.1 ± 12.0	81.4 ± 0.8	81.4 ± 0.8
Imp. Noise	76.9 ± 0.9	87.9 ± 0.2	87.4 ± 0.9	87.7 ± 0.5
Sky	4.1 ± 0.5	35.2 ± 4.9	74.4 ± 3.9	74.4 ± 3.9
Stripe	16.3 ± 1.1	70.0 ± 1.4	74.3 ± 0.9	72.8 ± 2.1
Avg High	0.0 ± 0.0	55.4 ± 1.4	89.6 ± 1.5	36.5 ± 6.7
Avg Low	41.2 ± 0.3	61.9 ± 2.4	79.6 ± 1.2	79.0 ± 1.1
Avg All	28.8 ± 0.2	60.0 ± 1.9	82.6 ± 0.7	66.3 ± 2.5

Table C.3: 5-shot accuracy across shifts: training different layers of a CNN.

	Conv1	Conv2	Conv3	FC1	FC2	FC1 + FC2
H1	0.0 ± 0.0	0.1 ± 0.1	29.8 ± 8.2	82.8 ± 7.3	96.4 ± 1.0	91.9 ± 2.0
H2	0.0 ± 0.0	0.5 ± 0.9	26.1 ± 6.2	76.7 ± 6.6	91.3 ± 1.5	86.2 ± 3.8
H3	0.0 ± 0.0	0.3 ± 0.4	20.3 ± 6.1	82.3 ± 5.3	96.0 ± 0.6	93.0 ± 2.8
Crystals	78.1 ± 1.7	57.5 ± 0.9	52.3 ± 0.8	51.5 ± 0.5	49.2 ± 0.8	51.4 ± 0.6
Fog	89.6 ± 0.3	86.5 ± 0.7	84.8 ± 0.5	83.1 ± 0.6	81.3 ± 0.6	82.6 ± 0.6
Gauss. Blur	84.0 ± 0.4	82.5 ± 0.7	81.7 ± 0.6	78.3 ± 0.9	75.9 ± 0.5	78.1 ± 0.4
Grass	82.0 ± 2.2	41.7 ± 7.4	10.3 ± 1.3	8.4 ± 0.9	7.3 ± 0.5	9.4 ± 0.4
Imp. Noise	88.6 ± 0.3	86.8 ± 0.1	85.7 ± 0.3	84.3 ± 0.9	81.5 ± 0.4	83.9 ± 0.4
Sky	79.6 ± 0.7	65.9 ± 0.5	44.3 ± 1.9	27.9 ± 1.8	18.3 ± 2.4	28.8 ± 1.2
Stripe	77.1 ± 1.8	70.9 ± 2.6	70.6 ± 0.6	60.4 ± 1.6	47.0 ± 1.3	60.9 ± 2.5
Avg High	0.0 ± 0.0	0.3 ± 0.5	25.4 ± 3.8	80.6 ± 4.6	94.5 ± 0.9	90.4 ± 1.1
Avg Low	82.7 ± 0.4	70.2 ± 1.2	61.4 ± 0.2	56.3 ± 0.2	51.5 ± 0.3	56.4 ± 0.3
Avg All	57.9 ± 0.3	49.3 ± 0.8	50.6 ± 1.2	63.6 ± 1.2	64.4 ± 0.3	66.6 ± 0.1

Table C.4: 5-shot accuracy across shifts: comparison of different responses. Zero-shot is the accuracy before any updates are performed.

	Zero-shot	SGD	FlexTune	Upd. Rule
H1	0.0 ± 0.0	68.0 ± 13.8	96.4 ± 1.0	80.3 ± 4.8
H2	0.0 ± 0.0	78.0 ± 2.0	91.3 ± 1.5	77.3 ± 7.2
H3	0.0 ± 0.0	77.1 ± 5.0	96.0 ± 0.6	79.9 ± 9.1
Crystals	46.3 ± 0.4	57.5 ± 3.8	78.1 ± 1.7	77.0 ± 3.4
Fog	78.4 ± 0.4	86.3 ± 0.2	89.6 ± 0.3	89.6 ± 0.3
Gauss. Blur	60.4 ± 2.1	84.3 ± 0.7	84.0 ± 0.4	83.9 ± 0.3
Grass	5.8 ± 0.2	50.9 ± 6.8	82.0 ± 2.2	82.3 ± 2.6
Imp. Noise	76.9 ± 0.9	88.3 ± 0.1	88.6 ± 0.3	87.6 ± 0.3
Sky	4.1 ± 0.5	56.6 ± 4.7	79.6 ± 0.7	80.8 ± 0.9
Stripe	16.3 ± 1.1	75.4 ± 1.0	77.1 ± 1.8	79.3 ± 0.8
Avg High	0.0 ± 0.0	74.4 ± 5.2	94.5 ± 0.9	79.2 ± 4.2
Avg Low	41.2 ± 0.3	71.3 ± 2.1	82.7 ± 0.4	82.9 ± 0.5
Avg All	28.8 ± 0.2	72.2 ± 2.9	86.3 ± 0.5	81.8 ± 1.2

Table C.5: 10-shot accuracy across shifts: training different layers of a CNN.

	Conv1	Conv2	Conv3	FC1	FC2	FC1 + FC2
H1	0.0 ± 0.0	0.5 ± 0.8	50.1 ± 5.1	91.3 ± 2.0	96.1 ± 0.8	94.5 ± 0.6
H2	0.0 ± 0.0	0.9 ± 0.8	45.7 ± 8.8	88.5 ± 1.7	92.8 ± 0.3	89.0 ± 2.2
H3	0.0 ± 0.0	0.3 ± 0.4	46.6 ± 16.0	89.7 ± 5.3	95.0 ± 1.1	92.8 ± 1.8
Crystals	80.5 ± 1.5	59.9 ± 3.1	54.4 ± 0.1	52.6 ± 0.7	50.6 ± 0.8	52.6 ± 0.8
Fog	90.1 ± 0.1	87.8 ± 0.5	85.6 ± 0.3	84.1 ± 0.2	82.1 ± 1.3	84.3 ± 0.4
Gauss. Blur	85.0 ± 0.4	83.9 ± 1.1	82.6 ± 0.4	81.4 ± 0.9	77.9 ± 2.3	80.7 ± 0.8
Grass	83.8 ± 0.9	57.1 ± 3.2	15.1 ± 0.8	10.0 ± 0.9	7.8 ± 0.1	11.2 ± 1.0
Imp. Noise	89.1 ± 0.0	87.2 ± 0.4	86.0 ± 0.3	84.7 ± 0.6	82.7 ± 0.1	84.5 ± 0.2
Sky	83.3 ± 1.5	71.4 ± 1.0	52.5 ± 0.5	36.7 ± 1.2	24.5 ± 2.4	38.0 ± 0.2
Stripe	82.4 ± 0.7	76.1 ± 2.8	76.1 ± 1.1	69.2 ± 0.2	56.8 ± 0.8	68.1 ± 1.8
Avg High	0.0 ± 0.0	0.6 ± 0.6	47.5 ± 6.4	89.8 ± 1.2	94.6 ± 0.6	92.1 ± 0.5
Avg Low	84.9 ± 0.3	74.8 ± 1.3	64.6 ± 0.2	59.8 ± 0.4	54.6 ± 0.5	59.9 ± 0.4
Avg All	59.4 ± 0.2	52.5 ± 0.8	59.5 ± 1.8	68.8 ± 0.5	66.6 ± 0.4	69.6 ± 0.4

Table C.6: 10-shot accuracy across shifts: comparison of different responses. Zero-shot is the accuracy before any updates are performed.

	Zero-shot	SGD	FlexTune	Upd. Rule
H1	0.0 ± 0.0	84.8 ± 0.5	96.1 ± 0.8	90.4 ± 3.0
H2	0.0 ± 0.0	79.9 ± 1.7	92.8 ± 0.3	88.3 ± 2.0
H3	0.0 ± 0.0	87.1 ± 2.7	95.0 ± 1.1	89.7 ± 5.1
Crystals	46.3 ± 0.4	61.2 ± 1.6	80.5 ± 1.5	79.8 ± 0.9
Fog	78.4 ± 0.4	87.4 ± 0.2	90.1 ± 0.1	90.1 ± 0.1
Gauss. Blur	60.4 ± 2.1	85.7 ± 0.6	85.0 ± 0.4	85.3 ± 1.0
Grass	5.8 ± 0.2	69.2 ± 2.4	83.8 ± 0.9	84.3 ± 1.3
Imp. Noise	76.9 ± 0.9	88.2 ± 0.4	89.1 ± 0.0	88.0 ± 0.2
Sky	4.1 ± 0.5	66.8 ± 1.1	83.3 ± 1.5	83.4 ± 1.5
Stripe	16.3 ± 1.1	78.8 ± 1.1	82.4 ± 0.7	82.5 ± 0.9
Avg High	0.0 ± 0.0	84.0 ± 0.5	94.6 ± 0.6	89.5 ± 1.2
Avg Low	41.2 ± 0.3	76.8 ± 0.7	84.9 ± 0.3	84.8 ± 0.5
Avg All	28.8 ± 0.2	78.9 ± 0.5	87.8 ± 0.4	86.2 ± 0.6

Table C.7: 20-shot accuracy across shifts: training different layers of a CNN.

	Conv1	Conv2	Conv3	FC1	FC2	FC1 + FC2
H1	0.0 ± 0.0	2.1 ± 3.4	66.4 ± 8.9	95.7 ± 0.9	97.2 ± 0.7	95.1 ± 0.6
H2	0.0 ± 0.0	5.6 ± 0.9	64.9 ± 8.7	90.3 ± 1.5	94.8 ± 0.6	91.6 ± 1.1
H3	0.0 ± 0.0	9.5 ± 1.3	69.3 ± 6.2	93.5 ± 2.2	96.1 ± 1.0	95.3 ± 0.9
Crystals	83.7 ± 0.5	65.6 ± 2.3	56.7 ± 0.2	54.6 ± 0.3	51.5 ± 0.7	54.6 ± 0.4
Fog	90.4 ± 0.4	88.4 ± 0.1	86.1 ± 1.0	85.2 ± 0.2	84.0 ± 0.3	85.1 ± 0.2
Gauss. Blur	86.6 ± 0.4	85.0 ± 1.4	83.4 ± 0.6	82.2 ± 0.8	81.0 ± 0.9	82.3 ± 1.1
Grass	85.6 ± 1.0	67.6 ± 2.8	20.7 ± 1.8	13.4 ± 1.0	8.3 ± 0.3	14.1 ± 0.9
Imp. Noise	89.1 ± 0.6	87.7 ± 0.2	86.6 ± 0.2	85.2 ± 0.5	83.7 ± 0.5	84.9 ± 0.2
Sky	84.8 ± 1.0	76.3 ± 0.7	59.5 ± 0.4	45.1 ± 1.5	31.7 ± 0.7	47.0 ± 0.7
Stripe	83.9 ± 1.3	79.7 ± 2.0	78.9 ± 0.9	74.7 ± 0.2	62.2 ± 0.9	75.3 ± 0.3
Avg High	0.0 ± 0.0	5.7 ± 0.8	66.8 ± 2.9	93.2 ± 1.2	96.1 ± 0.3	94.0 ± 0.2
Avg Low	86.3 ± 0.3	78.6 ± 1.1	67.4 ± 0.2	62.9 ± 0.3	57.5 ± 0.4	63.3 ± 0.3
Avg All	60.4 ± 0.2	56.7 ± 0.9	67.3 ± 0.8	72.0 ± 0.5	69.0 ± 0.4	72.5 ± 0.3

Table C.8: 20-shot accuracy across shifts: comparison of different responses. Zero-shot is the accuracy before any updates are performed.

	Zero-shot	SGD	FlexTune	Upd. Rule
H1	0.0 ± 0.0	89.3 ± 1.6	97.2 ± 0.7	95.5 ± 1.2
H2	0.0 ± 0.0	85.4 ± 3.1	94.8 ± 0.6	89.9 ± 1.4
H3	0.0 ± 0.0	90.3 ± 0.6	96.3 ± 0.9	93.5 ± 2.4
Crystals	46.3 ± 0.4	66.5 ± 1.8	83.7 ± 0.5	82.1 ± 0.9
Fog	78.4 ± 0.4	88.1 ± 0.3	90.4 ± 0.4	90.4 ± 0.4
Gauss. Blur	60.4 ± 2.1	86.0 ± 0.5	86.6 ± 0.4	86.9 ± 0.4
Grass	5.8 ± 0.2	77.0 ± 1.2	85.6 ± 1.0	85.5 ± 0.2
Imp. Noise	76.9 ± 0.9	88.5 ± 0.1	89.1 ± 0.6	88.3 ± 0.3
Sky	4.1 ± 0.5	73.3 ± 0.8	84.8 ± 1.0	84.7 ± 0.9
Stripe	16.3 ± 1.1	81.6 ± 0.4	83.9 ± 1.3	84.7 ± 1.1
Avg High	0.0 ± 0.0	88.3 ± 1.4	96.1 ± 0.2	93.0 ± 1.1
Avg Low	41.2 ± 0.3	80.1 ± 0.3	86.3 ± 0.3	86.1 ± 0.3
Avg All	28.8 ± 0.2	82.6 ± 0.2	89.2 ± 0.2	88.2 ± 0.4

Table C.9: 50-shot accuracy across shifts: training different layers of a CNN.

	Conv1	Conv2	Conv3	FC1	FC2	FC1 + FC2
H1	0.0 ± 0.0	25.8 ± 24.0	83.5 ± 5.3	97.0 ± 0.2	97.2 ± 0.5	95.9 ± 0.9
H2	0.0 ± 0.0	32.8 ± 4.5	84.4 ± 1.9	94.8 ± 0.7	95.2 ± 0.9	93.5 ± 1.2
H3	0.0 ± 0.0	28.1 ± 17.4	85.0 ± 4.7	95.5 ± 1.9	97.6 ± 0.5	96.1 ± 0.6
Crystals	85.2 ± 0.5	72.9 ± 1.2	59.3 ± 0.3	57.8 ± 0.3	53.1 ± 0.6	57.8 ± 0.2
Fog	90.6 ± 0.2	89.3 ± 0.1	87.7 ± 0.2	86.7 ± 0.2	85.1 ± 0.3	86.6 ± 0.2
Gauss. Blur	87.9 ± 0.5	87.3 ± 0.3	85.8 ± 0.4	84.6 ± 0.2	83.2 ± 0.6	84.6 ± 0.2
Grass	87.1 ± 0.6	75.2 ± 1.9	29.8 ± 1.6	19.3 ± 0.8	9.0 ± 0.2	19.7 ± 0.6
Imp. Noise	89.4 ± 0.3	88.3 ± 0.2	86.9 ± 0.3	85.9 ± 0.3	84.8 ± 0.2	85.9 ± 0.2
Sky	86.7 ± 0.5	80.9 ± 0.3	66.4 ± 0.6	54.3 ± 1.8	38.2 ± 0.8	56.7 ± 0.4
Stripe	87.4 ± 0.3	84.6 ± 0.7	83.2 ± 0.5	79.9 ± 0.9	67.6 ± 0.8	79.8 ± 0.8
Avg High	0.0 ± 0.0	28.9 ± 12.2	84.3 ± 3.7	95.8 ± 0.6	96.7 ± 0.6	95.2 ± 0.2
Avg Low	87.8 ± 0.1	82.6 ± 0.4	71.3 ± 0.3	66.9 ± 0.3	60.1 ± 0.3	67.3 ± 0.0
Avg All	61.4 ± 0.1	66.5 ± 3.8	75.2 ± 0.9	75.6 ± 0.4	71.1 ± 0.3	75.7 ± 0.1

Table C.10: 50-shot accuracy across shifts: comparison of different responses. Zero-shot is the accuracy before any updates are performed.

	Zero-shot	SGD	FlexTune	Upd. Rule
H1	0.0 ± 0.0	93.3 ± 0.5	97.2 ± 0.5	96.8 ± 0.3
H2	0.0 ± 0.0	89.9 ± 0.9	95.2 ± 0.9	94.4 ± 0.6
H3	0.0 ± 0.0	93.8 ± 1.4	97.6 ± 0.5	95.5 ± 2.0
Crystals	46.3 ± 0.4	73.2 ± 0.9	85.2 ± 0.5	83.5 ± 0.5
Fog	78.4 ± 0.4	88.9 ± 0.3	90.6 ± 0.2	90.5 ± 0.3
Gauss. Blur	60.4 ± 2.1	87.3 ± 0.4	87.9 ± 0.5	87.4 ± 0.3
Grass	5.8 ± 0.2	81.8 ± 0.4	87.1 ± 0.6	86.2 ± 0.4
Imp. Noise	76.9 ± 0.9	88.9 ± 0.2	89.4 ± 0.3	88.5 ± 0.5
Sky	4.1 ± 0.5	79.1 ± 0.5	86.7 ± 0.5	86.9 ± 0.3
Stripe	16.3 ± 1.1	84.8 ± 1.3	87.4 ± 0.3	87.5 ± 0.4
Avg High	0.0 ± 0.0	92.3 ± 0.6	96.7 ± 0.6	95.6 ± 0.6
Avg Low	41.2 ± 0.3	83.4 ± 0.2	87.8 ± 0.1	87.2 ± 0.1
Avg All	28.8 ± 0.2	86.1 ± 0.3	90.4 ± 0.2	89.7 ± 0.1

Table C.11: 2000-shot (all data) accuracy across shifts: training different layers of a CNN.

	Conv1	Conv2	Conv3	FC1	FC2	FC1 + FC2
H1	0.0 ± 0.1	80.0 ± 9.9	97.6 ± 0.2	98.7 ± 0.1	98.6 ± 0.1	98.8 ± 0.1
H2	2.0 ± 3.2	78.2 ± 8.1	96.7 ± 0.2	98.0 ± 0.2	98.0 ± 0.2	98.3 ± 0.2
H3	0.0 ± 0.0	81.9 ± 2.8	97.7 ± 0.4	98.6 ± 0.3	98.7 ± 0.2	98.8 ± 0.0
Crystals	88.0 ± 0.3	85.5 ± 0.3	72.6 ± 0.2	69.4 ± 0.6	57.4 ± 0.4	68.2 ± 0.3
Fog	91.2 ± 0.3	91.2 ± 0.1	90.6 ± 0.1	90.2 ± 0.1	88.0 ± 0.3	89.6 ± 0.1
Gauss. Blur	90.1 ± 0.1	90.9 ± 0.1	90.3 ± 0.0	90.2 ± 0.1	86.7 ± 0.1	89.7 ± 0.2
Grass	89.1 ± 0.2	86.1 ± 0.5	56.6 ± 1.1	40.8 ± 0.9	17.0 ± 0.3	38.8 ± 1.0
Imp. Noise	89.8 ± 0.1	89.7 ± 0.2	88.2 ± 0.1	88.3 ± 0.2	86.6 ± 0.3	87.6 ± 0.2
Sky	89.0 ± 0.1	88.2 ± 0.3	81.7 ± 0.1	75.2 ± 0.3	48.8 ± 0.4	73.7 ± 0.4
Stripe	89.9 ± 0.1	90.8 ± 0.3	90.3 ± 0.1	89.0 ± 0.0	76.3 ± 0.5	88.3 ± 0.2
Avg High	0.7 ± 1.1	80.0 ± 5.1	97.4 ± 0.2	98.5 ± 0.1	98.4 ± 0.1	98.6 ± 0.0
Avg Low	89.6 ± 0.1	88.9 ± 0.0	81.5 ± 0.2	77.6 ± 0.1	65.8 ± 0.1	76.5 ± 0.1
Avg All	62.9 ± 0.3	86.3 ± 1.5	86.2 ± 0.1	83.8 ± 0.1	75.6 ± 0.1	83.2 ± 0.1

Table C.12: 2000-shot (all data) accuracy across shifts: comparison of different responses. Zero-shot is the accuracy before any updates are performed.

	Zero-shot	SGD	FlexTune	Upd. Rule
H1	0.0 ± 0.0	97.9 ± 0.1	98.8 ± 0.1	98.7 ± 0.2
H2	0.0 ± 0.0	97.2 ± 0.4	98.3 ± 0.1	98.1 ± 0.2
H3	0.0 ± 0.0	98.2 ± 0.1	98.8 ± 0.0	98.7 ± 0.3
Crystals	46.3 ± 0.4	87.7 ± 0.3	88.0 ± 0.3	88.3 ± 0.2
Fog	78.4 ± 0.4	90.9 ± 0.0	91.3 ± 0.2	90.8 ± 0.2
Gauss. Blur	60.4 ± 2.1	90.5 ± 0.1	90.9 ± 0.1	90.6 ± 0.1
Grass	5.8 ± 0.2	89.2 ± 0.1	89.1 ± 0.2	89.5 ± 0.4
Imp. Noise	76.9 ± 0.9	89.3 ± 0.2	89.8 ± 0.1	89.5 ± 0.2
Sky	4.1 ± 0.5	89.1 ± 0.2	89.0 ± 0.1	89.4 ± 0.1
Stripe	16.3 ± 1.1	90.6 ± 0.3	90.8 ± 0.3	90.0 ± 0.2
Avg High	0.0 ± 0.0	97.8 ± 0.1	98.6 ± 0.0	98.5 ± 0.1
Avg Low	41.2 ± 0.3	89.6 ± 0.1	89.8 ± 0.0	89.7 ± 0.1
Avg All	28.8 ± 0.2	92.1 ± 0.1	92.5 ± 0.0	92.4 ± 0.0