The Effects of Hybrid Neural Networks on Meta-Learning Objectives

Franz Anthony Varela

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

Master of Science

University of Washington

2022

Committee:

Michael Stiber

Erika Parsons

Clark Olson

Program Authorized to Offer Degree:

Computing & Software Systems

University of Washington

Abstract

The Effects of Hybrid Neural Networks on Meta-Learning Objectives

Franz Anthony Varela

Chair of the Supervisory Committee:
Michael Stiber
Computing & Software Systems

Historically, deep neural networks do not generalize well when they are trained solely on a dataset/task's objective, despite the plethora of data and computing available in the modern digital era. We propose that this is due, at least partially, to the model representations being inflexible. In this paper, we experiment with a hybrid neural network architecture that has an unsupervised model at its head (the Knowledge Representation module) and a supervised model at its tail (the Task Inference module) with the idea that we can supplement the learning of a set of related tasks with a reusable knowledge base. We analyze the two-part model in the contexts of transfer learning, few-shot learning, and curriculum learning, and train on the MNIST and SVHN datasets. The results of the experiment demonstrate that our architecture on average achieves a similar test accuracy as the end-to-end baselines, and sometimes marginally better in certain experiments depending on the sub-network combination.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

AE: Autoencoder.

E2E: End-to-End, describing a model whose components are all trained jointly.

MNIST: The Modified National Institute of Standards and Technology dataset, a collection of preprocessed decimal digit images with approximately 6000 samples per class.

SVHN: The Street View House Number dataset, a colored image collection of house number digits from Google Street View.

KR: Knowledge Representation, the sub-network solely responsible for learning feature representations without the need for labels.

TI: Task Inference, the sub-network solely responsible for learning from the labels of the dataset (i.e. learning the task).

KRTI: Knowledge Representation and Task Inference model.

BAE: Basic AutoEncoder.

BVAE: Basic Variational AutoEncoder.

BHAE: Basic Hopfield AutoEncoder.

BSNN: Basic Self-Normalizing Neural Network.

BSHN: Basic Hopfield Neural Network.

ACKNOWLEDGMENTS

I am utmost grateful first and foremost to my committee, in particular Prof. Stiber for his support and constructive criticism that helped narrow the scope of my paper into the feasible project that it is currently. Thank you to Shruti Chakraborty and Tamy Do for their peer-review and helpful comments on the paper and figures. Last but not least, Yerlan Idelbayev, who inspired me to pursue graduate school to begin with.

DEDICATION

To all of my loved ones.

Chapter 1

INTRODUCTION

The majority of modern deep learning is still focused on achieving superhuman performance on benchmarks [1]. Some may misinterpret this as the models getting "smarter" when in actuality it is primarily driven by the past decade's surge in compute and data availability [2] [3], and their brittleness can be exposed with corner-case samples. For example, even though AI has shown to surpass humans in games like Go [4] [5] and Defense of the Ancients (DotA) [6], when factoring their compute and training time requirements (millions of games, hundreds of millions of human hours) it only shows how far near-infinite resources can take an agent in its own improvement at a particular skill. Even OpenAI's GPT-3 [7], despite its enormous parameter count and seemingly impressive generalizability, has inhuman limitations [8] [9] that show a gap in learning between humans and machines that we have still yet to decipher.

We should move beyond this "narrow AI" perspective, and redefine the objectives of discovering true artificial intelligence to more closely emulate the higher cognition functions found in humans for Artificial General Intelligence (AGI) [10]. Indeed, it can be argued that the largest strides in AI performance can be found in models that replicate biological processes, and scale them up; some examples include neural networks [11] themselves, convolutional layers [12] [13], and skip connections [14] [15] [16].

This paper attempts to address one of the largest critiques of modern AI in a relatively small scope: its inability to accurately approximate even that of a human child's exceptional sample efficiency and generalizability. Why is it so difficult to create a model that can learn the concept of a number after only seeing an image of it a few (~ 10) times and that can effectively use its previously learned features to expedite the learning process on a related

dataset or task? In the meta-learning literature, these objectives are known as few-shot learning [17] and transfer learning [18] respectively.

We hypothesize that this difficulty is caused by a model's entanglement of its learned representations of the inputs with the task-specific objectives, such as training labels in the case of classification, because the models are structured to optimize for only a specific task and therefore the learned features are not flexible enough to be reused in other related problems. We propose to split the regular end-to-end deep neural network into two sub-models — namely the knowledge representation (KR) and task inference (TI) models – one can achieve better generalizability, as the KR can be reused across different tasks, while training for a particular task involves creating and training a new TI network jointly with same KR; altogether, the hybrid architecture is known as the KRTI. Figure 1.1 is a diagram showcasing the general architecture of the KRTI, where the KR is an autoencoder and the TI is a regular classifier network. Both networks could stand alone separately and be evaluated on test loss and test accuracy respectively, but the KRTI as a whole is dominated in the short term by the task that the TI sub-network learns.

Unlike traditional modern neural networks, where the model is trained end-to-end directly on the dataset labels, we instead have two different objectives influencing the weights of certain model sections. One can imagine that in this hybrid model setup, the KR acts as a knowledge base that learns features from the training distribution; then the TI is trained with the labels and the inputs fed through the KR, as the TI learns the weights to "query" the appropriate data from the KR. The key idea is that the disentanglement of the knowledge representation from the task is an important distinction that opens up the opportunity to more directly tackle the meta-learning problems of transfer and few-shot learning. We explore a variety of scenarios – such as model choices for the KR and TI, pre-training versus lifelong/joint training, performance on restricted training samples – and analyze the pros and cons of these hybrid models when compared to a singular end-to-end network. In the case where we train both components of the KRTI jointly, there are two objectives that

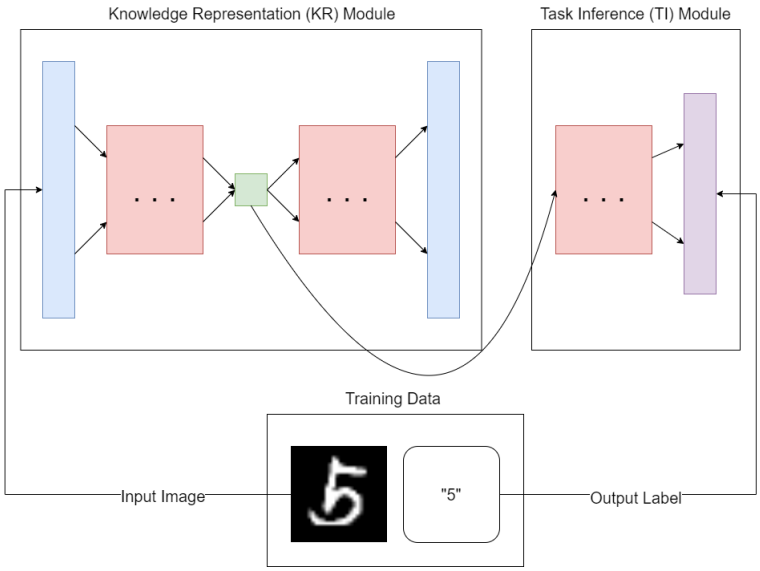


Figure 1.1: Diagram of the KRTI architecture and forward pass signal flow. There are two parts to a single training step in the KRTI, one for each sub-network. In this particular example, the KR is an autoencoder, with its input and output layers in blue, the hidden layers of the encoder and decoder in red, and the latent space in green. The KR portion of this network is trained on the reconstruction error of the input data. The TI takes the latent block from the KR as input; the TI has its own set of hidden layers in red, while the output layer is a softmax over the dataset classes. Since the TI is connected to the latent block of the KR, the error propagation from the label prediction also affects the encoder portion of the KR.

alternate between updating the trainable parameters θ of the KRTI:

$$\nabla_{\theta_{K_E}+\theta_T} = \frac{1}{m}\sum_{i=1}^{m} NLL(\hat{\vec{y}}^{(i)}, \vec{y}^{(i)}) \tag{1.1}$$

$$\nabla_{\theta_K} = \nabla_{\theta_{K_E}+\theta_{K_D}} = \frac{1}{m}\sum_{i=1}^{m} MSE(\hat{\vec{x}}^{(i)}, \vec{x}^{(i)}) \tag{1.2}$$

In equations 1.1 and 1.2, m is the mini-batch size, i is the index of the datapoint in the mini-batch, $\{(\vec{x}^{(i)}, \vec{y}^{(i)})\}_{i=1}^{m}$ are samples and labels from the training data, and $\{\hat{\vec{x}}^{(i)}, \hat{\vec{y}}^{(i)}\}_{i=1}^{m}$ are the outputs from the KR and TI respectively. θ_T is the parameter set for the TI, and θ_K is the parameter set for the KR with θ_{K_E} and θ_{K_D} being the parameters for the encoder and decoder of the AE used as the KR. NLL is the negative-log likelihood for the classification loss, and MSE is the mean squared error loss for the reconstruction loss.

Chapter 2 covers the background knowledge on the inspiration, relevant meta-learning objectives that our hybrid model experiments cover, and related hybrid models that also do not directly optimize for the task. Chapter 3 describes the setup of the experiments and model choices. In Chapter 4 we depict the results of the experiments, followed by a discussion of the outcomes in Chapter 5. Finally, Chapter 6 wraps up with the proposed implications of the work and possibly future directions.

Chapter 2
RELATED WORK

This work aims to settle itself within a minuscule corner of Artificial General Intelligence (AGI); the papers that initially primed this investigation were Francois Chollet's works [19] [1] that discussed the current narrow limitations of modern deep learning and proposed more human-based benchmarks of intelligence in AI models. The notion of moving towards biologically plausible AI (a "bottom-up" approach) to achieve AGI is an ideology shared by others in the community [20] [21], and is the central dogma of our work as well. On the other hand, the second main viewpoint studies the higher-level cognitive functions expressed in terms of mathematical formulae (the "top-down" approach) [22] [20]. Our work does not count this outlook as invalid; they are two sides of the same coin. However, we believe that the former path is more feasibly attainable than the latter one due to past studies in the related fields of psychology, psychometrics, and neuroscience [23] that provide insight on the foundations of intelligence as an effect of physical anatomy of the brain.

2.1 Deep Neural Networks

We presume that the reader has a basic understanding of deep neural networks (DNN) as a computational graph with nonlinear activations and that weight updates occur as an effect of minimizing some loss criteria. Hence, this section provides background on some specifics of neural networks that are pertinent to the main topic of our study.

2.1.1 Networks with Multiple Objectives

Simple DNNs typically have only one objective to minimize, and usually all of the trainable weights are updated relative to this error signal. For example, in the case of image classifica-

tion, a regular DNN outputs a softmax over a vector of class probabilities, and the result is compared with the input's corresponding label (which is also vectorized in the same manner). However there are other DNN models that employ more than one objective function that can affect different subsets of the entire network.

An example of this is the Generative Adversarial Network (GAN) [24], which trains two networks (a Generator $\mathcal{G}$ and a Discriminator $\mathcal{D}$) with different objectives ($\mathcal{G}$'s goal is to fool $\mathcal{D}$ by generating realistic samples, and $\mathcal{D}$ aims to recognize the fake and real samples), and the antagonistic nature of their alternating training steps lead to empirically good synthetic samples. While there is still only one global loss function in GANs that propagate the error signal backwards to both the $\mathcal{D}$ and $\mathcal{G}$, the weight updates between $\mathcal{D}$ and $\mathcal{G}$ are different:

$$\nabla_{\theta_{\mathcal{D}}} = \frac{1}{m}\sum_{i=1}^{m}\left[\log \mathcal{D}(\vec{x}^{(i)}) + \log\left(1 - \mathcal{D}(\mathcal{G}(\vec{z}^{(i)}))\right)\right] \tag{2.1}$$

$$\nabla_{\theta_{\mathcal{G}}} = \frac{1}{m}\sum_{i=1}^{m}\log\left(1 - \mathcal{D}(\mathcal{G}(\vec{z}^{(i)}))\right) \tag{2.2}$$

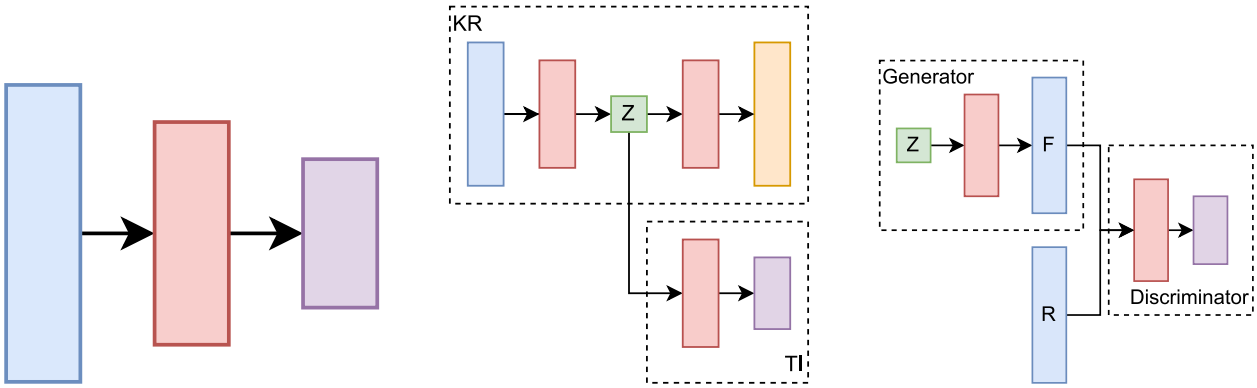
In equations 2.1 and 2.2, m, i, and $\vec{x}$ represent the same values from the weight updates of the KRTI and $\vec{z}$ is the latent input space that drives $\mathcal{G}$. Comparing equations 1.1 and 1.2 with equations 2.1 and 2.2, the main differences between the KRTI and the GAN are:

1. The KRTI has two loss objectives (classification and reconstruction) while GANs are trained on only one loss objective (binary classification).

2. The weights of the component networks in GANs are not necessarily shared (i.e. $\mathcal{D}$ and $\mathcal{G}$ can exist as separate networks), but the TI depends on the latent space of the KR as input.

Figure 2.1 visualizes the regular DNN with one objective, the KRTI, and the GAN as a guide for the reader to understand the architectural differences between them.

Although the idea of the KRTI was originally conceived independently, our literature review revealed that a previous study by Weston et al. [25] from 2012 that also proposed a



(a) A regular classifier DNN.

(b) A simplified diagram of the KRTI network.

(c) A simplified architecture of a GAN.

Figure 2.1: A set of figures of DNN variants. Each type of layer is color coded as follows: blue for input, red for hidden layers, purple for the supervised output layer, orange for the AE reconstruction output, and green for the latent space. (a) a simple classifier DNN with one hidden layer and a single objective. (b) A diagram of the KRTI, with an AE as the KR and a regular DNN as the TI. There are two objectives in this network, and the encoder weights are shared between both sub-networks. The latent space $\vec{z}$ is fed as input to the TI, which is driven by the encoder. (c) A diagram of a GAN with its discriminator $\mathcal{D}$ and generator $\mathcal{G}$. There is only one objective from $\mathcal{D}$ that will guide the weight updates of both $\mathcal{D}$ and $\mathcal{G}$.

similar design for deep learning architectures to learn in a semi-supervised manner. Specifically, they discussed using an embedding algorithm (such as kernel methods) to be jointly trained with a DNN, acting as a regularizer on the supervised signal.

One of the distinctions between our work and theirs is that their family of embedding algorithms are only encoders, whereas the KRs in this study are autoencoders which have both encoder and decoder sections. They argued that the decoders are an unnecessary hindrance, as the additional layers and objective add to the cost of training time, but we counter that their concern is not a limitation with modern technology and with the shallow models we focus on in our study. Second, in their algorithm they first update the weights of the network w.r.t. the classification objective first before the embedding function, while in a training step of the KRTI this order is reversed (i.e. reconstruction loss before label loss). Third, Weston et al. did not analyze their model's performance within the context of other meta-learning tasks as we do; they touch on the topic of few-shot learning (see Chapter 2.3.3) with their experiments on restricted subsets of MNIST, but they do not acknowledge this explicitly.

2.1.2 Hybrid Models

There are other hybrid models in the literature that demonstrate a better approximation of human cognitive features that are based on using traditional neural networks as supplements to a larger system. For example, OpenAI's CLIP model [26] jointly trains a vision model (ResNet [15]) with a language model (Transformer [27]) without directly optimizing for a particular benchmark, which resulted in better generalization performance on other datasets with similar classes. Interestingly, they also documented the discovery of multi-modal neurons they found in the higher-level layers of their networks that strongly responded to concepts of objects [28].

DreamCoder [29] is a neural program synthesis [30] [31] system that viewed learning as a form of program induction; DreamCoder is trained on input-output pairs and not only generated programs in a domain-specific language (DSL) that satisfied the problems but also bootstrapped its own knowledge base with reusable common concepts from previously seen

solutions using a variation on the wake-sleep algorithm [32]. To search for these common motifs in the programs that the system proposed, DreamCoder used a neural network for efficiency. While we do not touch program induction in this paper, DreamCoder is significant because it inspired us to search for techniques and models (such as Hopfield nets) that have only recently found a resurgence in popularity due to modern computing capabilities unlocking their feasibility.

The main ideas of the Transformer [27] also served as a loose conceptual inspiration for our architecture (the original Transformer architecture itself is an autoencoder with specialized self-attention blocks), in particular the purpose of the query/key/value learnable vectors. However we do not provide as much of a rigorous mathematical foundation of our architecture as they do. Another key difference is that transformers typically work on an input being broken up into a sequence, such as the Vision Transformer (ViT) [33] which breaks up an image into patches; however, in our paper we do not pre-process the images into sequences. A variant of the ViT called the Token-to-Token Vision Transformer (T2T-ViT) [34] also aimed to create an efficient ViT architecture that could outperform convolution-based networks with training on the dataset from scratch; this is similar to how we also compare baseline E2E models with our hybrid models on different training variations (i.e. pre-training versus full joint training).

2.1.3 Autoencoders

Autoencoders (AE) are unsupervised DNNs comprised of an encoder network, a decoder network, and a latent space (where most often the dimensionality of the latent space is smaller than either of the encoder or decoder layers). The encoder learns features from the raw input and compresses it into the latent space, while the decoder's goal is to reconstruct the original image by processing the latent space as input. AEs are critical to our work because they are the primary implementation of the KR we have chosen specifically for this paper. There are two main hyperparameters that AE variants tweak: the latent space and the loss function. In this paper, we focus only on varying the former, while keeping the latter

fixed with MSE.

One of the modified AEs we experiment with is the variational autoencoder (VAE) [35], which deviates from a vanilla AE in that the latent space becomes two fully connected layers $\vec{\mu}$ and $\vec{\sigma}$ that we pass as parameters into a multi-dimensional normal distribution to sample from; the result from this sample then becomes the latent vector $\vec{z}$ that the decoder uses to drive a reconstruction. Additionally, in a VAE the MSE loss is regularized with the addition of measure the Kullback-Leibler Divergence between the encoder distribution $q_\phi(\vec{z}|\vec{x})$ parameterized by $\phi = (\vec{\mu}, \vec{\sigma})$ and the prior distribution $p(z)$:

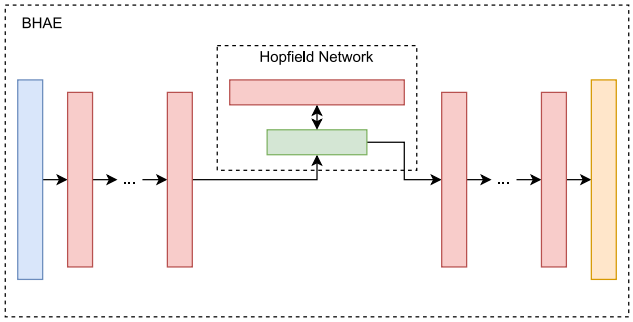
$$\nabla_{\theta_V} = \frac{1}{m}\sum_{i=1}^{m} MSE(\hat{\vec{x}}^{(i)}, \vec{x}^{(i)}) + KL(q_\phi(\vec{z}|\vec{x}^{(i)})||p(\vec{z})) \tag{2.3}$$

2.1.4 Hopfield Networks

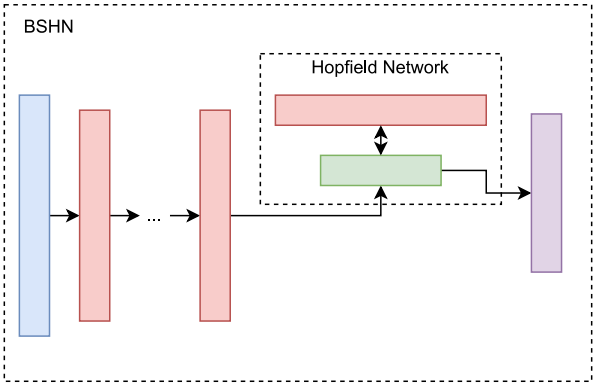
A Hopfield network is one of the earliest examples of a neural network, and a special type of recurrent neural network [36]. In its original formulation from 1984, it had binary activations and only consisted of fully connected visible units with symmetric weights; for modern Hopfield networks [37] the activations are continuous and the network supports hidden neurons as well. In either case, the main goal of a Hopfield network is to store and retrieve "memories", and is the primary reason why we chose to include them in our experiments. There are several other important properties to modern Hopfield networks, but we refrain from going into more detail because that knowledge is not required to understand the usage of Hopfield networks in the context of this study. We use Hopfield networks in both the latent space of the AE (BHAE) and the layer before the output in the classifier network (BSHN), as visualized in Figure 2.2.

2.1.5 Self-Normalizing Neural Networks & SELU

As DNNs grow deeper, the risk of vanishing or exploding gradients grows unless it is mitigated by some normalization scheme to control the values. Self-Normalizing Neural Networks



(a) The Hopfield Network inserted as the latent space within an autoencoder.



(b) The Hopfield Network inserted at the layer immediately before the output in a classifier network.

Figure 2.2: Visualizations of the role Hopfield Networks play in the KR and TI variants.

(SNN) [38] proposed that a network can achieve this with only the scaled exponential linear unit (SELU) activation function and proper weight initialization, and has been shown to converge faster and lower on the test error for MNIST with deeper (32 layers) SNNs. We use SNNs in this paper as the basis for our KR and TI candidate models.

2.2 MNIST & SVHN

The MNIST and SVHN datasets are the main sources we train our models on and generate results for. We chose to only focus on these two datasets in particular for a few reasons:

1. For the clarity of the study's main objective. While more datasets would certainly give us more information about the effectiveness of our hypothesis, this paper is only considered to be a "first step" and did not aim to be comprehensive in the wide variety of datasets and tasks in the field of AI (see Chapter 6.2).

2. Conceptual similarity, as both datasets contain the same types of input data (which are just the ten digits of the decimal system).

3. A notion of increasing difficulty (from MNIST to SVHN); MNIST is a simpler benchmark where smaller models can still achieve near perfect test accuracy, while models trained and evaluated on SVHN have a harder time with naively reaching higher test scores without considerably more effort.

MNIST [39] is a dataset of 70000 (60000 training, 10000 testing, equal class distributions) grayscale images and labels of handwritten digits that have been heavily pre-processed. Due to the image transformations, achieving high test accuracy scores on this dataset is not difficult; simple neural networks (e.g. two layers, 300 hidden units) have easily achieved at least 0.95 test accuracy [39], and fine-tuned variations of DNNs such as an ensemble of convolutional neural networks [40] can reach upwards of 0.99 test accuracy.
SVHN [41] is a dataset of around 99000 ($\sim$ 73000 training, $\sim$ 26000 testing, unequal class distributions) images and labels of digits collected from Google Street View. The SVHN

(a) A sample from MNIST of the number seven. (b) A sample from SVHN of the number five.

Figure 2.3: Samples from the test sets of MNIST and SVHN.

images are MNIST-like in the way that they are cropped and centered around a single digit, but the data is colored and may contain distracting elements (e.g. other digits) surrounding the main digit. Although the dataset is within the same conceptual domain as MNIST, it is much more difficult to do image classification on because of the complexity of the input samples. A simple convolutional neural network reaches around 0.71 test accuracy, and using a deep convolutional generative adversarial network (DCGAN) to learn on the features before training an SVM classifier on top only improves this by around 0.06 [42]. More complex (deeper and wider) models (such as some variations of the WideResNet [43] [44]) have reported test set error percentages within 1%.

2.3 Meta-Learning

In its most literal definition, meta-learning is about learning to learn. Rather than solely focusing on the literal task objective (e.g. reconstructing an image, predicting the correct class, or making a move that maximizes a score), meta-learning is about how the learning algorithm can improve itself over the course of training [45]. We describe the relevant subtopics within meta-learning that the models in our paper have attempted to address.

2.3.1 Representation Learning

Representation learning is the study of building abstractions of real-world data in order to effectively optimize the amount of relevant information processed by a model. It can be thought of as a byproduct objective of unsupervised learning in classical ML, or as an automated version of feature engineering; the key distinction of representation learning as its own subfield though is acknowledging the importance of a "good" representation being one of the salient factors in a performant model. One of the ideal benefits of learning appropriate representations is to gain a reusable knowledge base in a single model across multiple related tasks.

There is no exact definition of what constitutes an adequate representation, but [46] recommended a family of general-purpose priors that the authors proposed as crucial factors. For example, a model's architecture (e.g. support vector machines and convolutional neural networks) is a prior over its own representation learning capabilities (how information is interpreted and extracted by the model matters) and therefore has a significant impact on its performance, such as accuracy and sample efficiency. The core part of our paper is using the idea of representation learning as an explicit meta-learning objective and module in the hybrid model, rather than being implicit; furthermore, as a consequence of optimizing for representation learning, we explored how it affects the related meta-learning fields discussed below.

2.3.2 Transfer Learning

Transfer Learning is about using previous experience to aid in the learning of a new related task. The most common method of transferring the "knowledge" learned from a model on one task to another is to create a new model where the former model's parameters are used in tandem with the newer learned parameters of the latter. The main problem is that, to our knowledge, there is no general-purpose architecture nor learning algorithm that can truly generalize to a wide variety of tasks without additional structural modifications

(e.g. adding nodes per subsequent problems). [18] is a comprehensive review of the various modern techniques used to address transfer learning partially, but their experiment results only conclude their model/algorithm performance still depended on the particular problem. Our paper experimented with transferring a hybrid model's learned features between the MNIST and SVHN datasets.

2.3.3 Few-Shot Learning

Few-Shot Learning (FSL) is the meta-learning task of optimizing model performance when the training data amount is significantly small. To put it formally, FSL is the machine learning problem where a model $\mathcal{M}$ must learn a task $\mathcal{T}$ by achieving a high score on performance measure $\mathcal{P}$ with limited experience $\mathcal{E}$ [17]. At its core, the main approaches in FSL are rooted in the idea that we can use priors (such as data augmentation, model choice, and parameter optimization algorithm) to aid the search for the optimal hypothesis, despite sparse training data. There are three intuitive benefits to prioritizing FSL as a model's objective [17]:

1. Biological plausibility, approximating how humans learn.

2. Learning for rare cases, when real-world data is scarce.

3. Reducing data-gathering effort.

Several SOTA techniques and models in FSL are organized and described in [17]; among them, our paper's hybrid model most closely aligns with the category of models that learn with external memory, such as the Memory Augmented Neural Network [47]. The autoencoder architectures used as the KR in our experiments are essentially a key-value learned database; one key difference however is that one autoencoder variant we used utilized a modern Hopfield net that is formulaically equivalent to a generalized Transformer [27]. Furthermore, those models added a temporal component to be learned in their data, whereas in our hybrid model there is no such extra factor to be explicitly learned.

2.3.4 Curriculum Learning

Heavily inspired by how humans learn, curriculum learning posits that organizing training examples in a meaningful order has an impact on the convergence speed and value of the ML model [48] [49]. In our work, the transfer learning experiments can also be viewed partially as a form of curriculum learning examples; the MNIST dataset can be classified as "easier" samples of digit images while the SVHN dataset is considerably more difficult.

Chapter 3

METHODS

3.1 General KR-TI Parameters

The primary theme across the experiment setups is a series of ablation studies to understand the contributions of the hybrid architecture in a specific meta-learning environment when trained on the same labeled dataset and classification task; this involved a variety of neural networks being mixed and matched based on model depth and type. Figure 3.1 depicts the variations of the KR and TI we used in this study.

Conceptually, the goal of the KR is to learn information from only raw data. For the KR subnetwork, we assumed that unsupervised DNNs were an appropriate type of model to fulfill this role because their learned weights are not influenced by dataset-specific labels. Specifically, three different variations of autoencoders are candidates for the KR in this paper: vanilla autoencoders (BAE), variational autoencoders (BVAE), and autoencoders with a Hopfield network as their latent block (BHAE).The reconstruction loss function used to train these KR networks is the Mean Squared Error (MSE) loss. This is not to say that the AE is the only possible model that the KR can be, but the simplicity and interpretability of the AE makes it a good starting point of investigation.

The TI sub-network needed to be a model that supported supervised learning, specifically in image classification. Rather than the input layer taking in the raw input data, the TI instead expects the latent dimension of the KR as input to its network. The TI network's output layer outputs a softmax, which is fed into a negative log likelihood loss function. Depending on the training mode, this loss will only reach a portion of the entire KRTI network (see Chapter 3.3). We also implemented a version of the classification Hopfield network from [37], which just has a modern Hopfield network inserted before the output softmax layer.

We were inspired by the classification experiment setup of [37] to generate our KR and TI sub-networks, where they created a grid search for a feed-forward network based on varying the number of hidden layers and the number of neurons per hidden layer. We extended this by also doing a grid search on the latent dimension length specifically for the AEs. A table with all of the hyperparameters shown (and the actual values selected in bold) is depicted in Table 3.1; please refer to Appendix A for the full details on the initial grid search results, as well as some sub-network variant specific hyperparameters. Based on the test accuracy from E2E models, we decided to use three hidden layers and 256 neurons per hidden layer. When fixing these hyperparameters for the AE and evaluating on test loss, we found that a latent dimension length of 128 provided the lowest loss. For the AEs, the number of hidden layers and hidden units per layer were equally applied to both the encoder and decoder (i.e. three hidden layers in the encoder, and three hidden layers in the decoder). Figure 3.2 is a diagram of the general KRTI architecture used in these experiments, with slight variations of the TI and the latent space of the KR to be anticipated depending on the KRTI combination. Although this grid search is greedy, it significantly decreased the training time of the E2E and KRTI models on the main experiments, since we ended up with a pool of only eight models for testing, as opposed to performing the entire architecture grid search for every experiment.

3.2 Data Transformations

As mentioned in Chapter 2.2, the MNIST and SVHN datasets are the only datasets we run our experiments on in this study. Three main data transformations were needed to be applied to the MNIST dataset so that the tensor dimensionality would match, and so the same KR could be used without any architecture modification.

First, the image dimensions needed to match; MNIST images are 28x28, but the SVHN images are 32x32. We implemented a nearest-pixel resize transformation for both datasets, but chose to only resize the MNIST images to match the SVHN image dimensions for the

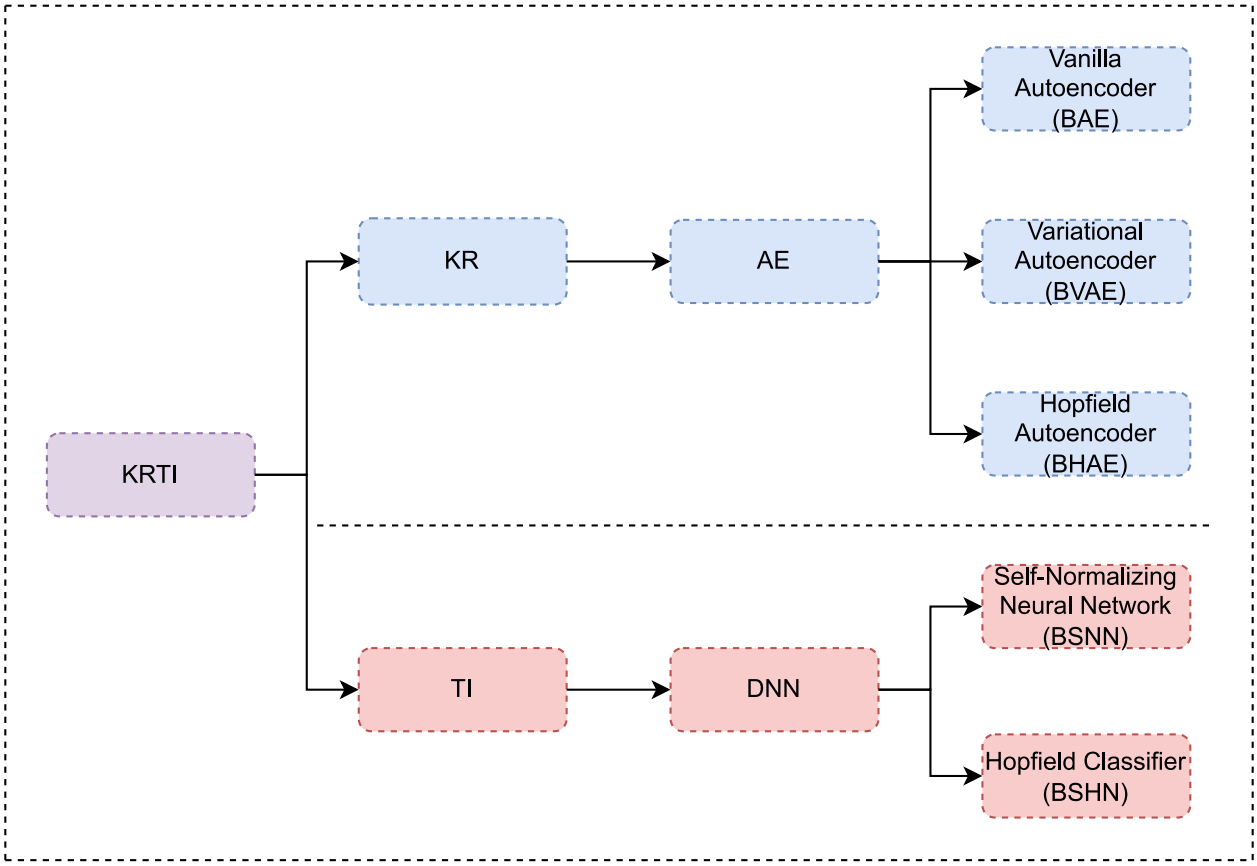


Figure 3.1: A tree of the KR and TI model variations we experiment with in this study. There are three different AEs for the KR: the vanilla autoencoder (BAE), the variational autoencoder (BVAE), and the Hopfield autoencoder (BHAE). For the TI, both networks are regular feedforward DNNs that use the SELU activation function, but the difference with the Hopfield classifier (BSHN) is that a Hopfield net is inserted in the layer right before the output. This means that there are six different KRTI model combinations in this particular set of components we've defined in this study.

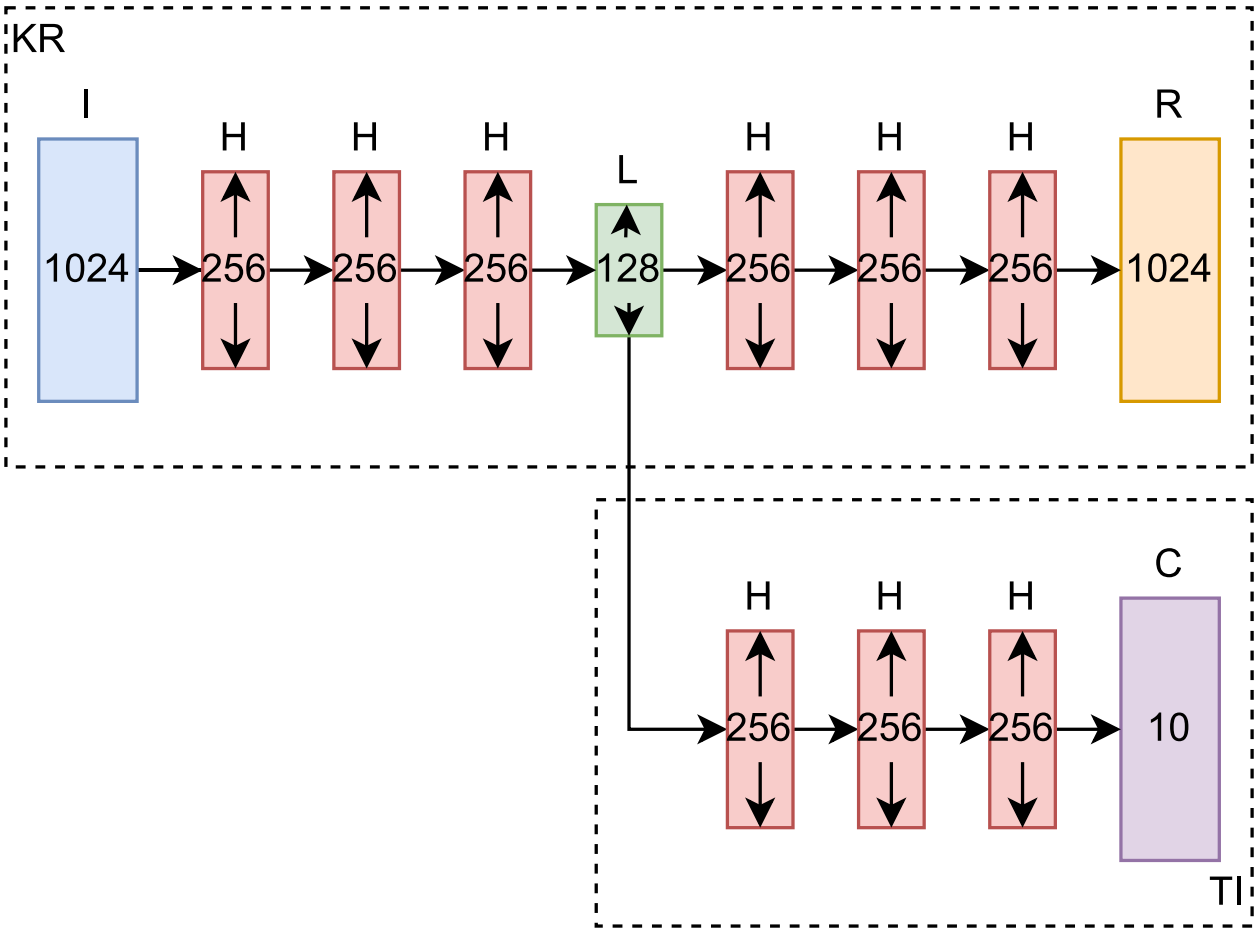


Figure 3.2: The KRTI archetype with chosen hyperparameters labelled per layer type. The input layer (blue, "I") has a length of 1024, which is the size of the 32x32 image dimensions flattened. The hidden layers (red, "H") each have a length of 256 neurons. In the KR, the number of hidden layers applies to both the encoder and decoder separately. The latent space (green, "L") is 128 neurons wide. The KR's output layer (orange, "R") matches the length of the input layer. The length of the TI's output layer (purple, "C") depends on the number of classes for the tasks (which is just 10).

Parameter	Value
Learning Rates	$\{1 \times 10^{-3}\}$
Hidden Layers	$\{1, \mathbf{3}, 5, 7\}$
Hidden Units	$\{64, \mathbf{256}, 512, 1024\}$
Input Resize	$\{32\}$
Image Channels	$\{3\}$
Latent Dim	$\{32, 64, \mathbf{128}, 256\}$
Optimizer	{Adagrad [50]}
Activation	{SELU [38]}

Table 3.1: Table of the hyperparameters used in our KRTI models.

experiments such that all of the images across both datasets being fed into the KR were the same 32x32 sizes.

Second, SVHN images are in color (three color channels), but the MNIST images are grayscale (one color channel). Therefore an image transformation that repeated every value in the tensor two more times was applied to MNIST, effectively transforming each of the MNIST samples into a colored image that matched the color channel count of the SVHN images.

Lastly, for the FSL experiment we needed to create smaller training subsets, so we implemented it with a restriction that enforces an equal class distribution in the new dataset (i.e. the same number of samples per class). The full MNIST dataset has an equal class distribution ($\sim$ 6000 samples per class), but since SVHN does not have an equal class distribution, this creates a discrepancy between the maximum subset size between MNIST and SVHN (please refer to Chapter 2.3.3 for more info).

3.3 Training Variations

The nature of the subnetworks in the KRTI mean that there are nonstandard variations on the ways we could have trained the entire network; the KR's AE has its own separate objective optimizer, while the entire KRTI's optimizer can affect both the TI and the AE's weights. There are three different scenarios we assessed (visualized in Figure 3.3):

1. Pre-training the KR, and then training the TI with the KR's weights frozen. When training the entire KRTI, the TI's weights are allowed to be updated, but the KR's weights remain unchanged.

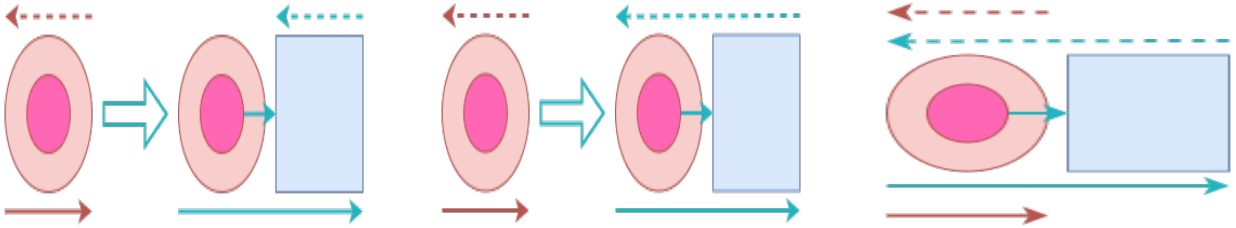
2. Pre-training the KR, and then training the TI with the KR's weights unfrozen. When training the entire KRTI, both the TI's and KR's weights are allowed to be updated.

3. The KR and TI are trained jointly with two objectives. Since there are two objectives, the KRTI's training step alternates between first calculating the KR's reconstruction loss, and then the KRTI's classification error afterwards.

3.3.1 Training the KR on the Test Set

We also explored the feasibility of continuously learning features with the KR on the test set while the TI's weights are frozen since the KR does not rely on labels. It is important to note that this test variation is still valid for two reasons:

1. When the KRTI is evaluated on its test set accuracy, this ultimately relies on the TI, while the KR can be interpreted as a learnable pre-processing step. Updating the KR on the test set surely affects the entire KRTI's class prediction on a sample, but since the TI is still frozen during this phase, the test labels do not affect any part of the KRTI.



(a) Pre-trained frozen KR.

(b) Pre-trained unfrozen KR.

(c) End-to-End training with two objectives.

Figure 3.3: The three main training scheme variations of the KRTI. The light-red circle is the KR module, while the inner pink circle is the latent block. The blue square is the TI module. The arrows below the modules represent the path of the signal during a forward pass, and the arrows above the modules represent the backward pass. (a) The KR is pre-trained, and then attached to the head of an untrained TI. In the backward pass of the full KRTI, the KR module's weights do not change. (b) similar to (a), but the weights of the KR are now allowed to be influenced by the loss of the classification task. (c) Both the KR and the TI are trained jointly, with the reconstruction loss objective changing only the KR's weights and the classification loss objective affecting TI and the encoder section of the KR.

2. The test accuracy is calculated by one pass over the test set, so the KR never sees a test sample more than once – this could not possibly affect the weights of the KR drastically enough to overwrite the learned parameters from the training samples that were repeatedly shown over several epochs.

Empirically the additional training seemingly provides only an insignificant benefit to test accuracy for the KRTI (see Chapter 4), but it is still an interesting testing variation to explore within the context of the KRTI being a pseudo-continuous/online learner.

3.4 Experiment Breakdown

Across all experiments, we use the validation loss as a signal for an early stopping mechanism and the test accuracy as the final performance metric for a model. The four main experiments are as follow:

1. In the first experiment we observed how the E2E models compared with the KRTI hybrids trained on a single dataset.

2. We moved on to testing the few-short learning capabilities of the hybrid models by retraining them on increasingly restrictive subsets of the original training data (e.g. 1000, 100, and 10 samples per class)[1]. The main idea here is to observe if the KRTI as an architecture with an explicit representation learning module provides any advantage in capturing generalizable features from fewer samples when compared to the E2E models.

3. The transfer learning experiment involved first training a KRTI on MNIST, and then reusing the same KR for a new KRTI on SVHN. We theorize that the KRTIs should benefit more from this additive training than the E2E models because the KR would be learning features that are unbiased by either task.

[1] We could not use the same subsets across both datasets, since the class distributions were not the same.

4. The final experiment, closely related to the transfer learning experiment, is when we reversed the training order and first train on SVHN before training on MNIST. Furthermore, we also examine reconstructions and KRTI performances after transfer learning when they are evaluated on the datasets they were originally trained on. The juxtaposition of this experiment with the aforementioned transfer learning experiment is meant to highlight any salience in the order of the training datasets when training the KRTIs.

3.5 Implementation Details

The experiments were run on a machine with Windows 11, and an Nvidia RTX 3090 (Ampere) GPU for compute acceleration. We chose PyTorch Lightning [51] as the Deep Learning research framework for our model and experiment implementations, as it provided a convenient API of typical ML callbacks for us to use; the code was written in Jupyter Notebook for flexible exploratory computing[2]. To prevent overfitting, we used an EarlyStopping callback that monitored the validation loss and had the criteria of a 0.01 minimum delta, 0.001 stopping threshold, and a patience of 10 epochs. Due to these parameters, each model required at least 20 epochs and an average $40 - 50$ epochs for training per experiment. Even with the drastically reduced model count to experiment with, a full suite of experiments needed about two to three days to complete and record the results. The PyTorch implementation of the continuous Modern Hopfield Network was provided as a PyTorch layer by Ramsauer et al. [37]. The code from this paper can be found on GitHub.

[2]The caveat was that spawning more worker threads for any of the multithread-able PyTorch functionalities was unavailable on Windows, which lead to considerably longer training times.

Chapter 4

RESULTS

4.1 Autoencoder Reconstructions

We validated the reconstructions of the standalone autoencoders and the first half of the KRTIs after training against a baseline test sample from both datasets to be certain that our implementations were learning appropriate features, as visualized in Figure 4.1. The MNIST reconstructions appeared to suffer from a higher frequency of nonuniform noise presumably because of us not imposing an equality restriction for the color components of a pixel. Although the quality of the SVHN reconstructions varied more than the MNIST reconstructions, the results were still acceptable enough to move onto the other experiments without the need for any other hyperparameter tweaking. Furthermore, recall that the KRTI has two objectives, which both update the encoder weights of our KRs; the image outputs of the trained KRs and the test accuracy performance of the entire KRTI (discussed in the sections below) suggests that the alternating updates can still produce shared weights that benefit both objectives simultaneously.

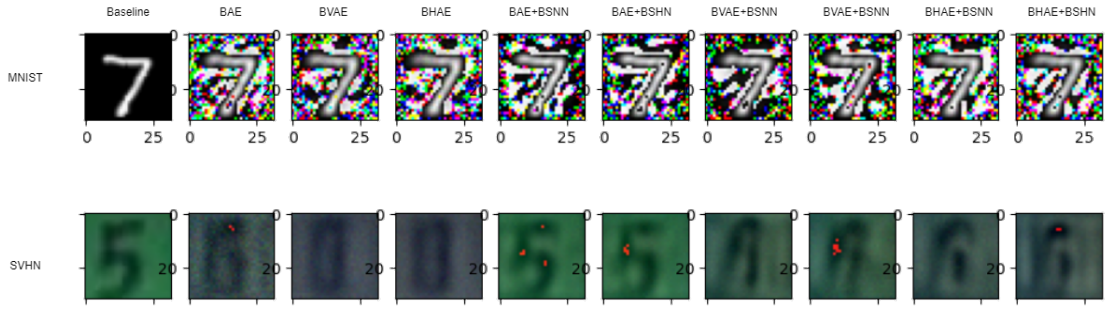


Figure 4.1: Autoencoder reconstructions of the input data from the MNIST and SVHN datasets. The leftmost images are the reference images taken from the test sets.

4.2 Unrestricted Training

After the hyperparameter grid search and reconstruction validation, our first core experiment involved training the E2Es and KRTIs on MNIST and SVHN separately (i.e. individual instances per dataset), plotted as a bar graph in Figure 4.3. Originally, with the training variations and allowing KR weight updates on the test set (Chapter 3.3), there were 32 different models per dataset (64 bars total); this not only made it difficult to see due to the close proximity of the data, but also the number of models was too large to complete the next experiments within a reasonable amount of time (see Figure 4.2). We decided to move forward with only the KRTI hybrids that were trained fully joint and that were allowed to update the KR on the test set for three reasons main reasons:

1. Pre-training networks is a practice that is already well-known in literature, whereas the joint training of multiple sub-networks in the KRTI is unique.

2. Similarly, the training of the KR on the test set is also a special feature that is a consequence of the KRTI – and according to the plot, this does not seem to skew the data with significant benefits.

3. Although the plot shows that pre-training the KR and training the KRTI does perform better on average than the full joint training method, it only does so by a relatively small margin.

Therefore, to ease the understanding of the plots and for the sake of time, we decided to conduct the following experiments with a smaller subset of KRTIs (bringing our total down to just eight models per dataset).

Across both datasets, we observed that the E2Es and KRTIs achieved similar test accuracy, with a score of about $0.95 - 0.97$ on MNIST and $0.72 - 0.77$ on SVHN. Despite the seemingly close results, a majority of the KRTIs still appeared to beat the E2Es (with the gap

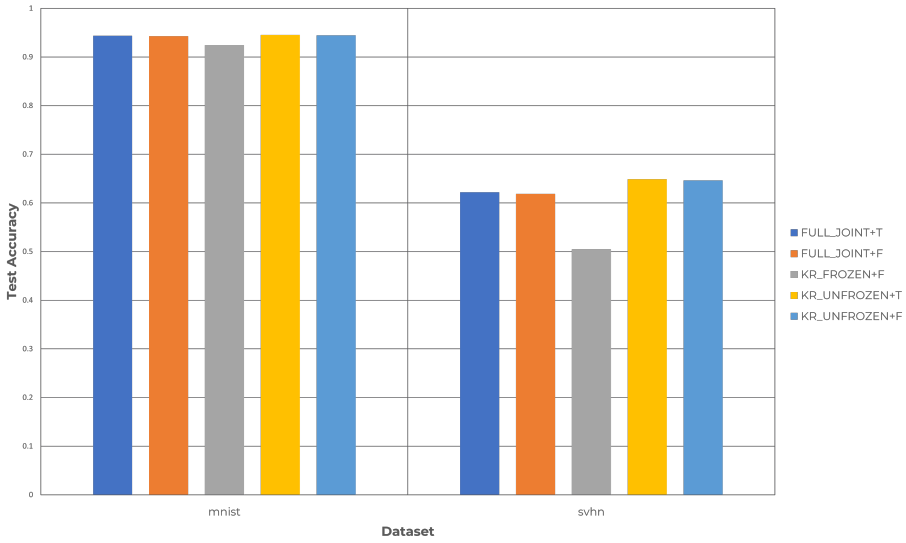


Figure 4.2: Evaluating the different training variations averaged across all KRTI combinations test accuracy scores. The first part of each legend item (KR_FROZEN, KR_UNFROZEN, FULL_JOINT) corresponding to a certain training variation discussed in Chapter 3.3. The second part (T, F) is a boolean flag to mark if the KR was allowed to train on the test set during evaluation.

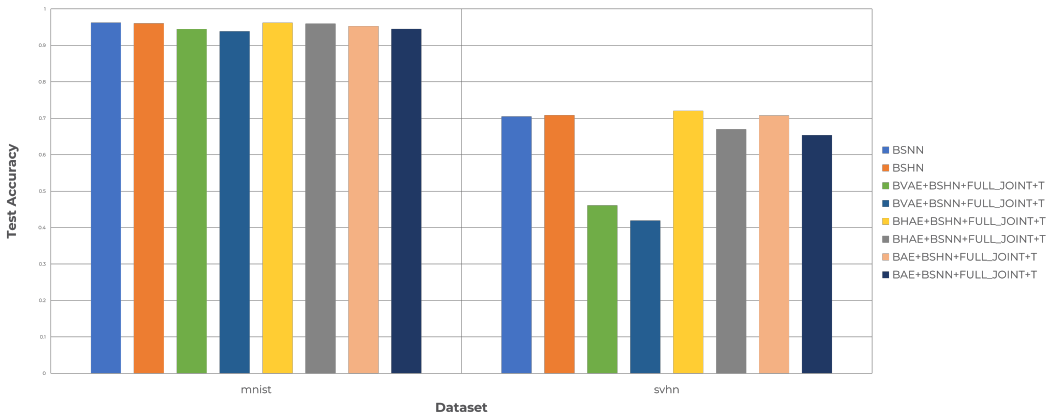


Figure 4.3: A comparison between the E2E and KRTI models' test accuracy, trained separately and fully on the MNIST and SVHN datasets.

more noticeable on SVHN than on MNIST); in particular, the BHAE+BSHN combination performed the best among all of the models and on both datasets.

4.3 Restricted Samples (FSL)

We compared the E2Es and KRTIs when trained on varying subsets of the training data with an equal amount of samples per class. Figure 4.4 shows the results of the models trained on MNIST, while Figure 4.5 visualizes the results from the models trained on SVHN. We could not generate the results for the same training samples ranges across both datasets since SVHN did not have equal class distributions; even though SVHN had more training samples ($\sim$ 73000) than MNIST ($\sim$ 60000), the largest possible subset of equal class samples created for SVHN was empirically found to only be around 10000 (1000 per class).

In Figure 4.4, the graph shows that both the E2Es and KRTIs achieved similar scores

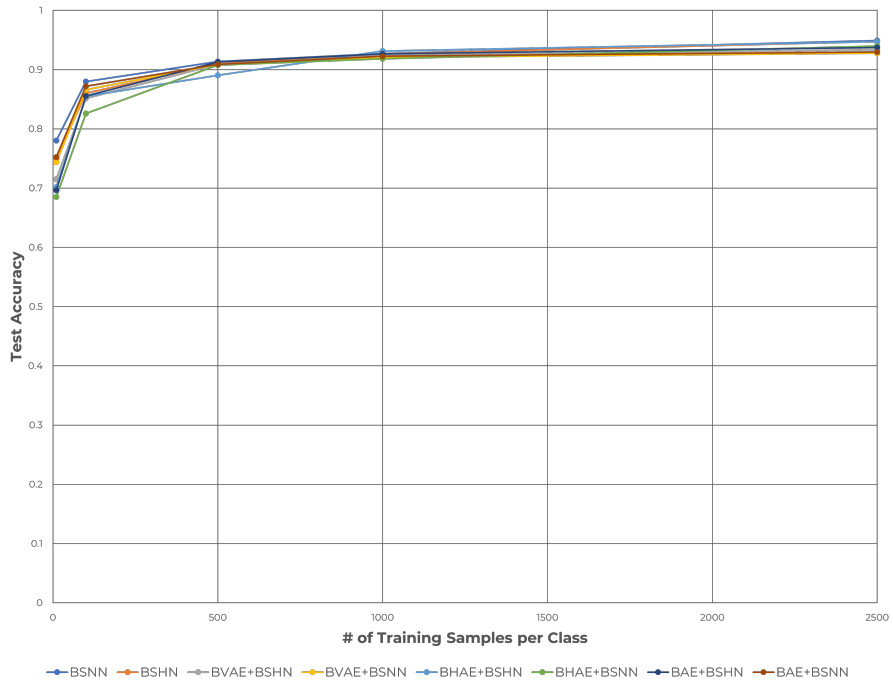


Figure 4.4: The effects of varying the number of equal class training samples on the E2E and KRTI models when trained on the MNIST dataset.

across all tested subset sample sizes, managed $0.70 - 0.80$ trained on only 10 samples per class, and reached above 0.90 with only 1000 samples per class. Figure 4.5 showed a slightly different story — the models appeared to perform the same on 10 samples per class, however the test accuracy variance increased as we expanded the training sample size. At 1000 training samples per class, the BSHN E2E reached the highest test accuracy among the rest of the models (just under 0.60) and the BAE+BSHN KRTI reached the highest test accuracy among the hybrid models, around 0.04 below the BSHN.

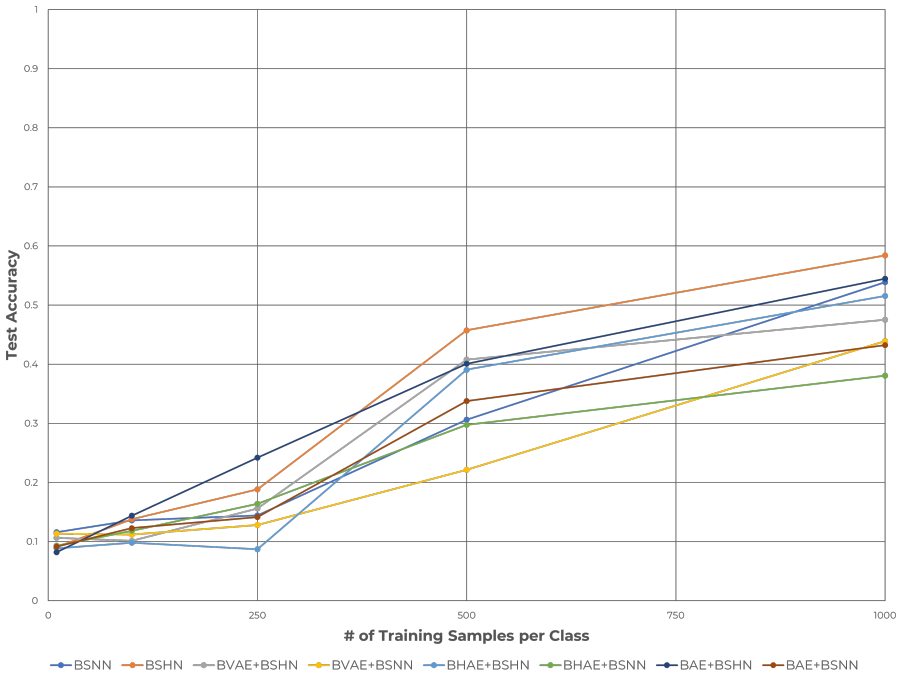


Figure 4.5: The effects of varying the number of equal class training samples on the E2E and KRTI models when trained on the SVHN dataset.

4.4 Training on MNIST, then SVHN (TL)

To analyze the transfer learning capabilities of the KRTIs, we first trained a version of the KRTI on MNIST, and then trained another KRTI (reusing the same KR, but initializing a

new TI) on SVHN. This was compared against the E2E models that were reused fully across both datasets (i.e. unlike the KRTIs, no sections of the original network were modified or replaced). When compared to the results from the first experiment, we observed that the E2E models actually performed worse ($\sim .05 - .07$) in this transfer learning experiment than when they were trained on the target dataset solely; conversely, the KRTIs all reportedly performed better than their counterparts from the first experiment results. Furthermore, almost all of the KRTIs achieved higher test accuracy than the E2Es with scores of at least ~ 0.70 (compared to the $\sim .63 - .65$ test accuracy reported from the E2E), but the exception to this was the BVAE+BSNN KRTI which reported a score closer to the E2E results.

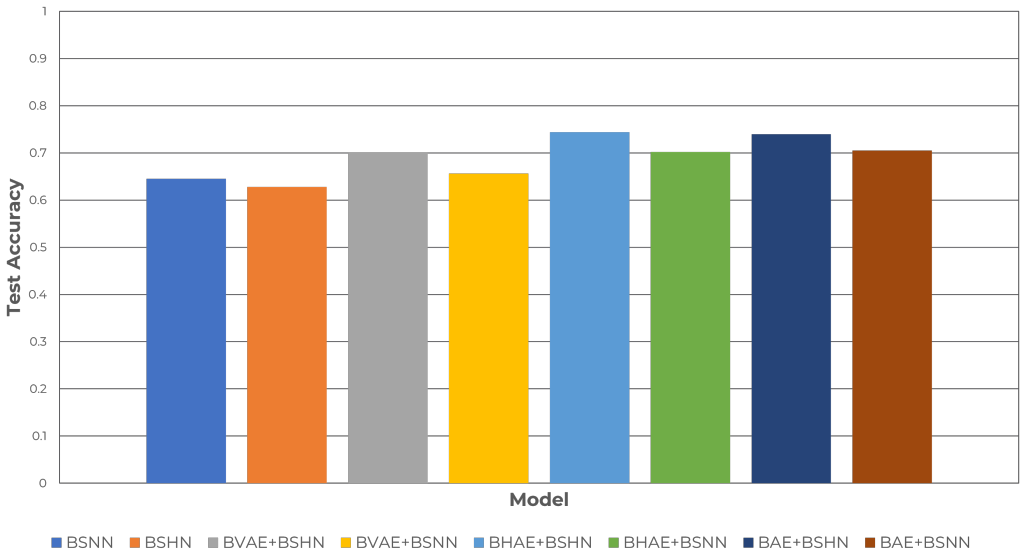


Figure 4.6: A plot of the models' accuracy on the SVHN test set after being trained on MNIST first.

4.5 Training on SVHN, then MNIST (CL)

This is similar to the previous experiment, except the training and evaluation datasets were switched (i.e. first trained on SVHN, then trained and tested on MNIST afterwards). The

results of Figure 4.7 were almost identical to Figure 4.3, which were both in the range of $0.95 - 0.97$; this suggests that there is little to no benefit of learning from a more complex task/dataset before moving onto a simpler one.

We investigated this further by observing the KRTI reconstructions of the test samples

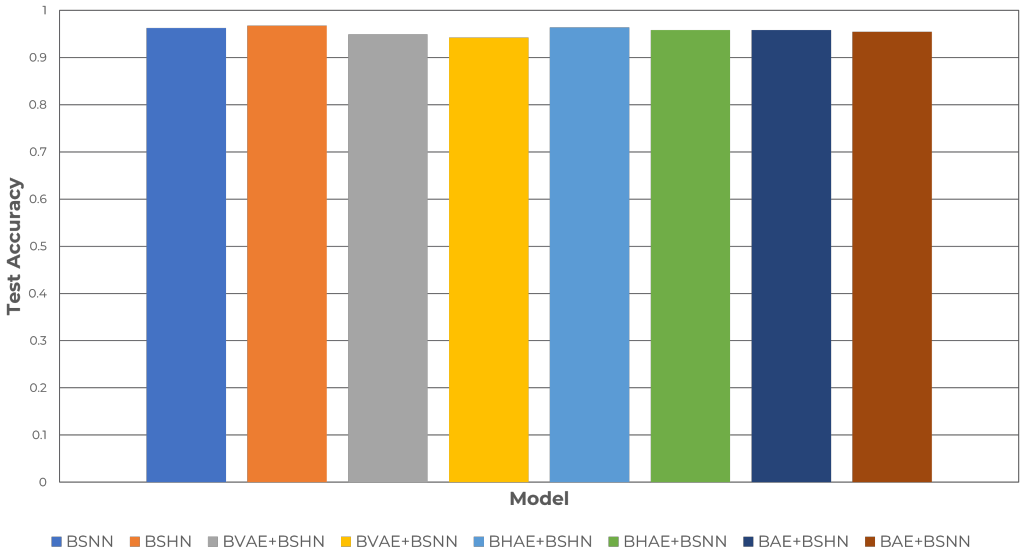


Figure 4.7: A plot of the models' accuracy on the MNIST test set after being trained on SVHN first.

after being trained on an additional dataset, depicted in Figure 4.8. The reference test samples used to generate these reconstructions are the same ones from Chapter 4.1; the image sequence in the top row was the set of reconstructions after performing the experiments from Chapter 4.4, and the bottom image sequence was from a model trained in this section's experiment. The similarity in both rows was that the KRTI was able to reconstruct images from the most recent dataset they were trained on, however there was a discrepancy in the quality of the rightmost column of reconstructions: there appeared to be difficulty in reconstructing the SVHN sample after training on MNIST, while the MNIST reconstruction after training on SVHN remained interpretable. This further supports the idea that some

aspect(s) of the datasets affect the KRTIs in such a way that prevent them from naively improving their performance with the addition of more training samples.

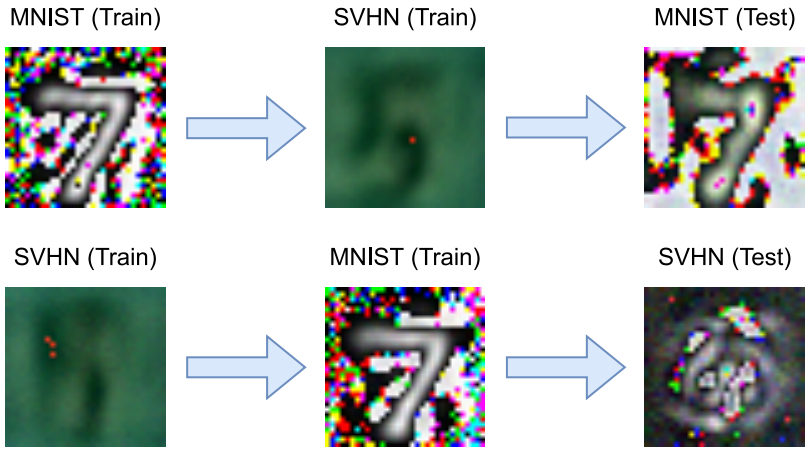


Figure 4.8: A sample of the reconstructions before and after transfer learning. In either row, the left and middle images are reconstructions from the KR after the KRTI is trained on the respective dataset. The images from the right column are reconstructions of the test sample from the dataset the model was *originally* trained on.

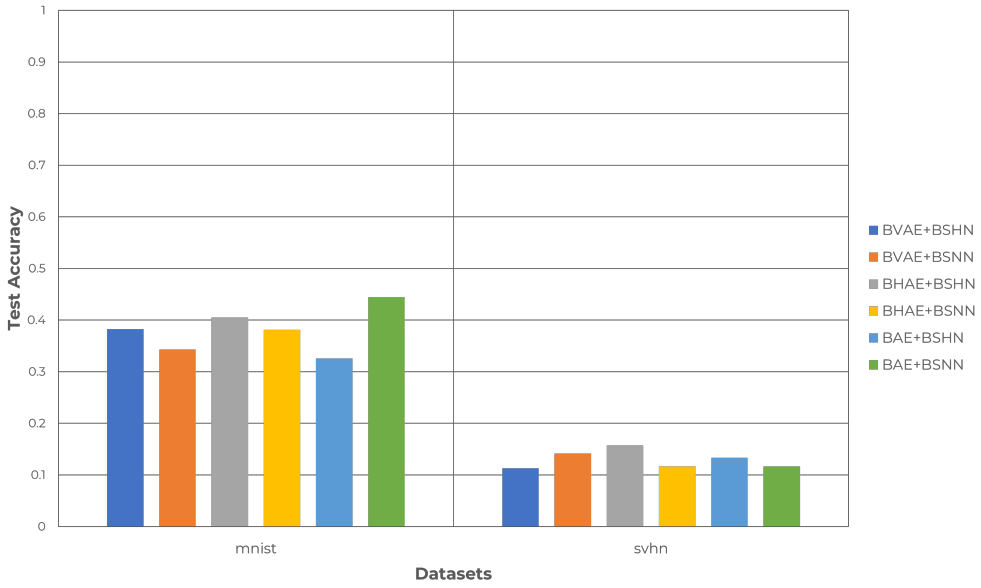


Figure 4.9: A plot of the test set accuracy of the models after transfer learning. Compared to Figure 4.3, the reevaluated test accuracy has fallen well below half of the original percentage scores for both datasets.

Chapter 5

DISCUSSION

One of the takeaways from the results reported in the previous chapter was that the model performance was not dominated by the architecture only, but by the complexity of the datasets as well. Firstly, MNIST is clearly too easy of a dataset, because every E2E and KRTI model achieved similarly high test accuracy scores across each experiment, and the effects of these variations, if any, are not as clearly distinguished as they are in SVHN. In the first two experiments, we observed that the KRTIs achieved similar test accuracy scores relative to the E2Es, which indicated that the features learned by the KR do not provide any benefit to the training or performance on a single dataset. In the FSL experiment, the KRTI combinations achieved varying results as we increased the amount of training samples per class in SVHN, suggesting that some of the models have may have certain properties that allow them to learn generalized features with fewer samples better than others. Ultimately none of the KRTIs achieved a significantly higher accuracy than the E2Es on any sample subset size.

In the third and fourth experiments that dealt with transfer learning and curriculum learning respectively, the KRTIs did appear to have a more noticeable benefit to the test accuracy scores over the E2Es when transferring to the SVHN dataset, however the same could not be said for MNIST. This suggests that there are features learned in the KR from MNIST that aid in the additional training on SVHN, and the variance in the scores among the models also indicate that the components used in the KR and TI have an impact on this effectiveness as well.

Lastly, the results of Figure 4.8 are interesting because the training order on the datasets appear to matter in the KRTI's ability to reconstruct data previously trained on. The KRTIs

appeared to "forget" how to reconstruct samples from SVHN after being updated on MNIST, but not the other way around. Additionally, Figure 4.9 shows that the entire KRTI performs poorly when reevaluated on the original dataset they were trained on after transfer learning. On both test datasets, they report accuracy scores below half of what they originally scored (compared to Figure 4.6 and Figure 4.7), again supporting the idea of the KRTI seemingly "forgetting" about older data. There were some plausible explanations to this behavior that we considered. One possible explanation is the difference in complexity between the datasets, since the MNIST data are cleaner, simpler, and more uniform than the data from SVHN. Perhaps because of this, training on MNIST initializes the weights in such a way that does not drastically affect the the KR when trained on SVHN afterwards, and that conversely the weights of the KR are essentially overridden when training from SVHN to MNIST. Another feasible reason might be that the class distribution discrepancy between MNIST and SVHN may affect how the KR memorizes certain class patterns. Further investigation in the training variations of the KRTI will be required to shed a better light on the cause and solution.

Chapter 6

CONCLUSION

We implemented our proposed KRTI neural network, comprised of a two-part objective optimizer that utilized an AE at the head of the network for representation learning and a supervised network at its tail for task-specific classification. We evaluated its performance across four main experiments on the MNIST and SVHN datasets, with a variety of combinations of its sub-module types and training schemes. The following points are the definitive takeaways from this work:

- When compared to the E2E classification networks, the KRTI variants performed comparatively the same in terms of test accuracy percentages. For one, this shows that although they are relatively deeper than the E2Es, they are not prone to overfitting. However, more critically this shows that this hybrid architecture does not simply give us performance out of the box in the typical training setting where we only optimize for the dataset/task.

- In the few-shot learning scenario, we observed that the KRTI networks have a greater variance in performance across restricted training sample sizes on the SVHN dataset, but none of them outperform the E2Es.

- In the transfer-learning experiment where the models are first trained on MNIST before training on SVHN, we saw that the KRTIs clearly had an advantage on the SVHN target dataset over the E2E models. In fact, when compared to the first experiment where the models are trained solely on one dataset, the E2E models perform worse when they are transferred from MNIST, but the KRTIs perform better than when they are trained just on MNIST. We attribute this behavior to the idea that the

E2E model's learned representations are inflexible to adapting to the target dataset, whereas the KRTI is more flexible with its KR module reused across both datasets and the creation of a new TI per dataset.

- In the curriculum learning experiment, the results show that across both datasets, the order in which we train the KRTI models on matter; the hybrid architecture achieved a higher test accuracy when they are first trained on MNIST before being introduced to SVHN. Not only this, but also that the KRTIs show signs of "forgetting" about the datasets they were originally trained on after transfer learning.

While our work was not able to successfully show any significant benefit to the KRTI architecture, we still find the results important as first-steps towards understanding how this hybrid model performs within the contexts of different meta-learning objectives. Each of of the meta-learning subfields we focused on have a plethora of models that tackle their problems separately, but our study is unique because we analyzed how our proposed architecture performed in multiple meta-learning objectives altogether. We have shown that naively constructing this hybrid model with a small variety of candidate sub-models still performs comparatively the same as their E2E counterparts, and in certain settings (e.g. the transfer learning experiment) can even perform marginally better; now we can move forward with finding the optimal model parameters that can increase the average KRTI performance even further (see Chapter 6.2 for more suggestions on this front).

Furthermore, as a consequence of how the KRTI was constructed, we showed that by just naively alternating the weight updates from the KR and TI's loss objectives, we still were able to create shared weights (e.g. the encoder portion of the AE was shared by both the KR and TI) that empirically balanced the KR's reconstruction quality and the TI's test accuracy. The KRTI also inadvertently contributes to the idea of model interpretability, as we have been able to indirectly validate the learned representations of the KR from the reconstruction outputs.

6.1 Limitations

Although the results from this paper demonstrate the feasibility of this hybrid architecture for more generalizable models, our scope was extremely limited with the model choices and dataset complexities. MNIST and SVHN are essentially toy datasets that already have fine-tuned solutions recorded in the literature – the current KR and TI choices do not provide any significant benefit over E2E models trained on the datasets solely, and we were unable to explore other problem domains that would have given us a broader view of the architecture's impact on model performance in meta-learning tasks. This supports the idea that there is a two-way dependency on the data complexity and the robustness of the sub-network model choices; the set of KRTI combinations in this paper were clearly lacking critical functional pieces that would have captured the intricacies of the SVHN dataset.

A possible shortcoming might have been the grid search of hyperparameters, since we generated them from sub-networks that were trained separately instead of altogether. For example, perhaps the AEs of the KR could have benefited from a different hidden layer count than the TI. This combined grid search however would blow up combinatorially, so a heuristic would most likely be needed to control the trade-off between the grid search time and being able to actually conduct the experiments.

6.2 Future Work

The first main direction of future work could come from diving deeper into the literature to find other candidate models for the KR and TI sub-networks, such as Spiking Neural Networks [52] or Vision Transformers [33]. Additionally, we should investigate how reweighing the two objective losses with some scalar coefficients might affect the model performances, as currently we do not implement hyperparameters to control how much both losses influence their respective network sections.

When we find a KRTI combination that does perform better in all four of these originally

proposed experiments, we can move onto exploring how the KRTIs perform on other datasets; not only on other popular image datasets such as CIFAR [53] and ImageNet [54], but also datasets from other domains such as NLP.

Furthermore, we were unable to give any concrete mathematical proofs to the behavior of the KRTI in the experiments, only empirical plausibilities; the models we ended up experimenting with were chosen for their simplicity or readiness in implementation (such as the Hopfield Network [37]). Delving into the math underneath these implementations would provide key insights to empirical behavior of the KRTI as a whole.

Now that there is a baseline for the simple addition of an explicit representation learning module in a neural network, we can also experiment with adding other sub-networks, or specifically tackle another meta-learning objective for a more modular neural network and a better understanding of how these meta-learning objectives contribute to higher levels of intelligence in AI.

BIBLIOGRAPHY

[1] F. Chollet, "On the Measure of Intelligence," *arXiv:1911.01547 [cs]*, Nov. 2019, arXiv: 1911.01547. [Online]. Available: http://arxiv.org/abs/1911.01547

[2] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[3] J. Sevilla, L. Heim, A. Ho, T. Besiroglu, M. Hobbhahn, and P. Villalobos, "Compute Trends Across Three Eras of Machine Learning," *arXiv:2202.05924 [cs]*, Mar. 2022, arXiv: 2202.05924. [Online]. Available: http://arxiv.org/abs/2202.05924

[4] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016. [Online]. Available: http://www.nature.com/articles/nature16961

[5] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm," *arXiv:1712.01815 [cs]*, Dec. 2017, arXiv: 1712.01815. [Online]. Available: http://arxiv.org/abs/1712.01815

[6] OpenAI, C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. d. O. Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang, "Dota 2 with Large Scale Deep Reinforcement Learning," *arXiv:1912.06680 [cs, stat]*, Dec. 2019, arXiv: 1912.06680. [Online]. Available: http://arxiv.org/abs/1912.06680

[7] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language Models are Few-Shot Learners," *arXiv:2005.14165 [cs]*, Jul. 2020, arXiv: 2005.14165. [Online]. Available: http://arxiv.org/abs/2005.14165

[8] L. Floridi and M. Chiriatti, "GPT-3: Its Nature, Scope, Limits, and Consequences," *Minds and Machines*, vol. 30, no. 4, pp. 681–694, Dec. 2020. [Online]. Available: http://link.springer.com/10.1007/s11023-020-09548-1

[9] M. Mitchell, "Can GPT-3 Make Analogies?" Aug. 2020. [Online]. Available: https://medium.com/@melaniemitchell.me/can-gpt-3-make-analogies-16436605c446

[10] B. Goertzel, C. Pennachin, D. M. Gabbay, J. Siekmann, A. Bundy, J. G. Carbonell, M. Pinkal, H. Uszkoreit, M. Veloso, W. Wahlster, and M. J. Wooldridge, Eds., *Artificial General Intelligence*, ser. Cognitive Technologies. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. [Online]. Available: http://link.springer.com/10.1007/978-3-540-68677-4

[11] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, Dec. 1943. [Online]. Available: http://link.springer.com/10.1007/BF02478259

[12] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, no. 4, pp. 193–202, Apr. 1980. [Online]. Available: http://link.springer.com/10.1007/BF00344251

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems*, vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html

[14] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Highway Networks," *arXiv:1505.00387 [cs]*, Nov. 2015, arXiv: 1505.00387. [Online]. Available: http://arxiv.org/abs/1505.00387

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *arXiv:1512.03385 [cs]*, Dec. 2015, arXiv: 1512.03385. [Online]. Available: http://arxiv.org/abs/1512.03385

[16] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely Connected Convolutional Networks," *arXiv:1608.06993 [cs]*, Jan. 2018, arXiv: 1608.06993. [Online]. Available: http://arxiv.org/abs/1608.06993

[17] Y. Wang, Q. Yao, J. Kwok, and L. M. Ni, "Generalizing from a Few Examples: A Survey on Few-Shot Learning," *arXiv:1904.05046 [cs]*, Mar. 2020, arXiv: 1904.05046. [Online]. Available: http://arxiv.org/abs/1904.05046

[18] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A Comprehensive Survey on Transfer Learning," *arXiv:1911.02685 [cs, stat]*, Jun. 2020, arXiv: 1911.02685. [Online]. Available: http://arxiv.org/abs/1911.02685

[19] F. Chollet, "The limitations of deep learning." [Online]. Available: https://blog.keras.io/the-limitations-of-deep-learning.html

[20] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman, "Building machines that learn and think like people," 2016.

[21] Y. Bengio, D.-H. Lee, J. Bornschein, T. Mesnard, and Z. Lin, "Towards Biologically Plausible Deep Learning," *arXiv:1502.04156 [cs]*, Aug. 2016, arXiv: 1502.04156. [Online]. Available: http://arxiv.org/abs/1502.04156

[22] Y. Bengio, "The Consciousness Prior," *arXiv:1709.08568 [cs, stat]*, Dec. 2019, arXiv: 1709.08568. [Online]. Available: http://arxiv.org/abs/1709.08568

[23] R. J. Haier, *The Neuroscience of Intelligence*, ser. Cambridge Fundamentals of Neuroscience in Psychology. Cambridge University Press, 2016.

[24] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014. [Online]. Available: https://arxiv.org/abs/1406.2661

[25] J. Weston, F. Ratle, and R. Collobert, "Deep Learning via Semi-Supervised Embedding," p. 8.

[26] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, "Learning Transferable Visual Models From Natural Language Supervision," *arXiv:2103.00020 [cs]*, Feb. 2021, arXiv: 2103.00020. [Online]. Available: http://arxiv.org/abs/2103.00020

[27] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," *arXiv:1706.03762 [cs]*, Dec. 2017, arXiv: 1706.03762. [Online]. Available: http://arxiv.org/abs/1706.03762

[28] G. Goh, N. C. †, C. V. †, S. Carter, M. Petrov, L. Schubert, A. Radford, and C. Olah, "Multimodal Neurons in Artificial Neural Networks," *Distill*, vol. 6, no. 3, p. e30, Mar. 2021. [Online]. Available: https://distill.pub/2021/multimodal-neurons

[29] K. Ellis, C. Wong, M. Nye, M. Sable-Meyer, L. Cary, L. Morales, L. Hewitt, A. Solar-Lezama, and J. B. Tenenbaum, "DreamCoder: Growing generalizable, interpretable knowledge with wake-sleep Bayesian program learning," *arXiv:2006.08381 [cs]*, Jun. 2020, arXiv: 2006.08381. [Online]. Available: http://arxiv.org/abs/2006.08381

[30] S. Gulwani, A. Polozov, and R. Singh, *Program Synthesis*. NOW, Aug. 2017, vol. 4, publication Title: Foundations and Trends in Programming Languages. [Online]. Available: https://www.microsoft.com/en-us/research/publication/program-synthesis/

[31] S. Chaudhuri, K. Ellis, O. Polozov, R. Singh, A. Solar-Lezama, and Y. Yue, "Neurosymbolic programming," *Foundations and Trends® in Programming Languages*, vol. 7, no. 3, pp. 158–243, 2021. [Online]. Available: http://dx.doi.org/10.1561/2500000049

[32] G. E. Hinton, P. Dayan, B. J. Frey, and R. M. Neal, "The "Wake-Sleep" Algorithm for Unsupervised Neural Networks," *Science*, vol. 268, no. 5214, pp. 1158–1161, May 1995. [Online]. Available: https://www.science.org/doi/10.1126/science.7761831

[33] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," 2020. [Online]. Available: https://arxiv.org/abs/2010.11929

[34] L. Yuan, Y. Chen, T. Wang, W. Yu, Y. Shi, Z. Jiang, F. E. Tay, J. Feng, and S. Yan, "Tokens-to-token vit: Training vision transformers from scratch on imagenet," 2021. [Online]. Available: https://arxiv.org/abs/2101.11986

[35] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," *arXiv:1312.6114 [cs, stat]*, May 2014, arXiv: 1312.6114. [Online]. Available: http://arxiv.org/abs/1312.6114

[36] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities." *Proceedings of the National Academy of Sciences*, vol. 79, no. 8, pp. 2554–2558, Apr. 1982. [Online]. Available: https://doi.org/10.1073/pnas.79.8.2554

[37] H. Ramsauer, B. Schäfl, J. Lehner, P. Seidl, M. Widrich, T. Adler, L. Gruber, M. Holzleitner, M. Pavlović, G. K. Sandve, V. Greiff, D. Kreil, M. Kopp, G. Klambauer, J. Brandstetter, and S. Hochreiter, "Hopfield Networks is All You Need," *arXiv:2008.02217 [cs, stat]*, Apr. 2021, arXiv: 2008.02217. [Online]. Available: http://arxiv.org/abs/2008.02217

[38] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-normalizing neural networks," 2017.

[39] "MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges." [Online]. Available: http://yann.lecun.com/exdb/mnist/

[40] S. An, M. Lee, S. Park, H. Yang, and J. So, "An ensemble of simple convolutional neural network models for mnist digit recognition," 2020.

[41] "The Street View House Numbers (SVHN) Dataset." [Online]. Available: http://ufldl.stanford.edu/housenumbers/

[42] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," 2015.

[43] S. Zagoruyko and N. Komodakis, "Wide residual networks," 2016.

[44] P. Foret, A. Kleiner, H. Mobahi, and B. Neyshabur, "Sharpness-aware minimization for efficiently improving generalization," 2020.

[45] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey, "Meta-learning in neural networks: A survey," 2020. [Online]. Available: https://arxiv.org/abs/2004.05439

[46] Y. Bengio, A. Courville, and P. Vincent, "Representation Learning: A Review and New Perspectives," *arXiv:1206.5538 [cs]*, Apr. 2014, arXiv: 1206.5538. [Online]. Available: http://arxiv.org/abs/1206.5538

[47] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap, "Meta-learning with memory-augmented neural networks," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ser. ICML'16. JMLR.org, 2016, p. 1842–1850.

[48] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09*. Montreal, Quebec, Canada: ACM Press, 2009, pp. 1–8. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1553374.1553380

[49] P. Soviany, R. T. Ionescu, P. Rota, and N. Sebe, "Curriculum Learning: A Survey," *arXiv:2101.10382 [cs]*, Apr. 2022, arXiv: 2101.10382. [Online]. Available: http://arxiv.org/abs/2101.10382

[50] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. 61, pp. 2121–2159, 2011. [Online]. Available: http://jmlr.org/papers/v12/duchi11a.html

[51] W. Falcon, J. Borovec, A. Wälchli, N. Eggert, J. Schock, J. Jordan, N. Skafte, Ir1dXD, V. Bereznyuk, E. Harris, T. Murrell, P. Yu, S. Præsius, T. Addair, J. Zhong, D. Lipin, S. Uchida, S. Bapat, H. Schröter, B. Dayma, A. Karnachev, A. Kulkarni, S. Komatsu, Martin.B, J.-B. SCHIRATTI, H. Mary, D. Byrne, C. Eyzaguirre, Cinjon, and A. Bakhtin, "PyTorchLightning/pytorch-lightning: 0.7.6 release," May 2020. [Online]. Available: https://zenodo.org/record/3828935

[52] H. Hazan, D. J. Saunders, H. Khan, D. Patel, D. T. Sanghavi, H. T. Siegelmann, and R. Kozma, "Bindsnet: A machine learning-oriented spiking neural networks library in python," *Frontiers in Neuroinformatics*, vol. 12, p. 89, 2018. [Online]. Available: https://www.frontiersin.org/article/10.3389/fninf.2018.00089

[53] A. Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.

[54] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[55] B. Goertzel, "The General Theory of General Intelligence: A Pragmatic Patternist Perspective," *arXiv:2103.15100 [cs]*, Apr. 2021, arXiv: 2103.15100. [Online]. Available: http://arxiv.org/abs/2103.15100

[56] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," 2017.

Appendix A

HYPERPARAMETER GRID SEARCH

Model Name	Dataset Name	# of Layers	# of Trainable Parameters	Estimated Model Size (MB)	Test Loss	Test Accuracy
BSNN_1_64	mnist	7	197322	1.201504	0.13283874094486237	0.9710000157356262
BSNN_1_256	mnist	7	789258	3.6184	0.10927214473485947	0.9772999882698059
BSNN_1_512	mnist	7	1578506	6.840928	0.15704329311847687	0.9749000072479248
BSNN_1_1024	mnist	7	3157002	13.285984	0.18791453540325165	0.9768000245094299
BSNN_3_64	mnist	9	201482	1.234528	0.108543261885643	0.9707000255584717
BSNN_3_256	mnist	9	855050	3.947104	0.11137384176254272	**0.9761999845504761**
BSNN_3_512	mnist	9	1841162	8.022624	0.2002382129430771	0.9728999733924866
BSNN_3_1024	mnist	9	4206602	17.746528	0.2574540972709656	0.9743000268936157
BSNN_5_64	mnist	13	209802	1.300576	0.09437792003154755	0.9775999784469604
BSNN_5_256	mnist	13	986634	4.604512	0.12267565727233887	0.9747999906539917
BSNN_5_512	mnist	13	2366474	10.386016	0.16003504395484924	0.9724000096321106
BSNN_5_1024	mnist	13	6305802	26.667616	0.16247911751270294	0.9772999882698059
BSNN_7_64	mnist	17	218122	1.366624	0.09978882968425751	0.9733999967575073
BSNN_7_256	mnist	17	1118218	5.26192	0.12426723539829254	0.9733999967575073
BSNN_7_512	mnist	17	2891786	12.749408	189.82421875	0.9670000076293945
BSNN_7_1024	mnist	17	8405002	35.588704	0.1729157567024231	0.9710000157356262
BSNN_1_64	svhn	7	197322	1.201504	0.9748027920722961	0.7315611839294434
BSNN_1_256	svhn	7	789258	3.6184	0.9763710498809814	0.7306007742881775
BSNN_1_512	svhn	7	1578506	6.840928	1.138135313987732	0.6806622743606567
BSNN_1_1024	svhn	7	3157002	13.285984	0.9724206328392029	0.748578667640686
BSNN_3_64	svhn	9	201482	1.234528	0.9315505623817444	0.731945276260376
BSNN_3_256	svhn	9	855050	3.947104	0.8766571879386902	0.7569913864135742
BSNN_3_512	svhn	9	1841162	8.022624	0.9407941102981567	0.7231868505477905
BSNN_3_1024	svhn	9	4206602	17.746528	1.0281543731689453	0.7115089297294617
BSNN_5_64	svhn	13	209802	1.300576	0.8721225261688232	0.7429701685905457
BSNN_5_256	svhn	13	986634	4.604512	1.030293583869934	0.6900737285614014
BSNN_5_512	svhn	13	2366474	10.386016	2.427436590194702	0.11070989817380905
BSNN_5_1024	svhn	13	6305802	26.667616	1.1425855159759521	0.6763598918914795
BSNN_7_64	svhn	17	218122	1.366624	0.9078353643417358	0.7195374965667725
BSNN_7_256	svhn	17	1118218	5.26192	2.2826740741729736	0.19587430357933044
BSNN_7_512	svhn	17	2891786	12.749408	2.2683629989624023	0.19587430357933044
BSNN_7_1024	svhn	17	8405002	35.588704	2.353313684463501	0.19587430357933044

Table A.1: The baseline hyper-parameter search, varying the number of hidden layers and nodes per layer. The network is a simple feed-forward neural network with layers using the SELU activation (otherwise known as Self-Normalizing Neural Networks [38]). The combinations are evaluated on both the MNIST and SVHN datasets separately, and we used the architecture that lead to the highest test accuracy across the datasets. We ultimately landed on three hidden layers with 256 neurons per layer, and apply this template architecture to all of the other model types we experiment with.

Model Name	Number of Heads	Beta	Dataset Name	Test Accuracy
BSHN_3_256_1_10.0	1	10.0	mnist	0.9812999963760376
BSHN_3_256_1_1.0	1	1.0	mnist	0.9796000123023987
BSHN_3_256_1_0.1	1	0.1	mnist	0.9779000282287598
BSHN_3_256_1_0.01	1	0.01	mnist	0.9807999730110168
BSHN_3_256_1_0.001	1	0.001	mnist	0.9779999852180481
BSHN_3_256_4_10.0	4	10.0	mnist	0.9768999814987183
BSHN_3_256_4_1.0	4	1.0	mnist	0.9735999703407288
BSHN_3_256_4_0.1	4	0.1	mnist	0.979200005531311
BSHN_3_256_4_0.01	4	0.01	mnist	0.9775999784469604
BSHN_3_256_4_0.001	4	0.001	mnist	0.9786999821662903
BSHN_3_256_8_10.0	8	10.0	mnist	0.9804999828338623
BSHN_3_256_8_1.0	8	1.0	mnist	0.977400004863739
BSHN_3_256_8_0.1	8	0.1	mnist	0.9764999747276306
BSHN_3_256_8_0.01	8	0.01	mnist	0.9765999913215637
BSHN_3_256_8_0.001	8	0.001	mnist	0.973800003528595
BSHN_3_256_16_10.0	16	10.0	mnist	0.9740999937057495
BSHN_3_256_16_1.0	16	1.0	mnist	0.9745000004768372
BSHN_3_256_16_0.1	16	0.1	mnist	0.9779000282287598
BSHN_3_256_16_0.01	16	0.01	mnist	0.9750999808311462
BSHN_3_256_16_0.001	16	0.001	mnist	0.9769999980926514
BSHN_3_256_32_10.0	3	10.0	mnist	0.9702000021934509
BSHN_3_256_32_1.0	32	1.0	mnist	0.9732000231742859
BSHN_3_256_32_0.1	32	0.1	mnist	0.9782000184059143
BSHN_3_256_32_0.01	32	0.01	mnist	0.9743000268936157
BSHN_3_256_32_0.001	32	0.001	mnist	0.9732999801635742
BSHN_3_256_1_10.0	1	10.0	svhn	0.7851490378379822
BSHN_3_256_1_1.0	1	1.0	svhn	0.7578749060630798
BSHN_3_256_1_0.1	1	0.1	svhn	0.7617547512054443
BSHN_3_256_1_0.01	1	0.01	svhn	0.7730485796928406
BSHN_3_256_1_0.001	1	0.001	svhn	0.7743546366691589
BSHN_3_256_4_10.0	4	10.0	svhn	0.773279070854187
BSHN_3_256_4_1.0	4	1.0	svhn	0.7735095024108887
BSHN_3_256_4_0.1	4	0.1	svhn	0.783074676990509
BSHN_3_256_4_0.01	4	0.01	svhn	0.7747387886047363
BSHN_3_256_4_0.001	4	0.001	svhn	0.7573371529579163
BSHN_3_256_8_10.0	8	10.0	svhn	0.7462354302406311
BSHN_3_256_8_1.0	8	1.0	svhn	0.7614474296569824
BSHN_3_256_8_0.1	8	0.1	svhn	0.7438921332359314
BSHN_3_256_8_0.01	8	0.01	svhn	0.7453134655952454
BSHN_3_256_8_0.001	8	0.001	svhn	0.7673632502555847
BSHN_3_256_16_10.0	16	10.0	svhn	0.09691917896270752
BSHN_3_256_16_1.0	16	1.0	svhn	0.15938076376914978
BSHN_3_256_16_0.1	16	0.1	svhn	0.19587430357933044
BSHN_3_256_16_0.01	16	0.01	svhn	0.19587430357933044
BSHN_3_256_16_0.001	16	0.001	svhn	0.09691917896270752
BSHN_3_256_32_10.0	32	10.0	svhn	0.11070989817380905
BSHN_3_256_32_1.0	32	1.0	svhn	0.15938076376914978
BSHN_3_256_32_0.1	32	0.1	svhn	0.11070989817380905
BSHN_3_256_32_0.01	32	0.01	svhn	0.19587430357933044
BSHN_3_256_32_0.001	32	0.001	svhn	0.19587430357933044

Table A.2: Hopfield hyperparameter grid search results.

Model Name	Latent Dimension Length	Dataset Name	Test Loss
BAE_3_256_8	8	mnist	0.016873251646757126
BAE_3_256_16	16	mnist	0.011147977784276009
BAE_3_256_32	32	mnist	0.009475179016590118
BAE_3_256_64	64	mnist	0.006305108778178692
BAE_3_256_128	128	mnist	0.005166282877326012
BAE_3_256_8	8	svhn	0.00904475525021553
BAE_3_256_16	16	svhn	0.006857775151729584
BAE_3_256_32	32	svhn	0.005802800878882408
BAE_3_256_64	64	svhn	0.0070210788398981094
BAE_3_256_128	128	svhn	0.005341238342225552

Table A.3: Searching for the optimal latent dimension length in the basic autoencoder (within the model name, represented as the number after the last underscore), based on the architecture's test loss across both datasets.