Strategies for Conceptual Change in Convolutional Neural Networks

Maarten Grachten
Carlos Eduardo Cancino Chacón

Abstract

A remarkable feature of human beings is their capacity for creative behaviour, referring to their ability to react to problems in ways that are novel, surprising, and useful. Transformational creativity is a form of creativity where the creative behaviour is induced by a transformation of the actor's *conceptual space*, that is, the representational system with which the actor interprets its environment. In this report, we focus on ways of adapting systems of learned representations as they switch from performing one task to performing another. We describe an experimental comparison of multiple strategies for adaptation of learned features, and evaluate how effectively each of these strategies realizes the adaptation, in terms of the amount of training, and in terms of their ability to cope with restricted availability of training data. We show, among other things, that across handwritten digits, natural images, and classical music, adaptive strategies are systematically more effective than a baseline method that starts learning from scratch.

Contents

1 Introduction

In order to survive, any living organism has to adapt to its environment. Far from being static, the environment for most organisms is subject to constant change. Arguably, complex and plastic information processing structures such as the mammal brain are an evolutionary answer to the requirement to adapt to unforeseen circumstances during the lifespan of the organism [Allen, 2012].

Some aspects of this adaptive behaviour, most notable in higher mammals such as humans, are called *creative*. Although the term is notoriously evasive of a precise definition, there is common agreement that creative behaviour involves elements such as novelty, surprise, and value [Weisberg, 1993; Csikszentmihalyi, 1996]. An example of creative behaviour can be found in the use of domain names in the world wide web. Although Top Level Domains (such as *.com*, *.org*, *it*) are intended as a means of structuring the world wide web, they are nowadays used for their linguistic meaning, rather than their original denotation of thematical or geographical structure. Examples of such uses are *youtu.be*, *friend.ly*, and *podca.st*. Arguably, this creative use is driven by the increasing use and communication of URLs, and therefore a growing need for shorter URLs, that are easy to memorize.

One of the most seminal attempts at formalizing the notion of creativity is that of Boden [2004]. She links creative behaviour of an agent to its *conceptual space*. This space represents the way the agent interprets its perception and actions. According to Boden, creativity amounts to mapping, exploration, and transformation of a conceptual space by an agent. An example of such a transformation given by Boden [1994, p. 522] is the transformation of the conceptual space for music by Arnold Schoenberg, who invented radically new music by dropping the constraint of the home key from the existing conceptual space of music.

Most literature on creativity focuses on the high level, cognitive phenomena it involves, taking for granted that a conceptual space, a way of dealing with the environment, is already present. In contrast to viewing creativity as a discrete transformation of one static conceptual space into another, a promising alternative perspective is to regard conceptual spaces as inherently dynamic. In this view, creativity is an inherent property of the dynamics of conceptual spaces that brings them into existence in the first place. This view also suggests a more gradual distinction between perception and cognition. Hofstadter [2008, p. 308] goes so far as to say that the ability to *reperceive* is a critical element of creativity.

This focus on the dynamic aspects of conceptual spaces naturally leads to *representation learning* [Bengio et al., 2013], an active field of research in machine learning. In this area, computational models, predominantly in the form of neural networks, are trained on data to form hierarchically structured representations of the data. Often, such representations capture semantically relevant characteristics of the data at several levels, and as such form a robust basis for subsequent tasks such as classification, or organization of data.

Typical methods for representation learning deal with the classical machine learning scenario in which a model is trained on a set of data $\mathbf{x}$, possibly with labels $\mathbf{y}$, drawn from a distribution $p(\mathbf{x}, \mathbf{y})$ (the generating distribution), and evaluated on further data that is also assumed to be from the same distribution $p(\mathbf{x}, \mathbf{y})$. This implies that the training methods are designed to converge to a

single set (or hierarchical structure) of representations, that is optimal given $p(\mathbf{x}, \mathbf{y})$. As stated at the beginning of this Section however, creativity is often driven by a need to adapt to a changing environment. In terms of the machine learning paradigm, this violates the assumption that $p(\mathbf{x}, \mathbf{y})$ is static over time.

Thus, for a representation learning model to stay effective in the face of a changing environment requires methods beyond standard representation learning algorithms. This shift of focus from learning representations in a static environment to adapting learned representations in a dynamic environment brings on multiple non-trivial problems to be addressed. For example, there is a need for a measure of how well a given representation suits the environmental requirements. Furthermore, in the light of an environmental change, sometimes gradual change of the learned representations may be beneficial, whereas on other occasions, a more radical change (for example by learning representations from scratch) may be more effective.

It is not in the scope of the current report to address all of these problems. In this document, we restrict ourselves to the scenario where a radical change in environment is given, and investigate which of several alternative strategies for dealing with that change are most effective in neural network models for representation learning. We are ultimately interested in finding generally useful strategies that allow a computational system to adapt efficiently to changing environmental requirements. We start by a precise description of the problem, and the model architecture we use for learning. Then we define a number of possible adaptation strategies, and report on an experiment in which these strategies are compared by evaluating them on several data sets. Before that, we discuss how the work presented here relates to creativity in a broader sense.

1.1 From adaptive representations to creative behaviour

In this document, we focus on strategies for conceptual change to adapt learned representations from one task to another task. The tasks we consider here are classification, and autoencoding. A change of task (for the model to adapt to) may either mean switching from classification to auto-encoding or vice versa, or in the case of classification, a change of target classes. Note however, that the change of task inducing the conceptual change (the changes in the learned representations) in these cases is an extrinsic factor. From the perspective of an agent interacting with an outside world, such changes may reflect changes in its environment (e.g. a change of habitat, with different species and objects). This scenario corresponds to the type of creativity described above, a form of adaptive, problem solving behaviour. A different, but related interpretation emphasizes the generative, aesthetic aspects of creative behaviour, that may lead to novel artefacts. In this case, the conceptual change manifest in the creative behaviour is thought to be driven by intrinsic factors rather than as a response to changes in the environment.

Are the methods for conceptual change described here, and evaluated in the context of classification and autoencoding, also of use in the realization of this second kind of creativity? Two current computational theories of creative behaviour suggest they may be, since both posit that at the basis of creative behaviour are mechanisms that encode incoming information, and adapt to a dynamic environment to provide optimal encodings, in line with evidence for learning principles in living organisms, such as *minimum entropy coding*

[Barlow, 1989; Atick, 1992; Olshausen and Field, 1996]. We will briefly describe both theories.

Wiggins and Forth [2015] propose a theory of creativity based on the Global Workspace Theory [Baars, 2002]. The criterion that determines what representations develop in this model is based on information-theoretic principles, demanding that the representations facilitate accurate prediction of the perceptual future, leading to more efficient information processing. In their model, multiple competing *generators* match perceptual input to learned representations from memory to predict future input. The representations most of the effective generators surface into the *Global Workspace*, reflecting *conscious awareness*. This awareness in turn, updates an associative memory that stores the adaptive representations to be used by the generators. The changes in semantics as a result of adapting the conceptual space (for instance in the form of *aberration* [Wiggins, 2006]) may be regarded as a restricted form of creative behaviour. In addition, Wiggins and Forth [2015] theorize that an instantiation of the theory in the form of a computational model, after being exposed to stimuli that have imprinted memories, is able to generate novel artefacts: In the absence of external stimuli, the system can fill its perceptual buffers from memory, thus continuing the cycle of generation from memory, selection, and updating memory.

Schmidhuber [2010] also argues that a fundamental aspect of cognition in an agent is to learn to compress (or equivalently, predict), the outside world and its affordances. Plausible drivers for this goal of optimal compression/prediction are extrinsic rewards such as sparing use of physiological resources (information that can be sparsely represented takes less energy to process and retain), and a selective advantage (by allowing better anticipation of future events). But Schmidhuber argues that an important explanatory factor for human behaviour is an *intrinsic* motivation to *improve* compression/prediction capabilities. This means that for an agent it is desirable in itself to discover ways of better compressing and predicting its experience, where the (intrinsic) reward is proportional to the degree of improvement. Schmidhuber claims that this intrinsic motivation manifests as *curiosity* in human beings, and that the subjective experience of improving compression/prediction is aesthetic (that is: fun, beauty, or elegance). This also leads to a notion of surprise that is different from the usual information-theoretic notion of surprise, in the sense that rather than the information content of an event itself, it is the degree to which the information content of an event triggers an improvement of the agent's compression/predicting capabilities. Thus, just as a sequence of repeated events (no information content), a sequence of random events (high information-content) is not surprising, and will soon become boring.

Both of the theories of creative behaviour described above involve an adaptive component that generates, predicts, or compresses perceptual input, in other words, it maps incoming data to an internal representation that adapts to accommodate novel patterns in the data. The precise nature of this accommodation process is not part of the above (rather high level) theories, but the methods evaluated here hint at possible implementations of such a process, especially in situations where the patterns in the data change drastically. Furthermore, the fact that the methods proposed here concern (deep) neural networks suits the *reinforcement learning* (RL) paradigm of Schmidhubers theory, where neural networks in combination with Q-learning [Watkins, 1989] have been shown to be very successful at a variety of game-playing tasks [Mnih et al., 2013; van

Hasselt et al., 2015].

The remainder of this document is structured as follows. In Section 2, we discuss a number of related problems known from machine learning and discuss their relevance for the problem at hand. In Section 3, we introduce the architecture of the neural network model we use, and a number of possible adaptation strategies to facilitate conceptual change. Section 4 presents the setup of the comparative evaluation of the strategies. The results of this evaluation are reported, and discussed in Section 5. Conclusions are presented in Section 6.

2 Related work

2.1 Transfer learning

Transfer learning is a sub-field of machine learning that deals with the question how to wield knowledge obtained for some task in some domain (the *source* task and domain, respectively), to better solve another task in a (possibly different) domain (the *target* task and domain, respectively). This rather general definition leaves room for a number of variations of transfer learning problems, depending on the conditions [Pan and Yang, 2010]. Most of these transfer learning problems can be characterized with the help of a probabilistic formalization: Let $\mathcal{X}$ be a feature space, and $P(X)$ ($X \in \mathcal{X}$) a marginal distribution over $\mathcal{X}$. Then the tuple $\mathcal{D} = \{\mathcal{X}, P(X)\}$ is called a *domain*. Furthermore, let $\mathcal{Y}$ be a label space and $P(Y|X)$ ($X \in \mathcal{X}, Y \in \mathcal{Y}$) a conditional distribution over $\mathcal{Y}$, given X. Then the tuple $\mathcal{T} = \{\mathcal{Y}, P(Y|X)\}$ is called a *task*[1]. Where necessary, the sub-scripts s and t are used on any of these elements to denote that the elements belong to the *source* and *target* domains, respectively.

Domain adaptation (see section 2.2) is a form of transfer learning that assumes the same task, but a different domain (i.e. $\mathcal{D}_s \neq \mathcal{D}_t$)), whereas in *inductive transfer* the essential element is that the task is different (i.e. $\mathcal{T}_s \neq \mathcal{T}_t$) [Ben-David et al., 2007; Pan and Yang, 2010]. Another sub-problem of transfer learning is *class imbalance*, where a particular class is substantially underrepresented in the source domain, with respect to the target domain, or vice versa: $P_s(X \mid Y) \neq P_t(X \mid Y)$ [Jiang, 2008]. Finally, *covariate shift* refers to the situation where the relation of a class label to the feature space changes from the source to the target domain: $P_s(Y \mid X) \neq P_t(Y \mid X)$ [Jiang, 2008]. This problem is also known as *concept drift* (see Section 2.4), with the difference that the latter term is more commonly used in contexts of online learning [Widmer and Kubat, 1996]: rather than dealing with distinct source and target domains/tasks, concept drift refers to a gradual change of $P(Y \mid X)$ over time within a domain.

Torrey and Shavlik [2009] describe three ways in which transfer from a source task/domain can help a model perform a target task, as illustrated in Figure 1. Most studies concerning transfer loss use only the asymptotic performance as a criterion for successful transfer. A common measure is the *transfer-loss* [Glorot et al., 2011]. It measures the loss of the transfer method minus the loss of an in-domain baseline method for the target domain, that is not informed by the source domain. The transfer-loss is thus proportional to the difference in asymptotic performance in Figure 1.

---

[1]The task specifies the marginal distribution $P(Y|X)$ to be approximated
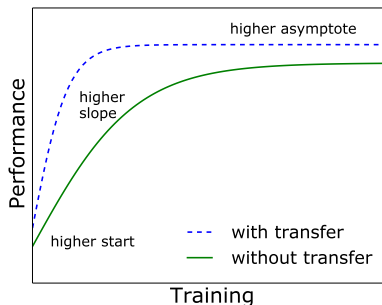
Figure 1: Three possible ways in which transfer learning may enhance performance on the target task. Adapted from [Torrey and Shavlik, 2009]

Furthermore, *unsupervised pre-training* of deep architectures [Hinton et al., 2006] can be regarded as forms of transfer learning, since the unsupervised learning task on which the model is trained in the first phase is different from the final, supervised task.

2.2 Domain adaptation

As previously stated, domain adaptation is a form of transfer learning where out-of-domain data (i.e. source domain) is used to improve the performance of a model solving the same task in the target domain. In this framework, the marginal distributions $P_s(Y|X)$ and $P_t(Y|X)$ are assumed to be neither identical nor independent [Daumé III and Marcu, 2006]. Previous work in domain adaptation use source data as "prior knowledge" to estimate the model parameters of the target domain using a Maximum a Posteriori (MAP) approach for tasks involving language modelling and parsing [Bacchiani and Roark, 2003]. Chelba and Acero [2006] propose a model using a Maximum Entropy model for the MAP estimation of the model parameters of the target domain for capitalization of text. This strategy was later revisited in [Daumé III, 2007], where the parameters of a model trained in the source domain are used to regularise the parameters of a model trained in the target domain.

2.3 Multi-task learning

Multi-task learning is related to transfer learning in the sense that a model is used to address different (but related) tasks, but where most transfer learning problems involve a phase where a model is trained on one task before turning to another task, multi-task learning involves the simultaneous training of the model for the different tasks. The motivation for this is that the training signals from related tasks serve to regularize the model, leading it to generalize better. For example, in a task where the steering direction of a vehicle was to be predicted based on camera images of the road in front of the vehicle, Caruana [1997] found that it is beneficial to train the model (a neural network) simultaneously on a number of additional tasks, such as predicting the left and right borders of the road, and whether the road has one or two tracks. Simultaneous training in this

case was realized by having one output unit for each task, so effectively, all but the upper layer of the model are affected by the training on multiple tasks.

2.4 Concept drift

Multi-task learning is related to transfer learning in the sense that a model is used to address different (but related) tasks, but where most transfer learning problems involve a phase where a model is trained on one task before turning to another task, multi-task learning involves the simultaneous training of the model for the different tasks. The motivation for this is that the training signals from related tasks serve to regularize the model, leading it to generalize better. For example, in a task where the steering direction of a vehicle was to be predicted based on camera images of the road in front of the vehicle, Caruana [1997] found that it is beneficial to train the model (a neural network) simultaneously on a number of additional tasks, such as predicting the left and right borders of the road, and whether the road has one or two tracks. Simultaneous training in this case was realized by having one output unit for each task, so effectively, all but the upper layer of the model are affected by the training on multiple tasks.

3 Method

The specific problem we focus on in the current experiment is a form of transfer learning. We are interested in the question how a model that has been trained to perform a particular task, can adapt to a novel task most effectively. As illustrated in Figure 1, there are several aspects to the notion of effective adaptation — most importantly the asymptotic accuracy on the novel task, and the amount of training it takes to perform a task accurately. We define the criteria we use for evaluation adaptation strategies in Section 3.2, but before that, we will give a more precise description of the problem we are addressing.

3.1 Problem description

Given a labelled data set, we divide the data set into two subsets, such that one subset contains the data pertaining to one half of the labels, and the other subset contains the data pertaining to the other half of the labels. We call one subset the *source* domain, and the other the *target* domain. This implies that the label space is different, i.e. $\mathcal{Y}_s \neq \mathcal{Y}_t$, and therefore $P_s(Y \mid X) \neq P_t(Y \mid X)$. Although splitting the data set by labels does not strictly imply that the marginal data distributions of the domains are different, the labels usually relate to some morphological aspect of the features, so it is likely that $P_s(X) \neq P_t(X)$.

The tasks we consider are classification, and autoencoding, and multi-task learning, in which a model is simultaneously trained to perform classification and autoencoding. The classification task is formalized as $\mathcal{T}^{CL} = \{\mathcal{Y}, P(Y \mid X)\}$. Note that in the autoencoding task, the labels Y are not used. Unfortunately, the probabilistic formalism is not well-suited to formalize the autoencoding task: autoencoding would be regarded as a case where $\mathcal{Y} = \mathcal{X}$, which implies the trivial task $\mathcal{T}^{AE} = \{\mathcal{X}, P(X \mid X)\}$. The actual value of the autoencoding task is in the fact that the *bottleneck* architecture of the models used to approximate $P(X \mid X)$ prevents them from learning a trivial mapping. A more precise description

of the autoencoding task is given in Section 3.3.1. Finally, multi-task learning involving simultaneous classification and autoencoding can be formalized as follows. Instead of the label space $\mathcal{Y}$, we define $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$. The multi-task formalization is then $\mathcal{T}^{MT} = \{\mathcal{Z}, P(Z|X)\}$, where $Z \in \mathcal{Z}$.

The aim of an adaptation strategy is then to transform a model that is optimal for task $\mathcal{T}_s$ on $\mathcal{D}_s$ into a model that is optimal on $\mathcal{T}_t$ on $\mathcal{D}_t$. For example, a model trained for a classification task $\mathcal{T}^{CL}$ in domain $\mathcal{D}_s$, might be adapted to perform a classification task $\mathcal{T}^{CL}$ in domain $\mathcal{D}_t$, but it may also be adapted more radically, to perform autoencoding $\mathcal{T}^{AE}$ in domain $\mathcal{D}_t$.

Since the architecture of a model is determined by the task it performs, it is not obvious what we mean by adapting a model from one task to another. In Section 3.3.3 we will describe this procedure in more detail. Another aspect to be clarified is how we measure the success of adaptation strategies. We address this issue in Section 3.2.

3.2 Evaluation criteria

As described above, our goal is to find adaptation strategies for representation learning models that allow a model to perform as well as possible on a new task, with as little training as necessary. In terms of the schematic plot in Figure 1, our primary interest is in the slope of the learning curve, but obviously, a model that adapts quickly but has substantially lower asymptotic performance than is possible with a baseline method is undesirable. Therefore, both the slope and the asymptotic performance with respect to some baseline methods should be taken into account when evaluating the methods.

We are ultimately interested in adaptation strategies that work well independent of the source and target tasks involved. This suggests that we define a single quality measure, that aggregates the the slopes and asymptotic performances of a strategy over the different combinations of source and target tasks. We chosen not to do so for the following reasons. Firstly, since they are different quantities, there is no obvious way to combine the slope and asymptotic performance of an adaptation strategy in a principled manner. A weighting of the two quantities necessarily reflects a personal judgement on their relative importance. Secondly, the experiments reported here should be regarded as explorative. Although we are interested in establishing the superiority of one adaptation strategy over another, independent of the tasks and domains involved, we have no a priori indication that such a ranking can be established across tasks and domains.

For these reasons, we refrain from defining a single evaluation criterion for the adaptation strategies. We believe that more insight is gained by a qualitative analysis of the evolution of task performance in target domain for each of the adaptation strategies, as a function of training, using plots similar to the diagram in Figure 1.

3.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a special kind of Feed Forward Neural Networks (FFNNs) that build some invariance properties into the structure of the neural network [Bishop, 2006]. CNNs have been successfully used in several machine learning applications, including natural language processing and image

classification [Krizhevsky et al., 2012]. CNNs have a number of advantages over fully connect FFNNs. Firstly, the convolutional nature of the architecture, using small convolutional filters enforces the extraction of local features [LeCun et al., 1998]. Secondly, they typically have shared weights, which greatly reduces number of parameters compared with similar sized FFNNs [Krizhevsky et al., 2012]. Lastly, they typically perform spatial sub-sampling, which adds robustness against noise and local distortions. A typical CNN has three building blocks: convolutional, subsampling, and dense (fully connected) layers. A CNN is illustrated in Figure 2.

The basic building block of CNNs are the convolutional layers. The input of l-th convolutional layer consists of $m_1^{(l-1)}$ feature maps from the previous layer, while its output consists of $m_1^{(l)}$ feature maps. The i-th feature map in this layer is given by

$$\mathbf{y}_i^{(l)} = f_l \left(\sum_{j=1}^{m_1^{(l-1)}} \mathbf{W}_{i,j}^{(l)} * \mathbf{y}_j^{(l-1)} + \mathbf{B}_i^{(l)} \right), \tag{1}$$

where $*$ represents the convolution of $\mathbf{y}_j^{(l-1)}$ with $\mathbf{W}_{i,j}^{(l)}$, a kernel of size $h_1^{(l)} \times h_2^{(l)}$ connecting the j-th feature map in layer $(l-1)$ with the i-th feature map in layer l, $\mathbf{B}_i^{(l)}$ is a bias (matrix) and f_l is an elementwise non-linear activation function.

Pooling can be understood as a form of non-linear downsampling. Max-Pooling layers partition an input feature map into a set of non-overlapping rectangles (*pool*), and for each such pool, outputs its maximum value.

Finally, dense layers are the standard fully connected layers in FFNNs. The output of the l-th dense layer can be computed as

$$\mathbf{y}^{(l)} = f_l \left(\mathbf{W}^{(l)} \mathbf{y}^{(l-1)} + \mathbf{b}^{(l)} \right), \tag{2}$$

where $\mathbf{W}^{(l)}$ is a filter connecting layer $(l-1)$ to layer l, $\mathbf{b}^{(l)}$ is a bias vector and f_l is an elementwise non-linear activation function. The set of all parameters of a CNN, i.e. kernels, filters and biases, will be denoted $\boldsymbol{\theta}$.

Common activation functions for CNNs include sigmoid, tanh, rectifier, and softmax, which is particularly useful as the output of a multi-class classifier [Bishop, 2006]. See Appendix A for explicit definitions of these activation functions.

In practice, convolutional and pooling layers are used to learn a feature hierarchy, while dense layers are used for classification purposes based on the computed features [LeCun et al., 1998]. In the following, we will refer to the stack of a convolutional and pooling layers as a *convolutional stage*, and the stack of dense layers as fully connected stage, or as a classification stage, if its primary objective is classification.

The classification task for CNNs can be formally described as follows. Given a set of input images $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, and a set of targets $\{\mathbf{t}_1, \ldots, \mathbf{t}_N\}$, where $\mathbf{t}_i$ is a one-hot encoding of the class of $\mathbf{x}_i$, the parameters of a CNN can be learned in a supervised way as

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\arg\min} \, L(\boldsymbol{\theta}), \tag{3}$$

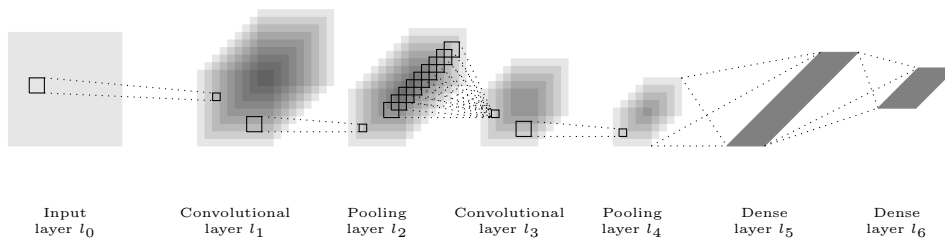| Input
layer l_0 | Convolutional
layer l_1 | Pooling
layer l_2 | Convolutional
layer l_3 | Pooling
layer l_4 | Dense
layer l_5 | Dense
layer l_6 |

Figure 2: Example of the architecture of a Convolutional Neural Network classifier, including two convolutional-pooling substages (layers l_1 and l_2, and l_3 and l_4), and a classification stage consisting of two fully connected layer (as a coding substage l_5, and the proper classifier l_6).

where $L(\boldsymbol{\theta})$ is the loss function. The standard loss function for a multi-class classification problem is the mean categorical cross entropy[2] [Bishop, 2006].

3.3.1 Convolutional autoencoders

Most methods in unsupervised learning are based on the "encoder-decoder" paradigm [Masci et al., 2011], where the input is first transformed into a (typically) lower-dimensional space (*encoding stage*) and then expanded to reproduce the initial data (*decoding stage*). Examples of this paradigm include Low-Complexity Coding and Decoding Machines, Predictability Minimization layers, Restricted Boltzmann Machines and autoencoders [Masci et al., 2011].

An autoencoder (AE) is a particular neural network architecture used for feature learning. Its aim is to learn an encodings, i.e., distributed (and usually compact) representation of a set of data. Formally, an AE takes an input $\mathbf{x} \in \mathbb{R}^{N_x}$ and maps it to a latent representation or *encoding* $\mathbf{y}_h \in \mathbb{R}^{N_h}$, using a deterministic function of type

$$\mathbf{y}_h = f(\mathbf{x}; \boldsymbol{\theta}) = f(\mathbf{W}\mathbf{x} + \mathbf{b}) \tag{4}$$

where $\boldsymbol{\theta} = \{\mathbf{W}, \mathbf{b}\}$ are the encoding parameters. A typical autoencoder uses functions similar to those of the fully connected layers (see Eq. (2)). This code is used to reconstruct the input by a reverse mapping (*decoding*), i.e.,

$$\mathbf{y}_{decoding} = f\left(\mathbf{y}_h; \boldsymbol{\theta}'\right) = f\left(\mathbf{W}'\mathbf{y}_h + \mathbf{b}'\right), \tag{5}$$

where $\boldsymbol{\theta}' = \{\mathbf{W}', \mathbf{b}'\}$ are the decoding parameters. A usual constraint of the decoding parameters if for the weights to take the form $\mathbf{W}' = \mathbf{W}^T$, i.e., use the same weights for encoding of the input and decoding of the latent representation.

Convolutional Autoencoders (CAEs) are (deep) autoencoders that use CNNs in the in the encoding stage [Masci et al., 2011]. As previously stated in Section 3.3, CNNs allow for discovery of localized features that appear over 2D input. In a CAE, the reconstruction of the input data is due to a combination of basic image patches based on the latent code.

---

[2]See Eq. (16) in Appendix A.2.

The decoding layers corresponding to fully connected layers are given in the same form as in Eq. (5). For convolutional layers, the corresponding i-th feature map of decoding layer l is given by

$$\mathbf{y}_i^{(l)} = f\left(\mathbf{y}^{h_{(l-1)}}; \boldsymbol{\theta}'\right) = f\left(\sum_{j=1}^{m_l^{h_{(l-1)}}} \tilde{\mathbf{W}}_{i,j}^{(l)} * \mathbf{y}_j^{h_{(l-1)}} + \mathbf{B}'^{(l)}\right), \qquad (6)$$

where $\mathbf{y}^{h_{(l-1)}}$ represents all $m_l^{h_{(l-1)}}$ feature maps of the encoding layer $h_{(l-1)}$, $\tilde{\mathbf{W}}$ represents the flip operation over both weight dimensions of the corresponding encoding parameter $\mathbf{W}$, and $\mathbf{B}'^{(l)}$ is a bias.

For downsampling layers in the encoding stage, the corresponding decoding layer corresponds to an upsampling layer. One possible upsampling strategy for max-pooling layers is to "remember" the position of the input that had the maximum value for every max-pooling during the forward propagation of the input data through the network. We will refer to a a layer that implements such a strategy as an *unpooling* layer.

Given a set of training data $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, the parameters of a (C)AE can be optimized in an unsupervised way by minimizing the reconstruction error of $\mathbf{X}$, in a similar fashion to Eq. (3). The typical loss function for (C)AEs is the mean squared error[3].

3.3.2 Multi-task learning

Solving a problem constrained to multiple objective functions is the object of study of *muticriterion optimization* [Boyd and Vandenberghe, 2004]. A vector optimization problem can be formalized as follows. Given a vector loss function $\mathbf{L}(\boldsymbol{\theta}) = [L_1, \ldots, L_q]^T$, whose components can be interpreted as q different scalar objectives, to be minimized. In order to apply standard scalar optimization methods, such as gradient descent, we can "scalarise" a multi-criterion problem by forming a weighted sum objective, i.e.

$$L = \boldsymbol{\alpha}^T \mathbf{L}(\boldsymbol{\theta}) = \sum_{i=1}^{q} \alpha_i L_i(\boldsymbol{\theta}), \qquad (7)$$

where α_i represents a weighting coefficient for the i-th component of $\mathbf{L}(\boldsymbol{\theta})$. This weights are usually constrained to $\alpha \le 0$ and $\sum_i^q \alpha_i = 1$. Using this framework, it is possible to express the multi-task objective as minimizing a joint loss function for both classification and autoencoding as follows

$$L(\boldsymbol{\theta}) = (1 - \alpha_{MT})L_{CL}(\boldsymbol{\theta}) + \alpha_{MT}L_{AE}(\boldsymbol{\theta}), \qquad (8)$$

where α_{MT} is the multi-task weight coefficient and $L_{CL} = L_{CCE}$ and $L_{AE} = L_{MSE}$ represent the loss functions for the classification and autoencoding tasks, respectively.

3.3.3 Transfer of a CNN model from one task to another

Given a CNN with N layers used for classification, an CAE can be built as follows. For layers $l_{N-1}, \ldots l_1$, (i.e., all layers in the CNN except the classification stage l_N), build the mirroring decoding layer, corresponding to Eq. (5)

---

[3]See Eq. (15) in Appendix A.2.

if the encoding layer is a fully connected layer, Eq. (6) if the encoding layer is a convolutional layer or an unpooling (upsampling) layer if the encoding layer is a max-pooling (subsampling) layer. Conversely, given a CAE with $2(N-1)$ layers, a CNN classifier can be built by removing the decoding layers from the CAE (layers l_N to $l_{2(N-1)}$) and appending a fully connected layer with with as many softmax units as the desired number of classes.

In order to reduce the degrees of freedom, in the transfer of a model from one task to another, we do not use biases in the decoding layers of a CAE built from an autoencoder. This guarantees that only the weights (present in both models) are responsible for learning the task.

3.4 Strategies for conceptual change in CNNs

In this Subsection, we propose a number of strategies for adapting learned representations in response to new tasks. For clarity, we label each choice with a keyword, and refer to the final strategies as a combination of keywords, listed at the end of this section.

3.4.1 Two baselines

There are two obvious baseline approaches to the adaptation of a model to a new task. One is to make no changes to the model upon the change of task. In this case, the only change is that the training data of the old task is replaced by the training data of the new task.

A practical issue is that a new task implies a new interpretation of the outputs, and possibly a different number of outputs. Reusing the parameters of the output layer for the new task raises the question of how the output variables for the old task should be mapped to those of the new task, a mapping that is necessarily arbitrary. For this reason, when we reuse the parameters of a prior model, we replace its output layer (the rightmost layer in Figure 2, page 11) with a new layer, initialized with random parameter values. Ignoring this detail, we refer to this baseline strategy as REUSE ALL.

Another, contrary approach is to altogether ignore the representations learned on the prior task, and start with a randomly initialized model to learn the new task. This can be regarded as a case of infinite plasticity, where the new task forms the representations without any trace of the representations that were learned in the prior task. This strategy will be referred to as RESET.

3.4.2 Selective reuse of the prior model: keep convolutional filters

Considering the dynamics of learning, REUSE ALL may not be ideal, since it starts learning task 2 with a model that is specialized on task 1. It may take more effort to "unlearn" aspects of task1 that are irrelevant for task 2, than to learn task 2 from scratch.

However, if the data in task 1 and task 2 are similar in nature, it is likely that the data have at least some common structure. For example, in the case of natural images depicting different classes of objects, it is likely that certain low level representations, such as local edges in natural images, are useful for different tasks, such as the recognition of different object classes. Different classes of objects may involve different constellations of similar forms. For example,

a dark round shape may represent wheels on a car, portholes in a ship, or the eyes of an animal.

In the current experiment, this observation inspires an adaptation strategy where the convolutional filters — representing the low level representations — are preserved across tasks, whereas the rest of the network is re-initialized to a random state. This option is referred to as REUSE CF.

3.4.3 Prior regularisation

A strategy for domain adaptation proposed by Chelba and Acero [2006] is to take a maximum a-posteriori approach to the estimation of the model parameters for the new task, where the model learned on the first task provides a prior estimate of the parameters. This prior serves as a regulariser for the parameters.

Standard parameter regularisation schemes are based on the assumption that the parameters $\boldsymbol{\theta}$ follow a zero-mean gaussian distribution, that is $p(\boldsymbol{\theta}) \sim \mathcal{N}(\mathbf{0}, \frac{1}{\lambda}\mathbf{I})$, leading to the addition of a regularisation term $\frac{\lambda}{2}||\boldsymbol{\theta}||_2^2$ to the standard expression for the loss, in Equations (15) and (16), page (28).

The prior proposed by Chelba and Acero [2006] is:

$$p(\boldsymbol{\theta}) \sim \mathcal{N}(\boldsymbol{\theta}_{old}, \frac{1}{\lambda}\mathbf{I}), \tag{9}$$

a gaussian distribution around the parameters $\boldsymbol{\theta}_{old}$ that were learned on the first task. This assumption leads to an alternative loss function L_{PR}, including the prior regularisation (PR) term [Daumé III, 2007]:

$$L_{PR}(\boldsymbol{\theta}) = L(\boldsymbol{\theta}) + \frac{\lambda}{2}||\boldsymbol{\theta}_{old} - \boldsymbol{\theta}||_2^2. \tag{10}$$

In the current experiment, the prior regularisation is applied to the convolutional filters, and used in combination with the RESET option; it is referred to as RESET PRF.

3.4.4 A note on random initialization

The idea of investigating the variance of the layer outputs to improve weight initialization for deep learning was introduced in [Glorot and Bengio, 2010]. This result has motivated the search for careful initialization, rather than unsupervised pretraining of networks with methods such as RBMs, thus representing a considerable speedup in the training of neural networks . Glorot and Bengio [2010] suggest that keeping the layer-to-layer transformations such that the singular values of the Jacobian matrices associated with each layer[4] are approximately 1 is equivalent to keeping the ratio of the average activation variance going from layer $\mathbf{y}^{(l)}$ to layer $\mathbf{y}^{(l+1)}$. This result implies that the random initialization of the model parameters for each layer depends on the nonlinear activation function and the size of the layer. The main intuition behind this initialization strategy is that if the network parameters are initialized to small, the output shrinks while passing through each layer, and eventually being to small. On the other hand, if the network parameters are too large, the output of each layer keeps growing, until the output of the network is saturated.

---

[4]the Jacobian matrix associated with the l-th layer is given by $\mathbf{J}^{(l)} = \frac{\partial \mathbf{y}^{(l+1)}}{\partial \mathbf{y}^{(l)}}$.

RESET	Initialize parameters with random values
RESET PRF	Prior regularisation on convolutional filters
REUSE ALL	Initialize all parameters (except output layer) from prior model
REUSE CF	Initialize only convolutional filter parameters from prior model
(CL)	Prior model trained as classifier
(AE)	Prior model trained as autoencoder
(MT)	Multi-task: Prior model trained simultaneously as classifier and autoencoder

Table 1: Labels denoting adaptation strategies, and their meaning, as used in the results. The labels in parentheses represent prior models, to be used in conjunction with one of the adaptation strategies

We use the same random initialized parameters across adaptation strategies to rule out initialization as a noise factor for each run of the experiment.

4 Experiment

The adaptation strategies are compared on three different data sets, described in Section 4.1. For each data set the following procedure is followed.

1. The data set is split into a source and a target domain, as described in Section 3.1. For the specific partitioning per data set, see Section 4.1

2. A model is trained in the source domain for each of three tasks: classification, autoencoding, and the multi-task of simultaneous classification and autoencoding

3. A model is trained in the target domain for each of two tasks: classification, and autoencoding. For each adaptation strategy, two training runs are realized

4. Each model is evaluated on the target test set during training, in order to monitor adaptation. Per adaptation strategy, the results of the two training runs are averaged, in order to reduce the impact of random effects

A schematic overview of the above process is given in Figure 3. Since the above process produces a multitude of models, each trained under different conditions, we will refer to them using combinations of labels denoting the adaptation strategy used, and labels denoting the prior model used by the adaptation strategy. The labels are listed in Table 1.

Since we are interested in adaptation methods that allow for a quick adaptation of a model from the source task/domain to the target task/domain, we intentionally limit the size of of the training (and validation) set in the target domain. The scarcity of training samples makes it harder for the model to adequately generalize in the target domain, and thus it increases the potential benefit of adapting a model from another domain (although the actual benefit of course depends on the resemblance of the domains).
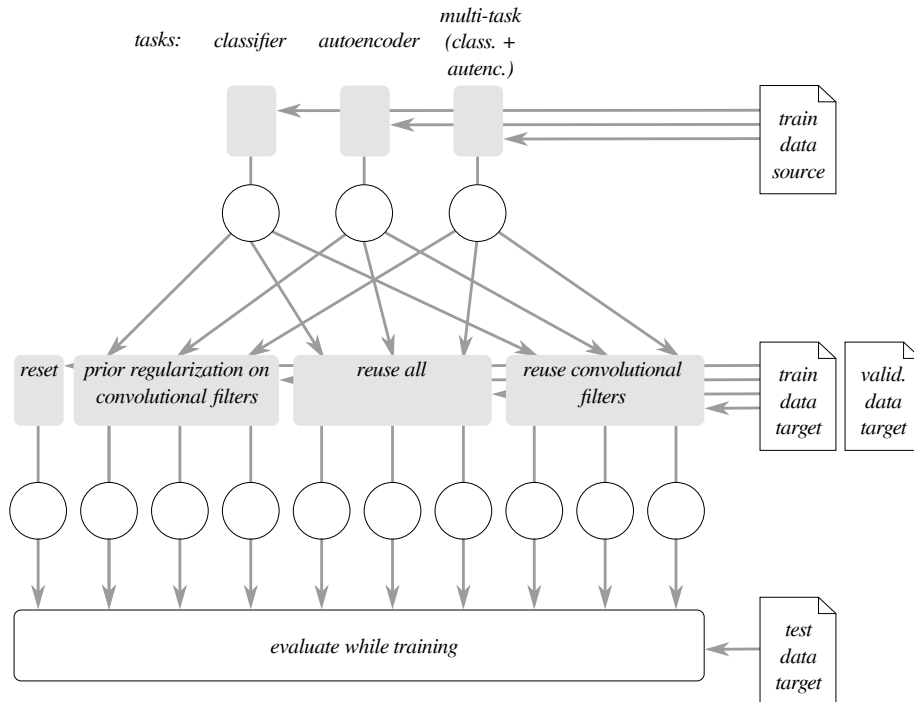
Figure 3: Schematic overview of the experiment for a single data set; grey rounded boxes represent training methods, circles represent models, document shapes represent data instances, and the white rounded box represents the evaluation method

4.1 Data Sets

MNIST   The Mixed National Institute of Standards and Technology (MNIST) database consists of handwritten digits collected by American high school students and employees of the United States Census Bureau [LeCun et al., 1998]. This database constitutes one of the most used data sets for benchmarking machine learning algorithms [Bishop, 2006]. The MNIST consists of 70,000 gray-scaled images, rescaled to fit in a 20×20 pixel box, and then centred in a 28×20 box. For these experiments, MNIST was divided into two subsets, namely Data Set 1, consisting of digits $\{0, 1, 2, 3, 4\}$, and Data Set 2, consisting of digits $\{5, 6, 7, 8, 9\}$. The training set is divided into 50,000 examples for computing the parameter updates, 10,000 examples for validation and 10,000 examples for testing. Data Set 1 contains 25,538 examples for training, 5,058 for validating and 5139 for testing. For Data Set 2, 10 samples per class were randomly selected for both training and validating, making a total of 50 examples for training, 50 for validating and 4861 for testing.

CIFAR-10   The CIFAR-10 is a labelled subset of the *80 million tiny images* data set collected by Krizhevsky, Nair and Hinton [Krizhevsky, 2009]. This data set has been used to evaluate the performance of algorithms in machine learning and computer vision. CIFAR-10 consists of 60,000 32×32 colour (RGB) images

divided into 10 classes, that comprising vehicles and animals. In this paper, Dataset 1 was chosen to include all instances of classes "airplane", "automobile", "bird", "cat", and "deer", while Data Set 2 consists of classes "dog", "frog", "horse", "ship", and "truck". Each training set was split into training and a validation sets, consisting in 75% (37,500 examples) and 25% (12,500 examples) of the data, respectively, while the test set contains 10,000 examples. Data Set 1 consists of 18,681, 6,319 and 5,000 examples, respectively for training, validating and testing, while Data Set 2 contains 10 randomly selected samples per class for training and 10 randomly selected samples per class for testing, to make to make a total of 50 training examples, 50 validating examples and 5,000 examples for testing.

Composers As an application of the proposed methods in a musical domain, a data set consisting of excerpts of musical pieces of the baroque and classical periods was used. These excerpts are represented as piano-rolls, i.e., images where each pixel in the y axis corresponds to a musical note (using the MIDI note number convention), and each pixel in the x axis represents a unit of time. The scores where taken from Muse Data database[5], an electronic library of classical music scores created by the Center for Computer Assisted Research in the Humanities at Stanford University. Each excerpt has a length of 50 quarter notes, with a sample rate of a 32nd note (an 8th of a quarter), with a hop size of 10 quarter notes between contiguous windows. All pieces have been centred to fit in a MIDI range of 68 notes. To signalize the end of a note, the its last 32nd is left blank. Data Set 1 consists of a selection from excerpts from G. P. Telemann's Cantatas, J. S. Bach's Cantatas and G. F. Händel's Concerti Grossi and Trio Sonatas, while Data Set 2 consists of excepts from string quartets by F. J. Haydn and W. A. Mozart[6]. All excerpts coming from a single piece[6] appear only in either the training, validation or test sets, i.e., there are no pieces that appear on more than one data set. There are a total of 2073 training examples, 1393 validation examples and 993 testing examples, with Data Set 1 containing 1141, 755 and 541 examples for training, validating and testing, respectively and Data Set 2 consisting of 10 randomly selected samples per class for training and 10 randomly selected samples per class for validation, to make a total of 20 examples for training, 20 examples for validating and 452 examples for testing.

4.2 Model training

All models were trained using RMSProp [Dauphin et al., 2015], a derivative of the traditional backpropagation algorithm. This method is a mini batch variant of stochastic gradient descent that adaptively adjusts the learning rate by dividing the gradient by an average of its recent magnitude. In order to accelerate gradient descent, we use Nesterov's method for accelerating gradient descent [Sutskever et al., 2013]. In order to avoid overfitting, several strategies are used, including l_2-norm weight regularisation, enforcing sparseness in layer activations, early stopping and dropout. Regularisation of the l_2 norm enforces sparse parameters [Bishop, 2006], while the sparsity in layer activations was enforced using Hoyer's sparseness measure [Hoyer, 2004]. Dropout prevents

---

[5]http://www.musedata.org
[6]Movements of a work are considered individual pieces.

overfitting and provides a way of approximately combining different neural networks efficiently by randomly removing units in the network, along with all its incoming and outgoing connections [Srivastava et al., 2014; Hinton et al., 2012]. The network architectures and hyper-parameters were empirically selected by optimizing the models to their respective validation sets.

MNIST   The classifier consists of a CNN with a convolutional layer with 32 kernels of size 5×5 and sigmoid activations (l_1), followed by a max-pooling layer with a pool size of 2×2 (l_2). The convolutional stage is followed by a classification stage consisting of a fully connected layer with 40 sigmoid units and (l_3) and a fully connected layer with 10 softmax units. Dropout is used after the convolutional stage, i.e., after l_2.

The learning rate for RMSProp is set to 10^{-5}, Nesterov's momentum to 0.5, the probability of dropout is set to 0.5, the regularisation coefficient is 0.001, the sparsity coefficient is set to 10^{-4}, the target sparseness is 0.9 and the batch size is 50. The network was trained for a maximum of 2000 epochs, with a maximum of 200 epochs from the best result for early stopping. The multi-criterion weighting coefficient is set to 0.01 in multi-task models.

CIFAR-10   The classifier consists of a CNN with a convolutional layer with 32 kernels of size 5×5 and sigmoid activations (l_1), followed by a max-pooling layer with a pool size of 2×2 (l_2). The classification stage consists of a fully connected layer with 40 sigmoid units and (l_3) followed by a fully connected layer with 10 softmax units. As in the previous case, dropout is used after the convolutional stage, i.e., after l_2.

The learning rate for RMSProp is set to 10^{-5}, Nesterov's momentum to 0.5, the probability of dropout is set to 0.5, the regularisation coefficient is 0.001, the sparsity coefficient is set to 10^{-4}, the target sparseness is 0.9 and the batch size is 50. The network was trained for a maximum of 3000 epochs, with a maximum of 200 epochs from the best result for early stopping. In multi-task models, the multi-criterion weighting coefficient is set to 0.01.

Composers   The classifier consists of a CNN with a convolutional layer with 9 kernels of size 9×9 and rectified linear activations (l_1), followed by a max-pooling layer with a pool size of 2×2 (l_2), then a second convolutional layer with 5 kernels of size 5×5 with rectified linear activations (l_3), followed by a max-pooling layer with a pool size of 2×2 (l_4). The convolutional stage is followed by a fully connected layer with 256 rectified linear units and (l_5) and fully connected layer with 4 softmax units (l_6), both conforming the classification stage. In a similar fashion as in the previous cases, dropout is used after the convolutional stage, i.e., after l_4.

The learning rate for RMSProp is set to 10^{-6}, Nesterov's momentum to 0.5, the probability of dropout is set to 0.5, the regularisation coefficient is 0.001, the sparsity coefficient is set to 10^{-4}, the target sparseness is 0.5 and the batch size is 50. All networks was trained for a maximum of 2000 epochs, with a maximum of 200 epochs from the best result for early stopping. For the models solving the joint classifier and autoencoding task, the multi-criterion weighting coefficient is set to 0.01.

5 Results and discussion

The results of the various adaptation scenarios and data sets are displayed in Figures 4, 5, and 6. Figure 4 shows the classification accuracies for the classification task $\mathcal{T}_t^{CL}$ in the target domain $\mathcal{D}_t$, for each of the three data sets. The fact that the accuracy curves are rather noisy is because the classification accuracy is not the training objective, but rather the categorical cross-entropy of the model output with the one-hot representation of the class labels (plotted in Figure 5). Furthermore, the occasional discontinuities in the curves are due to the averaging over multiple runs, where some runs converge (and thus halt) after fewer epochs than others.

The first baseline strategy (RESET) consists in a random re-initialisation of all parameters (conform Section 3.4.4), meaning that no knowledge from the source task $\mathcal{T}_s$ in the source domain $\mathcal{D}_t$, is used at all. The second baseline strategy (REUSE ALL), is to initialize the model with the parameters of the model trained on $\mathcal{T}_s$ in the source domain $\mathcal{D}_t$. Note that there are three source tasks (CL, AE, and MT).

In Figure 5, there is a consistent trend that the REUSE ALL baseline strategy is slow to adapt, independent of the source task. Interestingly, among the REUSE ALL conditions (dotted lines), the adaptation of the autoencoder model of the source domain—REUSE ALL (AE)—is more beneficial to learning the target classification task than the adaptation of the classifier—REUSE ALL (CL)— from the source domain. This may be an indication that the autoencoder learns representations that are useful for encoding the data in general, whereas the classifier learns more specialised features that are useful primarily for recognising the specific classes that happen to be in the source domain. To some degree, this finding underlines the rationale for *unsupervised pre-training* [Erhan et al., 2010], that learning to encode the data independent of any classification objective provides more robust representations (for subsequent classifcation) than driving the representations by a classification objective (with different class labels) from the start. The fact that the unsupervised-pretraining—REUSE ALL (AE)—does not surpass the RESET strategy suggests that the distribution of the data that the autoencoder has seen is not sufficiently representative for the target domain, i.e. $P_s(X) \neq P_t(X)$.

The low loss values of REUSE ALL (AE) with respect to REUSE ALL (CL) and REUSE ALL (MT) does translate into higher accuracy rates (Figure 4) for the CIFAR and Composers data sets, but surprisingly, it does not for the MNIST data set.

The REUSE CF strategy (Figure 5, dashed lines), which retains the first layer of convolutional filters learnt from the source domain, but resets all other parameters of the model, considerably speeds up adaptation to the target task with respect to the RESET baseline. With this strategy, the adaptation from the classification task in the source domain is generally most successful, but even when adapting models from the AE and MT source tasks, the adaptation is mostly quicker than RESET. A plausible explanation for this is that the low level structure of the data is usually more general, whereas the higher level structure (say, the configuration of lower level features into shapes) is more specific to particular data categories.

For the autoencoding task $\mathcal{T}_t^{AE}$ in the target domain $\mathcal{D}_t$ (Figure 6), the REUSE ALL (AE) strategy leads to substantially better autoencoding results

from the start in the target domain, but as opposed to the other adaptation strategies, the training of the model using this strategy does hardly improve the autoencoding loss beyond its initial value. The REUSE ALL (CL) strategy on the other hand, adapts much worse on than the RESET baseline on CIFAR and MNIST, on a par with REUSE CF (CL).

The autoencoder task in the target domain of the Composers data set shows a very different pattern (Figure 6, right). Here, all adaptation strategies outperform the RESET baseline that discards any knowledge from the source domain. This may be an indication that the source and target domains in the Composers data set have rather similar marginal distributions, i.e., $P_s(X) \approx P_t(X)$, giving the adaptation strategies an advantage over the RESET baseline. An explanation for this may be that there is a lot of structure in the musical data that is common across the music of the different composers (tonal structure, rhythmical structure). The prevalence of this common structure would also explain the rather low overall accuracy in the composer classification task (Figure 4, right).

The simultaneous learning of both autoencoding and classification in the source domain appears to provide more useful knowledge to be transferred to the autoencoding target task (Figure 6), than the classification target task (Figures 4, and 5). In the latter case, adaptation is more successful when the source task is also classification.

In the prior regularization strategy (RESET PRF), the first layer convolutional filters of a randomly initialised network are biased towards the filters learned from the source domain and task. This regularization does improve a bit upon the RESET baseline sometimes, but not very consistently, and the gains are usually moderate.

One issue that we have not discussed is whether the asymptotic performance of some strategies is substantially different from that of others. From the Figures it is clear that broadly speaking, the adaptation strategies converge to a certain range, even if there are individual differences. To make better judgements on the asymptotic performance of the adaptation strategies however, it is necessary to do more extensive experimentation, with longer training phases, more data sets, and averaging results from more runs of the same condition. Moreover, although the hyperparameters of the models (e.g. the number of convolutional filters, the pooling size, regularisation constants) were selected through exploration prior to the experiment, a more exhaustive study of the hyperparameters may provide a better view on the value of the adaptation strategies. From a computational point of view, this is a considerable undertaking, given that the results presented here require several weeks of continuous computation on multiple GPGPU-enabled machines.
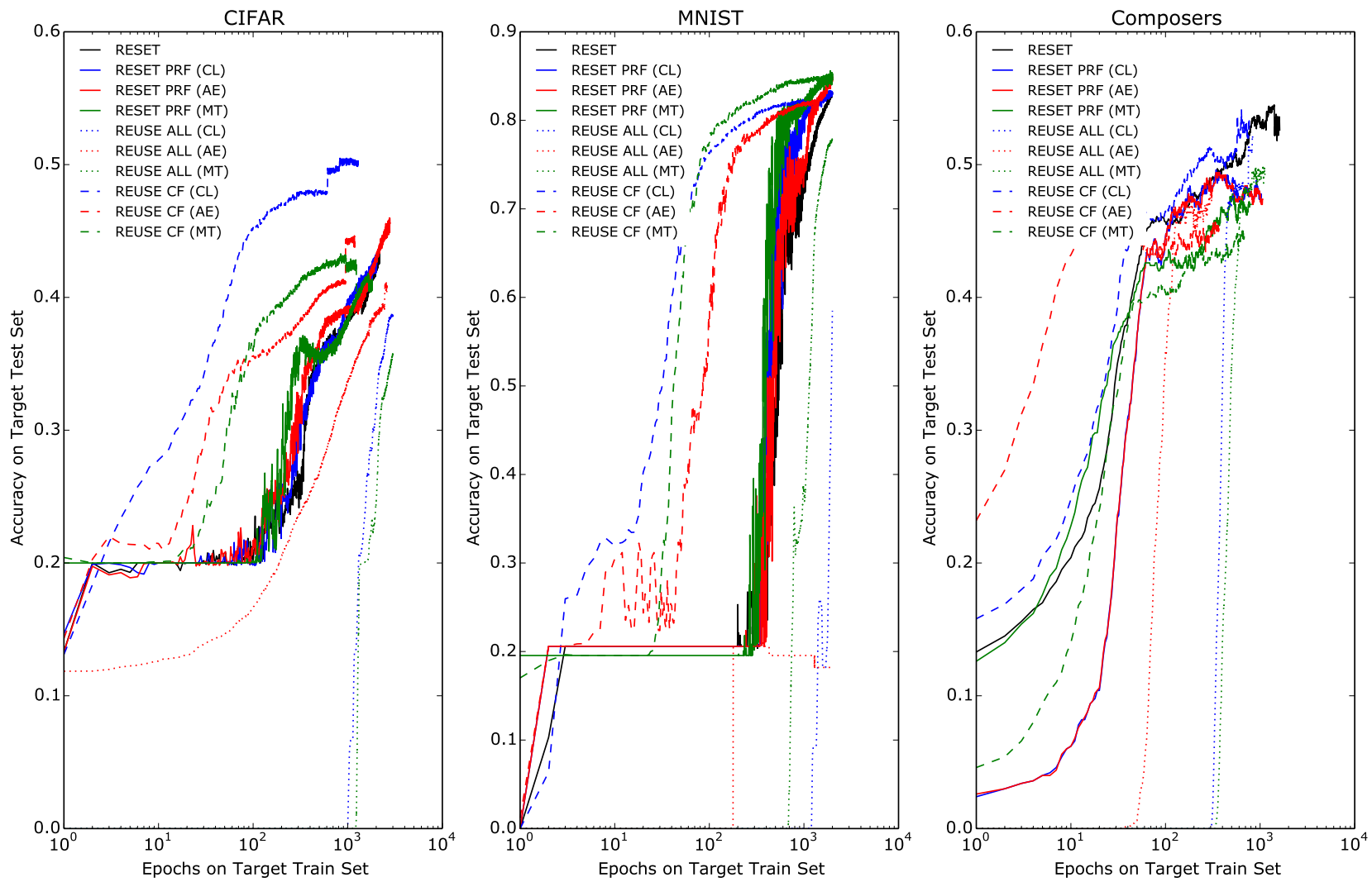
Figure 4: Classification accuracy (proportion of correctly classified instances) on the target test-set of the different data sets, for different adaptation strategies. Curves are averaged over two runs. See table 1 (page 15) for an explanation of the legend
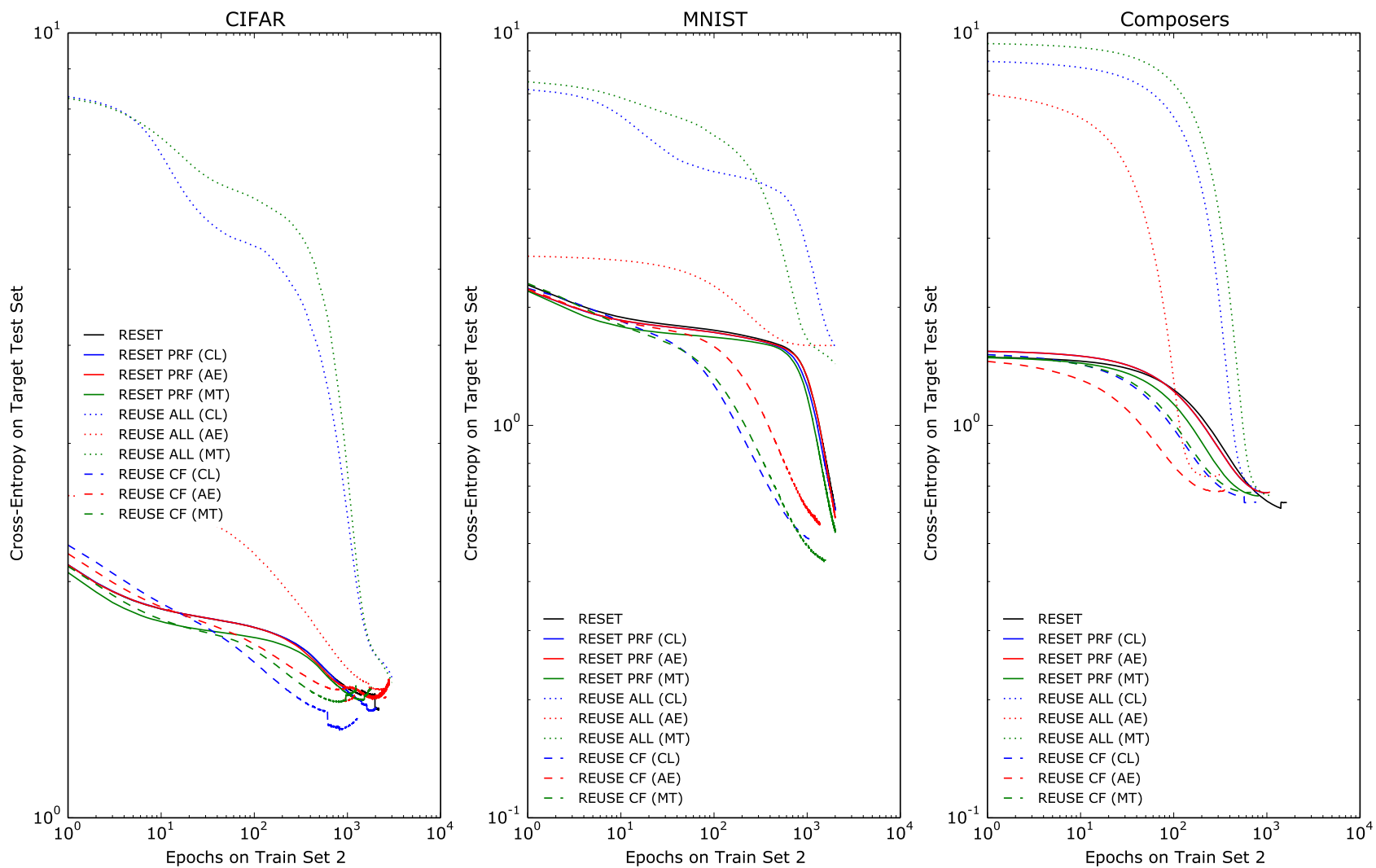
Figure 5: Classification loss (categorical cross-entropy) on the target test-set of the different data sets, for different adaptation strategies. Curves are averaged over two runs. See table 1 (page 15) for an explanation of the legend
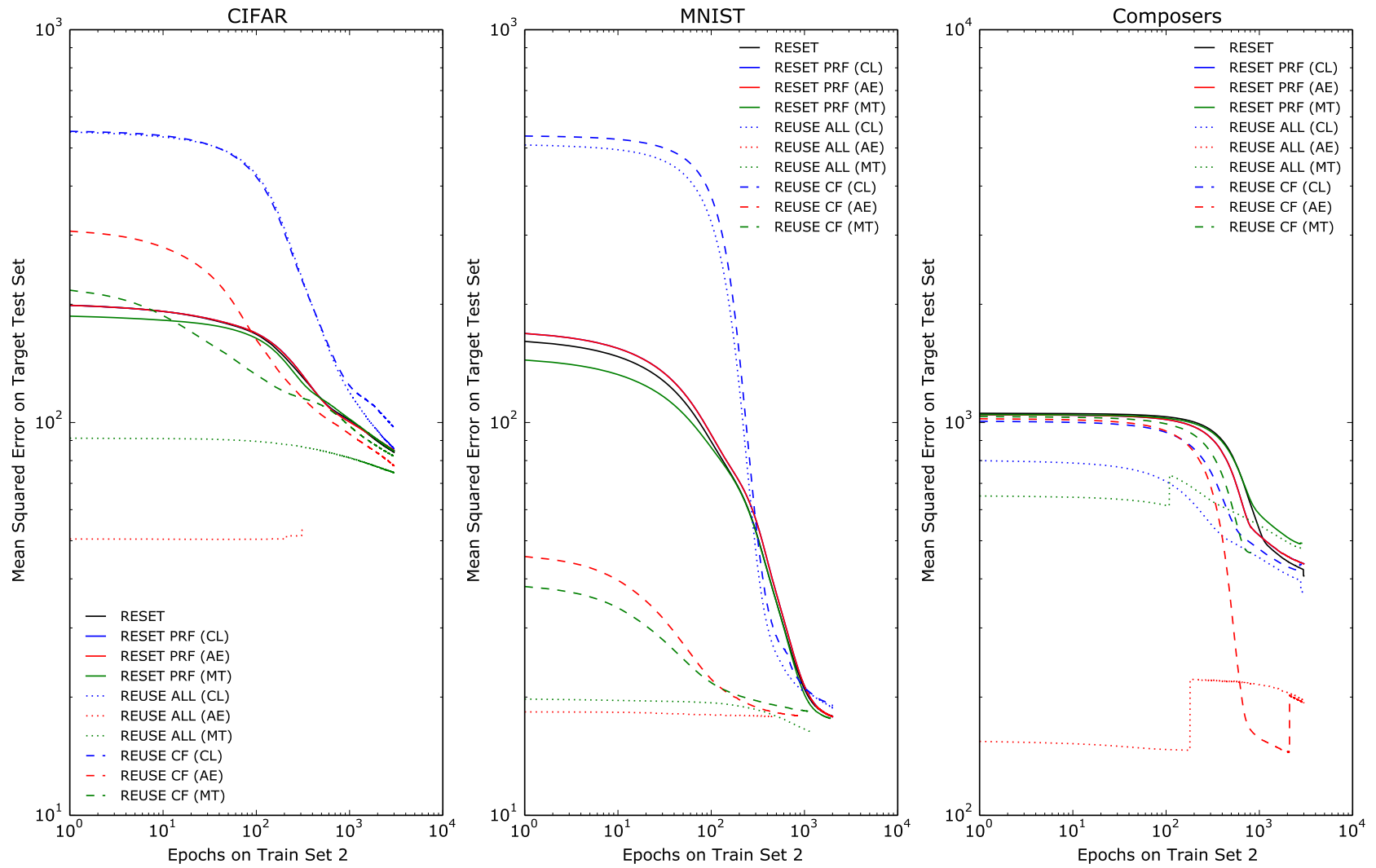
Figure 6: Autoencoding loss (mean squared error) on the target test-set of the different data sets, for different adaptation strategies. Curves are averaged over two runs. See table 1 (page 15) for an explanation of the legend

6   Conclusions

When interacting with a dynamical and unpredictable environment, the ability of an agent to easily adapt the conceptual space with which it interprets the environment, is a strong advantage. As argued in Section 1.1, this adaptability may also be one of the underlying mechanisms of creative behaviour. In this document, we have described and evaluated several adaptation strategies to transform learned representations optimized for a specific task in a specific domain (the *source* task/domain), to another task in another domain (the *target* task/domain). We have evaluated the strategies using a convolutional neural network, on multiple data sets, comprising natural images, handwritten digits, and classical music.

The results show that, across domains and tasks, adaptation strategies that transform existing representations allow for a quicker adaptation to a new task in a new domain than starting the representation learning from scratch. Although there is no single adapatation strategy that is universally superior to others, some clear patterns do emerge from the results. Firstly, when the target task is classification, then a successful adaptation strategy is to keep the first level convolutional filters (i.e. the lower level representations) from the source task/domain, and reset the rest of the parameters. This strategy is even beneficial when the source task is autoencoding, rather than classification. Secondly, for the autencoding target task, the reuse of the full model (rather than just the convolutional filters) is a successful strategy, in the sense that even from the start, the task performance with that strategy is comparable with the asymptotic performance of other strategies. But in contrast to the other strategies, this strategy does not substantially improve the model beyond the initial performance. The experiments do not provide evidence for a strong advantage of the prior regularization of the convolutional filters (Section 3.4.3), over the condition where the model is trained from scratch with only standard regularization.

More elaborate experiments are required to provide firmer conclusions on the merits of each of the adaptation strategies. In particular, a longer training phase is necessary to provide better insight in the asymptotic behaviour of the strategies.

Acknowledgement

References

Allen, J. (2012). *The Lives of the Brain: Human Evolution and the Organ of Mind*. Harvard University Press.

Atick, J. J. (1992). Could information theory provide an ecological theory of sensory processing? *Network*, 3:213–251.

Baars, B. J. (2002). The conscious access hypothesis: origins and recent evidence. *Trends in cognitive sciences*, 6(1):47–52.

Bacchiani, M. and Roark, B. (2003). Unsupervised language model adaptation. *International Conference on Acoustics, Speech and Signal Processing*, 1:224–227.

Barlow, H. (1989). Unsupervised learning. *Neural Computation*, 1(3):295–311.

Ben-David, S., Blitzer, J., Crammer, K., and Pereira, F. (2007). Analysis of representations for domain adaptation. In *Advances in Neural Information Processing Systems 19*.

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

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer Verlag, Microsoft Research Ltd.

Boden, M. A. (1994). Précis of the creative mind: Myths and mechanisms. *Behavioral and brain sciences*, 17(3):519–531.

Boden, M. A. (2004). *The Creative Mind: Myths and Mechanisms, Second Edition*. Routledge, London, 2nd edition.

Boyd, S. P. and Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press.

Caruana, R. (1997). Multitask learning. *Machine Learning*, 28:41–75. 10.1023/A:1007379606734.

Chelba, C. and Acero, A. (2006). Adaptation of maximum entropy capitalizer: Little data can help a lot. *Computer Speech & Language*, 20(4):382–399.

Csikszentmihalyi, M. (1996). *Creativity: Flow and the Psychology of Discovery and Invention*. Modern classics. HarperCollinsPublishers.

Daumé III, H. (2007). Frustratingly easy domain adaptation. In *Conference of the Association for Computational Linguistics (ACL)*, Prague, Czech Republic.

Daumé III, H. and Marcu, D. (2006). Domain Adaptation for Statistical Classifiers. *Journal of Artificial Intelligence Research*, pages 101–126.

Dauphin, Y. N., de Vries, H., Chung, J., and Bengio, Y. (2015). RMSProp and equilibrated adaptive learning rates for non-convex optimization. *arXiv*, 1502:4390.

Erhan, D., Bengio, Y., Courville, A., Manzagol, P.-A., Vincent, P., and Bengio, S. (2010). Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research (JMLR)*, 11:625–660.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, pages 249–256.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Domain adaptation for large-scale sentiment classification: A deep learning approach. In Getoor, L. and Scheffer, T., editors, *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 513–520, New York, NY, USA. ACM.

Hinton, G. E., Osindero, S., and Teh, Y. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv*, 1207:580.

Hofstadter, D. R. (2008). *Fluid Concepts and Creative Analogies: Computer Models of the Fundamental Mechanisms of Thought*. Basic Books.

Hoyer, P. O. (2004). Non-negative matrix factorization with sparseness constraints. *arXiv.org*, page 8058.

Jiang, J. (2008). A literature survey on domain adaptation of statistical classifiers. Technical report, University of Illinois Urbana-Champaign.

Krizhevsky, A. (2009). Learning Multiple Layers of Features from Tiny Images. Technical report.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems*, pages 1106–1114.

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.

Masci, J., Meier, U., Cireşan, D., and Schmidhuber, J. (2011). Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction. In Honkela, T., Duch, W., Girolami, M., and Kaski, S., editors, *Artificial Neural Networks and Machine Learning - ICANN 2011*, pages 52–59.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning. *arXiv.org*, page 5602.

Olshausen, B. and Field, D. (1996). Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381:607–609.

Pan, S. J. and Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359.

Schmidhuber, J. (2010). Formal theory of creativity, fun, and intrinsic motivation (1990–2010). *Autonomous Mental Development, IEEE Transactions on*, 2(3):230–247.

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*, 2014(15):1929–1958.

Sutskever, I., Martens, J., Dahl, G., and Hinton, G. E. (2013). On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning*, Atlanta, Georgia, USA.

Torrey, L. and Shavlik, J. (2009). Transfer learning. In Soria, E., Martín, J., Magdalena, R., Martinez, M., and Serrano, A., editors, *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques*. IGI Global.

van Hasselt, H., Guez, A., and Silver, D. (2015). Deep Reinforcement Learning with Double Q-learning. *ArXiv e-prints*.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. PhD thesis, King's College, Cambridge.

Weisberg, R. (1993). *Creativity: Beyond the Myth of Genius*. Books in psychology. W.H. Freeman.

Widmer, G. and Kubat, M. (1996). Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101.

Wiggins, G. and Forth, J. (2015). Idyot: A computational theory of creativity as everyday reasoning from learned information. In Besold, T. R., Schorlemmer, M., and Smaill, A., editors, *Computational Creativity Research: Towards Creative Machines*, volume 7 of *Atlantis Thinking Machines*, pages 127–148. Atlantis Press.

Wiggins, G. A. (2006). A preliminary framework for description, analysis and comparison of creative systems. *Knowledge-Based Systems*, 19(7):449–458.

A Activation and Loss functions

In this section, we provide definitions of the nonlinear activation and loss functions mentioned in the main text.

A.1 Nonlinear activation functions

The follow nonlinear activation functions are defined as scalar functions as $f \colon \mathbb{R} \mapsto \mathbb{R}$. Following conventions in the machine learning literature [Bishop, 2006], when applied to general tensors (i.e., vectors, matrices or higher order tensors), these functions take the form $f \colon \mathbb{R}^{N_1 \times \cdots \times N_p} \mapsto \mathbb{R}^{N_1 \times \cdots \times N_p}$, with the function being applied elementwise.

1. Sigmoid

$$\sigma(y) = \frac{1}{1 + \exp(-y)} \tag{11}$$

2. Hyperbolic tangent

$$\tanh(y) = \frac{\exp(y) - \exp(-y)}{\exp(y) + \exp(-y)} = 2\sigma(2y) - 1 \tag{12}$$

3. Rectified Linear Units

$$\mathrm{ReLU}(y) = \begin{cases} y & \text{for } y > 0 \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

4. Softmax. Given an input $\mathbf{y} \in \mathbb{R}^{N_1 \times \cdots \times N_p}$, the softmax activation function is given by

$$\mathrm{Softmax}(y_{j_1 \ldots j_p}) = \frac{\exp(y_{j_1 \ldots j_p})}{\sum_{k_1=1}^{N_1} \cdots \sum_{k_p}^{N_p} \exp(y_{k_1 \ldots k_p})}, \tag{14}$$

where $y_{j_1 \ldots j_p}$ is the $j_1 \ldots j_p$-th element of $\mathbf{y}$.

A.2 Loss functions

Let $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ be a set of inputs, whose corresponding outputs are given by $\mathbf{y}(\mathbf{x}_i; \boldsymbol{\theta})$ and $\mathbf{T} = \{\mathbf{t}_1, \ldots, \mathbf{t}_N\}$ a set of targets, with both $\mathbf{y}, \mathbf{t} \in \mathbb{R}^{N_1 \times \cdots \times N_p}$.

1. Mean Squared Error

$$L_{MSE}(\boldsymbol{\theta}) = \sum_{n=1}^{N} \|\mathbf{t} - \mathbf{y}(\mathbf{x}_n; \boldsymbol{\theta})\|_2^2 \tag{15}$$

2. Categorical Cross Entropy

$$L_{CCE}(\boldsymbol{\theta}) = -\frac{1}{N} \sum_{n=1}^{N} \sum_{k_1=1}^{N_1} \cdots \sum_{k_p=1}^{N_p} t_{n;k_1 \ldots k_p} \log \left(y_{k_1 \ldots k_p}(\mathbf{x}_n; \boldsymbol{\theta}) \right), \tag{16}$$

where $t_{n;k_1 \ldots k_p}$ represents the $k_1 \ldots k_p$-th component of the n-th element of set $\mathbf{T}$.