Active learning in VAE latent space

Tonnaer, L.M.A.

Award date:
2017

Technische Universiteit
Eindhoven
University of Technology

Department of Mathematics and Computer Science
Data Mining Research Group

Active Learning in VAE Latent Space

Master Thesis

L.M.A. Tonnaer

Supervisors:
dr. V. Menkovski
dr. J. W. Portegies
dr. M. J. Holenderski

Final version

Eindhoven, August 2017

Abstract

Active learning deals with incrementally obtaining more labelled data for supervised learning by querying an oracle (e.g. a human annotator) to provide labels for unlabelled data instances. This involves retraining a learning model many times, whenever newly labelled data is obtained, in order to make new predictions that allow us to evaluate how informative certain unlabelled data instances are if we were to obtain their label. For high-dimensional data such as images however, models often take long to train, in particular (convolutional) neural networks. Thus, having to retrain complex neural networks after each newly queried label is very slow, and often infeasible in practice.

Although active learning deals with situations where labelled data is scarce, it often is possible to obtain large amounts of unlabelled data. We propose an active learning scheme for image data that fully utilises this pool of unlabelled data by first learning useful representations from it. We use a (convolutional) variational autoencoder (VAE) to obtain latent space representations of all the data. Such a VAE is trained under the assumption that the latent variables come from some give prior distribution, which results in a latent space that provides a more structured and lower-dimensional representation of the data. These properties allow us to use a relatively simple and fast classification neural network, a multilayer perceptron (MLP), in an active learning setting. This way can perform much faster active learning iterations than if we would have to retrain a more slow and complex neural network in every iteration.

Our experiments on the MNIST and SVHN image data sets show that it is possible to train simple MLPs that achieve decent accuracy on latent space representations obtained by convolutional VAEs. We show that while performing active learning in latent space, querying labels by means of a technique called uncertainty sampling can provide significant improvements over passive learning (i.e. querying random samples from the unlabelled data). Moreover, the latent space seems to make (active) learning more reliable in the sense that there are less fluctuations in classification accuracy between different experiments.

We also investigate whether modelling representativeness explicitly can prevent uncertainty sampling from querying outliers, but we observed no improvements in performance for the methods we used.

To further speed up the active learning iterations, we query batches of informative instances at once, rather than single instances, before retraining the classification model. We evaluate some diversity techniques that aim to reduce redundancy in batches, which only leads to small improvements in a limited number of situations.

Keywords: active learning, machine learning, deep learning, variational autoencoder (VAE), latent space, uncertainty sampling

Preface

This master thesis signifies the conclusion of my Computer Science and Engineering studies at Eindhoven University of Technology. I executed my graduation project internally at the Data Mining research group of the Mathematics and Computer Science department. I started my student career with an Industrial and Applied Mathematics bachelor, before entering the field of Computer Science. Here I soon discovered the endless opportunities of data science, and the fascinating domains of machine learning and deep learning in particular. Although I learned a lot about these domain during my master's degree, I have also learned that this is just the start. There will always remain much to be learned about data science and machine learning, and I am convinced that it will only become more and more relevant in all of our lives.

I would first of all like to thank Vlado Menkovski for his guidance during this project, for providing ideas and inspiration for the topic of this research, and for bringing together all those fellow students and researchers working on deep learning topics during interesting weekly meetings. I would also like to thank all professors and educators at the university, as well as my fellow students, for inspiring me and helping me grow and learn during my student years.

Furthermore I would like to thank my family, my parents in particular, for supporting me and encouraging me to pursue higher education, for providing me with opportunities that should not be taken for granted. I also want to thank my girlfriend Elene for inspiring and motivating me, believing in me, and for just providing daily support and comfort while I was working on this thesis.

Lastly, I would like to thank Joos Buijs and Thijs Nugteren for kindly making their LaTeX template, on which this thesis is based, publicly available.

Loek Tonnaer
Eindhoven,
August 2017

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Machine learning has allowed us to accurately approximate complicated functions by training models on large amounts of data that exemplify the function that needs to be learned. In order for machine learning to be successful we need a sufficient amount of labelled data, where each data instance is associated with a label that describes the required output. Such an output can for instance be one of a fixed amount of classes (classification), or some real-valued number (regression). We refer to learning from such labelled data as *supervised learning*. When dealing with high-dimensional data, such as image data, we tend to need particularly large amounts of labelled data before machine learning models can learn well (this is sometimes referred to as the *Hughes phenomenon* [23]).

In many real-life situations however, not much labelled data is available. It is often possible though to obtain large amounts of unlabelled data, which can be labelled by e.g. human annotators. Annotating the data can be costly however, as it often requires human labour, possibly by domain experts.

The field of *active learning* deals with obtaining labels in an efficient way, such that with a relatively small amount of cleverly chosen labelled instances a high-performing model can be trained. Many methods leverage the fact that lots of unlabelled data can usually freely (or cheaply) be obtained, even if labelling it is expensive. This unlabelled data is used to e.g. perform realistic sampling from the underlying data distribution, or to compute expected informativeness measures for all available data instances. Such heuristics are generally computed before any learning takes place; only when new data has been labelled will a new model be trained, and training is done *only* with labelled data.

Many *unsupervised learning* techniques exist however that can learn meaningful features and representations of unlabelled data, in particular for high-dimensional data such as images, where standard (pixel) representations have no immediate link to the shapes and semantics of the image. The field that deals with such techniques is often called *representation learning* or *feature learning* [6]. Such learned representations can bring more (semantic) structure to the data, and can allow us to learn information from the unlabelled data that can later be useful for supervised tasks such as classification or regression, in particular when labelled data is scarce [6].

These representations can then also provide information on the informativeness of unlabelled samples, in the context of active learning. Representation learning models tend to group together similar instances in its feature space. We can use these similarities in active learning strategies to make predictions for unlabelled instances, based on our current labelled data. Thus, we can form strategies for querying unlabelled data instances to be labelled by an annotator that are based not only on a model trained on our currently available labelled data (which is often limited), but also on representations learned from the usually much larger set of unlabelled data.

In this thesis we explore how a powerful recent representation learning technique, the *variational autoencoder* (VAE), can improve active learning strategies for classification of image data. We choose the VAE because it has been shown to yield good representations for images in a latent

space that is densely populated by the data, and where similar instances are often close together. The main reason for these properties is that the VAE encodes data instances to distributions in the latent space, rather than to single coordinates, and that it is trained under the assumption that these distributions follow a certain predefined prior distribution. Moreover, the VAE is actually a generative model, which can be used to generate new sensible data instances (images) from points in its (learned) latent space. This provides interesting opportunities to extend the approach in this thesis into a different active learning framework. Since we work with image data, we consider VAEs that contain convolutional layers as well as fully connected layers.

We use VAEs on large sets of unlabelled image data to obtain latent space representations, which we then use in an active learning setting. Figure 1.1 illustrates the main idea of our approach. First we query a random batch to obtain a first set of labelled latent data. We then train a fairly simple classification model, a multilayer perceptron (MLP), on the currently labelled data (in latent space). With this MLP we can obtain predictions for the remaining unlabelled data. Using these predictions we can evaluate the expected informativeness of unlabelled data instances if we were to obtain their labels. We do this by evaluating the *uncertainty* of the model about how to label an instance, where we query those instances for which the model is most uncertain. We then obtain more labels for our data, which allows us to retrain an MLP in order to obtain better predictions for the unlabelled data. This in turn allows us to re-evaluate the informativeness of unlabelled instances if we would know their labels. Thus we can incrementally obtain more labels in an effective way, such that our overall classification score is better than if we would query random instances from the data instead.

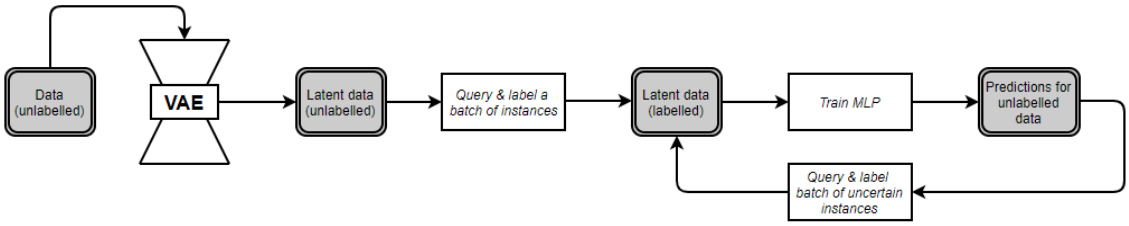


Figure 1.1: The main workflow of the active learning strategy

Using uncertainty to evaluate informativeness in an active learning setting is a computationally inexpensive method. A downside however is that it is prone to querying outliers; data instances that are not *representative* of the underlying data distribution. By means of a VAE we can obtain a latent space representation of the data in which all data is conditioned to be close to a given prior distribution. This makes all data points more representative in this latent space representation, such that querying outliers becomes less of a problem. We propose a number of ways to express the *representativeness* of an instance in latent space, as well as in the original (pixel) space, in order to compare how the data is distributed in each of these spaces. We can also use these measures explicitly by only querying instances that score high on both uncertainty and representativeness.

Training a (convolutional) VAE first on all the unlabelled data allows us to learn much useful information (in the form of powerful representations) right at the start, using all available data. Having learned these representations already, we can then use simple and fast models such as MLPs for active learning, where we are required to retrain a model often, whenever new labels are available. This makes the active learning significantly faster. Having to retrain an MLP every time we obtain a new label however is still rather slow. Therefore we query batches of instances to be labelled at once, such that we only need to retrain the MLP for each new batch of labels. This introduces a new challenge however. Similar instances are likely to have similar uncertainty. Obtaining labels for two very similar instances that both have high uncertainty however seems inefficient; as soon as we know the label for one of the two, we likely won't need to label the other one as well. Therefore it seems appropriate to construct batches of uncertain (and representative) instances that contain *diversity*, i.e. we want low redundancy within batches. We propose a

number of simple heuristics that encourage diversity in query batches, as well as greedy methods that can construct batches with low diversity based on some distance measure in latent space.

Our experiments indicate that training on latent space representations obtained through a (convolutional) VAE can improve the performance of a simple classification model such as an MLP. It turns out that using uncertainty sampling to query data instances to be labelled can result in significant performance improvements over querying random samples of data, when training in latent space. Moreover, active learning in latent space shows more stable and reliable results than active learning in the original image space.

We do not however observe any improvements when incorporating representativeness measures to prevent the querying of outliers. Our techniques for encouraging diversity in query batches show only very limited improvements for certain cases, and no improvement at all in other settings.

The remainder of this thesis is structured as follows. In Chapter 2, we provide an overview of the fields of active learning and representation learning, and go into more detail about the methods relevant for this thesis and the motivation for using these methods. We also refer to some related work that also combines representation learning with active learning. In Chapter 3, we describe in detail how we use VAEs to obtain latent space representations of the data, and how we utilise these representations for active learning. Chapter 4 describes the experimental setup for the evaluation of our proposed methods, as well as the benchmark image data sets on which we ran our experiments. Chapter 5 contains the results and evaluations of our experiments. In Chapter 6 we draw conclusions from our experiments and results. Finally, in Chapter 7, we lay out opportunities for future work that spawn from the observations and conclusions in this thesis.

Chapter 2

Background

This thesis focuses on combining techniques from *representation learning* and *active learning* for *artificial neural networks*. In this chapter we will look into these domains, and elaborate on the approaches most relevant to the work presented in this thesis. We will provide an overview of the current state of art, and address areas of potential improvement.

2.1 Active Learning

Active learning [44] is a domain within *machine learning* that deals with situations where there is a lack of labelled data. The main hypothesis is that if a learning algorithm is allowed to choose the data to be labelled, it can achieve better performance with fewer labels. It is assumed that there is some *oracle* (e.g. a human annotator) from which a label can be queried for a given data point. Obtaining such a label comes at a cost, thus the goal is to achieve good performance while querying few labels. B. Settles [44] provides an extensive literature survey of the various approaches and frameworks that have been proposed in this field. Here we will give an overview of the field, and discuss the aspects most relevant to our own approach.

2.1.1 Query Scenarios

Settles [44] recognises three scenarios that are mostly considered in active learning: membership query synthesis, stream-based selective sampling, and pool-based sampling.

Membership Query Synthesis

In the *membership query synthesis* scenario [2], the learner may request labels for any instance in the input space. This typically means that the learner can create queries on its own (fitting the input space), not utilising any data source. An advantage of synthesising membership queries is that it is often tractable and efficient for finite problem domains [3]. It can however be hard (or impossible) for a human expert to label such arbitrary instances, as was e.g. seen by Lang and Baum [5] when they tried membership query learning with human oracles to train a neural network for handwritten character classification; many of the query images generated by the learner did not resemble any known symbol. Another disadvantage is that such queries do not follow any underlying natural distribution, risking training a learner that does not generalise well.

The approach of this thesis suggests interesting opportunities for addressing both these problems, while still remaining in the scenario of synthesising queries. We will however not investigate this scenario further, mainly for the reason that it is difficult to objectively evaluate the performance of such methods. Possible applications of this scenario are discussed in Chapter 7. Here we instead focus on sampling approaches, as explained in the next sections. In this way, we can use a labelled test set to perform objective evaluation of our methods.

Stream-Based Selective Sampling

Selective sampling [4] is an alternative to synthesising queries that targets some of the limitations described in the previous section. The main assumption here is that an unlabelled instance can be freely (or cheaply) obtained, so it can be sampled from the actual data distribution. The learner can then decide whether or not to request its label. Unlabelled instances are typically drawn one at a time from a data source, thus this scenario is often referred to as stream-based or sequential active learning.

If the input distribution is uniform, selective sampling can be expected to behave similar to query synthesis. If the input distribution however is non-uniform and unknown, selective sampling guarantees that queries are sensible and come from a real underlying distribution; a guarantee that query synthesis does not provide.

This scenario is however still limited in the sense that it requires a decision based on seeing a single data instance, ignoring any information from the rest of the data other than the fact that the current instance is drawn from the actual data distribution. In this thesis we consider situations where large amounts of unlabelled data are available, allowing us to evaluate an entire pool of unlabelled instance at once (instead of sampling instances one by one). This allows for a more powerful scenario that will be explained in the next section, and as such we do not consider the stream-based selective sampling scenario in this thesis.

Pool-Based Sampling

In stream-based selective sampling, as described in the previous section, instances are obtained one at a time. In many real-world learning problems however it is possible to gather large collections of unlabelled data at once. This is the motivation for another scenario called *pool-based sampling* [32], in which it is assumed that there is a small set $\mathcal{L}$ of labelled data available, as well as a big pool $\mathcal{U}$ of unlabelled data. Queries are then selectively drawn from the pool $\mathcal{U}$, typically in a greedy fashion according to some informativeness measure used to evaluate all instances in (a subsample of) the pool.

The pool-based sampling scenario is similar to the stream-based scenario described before, with the main difference that stream-based sampling scans through the data sequentially and makes query decisions on an individual basis, whereas pool-based sampling evaluates and ranks the entire collection before selecting the best query. The pool-based scenario generally appears to be more powerful, but in certain settings when e.g. memory or processing power may be limited the stream-based scenario seems more appropriate.

In this thesis we consider the pool-based sampling scenario, since we are investigating scenarios in which large amounts of unlabelled (image) data are available, and because it provides good possibilities for objective performance evaluation.

2.1.2 Query Strategies

For all query scenarios some informativeness measure is necessary to determine which data instance to query next. In the pool-based sampling scenario in particular this amounts to having some measure to compute the (expected) informativeness for each unlabelled data instance in the pool $\mathcal{U}$, in order to query the highest-scoring instance for this measure. Settles [44] identifies a number of strategies to formulate such a measure:

- *Uncertainty sampling*; query instances of which the learner is least certain about their correct output.
- *Query by committee (QBC)*; maintain a committee of learners, query instances on which the committee disagrees most.
- *Expected model change*; query instances that would most change the current model.

- *Expected error reduction*; query instances that would most reduce the model's generalisation error.
- *Variance reduction*; query instances that would minimise output variance.
- *Density-weighted methods*; query instances based on their representativeness of the input data distribution, as well as a simple "base" strategy such as uncertainty sampling or QBC.

In this thesis we focus on *uncertainty sampling*, possibly extended through *density-weighted methods*, both of which will be described in the next sections. First, however, we briefly discuss the other strategies, and point out why they are less suitable for our situation.

This thesis deals with active learning for (deep) neural networks. Particular challenges that come with neural networks is that they are expensive to (re)train and that the model is often hard to analyse. This poses challenges for many active learning query strategies.

Query by committee [48] requires multiple models to be trained simultaneously, to see on which instances they disagree most. This however increases the time spent on training neural networks during active learning by a factor equal to the number of models in the committee.

Methods based on *expected model change* [47] require the active learner to be able to analyse the effect of labelling certain data instances with certain labels on the model, which is computationally expensive to even approximate when working with neural networks.

Similarly, methods based on *expected error reduction* [39] require expensive computations or approximations, rendering them less suitable for use with neural networks.

The expected future error of a model can however be decomposed into a noise, bias, and variance term [17], where the noise and bias terms cannot be influenced given a fixed model class. Thus, one can instead focus only on *variance reduction*. For this, closed-form solutions or approximations for neural networks do exist in certain cases [13][35]. Such methods generally come down to minimising certain properties of the inverse Fisher information matrix [42], such as its trace, determinant, or maximum eigenvalue.

Uncertainty Sampling

This thesis focuses on *uncertainty sampling* [32], a common and fairly simple query framework in which an active learner queries those instances for which it is least certain how to label them. Its main advantages are that it is simple and computationally inexpensive.

The approach is generally quite straightforward for probabilistic learning models. For binary classification for instance, if using a probabilistic model, then uncertainty sampling simply queries the instance whose posterior probability of being positive is closest to 0.5 [32]. For multiclass classification, one way to generalise this is to query the instance whose prediction is *least confident*, i.e. query the instance x that minimises $P_\theta(\hat{y}|x)$, where $\hat{y}$ is the class label with the highest posterior probability under model θ. This uncertainty measure can be interpreted as the expected 0/1-loss; the model's belief that it will incorrectly label x. This type of strategy has been used with for instance statistical sequence models in information extraction tasks [14].

This least confident strategy however only considers information on the most probable label, thus disregarding information about the remaining label distribution. *Margin sampling* [40] is a method that corrects for this; it queries the instance x that minimises $P_\theta(\hat{y}_1|x) - P_\theta(\hat{y}_2|x)$, where $\hat{y}_1$ and $\hat{y}_2$ are the first and second most probable class labels under the model θ, respectively.

An even more general uncertainty sampling strategy that takes into account the predictions for all the possible classes is based on *Shannon entropy* [49] as an uncertainty measure, i.e. it queries the instance x that maximises the entropy $-\sum_i P_\theta(y_i|x) \log P_\theta(y_i|x)$ over all possible class labels y_i.

Empirical comparisons [27][41][46] of these three uncertainty measures have shown mixed results, which suggests that the best method may be application-dependent. We can for example see that the entropy measure does not favour instances where most of the labels are highly unlikely, since the model is fairly certain those are not the correct labels. The least confident and margin

sampling measures on the other hand consider such instances useful if the model cannot distinguish between the remaining classes. Thus, intuitively, entropy seems more appropriate if we are mainly interested in minimising log-loss, whereas the least confident measure and (particularly) margin sampling seem more fit if we aim to reduce the classification error.

Density-Weighted Methods

The main motivation for the aforementioned expected error and variance reduction frameworks is that they focus on the entire input space, rather than on individual instances. Therefore they are less likely to query outliers than simpler query strategies such as uncertainty sampling, query by committee, and expected model change. With uncertainty sampling for example, the least certain instance may well be an outlier that doesn't really represent the input distribution well. The expected value and variance reduction strategies implicitly avoid such problems by utilising the unlabelled pool $\mathcal{U}$ when estimating future errors and output variances. This tends to result in much more computationally expensive methods however. Another strategy to overcome these problems is to model the input distribution explicitly while selecting queries.

Settles and Craven [46] describe a general density-weighted method technique, the *information density* framework. The main idea is that the informativeness of instances also depends on how representative they are of the underlying distribution, i.e. whether they inhabit dense regions of the input space. To this end they define the density of a particular point in the input space as the average similarity to all data points, or possibly to all data points within the same cluster if some clustering is available. Thus, we wish to query the instance x that maximises

$$\phi_A(x) \cdot \left(\frac{1}{|\mathcal{U}|} \sum_{x_u \in \mathcal{U}} \text{sim}(x, x_u) \right)^{\beta},$$

where $\phi_A(x)$ represents the informativeness of x according to some "base" query strategy A such as uncertainty sampling, and where the second term weights the informativeness of x by its average similarity to all other instances in the input distribution (as approximated by the pool $\mathcal{U}$), for some similarity measure. The parameter β controls the relative importance of the density term.

What similarity measure to use may depend on the application. For bag of words representations in natural language processing for instance, cosine similarity is a common choice [46]. For image data, a simple straightforward measure is cross-entropy [53] between pixel values. This measure often fails to qualify semantic similarity however, therefore deep learning techniques could help to obtain more appropriate similarity measures that fit a particular data set. *Representation learning*, as described in Section 2.2, can help with providing better representations of the image data that support more suitable similarity measures.

The density can generally be precomputed for all data instances, as it will not change during the active learning process, thus incorporating such a density-weighted method in e.g. an uncertainty sampling approach essentially does not affect the time to select the next query.

In this thesis we will look into new methods to express the representativeness of data, as well as methods to prevent the problem of instances not being representative of their data distribution from occurring altogether.

2.1.3 Batch-Mode Active Learning

Traditionally, active learning methods involve querying instances one by one according to some query strategy, where the model is retrained after each query in order to evaluate the next most informative instance. For neural networks in particular however, retraining after each query is computationally expensive and therefore slow. This can be improved upon by querying in *batches*, labelling multiple instances at the same time before retraining the model. For batches of size k, a naive method is to simply take the "top k" most informative instances. B. Settles [45] describes why this approach doesn't work well in practice however; it fails to recognise the "overlap" within a batch, possibly introducing redundancy between instances in the batch. The best two queries

might for example be ranked high because they are virtually identical, meaning that labelling both is likely unnecessary. Thus, instances in the batch need to be both informative and *diverse* to prevent wasting labelling resources.

Settles [45] recognises a number of approaches to achieve diversity in batches for active learning. The first is explicitly incorporating a diversity measure. Xu et al. [54] for instance incorporate a diversity measure as well as a density measure for representativeness (as discussed in the previous section) together with a base informativeness measure, for relevance feedback formulated as a batch-mode active learning problem. The three measures, informativeness (relevancy), diversity, and density, are combined into one informativeness measure by means of a weighted sum. This is similar to the information density framework [46] described in the previous section, but instead of combining two measures by means of a product, here three measures are combined by means of a sum. Others, such as [9] and [25], use a similar approach, combining traditional active learning measures with a diversity measure for querying in batches. Greedy techniques are used to incrementally construct batches of informative queries with high diversity.

Another category views the task of finding diverse batches as a set optimisation problem, where the utility function for a batch is the expected joint reduction in uncertainty of the model using Bayesian design techniques [21][22].

Although these batch-mode active learning approaches generally seem to outperform passive (random) query selection, which in turn usually works better than selecting the "top k" most informative instances as a query batch, there are data sets for which random query selection can still outperform active methods [18]. This indicates that there are still improvements to be made in characterising the situations where batch-mode active learning is likely to help, and in pushing the state of the art. Moreover, even though these approaches often involve greedy methods to speed up the process, they still generally involve multiple passes over the full pool $\mathcal{U}$ of unlabelled data, which makes them a lot slower than simply querying a batch of the "top k" most informative instances.

2.2 Representation Learning

Although many active learning strategies leverage the existence of a large pool of unlabelled data, this pool is generally only used to model the underlying data distribution and to compute some informativeness heuristic over the instances in the pool. The actual machine learning is only done after instances have been selected for querying, and their labels have been obtained. There is however much to be learned from the unlabelled data alone, in particular for high-dimensional data such as images. Image data is generally represented by pixel values, which hide most of its semantics. We can use unsupervised learning methods to learn better representations of the unlabelled data, before we even consider an active learning scenario to obtain labels for supervised learning.

Representation learning is a domain within machine learning that aims at obtaining a (more) useful representation of data. In a way this can be interpreted as learning meaningful features of the data, as such it is also referred to as *feature learning*. The motivation behind such techniques is often to find good data representations for use with classification or regression tasks, in an automated and generalised manner. Deep neural networks for instance can in fact be viewed as *supervised* representation learning; learning a hierarchy of distributed representations (the hidden layers) that build from low level to high level representations towards the final supervised goal. It is also possible however to perform *unsupervised* representation learning, allowing models to work on (large) unlabelled data sets instead, which are often a lot easier to obtain.

Y. Bengio et al. [6] provide an extensive overview of representation learning background, motivations, applications, quality criteria, and techniques, up until the year 2013. Here we specifically focus however on a newer technique, which was first introduced in 2014, the *variational autoencoder* (VAE) [26][38]. As such, we will first look at the general concept of autoencoders in Section 2.2.1, before describing VAEs in Section 2.2.2.

In principle, any representation learning method could be used for our approach. We focus

however on VAEs because they link representation learning to *generative modelling*; they provide means to generate sensible data from scratch. First of all, it has been shown that this can lead to good representations of image data in the form of latent variables that form a densely populated region in the latent space, in which semantically similar data points are clustered together (see e.g. [15] and [43]). We conjecture that the density of the data distribution in this latent space can help to prevent the problem in active learning strategies where data is queried that is not representative to the underlying data distribution, as described in Section 2.1.2. As we will see in Section 2.2.2, a VAE tries to transform data to a latent representation that is close to some given prior distribution, thus it tries to force data to be representative of this distribution.

The latent space also provides more meaningful similarity measures, which allows for better diversity measures within a batch for batch-mode active learning as described in Section 2.1.3. A latent space that clusters semantically similar instances also supports more efficient active learning techniques.

Another advantage of the variational autoencoder is that the generative part provides interesting opportunities for active learning strategies that do not rely on the data set any more after a VAE is trained. Instead, we can use the generative model to evaluate the expected informativeness of any point in the latent space obtained through the VAE (see also Chapter 7).

2.2.1 Autoencoders

Autoencoders have been around for decades [55][8][20], and are traditionally used for dimensionality reduction or representation learning.

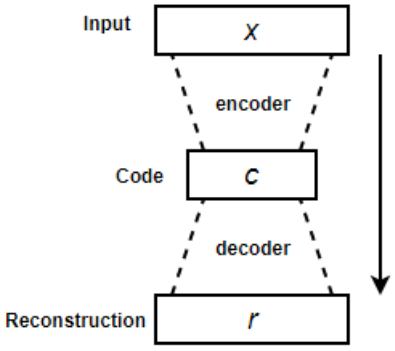


Figure 2.1: The general structure of an autoencoder

An autoencoder is an unsupervised (also referred to as self-supervised) neural network that is trained to attempt to copy its input to its output. As illustrated in Figure 2.1, it consists of an *encoder* and *decoder* part, each of which are neural networks themselves. The encoder takes a data instance x as input, and transforms it into some *code* c that is meant to describe the input. The decoder then takes this code c, and attempts to transform it back to the original input, resulting in a reconstruction r. Thus, the objective function used to train an autoencoder is based on the difference between the input x and the output r, according to some suitable difference measure. The general goal of an autoencoder is to learn useful properties of the input x, and encode them in the code c.

Although Figure 2.1 suggests that the code c must have lower dimensionality than the input x, this is in fact not required. However, autoencoders with codes of higher dimensionality than its inputs require some form of regularisation, generally by introducing an extra loss term into its objective function, to prevent it from simply learning the identity function to reconstruct its input.

In the simplest case, the encoder and decoder are simple fully connected feedforward networks, each of which can have zero or more hidden layers. The entire autoencoder can then also be seen as a fully connected feedforward network, and as such be trained using backpropagation and other

standard deep learning methods. But in general, all types of architectures can be used, and many variations exist.

2.2.2 Variational Autoencoder

Autoencoders were traditionally mainly used for dimensionality reduction and representation learning. More recently however, theoretical connections to latent variable models have resulted in the variational autoencoder (VAE) [26][38], which is mainly used as a generative model instead.

Problem Scenario

To understand the VAE well, we first need to define a clear problem scenario. We assume that the data set $\boldsymbol{X} = \{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(N)}\}$ consists of N i.i.d. samples from some continuous or discrete variable $\boldsymbol{x}$, and that the data is generated by some random process that involves a random variable $\boldsymbol{z}$, a latent variable. In this process, a latent value $\boldsymbol{z}^{(i)}$ is generated from some prior distribution $p_{\boldsymbol{\theta}^*}(\boldsymbol{z})$. Then, a value $\boldsymbol{x}^{(i)}$ is generated from some conditional distribution $p_{\boldsymbol{\theta}^*}(\boldsymbol{x}|\boldsymbol{z})$. We assume that this prior $p_{\boldsymbol{\theta}^*}(\boldsymbol{z})$ and likelihood $p_{\boldsymbol{\theta}^*}(\boldsymbol{x}|\boldsymbol{z})$ come from parametric families of distributions $p_{\boldsymbol{\theta}}(\boldsymbol{z})$ and $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$, respectively, and that their probability density functions are differentiable almost everywhere w.r.t. $\boldsymbol{\theta}$ and $\boldsymbol{z}$. The true parameters $\boldsymbol{\theta}^*$ as well as the values of the latent variables $\boldsymbol{z}^{(i)}$ are unknown to us.

To describe the original data distribution and its relation to the latent variables, we are interested in the marginal likelihood $p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{z}) p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\, d\boldsymbol{z}$, as well as the true posterior density $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) = p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) p_{\boldsymbol{\theta}}(\boldsymbol{z}) / p_{\boldsymbol{\theta}}(\boldsymbol{x})$. This integral and posterior are intractable however for a likelihood function $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ expressed by a neural network with a nonlinear hidden layer. Thus, we are interested in efficient approximations for these distributions.

We now introduce a recognition model $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$, which is an approximation of the intractable true posterior $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$. We then need a method to learn the recognition model parameters $\boldsymbol{\phi}$ together with the generative model parameters $\boldsymbol{\theta}$.

Autoencoder Architecture

In the context of an autoencoder, we can view the unobserved latent variables z as the *code* of the autoencoder. The recognition model $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ can then be seen as the probabilistic *encoder*; given a data point $\boldsymbol{x}$ it produces a distribution over the possible values of the code $\boldsymbol{z}$ from which $\boldsymbol{x}$ could have been generated. Similarly, $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ can be seen as a probabilistic *decoder*; given a code $\boldsymbol{z}$ it produces a distribution over the possible corresponding values of $\boldsymbol{x}$. Note that the encoder in this case doesn't actually output a code value z, but rather a distribution for such values. In the flow of the autoencoder we can sample a value z from this distribution however, which we can then input into the decoder. The resulting variational autoencoder is illustrated in Figure 2.2.

The encoder and decoder network can generally be any type of neural network, but a common choice are multilayer perceptrons (MLPs), also referred to as fully connected feedforward networks or networks with dense layers. Particularly for image data, it is also common to use convolutional [31] and transposed convolutional (also called deconvolutional) layers for the encoder and decoder, respectively. Such a VAE we refer to as a convolutional variational autoencoder.

Variational Lower Bound

D. P. Kingma & M. Welling [26] describe how to obtain a suitable objective function to train the variational autoencoder. We want to maximise the marginal likelihood $\log p_{\boldsymbol{\theta}}(\boldsymbol{X})$ of the data under the model, which consists of the sum over the marginal likelihoods of individual data points,

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(N)}) = \sum_{i=1}^{N} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}).$$

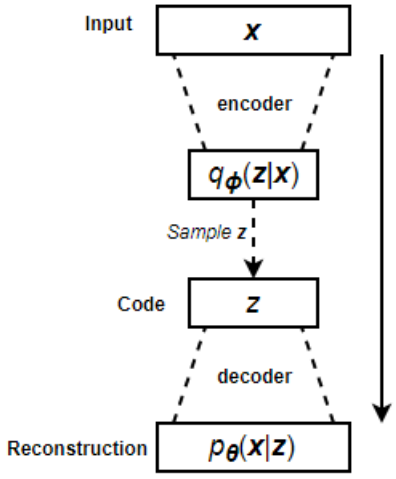


Figure 2.2: The general structure of a variational autoencoder

Each of the terms in this sum can be rewritten as

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) = D_{KL}\Big(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})||p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})\Big) + \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}^{(i)}).$$

The first term on the right-hand side is the KullbackLeibler (KL) Divergence [28] of the approximate posterior from the true posterior. This term is non-negative, therefore the second term is called the *variational lower bound* on the marginal likelihood of data point $\boldsymbol{x}^{(i)}$. It can be written as

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}^{(i)}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\Big[-\log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) + \log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})\Big] \tag{2.1}$$

$$= -D_{KL}\Big(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})||p_{\boldsymbol{\theta}}(\boldsymbol{z})\Big) + \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})}\Big[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z})\Big]. \tag{2.2}$$

We want to differentiate and maximise this lower bound w.r.t. both the variational parameters $\boldsymbol{\phi}$ and the generative parameters $\boldsymbol{\theta}$, as a tractable approximation of the total marginal likelihood of the data under the model. Note that when phrasing this as a loss function, we wish to minimise the negative lower bound $-\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}^{(i)})$.

Loss Function

If we first look at the *second* term of Equation 2.2, we see that it is in fact the probability density of generated output given the inferred latent distribution over z, i.e. the (negative) expected reconstruction error. The expected reconstruction error is a loss term used for training (almost) every autoencoder, it describes how accurately the output replicates the input. A common metric for this is binary cross-entropy, which we will use in this thesis, although this has a tendency of producing blurry images [24][51] in generative models (such as the decoder part of the VAE). More advanced similarity metrics have been proposed to target this problem, such as a metric that uses learned representations obtained with a generative adversarial network (GAN) [29].

The *first* term of Equation 2.2 is the KL Divergence of the approximate posterior $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})$ from the prior $p_{\boldsymbol{\theta}}(\boldsymbol{z})$. This can be seen as a measure of "surprise"; the extent to which the model must update its parameters to accommodate new observations. This KL Divergence term in the loss function encourages robustness to small perturbations along the latent manifold by inducing the learned approximation $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})$ (the encoder) to match some continuous chosen prior $p_{\boldsymbol{\theta}}(\boldsymbol{z})$.

By choosing a conjugate prior over z such that we can analytically integrate the KL Divergence, we can obtain a closed-form equation for this loss term. A common choice is the spherical Gaussian (or standard multivariate normal distribution), $z \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. As an approximate posterior we then

use $\boldsymbol{z}|\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, diag(\boldsymbol{\sigma}))$, for some parameters $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_k)$ and $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_k)$ (with latent dimension k). For this prior and posterior, we find the closed form

$$-D_{KL}\Big(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})||p_{\boldsymbol{\theta}}(\boldsymbol{z})\Big) = \frac{1}{2}\sum_{i=1}^{k}(1 + \log\sigma_i^2 - \mu_i^2 - \sigma_i^2).$$

Combining these two loss terms, we obtain a loss function that encourages a trade-off between expressiveness and conciseness, pushing the model towards reconstructing its input well yet also learning a simple latent space representation based on the chosen prior.

Training the Variational Autoencoder

If we impose a standard multivariate Gaussian distribution $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$ as the prior $p_{\boldsymbol{\theta}}(\boldsymbol{z})$, and approximate the posterior $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})$ with a multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, diag(\boldsymbol{\sigma}))$ with parameters $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_k)$ and $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_k)$, we can finally fully define the architecture of a variational autoencoder, together with an associated loss function. This architecture is illustrated in Figure 2.3.

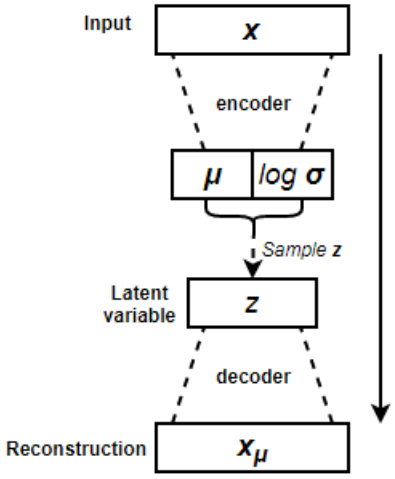


Figure 2.3: A variational autoencoder with multivariate Guassian prior and posterior

The VAE takes a data point $\boldsymbol{x}$ as input, and transforms it through a neural network into parameters $\boldsymbol{\mu}$ and $\log\boldsymbol{\sigma}$ that describe the (approximate) posterior $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})$ (thus in this case, $\boldsymbol{\phi} = (\boldsymbol{\mu}, \boldsymbol{\sigma})$). For convenience and numerical precision, we learn the logarithm of the parameter $\boldsymbol{\sigma}$ instead of $\boldsymbol{\sigma}$ itself. Then, a latent value $\boldsymbol{z}$ is sampled from the distribution $\mathcal{N}(\boldsymbol{\mu}, diag(\boldsymbol{\sigma}))$. Finally, this $\boldsymbol{z}$ is transformed through a decoder network (often with the inverse architecture of the encoder) into a reconstruction $\boldsymbol{x}_{\boldsymbol{\mu}}$ of the input $\boldsymbol{x}$. This reconstruction represents the mean value of the distribution $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$, which we use to evaluate the reconstruction error. We do not need the covariance diagonal for this distribution, thus we do not learn it.

Parametrisation Trick

We have shown before that we can define a closed-form loss function under the chosen prior and posterior, using binary cross-entropy as the reconstruction loss term. However, in order to train the network by means of backpropagation with (stochastic) gradient descent, we need our model to be differentiable w.r.t. its learned parameters. For this, the model needs to be deterministic, such that only the inputs are stochastic. This is clearly not the case if we sample $\boldsymbol{z}$ based on a probability distribution with parameters that come from inside the model.

We can solve this problem with a simple *parametrisation trick*. We define auxiliary random variables $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, which we treat as input. We then simulate sampling $\boldsymbol{z}$ from $\mathcal{N}(\boldsymbol{\mu}, diag(\boldsymbol{\sigma}))$ by computing $\boldsymbol{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$ (where $\odot$ signifies the element-wise product).

We thus have defined a deterministic model, with input $\boldsymbol{x}$ and $\boldsymbol{\epsilon}$, with a loss function that can be automatically differentiated by tools such as Keras [12] or Tensorflow [1]. As such, we can train this model using backpropagation with stochastic gradient descent, and related techniques and optimisers.

2.3 Related Methods

In this thesis we use a variational autoencoder to obtain a latent space representation of (image) data, which we use to perform active learning. To the best of our knowledge, this is the first work that uses a VAE for active learning purposes.

M. Ducoffe et al. [16] do propose an active learning strategy for (convolutional) neural networks that, like variational autoencoders, is based on variational inference. This approach however involves variational inference only for the purpose of obtaining an informativeness measure for unlabelled data. Our approach instead uses it to learn a representation for the entire data set, and has the advantage that improvements in the field of VAEs may directly lead to better results for our active learning strategy. Moreover, training a VAE on all unlabelled data first prevents us from having to retrain low level features in every active learning iteration, which allows us to use simpler and therefore faster models for active learning.

S. Berardo et al. [7] also use representation learning techniques for active learning. They stack denoising autoencoders [52] as a means for extracting meaningful features of (image) data. They then use a clustering algorithm to find clusters in the feature space. From these clusters they query representatives, and label all points (or the most centric points) in the clusters with the same label. They then use the labels obtained to train a neural network classifier. This neural network is however still trained on the original pixel representations of the data; the clustering technique is only used to obtain labels. In this thesis we will explore how the representations learned by VAEs may allow us to directly use the learned latent space for training a neural network classifier.

Other methods exist that use pre-training methods that have been proposed for deep learning, which could be interpreted as representation learning. S. Zhou et al. [56] e.g. use Restricted Boltzman Machines (RBMs) to pre-train a deep neural network on unlabelled data, before fine-tuning it with the (limitedly available) supervised data. They then query new labels based on this model. Similarly, P. Liu et al. [34] train a Deep Belief Network, which also uses (unsupervised) RBM pre-training before fine-tuning the network in a supervised manner and querying more labels based on uncertainty and data representativeness measures. Such methods however do not use the full power of current representation learning technology; techniques such as the variational autoencoder are capable of finding much better data representations, and can be tailored to work well with the type of data (e.g. convolutional autoencoders for image data).

Chapter 3

Approach

In the previous chapters we described the motivation behind our approach, as well as the relevant background that is necessary to understand the key elements. In this chapter we first provide a general overview of our approach, after which we go into more detail about the various elements of our method and the possibilities, variations, and considerations.

3.1 General Structure

Figure 3.1 illustrates the general structure of the active learning strategy we propose. The procedure can be summarised in a number of steps. We first execute the following steps once:

(a) We train a variational autoencoder (VAE, see Section 2.2.2) on the unlabelled data pool $\mathcal{U}$.

(b) After the VAE is trained, we feed the unlabelled data $\mathcal{U}$ to the decoder to obtain a latent space representation for all the data, i.e. we transform each data instance from $\mathcal{U}$ into the mean value $\boldsymbol{\mu}$ of its encoding. We thus obtain a latent space representation $\mathcal{U}_{\boldsymbol{z}}$ of our unlabelled data.

(c) We first randomly select a batch (of a given batch size) of unlabelled data, which we query to the *oracle* to be labelled. This selection of data can be fully random, or partly based on some strategy that supports diversity within the batch (see Section 3.4.3). Note that although our data is in latent representation, we also still have access to the original (image) data. Therefore we link each latent representation to its original image, such that an oracle can provide a label based on the original image. In this way, a reliable label can be obtained even if the decoder part of the autoencoder would not provide a good reconstruction. We thus obtain a first small set $\mathcal{L}_{\boldsymbol{z}}$ of labelled data.

Then, we repeatedly perform the following steps for a fixed number of iterations (which we call the *active learning iterations*):

(d) We train a model on the current labelled latent data $\mathcal{L}_{\boldsymbol{z}}$. Here we use a multilayer perceptron (MLP) that consists of one or more fully connected layers, followed by a softmax with the desired number of output classes. In principle, any (probabilistic) supervised machine learning algorithm could be used, but we chose an MLP because it is relatively inexpensive to train, yet still powerful enough to learn well if a good latent representation is already acquired through the VAE.

(e) Once the MLP has been trained, we use it to make predictions for the remaining unlabelled data $\mathcal{U}_{\boldsymbol{z}}$. Since the MLP ends in a softmax, this gives us probabilities for all possible labels for each data instance in $\mathcal{U}_{\boldsymbol{z}}$.

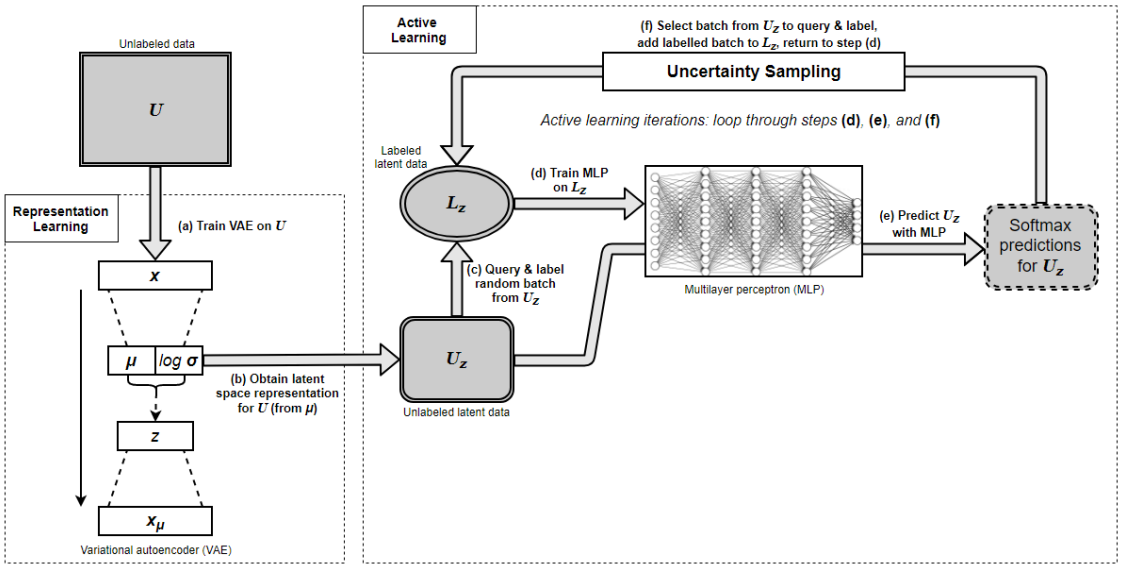


Figure 3.1: The general structure of the active learning strategy

(f) Using these probabilities, we can compute the uncertainty (according to some uncertainty measure, see Section 2.1.2) of the predictions for each data instance. We then use these uncertainties, possibly in combination with some representativeness and/or diversity measure (see Sections 3.4.1 and 3.4.3), to obtain a batch of instances to query to the oracle for labelling. We add the newly labelled data to our set of labelled data $\mathcal{L}_z$.

After a number of active learning iterations, when we have obtained the required number of labels, we can train the MLP one last time with $\mathcal{L}_z$ to obtain our final classification model. In order to then classify new data, we first need to use the VAE's encoder to obtain a latent space representation of this data, after which we can use the MLP to make predictions for the possible labels.

3.2 Variational Autoencoder

In Section 2.2.1 we provided details about how to construct and train a variational autoencoder, in particular when assuming a standard multivariate Gaussian distribution as prior, and a multivariate Gaussian with diagonal covariance matrix as posterior. This still leaves a number of choices to be made before we have fully defined a VAE.

The first thing to consider is the architecture of the encoder and decoder. Here we consider two types; VAEs with only fully connected layers, and VAEs with convolutional layers as well.

Fully connected layers are also referred to as *dense* layers, therefore we will call a VAE with only fully connected layers a *dense VAE*. In a dense VAE, the encoder consists of one or more dense hidden layers connecting the input layer to the $(\boldsymbol{\mu}, \log \boldsymbol{\sigma})$ layer (see Figure 2.3). The decoder then consists of one or more dense hidden layers connecting the sampled $\boldsymbol{z}$-value to the output layer. We only consider symmetric VAEs here, meaning that dimensions of the hidden layers of the decoder are the same as those of the encoder, but in reversed order. For the activations in all the hidden units we will use rectified linear units (ReLUs) [19]. For the output units we will use the sigmoid function, in order to retrieve greyscale pixel values between 0 and 1. All that remains then to fully define the dense VAE is to choose the dimensions of the hidden layers of the encoder.

Since we are dealing with image data, it makes sense to also incorporate convolutional layers [31] in our VAE, since models with convolutional layers have been shown to perform much better on images than models with only dense layers. We refer to such a VAE as a *convolutional VAE*. The

encoder will consist of one or more convolutional layers, followed by one or more dense layers, which together connect the input layer to the $(\boldsymbol{\mu}, \log \boldsymbol{\sigma})$ layer. The decoder then consists of one or more dense layers, followed by one or more transposed convolutional (or deconvolutional) layers, connecting the sampled $\boldsymbol{z}$-value to the output layer. We again only consider symmetrical architectures for the dense hidden layers. For the architectures of the (de)convolutional layers, things get a bit more complicated to ensure we get the right sizes to encode and reconstruct the input. For each deconvolutional layer, we need to specify the number of filters (the depth of the output), a kernel size (the size of the filter), the stride (the step size with which to slide the filter), and the type of padding (either "valid" or "same"). As activations for all the hidden units we again use ReLUs, whereas for the output units we use the sigmoid function to obtain greyscale pixel values between 0 and 1.

3.3 Multilayer Perceptron

Once we have obtained a latent space representation of the unlabelled data through a VAE, we randomly take a batch of instances from this latent data to train a first classification model. Here we use a multilayer perceptron (MLP) with one or more fully connected hidden layers, with dropout [50] to prevent overfitting. The final layer of the MLP is a (fully connected) softmax with the desired number of output classes. Generally we do not posses validation data in an active learning setting, since labels are expensive and therefore too valuable not to use for training, thus we cannot use convergence of the validation accuracy to determine how long to train the MLP. Therefore we train the MLP for a fixed number of epochs, paying attention that the training accuracy does not converge long before we finish training. The dropout in the hidden layers helps to prevent overfitting, thus a fixed amount of epochs (if not too few) can be expected to work reasonably well.

After training, the MLP can give us predictions for the likelihood of each of the possible classes for every unlabelled data instance. We can use these predictions to compute uncertainty, which we then use in an active learning algorithm to obtain more labelled data. More details of this active learning procedure are described in the next section.

In each next active learning iteration, i.e. when we have obtained more labels, we reinitialise the weights of the MLP and train it again for a fixed number of epochs. We retrain the model from scratch to prevent overfitting on labelled data that has been available for more iterations already. In other words, we want to ensure that newly labelled data has the same influence as "older" data. Since MLPs are quite fast to train (compared to e.g. convolutional networks), our retraining procedure remains very fast even if the learned model is "forgotten" after each active learning iteration. If we can combine this with a fast active learning query selection method, then the entire procedure can be quite fast and fluent; annotators will hardly experience any downtime while providing labels for the active learning procedure. Nevertheless we can utilise the power and expressiveness of convolutional networks for image data, by having used a convolutional VAE to produce the latent data.

3.4 Active Learning

3.4.1 Uncertainty Sampling

For our active learning strategy we use uncertainty sampling, as previously described in Section 2.1.2. We consider all three different uncertainty measures; *least confident*, *margin sampling* and *entropy*. For reasons explained in the next section, we want to ensure that these uncertainty measures are values between 0 and 1, where 0 implies no uncertainty (thus low informativeness), whereas 1 means high uncertainty.

The least confident and margin sampling strategies already guarantee that uncertainty values are between 0 and 1. All we need to do is reverse them by subtracting the value from 1, since low confidence or small margins imply high uncertainty. The entropy uncertainty value does not

need to be reversed, since high entropy corresponds to high uncertainty. But the entropy of a prediction can in general be greater than 1. The entropy can however not exceed $\log n$, where n is the number of classes (this value is obtained when all classes are equally likely). Thus, we can normalise entropy uncertainty by dividing by $\log n$.

3.4.2 Representativeness

We are particularly interested in whether we can make the standard uncertainty sampling strategy more efficient by utilising the latent space obtained through the VAE. As we have mentioned before, a common problem in uncertainty sampling is that it is prone to querying outliers; instances that do not represent the underlying data distribution well. Knowing the label of such instances generally does not provide much useful information to the learner. To prevent querying outliers, we want to formulate some representativeness measure, and prevent querying instances that score low on this measure. For practical reasons we want to ensure this measure is a value between 0 and 1, where 0 means bad representativeness (outliers) and 1 means good representative data.

There are several ways to express the representativeness of a data instance, based on the full data distribution. In the density-weighted framework as described in Section 2.1.2, representativeness is defined as the average similarity of a data instance x to all other instances in the data pool $\mathcal{U}$,

$$\text{repr}(\boldsymbol{x}) = \frac{1}{|\mathcal{U}|} \sum_{\boldsymbol{x}_u \in \mathcal{U}} \text{sim}(\boldsymbol{x}, \boldsymbol{x}_u),$$

according to some similarity measure. We consider a number of similarity measures for this purpose.

Image Data

We consider greyscale image data, stored by their pixel intensities. Suppose we have two images $\boldsymbol{x}$ and $\boldsymbol{y}$ of p pixels each, with pixel intensities $(x_1, \ldots, x_p)$ and $(y_1, \ldots, y_p)$ (with $0 \leq x_i, y_i \leq 1$). A possible similarity measure is the *cosine similarity*, which is then defined as

$$\text{sim}_{cos}(\boldsymbol{x}, \boldsymbol{y}) = \frac{\sum_{i=1}^{p} x_i \cdot y_i}{\sqrt{\sum_{i=1}^{p} x_i^2 \cdot \sum_{i=1}^{p} y_i^2}} = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{\|\boldsymbol{x}\|_2 \cdot \|\boldsymbol{y}\|_2}.$$

The similarity between two images is then guaranteed to be a value between 0 and 1, since all pixel intensities are non-negative.

We can also use a distance measure to obtain a density measure between 0 and 1. A given distance measure $d(\boldsymbol{x}, \boldsymbol{y})$ can be used to express representativeness as

$$\text{repr}(\boldsymbol{x}) = 1 - \frac{1}{d_{max}} \frac{1}{|\mathcal{U}|} \sum_{\boldsymbol{x}_u \in \mathcal{U}} d(\boldsymbol{x}, \boldsymbol{x}_u),$$

where d_{max} is either some upper bound of the distance measure (if such a bound exists), or the maximum observed average distance for the given data.

One of the most common distance metrics is the *Euclidean distance*, which is defined in this context as

$$d_E(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{\sum_{i=1}^{p} (x_i - y_i)^2}.$$

Since we know that pixel intensities are values between 0 and 1, we can obtain a *normalised Euclidean distance* as follows:

$$d_{NE}(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{\frac{1}{p} \sum_{i=1}^{p} (x_i - y_i)^2}.$$

This measure always yields a measure between 0 and 1 (for pixel intensities between 0 and 1), and can thus easily be transformed into a similarity measure by taking $\text{sim}_{NE}(\boldsymbol{x}, \boldsymbol{y}) = 1 - d_{NE}(\boldsymbol{x}, \boldsymbol{y})$.

For our purposes we can also consider the *squared Euclidean distance*, which is given by

$$d_{SE}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{p} (x_i - y_i)^2.$$

This measure yields puts a progressively larger weight on bigger differences between pixel values. Note that this is not a true metric, since it doesn't satisfy the triangle equality. We can however still use this measure for our purpose of expressing representativeness. We can also normalise this measure by considering the *mean squared Euclidean distance* instead,

$$d_{MSE}(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{p} \sum_{i=1}^{p} (x_i - y_i)^2,$$

which yields values between 0 and 1 (for pixel intensities between 0 and 1). Thus, it can be transformed into a similarity measure in the same way as for the normalised Euclidean distance.

Latent Data

In the active learning scheme of this thesis we do not work with images directly however. Instead we transform images (through a VAE) into a latent space representation, or more precisely into Gaussian distributions $\mathcal{N}(\boldsymbol{\mu}, diag(\boldsymbol{\sigma}))$ in this latent space, characterised by parameters $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$. Given two such distributions $\boldsymbol{z_x} \sim \mathcal{N}(\boldsymbol{\mu_x}, diag(\boldsymbol{\sigma_x}))$ and $\boldsymbol{z_y} \sim \mathcal{N}(\boldsymbol{\mu_y}, diag(\boldsymbol{\sigma_y}))$, we can consider some other similarity and distance measures. Let d be the number of latent space dimensions.

The aforementioned *cosine similarity* also yields a valid similarity measure for latent data, if used on the mean values of the distributions:

$$\text{sim}_{cos}(\boldsymbol{z_x}, \boldsymbol{z_y}) = \frac{\boldsymbol{\mu_x} \cdot \boldsymbol{\mu_y}}{\|\boldsymbol{\mu_x}\|_2 \cdot \|\boldsymbol{\mu_y}\|_2}.$$

Since the $\boldsymbol{\mu}$-values in latent space can also be negative, this yields values between -1 and 1. We can easily obtain values between 0 and 1 by adding 1 and then dividing by 2.

For the $\boldsymbol{\mu}$-values we can also use the *Euclidean distance* and the *squared Euclidean distance* as before, however we cannot normalise these measures in the same way as before, since generally the components of $\boldsymbol{\mu}$ are not between 0 and 1. For a given data set however, we can normalise average distances by dividing by the largest observed average distance.

Since the latent data represents distributions rather than single images, there are some other plausible distance measures that can be considered. First of all there is the KL Divergence [28], which we saw before in Section 2.2.2. For two multivariate Gaussian distributions with diagonal covariance matrices $\mathcal{N}(\boldsymbol{\mu_x}, diag(\boldsymbol{\sigma_x}))$ and $\boldsymbol{z_y} \sim \mathcal{N}(\boldsymbol{\mu_y}, diag(\boldsymbol{\sigma_y}))$, the KL Divergence can be computed as

$$D_{KL}(\boldsymbol{z_x} || \boldsymbol{z_y}) = -\frac{1}{2} \sum_{i=1}^{d} \Big(1 - \frac{(\mu_{x,i} - \mu_{y,i})^2 + \sigma_{x,i}^2}{\sigma_{y,i}^2} + \log \sigma_{x,i}^2 - \log \sigma_{y,i}^2\Big).$$

This measure is not symmetric, i.e. in general $D_{KL}(\boldsymbol{z_x} || \boldsymbol{z_y}) \neq D_{KL}(\boldsymbol{z_y} || \boldsymbol{z_x})$, which is a quality that may be sensible if we want to use the measure to compute the average distance of one data point to all the others. To obtain a similar yet symmetric measure, we can use the J Divergence [33], which is simply the sum of the KL Divergences both ways:

$$D_J(\boldsymbol{z_x} || \boldsymbol{z_y}) = D_{KL}(\boldsymbol{z_x} || \boldsymbol{z_y}) + D_{KL}(\boldsymbol{z_y} || \boldsymbol{z_x}).$$

In the case of multivariate Gaussian distributions with diagonal covariance matrices, we find

$$D_J(\boldsymbol{z_x} || \boldsymbol{z_y}) = -\frac{1}{2} \sum_{i=1}^{d} \Big(2 - \frac{(\mu_{x,i} - \mu_{y,i})^2 + \sigma_{x,i}^2}{\sigma_{y,i}^2} - \frac{(\mu_{y,i} - \mu_{x,i})^2 + \sigma_{y,i}^2}{\sigma_{x,i}^2}\Big).$$

Other measures to quantify the difference between two probability distributions exist (see e.g. [33]), but those are out of the scope of this thesis. Note that none of these measures guarantee to yield values of at most 1, so to normalise we can again divide by the highest observed average distance (divergence) in the data.

Computing the representativeness of a data instance as the average similarity is rather slow, since we must evaluate some distance or similarity measure for all pairs of instances in the data set. We know from the training procedure of the VAE that part of the loss function tried to produce distributions that remain close to the chosen prior $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$, a standard multivariate Gaussian distribution, by means of the KL Divergence. Therefore, we can also consider the KL Divergence of a latent distribution to this prior as a way to express representativeness (where low divergence means high representativeness). We can normalise this by dividing by the highest observed KL Divergence to the prior in the data.

Similarly, we can also consider the Euclidean distance and squared Euclidean distance of a distribution's mean $\boldsymbol{\mu}$ to the prior's mean $\mathbf{0}$ as a measure of representativeness. Here we need to normalise this (squared) distance by the highest observed (squared) distance and subtract it from one, to obtain a representativeness measure.

Explicitly Incorporating the Representativeness Measures in Uncertainty Sampling

We can use the various representativeness measures mentioned above to evaluate whether data can be considered "more representative" when transformed to latent space data. In other words: are instances that were considered outliers in the original space mapped to dense areas in the latent space? If this is the case, we can expect uncertainty sampling to function better in latent space than in the original space.

We can also explicitly use these representativeness measures in combination with an uncertainty measure to compute an informativeness measure for active learning as follows:

$$\text{informativeness}(\boldsymbol{x}) = \gamma \cdot \text{uncertainty}(\boldsymbol{x}) + (1 - \gamma) \cdot \text{representativeness}(\boldsymbol{x}),$$

where $0 \leq \gamma \leq 1$ is a constant that determines the relative importance of the uncertainty term over the representativeness term. Setting $\gamma = 1$ is equivalent to using only standard uncertainty sampling. This informativeness computation is the reason that we required uncertainty and representativeness measures to yield values between 0 and 1; in this way both measures are on a similar scale, and we can combine them. This proposed informativeness measure is then also always between 0 and 1. This approach is similar to the density-weighted framework proposed in Section 2.1.2, but instead of a multiplication here we use a weighted sum to combine uncertainty with representativeness, as is done in e.g. [54].

3.4.3 Batch-Mode Active Learning

The informativeness measures described in the previous section provide us with a strategy to select a single instance to query next. As discussed in Section 2.1.3 however, it is desirable to be able to query a full batch of k informative instances at once before retraining the classification model, in such a way that we prevent redundancy among the instances in this batch. We consider a number of strategies to query a batch of informative instances at once.

Top k

The simplest strategy is the *top k* strategy; we rank all unlabelled instances based on the informativeness measure, and select the k instances that score highest. This method generally introduces high redundancy in the batch, as uncertain instances often group together, thus it is not very suitable for this task.

Random Partitions

A simple way to prevent some of this redundancy is a method we will call *random partitions*; we randomly partition the unlabelled pool $\mathcal{U}$ in k equally sized parts, and query the instance with the highest informativeness value from each partition. For every active learning iteration, we compute new random partitions. This helps to prevent some redundancy, but if e.g. a large cluster of many more than k very uncertain instances exists, it is still likely that we obtain many redundant instances.

Greedy Method

We can also try to find a batch in which the distance between the instances is large. For this we can use the distance measures described in the previous section for computing representativeness of latent data. W. Xu et al. [54] e.g. describe a *greedy* method to find a query batch for active learning that attempts to optimise for informativeness, representativeness, and diversity at the same time. We can use a similar technique in our setting. We start by initialising the batch $B = \{\boldsymbol{x}_0\}$ to contain only the instance $\boldsymbol{x}_0$ with the highest combined uncertainty and representativeness score (computed as a weighted sum, see previous section). The idea is then to construct a batch by incrementally adding the next instance $\boldsymbol{x}$ to B that maximises the following weighted sum:

$$\alpha \cdot \text{uncertainty}(\boldsymbol{x}) + \beta \cdot \text{representativeness}(\boldsymbol{x}) + (1 - \alpha - \beta) \cdot d(\boldsymbol{x}, B).$$

Here $d(\boldsymbol{x}, B) = \min_{\boldsymbol{y} \in B} d(\boldsymbol{x}, \boldsymbol{y})$ is the distance of x to the nearest element already in the batch B, according to some distance measure d. The parameters $0 \leq \alpha, \beta \leq 1$ determine the relative importance of each of the three terms, where we require that $\alpha + \beta \leq 0$. Since we have already ensured that the uncertainty and representativeness values are between 0 and 1, we would like to do the same for the distance $d(\boldsymbol{x}, B)$, such that all three measures are more or less on the same scale.

For the distance $d(\boldsymbol{x}, \boldsymbol{y})$ we can use the same measures as discussed in the previous section, such as (squared) Euclidean distance, J Divergence, or the reversed cosine similarity. The advantage of cosine similarity is that it is always between -1 and 1, making it easy to obtain a distance measure between 0 and 1. For the other measures, we can find the maximum observed distance in the data, and divide each distance by that maximum. Finding this maximum is very slow however (quadratic time in the number of instances, as it requires evaluating each pair of instances), so even though this only has to be done once for each latent data set, we use a significantly faster method to approximate this value. The VAE transforms data in such a way that it roughly forms a sphere around the origin in latent space (see also Section 5.3.3). Thus, we instead find the data instance $\boldsymbol{x}$ with the highest distance to the origin (in latent space), multiply its distance to the origin by two, and use that as an approximation for the maximum distance. It follows directly from the triangle inequality on $\boldsymbol{x}$, the origin, and any other data point that this approximation provides an upper bound for the highest observed distance, in case our distance is a true metric. Some of our distance measures do not satisfy the triangle inequality however, but this simple approach can nevertheless serve as a fast heuristic to normalise the distance measure. Since this method only needs to check each data instance once (for its distance to the origin) it runs in linear rather than quadratic time.

The greedy approach gives more guarantees that a batch will not contain instances that are very similar to each other. A downside to this approach however is that it is quite computationally expensive. For every new instance that we add during the greedy incremental batch construction, we need to compute the minimum distance for all remaining unlabelled instances to any of the instances already in the batch. A way to speed up this process is to perform *subsampling*; we randomly select only a certain percentage of the data, and apply the greedy batch construction algorithm on this subsample only. For every active learning iteration, we compute a different subsample.

Subsampling

Subsampling can also be used as a method to improve the diversity within a query batch. We can take a small subsample of the data only, so that it likely contains fewer similar instances, and then use either the *top k* strategy or the *random partitions* strategy to select a batch of informative instances from this subsample. Although such methods are more heuristic in flavour and provide less guarantees, they are significantly faster than the greedy approach described above, and often still work well in practice.

Chapter 4

Experimental Setup

In this chapter we provide details about the experiments we ran to evaluate the approach described in Chapter 3. The results of these experiments will be discussed in Chapter 5. All coding was done in `Python`, using the `Keras` [12] library with `TensorFlow` [1] back-end to implement neural networks.

4.1 Data

To evaluate our approach, we used two benchmark image data sets, each containing small square images of single digits, with labels 0 to 9. Each data set consists of a training and test set, both fully labelled. From the training sets we take a small fixed portion apart as validation data. We only use the training set (without the validation data) for training VAEs and in the active learning procedure. The validation data is used to provide better indications about how well a model (VAE or MLP) is learning, and the test data is only used to evaluate the final performance of a classification model (an MLP) after all active learning iterations are done.

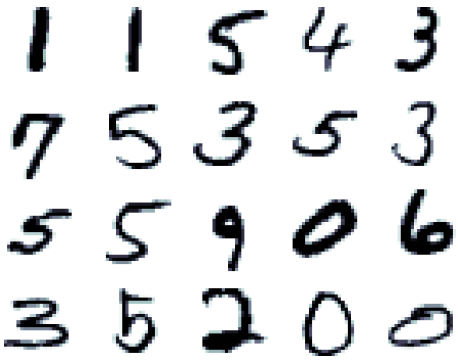


Figure 4.1: Examples of MNIST digits

Although active learning deals with situations where labels are scarce, in order to evaluate an active learning scheme well we need labelled data to evaluate performance and to be able to quickly simulate the querying for labels. When testing an active learning procedure, we hide all labels at first, pretending that all data is unlabelled. Only when we decide to query a particular instance of the training set will we reveal and use its label. The validation accuracies that we obtain while training MLPs are only for evaluation purposes, and may not be used for decisions during the active learning procedure, since in a normal setting labelled validation data would not be available.

4.1.1 MNIST

The first data set that we use is the MNIST [30] data set of handwritten digits. The data consists of greyscale images of 28 by 28 pixels, each pixel given as a value between 0 (white) and 1 (black). We obtain the data through the `TensorFlow` [1] library, which automatically splits it into a training set of 55,000 images, a validation set of 5,000 images, and a test set of 10,000 images. Figure 4.1 shows an example of some images from MNIST.

MNIST is one of the most standard benchmark data sets for machine learning, and is generally known to be quite easy to work with. As we will see in Chapter 5, it is easy to train a simple model that already has a test set accuracy of over 98%.

4.1.2 SVHN

The next data set we consider is the SVHN [36] (Street View House Numbers) data set with cropped digits. Just like MNIST it contains digits from 0 to 9, but instead of handwritten greyscale images these are colour images obtained from pictures of house numbers in Google Street View. We use the version of the data set in which images of full house numbers (as shown in Figure 4.2a) have been cropped to the bounding box of a single digit (as in Figure 4.2b). This bounding box is first extended to a square image, which results in some distracting digits to the sides of the digit of interest. All cropped images are resized to 32 by 32 pixels.

(a) The original images with bounding boxes (b) Cropped square images

Figure 4.2: Examples of the SVHN data set

The SVHN data set consists of a training set of 73,257 instances and a test set of 26,032. We use the first 5,000 instances of the training set as validation data, and the remaining instances as training data.

The images are given as RGB values for each pixel. To simplify the data and have it in a similar format as MNIST, we transform the RGB values (where R, G, and B are integers from 0 to 255) to greyscale values between 0 and 1 with the following formula [37]:

$$\text{Greyscale} = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B.$$

Since colour is usually not a relevant factor in digit recognition, this eliminates some unnecessary complexity, while it still generally yields recognisable images. See Figure 4.3 for an example of how RGB digits look when transformed to greyscale.

Because of the distracting digits that can occur on the sides of the images, other distracting shapes in the background, and the fact that digits have different colours and intensities, classification tasks for this data set are notoriously harder than for a data set like MNIST where digits are nicely black on white and with no distracting shapes in the image.

(a) Digits in colour (RGB)

(b) Digits in greyscale

Figure 4.3: Transforming coloured SVHN digits to greyscale

4.2 Neural Networks

4.2.1 Variational Autoencoder

In Section 3.2 we discussed the types of VAEs that we use (dense and convolutional), and which hyperparameters we need to choose to determine their full architectures. Our VAE implementations are based on the `Keras` [12] example implementations of a normal VAE [11] and a convolutional VAE [10]. For each of the VAE types, we use a loss function based on variational inference as described in Section 2.2.2 to train the model by means of stochastic gradient descent, with the "rmsprop" optimizer as implemented in `Keras`. The batch size for training is 100.

For the dense VAE we used an encoder with two hidden layers of 512 and 256 hidden units respectively, and a decoder with the same architecture but reversed. We used three different latent space dimensions: 2, 30, and 50. The input dimension naturally depends on the input, and all other aspects of the architecture have been explained in Sections 2.2.2 and 3.2. Thus, for each data set we used three different dense VAEs.

The architecture of the convolutional VAE is shown in Table 4.1. For the *parametrisation trick* (see Section 2.2.2) we need to generate random noise as input, therefore we need to set a batch size for propagating data through the model in advance. Here we choose a batch size of 100. Convolutional layers expect data instances to be given as three-dimensional arrays, of dimensions *width, height* and *number of channels*, thus the input must be given in this format. Since we deal with square images only, the width and height of the images are the same, equal to some value s. Recall that for MNIST and SVHN we have $s = 28$ and $s = 32$, respectively. Since we only deal with greyscale images, the number of channels is 1, which is also the depth (i.e. the number of filters) of the first convolution layer.

The given *kernel sizes* for the (transposed) convolutional layers represent the size of a side of the (square) convolution window, and the *strides* represent the step size of the convolution window both horizontally and vertically. The *padding* method is either "same" (trailing 0's at the edges of the image to ensure that the output size is equal to the input size divided by the strides) or "valid" (moving the convolution window only over real inputs, not adding any 0's).

Between the convolutional and dense layers we need to reshape the data, since dense layers expect data to be given in one dimension instead of three.

The middle part of the convolutional VAE has the same structure as that of the dense VAE, where we again use three different latent dimensions: $d = 2$, $d = 30$, and $d = 50$. After this follow more dense layers that transform the data back to the dimensions we had after the convolutional layers.

We then reshape the data into a three-dimensional array again, and follow with three transposed convolutional layers and a final convolutional layer with sigmoid activations to reconstruct the input (recall that all hidden activations are ReLUs as explained in Section 3.2).

Layer type	Output dimension(s)	Details
Input	(100, s, s, 1)	batch size = 100, width = s, height = s, #channels = 1
Conv2D	(100, s, s, 1)	#filters = 1, kernel size = 2, padding = "same", strides = 1
Conv2D	(100, $s/2$, $s/2$, 64)	#filters = 64, kernel size = 2, padding = "same", strides = 2
Conv2D	(100, $s/2$, $s/2$, 64)	#filters = 64, kernel size = 3, padding = "same", strides = 1
Conv2D	(100, $s/2$, $s/2$, 64)	#filters = 64, kernel size = 3, padding = "same", strides = 1
Flatten	(100, $(s/2) \cdot (s/2) \cdot 64$)	reshapes data to be suitable for dense layer
Dense	(100, 128)	#hidden units = 128
Dense ($\boldsymbol{\mu}, \log \boldsymbol{\sigma}$)	(100, $d + d$)	latent dimension = d
Sample $\boldsymbol{z}$	(100, d)	latent dimension = d
Dense	(100, 128)	#hidden units = 128
Dense (upsampling)	(100, $(s/2) \cdot (s/2) \cdot 64$)	upsamples data for deconvolutional layers
Reshape	(100, $s/2$, $s/2$, 64)	reshapes data to be suitable for deconvolutional layers
Conv2Dtranspose	(100, $s/2$, $s/2$, 64)	#filters = 64, kernel size = 3, padding = "same", strides = 1
Conv2Dtranspose	(100, $s/2$, $s/2$, 64)	#filters = 64, kernel size = 3, padding = "same", strides = 1
Conv2Dtranspose	(100, $s + 1$, $s + 1$, 64)	#filters = 64, kernel size = 3, padding = "valid", strides = 2
Conv2D (sigmoid)	(100, s, s, 1)	#filters = 1, kernel size = 2, padding = "valid", strides = 1

Table 4.1: Architecture of the convolutional VAE

We thus have three different dense VAE architectures and three different convolutional VAE architectures (the only difference for each being the number of latent dimensions), six VAEs in total. We use each of these to transform each of the two data sets into a latent representation, where we use convergence in validation set loss as a criterion to decide how many epochs to train each VAE. Thus we end up with twelve latent data sets; six for MNIST and six for SVHN.

4.2.2 Multilayer Perceptron

For the classification task in each active learning iteration we use a simple multilayer perceptron (MLP) with one or more hidden layers and a softmax as output, implemented in `Keras`. The input size depends on the number of latent dimensions of the latent data, either 2, 30, or 50. The softmax has 10 outputs, for each of the digits 0 to 9. For the hidden layer(s) we try out a number of architectures with either one or two hidden layers, with the following options for the number of hidden layers:

- 128
- 256
- 512
- 256, 256

- 512, 256
- 512, 512

For each of these architectures we add *dropout* to all the hidden layers, with a value of either 0.2, 0.35, or 0.5 (i.e. 20%, 35%, or 50%). Thus in total we try out 18 different MLP architectures. Training is done with stochastic gradient descent for a categorical cross-entropy loss function, with the *RMSprop* optimizer as implemented in `Keras`. The batch size for training is 100, and we train each model for 20 epochs.

We first try out these 18 architectures on each of the twelve latent data sets, as well as on the original image data, using all the labels of the training sets. Obviously this would not be possible in a true active learning setting, since labels are not available, thus the results cannot be used as a method to select the appropriate classification model in our active learning approach. Nevertheless it is interesting to evaluate how close our method can get to the state-of-the-art performance on each of the data sets, if all labels are available. Moreover, this gives us a kind of upper bound for the performance of our active learning algorithm after obtaining a certain amount if labels, since we do not expect a model to perform worse on a full data set than when a limited number of labels is available.

4.3 Active Learning

We will run experiments for the active learning approach as described in Chapter 3 with various settings and models. We always use a batch size of 100, both for training models and for batch-mode active learning. At the start of the active learning procedure, we sample a random batch of 100 instances. Then we perform a total of 29 active learning iterations, such that at the end of our procedure we have 3000 labelled instances. We will evaluate the performance (classification accuracy) on the validation set after each active learning iteration, which allows us to see whether and how much the accuracy improves when we collect more data.

For uncertainty sampling, we use three methods to evaluate uncertainty, as described in Section 3.4.1:

- *Least confident*
- *Margin sampling*
- *Entropy*

For some active learning experiments, we combine the uncertainty measures with a representativeness measure through a weighted sum, as detailed in Section 3.4.2. The representativeness measures that we use in these experiments are:

- *Density (cosine)*: density computed with cosine similarity
- *Density (Euclidean)*: density computed with Euclidean distance, normalised by the highest observed average distance
- *Density (squared Euclidean)*: density computed with squared Euclidean distance, normalised by the highest observed average squared distance
- *Density (J Divergence)*: density computed with J Divergence, normalised by the highest observed average divergence
- *KL to prior*: the KL Divergence to the prior $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$, normalised by the highest observed divergence
- *Distance to origin*: the Euclidean distance to the mean $\mathbf{0}$ of the prior, normalised by the highest observed distance

- *Squared distance to origin*: the squared Euclidean distance to the mean $\mathbf{0}$ of the prior, normalised by the highest observed squared distance

We also use these representativeness measures to evaluate the overall distribution of representativeness within a latent data set, by plotting histograms. We then compare these with histograms of representativeness in the original data sets. For the original data we use the following representativeness measures:

- *Density (cosine)*: density computed with cosine similarity
- *Density (Euclidean)*: density computed with normalised Euclidean distance
- *Density (squared Euclidean)*: density computed with mean squared Euclidean distance

Computing density as an average distance to all other data instances is very slow, already for the latent data but in particular for the higher-dimensional image data. Therefore we use a significantly faster approximation to compute the density values for the original data sets. For each instance, we compute its density as the average distance to 1% of the instances of the full data set, rather than to all the instances. This 1% is selected as a random sample, different for each instance.

In our active learning strategy we want to query batches of instances at once, instead of single instances, before retraining the classification model. To prevent querying batches with a lot redundancy, we use a number of techniques to encourage diversity within batches. We first use a few simple and fast techniques to query batches, as described in Section 3.4.3:

- *Top k*
- *Random partitions*
- *Subsampling* with *top k*
- *Subsampling* with *random partitions*

Furthermore, we use a *greedy* method as described in Section 3.4.3 to enforce diversity, with the following distance measures:

- *Cosine similarity* (reversed and normalised to be between 0 and 1)
- *Euclidean distance* (normalised by the highest observed distance)
- *Squared Euclidean distance* (normalised by the highest observed squared distance)
- *J Divergence* (normalised by the highest observed divergence)

These greedy methods are much slower however when used on a full data set. Therefore we perform them on a very small subsample of only 1% of the data.

We compare all our techniques with *random sampling* as a baseline approach, where we simply sample a random batch of instances before retraining the classification model. We also refer to this as passive learning, as opposed to active learning.

Chapter 5

Results

In this chapter we describe the results of our experiments in detail. We first look into the performance of VAEs to obtain latent representations. Then we evaluate various MLP architectures for the classification model in our active learning procedure. Based on those results, we fix the VAE and MLP architectures to work with, and we experiment with the various settings and possible techniques of the active learning scheme.

Most experiments include evaluation of a wide range of settings, where we test each setting only once. In the final section of this chapter, we repeat the experiments that proved most successful multiple times, in order to achieve more statistical significance. Thus, for a more concise report of our strongest results, we refer directly to Section 5.4.

5.1 VAE Latent Space

5.1.1 Trained VAEs

Table 5.1 gives an overview of the different VAEs we trained (the architectures of the dense and convolutional VAEs are explained in Section 4.2.1). We trained each VAE for either 32 or 64 epochs, where we checked whether the validation loss was converging to decide how long to train. The table shows both the validation and training loss at the end of training for each of the models. These values are always close together, indicating that there is likely not much overfitting. We use each of these VAEs to obtain a latent data set, thus we have twelve latent data sets in total, each of which can be identified by the following three values: original data set, VAE type, and latent dimension (e.g. MNIST, convolutional, 30d).

The (validation) loss gives us some objective indication about how well a VAE performs. In general it seems clear that it is much easier to represent MNIST than SVHN, as can be expected. It is also clear that convolutional VAEs outperform dense VAEs, which is unsurprising for image data. It remains to be seen however if lower loss values truly provide a good indication for whether the learned representations are indeed better structured and give better results in our active learning approach.

Since labels are scarce (or unavailable) in an active learning setting, we need methods to evaluate the VAE latent spaces (besides looking at the validation loss) that do not require labelled data. Later we will look into some more subjective methods to evaluate the representations learned from a VAE, but first we discuss another more objective method that can give us information about a VAE's performance.

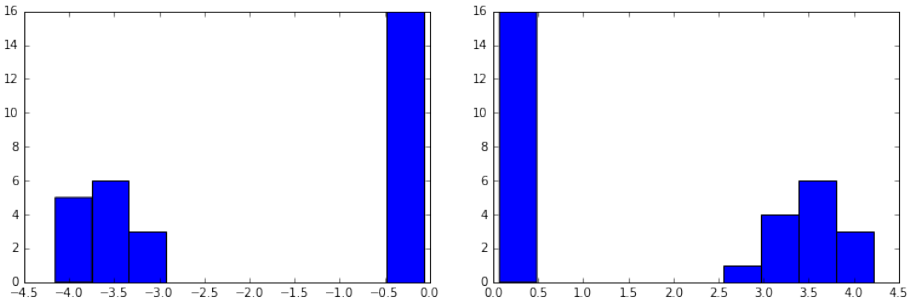
5.1.2 Used Latent Dimensions

Although we trained VAEs with latent dimensions 2, 30, and 50, it turns out that the resulting latent data does not always "use" all of these dimensions, i.e. for some dimensions all latent variables remain very close to zero. We can see this by finding the minimum and maximum value

Data	VAE type	Latent dim. (used)	Epochs	Validation loss	Training loss
MNIST	dense	2 (2)	64	139.079	137.4715
MNIST	dense	30 (14)	64	101.384	99.30045
MNIST	dense	50 (14)	64	101.289	99.76947
MNIST	conv.	2 (2)	32	139.1141	133.7208
MNIST	conv.	30 (30)	64	58.65009	55.29481
MNIST	conv.	50 (50)	64	56.17006	53.87701
SVHN	dense	2 (2)	32	651.0906	651.8179
SVHN	dense	30 (7)	64	645.6823	646.3857
SVHN	dense	50 (6)	64	645.8237	646.6826
SVHN	conv.	2 (2)	32	647.0779	648.0242
SVHN	conv.	30 (30)	64	624.4748	623.3472
SVHN	conv.	50 (48)	64	622.1815	622.1009

Table 5.1: Overview of trained VAEs

obtained for each dimension in the latent space, for a given latent data set. Figure 5.1 shows histograms for these minimum and maximum values for the latent space data (training data only) obtained through the dense VAE with latent dimension 30 on the MNIST data set. We can clearly see that for 16 dimensions, all values remain very close to zero, whereas for the other 14 dimensions values fall within a much wider range. Thus we say that only 14 dimensions are "used".



(a) Minimum values

(b) Maximum values

Figure 5.1: Histogram of minimum/maximum values for each dimension in 30-dimensional latent data from dense VAE on MNIST

In Table 5.1 the number of latent dimensions that is really used is shown in brackets after the number of latent dimensions of the VAE. It is clear that the dense VAEs tend to not use all latent dimensions, which likely leads to less useful latent space representations for our active learning setting. The convolutional VAEs do seem to utilise the full dimensionality of the latent space. Checking whether all latent dimensions are used gives us another objective way to evaluate the performance of each VAE.

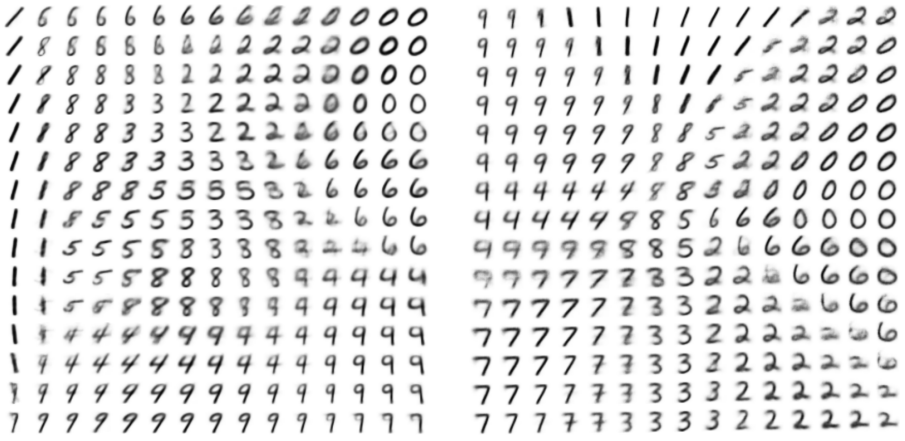
5.1.3 2D Visualisations

For two-dimensional latent spaces we can generate nice plots that can give us good insights into the structure of the latent space, and how well the generative part of the model performs. We show two types of two-dimensional visualisations, one that doesn't require any labels (and thus works in any active learning setting), and one that only works for a fully labelled data set.

Manifold Visualisations

Without needing any labels, we can sample latent space coordinates proportionally to the model's distribution over latent space, and decode them into images. Since our VAEs are trained with a standard multivariate Gaussian as the prior distribution, we can do this by sampling linearly spaced percentiles of the inverse CDF (cumulative distribution function) of a standard multivariate Gaussian. We can layout these decoded images in a two-dimensional grid, which gives us a sense of the full latent manifold.

Figure 5.2 shows such manifold visualisations for the 2-dimensional latent data for MNIST, obtained through a dense and convolutional VAE, respectively. We clearly see that digits are grouped together, and many latent space coordinates get encoded into clearly distinguishable digits. In particular we also notice that all digits are actually there, we can generate each of the 10 digits from some latent space coordinates. Both visualisations show some border areas in which the generated images are blurred symbols that aren't really recognisable and often seem to be a combination of two digits. But we cannot see a clear difference in quality between the results of the dense VAE and the convolutional VAE, both seem to perform fairly well.



(a) Dense VAE (b) Convolutional VAE

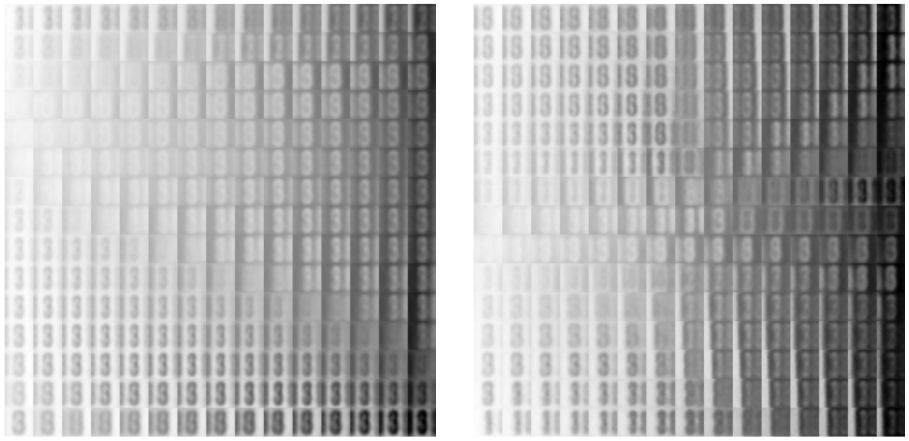
Figure 5.2: Manifold visualisations for two-dimensional VAEs on MNIST

Now we look at the manifold visualisations for the SVHN data, as shown in Figure 5.3. Here we see that the model has not been able to learn the same type of structure as for MNIST. All generated images seem to resemble either a 3 or an 8 in more or less the same shape, and differences are mainly in shading; whether there is a light image on a dark background or a dark image on a light background. This seems to imply that we simply need more than two dimensions to properly represent this data set. We cannot visualise manifolds of higher dimensions in a nice way such as this, but there are other ways we can get a subjective insight into how good the latent space representation is, without requiring any labels. We will look into this in the next sections.

Distribution of Classes in Latent Space

Since we do have fully labelled data sets available here, there is another type of two-dimensional visualisation that we can use to gain insight into how well the VAEs structure the data. We simply plot all instances as dots in the two-dimensional latent space, assigning a different colour to each different class (i.e. each different digit). We visualise the test sets like this to get a view of how well the VAE generalises.

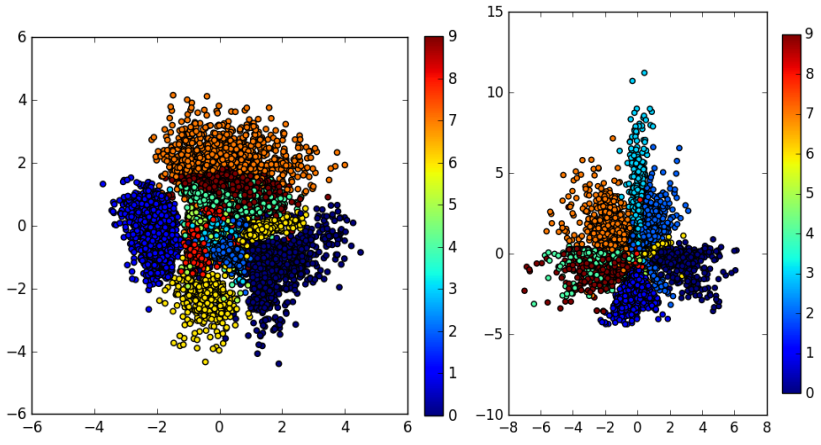
Figure 5.4 shows these visualisation for our two latent data test sets for MNIST. The structure of the latent space is different for the dense and for the convolutional VAE, but both show similar behaviour. We can see that both VAEs structure the different classes quite well; groups of instances



(a) Dense VAE (b) Convolutional VAE

Figure 5.3: Manifold visualisations for two-dimensional VAEs on SVHN

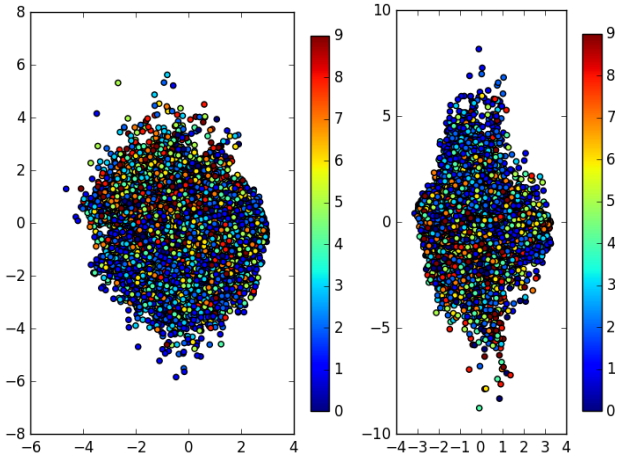
from the same class generally form dense and uninterrupted regions in the latent space. There are however still regions with a lot of overlap between classes, and certain classes inhabit only very small regions of the latent space compared to others. All in all the VAEs seem to certainly have learned a good and useful structure of the data, but two dimensions may be too few to really separate the data well enough to be able to learn a simple classifier on the latent data that performs well.



(a) Dense VAE (b) Convolutional VAE

Figure 5.4: Distribution of digit classes for two-dimensional VAE latent data for MNIST (test sets)

The class distribution over the latent space for the two-dimensional SVHN data sets is visualised in Figure 5.5. Here we clearly see that the VAEs have not been able to learn enough to meaningfully structure the data in a two-dimensional latent space. Together with what we saw from the manifold visualisation in Figure 5.3 we can conclude that these latent representations are likely not going to be good enough to be used in our active learning approach.



(a) Dense VAE (b) Convolutional VAE

Figure 5.5: Distribution of digit classes for two-dimensional VAE latent data for SVHN (test sets)

5.1.4 Reconstructions and Interpolations

We saw how to generate useful visualisations to (subjectively) evaluate the structure of two-dimensional latent space representations. But we cannot make the same kind of visualisations for latent spaces with more dimensions. Instead we look at two other types of visualisations, that work for any number of latent dimensions.

Reconstructions

First of all, we can look at how well the VAEs reconstruct the data. To do this, we randomly sample a few images from (the test set of) the original data, and use them as input for the VAE. We then compare the original images with their reconstruction (the output of the VAE).

Figure 5.6 shows ten randomly sampled images and their reconstructions for all VAEs on MNIST and SVHN. For MNIST we observe that especially the two-dimensional VAEs produce blurry images that sometimes resemble a different digit than the original. The other dense VAEs also produce slightly blurry images that are sometimes unclear, whereas the higher-dimensional convolutional VAEs produce clear and quite sharp reconstructions.

Looking at the SVHN reconstructions, we see that the 2-dimensional VAEs only seem to be able to construct one particular shape (either a 3 or an 8), in different shades. The other dense VAEs do not perform much better, generating badly recognisable shapes that often resemble a different digit than the original. Only the higher-dimensional convolutional VAEs seem to produce useful output, generating reconstructions that are mostly recognisable as the same digit as their original.

All in all we can conclude that the convolutional VAEs perform much better on either data set, unsurprisingly. Moreover, it seems that we need sufficient dimensions (more than two), especially for harder data sets such as SVHN, to produce encodings that allow for good reconstruction of the data.

Interpolations

Another way to gain some insight into how well the latent space describes and structures the data is to look at interpolations between two data points in latent space. To do this, we first sample two random data instances and obtain their corresponding latent space coordinates. We then linearly interpolate from the first to the second digit in latent space, obtaining latent coordinates along the way (with equal distances in between). We decode each of these coordinates, and plot them

(a) MNIST, Dense VAE 2d (b) SVHN, Dense VAE 2d

(c) MNIST, Dense VAE 30d (d) SVHN, Dense VAE 30d

(e) MNIST, Dense VAE 50d (f) SVHN, Dense VAE 50d

(g) MNIST, Convolutional VAE 2d (h) SVHN, Convolutional VAE 2d

(i) MNIST, Convolutional VAE 30d (j) SVHN, Convolutional VAE 30d

(k) MNIST, Convolutional VAE 50d (l) SVHN, Convolutional VAE 50d

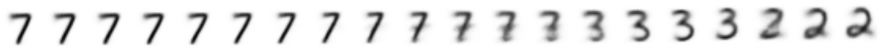
Figure 5.6: Randomly sampled MNIST and SVHN test set digits and their VAE Reconstructions

next to each other. This way we obtain an interpolation from one digit to another, which allows us to see how latent space coordinates in between data points are decoded; whether they represent proper and recognisable shapes, and whether there is a direct switch from the first digit class to the second, or whether other classes are "visited" in between as well.

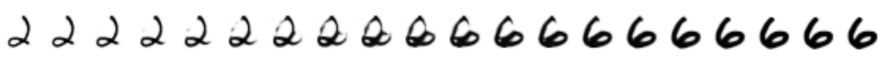
Although such visualisations do not provide much insight into the full distribution of the data in the latent space, it can show some close-ups that show us what the latent space is like locally. In Figure 5.7 we see some examples of interpolations for MNIST latent spaces from convolutional VAEs with latent dimensions 2 and 50. For the two-dimensional one we see that the (interpolated) images are a bit blurry, and when interpolating from a 7 to a 2, we actually pass another digit class corresponding to the digit 3. The interpolation in 50 dimensions on the other hand shows sharper images, and a clear switch from the first digit class directly to the second. We see some unrecognisable shapes in the middle that do not resemble any digit, but this means that there are likely no data points in this part of the latent space. Thus, from this point of view, it appears that the digit classes 2 and 6 are well separated in this direction.

Figure 5.8 shows interpolations for SVHN latent spaces from convolutional VAEs with latent dimensions 30 and 50. Here we see that the results are more blurry than for MNIST; SVHN digits are harder to reconstruct. We do however see clear switches from one digit class to the other, which gives us some indications that the digit classes are well separated, at least in these directions.

When inspecting more of such interpolations (not shown here) for all the different VAE latent



(a) Convolutional VAE 2d



(b) Convolutional VAE 50d

Figure 5.7: Linear interpolation in latent space for MNIST

(a) Convolutional VAE 30d

(b) Convolutional VAE 50d

Figure 5.8: Linear interpolation in latent space for SVHN

spaces, we can draw similar conclusions as before when we looked at reconstructions. The convolutional VAEs perform much better, in particular those with 30 or 50 latent dimensions. For MNIST, other VAEs also produce acceptable results, but for SVHN we really need a convolutional VAE with sufficient dimensions.

5.2 MLP Classification on Full Data Sets

As explained in Section 4.2.2, we tried out a number of MLP architectures to serve as the classification model in our active learning procedure. Since we do have all labels available for our test sets, we can evaluate these models on the full (latent) data sets as well. This gives us some upper bounds for the classification performance in our active learning scheme. The results can be found in Table 5.2 (for MNIST) and Table 5.3 (for SVHN). All MLPs were trained for 20 epochs, at which point the validation accuracy had converged in all cases. We also trained the MLPs on the original data sets, for comparison. The three best performances on each (latent) data set are shown in bold face, and the single highest performance for each MLP is underlined.

Looking at the results for MNIST, we can confirm what we already saw in the previous section; convolutional VAEs with 30 or 50 latent dimensions perform best. In particular it seems that 30 dimensions yields slightly better results, mostly even better than on the original data (although MLPs on MNIST work fairly well). As for MLP architectures, architectures with two layers generally perform slightly better than those with only one layer. Lower dropout values also seem to work better, although this effect is not very significant. The best performing combination appears to be the a two-layer MLP with twice 512 hidden layers on a 30-dimensional latent space from a convolutional VAE.

For SVHN we can also reach the same conclusions as before; we need a convolutional VAE with sufficient latent dimensions (30 or 50 in our case) to get good performance on SVHN. In this case the 50-dimensional latent space clearly outperforms the one with 30 dimensions. An MLP on these latent spaces performs a lot better than on the original data. This is not surprising, as MLPs do not generally perform well on image data. Having already used convolutional layers in the VAE thus helped obtaining a representation from which we can learn well with a simple MLP. From all the different MLP architectures we again see that those with two hidden layers generally perform slightly better.

Hidden units:	**Dropout:**	MNIST dense 2d	MNIST dense 30d	MNIST dense 50d	MNIST conv. 2d	MNIST conv. 30d	MNIST conv. 50d	*MNIST original*
512	**0.2**	0.819	0.9756	0.9764	0.8194	0.985	0.9825	*0.9817*
512	**0.35**	0.8144	0.9754	0.9758	0.8224	0.9844	0.9843	***0.9832***
512	**0.5**	0.8141	0.9752	0.9749	0.8206	0.9855	0.9836	*0.9813*
256	**0.2**	0.8164	0.9748	0.9749	0.8232	0.9835	0.9835	*0.9822*
256	**0.35**	0.8117	0.9739	0.9743	0.8188	0.9819	0.9823	*0.9818*
256	**0.5**	0.8086	0.9722	0.973	0.8205	0.9828	0.9805	*0.9807*
128	**0.2**	0.8061	0.9723	0.9718	0.8193	0.9819	0.9798	*0.9781*
128	**0.35**	0.8067	0.9708	0.9706	0.8187	0.9779	0.9771	*0.9786*
128	**0.5**	0.8034	0.9654	0.9676	0.8187	0.9759	0.9705	*0.9768*
256, 256	**0.2**	**0.8211**	**0.978**	**0.9788**	**0.8345**	0.987	0.9849	*0.9826*
256, 256	**0.35**	**0.8212**	0.9777	0.9767	**0.8328**	0.986	**0.9857**	*0.9821*
256, 256	**0.5**	0.8181	0.9743	0.9732	0.8278	0.9813	0.981	*0.9782*
512, 256	**0.2**	0.8198	**0.9783**	**0.9777**	0.83	0.9863	**0.9866**	***0.9847***
512, 256	**0.35**	0.8187	**0.9791**	0.9773	0.8301	**0.9873**	0.9853	*0.9831*
512, 256	**0.5**	0.8159	0.977	0.9758	0.828	0.9859	0.9846	*0.979*
512, 512	**0.2**	**0.8205**	**0.978**	**0.9778**	0.8312	**0.9874**	**0.9858**	*0.9809*
512, 512	**0.35**	0.8165	0.9774	0.9755	0.8316	**0.9872**	**0.9857**	***0.9834***
512, 512	**0.5**	0.8191	0.9764	0.9767	**0.8319**	**0.9872**	0.9844	*0.9824*

Table 5.2: Classification accuracies for various MLP architectures on various VAE latent spaces for MNIST, after 20 epochs of training

Hidden units:	**Dropout:**	SVHN dense 2d	SVHN dense 30d	SVHN dense 50d	SVHN conv. 2d	SVHN conv. 30d	SVHN conv. 50d	*SVHN original*
512	**0.2**	0.200885	0.365269	**0.3675**	0.199115	0.782615	0.823	***0.698***
512	**0.35**	0.197231	0.367462	0.361154	0.198962	0.779385	0.818231	*0.641308*
512	**0.5**	0.198385	0.361385	0.359615	0.199269	0.773462	0.817923	*0.587*
256	**0.2**	0.201538	0.368	0.361923	0.200731	0.7765	0.817385	*0.656885*
256	**0.35**	0.200115	0.362077	0.365192	0.199077	0.769346	0.811154	*0.635769*
256	**0.5**	0.201731	0.361	0.360923	0.198231	0.757654	0.800808	*0.457231*
128	**0.2**	0.198038	0.3605	0.361462	0.197962	0.764654	0.803423	*0.532731*
128	**0.35**	0.198423	0.357846	0.360423	0.199077	0.753038	0.787846	*0.451*
128	**0.5**	0.200846	0.355692	0.354692	0.198692	0.734615	0.770692	*0.3985*
256, 256	**0.2**	**0.213192**	**0.373038**	**0.368154**	**0.213**	0.779462	0.816654	*0.672115*
256, 256	**0.35**	0.210346	0.365923	0.361308	0.209192	0.774923	0.811577	*0.542846*
256, 256	**0.5**	0.206	0.358615	0.362962	0.203154	0.759769	0.796538	*0.351692*
512, 256	**0.2**	**0.212654**	**0.370692**	0.366269	**0.212769**	**0.786269**	**0.825269**	***0.726692***
512, 256	**0.35**	**0.210769**	**0.371192**	**0.368346**	0.208615	0.782269	0.818962	*0.628038*
512, 256	**0.5**	0.204192	0.362962	0.362654	0.205077	0.770154	0.808962	*0.380846*
512, 512	**0.2**	0.208962	0.366923	0.364077	**0.212**	**0.786885**	**0.823769**	***0.704***
512, 512	**0.35**	0.207038	0.362846	0.364308	0.207923	**0.788346**	**0.823769**	*0.619731*
512, 512	**0.5**	0.210269	0.368346	0.365385	0.207423	0.775769	0.810038	*0.298462*

Table 5.3: Classification accuracies for various MLP architectures on various VAE latent spaces for SVHN, after 20 epochs of training

5.3 Active Learning

Based on the results from the previous sections, we decide to continue with only one VAE latent space for each data set. For MNIST we choose the convolutional VAE with 30 latent dimensions, for SVHN the convolutional VAE with 50 dimensions. We use the same classification model for all our further experiments; an MLP with two hidden layers of 512 hidden units each, with dropout of 0.2 for each hidden layer. With these models and data sets we run the active learning approach as described in Chapter 3, with various settings as listed in Chapter 4.

5.3.1 Uncertainty Sampling

We first try our active learning scheme using only an uncertainty measure to express the informativeness of unlabelled instances. We try all three uncertainty measures (*least confident*, *margin sampling*, and *entropy*) for each of the two (latent) data sets. To query batches (of size 100) at

once we use two basic methods, *top k* and *random partitions*, both on the full data sets and with 10% *subsampling*. We compare the results with a basic random sampling approach, in which we just query batches of 100 randomly sampled instances in each active learning iteration.

The results for MNIST (30-dimensional latent space) are shown in Figure 5.9. Displayed are the validation (left) and training (right) accuracies (on the vertical axis) at the end of 20 epochs of training, after having obtained a certain amount of labelled instances (on the horizontal axis). Each active learning iteration queries and labels 100 new instances, so the more iterations done, the more labels we train on. We see that as we obtain more labels, the performance generally increases, although not always. All uncertainty sampling methods clearly outperform random sampling. They also seem to select data in such a way that it prevents overfitting, which does seem to happen when randomly sampling queries. We can see this from the behaviour of the training accuracy, compared to the validation accuracy. For random sampling the classification overfits on the training data, while performing worse on the validation data. There is not a clear and significant difference in performance between the various uncertainty methods however, they all perform quite similar, although there is a slight indication that margin sampling performs a bit better. It is in fact surprising that *top k* performs so well, since in general it is prone to selecting many similar instances.

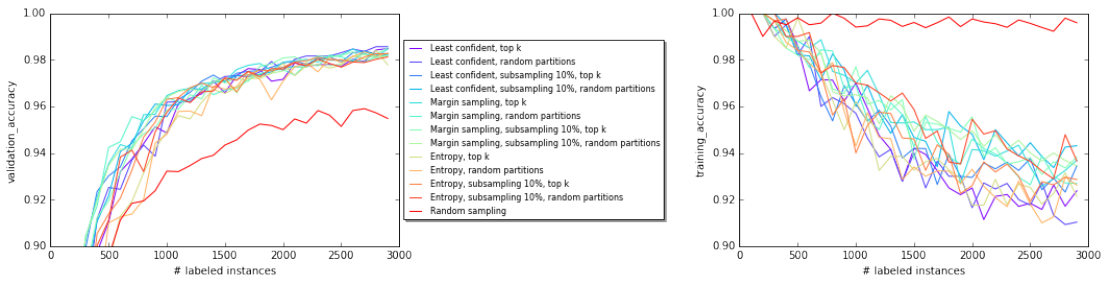


Figure 5.9: Uncertainty sampling validation (left) and training (right) accuracies after training on MNIST

Similar plots for the SVHN (50-dimensional latent) data set are shown in Figure 5.10. Here we see that in fact not many uncertainty sampling methods outperform random sampling. Only the marging sampling methods perform slightly better than random sampling, where it doesn't matter much which technique is used to query batches of uncertain instances. Again it is remarkable that *top k* works well as a batch selection technique.

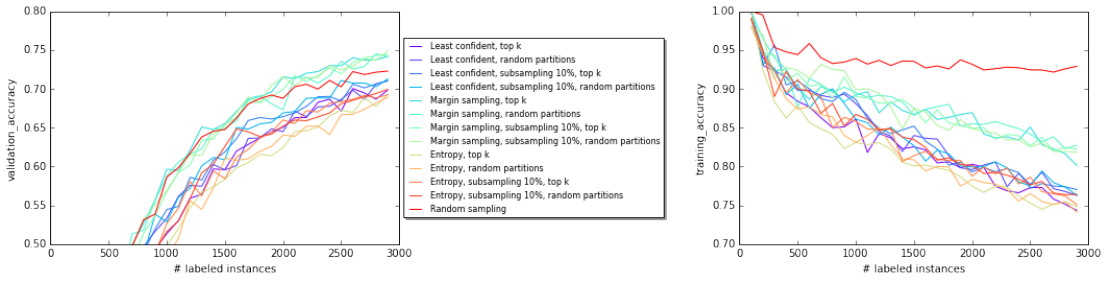


Figure 5.10: Uncertainty sampling validation (left) and training (right) accuracies after training on SVHN

Based on these experiments, margin sampling seems to be the best uncertainty measure in

our situation. Therefore we will consider this as our standard uncertainty measure in further experiments. Since the batch selection technique does not seem to make much of a difference so far, we choose to use 10% subsampling with random partitions for now. The subsampling makes uncertainty sampling even faster, since 90% less predictions need to be made by the MLP for unlabelled instances.

5.3.2 Training in Latent Space vs. Original Space

We saw that active learning through uncertainty sampling (in particular margin sampling) can outperform passive learning (i.e. random sampling), when training MLPs on the latent space data. But it is also interesting to evaluate the effect of training on the latent data as opposed to using the original pixel representations. In Section 5.2 we saw that our MLP can still achieve acceptable results on the original MNIST data, although it performs better on the latent data. We can compare active and passive learning using the same MLP on the original MNIST with our own approach of learning with latent space representations. This can show us whether it is worth training a VAE first for a simple data set as MNIST, rather than just performing active learning with a simple MLP (such that active learning iterations can be executed fast).

For SVHN we saw that MLPs simply perform significantly worse on the original data than on latent data, from which we may conclude in advance that an active learning scheme with an MLP as its classification model will not perform better than our VAE approach for this data set. This will likely also be the case for more complicated image data sets, where convolutional layers are essential. Therefore, we will only consider MNIST when comparing training in latent space with training on original data.

Figure 5.11 shows the validation and training accuracies after training the MLP for increasing numbers of labelled instances, queried either actively or passively, on both the latent and original MNIST data. Active learning clearly outperforms random sampling in all the cases, but learning on latent data also clearly outperforms learning on the original data. Generally, obtaining more labels improves the (validation) accuracy, but it is interesting that this effect seems more reliable for the latent data. MLPs on the original data sometimes make big drops in performance when trained with an extra batch of data, whereas the performance on the latent data smoothly increases almost monotonically. This effect is already visible for random sampling, but is particularly noticeable with uncertainty sampling.

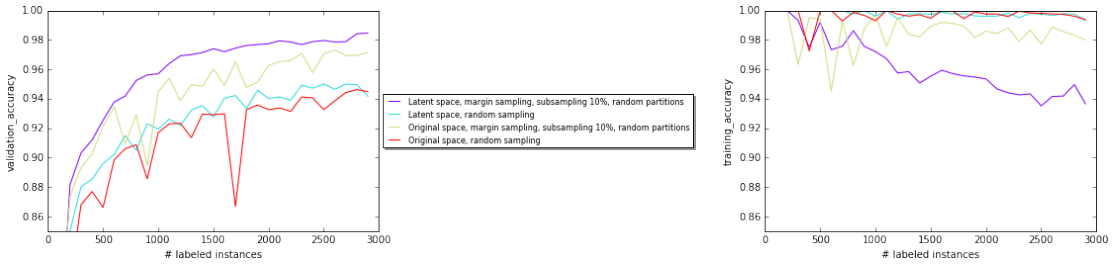


Figure 5.11: Margin sampling and random sampling validation (left) and training (right) accuracies after training on MNIST in 30-dimensional latent space and original image space

Looking at the training accuracy, it appears that random sampling tends to select data on which the model easily overfits. This effect is only slightly less with uncertainty sampling on the original data, but the problem particularly disappears for uncertainty sampling on latent data.

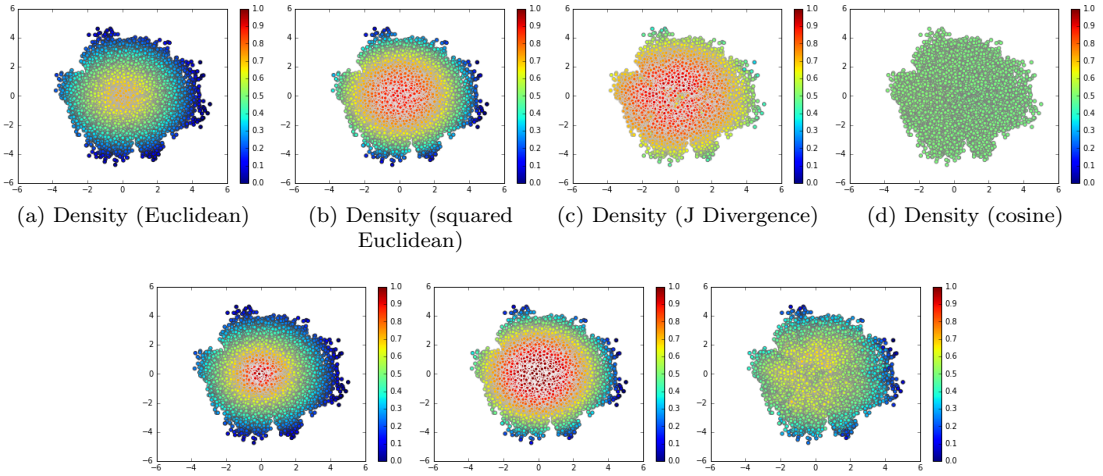
These results show that training a VAE first is definitely beneficial if we want to use an active learning scheme with a simple classification model like our MLP, even if this simple model already performs quite acceptably on the original data.

5.3.3 Representativeness

One of our hypotheses was that working in latent space can prevent uncertainty sampling from querying outliers that are not very informative because they do not represent the data well. To evaluate this, we will first look into how the various representativeness measures (as listed in Section 4.3) behave in latent space. We first look at the distribution of data instances in latent space based on their representativeness and compare this to the original data set, to see if the latent representations indeed have less outliers. Then we also explicitly incorporate representativeness measures in our active learning strategy, to prevent querying any remaining outliers, and see if this improves performance.

Two-Dimensional Representativeness Distribution

Although we are only interested in the chosen latent spaces (from convolutional VAEs, 30 latent dimensions for MNIST, 50 for SVHN), we first have a look at the representativeness for the latent space from the convolutional VAE with two latent dimensions for MNIST, to get a general idea about how the various representativeness measures behave. If we have only two latent dimensions, we can nicely plot the full latent space, using colours to express the representativeness of an instance, as is shown in Figure 5.12. We can also look at the frequencies of representative values for each of the measures, by means of histograms, as shown in Figure 5.13.



(a) Density (Euclidean) (b) Density (squared Euclidean) (c) Density (J Divergence) (d) Density (cosine)

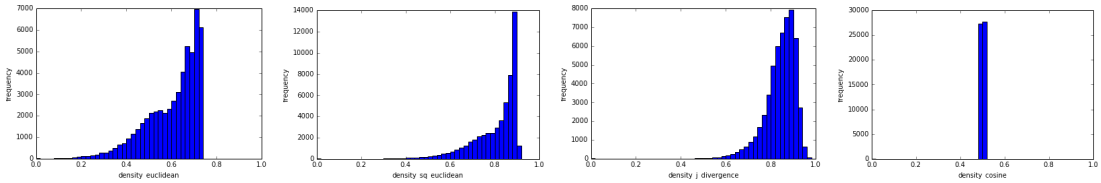
(e) Distance to origin (f) Squared distance to origin (g) KL to prior

Figure 5.12: Representativeness visualised in two-dimensional latent space for MNIST

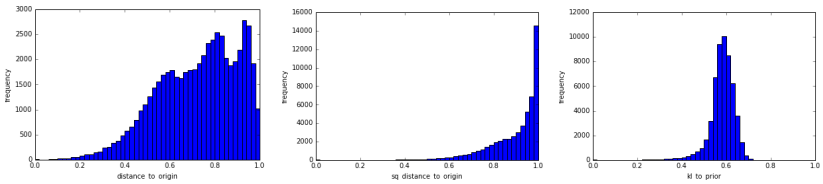
We can see that all measures but *Density (cosine)* consider instances near the origin more representative, although the scales in which they do this are different. In particular we can draw clear parallels between *Density (Euclidean)* and *Distance to origin*, between *Density (squared Euclidean)* and *Squared distance to origin*, and between *Density (J Divergence)* and *KL to prior*, although for the latter pair the distributions are a bit shifted compared to each other. The density measures are very slow to compute, requiring quadratic time in the number of data instances, whereas the other measures are very fast, requiring only linear time.

The *Density (cosine)* measure considers all data to be similarly representative. This can be explained because the VAE distributes the data instances quite evenly over a circle around the origin, as a result of the two terms in its loss function. Therefore it will not be useful to use this measure in active learning explicitly, but it does give an indication that data indeed becomes more representative in latent space.

Since the density measures are either very similar to a significantly faster method, or simply



(a) Density (Euclidean) (b) Density (squared Euclidean) (c) Density (J Divergence) (d) Density (cosine)



(e) Distance to origin (f) Squared distance to origin (g) KL to prior

Figure 5.13: Histograms of representativeness according to various measures in two-dimensional latent space for MNIST

assign the same value to almost all instances, it does not seem very useful to use these considerably slower density measures for representativeness. Instead, because of our latent space representation, we can compute representativeness measures a lot faster.

Higher-Dimensional Representativeness Distributions

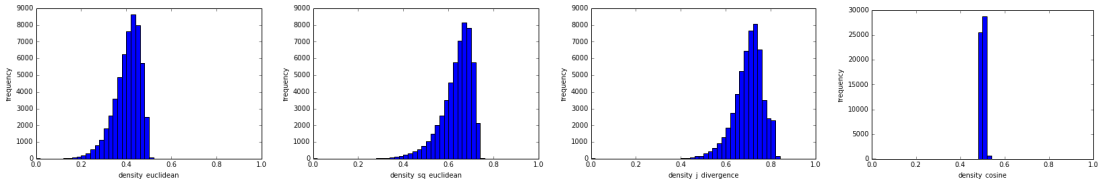
For higher-dimensional latent spaces we cannot easily visualise the full latent space, but we can still look at histograms of the distribution of representativeness values. Figure 5.14 shows these histograms for the 30-dimensional latent data for MNIST. We see that here all distributions (save *Density (cosine)*) are actually very similar, more so than in the two-dimensional situation we saw before. This seems to confirm that also for 30 dimensions we do not need to use the slow-to-compute density measures, but we can instead use measures that are significantly faster to compute, such as *(Squared) Distance to origin* or *KL to prior*.

The distribution of *Density (cosine)* gives us an indication that data is quite evenly distributed in a sphere around the origin in latent space, as it was in two dimensions (a circle is a two-dimensional sphere).

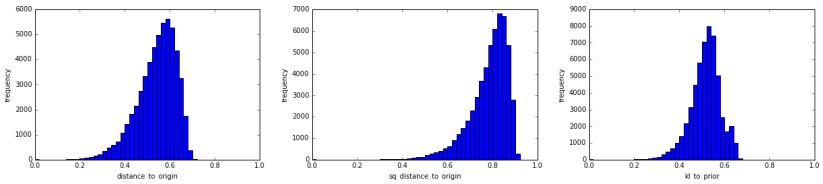
If we look at the representativeness histograms for the 50-dimensional latent data for SVHN, as shown in Figure 5.15, we see quite similar distributions as for the MNIST latent data. Thus we can draw the same conclusions here.

In general it seems that we can conclude that the VAEs indeed transform data such that it roughly forms a sphere around the origin in latent space. As such, density with cosine similarity gives very similar values for all instances, whereas all the other representativeness measures basically assign higher values to instances closer to the origin (the centre of mass of the sphere). In other words, the instances that are considered outliers are those nearer to the boundary of the sphere. If the data is quite densely distributed over the sphere however, classifying instances is important anywhere in the sphere, not just around the origin. This seems to make these representativeness measures not very suitable, and using these representativeness measures in latent space may not lead to much improvement for uncertainty sampling.

Transforming data to a latent space representation with a VAE may have eliminated the occurrence of data that does not represent its underlying distribution altogether, by explicitly imposing a prior distribution as part of the VAE loss function. This way the problem of uncertainty sampling being prone to querying outliers can be addressed by simply working in latent space, without needing to explicitly model representativeness as a density.

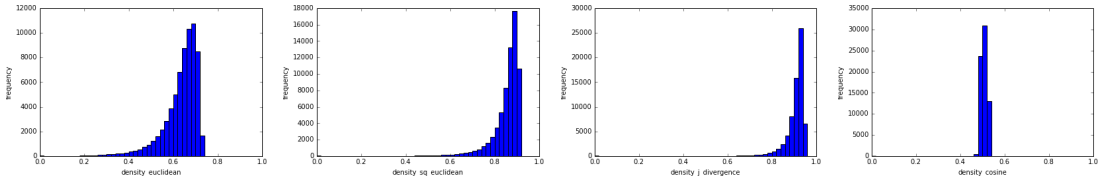


(a) Density (Euclidean) (b) Density (squared Euclidean) (c) Density (J Divergence) (d) Density (cosine)

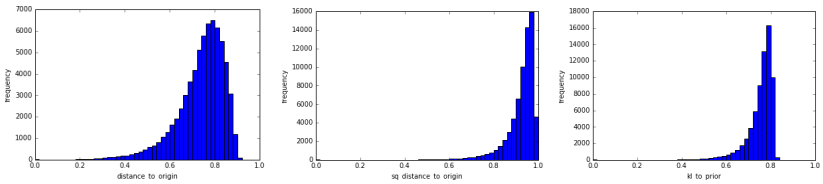


(e) Distance to origin (f) Squared distance to origin (g) KL to prior

Figure 5.14: Histograms of representativeness according to various measures in 30-dimensional latent space for MNIST



(a) Density (Euclidean) (b) Density (squared Euclidean) (c) Density (J Divergence) (d) Density (cosine)



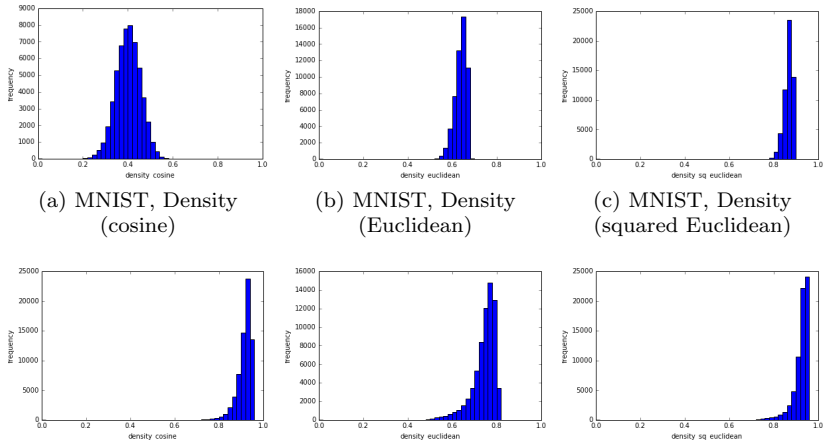
(e) Distance to origin (f) Squared distance to origin (g) KL to prior

Figure 5.15: Histograms of representativeness according to various measures in 50-dimensional latent space for SVHN

Representativeness Distribution for Original Data

We can also express the representativeness of original (image) data instances as densities in the original data space. Histograms for this are shown in Figure 5.16, for both MNIST and SVHN. We see that distributions are similar than those for the latent data sets when using (squared) Euclidean distance to compute density, and that most instances are assigned fairly similar representativeness values. This may imply that density measures with (squared) Euclidean distance are not very suitable for this kind of image data.

More interesting however is the fact that density with cosine similarity does not assign similar representativeness to all instances, as it did for latent data. The original data is given as pixel intensities. An image for which the pixel intensities are all (roughly) equal to those of another picture multiplied by some fixed scalar, is generally considered to be very similar to that image (consider e.g. two images containing the same shape, one in grey and one in black). Cosine similar-



(a) MNIST, Density (cosine)
(b) MNIST, Density (Euclidean)
(c) MNIST, Density (squared Euclidean)
(d) SVHN, Density (cosine)
(e) SVHN, Density (Euclidean)
(f) SVHN, Density (squared Euclidean)

Figure 5.16: Histograms of representativeness according to various measures for MNIST and SVHN (original image data)

ity captures such similarity, which makes it a quite decent similarity measure for greyscale images, especially if we are interested in shape recognition. Therefore, density with cosine similarity seems a reasonable measure for representativeness of image data. The histogram of this measure shows that not all instances are considered more or less equally representative, plenty score well above or below average.

Comparing this to the results for the latent data, it appears that we may indeed have prevented the existence of outliers in the data, by learning latent space representations of the data through VAEs.

Incorporate Representativeness in Active Learning

We evaluate whether explicitly incorporating our representativeness measures can improve uncertainty sampling by preventing to sample outliers, as described in Section 3.4.2. We do this for all our representativeness measures, for a few different ratios of relative importance between uncertainty and representativeness. As usual we use margin sampling as uncertainty measure and select batches by means of 10% subsampling with random partitions. The results for MNIST are shown in Figure 5.17.

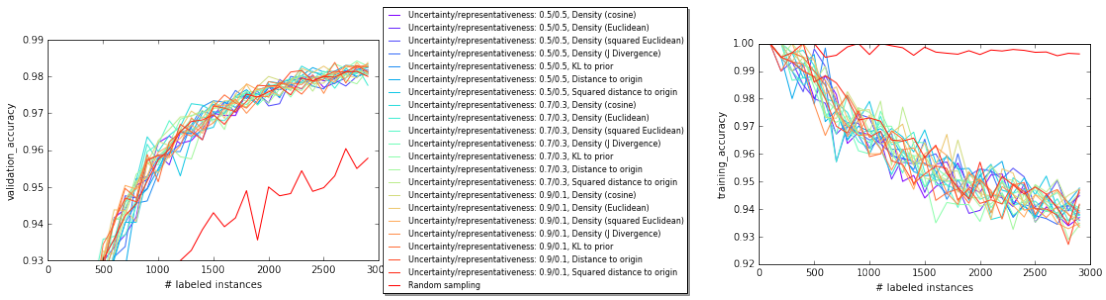


Figure 5.17: Margin sampling with representativeness validation (left) and training (right) accuracies after training on MNIST

Although the plot contains many experiments, it is easy to see that all of them perform roughly

the same. In particular, if we compare the results to those for simple uncertainty sampling in Figure 5.9, we see that there is no improvement for incorporating representativeness.

Figure 5.18 shows the results of the same experiment for SVHN. Here we see that again incorporating representativeness provides no improvement over simple margin sampling (see Figure 5.10). In fact, the performance is generally worse, and is often even worse than random sampling.

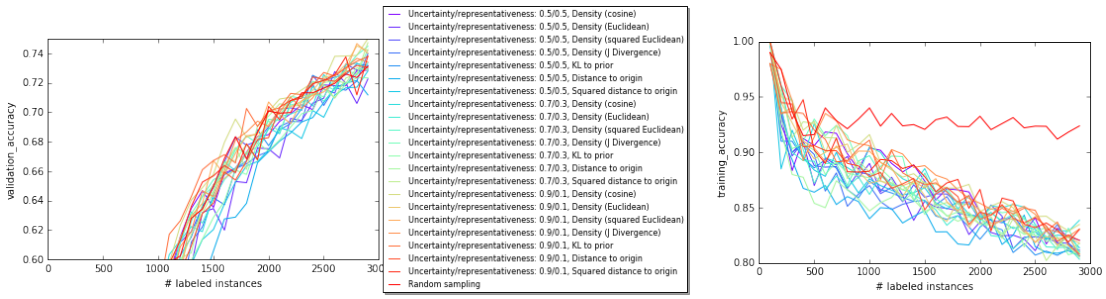


Figure 5.18: Margin sampling with representativeness validation (left) and training (right) accuracies after training on SVHN

The results so far match our earlier hypothesis that a VAE may transform data to a latent space representation in which all data instances match the underlying data distribution well. This would eliminate the need for an explicit representativeness measure in uncertainty sampling, and incorporating such a measure anyway would not lead to better results, and might even affect performance negatively. To be able to confirm this hypothesis however, we need to see if using representativeness measures indeed does provide improvements when using the original data for training the classification model. Otherwise the lack of improvement when incorporating representativeness while training in latent space might just as well mean that this method does not work at all, or that it is not necessary for the data that we use.

Incorporating Representativeness with Original Image Data

To see if our representativeness measures do improve the performance of the active learning scheme when not using the VAE latent representations, but training MLPs on original image data instead, we run some experiments with several different representativeness measures on the original data. The results are shown in Figure 5.19. If we compare these results with those for simple uncertainty sampling in original image space of Figure 5.11, we see that in fact we get similar or worse performance when incorporating representativeness measures. Thus, either the problem of querying outliers does not occur at all in this data set, or our density-weighted representativeness approach is not able to target this problem and either needs different measures or a better methodology.

Since incorporating representativeness has not given us any performance improvements in any of the experiments, we will no longer consider this feature in the rest of our experiments. We cannot confirm our hypothesis that active learning in VAE latent spaces can solve the querying outliers problem, as we have not been able to detect it with this data, but judging from the properties of the latent spaces we also do not reject the hypothesis either. Particularly for data more complex than our small greyscale images, it may be very useful to obtain representative latent spaces. To test whether this is indeed true, more research is needed.

5.3.4 Diversity

So far we have used simple methods to query batches of informative instances. In Section 3.4.3 we described a greedy diversity method that uses a distance measure explicitly to prevent redundancy

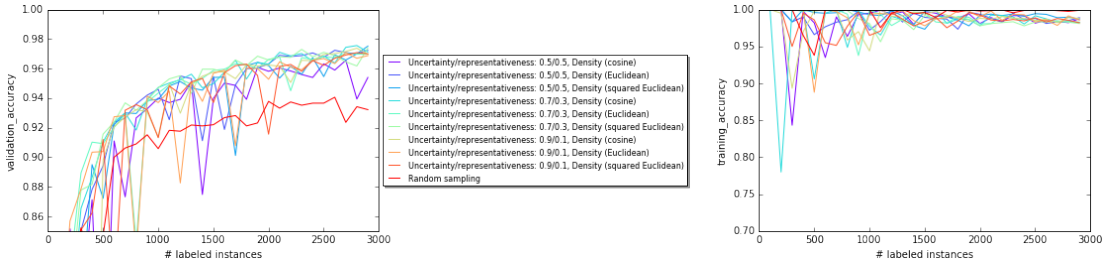


Figure 5.19: Margin sampling with representativeness validation (left) and training (right) accuracies after training on original MNIST image data

within a batch. We evaluate this method for different distance measures, with different ratios of relative importance between uncertainty and diversity. We do not consider the representativeness term in the greedy strategy, since the results of the previous section showed no improvements for incorporating representativeness. Since the greedy method is very slow on the full data sets, we combine it with subsampling of just 1% of the data. The results for MNIST are shown in Figure 5.20.

Figure 5.20: Margin sampling with greedy diversity batch selection and 1% subsampling, validation (left) and training (right) accuracies after training on MNIST

We see that all variants perform quite similar. And, comparing the results with simple uncertainty sampling in Figure 5.9, we see no improvement when using this greedy diversity strategy. This is in fact not very surprising, given the fact that *top k* already performed well for simple uncertainty sampling. Using *top k* as batch query technique tends to introduce the problem of redundancy within batches, but the fact that it performs well here indicates that this problem somehow does not appear in our experiments.

Figure 5.21 shows the results of using the greedy diversity measure for SVHN. Comparing these results to those for simple uncertainty sampling in Figure 5.10, we actually do see a slight improvement for especially the greedy diversity method with Euclidean distance with a 0.5/0.5 uncertainty/diversity ratio, in particular during earlier active learning iterations where less labels are available. With any of the diversity methods, the performance never drops below that for random sampling, something we did observe for simple uncertainty sampling.

Overall we see that redundancy in batches does not appear to be a big problem in our experiments. Nevertheless, we saw small improvements when using a greedy method to encourage diversity for SVHN. All our results so far are obtained from performing a single experiment for each particular setting however. To draw more confident conclusions we need to perform experiments with the same settings more than once.

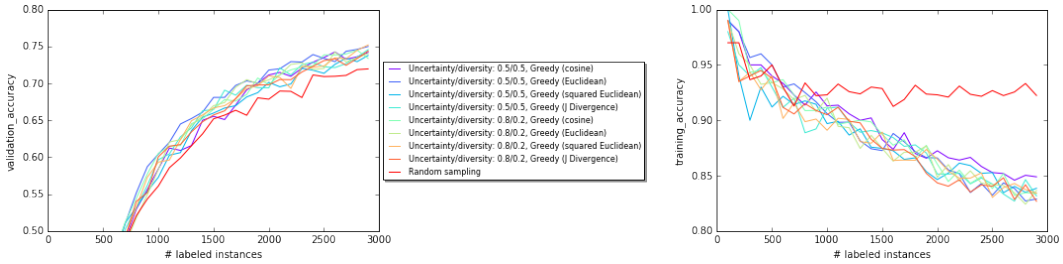


Figure 5.21: Margin sampling with greedy diversity batch selection and 1% subsampling, validation (left) and training (right) accuracies after training on SVHN

5.4 Evaluation of the Best Performing Methods

In the previous sections we tested and evaluated many different settings and approaches. To be able to try out many alternatives, we only ran each experiment once. Since almost all experiments involve random behaviour, we need to run the same experiment multiple times in order to draw stronger conclusions. Therefore, we select the most successful methods in the previous sections, and run each of them 10 times. For each set of $n = 10$ outcomes $\boldsymbol{y} = \{y_1, \ldots, y_n\}$, we compute the *mean* $\overline{y}$, as well as the *corrected sample standard deviation* s. This latter value is computed as follows:

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(y_i - \overline{y})^2}.$$

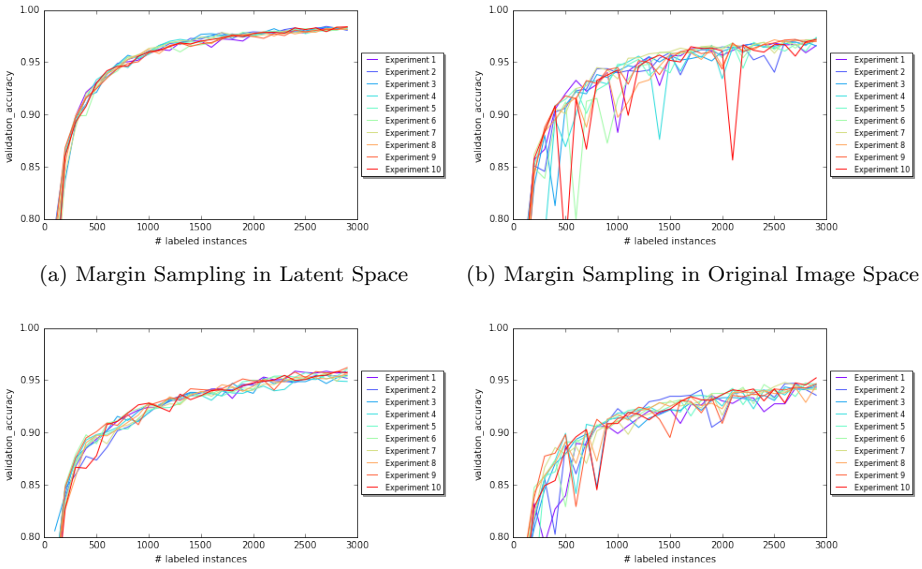
This gives us more reliable results for our various approaches, as well as an indication about how much the result values fluctuated in our experiments. All experiments are done with the chosen VAEs (convolutional, with 30 and 50 latent dimensions for MNIST and SVHN, respectively) and MLP (two hidden layers of 512 hidden units each, with 0.2 dropout). As an uncertainty measure, *margin sampling* is always used.

In the previous experiments we used the validation data to evaluate performance during active learning iterations. Here we will also compare the final test set accuracies of various methods at the end of the active learning procedures, i.e. when 3000 labels have been collected.

5.4.1 Training in Latent Space vs. Original Space

We first compare training on latent data with training on original data, for both active learning with uncertainty sampling as well as passive learning (i.e. random sampling). We look at the validation accuracies of the MLP after each active learning iterations. We only do this for the MNIST data set, since MLPs have been shown to not score well enough on SVHN. Instead of computing means and standard deviations of the experiments, we display all 10 experiments for each variation in a single plot. These can be seen in Figure 5.22.

Besides the fact that both active learning and passive learning performs better in latent space, and that active learning outperforms passive learning, an important observation is that learning in latent space produces much more reliable results. Learning from the original image data sometimes results in big performance drops when actually more labels have been obtained. Learning from latent data on the other hand mostly guarantees that more labels indeed give a higher accuracy, and in the worst case the performance doesn't drop much.



(a) Margin Sampling in Latent Space

(b) Margin Sampling in Original Image Space

(c) Random Sampling in Latent Space

(d) Random Sampling in Original Image Space

Figure 5.22: Validation accuracies for margin sampling (10% subsampling, random partitions) and random sampling on MNIST latent data and original image data

5.4.2 Validation Accuracies of Best Performing Methods

We now look at the validation accuracies for some methods that appeared to perform best in the previous sections. In particular, we consider:

- Margin sampling, with 10% subsampling and random partitions batch selection
- Random sampling
- Margin sampling, with 1% subsampling and a greedy diversity batch selection method with Euclidean distance

We use all these methods while training in latent space. For MNIST, we evaluate the first two methods on the original image data as well.

We run each different experiment 10 times, and compute the means and corrected sample standard deviations for each set of 10 validation accuracies from a particular experiment. Figure 5.23 and Figure 5.24 show these means as lines, with the standard deviations as vertical intervals above and below the means, for MNIST and SVHN, respectively.

For MNIST we see that the best method is simple uncertainty sampling (margin sampling) on the latent data. Using a greedy approach to encourage diversity in the query batches can improve performance a bit at first, but gives slightly worse accuracies when more data is collected. Margin sampling in the original image space still outperforms both the passive learning approaches in latent and original space, but is also less stable than approaches in latent space.

For SVHN we see that we can actually do slightly better than simple margin sampling by querying batches with a greedy diversity approach (with Euclidean distance), although this improvement appears to get less when more labels are collected. Active learning also clearly outperforms passive learning here.

5.4.3 Test Set Accuracies of Best Performing Methods

So far we used the validation data to analyse the performance during active learning iterations. Now we look at the final test set accuracy at the end of the active learning procedures, and

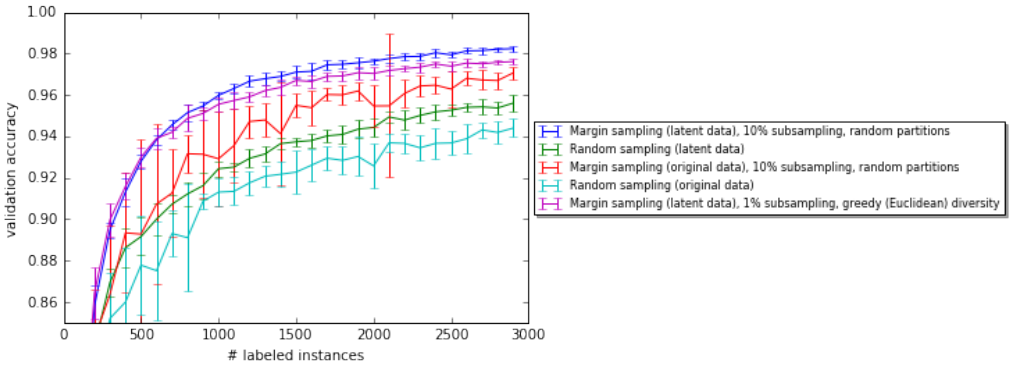


Figure 5.23: Mean validation accuracies with corrected sample standard deviation intervals for several methods on MNIST latent data and original image data

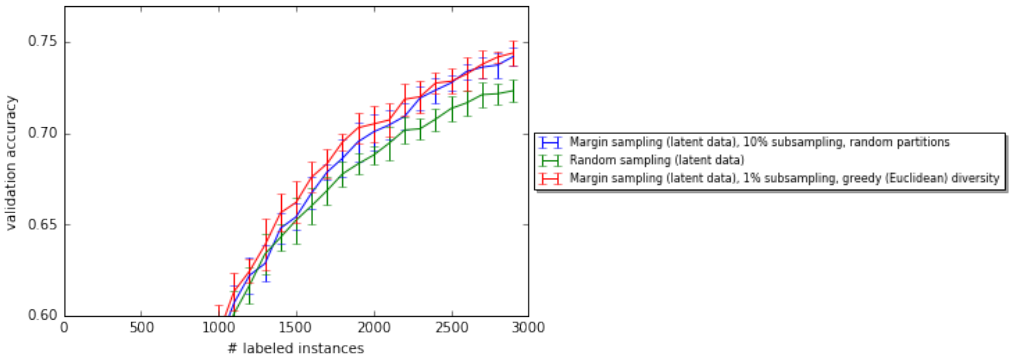


Figure 5.24: Mean validation accuracies with corrected sample standard deviation intervals for several methods on SVHN latent data

compare them with the test set performance of the same MLP as used for the active learning scheme, when trained on the fully labelled data sets. We perform each experiment 10 times, and report the means and corrected sample standard deviations of the test set accuracies. The results can be found in Table 5.4 and Table 5.5, for MNIST and SVHN, respectively.

Experiment	**Data**	**Mean**	**Std. dev.**
MLP on fully labelled latent data	Latent	**0.9868**	0.000548
MLP on fully labelled original image data	Original	**0.9830**	0.001328
Margin sampling, 10% subsampling, random partitions	Latent	**0.9796**	0.000835
Margin sampling, 1% subsampling, greedy (Euclidean)	Latent	**0.9742**	0.002022
Margin sampling, 10% subsampling, random partitions	Original	**0.9697**	0.002782
Random sampling	Latent	**0.9546**	0.004010
Random sampling	Original	**0.9402**	0.004881

Table 5.4: Means and corrected sample standard deviations of test set accuracies for various methods on MNIST

We see that the test set results mostly match what we saw before. It is noteworthy though that for SVHN, simple margin sampling has a slightly higher test set accuracy than margin sampling with a greedy diversity batch selection approach. As we saw before however, these methods perform quite similar, thus this is not very surprising. As before, it is clear that active learning easily outperforms passive learning.

Experiment	**Data**	**Mean**	**Std. dev.**
MLP on fully labelled latent data	Latent	**0.8218**	0.002535
MLP on fully labelled original image data	Original	**0.7109**	0.014790
Margin sampling, 10% subsampling, random partitions	Latent	**0.7079**	0.004421
Margin sampling, 1% subsampling, greedy (Euclidean)	Latent	**0.7067**	0.005649
Random sampling	Latent	**0.6854**	0.009284

Table 5.5: Means and corrected sample standard deviations of test set accuracies for various methods on SVHN

We see that we can get fairly close to the performance of the same model on all 55000 labelled MNIST instances, with only 3000 labelled instances. There is still a small gap in performance though. For SVHN, this gap is much bigger. Training on all 68200 labelled SVHN instances performs significantly better than training on the 3000 actively selected samples. This is likely partially due to the fact that SVHN is more complex; more data is needed to learn all necessary aspects that determine an instance's label. This also makes it harder to learn a useful latent space representation for this data set than it is for MNIST.

Chapter 6

Conclusions

We proposed an active learning scheme in which we first use a variational autoencoder (VAE) to obtain latent space representations of (unlabelled) image data. We then use an uncertainty sampling active learning approach, with a simple multilayer perceptron (MLP) as its classification model (although other classification models are possible too). We discussed a number of common shortcomings for uncertainty sampling, as well as some methods to overcome them.

The more structured and lower-dimensional latent space representations of the higher-dimensional image data provide opportunities for a very fast yet still effective active learning setting. The latent space representation allows us to use a simple, fast-to-train classification model for active learning, making active learning iterations very fast. This results in little downtime for annotators working on labelling queried instances. To further improve the speed of the active learning procedure, we query batches instead of single instances, before retraining the classification model.

Our experimental results on the MNIST and SVHN data sets show that an MLP trained on latent space representations can perform better than when trained on the original pixel representations of the image data. For this we generally do need to use VAEs with convolutional layers for image data, simpler VAEs with only dense layers are often not powerful enough to learn good representations from image data. Training MLPs on the fully labelled data sets (in latent space representation), we achieved acceptable performance that is not state of the art but provides a good basis for an active learning approach.

We showed that actively sampling data to be queried and labelled by means of *margin sampling*, a form of uncertainty sampling, performs significantly better than passively sampling random data instances for labelling. With the labelled data obtained through active learning, we can train models that achieve significantly higher classification accuracy than models trained on the same amount of randomly sampled labelled data.

Another interesting conclusion from our experiments is that models trained on latent data perform much more reliably than the same models trained on the original pixel representations. In particular, if we obtain more labelled data, we expect a model trained on this data to have higher accuracy. For models trained on the original data this is not always the case however; performance may drop significantly at times when more data is obtained, even if on average more data does imply higher accuracy. For models trained on latent data, this hardly occurs; we see fewer fluctuations in accuracy, and more data almost always leads to higher accuracy. This generally already holds for passively sampled (labelled) data, but even more so for data obtained through active learning.

A common weakness of uncertainty sampling is that it tends to query outliers that do not represent the underlying data distribution well. We investigated methods to explicitly model representativeness and prevent the querying of unrepresentative instances in our active learning approach. We did not however achieve any improvements over the regular uncertainty sampling method for the MNIST and SVHN data sets. We expect that learning on VAE latent space

representations already eliminates the need for such methods by transforming the data in such a way that there are no more outliers, but we have also not been able to confirm this theory for these data sets. We conjecture that either the given data sets are simple enough to not contain much unrepresentative data, or that the representativeness approach and measures that we investigated simply do not perform well on image data.

To speed up active learning iterations, we queried batches of informative instances at once before retraining the classification model. A common problem with such methods is that batches may contain redundant instances that would not be queried if we were to collect instances one by one. Therefore, we implemented a few simple techniques based on random samples, as well as a greedy approach, to select batches of informative instances that contain more diversity. The greedy approach is very slow in general, but can be sped up to be reasonable fast when using it only on a very small (randomly sampled) percentage of the data. For the MNIST data set this greedy approach did not lead to an improvement in performance however, but for SVHN we have shown to be able to obtain slightly higher accuracies, in particular when only small amounts of labels are obtained.

The results in this thesis show that there is definitely potential for using latent spaces obtained through VAEs for active learning purposes, both in accuracy and in speed. We have shown an active learning approach where it is only once needed to train a heavy model such as a convolutional VAE on high-dimensional data, after which fast and computationally inexpensive active learning iterations can be executed with simpler classification models. Better techniques for training VAEs are needed to truly be able to approach state of the art classification accuracy with simple classification models (such as MLPs) on VAE latent space representations. This would then likely also lead to improved results for active learning on this latent space, without affecting the speed of the active learning iterations.

Chapter 7

Future Work

The approach and experimental results in this thesis suggest ample opportunities for further research in various directions. In this chapter we lay out at some interesting directions for future work, and briefly discuss how more research could lead to improvements or additions to the approaches and ideas discussed in this thesis.

7.1 Membership Query Synthesis

In Section 2.1.1 we discussed three query scenarios for active learning, and motivated why we focused on *pool-based sampling* in this thesis. Our approach also provides interesting opportunities for the *membership query synthesis* scenario however. This scenario lets an active learner generate its own queries, not needing to select them from an available pool of unlabelled data.

The two main challenges in this scenario are that such queries generally do not follow any underlying natural distribution, and that generated instances may not actually resemble a realistic data point at all (think about unrecognisable shapes for image data). Variational autoencoders may be able to target both these challenges however. Since VAEs are generative models, we can form queries by selecting points in latent space, and using the decoder to generate data instances that can be labelled by an annotator. If the VAE is trained well, this should mostly generate realistic and sensible data instances. Moreover, a VAE explicitly trains its latent space to match some given prior distribution, which could ensure that queries do follow some natural underlying distribution.

A disadvantage of such a scenario is that it is hard to evaluate its performance without needing a lot of manpower, since we cannot simulate active learning by using labelled data if our queries do not come from this labelled data. Instead, we need to actually manually label those instances that the active learner synthesises.

7.2 Improvements in VAE Training

For our approach we used binary cross-entropy as the expected reconstruction error term of the loss function for training VAEs. This error metric has a tendency of producing blurry images [24] [51] however in generative models such as the (decoder of the) VAE. More advanced metrics have been proposed to solve this, such as a metric that uses learned representations obtained with a generative adversarial network (GAN) [29]. By using such improved metrics we may be able to obtain VAE latent spaces that better describe and structure the data, allowing simple classification models (such as MLPs) to achieve higher accuracy on these latent space representations, and active learning schemes to more easily select informative instances that allow for more successful learning.

7.3 Different Classification Models

Latent space representations learned through VAEs allow us to train simple classification models on these representations with good performance, which in turn allows us to perform fast active learning iterations. We used an MLP as classification model, but other simple models may work too, or even better. In particular, such models do not necessarily have to be neural networks, even if our original data is of high dimensions, since the VAE can already provide the power and expressiveness of neural networks by learning good latent space representations. Further research could investigate other methods for classification in latent space, and explore under which conditions such models would work well.

7.4 Representativeness

In this thesis we investigated ways to model the representativeness of data instances in different representations, and whether we could improve the performance of uncertainty sampling by using these representativeness measures to prevent querying outliers. This has not led to any improvements in accuracy though, even though literature [46] suggests that it is common for methods like uncertainty sampling to query less informative outliers. The reason that our representativeness approach did not improve performance could be that the data sets we used do not contain many outliers, or that the representativeness approach or measures simply do not work. More research or experiments on more complex data sets could be done to investigate this further.

7.5 Diversity

We implemented several techniques to encourage diversity within query batches, i.e. to prevent redundancy within a batch. This has however hardly led to improvements in performance. In fact, we saw that a simple *top* k batch selection approach often worked just as well as most of our other approaches (and thus significantly better than random sampling), whereas research [45] suggests that top k selection often performs even worse than random sampling. It is unclear why this has not been the case in our experiments. Further research could show if there are cases in which we do need diversity techniques to prevent redundancy in batches and improve performance for batch-mode active learning. More research could then qualify those cases as well as analyse why we did not observe such cases in our experiments.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. 14, 23, 24

[2] Dana Angluin. Queries and concept learning. *Machine learning*, 2(4):319–342, 1988. 5

[3] Dana Angluin. Queries revisited. In *International Conference on Algorithmic Learning Theory*, pages 12–31. Springer, 2001. 5

[4] Les E Atlas, David A Cohn, Richard E Ladner, Mohamed A El-Sharkawi, Robert J Marks, ME Aggoune, and DC Park. Training connectionist networks with queries and selective sampling. In *NIPS*, pages 566–573, 1989. 6

[5] Eric B Baum and Kenneth Lang. Query learning can work poorly when a human oracle is used. In *International Joint Conference on Neural Networks*, volume 8, page 8, 1992. 5

[6] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013. 1, 9

[7] Saul Berardo, Eloi Favero, and Nelson Neto. Active learning with clustering and unsupervised feature learning. In *Canadian Conference on Artificial Intelligence*, pages 281–290. Springer, 2015. 14

[8] Hervé Bourlard and Yves Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics*, 59(4):291–294, 1988. 10

[9] Klaus Brinker. Incorporating diversity in active learning with support vector machines. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 59–66, 2003. 9

[10] François Chollet. Convolutional variational autoencoder implementation. `https://github.com/fchollet/keras/blob/master/examples/variational_autoencoder_deconv.py`, 2017. 25

[11] François Chollet. Variational autoencoder implementation. `https://github.com/fchollet/keras/blob/master/examples/variational_autoencoder.py`, 2017. 25

[12] François Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015. 14, 23, 25

[13] David A Cohn. Neural network exploration using optimal experiment design. In *Advances in neural information processing systems*, pages 679–686, 1994. 7

[14] Aron Culotta and Andrew McCallum. Reducing labeling effort for structured prediction tasks. In *AAAI*, volume 5, pages 746–751, 2005. 7

[15] Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016. 10

[16] Melanie Ducoffe and Frederic Precioso. Introducing Active Learning for CNN under the light of Variational Inference. 2016. 14

[17] Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural computation*, 4(1):1–58, 1992. 7

[18] Yuhong Guo and Dale Schuurmans. Discriminative batch mode active learning. In *Advances in neural information processing systems*, pages 593–600, 2008. 9

[19] Richard HR Hahnloser, Rahul Sarpeshkar, Misha A Mahowald, Rodney J Douglas, and H Sebastian Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405(6789):947, 2000. 16

[20] Geoffrey E Hinton and Richard S Zemel. Autoencoders, minimum description length and helmholtz free energy. In *Advances in neural information processing systems*, pages 3–10, 1994. 10

[21] Steven CH Hoi, Rong Jin, and Michael R Lyu. Large-scale text categorization by batch mode active learning. In *Proceedings of the 15th international conference on World Wide Web*, pages 633–642. ACM, 2006. 9

[22] Steven CH Hoi, Rong Jin, Jianke Zhu, and Michael R Lyu. Batch mode active learning and its application to medical image classification. In *Proceedings of the 23rd international conference on Machine learning*, pages 417–424. ACM, 2006. 9

[23] Gordon Hughes. On the mean accuracy of statistical pattern recognizers. *IEEE transactions on information theory*, 14(1):55–63, 1968. 1

[24] Ferenc Huszár. How (not) to train your generative model: Scheduled sampling, likelihood, adversary? *arXiv preprint arXiv:1511.05101*, 2015. 12, 51

[25] Seokhwan Kim, Yu Song, Kyungduk Kim, Jeong-Won Cha, and Gary Geunbae Lee. Mmr-based active machine learning for bio named entity recognition. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, pages 69–72. Association for Computational Linguistics, 2006. 9

[26] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013. 9, 11

[27] Christine Körner and Stefan Wrobel. Multi-class ensemble-based active learning. In *ECML*, volume 6, pages 687–694. Springer, 2006. 7

[28] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951. 12, 19

[29] Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, Hugo Larochelle, and Ole Winther. Autoencoding beyond pixels using a learned similarity metric. *arXiv preprint arXiv:1512.09300*, 2015. 12, 51

[30] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 24

[31] Yann LeCun et al. Generalization and network design strategies. *Connectionism in perspective*, pages 143–155, 1989. 11, 16

[32] David D Lewis and William A Gale. A sequential algorithm for training text classifiers. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 3–12. Springer-Verlag New York, Inc., 1994. 6, 7

[33] Jianhua Lin. Divergence measures based on the shannon entropy. *IEEE Transactions on Information theory*, 37(1):145–151, 1991. 19, 20

[34] Peng Liu, Hui Zhang, and Kie B Eom. Active deep learning for classification of hyperspectral images. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 10(2):712–724, 2017. 14

[35] David JC MacKay. Information-based objective functions for active data selection. *Neural computation*, 4(4):590–604, 1992. 7

[36] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, page 5, 2011. 24

[37] ITUR Recommendation. Bt. 601-7,studio encoding parameters of digital television for standard 4: 3 and wide-screen 16: 9 aspect ratios,. Technical report, Tech. Rep., 2011. 54 BIBLIOGRAPHY. 24

[38] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014. 9, 11

[39] Nicholas Roy and Andrew McCallum. Toward optimal active learning through monte carlo estimation of error reduction. *ICML, Williamstown*, pages 441–448, 2001. 7

[40] Tobias Scheffer, Christian Decomain, and Stefan Wrobel. Active hidden markov models for information extraction. In *International Symposium on Intelligent Data Analysis*, pages 309–318. Springer, 2001. 7

[41] Andrew I Schein and Lyle H Ungar. Active learning for logistic regression: an evaluation. *Machine Learning*, 68(3):235–265, 2007. 7

[42] MJ Schervish, N Wermuth, and K Krickeberg. Theory of statistics. 1995. 7

[43] Miriam Schiffman. Introducing variational autoencoders (in prose and code). `http://blog.fastforwardlabs.com/2016/08/12/introducing-variational-autoencoders-in-prose-and.html`, 2016. 10

[44] Burr Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009. 5, 6

[45] Burr Settles. From theories to queries: Active learning in practice. In *Active Learning and Experimental Design workshop In conjunction with AISTATS 2010*, pages 1–18, 2011. 8, 9, 52

[46] Burr Settles and Mark Craven. An analysis of active learning strategies for sequence labeling tasks. In *Proceedings of the conference on empirical methods in natural language processing*, pages 1070–1079. Association for Computational Linguistics, 2008. 7, 8, 9, 52

[47] Burr Settles, Mark Craven, and Soumya Ray. Multiple-instance active learning. In *Advances in neural information processing systems*, pages 1289–1296, 2008. 7

[48] H Sebastian Seung, Manfred Opper, and Haim Sompolinsky. Query by committee. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 287–294. ACM, 1992. 7

[49] CE Shannon. A mathematical theory of communication, bell system technical journal 27: 379-423 and 623–656. *Mathematical Reviews (MathSciNet): MR10, 133e*, 1948. 7

[50] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014. 17

[51] Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. *arXiv preprint arXiv:1511.01844*, 2015. 12, 51

[52] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008. 14

[53] Ying Nian Wu. *Cross Entropy*, pages 154–154. Springer US, Boston, MA, 2014. 8

[54] Zuobing Xu, Ram Akella, and Yi Zhang. Incorporating diversity and density in active learning for relevance feedback. In *ECiR*, volume 7, pages 246–257. Springer, 2007. 9, 20, 21

[55] LE Yann. *Modèles connexionnistes de lapprentissage*. PhD thesis, These de Doctorat, Universite Paris 6, 1987. 10

[56] Shusen Zhou, Qingcai Chen, and Xiaolong Wang. Active deep networks for semi-supervised sentiment classification. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, pages 1515–1523. Association for Computational Linguistics, 2010. 14