University of Copenhagen

Department of Computer Science

Semi-supervised Clustering Techniques for use in End-to-End Processing of Entomological Collections

Masters Thesis

Roberta Hunt

March - September 2019

Supervisor: Kim Steenstrup Pedersen

Roberta Hunt
Semi-supervised Clustering Techniques for use in End-to-End Processing of Entomological Collections
Masters Thesis, March - September 2019
Supervisors: Kim Steenstrup Pedersen

University of Copenhagen
Department of Computer Science
Copenhagen, Denmark

Abstract

The ability to generate high-level groupings of images in an unsupervised or semi-supervised manner has gained importance in recent years, as more unlabelled datasets become available and are deemed of some scientific or societal interest. One such source of data are natural history museums, which continue to digitize their insect collections, and one common starting point is imaging their pinned insect specimens. As more and more images are generated each day, and the total number of specimens is easily in the millions, finding automated methods of analyzing the images and grouping them with minimal labelled data is highly desirable.

In light of this, we experiment with different autoencoder architectures to generate meaningful latent representations of specimens. We begin by developing a methodology using traditional image analysis techniques to segment, scale, translate and normalize images from the newly created Maculinea Alcon pinned-butterfly dataset. We then experiment with various kinds of autoencoders, including denoising autoencoders and variational autoencoders to generate latent representations of the datasets and evaluate how well these architectures can create meaningful and well seperated clusters in the latent space. During this process, we test two relatively new variations on the prior/posterior distribution of the variational autoencoder - Laplacian and Gaussian mixture model, and compare these with the zero-mean Gaussian variational autoencoder. To benchmark our results, we also use the MNIST dataset.

Through this, we show how autoencoders can be successfully applied to unsupervised clustering of pinned insect collections, and show methods to allow variational autoencoders to encourage cluster formation in the latent space.

Acknowledgements

This thesis would not have been possible without the help, advice and support of many others. First, thanks to my supervisor, Kim Steenstrup Pedersen, for patiently explaining many topics to me and directing the thesis to interesting places and questions. Thanks to David Nash and Philip Folman, for sharing their data and interest in this topic with us. Thanks to Martin Stein for sharing his office and thoughts with me during the first months. Thanks to my boyfriend, Esben, for supporting me through some stressful late nights and early mornings. Finally, thanks to all those I studied and discussed with who helped motivate me and shape this work.

Colophon

This thesis was typeset with LaTeX 2_ε. It uses the *Clean Thesis* style developed by Ricardo Langner. The design of the *Clean Thesis* style is inspired by user guide documents from Apple Inc.

Download the *Clean Thesis* style at `http://cleanthesis.der-ric.de/`.

Symbols

$I(p)$	The information contained in a probability density, p
$Tr(\boldsymbol{A})$	The trace of matrix $\boldsymbol{A}$
$\boldsymbol{A'}$	The transpose of matrix $\boldsymbol{A}$
$\boldsymbol{A}^+$	The Moore-Penrose pseudoinverse of matrix $\boldsymbol{A}$
$\boldsymbol{I}$	The identity matrix
$\boldsymbol{\Sigma}$	The covariance matrix of a multivariate distribution
$\hat{\boldsymbol{x}}$	Reconstruction of x produced by an autoencoder
$\boldsymbol{\omega}^{(l)}$	Weight matrix for neural network for layer l
$\boldsymbol{w}$	Weight vector for Gaussian mixture model
$\boldsymbol{x}$	Datapoint or image to be clustered
$\boldsymbol{z}$	Latent variable
$\boldsymbol{a} \bullet \boldsymbol{b}$	Dot product of vectors $\boldsymbol{a}$ and $\boldsymbol{b}$
$\boldsymbol{a} \odot \boldsymbol{b}$	Element-wise product of vectors $\boldsymbol{a}$ and $\boldsymbol{b}$
$\mathrm{diag}(\boldsymbol{u})$	Square matrix where diagonal entries are described by the vector $\boldsymbol{u}$
$\mathcal{N}$	The Gaussian (normal) distribution
μ	The mean of a distribution
σ_i	The standard deviation of a Gaussian distribution in dimension i
σ_i^2	The variance of a Gaussian distribution in dimension i
e	Euler's number, ~ 2.71828
log	Natural logarithm, base e
u	Variable used in pdfs, to avoid confusion with x

Acronyms

AE	Autoencoder
ARI	Adjusted Rand Index
BCE	Binary Cross Entropy
CNN	Convolutional Neural Network
FCN	Fully Connected Network
GMM	Gaussian Mixture Model
KL Divergence	Kullback-Leibler divergence
LReLU	Leaky Rectified Linear Unit
MLP	Multilayer Perceptron
MSE	Mean Squared Error
NHMA	Natural History Museum of Aarhus
NHMD	Natural History Museum of Denmark
PCA	Principal Component Analysis
PDF	Probability Density Function
PMF	Probability Mass Function
ReLU	Rectified Linear Unit
SGD	Stochastic Gradient Descent
VAE	Variational Autoencoder

Contents

List of Figures

List of Tables

1 Introduction

> *This book was written using 100% recycled words.*
>
> — **Terry Pratchett**
> Wyrd Sisters

Given a set of images relating to a topic, such as images of a set of butterflies, how can we learn meaningful underlying connections between the objects in those images with minimal human intervention, for example, how can we group the images of butterflies into different species, or genders? This is a broad question with a wide range of applications in today's plugged-in society. Through this thesis we will explore one small aspect of this question, and try to improve and compare some methods of unsupervised clustering of images. We will then apply this to a recently created dataset, consisting of images of the Maculinea Alcon butterfly[16].

This thesis was infact spurned by the creation of the Maculinea Alcon dataset in 2018, and a desire to use recent image processing techniques to clean up and analyze the dataset. This quickly evolved into a project on deep learning methods to learn underlying structures of the dataset. The project has finally matured into an investigation of the effect of a few deep architectures and loss functions on cluster formation using the benchmark dataset, MNIST, and the exploration of newer variations on loss functions, such as the LiPPy and GiMMPPy variational loss function.

Therefore the product of this thesis is three-fold. It presents:

1. Experimental results of the LiPPy and GiMMPPy variational loss functions (methods of unsupervised clustering which use modified versions of the variational autoencoder (VAE) with a **L**aplacian or **G**aussian **M**ixture **M**odel as the **P**rior and **P**osterior).
2. An exploratory analysis of the new Maculinea Alcon dataset which can be used to compare results with a previous study from 1964 on the same species [25] (We leave most of the actual comparison to the biologists).
3. A workflow for end-to-end semi-automatic processing of images of pinned insects to standardize the images for analyses.

To step through this analysis, we start in chapter 2, Datasets, by introducing the Maculinea Alcon and MNIST datasets that will be used throughout this thesis.

Then in chapter 3, Background, we introduce some background theory about unsupervised learning, probability distributions, kl divergence, clustering, autoencoders and other topics relevant to this thesis. This is intended as a reference, and can be skipped or skimmed depending on the reader's background knowledge.

Next in chapter 4, Metholodogy, we begin by explaining the LiPPy and GiMMPPy loss functions used in our variational autoencoder variations. We then present the image processing steps necessary to bring the Maculinea Alcon dataset into a more usable form.

Then in chapter 5, Results, we step through our experiments on autoencoder architecture and loss function in detail, and evaluate the cluster formation of each model.

Next in chapter **??**, Discussion, we look at an overview of the results from the previous chapter and draw some broader conclusions from these. We also present some of the interesting unanswered questions which came up throughout this thesis, but we were unable to tackle in the given time. This leads us into chapter 7, Future Work, where we list some topics which may be of interest to further research.

Finally, in chapter 8, we summarize our results and tie them back to our original goals.

1.1 Related work

Here we introduce related research which may be of interest to the reader, or can provide further reading on the topic. These are grouped by the topic at hand.

Unsupervised Learning / Clustering

Learning underlying connections between datasets, in the way we do in this thesis, has gained much research traction in recent years due to two main reasons. First, is the increased availability of data as social structures increasingly move to digital forms and more and more of the human experience is digitally documented. Second, is the increased availability of cheap computing power which make complex iterative optimization methods, like deep learning, feasible. Unsupervised learning, even unsupervised clustering, is a growing body of research which has received a lot of attention in recent years. An exhaustive attempt to cover all of the recent advances would be insurmountable and quickly outdated. However, if the reader is interested in a recent broad overview of unsupervised clustering methods, we can recommend [2] or the survey paper [31]. Both provide a good overview. If the reader is specifically interested in generative clustering models, another recent method is using generative adversarial networks, such as that presented in ClusterGAN [32]. Facebook also recently released a paper on their deep-cluster algorithm [7]. Below we provide further reading directly related to our unsupervised cluster method.

VAEs with non-Gaussian Priors used for Clustering

Here we will mention a couple papers which specifically focus on adapting variational autoencoders to create interpretable clustering using GMMs as prior/posterior distribution, as we do (as far as we know, no other work has tried using a Laplacian prior).

Dilokthanakul et al. [11] and Jiang et al. [24] both use a similar methodology to ours, specifically trying to apply a Gaussian mixture model to the latent space to encourage clustering, although Dilokthanakul et al. focuses on using a GMM as posterior only, and Jiang et al. pretrains the model first to initialize the GMM, whereas we complete no pretraining and instead continually update our model's prior using k-means. Jiang et al. also adapt the SGVB method of VAEs to use a GMM instead of approximating the KL divergence using Monte Carlo sampling as we do.

Additionally, earlier in 2019 Antoran et al presented N-VAE [3], an adaptation of the variational autoencoder which focuses on disentangling the latent space to make interpretation and clustering easier. Aside from these, there are a few examples where researchers use GMMs to fit the latent space to, such as [29], with some success, but this is less relevant to our area of interest which aims to have the model create more interpretable clusterings during training.

Pinned insect image processing workflows

We could find no literature directly related to the preprocessing of pinned insect images for clustering analysis. This is not too surprising, since this is a niche topic which does not present new image processing techniques, but rather involves combining different topics from traditional image processing techniques. There are a few articles which present the design of digitization stations for imaging pinned insect collections [23, 35]. But we could not find any papers providing details about any processing done on the images for further analysis. There are, of course, many articles and books which present methods for image processing in general. Digital image processing [20] and skimage's documentation [9] were of particular use.

2 Datasets

> *There was an AI made of dust*
> *Whose poetry gained it man's trust.*
> *If is follows ought,*
> *It'll do what they thought;*
> *In the end we all do what we must.*
>
> **— Universal Paperclips**

In this chapter we introduce the datasets which will be used for our investigation. Tab. 2.1 below summarizes the distribution of the datasets and the rest of the chapter is dedicated to the further introduction of them.

For our dataset of interest, the Maculinea Alcon dataset, we additionally set aside a small test set which we do not use in any experiments until the last moment to further validate our model. For the Maculinear Alcon dataset, the validation and test sets are randomly chosen. For MNIST, we use what is usually considered the MNIST test set, as a validation set in our models (meaning we use it to evaluate early stopping criteria as the model is running).

Dataset	Subset	Image Res.	No. Train	No. Val.	No. Test.
MNIST	0	28×28	5923	980	
MNIST	1	28×28	6742	1135	
MNIST	2	28×28	5958	1032	
MNIST	3	28×28	6131	1010	
MNIST	4	28×28	5842	982	
MNIST	5	28×28	5421	892	
MNIST	6	28×28	5918	958	
MNIST	7	28×28	6265	1028	
MNIST	8	28×28	5851	974	
MNIST	9	28×28	5949	1009	
MNIST	All	28×28	60000	10000	
Maculinea Alcon	Aarhus	1024×768	660	142	142
Maculinea Alcon	Copenhagen	2048×1536	924	198	198
Maculinea Alcon	All	-	1584	340	340

Tab. 2.1.: Summary of dataset distributions.

2.1 MNIST

The MNIST (Modified National Institute of Standards and Technology) dataset is a well known set of handwritten digits which is widely used to judge the effectiveness of supervised and unsupervised algorithms alike[28]. It's popularity is partially due to being large, well-curated, and containing small images which are relatively quick and easy to process. It contains ground truth labels of the digit in each image. See fig. 2.1 for example images and tab. 2.1 for statistics on the dataset.

Fig. 2.1.: Example images from MNIST dataset

2.2 The Maculinea Alcon Dataset

The images included in this dataset are of Maculinea Alcon butterflies from three danish collections - Natural History Museum of Aarhus (NHMA), Natural History Museum of Denmark (NHMD) in Copenhagen and a private collection.

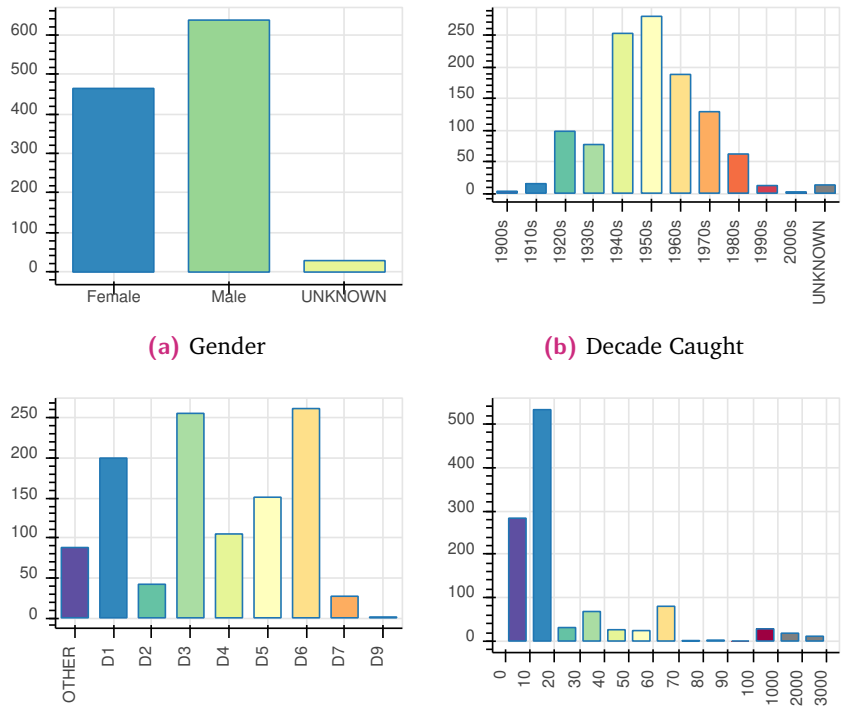
The images were taken by Philip Folman in 2018 as part of his Masters Thesis [16] to compare with the results of a previous study from 1964 (see sec. 2.2.1 or [25]). The original study makes some conclusions about whether or not the Maculinea Alcon are two separate species based on the variation over different danish regions.

The different camera setups used to take the images represent different subsets of this dataset and may be referred to as NHMA and NHMD or Aarhus dataset and Copenhagen dataset (see tab. 2.1). Each sub-dataset was imaged using different cameras, color targets and lighting based on what was available at the time. The setups and major differences are highlighted in sections 2.2.2 and 2.2.3 below. The private collection was imaged using the copenhagen setup.

The dataset contains ground truth data for 1132 butterflies of the Maculinea Alcon species. After removing some images due to issues in preprocessing, it includes 2258 images of the butterflies, half of the dorsal side of the specimen, and half of the ventral side (some of the images were removed due to difficulties during image processing). The dataset includes ground truth labelling per image of: Specimen ID, Genus, Species, Gender, Day/Month/Year of Collection, Locality Collected, Latitude, Longitude, Altitude, Country, Region, Collector, Associated Museum, Date of Registration, Registrant, Kaaber District, and a flag for whether or not the specimen was included in Kaaber's original study, plus some notes which also indicate if the specimen is suspected to be

another species, such as the Maculinea Rebeli species. Some of these labellings were difficult to determine and are labelled UNKNOWN in those cases. For this explorative study, we focus on the labellings for Gender, Year Collected, Country, Kaaber District, and Latitude/Longitude/Altitude. The ground truth labellings are provided in the alcon_data.xlsx excel file attached to this thesis.

Fig. 2.2 shows histograms of the gender, year, altitude and Kaaber district distributions, and tab. 2.2 lists the number of specimens per country. Since Denmark is the area where most of the specimens were caught, fig. 2.3 shows the distribution of specimens across Denmark, and the regions from Kaaber's original paper.



(a) Gender **(b)** Decade Caught

(c) Kaaber Region **(d)** Altitude

Fig. 2.2.: Histograms of data distributions for different ground truth labellings of interest for Maculinea Alcon dataset

	Denmark	France	Poland	Switzerland	Croatia	Germany	Italy	Austria	Russia	Spain	Hungary	Netherlands	Sweden	Unknown
Train	730	17	6	5	3	4	3	3	2	1	1	1		15
Validation	161	1	1	1	1	1				1				2
Test	155	2	1	2	2		2						1	5
Total	1048	20	8	8	6	5	5	3	2	2	1	1	1	22

Tab. 2.2.: Distribution of specimens in Maculinea Alcon dataset by country.

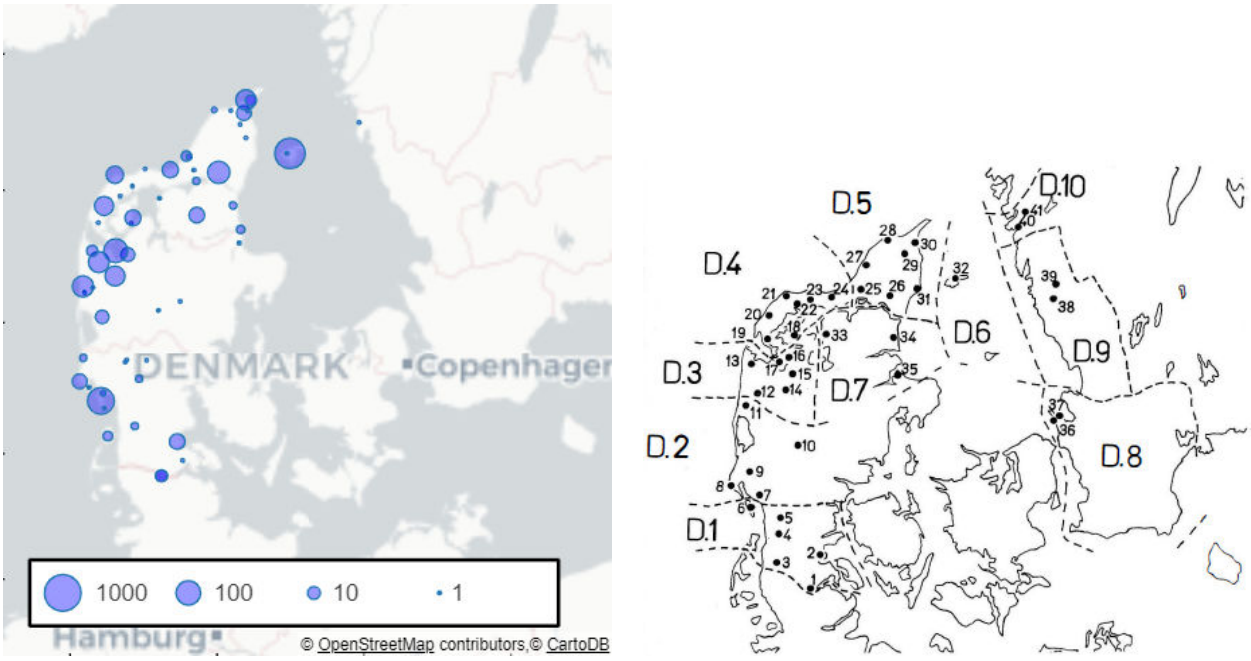


Fig. 2.3.: Map showing Kaaber's defined regions from his original paper (right) and the log-scale distribution of Maculinea Alcon dataset specimens across Denmark (left) where the size indicates the number of specimens caught at that location. There are also some international specimens, but these represent less than 10% of the dataset (see tab. 2.2 for details).

2.2.1 Kaaber's 1964 study

Svend Kaaber completed a study in 1964 [25] focusing on the distribution of Maculinea Alcon in Denmark and attempting to assertain if, as previously proposed, specimens which are typically identified as Maculinea Alcon are actually two distinct species: Maculinea Alcon and Maculinea Rebeli. The discussion was centered on a perceived difference in habitat (with Rebeli being more attracted to dry habitats), pattern variation on the wings, and differences in the shape of the male genitalia. The study included 480 males and 416 females, almost as many as included in our dataset. Kaaber had, however, many more specimens from Sweden than we do. Kaaber's analysis focused heavily on visual features on the dorsal wings, including the presence, size, and shape of spots, as well as amount of blue. Kaaber found no connection between the patterns on the ventral side of the wings and regional variations which might indicate a different species.

He also looked at the shape of the scales on the wings, and variation in the genetalia. For our analysis, we do not think these factors will be learnt by the autoencoder, as the resolution will be too low to identify the individual scales and the genetalia are difficult to distinguish in the images.

2.2.2 Aarhus

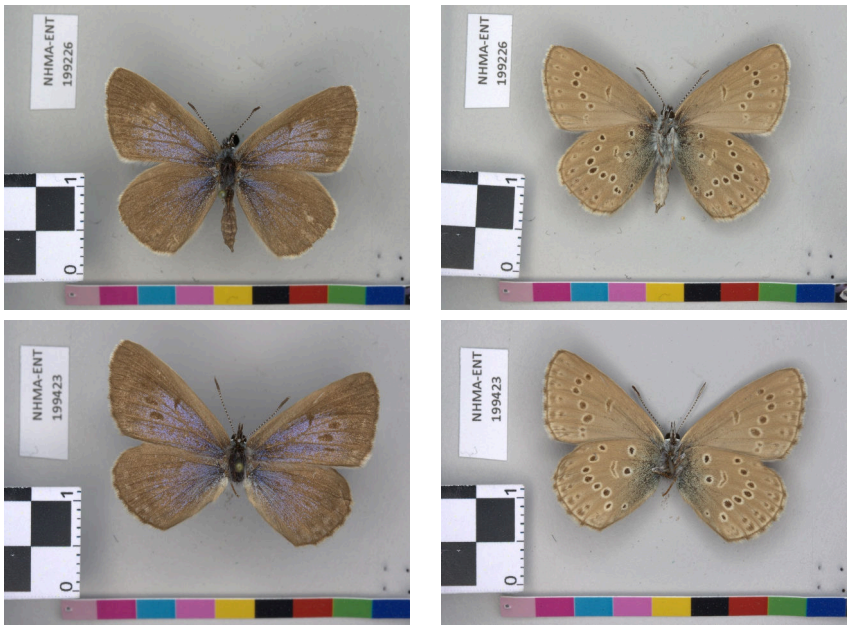


Fig. 2.4.: Example dorsal images (left) and ventral images (right) from Aarhus dataset of Maculinea Alcon dataset.

Fig. 2.4 shows examples of dorsal and ventral images from the Aarhus dataset. The images are mostly standardized, with a color bar below the specimen, a centimeter scale to the left, and the catalog number in the upper left corner. The color bar and centimeter scale are raised up to be aligned with the butterfly's wings - This is however not perfect, and not helped by the fact that the insect itself is not planar. This becomes relevant when we attempt to scale the images later.

The images were taken with a Canon EOS 7D MarkII(G) DS126461 with a 100mm macro lens EF and image stabilizer. The following settings were used: F16, iso 100, raw, WB shift 0, 0"3, HDR disabled.

2.2.3 Copenhagen

Fig. 2.5 shows examples of dorsal and ventral images from the Copenhagen dataset. The images are mostly standardized, with a color bar and scale below the specimen, labels to the left, and the catalog number/qr code in the upper left corner. The color bar and centimeter scale are again raised up to be aligned with the butterfly's wings.

The images were taken with a Sony alpha 7R with a macro lens, on a Kaiser RSD 5602 stand with Kaiser RB 5000DL lights. The following settings were used: F11, iso 200, raw, 1/15 shutter speed, color temperature: 5200. More details about the digitization station can be found in [23].

The copenhagen dataset also includes images from specimens which are not part of NHMD's collections, but are kept by a private collector. They are included with the copenhagen specimens because they were imaged using the same system, so for our purposes they require the same preprocessing, even though they don't have a qr code or catalog number.

A problem we noticed after taking the images in the copenhagen dataset, is some horizontal banding* is present due to the rolling electronic shutter that was used while taking the images. This leads to some variation in the copenhagen dataset, which does not appear in the aarhus dataset. As of yet we have been unable to correct for this banding effect.

Fig. 2.5.: Example dorsal images (left) and ventral images (right) from Copenhagen dataset

*horizontal banding: light-dark horizontal waves present in the image

3 Background

> *Insufficient facts always invite danger.*
>
> — **Spock**
> Star Trek

In this chapter we provide reference for the foundations of the analysis in this thesis. It is assumed that the reader has some basic knowledge of image processing and deep learning, but we will provide a refresher on key topics, and reference to further reading where appropriate.

First, in sec. 3.1 we will broadly introduce unsupervised learning.

Then in secs. 3.2-3.3 we will review some relevant probability distributions and how to compute their kl divergences, as this will be useful for variational autoencoders and our LiPPy and GiMMPPy variants later.

Next in secs. 3.4-3.5 we will provide some details on clustering methods and metrics, as these figure largely into how we assess our models.

Then, in secs 3.6-3.9, we will give a brief introduction to neural networks, convolutional neural networks, autoencoders and variational autoencoders.

Finally in sec. 3.10 we will introduce t-SNE which we use to visualize some of the results.

3.1 Unsupervised Learning

Typically machine learning algorithms are broadly separated into three categories: supervised, semisupervised and unsupervised, where supervision is an analogy for how much 'help' the programmer gives to the algorithm to learn something interesting about the data. Supervised algorithms are given a ground truth output for every input they are trained on. Semi-supervised algorithms are trained mainly with features with no ground truth labelled data, but use a small segment of labelled data to classify or create a model of the unlabelled data, and unsupervised algorithms try to learn something about the structure of the data without being given any ground truth labels.

These three levels of supervision are however, not as distinct as they seem - many algorithms which are designed for unsupervised learning can be adapted for supervised learning tasks and vice versa. This is reenforced by the chain rule of probability, as nicely explained in the deep learning book [21, p. 103], $p(x) = \prod_{i=1}^{n} p(x_i|x_1, ...x_{i-1})$ which shows that the chain rule of probability allows the unsupervised problem of determining $p(x)$ to be decomposed into the product of many supervised problems of finding $p(x|z)$. Similarly, $p(x|z)$ can also be determined using many unsupervised problems from the joint probability: $p(z|x) = \frac{p(x,z)}{\sum_{z'} p(x,z')}$.

Despite the fact that both unsupervised and supervised methods can be applied to the same problems, the broad categories of unsupervised and supervised learning can be helpful in categorizing and distinguishing problems. Anomaly detection, denoising and density estimation (estimating the generative probability density function of a dataset) are all typically examples of unsupervised learning.

Unsupervised learning is also a precursor to semi-supervised learning, which is very useful for analysis of large datasets with limited labelled data. Labelling data can be quite time-consuming, expensive and error prone, which make semi-supervised learning algorithms (and therefore unsupervised algorithms) quite attractive.

Two main uses of unsupervised learning are for clustering data and generative models. Real world applications for these algorithms vary from market segmentation in the advertising industry [15] and [1] to Openai's recently created language model which was trained on millions of webpages and was deemed too good of a model to release to the public as it could be used for malicious purposes [36]. Autoencoders, which figure largely in this thesis, are also a form of unsupervised learning.

3.2 Probability Distributions

In this thesis we will explore the use of different probability distributions as priors and posteriors on the latent space of variational autoencoders, so these distributions are introduced below.

3.2.1 The Gaussian Distribution

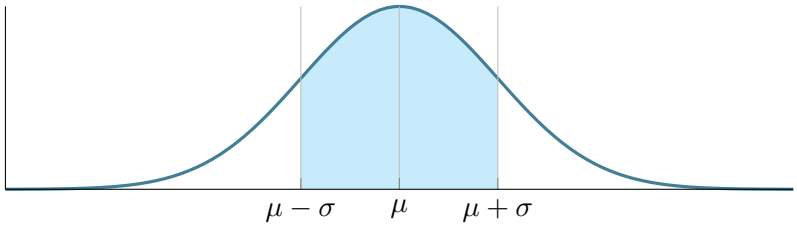


Fig. 3.1.: The univariate gaussian distribution, with mean μ and standard deviation, σ

The gaussian distribution is perhaps the most widely known/used distribution and it can be used to describe many natural processes. The multivariate gaussian can be described by the following probability density function (PDF) (3.1).

$$\mathcal{N}(\boldsymbol{u}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = (2\pi)^{\frac{-d}{2}} det(\boldsymbol{\Sigma})^{-\frac{1}{2}} e^{-\frac{1}{2}(\boldsymbol{u}-\boldsymbol{\mu})'\boldsymbol{\Sigma}^{-1}(\boldsymbol{u}-\boldsymbol{\mu})} \tag{3.1}$$

where $\boldsymbol{\mu}$ is a vector of the means, $\boldsymbol{\Sigma}$ is the covariance matrix, and $\boldsymbol{u} \in \mathbb{R}^d$ is a normally distributed random vector. We use $\boldsymbol{u}$ used instead of $\boldsymbol{x}$ since $\boldsymbol{x}$ is reserved in this thesis for images to be clustered.

An example of a 1D gaussian PDF is shown in fig. 3.1.

3.2.2 Gaussian Mixture Models

Gaussian mixture models (GMMs) are a subset of mixture models in general, which are combinations of probability density functions. A good example of a natural phenomenon that follows a GMM is given in [6] as the old faithful dataset, which was originally released in [5]. It contains waiting times between eruptions and duration of eruptions for the old faithful geyser in yellowstone national park. A plot of this dataset fit to a two component GMM is given in fig. 3.2.

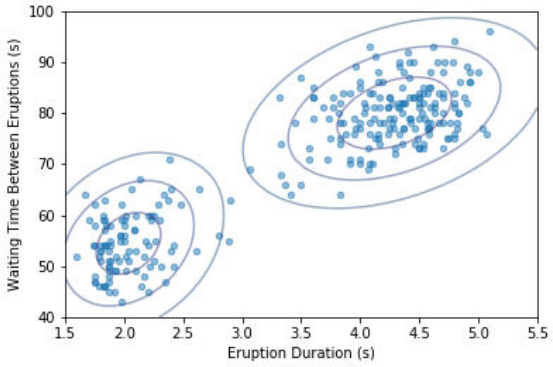


Fig. 3.2.: Plot of old faithful dataset fit to a two-component GMM

Importantly for this thesis, GMMs can describe data with well-separated clusters, such as those in the old faithful dataset shown in fig. 3.2, and due to the prominence of gaussian models, GMMs are well-explored in literature. In particular, even though the kl divergence between two GMMs is generally intractable, many scientific works focus on finding accurate methods of approximating the kl divergence between two GMMs, which will be of importance later.

Multivariate GMM can be described by the PDF:

$$\mathcal{M}(\boldsymbol{u}|w_1 \dots w_{n_g}, \boldsymbol{\mu_1} \dots \boldsymbol{\mu_{n_g}}, \boldsymbol{\Sigma_1} \dots \boldsymbol{\Sigma_{n_g}}) = \sum_{i=1}^{n_g} w_i \mathcal{N}(u|\boldsymbol{\mu}_i, \boldsymbol{\Sigma_i}) \tag{3.2}$$

where n_g is the number of gaussian components included in the GMM and $\sum_{i=1}^{n_g} w_i = 1$ and $0 \leq w_i \leq 1$

3.2.3 The Laplacian Distribution

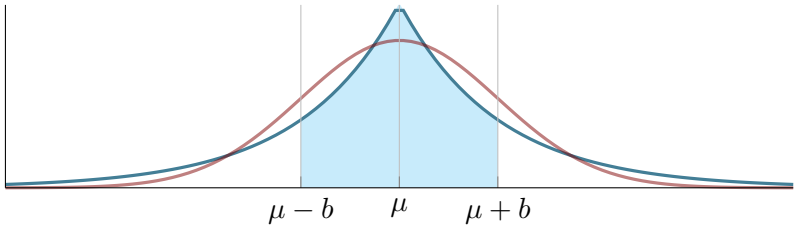


Fig. 3.3.: The Classic 1-D Laplacian distribution (blue) and Gaussian distribution (red). Here you can see the heavier tails of the Laplacian distribution.

The Laplace distribution is a probability distribution with PDF described by (3.3). Importantly for this thesis, it has heavier tails than the Gaussian distribution, meaning it fits datasets which are more spread out which will be important for clustering purposes later.

$$L(u|\mu, b) = \frac{1}{2b} e^{-\frac{|u-\mu|}{b}}, u \in \mathbb{R}, b > 0 \tag{3.3}$$

The multivariate symmetric Laplace distribution is described by the PDF:

$$L(\boldsymbol{u}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{2}{(2\pi)^{\frac{d}{2}} |\boldsymbol{\Sigma}|^{1/2}} \left(\frac{\boldsymbol{u'\Sigma}^{-1}\boldsymbol{u}}{2} \right)^{\frac{v}{2}} \mathbb{K}_v \left(\sqrt{2(\boldsymbol{u'\Sigma}^{-1}\boldsymbol{u})} \right) \tag{3.4}$$

where d is the dimensionality, $v = \frac{2-d}{2}$ and $\mathbb{K}_v$ is a modified Bessel function of the second kind, described by:

$$\mathbb{K}_v(u) = \frac{\pi}{2} \left(\frac{U_{-v}(u) - U_v(u)}{\sin v\pi} \right) \tag{3.5}$$

where $U_v(u)$ is the modified bessel function of the first kind, described by:

$$U_v(u) = \sum_{m=0}^{\infty} \frac{1}{m!\Gamma(m + v + 1)} \left(\frac{u}{2} \right)^{2m+v} \tag{3.6}$$

and $\Gamma(n)$ is the gamma function, described by:

$$\Gamma(n) = \begin{cases} (n-1)!, & \text{when } n \in \mathbb{Z}\text{(positive integers)} \\ \int_0^\infty u^{n-1} e^{-u} du, & \text{when } n \in \text{(complex numbers with a positive real part)} \end{cases} \tag{3.7}$$

3.3 KL Divergence

3.3.1 Mathematical Information Theory

To understand the KL divergence, it is important to briefly introduce some concepts from mathematical information theory, which is a subset of mathematics that deals with the optimal flow and structure of information. For the purpose of this thesis it provides a solid basis for the comparison of probability distributions, and is the foundation for the KL divergence which is explained below. Two important concepts are relevant: information and entropy.

Information, $I(p)$, in nats*, is defined as

$$I(p) = -\log p(u) \tag{3.8}$$

which is the information contained in the probability mass function / probability density function p. For example, if $p = 1$, or the probability of event u occuring is 100%, then the information contained in the occurance of that event is 0.

Entropy (H) is defined as the expected amount of information contained in $p(u)$, defined as

$$H(u) = -\int p(u) \log p(u) du \tag{3.9}$$

3.3.2 Kullback-Leibler Divergence

The Kullback-Leibler (KL) divergence between two (probability) distributions is defined as:

$$\mathrm{KL}(q||p) = \int q(u) \log \frac{q(u)}{p(u)} \tag{3.10}$$

It can be thought of, in information theory terms, as the amount of effort (or extra nats, if we use base e) required to send the information contained in $q(u)$ in the code optimized for $p(u)$.

The KL divergence is widely used as a loss function or regularizer in machine learning applications both because it has a solid theoretical foundation, and because it has the nice property that it is always greater than or equal to 0, and can in many cases be exactly computed. It should be noted however that the KL divergence is not symmetric ($\mathrm{KL}(p||q) \neq \mathrm{KL}(q||p)$).

*One nat is the information contained in an event with probability 1/e.

3.3.3 Cross Entropy

Cross entropy, which is the basis of the categorical cross entropy loss function, is defined for discrete densities as:

$$H(p,q) = -\sum_{u} p(u) \log q(u) du \tag{3.11}$$

where $q(u)$ is the probability density in question and $p(u)$ is the true probability density.

This is quite similar to the KL divergence, and has a direct relationship to it which can be described by (3.12).

$$H(q,p) = H(p) + \mathrm{KL}(p||q) \tag{3.12}$$

It also has an interpretation in information theory: it is the average number of nats required to identify an event drawn from distribution q in a coding optimized for use in p.

3.3.4 KL Divergence between two Gaussian Distributions

The KL Divergence between two multivariate gaussian distributions $\mathcal{N}(\boldsymbol{u}|\boldsymbol{\mu_1}, \boldsymbol{\Sigma_1})$ and $\mathcal{N}(\boldsymbol{u}|\boldsymbol{\mu_2}, \boldsymbol{\Sigma_2})$ has the closed form solution given in (3.13) [12].

$$\mathrm{KL}\left(\mathcal{N}(\boldsymbol{u}|\boldsymbol{\mu_1}, \boldsymbol{\Sigma_1})||\mathcal{N}(\boldsymbol{u}|\boldsymbol{\mu_2}, \boldsymbol{\Sigma_2})\right) = \frac{1}{2}\left[\log\frac{|\boldsymbol{\Sigma_2}|}{|\boldsymbol{\Sigma_1}|} - d + \mathrm{Tr}(\boldsymbol{\Sigma_2}^{-1}\boldsymbol{\Sigma_1}) + (\boldsymbol{\mu_2}-\boldsymbol{\mu_1})'\boldsymbol{\Sigma_2}^{-1}(\boldsymbol{\mu_2}-\boldsymbol{\mu_1})\right] \tag{3.13}$$

This can be simplified further if we make some assumptions about the nature of the Gaussian, as they do in [27]. For example, if we assume $\mathcal{N}(\boldsymbol{u}|\boldsymbol{\mu_2}, \boldsymbol{\Sigma_2}) \sim \mathcal{N}(\boldsymbol{u}|0, \boldsymbol{I})$, and $\mathcal{N}(\boldsymbol{u}|\boldsymbol{\mu_1}, \boldsymbol{\Sigma_1}) \sim \mathcal{N}(\boldsymbol{u}|\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma^2}))$ where $\mathrm{diag}(\boldsymbol{\sigma^2})$ is a diagonal covariance matrix[†], then, (3.13) simplifies to (3.14). This simplification will be useful when we talk about variational autoencoders in sec. 3.9.

$$\mathrm{KL}\left(\mathcal{N}(\boldsymbol{u}|\boldsymbol{\mu}, \boldsymbol{\Sigma})||\mathcal{N}(\boldsymbol{u}|0, \boldsymbol{I})\right) = \frac{1}{2}\sum_{j=1}^{d}\left(-1 + log(\sigma_j^2) + \mu_j^2 + \sigma_j^2\right) \tag{3.14}$$

where d is the number of dimensions, and $\boldsymbol{\sigma^2}$ is the d-dimensional covariance vector.

3.3.5 KL Divergence between two Gaussian Mixture Models

In general, the KL divergence between two Gaussian mixture models is intractable, however, many methods of approximating it have been established over the years. The following section is heavily based on the work of Hershey et al. [22], wherein they compare different approximations of the KL divergence directly. For the purpose of this thesis, we focus on the Monte Carlo Sampling approximation, which the paper regards as the only method of approximating the KL divergence of two GMMs in high dimensional space with arbitrary accuracy.

[†]all off-diagonal entries are 0, and diagonal entries described by the vector $\boldsymbol{\sigma^2}$

KL Divergence Estimation between two GMMs using Monte Carlo Sampling

In this approach, to approximate $\mathrm{KL}(q||p)$, we draw random samples, u_i from the PDF $q(u)$, then the KL divergence can be approximated by:

$$\mathrm{KL}(q||p) \approx \frac{1}{n}\sum_{i=1}^{n} \log \frac{q(u_i)}{p(u_i)} \tag{3.15}$$

In the paper they used Monte Carlo sampling with 1 million samples as their ground truth to compare other models to. They also plotted how accurate MC models would be with fewer samples compared to this ground truth value. This plot is shown in fig. 3.4 below.

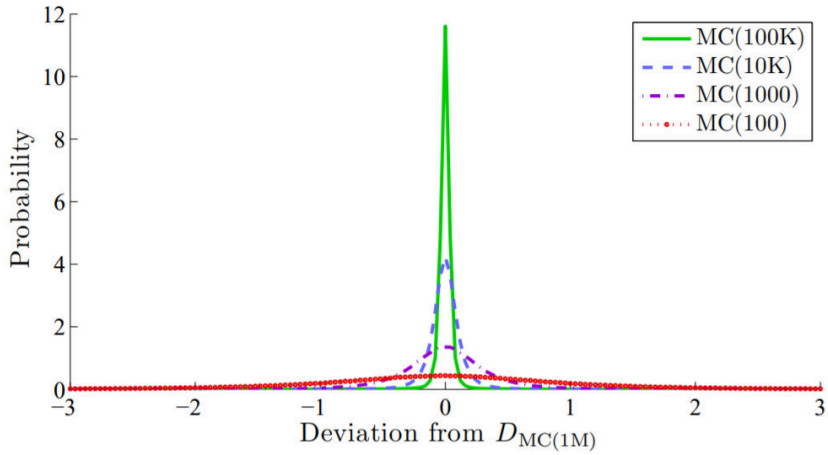


Fig. 3.4.: Taken from [22] with permission. Distribution of deviation of Monte Carlo sampling from 1-million sample estimate. Given a 39-dimensional space, averaged over 9,998 gaussians belonging to 826 different GMMs. The Gaussians all have diagonal covariance. The number of Gaussians per GMM varies between 1 to 76. Given for MC approximations with 100,1000,10000 and 100000 samples

Fig. 3.4 shows that the KL divergence estimation is actually quite good for values of even 1000 samples and upwards. In our GiMMPP model which we introduce later, we use only 100 samples per batch for many of the experiments (as this produces much smaller runtimes), which seems to still provide quite good convergence. This is probably because completing many batches allows the result to average.

3.3.6 KL Divergence between two Laplacian Distributions

The KL divergence between two 1-D classic Laplacian distributions is given by (3.16) [19, p. 36-38]:

$$\mathrm{KL}(L(\boldsymbol{u}|\boldsymbol{\mu_1}, \boldsymbol{b_1})||L(\boldsymbol{u}|\boldsymbol{\mu_2}, \boldsymbol{b_2})) = \log \frac{\boldsymbol{b_2}}{\boldsymbol{b_1}} + \frac{\boldsymbol{b_1}}{\boldsymbol{b_2}} e^{-\frac{|\mu_1 - \mu_2|}{b_1}} + \frac{|\boldsymbol{\mu_1} - \boldsymbol{\mu_2}|}{\boldsymbol{b_2}} - 1 \tag{3.16}$$

3.4 Clustering

Unsupervised clustering aims to take a dataset and gather similar data together such that distinct subsets of the data are self-evident [41, p. 528]. In our case, we would hope that the Maculinea Alcon dataset may automatically create well-separated clusters, clumped by gender or region, and that the MNIST dataset may similarly be clustered by number.

In sections 3.4.1 and 3.4.2 below, we will attempt to distinguish between two topics which are both commonly named cluster analysis or unsupervised clustering. Here we call them 'cluster formation' and 'cluster identification' to try to make the distinction more clear. These concepts are central to this thesis, so we will spend some time explaining them.

1. Cluster Formation - given a pre-defined feature space, use some transformation to create a new feature space from the data in which the clusters of interest are more easily distinguishable
2. Cluster Identification - given a pre-defined feature space, identify/classify distinct clusters within that space.

In this thesis we focus on methods of cluster formation to see if we can generate more easily identifiable clusters in an unsupervised way using deep learning techniques. However, to evaluate the cluster formation, we also need many of the tools used in cluster identification.

3.4.1 Cluster Formation

Cluster formation, particularly unsupervised cluster formation, is a relatively new field (see related work in sec. 1.1), and is used for preprocessing data before cluster identification. Traditionally, this would be done using subject-specific knowledge of the fundamental properties of the dataset to generate a new feature space that is better suited for some cluster identification algorithm. This can typically include: data normalization, outlier detection/ removal and non-linear transformations based on the subject matter. There are several examples of this in literature [44, 39, 18].

3.4.2 Cluster Identification

Cluster identification is a well-studied field which has many applications in a variety of fields from biology to information retrieval, climate, psychology, medicine, and business and can be used for a variety of tasks including classification, anomaly detection and summarization [41, p. 527].

Cluster identification methods can be grouped into hierarchical and partitional [41, p. 527]. A typical example of hierarchical clustering would be clustering species into a dendrogram, where

one species is part of many different clusters, whereas partitional clustering typically has no overlapping clusters.

Cluster identification methods can also be grouped into exclusive clustering (where each datapoint belongs to one cluster exclusively), overlapping clustering (where each datapoint can belong to one or many clusters), and soft/fuzzy clustering methods (where each datapoint is assigned a 'probability'/weight of belonging to each cluster).

In this thesis, for ease of use, we focus on exclusive, partitional methods for cluster classification. Some typical examples of cluster identification algorithms include k-means clustering, hierarchical clustering (divisive and agglomerative), and DBSCAN [41, p. 527]. However, we will limit ourselves to K-means as it is a popular and well established method.

All clustering methods require some form of similarity metic/distance measurement which defines how similar different datapoints are. Similarity metrics are used in clustering to give an idea of how close or far apart different points are in space. The most commonly used similarity metric is Euclidean distance, which is described briefly below. Other similarity metrics might be of interest, such as the Mahalanobis distance, which calculates the probability that a point is related to different distributions. However, for simplicity, we restrict ourselves to the Euclidean distance.

The Euclidean distance between two n-dimensional vectors, $\boldsymbol{u}$ and $\boldsymbol{v}$ is described by:

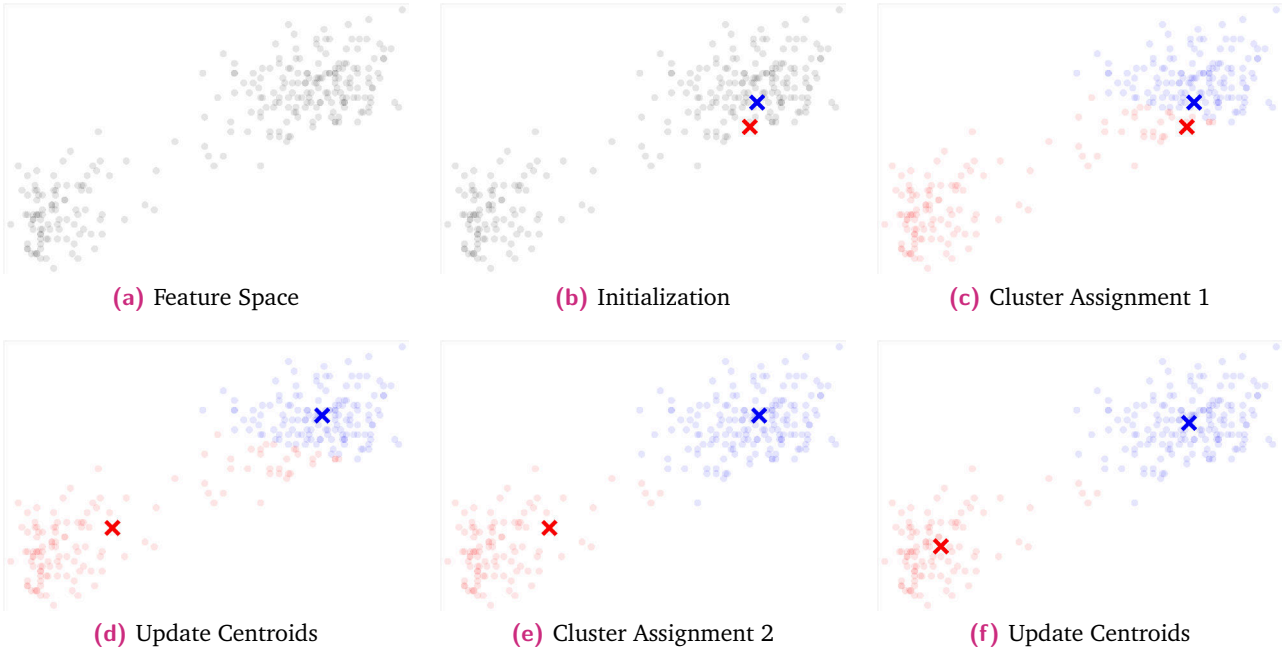
$$\mathcal{E}(\boldsymbol{u}, \boldsymbol{v}) = \sqrt{\sum_{i=1}^{n} (\boldsymbol{u_i} - \boldsymbol{v_i})^2} \tag{3.17}$$

3.4.3 K-means algorithm

K-means is an iterative method of cluster classification. In it, given: Z, a d-dimensional feature matrix, m, the maximum number of iterations, k, the number of desired clusters, and some distance measure (usually Euclidean), the algorithm can be written as follows:

1. Randomly initialize a $k \times d$ dimensional matrix of centroids, somewhere in space (preferably close to, or from, the datapoints in Z)
2. Cluster Assignment: Assign each datapoint from Z to each centroid based on which centroid is 'closest' (defined by the distance measure)
3. Update Centroids: Now that each centroid has some associated datapoints, update each centroid to be the average location of the datapoints
4. Repeat 2-3 until the centroids do not change (the model converges) or the maximum number of iterations is reached.

The K-means iterative process is demonstrated on the old-faithful dataset in fig. 3.5.



(a) Feature Space (b) Initialization (c) Cluster Assignment 1

(d) Update Centroids (e) Cluster Assignment 2 (f) Update Centroids

Fig. 3.5.: Initialization and first 2 iterations of 2-cluster K-means algorithm on old faithful dataset. X's mark the location of the centroid of each cluster over time.

For the purpose of this thesis, however, sklearn's implementation of K-means is used, which includes some additional features to encourage and speed up convergence. First, it by default initializes 10 times and takes the clustering with the lowest inertia (see sec. 3.5.1). Second, it uses K-means++ initialization, which makes it more likely to choose initial points which are far away from eachother, improving the chance of obtaining good clusterings (see [4]). Third, if the dataset is dense it uses the Elkan algorithm instead which is more efficient for dense data (see [14] for details).

3.5 Evaluation Metrics for Clusters

The following section is based heavily on sklearn's reference for cluster evaluation [10]. Below, inertia and the Adjusted Rand Index (ARI) are introduced. Inertia is an unsupervised measure of the quality of a clustering, whereas the ARI is a supervised measure. The ARI is also the main metric to evaluate cluster formation used throughout this thesis. There are many other metrics (such as normalized mutual information, homogenuity and silhouette coefficient), which sklearn provides a good summary of [10], but they are not the focus of this thesis. The main downside of the ARI is that it requires the ground truth labels for the data, thankfully we have this for all the datasets of interest to this thesis.

3.5.1 Inertia

Inertia is a measure commonly used to evaluate the quality of an unsupervised clustering. Inertia is defined as the sum of squared distances between the cluster centroids and their associated points, as in $\mathbb{I} = \sum_{i=0}^{n} m_i r_i^2$, where m_i is the 'mass' of point i (in our case, we can se t this to 1 so all points are weighted equally), and r_i is the distance from the centroid of the group to the point. A higher inertia usually indicates a poorer clustering.

3.5.2 Rand Index

The rand index (RI) is a number to compare the accuracy of a clustering to some ground truth clustering, ignoring permutations.

The RI is calculated by looking at if pairs of points in space are classified together or apart, and how this compares to the ground truth classification.

More concretely, given n points in d dimensions, $p_i \dots p_n$, some ground truth classification, G, and some clustering classification, C, which we want to measure the accuracy of.
a is the number of pairs that are classified the same in G and the same in C
b is the number of pairs that are classified differently in G and differently in C
c is the number of pairs that are classified either differently in G and the same in C or the same in G and differently in C

$$RI = \frac{a+b}{a+b+c} \tag{3.18}$$

where $0 \leq RI \leq 1$

In other words, given some random pair p_i and p_j, the RI is the probability that G and C will agree on whether or not that random pair is part of the same cluster.

3.5.3 Adjusted Rand Index

The Rand Index described above has the problem that random labelling are not ensured to get a value of 0. To counteract this, the adjusted rand index (ARI) was introduced, and is defined as:

$$ARI = \frac{RI - \mathbf{E}(RI)}{max(RI) - \mathbf{E}(RI)} \tag{3.19}$$

where $\mathbf{E}(RI)$ is the expected value of RI specified by a random model, and is traditionally calculated using a contingency table and $ARI \leq 1$. Note that the ARI can be less than 0, unlike the RI.

3.6 Neural Networks

Neural networks are commonly applied to solve some regression or classification problems and are the foundation of deep learning which will figure largely in this thesis, so we will spend some time describing basic concepts in the following section. This section is based heavily on, and more in-depth descriptions can be found in the textbooks [21, 6].

3.6.1 Perceptron

The Perceptron is the fundamental building block of artificial neural networks. Introduced in 1958, it is loosely based on biological neural networks. Perceptrons are commonly represented by graphs like that shown in fig. 3.6.

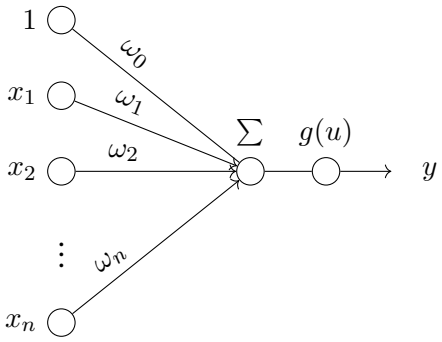


Fig. 3.6.: Diagram of perceptron. $\boldsymbol{x}$ is the input vector, $\boldsymbol{\omega}$ is the weight vector, including a bias term (ω_0). $g(u)$ is the activation function.

They can also be represented by their equation for forward propagation:

$$y(\boldsymbol{x}) = g(\boldsymbol{w} \cdot \boldsymbol{x} + w_0) \tag{3.20}$$

where $\boldsymbol{x}$ is some input vector we want to classify or regress, $\boldsymbol{\omega}$ is a weight vector which is updated via backpropagation (see sec. 3.6.5) ω_0 is commonly thought of as the bias term, and $g(u)$ is the activation function, usually some non-linear transformation (see sec. 3.6.3). This is further simplified if we pre-append a 1 to the x vector, and put the bias term, w_0 in the weight vector, as:

$$y(\boldsymbol{x}) = g(\boldsymbol{w} \cdot \boldsymbol{x}) \tag{3.21}$$

The idea is to then find optimal values of $\boldsymbol{\omega}$ such that we minimize some classification / regression loss. For example, assuming we wanted to create a model to predict housing prices, the vector $\boldsymbol{x}$ may contain various factors that affect housing prices: x_1 may be square meters, x_2, house prices in the surrounding area, x_3, year of construction, etc. y would then be the final price for that house. Now we want to find optimal values of $\boldsymbol{\omega}$ and $g(u)$ such that our prediction is as accurate as possible.

In general, the optimal values may not be directly computable, so instead of exactly deriving a solution, we randomly initialize $\boldsymbol{\omega}$ and use backpropagation and an optimizer, commonly, stochastic gradient descent, to find the gradient of $\boldsymbol{\omega}$ with respect to some loss function, E. We will explore these concepts in more detail in the following sections. First, we abstract the perceptron to many layers in sec. 3.6.2.

3.6.2 Fully Connected Neural Networks

Fully connected feed forward neural networks (FCNs), also called multilayer perceptrons (MLPs) are a further abstraction of the perceptron described in sec. 3.6.1. They involve creating a large network of perceptrons, such as the one shown graphically in fig. 3.7. They are called fully connected because each node is connected to every node in the previous layer and the next layer and so on.

It is widely known that the perceptron cannot learn the XOR function, or indeed many complex functions. The MLP solves this using increased depth to allow it to represent more non-linearities. This is reenforced by the universal approximation theorem which states that neural networks with a single hidden layer of a finite number of neurons can approximate any continuous function in the subset of $\mathbb{R}^n$[6, p. 230].

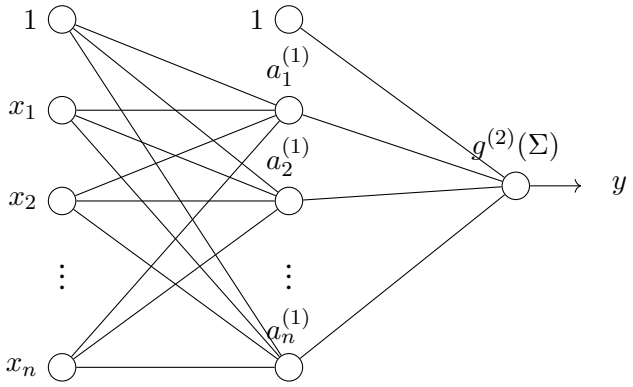


Fig. 3.7.: Diagram of a 1-layer fully connected neural network. To avoid clutter, weight vectors (ω) are not shown. The superscript in $a^{(1)}$ and $g^{(2)}$ is used to indicate which layer/vector we are talking about which is particularly useful for larger networks, or in the equations used to describe this network (see (3.22) and (3.23)).

The feedforward function for the network shown in fig. 3.7 can be described by (3.22) and (3.23):

$$y(\boldsymbol{x}) = g^{(2)}(\boldsymbol{w}^{(2)} \cdot \boldsymbol{a}^{(1)}(\boldsymbol{x})) \tag{3.22}$$

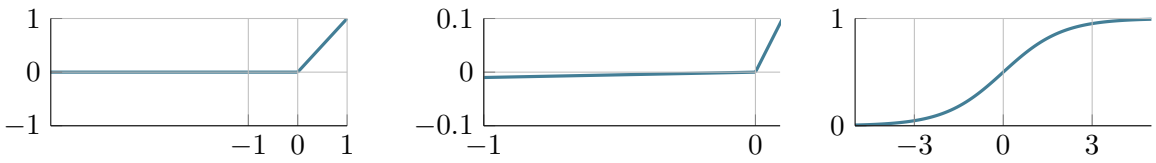
$$\boldsymbol{a}^{(1)}(\boldsymbol{x}) = g^{(1)}(\boldsymbol{w}^{(1)} \cdot \boldsymbol{x}) \tag{3.23}$$

where $g(u)$ is again the activation function.

3.6.3 Activation Functions

Activation functions are likened to the action potential in biological neurons, whereby a neuron needs to reach a certain critical threshold of input from other neurons before it fires. Activation functions in artificial neural networks provide non-linearities, and therefore allow the networks to learn more complex functions.

Below we list some popular activation functions, but this is not exhaustive. Fig. 3.8 compares them graphically, excluding the softmax function which is difficult to visualize.



(a) ReLU (b) Leaky ReLU (c) Sigmoid

Fig. 3.8.: Commonly used activation functions.

ReLU

The rectified linear unit (ReLU) function is very popular due to its simplicity which allows for quick calculations. See fig. 3.8a. It can be described by (3.24).

$$g(u) = \begin{cases} u, & \text{when } u \geq 0 \\ 0, & \text{elsewhere} \end{cases} \tag{3.24}$$

Leaky ReLU

The Leaky Rectified Linear Unit (LReLU) is an adaptation of the relu function that allows some small variance in the negative direction. See fig. 3.8b. It can be described by (3.25).

$$g(u) = \begin{cases} u, & \text{when } u \geq 0 \\ 0.01u, & \text{elsewhere} \end{cases} \tag{3.25}$$

Sigmoid

The sigmoid function is another popular non-linear activation function, but is more computationally expensive than ReLU or LReLU. See fig. 3.8c. It can be described by (3.26).

$$g(u) = \frac{1}{1 + e^{-u}} \tag{3.26}$$

Softmax

Softmax is different from the other activation functions because it takes as input a vector instead of a scalar. It is an activation function commonly used on the last layer of a classifying neural network. This is because it forces a vector to sum to 1, and the values are always positive, so can be thought of as outputting the probability of each classification. It can be described by (3.27).

$$g(u) = \frac{e_i^u}{\sum_{j=1}^{n} e^{u_j}} \tag{3.27}$$

3.6.4 Error Functions

Error functions (also called loss functions, we will use these terms interchangeably in this thesis) are any function we want to minimize. For example, if we are classifying images as containing either dogs or cats, this could be the cross entropy, or if we are predicting housing prices this could be the mean squared error between the actual house price and the prediction.

Almost any function could be used as a error function, however it should have a minimum at our desired solution and it must be differentiable to be used in backpropogation and therefore in typical neural networks.

The Mean Squared Error (MSE) and Binary Cross Entropy (BCE) are two error functions commonly used with autoencoders, so we introduce them below.

Mean Squared Error

The mean squared error loss (MSE) is a common choice of loss function for autoencoders. It can be described by the equation:

$$MSE(\boldsymbol{x}, \hat{\boldsymbol{x}}) = \frac{1}{n} \sum_{i=1}^{n} (\boldsymbol{x_i} - \hat{\boldsymbol{x}}_{\boldsymbol{i}})^2 \tag{3.28}$$

In the case where we use images as the input, n is the number of pixels in each image, $\boldsymbol{x_i}$ is the original pixel value, and $\hat{\boldsymbol{x}}_{\boldsymbol{i}}$ is the reconstructed pixel value.

Categorical Cross Entropy

The categorical cross entropy loss function is an application of cross entropy introduced in sec. 3.3.3. It is commonly used in classification problems, as it gives a measure of the log difference between the predicted class and the actual class, where the actual class is usually denoted using one-hot encoding. Cross entropy as defined in (3.11) gives us some measure of distance between two different probability densities, however, in machine learning we may be interested

in predicting many possible classifications. Also, we may want to make the function symmetric, since we probably find false positives just as important as false negatives. To account for this, we take the sum of all the cross entropies between all categories. This is described by (3.29).

$$\mathrm{CCE}(p, q) = -\sum_{c}\sum_{u} p_c(u) \log q_c(u) \tag{3.29}$$

where c is the category, $p_c(u)$ is the true labelling for that category (usually one-hot encoded, or a probability between 0 and 1),and $q_c(u)$ is the predicted labelling for that category (similarily, representing a probability of belonging to a category, $0 < q_c(u) \leq 1$).

There is a specific case of categorical cross entropy, where there are only two categories. This is called binary cross entropy. In this case we can simplify the equation above to be:

$$\mathrm{BCE}(p, q) = -\sum_{u} p_c(u) \log q_c(u) + (1 - p_c(u)) \log(1 - q_c(u)) \tag{3.30}$$

Although this was initially meant for classification problems, it is commonly used as a loss function for autoencoders which deal with normalized[‡] images as well. This has the nice property that it exponentially weights the deviation from the true image pixel values - it penalizes large individual errors exponentially more compared to MSE.

In our case, we can write (3.30) more explicitly:

$$H(\hat{\boldsymbol{x}}, \boldsymbol{x}) = -\sum_{i} x_i \log \hat{x}_i + (1 - x_i) \log(1 - \hat{x}_i) \tag{3.31}$$

where $\boldsymbol{x}$ is the vector of pixel values in our original image, $\hat{\boldsymbol{x}}$ is the vector of pixel values in our reconstructed image.

3.6.5 Backpropagation

In order to improve the model, we need to calculate how much to update the weight vector, $\boldsymbol{\omega}$. Backpropogation is the process by which we calculate the gradient of the error function with respect to each weight, which we can then use to update the weights using some optimization function (see sec. 3.6.6). For the simple perceptron described in sec. 3.6.1, to do a complete iteration, we first do forward propagation using (3.23) and (3.22), then compute the error function, E, between some ground truth value and our prediction. We can then work backwards through the model and calculate the partial derivative of E with respect to each activation function and weight (see (3.32)).

Using the chain rule, the gradient can be split as follows:

[‡]set to have values between 0 and 1, instead of 0 and 255 as images normally do

$$\frac{\partial E}{\partial \omega_i^{(l)}} = \frac{\partial E}{\partial g}\frac{\partial g}{\partial \omega_i^{(l)}} \tag{3.32}$$

where g is the activation function. Since the functions E and g are differentiable, we are able to compute this easily. This calculation becomes slightly more complex when we move to fully connected networks, but only because it involves more iterations of the chain rule - we apply the chain rule in the same way to each subsequent layer to find the derivative, until we finally reach the first layer. Now, we have a method of calculating the gradient, we need a method to update the weights based on this gradient.

3.6.6 Optimizers

The job of optimizers is to take the gradient, $\frac{\partial E}{\partial \boldsymbol{\omega}^{(l)}}$, found via backpropogation and decide how to update the weights ,$\boldsymbol{\omega}^{(l)}$[§], based on this. The most intuitive way of doing this is stochastic gradient descent, described below. However, many optimizers have been developed over the years, and another popular one, which we will use exclusively in this thesis, is adam, which is also introduced below. A more thorough comparison of different optimization algorithms can be found in [38].

Stochastic Gradient Descent

Stochastic gradient descent (SGD) is an iterative method of finding local minima, and is the building block of many other iterative algorithms. In it, given the gradient of a function at a point, you take a small step in the direction of maximum gradient. This can be described by (3.33).

$$\boldsymbol{u_{j+1}} = \boldsymbol{u_j} - \alpha\frac{\partial E}{\partial \boldsymbol{u_j}} \tag{3.33}$$

where $\boldsymbol{u_j}$ is some point in space, $\frac{\partial E}{\partial \boldsymbol{u_j}}$ is the gradient at that point in space, and alpha is the user defined learning rate that dictates how big the steps will be that the model takes. Too high of a learning rate can lead to divergence, so it is necessary to tune this parameter to find a good value.

The SGD algorithm is commonly likened to descending a hill in the dark - you cannot see the whole landscape around you so you look at the gradient of the land beneath your feet and travel in the direction that seems to be going down. This isn't a perfect mechanism, and is very prone to finding local minima instead of global minima. However, mapping the entire space is computationally prohibitive, so we resort to these kinds of iterative methods.

[§] $\boldsymbol{\omega}^{(l)}$ is the weight vector for layer l

Adam

Adam is an optimizer introduced by Kingma et al. in 2014 [26] designed to combine positive features of the other popular optimizers, AdaGrad and RMSProp. It adjusts the weight update not only based on the learning rate, but also on estimates of the first and second moments of the first order gradients. For brevity, the algorithm is not detailed here, but is well presented in the original paper. This serves to greatly speed up and improve model convergence.

3.6.7 Additional Terminology

Here we introduce some other terminology related to deep learning which will be used throughout this thesis.

Training, Validation, and Test sets: Datasets used in machine learning are typically split up into training, validation and test sets. The train set is what we train the model on and update its weights based on. The validation set is used to compare different models. The test set is set aside until the very end of training and model selection - it is supposed to be considered unseen data, so even if your validation set is biased in some way, you have a second set of data to verify your results on.

Epoch: The amount a model has 'trained' in deep learning is commonly measured in epochs, where one epoch means the model has trained on the entire training set once.

Batch: The datasets used in deep learning models are often extremely large (many gigabytes), and would not entirely fit in most computer's memory. Therefore most models take a random subset of the training data (called a batch), run one cycle of forward propogation and back propogation on the batch at once, and then load the next batch until one epoch is reached. This also allows the model to specialize more, by showing it subsets of the entire training set. There are various advantages of choosing the correct batch size which will not be explored in detail in this thesis.

Overfitting: Overfitting is when the model 'overfits' to the training data, or becomes really good at understanding the training data, but that understanding does not translate well to unseen data (ie the validation set). This is usually undesirable.

Early Stopping: Early stopping is when you put a requirement on the model that if it doesn't improve the error on the validation set after a certain number of epochs, you stop the model, and reload the one that produced the lowest validation set eroor. This is used in all experiments in this thesis.

Overflow/Underflow: Overflow and underflow are problems specific to computational accuracy. Computers normally store numbers with a certain level of precision, and very small or very large

numbers can cause the computer to essentially round the number up to infinity (overflow), or down to zero (underflow). Deep learning typically involves many small numbers very close to zero, or can divide by numbers very close to zero which can lead to overflow and underflow.

Data Augmentation: Deep learning generally benefits greatly from large datasets with some variation which represents all possible data in your field of interest. Usually however, we are limited to a subset of this data. To improve the model's performance on unseen data we can transform our dataset in small ways which make it still a viable part of our dataset, but what the models sees is different. For example, for the MNIST dataset we can add many small $< 5°$ rotations to the images, or increase the size of the digit in the image slightly, effectively increasing our training data, without having to gather new data. This can reduce the chance of overfitting. In the case of the Maculinea Alcon dataset we can additionally horizontally flip all the images, as we know the butterflies should be symmetric. In this thesis we apply small rotations and scalings as well as horizontal flips for the Maculinea Alcon dataset.

The Vanishing Gradient: When activation functions like sigmoid are used which squish space into lower values ((0,1) in the case of sigmoid), this means the gradients can become quite small. If many consecutive layers use these functions, it can result in what is called the vanishing gradient, where earlier layers have a gradient of almost zero, and therefore do not get updated appropriately by the optimizer. In this thesis we focus on using the ReLU function which does not typically have this problem.

3.6.8 Convolutional Layers

Particularily with images, fully connected networks can quickly become quite large. For example, a 1024×768 resolution image (considered small by today's standards), when flattened, requires 786,432 weights per hidden node in the next layer. This quickly creates cumbersome networks. One method to remedy this is to use convolutional layers. An understanding to convolutional layers first requires an understanding of kernel convolutions in traditional image processing. Further reading on image processing in deep learning can be found in [34].

Convolutional Kernels/Filters

Many basic image processing techniques, such as gaussian blurring and edge detection are completed using convolutional filters (also called kernels). In these, given some image, which can be represented by the matrix $\boldsymbol{A}$, we take another matrix, usually square and much smaller, maybe 3×3, we convolve the filter over $\boldsymbol{A}$, to produce a new image, $\boldsymbol{B}$, as shown in fig. 3.9. A very good further introduction to this process and applications is given in Computerphile's video[8].

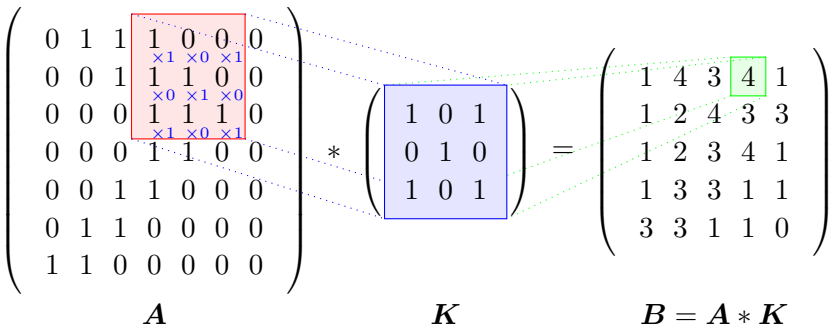


Fig. 3.9.: Diagram of convolutional kernel, $\boldsymbol{K}$, being convolved across $\boldsymbol{A}$ to produce $\boldsymbol{B}$.

Assuming we are using a square filter, of size $f \times f$, and $s = \frac{f-1}{2}$. The we can write the equation for $\boldsymbol{B}$ as in (3.34).

$$\boldsymbol{B}_{x,y} = \sum_{i=-s}^{s} \sum_{j=-s}^{s} \boldsymbol{K}_{i,j} \boldsymbol{A}_{x-i,y-k} \tag{3.34}$$

Applying many different convolutional kernels to an image, we can create many layers of output feature maps, as shown in fig. 3.10.

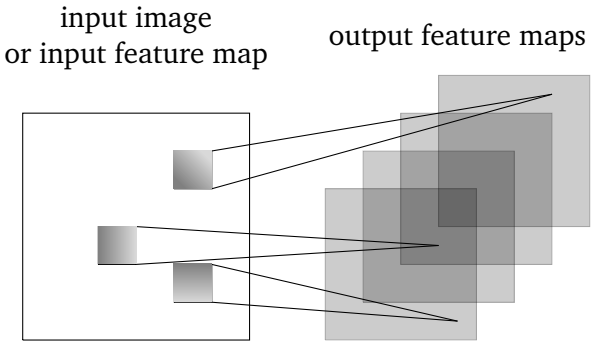


Fig. 3.10.: Diagram of convolutional layer.

Convolutional Neural Networks

Now we can return to convolutional neural networks (CNNs). In Convolutional layers, the idea is that the neural network will automatically learn the kernel values, and in practice this has been shown to effectively learn low-level features, such as those used for edge detection, which can be very helpful for a network to learn from images. Some examples of what these filters might look like are shown in fig. 3.11. Before continuing let's briefly review some other terminology related to CNNs. For more details, see [34, 8].

Stride: Stride defines how big of a step you take when moving the kernel across the image. Using a large stride is one way to reduce the input size across layers.

Padding: Padding defines how many pixels you increase the size of the image by before taking the convolution, and what values you want these pixels to take.

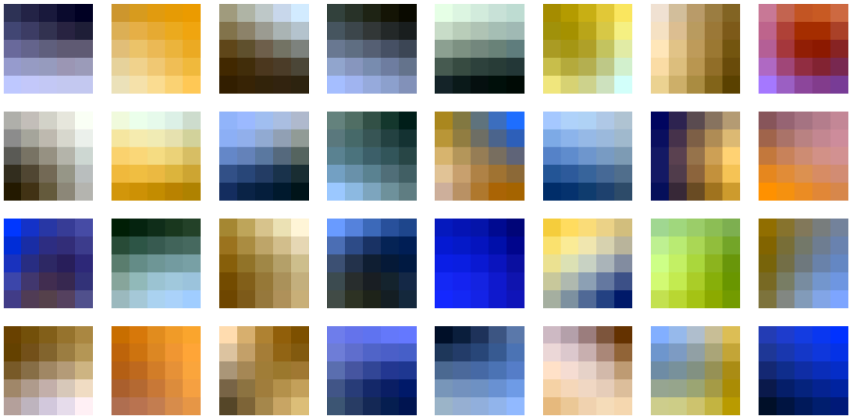


Fig. 3.11.: Example of learned convolutional filters.

3.6.9 Pooling Layers

Pooling layers are another popular method to reduce input size between layers. They are parameterless, and simply apply a function (not a kernel) to a patch of the image, usually a 2×2 patch. Common choices of functions are max (which just takes the largest input value in the patch) and average (which takes the average of the patch). Effectively this serves to reduce the input size by 1/4, and has been shown to still produce good results in image recognition challenges[40].

3.6.10 Unsupervised Clustering with Neural Networks

There are many different kinds of unsupervised clustering methods that can be used in conjunction with neural networks. These include (but aren't limited to) autoencoders, deep belief networks, generative adversarial networks and self-organizing maps. Autoencoders and generative adversarial networks are probably the ones which lend best to image related tasks and gain the most attention in the image processing community at this time [34]. In this thesis we began to focus solely on autoencoders as they seemed to be the least complicated of the two options, and ended up continuing with them. They will therefore be introduced in more detail in sec. 3.7.

3.7 Autoencoders

Autoencoders (AEs) are a subset of neural networks with the prescribed objective to reproduce the original input as well as possible while generating some latent representation. Our hope is that the latent representation learns some underlying fundamental descriptor of the dataset. An autoenoder has two main sections, an encoder and decoder, which are usually the inverse of eachother (at least, structurally) see fig. 3.12. Inbetween the encoder and decoder is the latent representation, z. If the dimensionality of z is less than the dimensionality of the input, as

in fig. 3.12, the network is called undercomplete. If it is larger, it is called overcomplete. The trouble with overcomplete networks is they can simply learn the identity function instead of some interesting properties of the data itself, so usually overcomplete autoencoders require some regularization on the latent space to prevent this from happening.

For example, we might have a dataset of images of 28x28 pixel handwritten digits (MNIST) which we want to learn something about. Let's say the images only contain the digits 1 and 7 (subset of MNIST), and we want to reduce the dimensionality of this dataset to two variables (maybe to save space), while capturing as much data from the dataset as possible.

A simplistic AE could do this with the following neural network structure:

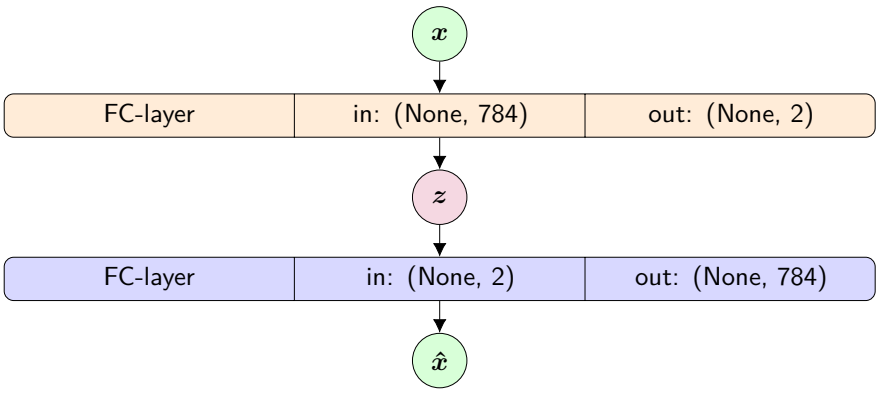


Fig. 3.12.: Simple AE, where FC is short for Fully-Connected, x is a (batch size $\times$ 784) sized matrix of all the pixel values for each image in the batch. The orange layer in this case is called the encoder, z is the latent space, which we are particularily interested in, the blue layer here is the decoder, and $\hat{x}$ is the reconstruction of x (sized (batch size $\times$ 784)). This kind of top-down representation, where the input and output of each layer are listed instead of shown is useful when layers become large and for more complex layers like convolutional layers, and so this representation of neural networks will be used from now on in this report.

Then we can run this neural network, but instead of a normal supervised loss function which would use some ground truth classification, it is common to use some distance measure between the original image ($\boldsymbol{x}$) and the reconstruction ($\hat{\boldsymbol{x}}$). A common choice is the mean squared error, described in sec. 3.6.4.

3.8 Denoising Autoencoders

Denoising autoencoders were introduced by Vincent et al. [42]. The general principal is to add noise to the original images, but have the model try to reproduce the original (to remove the noise). Vincent et al. showed that autoencoders are able to do this quite effectively. There are different kinds of ways to add noise to images, and we use two in this thesis: Gaussian noise and masking noise.

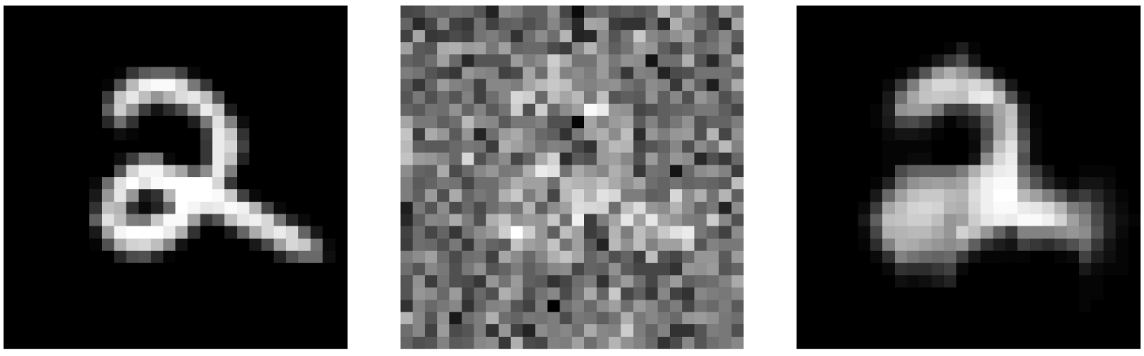
Gaussian noise

Adding Gaussian noise means means sampling from a Gaussian distribution with some standard deviation, σ, and usually zero mean, and adding the sample to a pixel, then repeating this process for every pixel. How noisy the image gets then depends on how high you set the standard deviation.

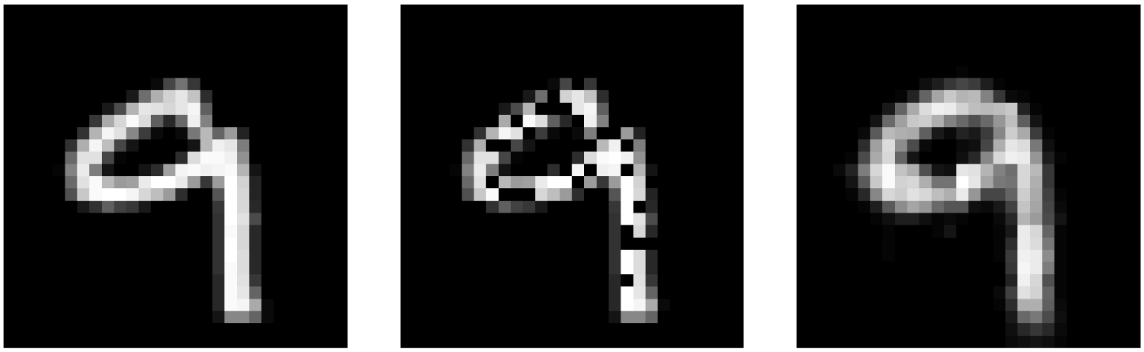
Masking noise

Adding masking noise means masking a subset of the pixels of an image, or essentially setting them to be black. How much masking noise is added is determined by some probability, p, of each pixel being masked.

Example inputs/outputs of denoising can be found in fig. 3.13.



(a) Original Image (b) Gaussian Noise Added, $\sigma = 0.5$ (c) Reconstruction



(d) Original Image (e) Masking noise added, $p = 0.5$ (f) Reconstruction

Fig. 3.13.: Examples of denoising process. Original images (left), Noise-added images (middle), Reconstructions (right). Top is with Gaussian noise added, bottom is with masking noise added

3.9 Variational Autoencoders

Variational autoencoders (VAEs) were introduced by Kingma et al. in 2013 [27]. In brief, they are a form of autoencoder that stems from following the assumption that the data you are interested in (x) is generated by some prior probability distribution ($p_\theta(z)$) of an unobserved continuous random variable z (often referred to as the latent variable), and that if we can make some assumptions about the nature of that distribution, and the nature of the distribution used to generate x ($p_\theta(x|z)$, or the decoder), we can use the KL divergence between our posterior, $q_\phi(z|x)$ (an approximation of $p_\theta(z|x)$) and the actual distribution of the latent space which generates the images, (also called prior), $p(z)$ to construct our loss function.

To do this, the authors of the paper use variational inference and introduce another tractable model (encoder) $q_\phi(z|x)$, meant to approximate the true but (possibly) intractable posterior $p_\theta(z|x)$, and then attempt to minimize the KL divergence between $p_\theta(z|x)$ and $q_\phi(z|x)$ ($\mathrm{KL}(q||p)$).

The following explanation is based heavily on [27] and Ali Ghodsi's lecture video on variational autoencoders [17].

Therefore, we want to minimize this function:

$$\mathrm{KL}(q(z|x)||p(z|x)) = -\int q(z|x) \log \frac{p(z|x)}{q(z|x)} \tag{3.35}$$

with some clever rearranging, we can modify this to be something we can work with. To start, $p(z|x)$ can be replaced by $\frac{p(x,z)}{p(x)}$, making it:

$$\mathrm{KL}(q(z|x)||p(z|x)) = -\int q(z|x) \log \frac{p(x,z)}{p(x)q(z|x)} dz$$

doing some further rearranging,

$$= -\int q(z|x)[\log \frac{p(x,z)}{q(z|x)} - \log p(x)]dz$$

$$= -\int q(z|x) \log \frac{p(x,z)}{q(z|x)} dz + \int q(z|x) \log p(x) dz$$

$$= -\int q(z|x) \log \frac{p(x,z)}{q(z|x)} dz + \log p(x) \int q(z|x) dz$$

Since $\int q(z|x)dz$ is just 1, this can be written as:

$$= -\int q(z|x) \log \frac{p(x,z)}{q(z|x)} dz + \log p(x)$$

Rearranging this we get the following:

$$\log p(x) = \mathrm{KL}(q(z|x)||p(z|x)) + \int q(z|x) \log \frac{p(x,z)}{q(z|x)} dz$$

Since $logp(x)$ is fixed, $L = \int q(z|x) \log \frac{p(x,z)}{q(z|x)} dz$ is considered a lower bound on $\mathrm{KL}(q(z|x)||p(z|x))$. Therefore, we can minimize the KL divergence by maximizing L, or maximizing the lower bound. Then, looking only at the lower bound:

$$L = \int q(z|x) \log \frac{p(x,z)}{q(z|x)} dz$$

we can replace $p(x,z)$ with $p(x|z)p(z)$

$$= \int q(z|x) \log \frac{p(x|z)p(z)}{q(z|x)} dz$$

$$= \int q(z|x) \log p(x|z) + \int q(z|x) \log \frac{p(z)}{q(z|x)} dz$$

$$L = \mathbb{E}_{q(z|x)} p(x|z) - \mathrm{KL}(q(z|x)||p(z)) \tag{3.36}$$

which is commonly thought of as the expected reconstruction error $(\mathbb{E}_{q(z|x)} p(x|z))$ minus the kl divergence between the posterior and the prior on the latent space.

Or, as is sometimes convenient, this can be rearranged again as:

$$L = \mathbb{E}_{q_\phi(z|x)}[-logq_\phi(z|x) + logp_\theta(x,z)] \tag{3.37}$$

Here we note that $q(z|x)$ is the encoder or posterior, $p(x|z)$ is the decoder. $p(z)$ is the prior.

Unfortunately, calculating the expectation of $p(x|z)$ with respect to $q(z|x)$(as in (3.36)) can still be problematic. The original paper presents two ways of approximating the lower bound instead: The Stochastic Gradient Variational Bayes (SGVB) method and the AutoEncoding Variational Bayes (AEVB) method, described briefly below. AEVB is most commonly used as it typically has less variance than the generic SGVB estimator [27].

Stochastic Gradient Variational Bayes (SGVB) algorithm

This method is based fully on estimating the lower bound using Monte Carlo sampling to solve the expectations described in (3.37). The estimator is then:

$$\tilde{L}^A(\theta, \phi; x) = \frac{1}{L} \sum_{l=1}^{L} \log p_\theta(x, z^l) - \log q_\phi(z^l|x) \tag{3.38}$$

Where L is the number of samples used in the minibatch.

Autoencoding Variational Bayes (AEVB) algorithm

The other method described in the original paper is the AEVB algorithm which approximates (3.36) in the case that you can solve the KL divergence exactly, as is the case if you assume the prior and posterior are Gaussian in nature. Then sampling is only needed to solve the reconstruction term, as in (3.39).

$$\tilde{L}^B(\theta, \phi; x^{)} = KL(q_\phi(z|x)||p_\theta(z)) + \frac{1}{L}\sum_{l=1}^{L} logp_\theta(x|z^l) \tag{3.39}$$

The Reparametization Trick

Both the SGVB and AEVB algorithms require sampling from a distribution to approximate the lower bound. Unfortunately, such stochasticities can cause problems for backpropagation. Therefore, the authors proposed a so called 'reparametization trick', wherein instead of having the encoder, $q(z|x)$ be itself stochastic, we use the encoder to predict the parameters (ie mean and variance) of $q(z|x)$, and then draw L samples from that distribution. This also has the added benefit of making the manifold / latent space more traversable.

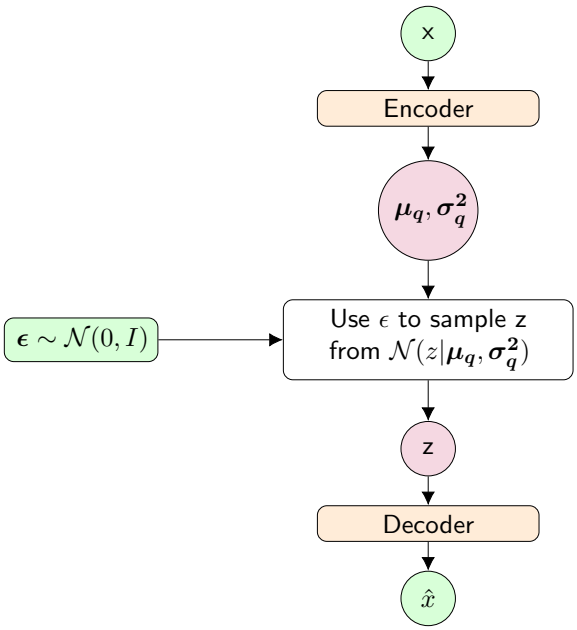


Fig. 3.14.: VAE with GMM Prior and Posterior, basic architecture

Choice of prior and posterior

The original paper specifically avoids making any assumptions as to the nature of the prior ($p(z)$) or posterior ($q(z|x)$), and theoretically VAEs can assume any probability distribution as their prior or posterior. However, most commonly the prior is chosen to be isotropic gaussian with 0 mean

and covariance matrix $\Sigma \equiv I$ as presented as an example at the end of the paper. This was an example given in the original paper, and is convenient both because the gaussian function is a function which appears naturally in random systems, and so it seems natural that many generative processes would follow a gaussian distribution, and also beause it is possible to exactly calculate the kl divergence of two multivariate gaussians, which is very useful in practical deep learning applications. This means we can use the more accurate AEVB algorithm.

In this thesis we present an alternative choice of posterior and prior, assuming a laplacian or a gaussian mixture model.

3.10 T-SNE for Visualization

T-SNE is a non-linear iterative process of dimensionality reduction, introduced by Laurens et al. It is usually not used directly for dimensionality reduction, but only for visualizing high dimensional data. The full process is described in detail in the original paper [30], with a nice interactive example provided by Wattenberg et al [43]. In brief, the process involves squashing the taking each point and 'pulling' close points

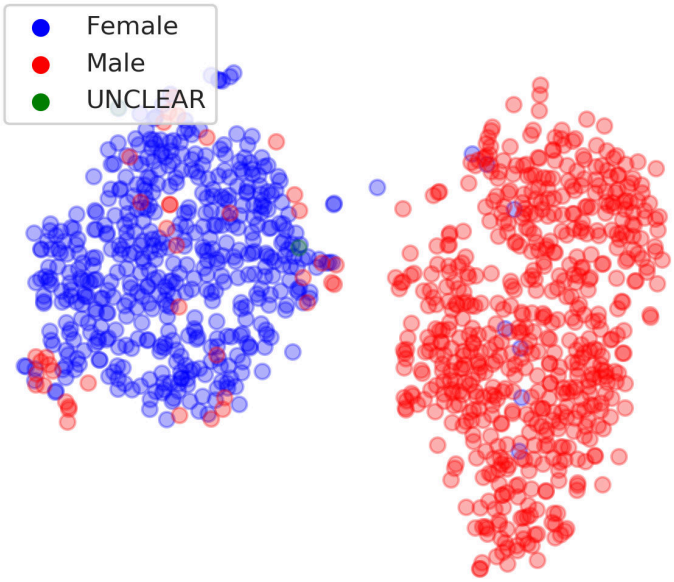


Fig. 3.15.: Example of 2D t-SNE visualization from later results. This is on the Maculinea Alcon dataset, and although the space here is 128-dimensional, we can quickly see that the space is well clustered by gender.

4 Methodology

> *"You've been hallucinating on cactus juice all day! And then you just lick something you find stuck to the wall of a cave?"*
> *"I have a natural curiosity."*
>
> **— Katara and Sokka**
> Avatar the Last Airbender

In this chapter we build on the background knowledge previously presented by justifying how we chose to apply the theories and methods therein to the unsupervised clustering problem stated in the introduction, and we go on to describe how we arrived at our LiPPy and GiMMPPy loss functions.

To start, let's once again reiterate the objective; We want to find a robust method to cluster image-based datasets with the ultimate application being to uncover underlying structures in the dataset and improving semi-supervised and supervised classification tasks.

More specifically, we have a new dataset of some biological interest, the Maculinea Alcon dataset, which we would like to explore. From speaking with the biologists who collected the images we already know that gender is highly correlated with visual features on the wings. Kaaber's study [25] also suggests visual aspects may vary over region. So we would like to find underlying patterns in the dataset and briefly compare these with the results obtained by Kaaber and others.

However, before we can even begin to work with this dataset, we realize it is not very wieldly in its present form - the images have a lot of other objects (besides the specimen) in them, unrelated to what we want our model to learn, such as labels and color bars. So we first preprocess the data to segment, register and scale the butterflies. The steps involved in this process are described later in this chapter in sec. 4.3. Since we could find no research detailing how others preprocess images of pinned insect specimens for use in analysis, and since this will most likely become increasingly important in coming years as museums continue to digitize their collections, we explain the preprocessing steps in great detail, in case this can be of use to others.

Once we have the images in a usable form we return to the question of how to learn interesting latent representations of them. We, as stated, decide to approach this as an unsupervised problem, where we want the model to learn inherent structures in the dataset without us specifying our area of interest (ie gender, region, or year). Then we can explore the latent representations

generated by that dataset and look for clusters and correlations that may indicate visual features are connected to different high level features of the dataset. Ideally, we would also like to model to be able to tell us what visual features it used - this is typically problematic in deep learning, however, we will later explore variational architectures that allow us to generate examples of each cluster's latent representations, which we can then use to indicate which features are distinct to each cluster.

Choice of Unsupervised Method

There are many unsupervised methods which involve neural networks which could be applied to this problem (see sec. 3.6.10). However, at this point we decide to focus soley on **autoencoders**, mainly because of their widespread usage and versatility.

Choice of Architecture

While we suspect that we may want some combination of convolutional neural networks and fully connected networks, as many implementations of autoencoders used in image processing include, there is no obvious choice of architecture for this application. Therefore we need to conduct a series of experiments to determine which architecture to use. We then note that it might be useful to test which architectures lend well to generating latent space representations by using a benchmark dataset. This importantly allows our results to be compared to other works, and ensures that we have a large well-curated dataset with few mistakes to reliably compare models. We choose **MNIST as a benchmark**, mainly because it is easily available, the images are small and therefore the models are quick to run, and it has discrete classifications which we can use in unsupervised clustering.

For these tests we choose to initially use the **mean squared error (MSE) loss** function, which is a common choice for autoencoders. We use this to explore different architectures and then will use our chosen architecture to explore the effects of different loss functions and some regularizations. We could have, for example chosen binary cross entropy, another popular choice, but as we will see later, this has little effect on the results we are interested in. L1 or L3 loss are other possibilities which could produce interesting results, but we restrict outselves to MSE in the interests of time.

We then, as shown in the results section, run many experiments with different numbers of fully connected and convolutional layers, and evaluate the results using the cluster identification evaluation metric: the **adjusted rand index (ARI)** to arrive at a final architecture for use with MNIST. We could choose other indices, such as the adjusted normalized mutual information, the ARI is just fairly simple to understand and we see no major advantages of other functions.

To start the process of finding the best architecture we begin with a simple 1-layer network, then test adding hidden layers to it to test the effect of depth on the clustering. We then fix the

fully connected depth and test the effect of adding different convolutional layers and parameter reduction methods. These experiments are of course not exhaustive, but in the interests of time, this gives us some idea of how the architecture affects clustering so we can begin to apply this to our problem of interest.

To begin the tests however, we need to first choose some hyperparameters for the model, these, along with a justification of their choice are listed below:

- **Batch Size**: A batch size of **200** is chosen for the MNIST dataset and **20** for the Maculinea Alcon dataset. This choice is fairly arbitrary, and based on the fact that there are, for most of the MNIST experiments there are 18,649 training examples, so the batch is not too large compared to the dataset, and because the images are small and the networks small, easily fit on the GPU used. The Maculinea Alcon dataset on the other hand has only 1,623 training examples, so we choose a smaller batch size to allow the model to learn more nuances of the dataset.
- **Early Stopping**: Early stopping is implemented for all the models, although the exact number of epochs varies between **10-20** based on intuition after running some experiments and looking at the loss plots.
- **Maximum Epochs**: All models are run for a maximum number of **500 epochs**, but due to early stopping, most never reach this number.
- **Adam Optimizer**: The adam optimizer is used for all experiments. This choice is somewhat arbitrary, however, the optimizer provides quicker convergence than stochastic gradient descent, and is still quite stable.
- **Learning Rate**: The learning rate is adjusted based on trial and error over the experiments, balancing relatively quick convergence with discouraging divergence. For normal models this is generally around **0.001**, for variational models it is closer to **0.0001**.
- **Data Augmentation**: Data augmentation is applied for all the experiments. For MNIST this is random rotations between -1 and 1 degrees and scaling of $\pm$ 10%. For the butterfly dataset this is the same, with the addition of random horizontal flips, as theoretically any horizontal asymmetry of the butterfly is due to random mutations, or random degradation of the specimen, and not particularily linked with any features we are interested in (gender, location, year).
- **Activation Function**: **ReLU** is used as the activation function for all experiments. This is chosen because it provides non-linearity with low computational cost.

After all completing the experiments described above, we arrive at the architectures shown in figures 4.1 and 4.2 for the MNIST and Maculinea Alcon datasets respectively.

Further justification for the choice of architecture is presented in sections 5.1 and 5.3 where we show the results from our architectural experiments. The choice of each round of experiments is largely based on the results of the previous, so the experiments are not justified in-depth here, see instead the results chapter.

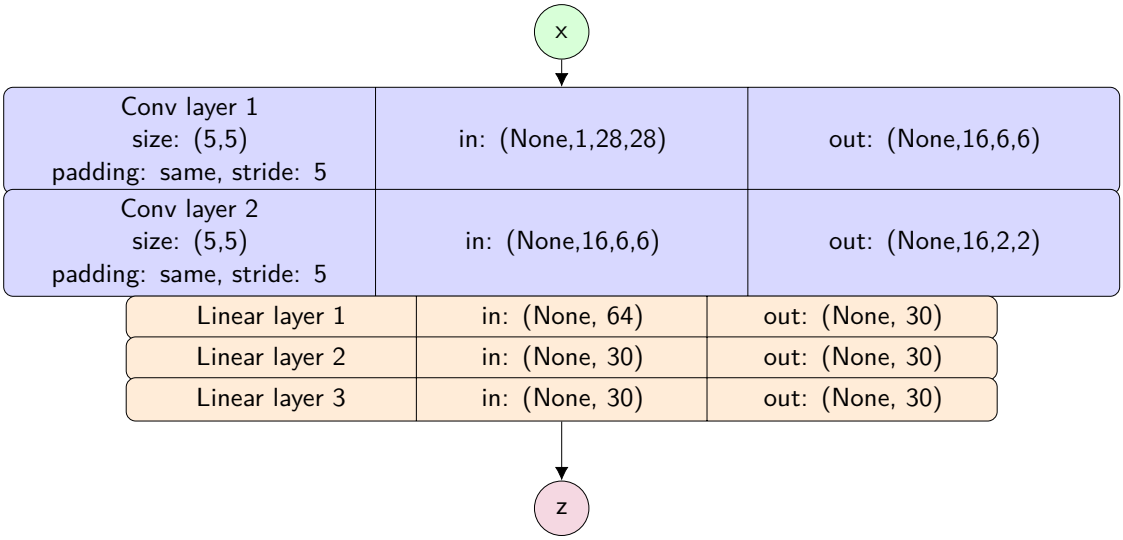


Fig. 4.1.: Final encoder architecture used for MNIST clustering.

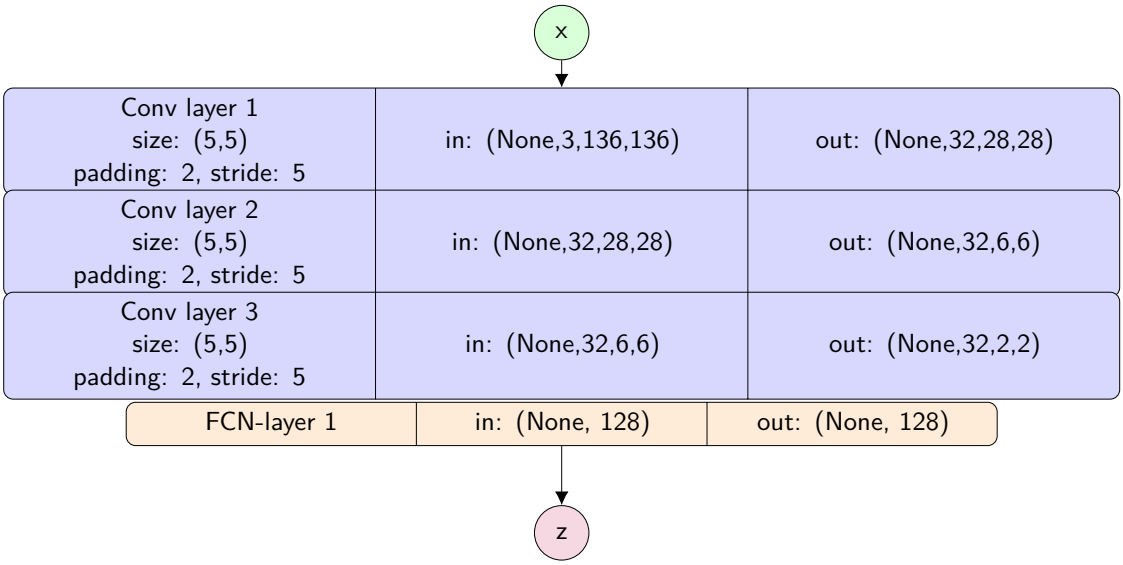


Fig. 4.2.: Final encoder architecture used for Maculinea Alcon clustering.

Choice of Loss function

After choosing the architecture, we can begin to modify the loss function and evaluate the results. To do this, we again start with MNIST, as a benchmark, and decide to test some popular autoencoder loss functions and regularizers, listed below along with a brief explanation of their usage.

- **MSE**: The mean squared error is our default loss function we use in this analysis, primarily because it is a popular choice for autoencoders
- **BCE**: Binary cross entropy is another popular loss function for images in autoencoders
- **Denoising**: Denoising autoencoders have gained some popularity as they allow the model to better generalize the data.
- **VAE - Gaussian**: Variational autoencoders are another popular autoencoder architecture. This is partially because they create manifolds in latent space which are traversible, and therefore can be better used as generative models, and also because they allow us to use some prior knowledge about the distribution of the dataset and apply that constraint to the latent representation. The zero-mean Gaussian is probably the most popular choice of prior for VAEs, so we begin with that.
- **VAE - LiPPy**: This is a VAE with a **L**aplacian **P**rior and **P**osterier. After trying some experiments with a zero-mean gaussian, we decided it might improve the clustering to try a distribution that allows the datapoints to spread out more, such as the heavier-tailed Laplacian. This architecture is explained in more detail in sec. 4.1 below.
- **VAE - GiMMPy**: Although the LiPPy model produces good results, we thought we could further improve them by using a **G**aussian **M**ixture **M**odel **P**rior and **P**osterior. This architecture is explained in more detail in sec. 4.2 below.

There are of course, many other architectures we could have explored, such as contractive autoencoders [37]. We limit ourselves to those listed above in the interests of time, and focus on variational autoencoders as we can also view them as generative models which give us interpretable results of the clustering.

4.1 LiPPy VAE

We decide to explore using a zero-mean **L**aplacian as the **P**rior and **P**osterior, similar to the popular zero-mean Gaussian. However, although we know the KL divergence between two 1D Laplacians (see sec. 3.3.6), we do not know it in the multivariate case. In fact, all my attempts to find a source for it, or calculate it myself were unsuccessful. Instead, we attempt to approximate the divergence by summing the KL divergences over the individual dimensions, as in (4.1).

$$\mathrm{KL}(L(\boldsymbol{u}|\boldsymbol{\mu_1}, \boldsymbol{b_1})||L(\boldsymbol{u}|\boldsymbol{\mu_2}, \boldsymbol{b_2})) \approx \sum_{i=1}^{d} \mathrm{KL}(L(u|\mu_{1,i}, b_{1,i})||L(u|\mu_{2,i}, b_{2,i})) \tag{4.1}$$

Intuitively, we expect this function to be minimized when the KL divergence between the multivariate Laplacian is minimized. Thankfully, although we cannot directly confirm this result, we can empirically test it using Monte Carlo sampling, in the same way we can approximate the KL divergence between two GMMs (see sec. 3.3.5), and run this approximation multiple times and look at the deviation. If we do this, we get the histogram of deviations in fig. 4.3

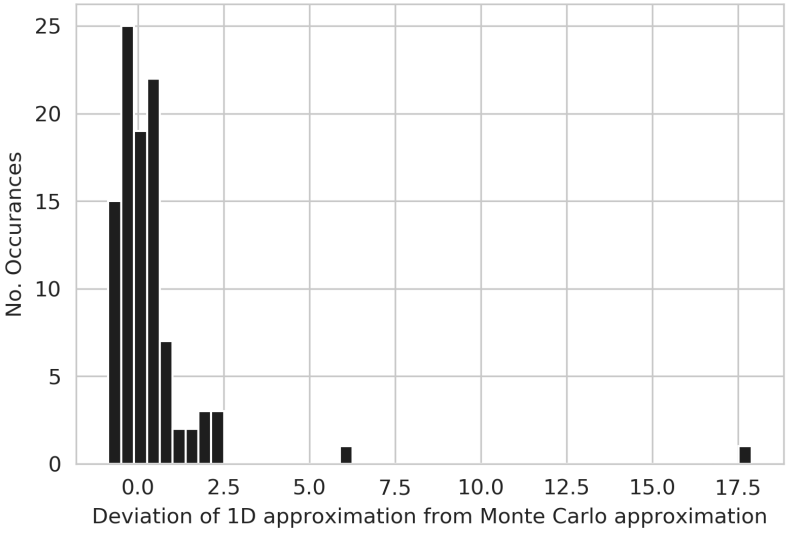


Fig. 4.3.: Histogram of deviation of Laplacian 1D-sum-of-KL-divergence-approximation from Monte Carlo approximation with 1000 samples. 100 experiments run, with $\boldsymbol{\mu}$ set between -1 and 1, $\boldsymbol{b}$ set between 0.1 and 5.

4.2 GiMMPy VAE

After some success with the Laplacian prior, we think we might be able to improve the results further by using a **G**aussian **M**ixture **M**odel **P**rior and **P**osterior, since this will allow the latent space to We could explore other mixture models, however, not only are GMMs the most widely studied mixture models, but they also have already been used for this application [11, 24], although in a slightly different way.

In order to adapt the VAE to use a GMM as the prior and posterior, we need to make some significant changes to the loss function and the latent space, as described in sections 4.2.1 and 4.2.2 below. For all these models we set the variance of all the Gaussians to 1.

4.2.1 Changes to the latent space

In the normal Gaussian VAE the output of the encoder is the mean and variance of the Gaussian posterior. To use a GMM as the posterior we instead need to do the following:

1. Add the weights ($\boldsymbol{w}$) for each Gaussian in the posterior GMM to the encoder output
2. Expand the encoder output to include a mean and variance vectors for each Gaussian in the posterior GMM

The reparametization trick applied to such a VAE is shown in fig. 4.4.

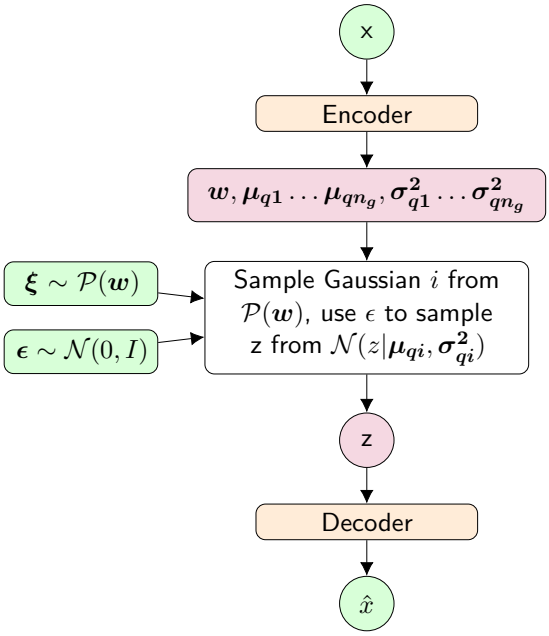


Fig. 4.4.: VAE with GMM Posterior, basic architecture. $\mathcal{P}(\boldsymbol{w})$ is the categorical distribution of mixture model weights, n_g is the number of gaussians in the posterior, $\boldsymbol{\mu}_{q1}$ is the mean vector for Gaussian 1 in the posterior, $\mathrm{diag}(\boldsymbol{\sigma}^2_{q1})$ is the covariance matrix for Gaussian 1 in the posterior.

4.2.2 Changes to the Loss Function

To start, in a VAE we would like to minimize the generic lower bound (see (3.39)). As described in the background, this can be decomposed into a KL divergence term on the latent space and a reconstruction term on the AE output. First, let's evaluate the kl divergence term. Since the KL divergence between two GMMs is in general intractable, we decide to estimate it instead using Monte Carlo sampling (see sec. 3.3.5). We do not make any changes to the reconstruction term.

Initialization of the Prior

In the normal Gaussian VAE, and the LiPPy VAE we assume the prior has a zero-mean, which simplifies our calculations. If we use a GMM we need to make some new assumptions about the location of the means. In [24] they pretrain the model to find the prior locations. Here we test two different methods instead:

1. **Random initialization**: We randomly initialize the prior, while ensuring that the Gaussian centers are at least 6 standard deviations away from eachother, to attempt to pull the clusters away from eachother and make them more distinct.
2. **EEK initialization**: Here, **E**ach **E**poch we run **K**-means on the posterior of the first batch, and set the means to the cluster centroids identified by K-means. This allows the model to adapt to the natural distribution of the data.

With these assumptions, we can use equations (3.2) and (3.1) to calculate the pdf of $q(z|x)$, sample k_i from $q(z|x)$, and then calculate the pdfs of $q(z|x)$ and $p(z)$ at k_i to finally calculate the monte carlo estimate of the KL divergence, and use this in the loss function.

Some notes on the implementation of the Monte Carlo sampling

Sampling: As shown in sec. 3.3.5, in high dimensional space Monte Carlo sampling requires many (usually upwards of 10,000) samples to provide good accuracy. Instead of using loops to take these samples, they are taken simultaneously for each batch using a large randomly initialized matrix. This helps keep the runtime reasonable.

Simplification of the pdf: Because we assume that both the covariance matrices are diagonal the calculation of the PDF is greatly simplified, since then the following replacements can be made:

1. $\mathbf{\Sigma}^{-1} = \text{diag}(\frac{1}{\mathbf{\Sigma}_{1,1}}, \frac{1}{\mathbf{\Sigma}_{2,2}}, ... \frac{1}{\mathbf{\Sigma}_{d,d}})$
2. $|\mathbf{\Sigma}| = \mathbf{\Sigma}_{1,1}\mathbf{\Sigma}_{2,2}...\mathbf{\Sigma}_{d,d}$

The PDF of the GMM is then reduced as follows:

$$\sum_{i=1}^{n_g} w_i \mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma_i}) \tag{4.2}$$

$$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = (2\pi)^{\frac{d}{2}} |\boldsymbol{\Sigma}|^{-\frac{1}{2}} e^{-\frac{1}{2}(\boldsymbol{k}-\boldsymbol{\mu})\frac{1}{\boldsymbol{\Sigma}}(\boldsymbol{k}-\boldsymbol{\mu})} \tag{4.3}$$

The removal of the inverse function and the replacement of the determinant with a simple product, greatly decrease runtime and make this approximation feasible.

Overflow/Underflow: Overflow and underflow were initially a problem in this implementation. In one case, overflow / underflow resulted in the monte carlo sampling itself, when $\log \frac{q(k_i)}{p(k_i)}$ is calculated, as both values tend to be very small, particularly in high-dimensional latent spaces. This was replaced with $\log(q(k_i)) - \log(p(k_i))$ which seemed to fix the issue for our purposes.

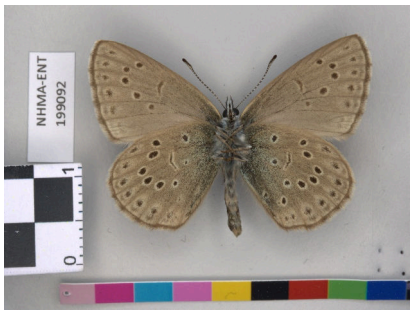
In another case, the $|\boldsymbol{\Sigma}|$ term tended to produce infinities enough of the variances were $\gg 1$. To circumvent this $det(\Sigma)$ was replaced with $e^{\log|\Sigma|}$. Since the determinant is the product of all the diagonal elements for a diagonal matrix, ($|\Sigma| = \Sigma_1 \Sigma_2 \dots \Sigma_n$), the log of this, is the sum of the individual logs $\log(|\Sigma|) = \log(\Sigma_1) + \log(\Sigma_2) + \dots + \log(\Sigma_n)$. With these, equation (3.2) finally becomes:

$$\mathcal{M}(\boldsymbol{u}|w_1 \dots w_{n_g}, \boldsymbol{\mu_1} \dots \boldsymbol{\mu_{n_g}}, \boldsymbol{\Sigma_1} \dots \boldsymbol{\Sigma_{n_g}}) = \sum_{i=1}^{n_g} w_i (2\pi)^{\frac{d}{2}} e^{-\frac{1}{2}\log(|\boldsymbol{\Sigma}|)} e^{-\frac{1}{2}(\boldsymbol{u}-\boldsymbol{\mu})'\boldsymbol{\Sigma}^{-1}(\boldsymbol{u}-\boldsymbol{\mu})} \tag{4.4}$$

4.3 Preprocessing of the Maculinea Alcon Dataset

The biggest challenge in using the Maculinea Alcon dataset, is getting it into a workable form. Although all the images were taken in similar lighting, with similar orientations of the butterflies, the camera height was not always the same, the lighting has some variation, and the butterflies themselves are not perfectly registered. Therefore, in order to make clustering of the images more successful and make image analysis more precise, we first complete some preprocessing of the images to segment, scale and transform them.

Our end goal here is to take the original images such as those on the left in fig. 4.5, and transform them into those on the right. For the purposes of this study, the antennae and body are not so considered crucial, as they provide very little indication of gender, and are difficult to segment, so many of the images may be missing parts of these, like the top image in fig. 4.5.



(a) Original NHMA Image

(b) Pre-processed NHMA Image

(c) Original NHMD Image

(d) Pre-processed NHMD Image

Fig. 4.5.: Original (left) and desired pre-processed images (right)

To begin the preprocessing, we can define some general steps that need to be taken:

1. Image conversion and resizing
2. Segmentation
3. Scaling
4. Translation
5. Rotation
6. Color Correction

It should be noted that the butterfly dataset is split into two groups, those imaged in Aarhus, and those imaged in Copenhagen. Each dataset was created with a different camera and in different lighting. Therefore each dataset is initially preprocessed separately using similar techniques with different hyperparameters. See chapter 2 for a more in-depth comparison of the two datasets. We will show the main distinctions in how these were treated as we go along.

4.3.1 Image Conversion and Resizing

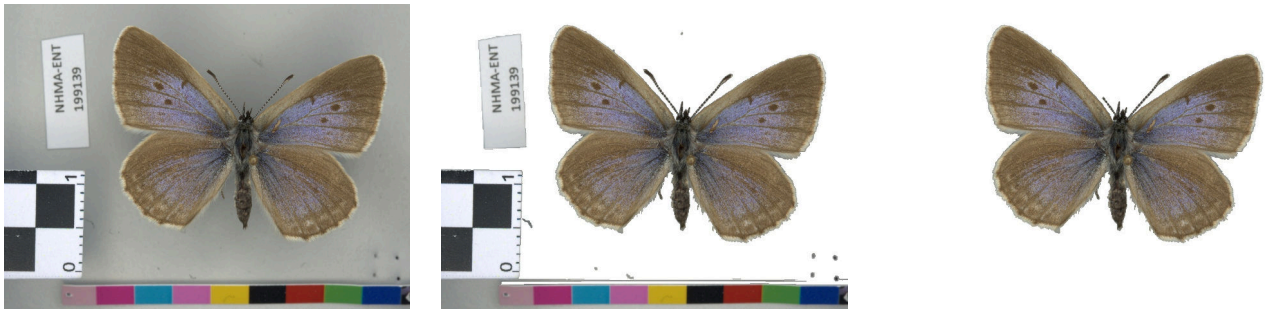
The images are provided in raw format (CR2 for the Aarhus images and ARW for the Copenhagen images) with high resolution (5496x3670 for both). In order to make them more easy to process, they are read using the rawpy python package, resized (to 1024x768 for Aarhus and 2048x1536 for Copenhagen - The sizes are initially different because the butterflies occupy a smaller section of the image in the Copenhagen images). They are then converted to JPEG images with 95% quality using opencv. This greatly reduces the time required for the rest of the preprocessing steps, while still providing sufficient quality for the analysis. Here, the dataset is also meticulously manually scoured to correct any misnamings of the images.

4.3.2 Segmentation

After resizing, we remove the background of the images to ensure that only image information relevant to the insect is fed into (and learnt by) the autoencoder. This can be done using felzenwalb image segmentation implemented in sklearn. It is completed in two steps:

1. Using low thresholds, the segment which represents the background of the butterfly can be found and removed. The background is easier to find as one segment than the butterfly itself because it has much smaller variation.
2. Once the background is removed, another iteration of felzenwalb with higher thresholds can be used to find the butterfly and remove everything else.

This two step process is illustrated in fig. 4.6.



(a) Original (b) Image after background removed (c) Final segmentation

Fig. 4.6.: Two-step segmentation process completed by first removing the background which is easy to find as one segment, then finding the butterfly and removing the rest.

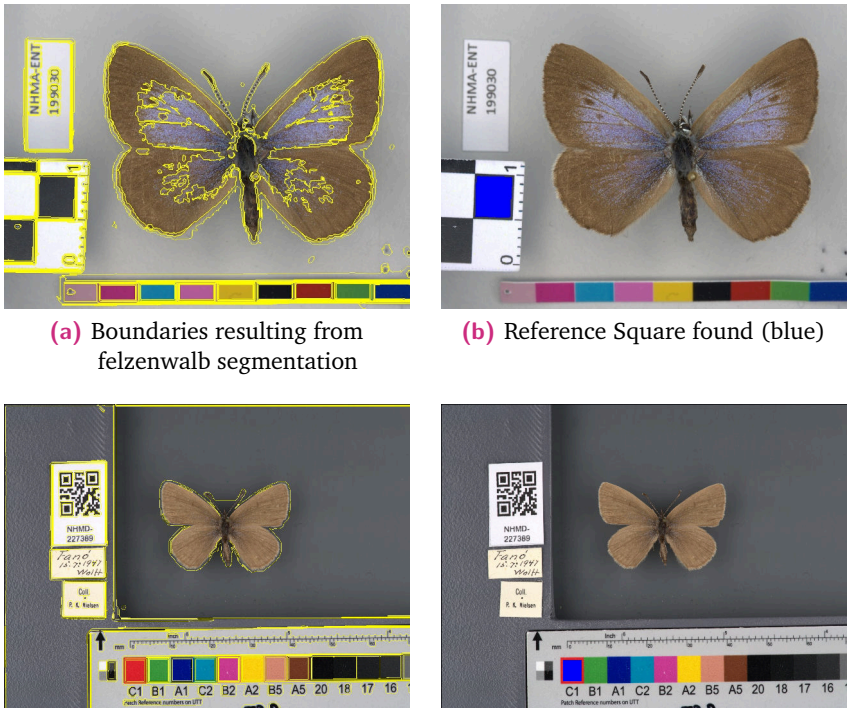
4.3.3 Scaling

After the images are segmented, all the images are scaled to 200pixels/cm. The following steps are taken to complete this. This process is detailed in the sections that follow.

1. The reference square is found/segmented in the original images
2. The corners of the segmented reference square are found using the harris corner detector
3. The general location of the corners is manually assessed and corrected as necessary
4. The exact location of the corners is refined using the filter response of handpicked filters
5. Each image is scaled to 200 pixels/cm, based on the distance between the corners of the segmented reference square. Any distortion resulting from previous resizing was also mitigated by scaling both horizontally and vertically.

Segmenting the reference square

Since the reference square is in roughly the same location in all the images, and is approximately uniform, we can first segment a reference image and manually locate the segment with the reference square. We can then later use this to find the reference square segment in new images. The two reference images for the NHMA and NHMD datasets are shown in fig. 4.7).



(a) Boundaries resulting from felzenwalb segmentation

(b) Reference Square found (blue)

(c) Boundaries resulting from felzenwalb segmentation

(d) Reference Square found (blue)

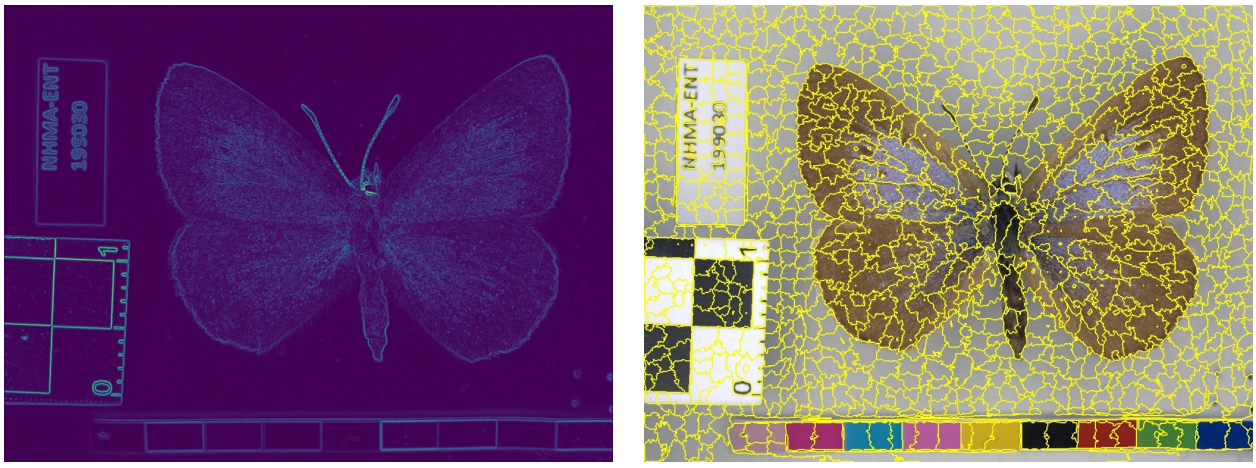
Fig. 4.7.: Felzenwalb segmentation and reference square segment (blue) found for NHMA (a and b) and NHMD (c and d) datasets. In reality the black square in the NHMA images is 0.5 cm, and the red square in the NHMD images is 0.58 cm.

We can then take any new image, segment it using the same parameters, and create a vector for each segment with the size (normalized by dividing by 100), location of the centroid and average rgb values. It looks like: [size/100, x-centroid, y-centroid, avg-red, avg-green, avg-blue]. We then calculate the Euclidean distance between these vectors, and the vector for the reference square we found in fig. 4.7. We flag the closest segment as the reference square segment for the new image, and repeat the process for all images.

Refining the edges of the reference square segment

If you look closely at the reference square segment in fig. 4.7, you will see that it doesn't quite reach the corners of the reference square, which is a problem since we want to use this to precisely scale the images. This is an artifact of the parameters used during felzenwalb segmentation which is good for finding large patches with low contrast, but not so good at finding edges. Therefore, we now need to refine these segments to ensure we get the right corner locations.

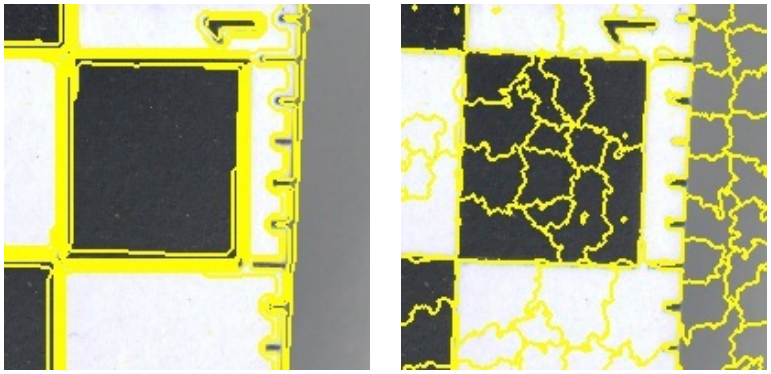
To do this, we will combine the segments we found with watershed segmentation applied to the sobel response of the image. This gives us more accurate edges, although the edges are also more jagged (see fig. 4.8 and fig. 4.9).



(a) Sobel response image **(b)** Resulting watershed segmentation

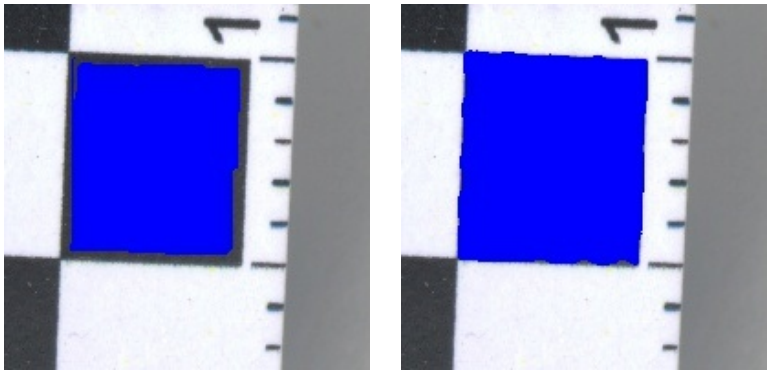
Fig. 4.8.: Sobel filter response image and corresponding watershed segmentation used to find edges of cm scale

Zooming into the reference square (fig. 4.9, top), we can see the difference between the felzenwalb segmentation and watershed segmentation in more detail. Once the image had a watershed segmentation, we can take the felzenwalb segments from before, and expand them to include any overlapping watershed segments. The result is show in fig. 4.9 (bottom). This now looks reasonable enough to attempt to find the corners which can then be used to calculate the real-life size of the objects in the image.



(a) Felzenwalb segmentation

(b) Watershed segmentation



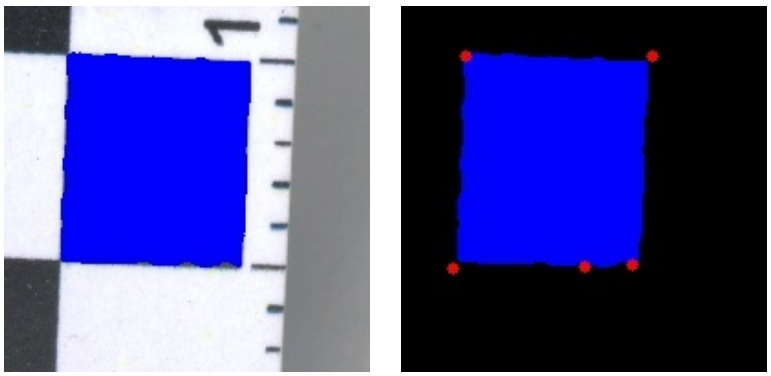
(c) Original felzenwalb segment

(d) Expanded felzenwalb segment to include overlapping watershed segments

Fig. 4.9.: Comparison of felzenwalb segmentation and watershed segmentation, zoomed into the reference square (top) and resulting segmentation (bottom).

Finding the corners of the reference square

Now the corners of the reference square can be found using the watershedded-Felzenwalb segments. This is done using a Harris corner detector on the binary image of the cm segment.



(a) Reference square segment

(b) Corners found

Fig. 4.10.: Corners found (red) by Harris corner detector on binary segment (blue)

There are some problems with this methodology: there are cases where too many corners were found (as above), or the corner locations are wrong. These are corrected as described below.

Manual Checks and Corrections for Corner Locations

Manual Checks were then made at this stage for all the images to ensure that the corner locations found using the harris corner detector were correct. This was unfortunately necessary as there were many cases where the corner detector would find too many corners in the image, or too few, or in slightly the wrong location. Some examples of these problems are shown in fig. 4.11, while fig. 4.12 shows the GUI used to complete the manual corrections.

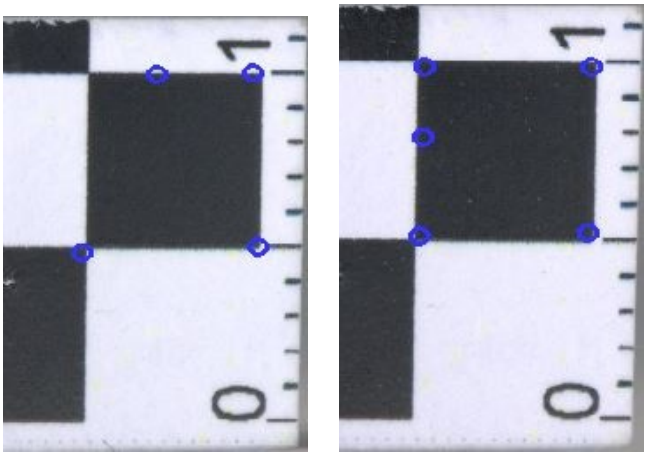
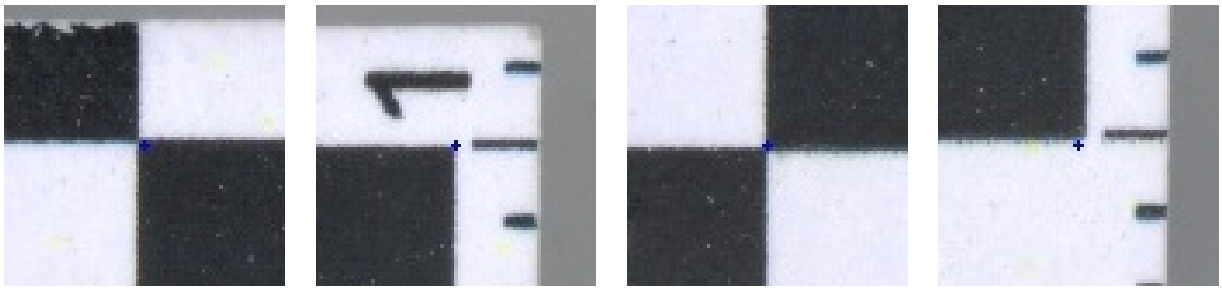


Fig. 4.11.: Examples of problems encountered when locating corners. Corner locations indicated by blue circles.

Fig. 4.12.: Graphical user interface made using pyqt to manually correct reference square corner locations and average color segmentation

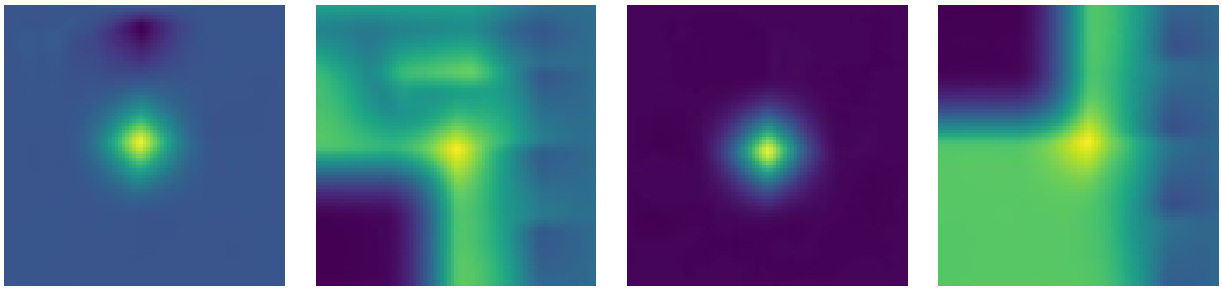
Fine-tuning the corner locations

As can be seen in fig. 4.13, some of the corner locations found (shown in blue) are still too far from the actual corner pixel. This was corrected by applying hand-picked filters to the area around the corner, then moving the corner location to the pixel with the highest filter response.

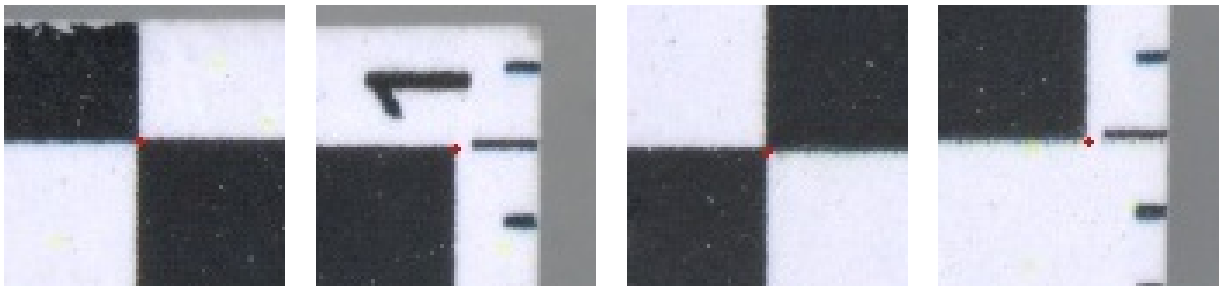


(a) Top left corner location found (blue)

(b) Filter applied to image patch



(c) Filter response image



(d) Corrected corner location found at location of maximum filter response (red).

Fig. 4.13.: Demonstration of corner location correction process using hand-picked filters

Calculating the scaling factors

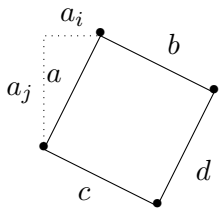


Fig. 4.14.: Labelling scheme for reference square, used in calculating the scaling factor in (4.5) below.

Now that we have the corner locations and their real-life distances, we can use them to calculate the scaling factors in the horizontal and vertical directions (i and j, respectively). To do this, we create a set of equations. To do this, we first assume that the only scaling present is in the i and j directions. Using the assumption that the distance between each adjacent corner should be equal to our predefined number of pixels, we get:

$$(s_i a_i)^2 + (s_j a_j)^2 = p^2$$

$$(s_i b_i)^2 + (s_j b_j)^2 = p^2$$

$$(s_i c_i)^2 + (s_j c_j)^2 = p^2$$

$$(s_i d_i)^2 + (s_j d_j)^2 = p^2$$

Then using the assumption that adjacent edges should be orthogonal to eachother (have a dot product of 0):

$$\boldsymbol{s} \odot \boldsymbol{a} \bullet \boldsymbol{s} \odot \boldsymbol{b} = 0$$

$$\boldsymbol{s} \odot \boldsymbol{a} \bullet \boldsymbol{s} \odot \boldsymbol{c} = 0$$

$$\boldsymbol{s} \odot \boldsymbol{c} \bullet \boldsymbol{s} \odot \boldsymbol{d} = 0$$

$$\boldsymbol{s} \odot \boldsymbol{d} \bullet \boldsymbol{s} \odot \boldsymbol{b} = 0$$

where $\boldsymbol{s} = [s_i, s_j]$ is the scaling vector containing the scaling factors in the i and j directions, p is the desired number of pixels between the corners. For the Aarhus dataset, since the corners are physically 0.5 cm apart, and we want to have 200 pixels per cm, p is 100. For the copenhagen dataset it is 116.

These equations can be rearranged into a linear equation in $\boldsymbol{s}^2$, as follows:

$$\boldsymbol{A}\boldsymbol{s}^2 = \boldsymbol{b} \tag{4.5}$$

where,

$$\boldsymbol{A} = \begin{bmatrix} a_i^2 & a_j^2 \\ b_i^2 & b_j^2 \\ c_i^2 & c_j^2 \\ d_i^2 & d_j^2 \\ a_i b_i & a_j b_j \\ a_i d_i & a_j d_j \\ c_i d_i & c_j d_j \\ b_i c_i & b_j c_j \end{bmatrix}, \boldsymbol{s}^2 = \begin{bmatrix} s_i^2 \\ s_j^2 \end{bmatrix}, \boldsymbol{b} = \begin{bmatrix} p^2 \\ p^2 \\ p^2 \\ p^2 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Now, the Moore-Penrose pseudo-inverse can be used to approximate s_i^2 and s_j^2 as follows:

$$\boldsymbol{A}^+ = \boldsymbol{A}'(\boldsymbol{A}\boldsymbol{A}')^{-1}$$

$$\boldsymbol{s}^2 \sim \boldsymbol{A}^+\boldsymbol{b}$$

Once s_i and s_j are calculated, the butterfly image is scaled appropriately, as shown in fig. 4.15

Fig. 4.15.: Segmented image before (left) and after (right) scaling

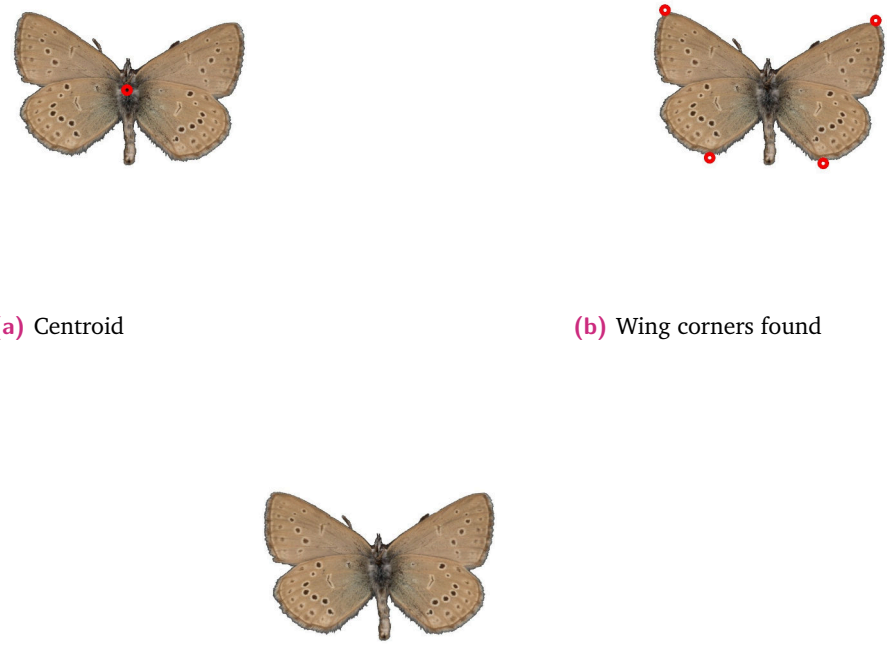
4.3.4 Translation and Rotation

After the images were scaled, the images were translated by first finding the centroid of the butterfly mask, then moving the butterfly so the centroid would be at the center of the image.

After the images were translated, they were rotated so that they were all approximately in the same place. This was done via the following:

1. The top and bottoms of the left and right wing sections were approximately found. The top points were found by passing over the segmentation mask from the top to the middle, until a pixel was found that only had pixels below it or to the side of it, but not within 10 pixels above. A similar procedure was used to find the bottom points on the left-lateral and right-lateral sections.
2. This algorithm also tended to find points on the butterfly's head and antennae, these points were removed by taking the top left, bottom left, bottom right, and top right-most points that fit the criteria.

This process is shown graphically in fig. 4.16



(a) Centroid

(b) Wing corners found

(c) Resulting translation and rotation

Fig. 4.16.: Centroid found (a), used to move the specimen to the center of the image. Wing corners found (b), used to rotate the specimen to be horizontal. Resulting translation shown in (c).

4.3.5 Color Correction

Since all images include a colorbar, we thought we could use this to approximately color correct the images. For this process, the Copenhagen dataset and Aarhus dataset were treated seperately since they used different cameras and camera settings, different lighting, and different backgrounds. In both setups the respective color bar was found using felzenwalb segmentation, and the average color (rgb) of each of 8 color segments in each image was calculated. Then the color correction was completed following this process:

1. The average color target rgb values for all the Aarhus images were calculated. The averages were then assumed to be the 'ground truth' color bar values for the Aarhus dataset, since we did not have actual ground truth rgb values for the color target.
2. Each image's color bar rgb values were compared to the 'ground truth' color bar, and the transformation from the image's color bar to the ground truth color bar was calculated using the pseudoinverse of these 8 values, with an added bias term.
3. This tranformation was then applied to all the pixels of the image.
4. In case the result of the color transformation created values below 0 or above 255, the values were clipped to be within [0,255].

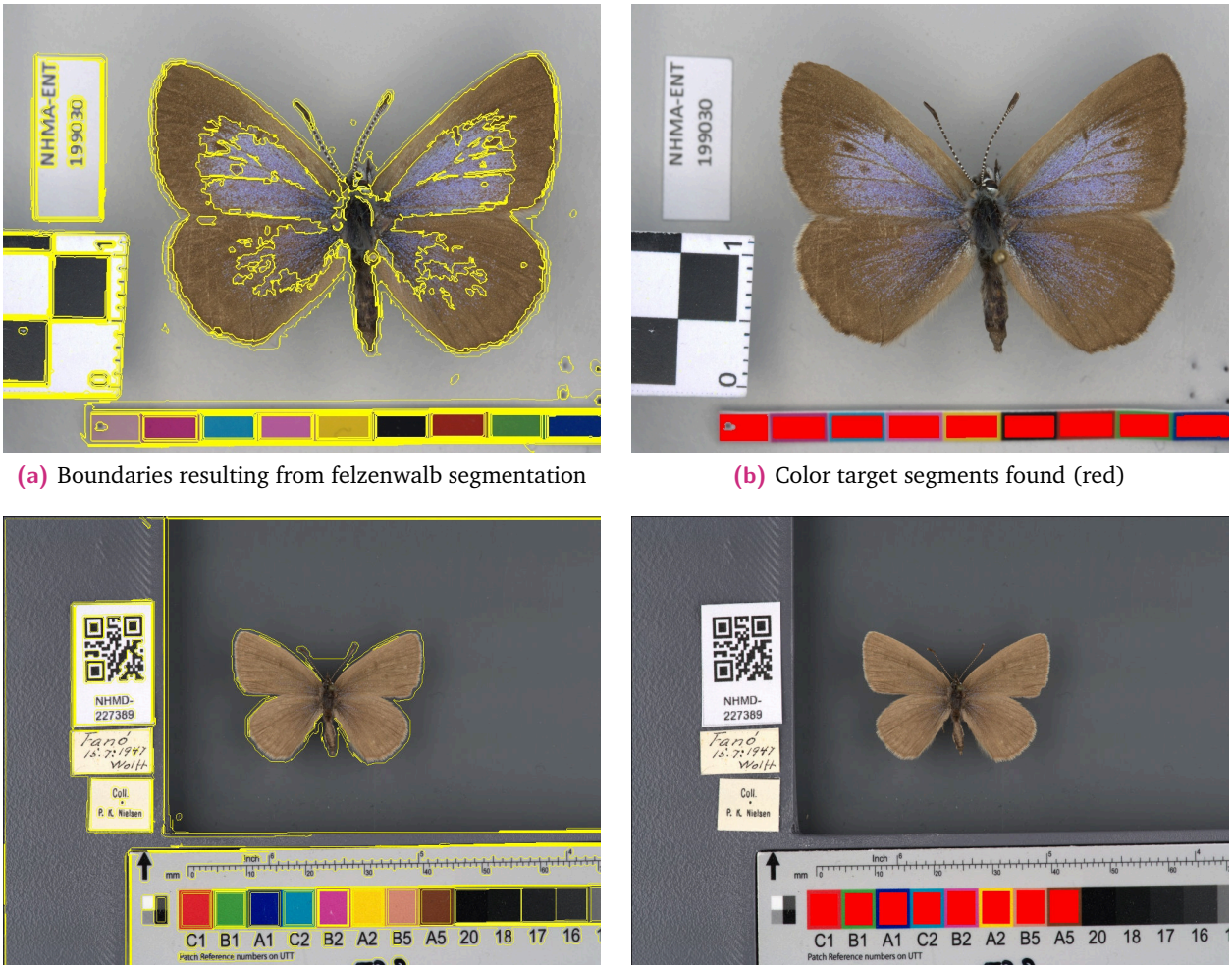
Finding the reference colors

Finding the color target is necessary to color correct the images. To find the locations we can use almost exactly the same process as we used to find the reference squares, described in sec. 4.3.3.

Since the color target is in roughly the same location in all the images in the same dataset, felzenwalb segmentation can again be used to automatically find segments which are rather large, mostly the same color and in a similar location. To do this, the same ground truth reference images can again be used and the color segments manually found (see the red segments in fig. 4.17).

Once this is found in one of the images for each dataset, we can, as we did for the reference square, segment all the other images in a similar way and again compare the [size/100, x-centroid, y-centroid, avg-red, avg-green, avg-blue] vector for each segment to find the most similar segment. We can then calculate and save the average rgb values for each color target square for each image.

In some cases the algorithm found the wrong segment - So I checked each image manually at the same time as checking the corner locations, using the gui shown in fig. 4.12.



(a) Boundaries resulting from felzenwalb segmentation

(b) Color target segments found (red)

(c) Boundaries resulting from felzenwalb segmentation

(d) Color target segments found (red)

Fig. 4.17.: Felzenwalb segmentation and color target segments (red) found for Aarhus (a and b) and Copenhagen (c and d) datasets.

Color Correcting the Aarhus Images

Since we had no ground truth rgb values for the Aarhus color target, we simply took the average of the color target values for all the images as the ground truth color values. We then took each image and used the pseudoinverse of the ground truth color target values with a bias term to calculate the optimal color correction, as follows. Assuming each color should be adjusted as a linear combination of the other colors and a bias term. ie: $R = rs_{r,r} + gs_{r,g} + bs_{r,b} + w_r$, where R is the desired values of the red channel, r is the current value of the r channel, s is a scaling factor for each of the other channels, and w_r is a bias term for the red channel to account for differences between intensities in images. We can rewrite this as a set of linear equations as in (4.6).

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{4.6}$$

where,

$$\boldsymbol{A} = \begin{bmatrix} r_1 & g_1 & b_1 & 1 \\ r_2 & g_2 & b_2 & 1 \\ r_3 & g_3 & b_3 & 1 \\ r_4 & g_4 & b_4 & 1 \\ r_5 & g_5 & b_5 & 1 \\ r_6 & g_6 & b_6 & 1 \\ r_7 & g_7 & b_7 & 1 \\ r_8 & g_8 & b_8 & 1 \end{bmatrix}, \boldsymbol{x} = \begin{bmatrix} s_{r,r} & s_{g,r} & s_{r,b} \\ s_{r,g} & s_{g,g} & s_{g,b} \\ s_{r,b} & s_{g,b} & s_{b,b} \\ w_r & w_g & w_b \end{bmatrix}, \boldsymbol{b} = \begin{bmatrix} R_1 & G_1 & B_1 \\ R_2 & G_2 & B_2 \\ R_3 & G_3 & B_3 \\ R_4 & G_4 & B_4 \\ R_5 & G_5 & B_5 \\ R_6 & G_6 & B_6 \\ R_7 & G_7 & B_7 \\ R_8 & G_8 & B_8 \end{bmatrix}$$

Color Correcting the Copenhagen Images

Once the Aarhus images are corrected, it seems surprisingly difficult to translate this color correction to the Copenhagen images, which has a different color bar. Initially we might try to use an image which has both color bars in it to translate between the two. But after trying this, we discovered that the colors bars were not evenly illuminated in the image, which led to the color correction producing much darker images for the Copenhagen dataset. However, thankfully in the process of creating these datasets, Philip Folman also took 60 images of Aarhus specimens in the Copenhagen setup. Therefore we could directly compare the results on the specimens for 60 images. Using this, the color correction was completed by:

1. Finding the average rgb values for each of the segmented/transformed resamples in the Aarhus setup and in the Copenhagen setup.
2. For each set of Aarhus and Copenhagen images, subtracting the difference between the average RGB values of the two images, and adding this to the Copenhage image. Effectively we are directly translating each resample to the Aarhus coloring.
3. Now that we have the RGB values that each resample should have to be in the Aarhus system, we can take the average RGB values of their color corrected color bars, and set this to be the ground truth color bar for the Copenhagen dataset.
4. Then we repeat the steps we took with the Aarhus dataset, to color correct the Copenhagen images to their new ground truth values. We also apply this transformation to the resampled images, so we can visually inspect the results.

Despite our best efforts however, there are still noticable differences between the images taken in the different setups. The reason could be related to the color temperature of the different lighting in the two setups. To illustrate this difference, and the slight differences between the segmentation processes, images of all the resamples are provided in appendix A. To illustrate this difference, an example is provided in fig. 4.18.

(a) Aarhus Before Color Correction

(b) Aarhus After Color Correction

(c) Copenhagen Resample Before Color Correction

(d) Copenhagen Resample After Color Correction

(e) Aarhus Before Color Correction

(f) Aarhus After Color Correction

(g) Copenhagen Resample Before Color Correction

(h) Copenhagen Resample After Color Correction

Fig. 4.18.: Example color correction done on Aarhus images and the Copenhagen resamples of the same specimen, before and after each color correction process.

4.3.6 Altering the Background

After running some preliminary experiments on the images, we saw that the model seemed to be spending some effort trying to learn the location of the body and antennae in the models. This makes sense, since they are black and therefore present a large error term against the background. Since we are pretty certain that the antennae and body are not imporant factors for our labels of interest, we want to minimize the amount of focus the model puts on these components. To do this, as a final preprocessing step, we replaced the background of the images with the average butterfly color - a kind of muddy brown. An example is shown in fig. 4.19.

(a) Color Corrected Image

(b) Color Corrected Image with Neutral Background

Fig. 4.19.: Example specimen image before and after background neutralization.

5 Results

> *More human than human is our motto.*
>
> — **Tyrell**
> Blade Runner

In the chapter we present the results of the experiments from this thesis.

First, in sections 5.1-5.2 we evaluate the effect of various architectures and loss functions on cluster formation for a subset of MNIST.

Second, in sec. 5.3 we build on the MNIST results and use them to design experiments for the Maculinea Alcon dataset, and evaluate the effect of different architectures on cluster formation there.

Then, in secs. 5.4-5.5 we use the final architecture and loss functions from the Maculinea Alcon experiments to explore the dataset further, looking at how the dataset is distributed in the resulting latent spaces.

Finally, in sec. 5.6 we make some qualitative remarks about the image processing methodology for the Maculinea Alcon dataset and the manual work still involved therein.

5.1 MNIST - Experiments on Architecture

To start exploring architectures, we make some key choices:

1. First, we decide to use an **autoencoder** (AE) to explore encouraging cluster formation in latent representations. There are other unsupervised learning methods such as deep belief networks and self-organizing maps (see. sec. 3.6.10) but AEs are a popular method of unsupervised learning which lend well to clustering in the latent space, so in this thesis we will focus entirely on them.

2. Then, we decide for convenience that the **decoder will look like the inverse of the encoder** * - there seems to be no reason not to do this, and it is a popular choice. However, we do not restrict the model to have tied weights† - this is mostly for convenience in coding.
3. Finally, we decide that we do not want the latent space to just learn the identity function - that won't be very helpful. Therefore, we know we need an undercomplete AE or an overcomplete AE with some kind of sparse regularization. Here, for simplicity, we will limit ourselves to only considering **undercomplete AEs**.

To approach the choice of architecture, we will start by looking at the simplest possible autoencoder that could meet our needs, and then add complexity to the model and evaluate how it affects the cluster formation.

5.1.1 The Simplest AE - Network 1

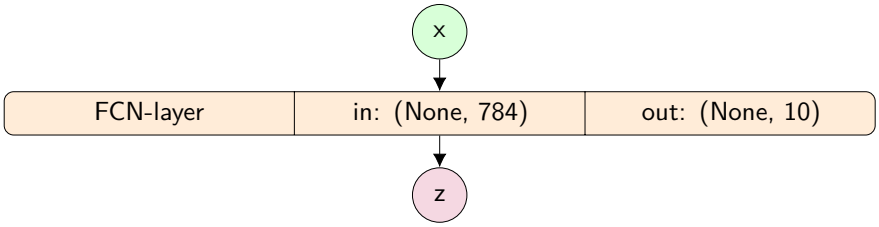


Fig. 5.1.: Simplest AE encoder (note: only encoder is shown here for brevity, as decoder is simply the reverse of the encoder. This will be the case for all the architectures shown).

Shown in fig. 5.1 is the most simple AE we could imagine, so let's start with that. We quickly choose to use the subset of MNIST that includes the digits 0 and 1, to start. The projection of the data in the latent space onto the first two principal components (obtained by PCA) is shown in fig. 5.2.

Looking at the PCA, it seems like our work is already done - the two digits are nicely arranged in well-separated clusters without any need of further depth or regularization... However, you might quickly suspect that this might not work so well for different number combinations - perhaps 0 and 1 are just easy to distinguish. So let's quickly run the same experiment for all the possible 2-digit combinations. We use k-means to cluster the latent space and instead of visually inspecting the PCA for all the combinations, we use the adjusted rand index (ARI) (see sec. 3.5.3) to quickly compare the results. We use the hyperparameters listed in tab. 5.1. The full table of results with various metrics can be found in Appendix B. A heatmap of the ARI results is shown in fig. 5.3.

*except for max pooling layers which are inverted using interpolation, and the variational autoencoder which requires a slight difference between the encoder and decoder just before the latent space

†have the encoder's weights be the inverse of the decoder's weights

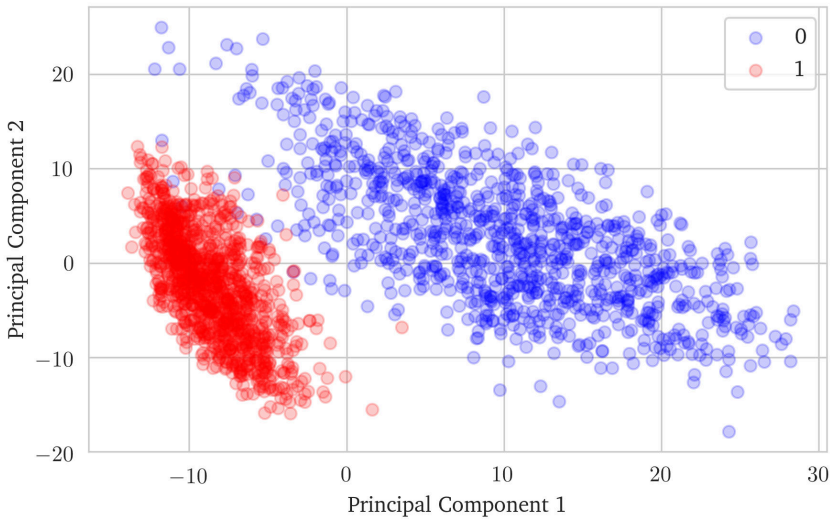


Fig. 5.2.: First two principal components of latent space of validation set of 0-1 mnist subset using simplest autoencoder

Hyperparameter	Value
Experiments	all 2-digit combinations of (0,1,2,3,4,5,6,7,8,9)
No. runs per experiment	1
latent dimensions	10
optimizer	adam
learning rate	0.001
loss function	MSE
max epochs	50
early stopping	10
batch size	200
activation function	ReLU

Tab. 5.1.: Hyperparameters for simplest AE experiment

From the results in fig. 5.3, we can see that 0-1 was a poor initial choice, as they are two of the most easily clustered numbers for this model. The three hardest sets of numbers to discern (shown in blue in the heatmap) are 6-8, 2-9 and 1-2. To keep the subset small going forward, but add some further complexity, **we decide to use the subset of 1-2-9 for further experiments**. We will also increase the latent space to 30 variables. This is still less than the 784 pixels in the images, but should be large enough to learn the digits well.

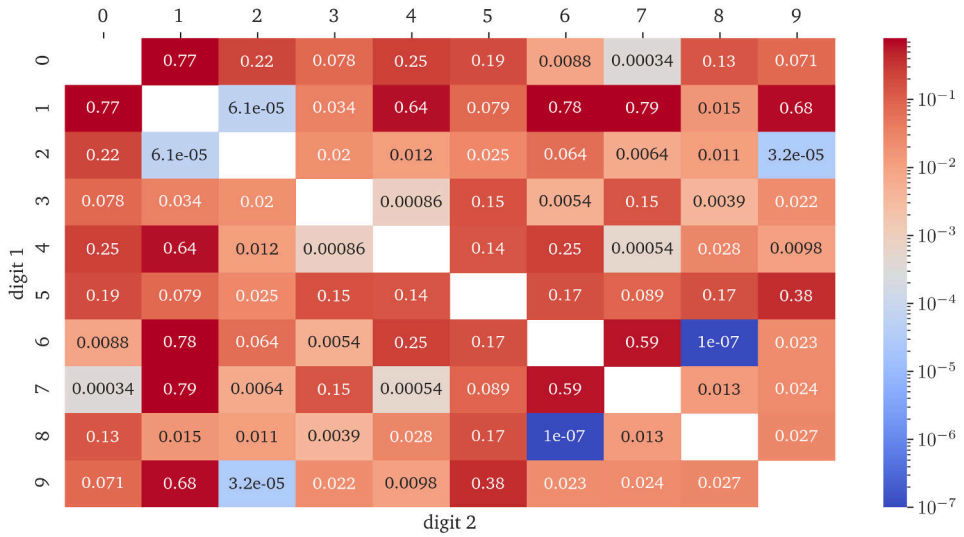


Fig. 5.3.: Log-scale heatmap of ARI for two digit combinations of MNIST. Red are the combinations with the highest ARI. Blue are the combinations with the lowest ARI, corresponding to the worst clustering.

5.1.2 Adding Hidden layers - Network 2

Next, we look at the effect of adding hidden layers to the model. To do this, we can run our model again on the 1-2-9 MNIST subset, while varying the number of hidden layers and the number of hidden features per layer. The basic architecture is shown in fig. 5.4.

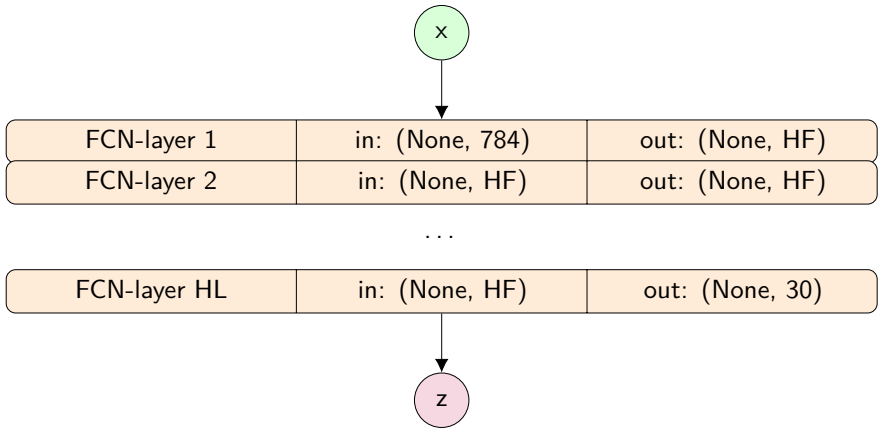


Fig. 5.4.: Architecture of encoder for experiments with variable hidden layers. HF: the number of hidden features, HL: the number of hidden layers

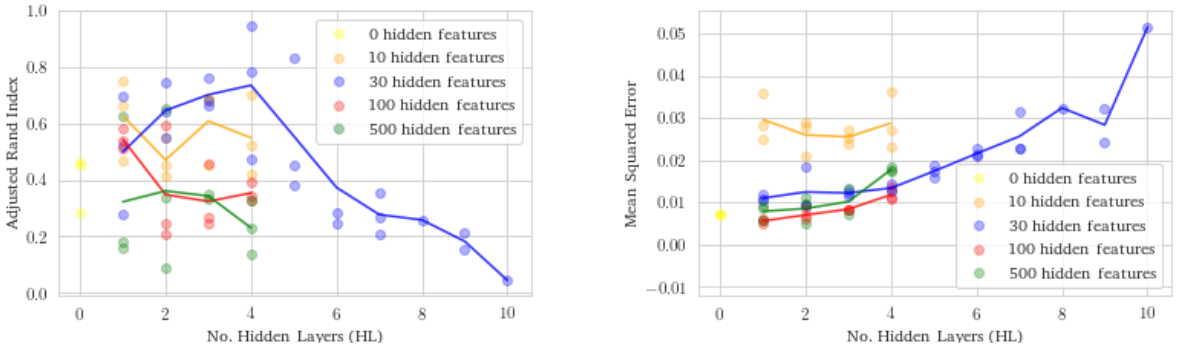
To test, we can choose, for example, to compare 0,1,2,3 and 4 hidden layers, each with 10,30,100, or 500 features. To account for random error, we run the model at least 3 times for each experiment. The hyperparameters for these runs are summarized in tab. 5.2.

Hyperparameter	Value
Experiments	hidden features: 10,30,100,500 hidden layers: 0,1,2,3,4
No. runs each experiment	3
latent dimensions	30
optimizer	adam
learning rate	0.005[a]
loss function	MSE
max epochs	500
early stopping	10
batch size	200
activation function	ReLU

Tab. 5.2.: Hyperparameters for hidden layer experiments

[a] 0.002 was used for the experiments with 500 hidden features, to avoid problems with convergence

The full table of results can be found in tab. C.1, and are summarized in fig. 5.5.



(a) Adjusted rand index **(b)** Mean squared error

Fig. 5.5.: Adjusted rand index (left) and Mean Squared reconstruction Error (right) on trials varying the number of hidden layers and hidden features. Each point on the scatter plot is one full experiment. The lines represent the mean of all the trials for that configuration of hidden features and hidden layers.

Fig. 5.5 shows how the average ARI changes when we vary the number of hidden layers and features per layer. We can see that in most cases the MSE increases as the number of layers in the model increases. This is not what we would expect, as we would expect that MNIST is better represented by more highly non-linear functions, which are better approximated with many layers. The depth also increases the vanishing gradient problem, and that could be a factor, but for such shallow networks it should not be significant. We also see that the MSE tends to decreasee as the number of hidden features per layer increases, this is along the lines of our expectations as we expect added complexity to be able to better fit the data.

Looking at the ARI however, we get another surprise. First, we notice that the ARI and MSE are not closely correlated. This leads us to believe that for cluster formation, using MSE loss may not be the optimal choice. For now, however, we will continue exploring the effect of the architecture using this loss since it is a standard loss function for this application. In the next section of this chapter (sec. 5.2) we will explore the effect of modifying the loss function on the cluster formation.

From these trials, the best model for cluster formation appears to be a 4-layer 30-feature model, although the results do exhibit a high variation across experiments. We also notice that the best model has 30-features which is the same dimensionality as our latent space,

5.1.3 Added Convolutional Layer - Network 3

The results of the hidden layer experiments look more convincing than our 1-layer 'simplest' model. However, we suspect we might further improve the result by adding convolutional layers as they should more easily allow the model to detect higher level image features such as edges.

To start, we fix the number of hidden layers to be 3, each with 30 features, and vary the number of convolutional layers. We choose 3 because it still performed well in the hidden layer experiments, but we want it to be less than the best number of layers, because we are adding more complexity to the model in the form of convolutional layers. We choose 30 because that also performed well in the hidden layer experiments. For the first fully connected layer, however, we use 500 to ease the network into the bottleneck, because the previous convolutional layer has at least 6,782 nodes. The architecture is shown in fig. 5.6, and the hyperparameters listed in tab. 5.3.

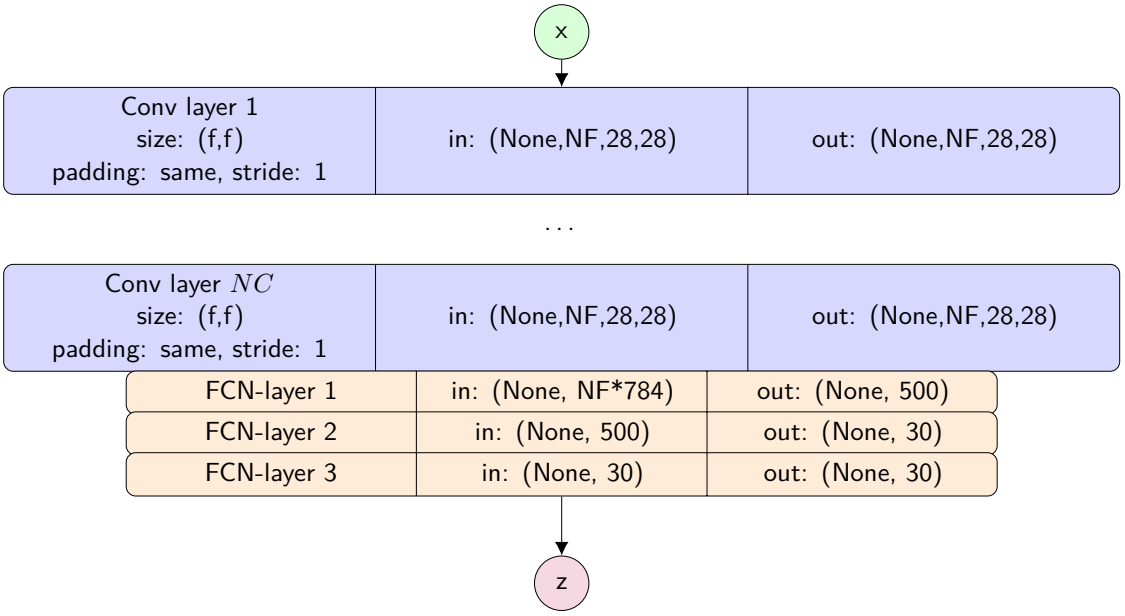
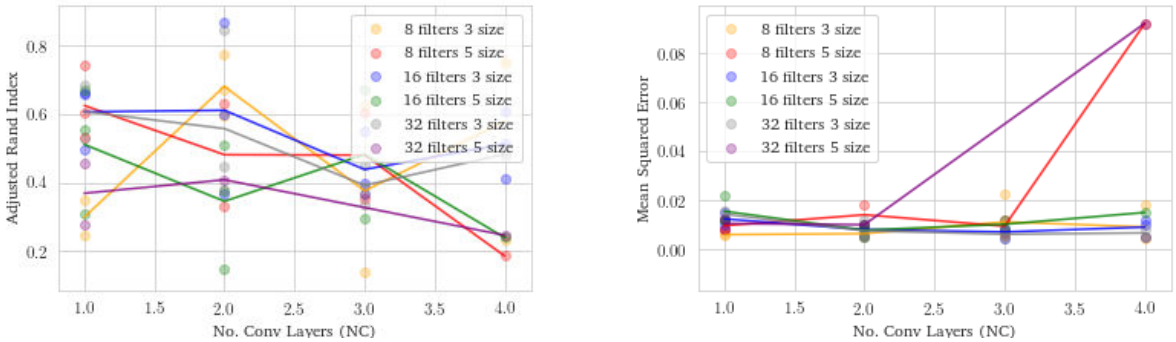


Fig. 5.6.: Architecture of encoder for experiments with variable convolutional layers. NF: the number of filters, NC: the number of convoluional layers, f: the filter size

Hyperparameter	Value
Experiments	number of conv layers, NC: 1,2,3 filter size, F: 3,5 n filters, NF: 8,16,32
No. runs each experiment	3
latent dimensions	30
optimizer	adam
learning rate	0.005
loss function	MSE
max epochs	500
early stopping	10
batch size	200
activation function	ReLU

Tab. 5.3.: Hyperparameters for convolutional layer experiments

The full table of results can be found in tab. C.2, and are summarized in fig. 5.7.



(a) Adjusted rand index **(b)** Mean squared error

Fig. 5.7.: Adjusted rand index (left) and Mean squared Reconstruction Error (right) on trials varying the number of convolutional layers, number of filters and filter size. Each point on the scatter plot is one full experiment. The lines represent the mean.

Looking at the results though, they don't look particularily good - in fact, they are worse than the best hidden layer model. After some thought, this isn't too surprising since what we effectively did by keeping the padding the same, using a stride of 1 and no pooling layers, is drastically increased the number of parameters of the model (around 600X in some cases). There are a couple of ways that convolutional layers can be used to reduce the size of the parameter space: usually by having a large filter size with no padding, which only slightly reduces the size per convolution, or adding stride to the convolution, or adding max pooling layers. So, first, let's try modifying the stride to be equal to the filter size in the next section (this choice is somewhat arbitrary, but does remove any potential checkerboarding effect in the deconvolutions, see [33] for examples of that).

5.1.4 Convolutional Stride - Network 4

This time we set the stride to be the same as the filter size, and otherwise basically reproduce the previous experiments. The architecture is shown in fig. 5.8 and the hyperparameters are listed in tab. 5.4 This time we go directly to 30 nodes in the first fully connected layer as the layer before has much fewer parameters compared to the non-strided network.

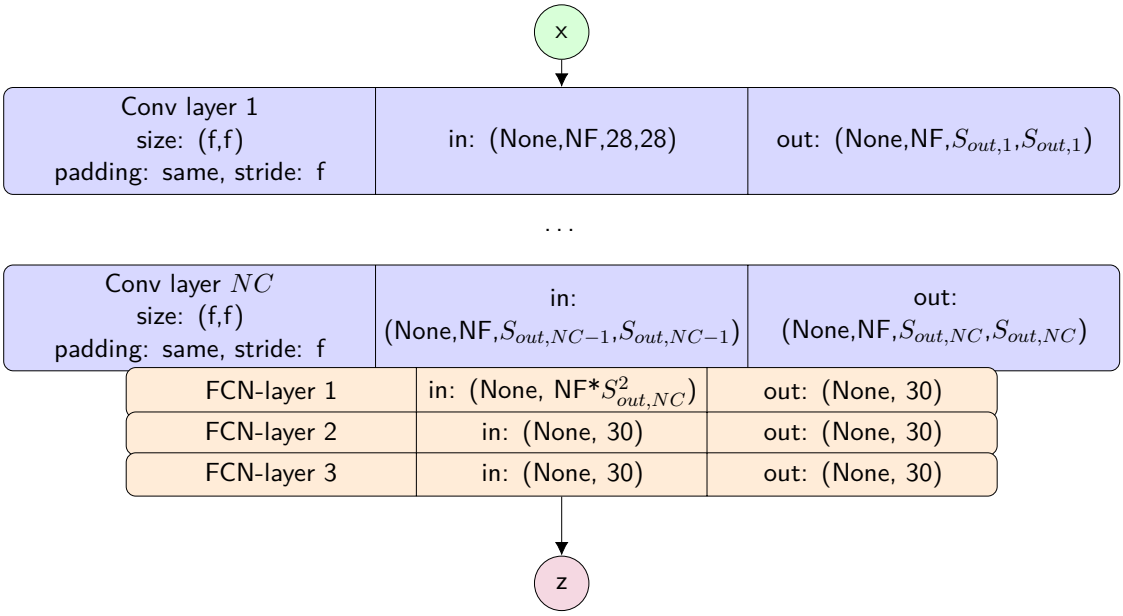


Fig. 5.8.: Architecture of encoder for experiments with variable convolutional layers with stride equal to filter size. NC: the number of convolutional layers, NF: the number of filters, F: filter size.

S_{out} is calculated differently depending on the filter size and number of convolutional layers (see (5.1)).

$$S_{out} = \frac{S_{in} - 2 * P - FS}{str} + 1 \tag{5.1}$$

where S_in is the dimension in, P is the padding, FS is the filter size, and str is the stride. In our case, we set padding to be $P = \frac{FS-1}{2}$, and stride to be equal to filter size, so this is simplified to:

$$S_{out} = \frac{S_{in} - 1}{FS} + 1 \tag{5.2}$$

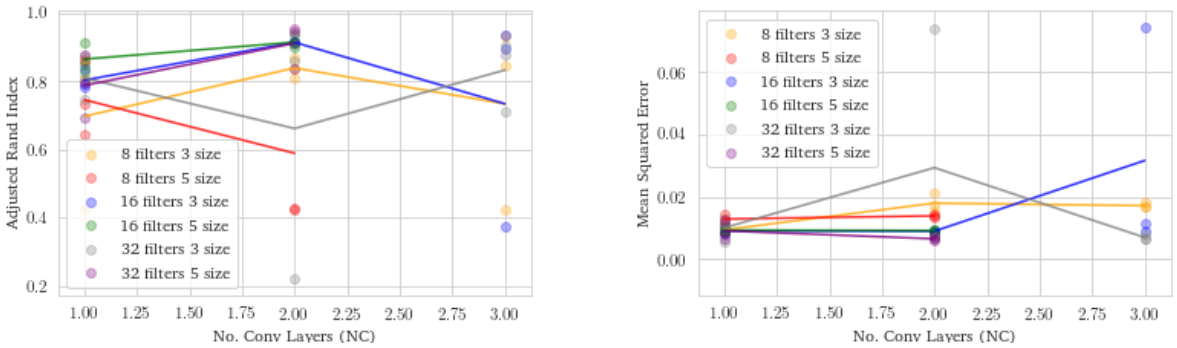
S_{out} for each layer then becomes S_{in} for the next layer. This is calculated recursively for each subsequent layer.

The following hyperparameters are used for the convolutional stride experiments:

Hyperparameter	Value
Experiments	number of conv layers, NC: 1,2,3 filter size/stride, FS: 3,5 n filters, NF: 8,16,32
No. runs each experiment	3
latent dimensions	30
optimizer	adam
learning rate	0.005
loss function	MSE
max epochs	500
early stopping	10
batch size	200
activation function	ReLU

Tab. 5.4.: Hyperparameters for convolutional layer experiments with stride equal to filter size

The full table of results can be found in tab. C.3, and are summarized in fig. 5.9.



(a) Adjusted rand index **(b)** Mean squared error

Fig. 5.9.: Adjusted rand index (left) and Mean squared Reconstruction Error (right) on trials varying the number of convolutional layers, number of filters and with a filter size equal to stride. Each point on the scatter plot is one full experiment. The lines represent the mean.

Now that we have drastically reduced the number of parameters in these convolutional models, these results now look much more convincing. In many trials the ARI is well above 0.8. Unfortunately, there is no obvious choice of number of filters or filter size here, but we notice that 2 convolutional layers, with 16 filters and a filter size of 3 seems to be a consistently good choice.

5.1.5 VGG-Like Networks - Network 5

Stride isn't the only way to reduce the parameters in a network - Max pooling layers are another popular choice, and a popular architecture that employs these is the VGG architecture [40]. Although we do not exactly replicate that here, we will explore the effect of adding multiple VGG-like blocks before our linear layers. The architecture for one VGG-like block is shown in fig. 5.10, with the full architecture shown in fig. 5.11 and the hyperparameters for the experiments are listed in tab. 5.5.

Fig. 5.10.: VGG-Like Block, each block halfs the size of the input.S_i: size of the input square, NF: number of filters, f: the filter size

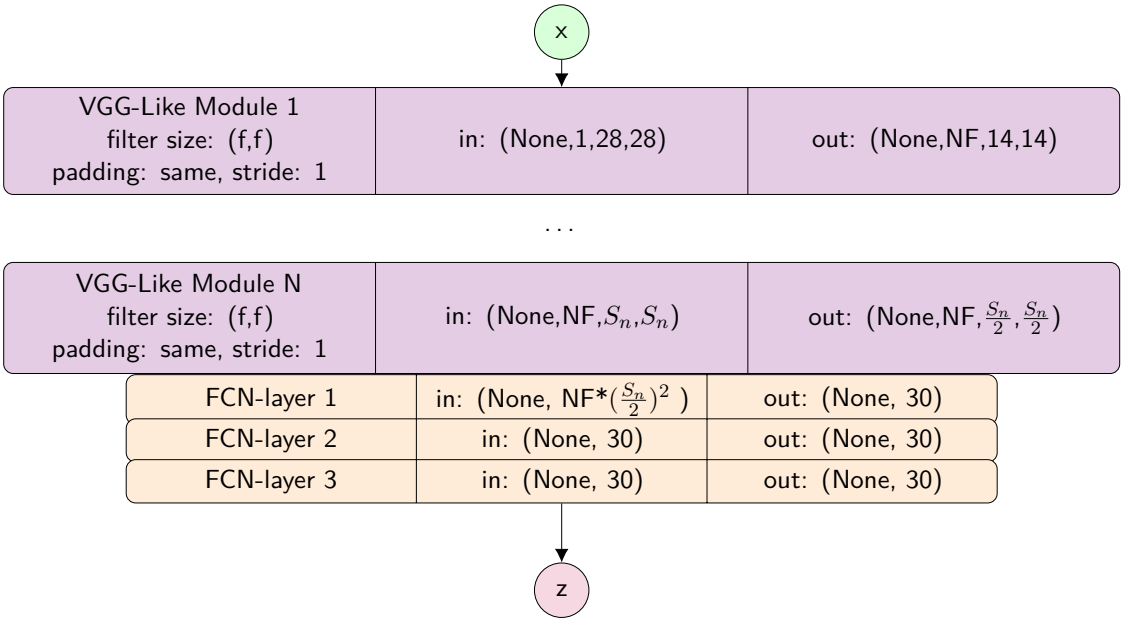
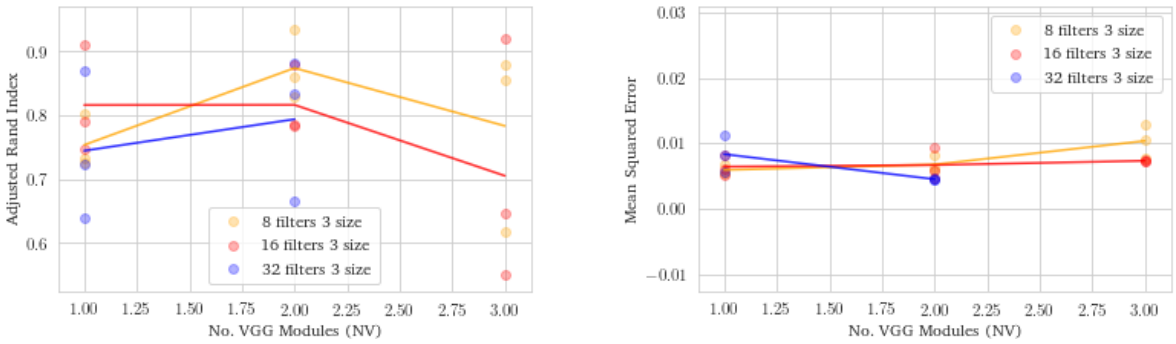


Fig. 5.11.: Architecture of encoder for experiments with variable number of VGG-like blocks. N: the number of vgg-like blocks, NF: the number of filters, f: the filter size

Hyperparameter	Value
Experiments	number of vgg-like blocks: 1,2,3 n filters: 8, 16, 32
filter size	3
No. runs each experiment	3
latent dimensions	30
optimizer	adam
learning rate	0.005
loss function	MSE
max epochs	500
early stopping	10
batch size	200
activation function	ReLU

Tab. 5.5.: Hyperparameters for hidden layer experiments

The full table of results can be found in tab. C.4, and are summarized in fig. 5.12.



(a) Adjusted rand index

(b) Mean squared error

Fig. 5.12.: Adjusted rand index (left) and Mean squared Reconstruction Error (right) on trials varying the number of vgg-like blocks and number of filters. Each point on the scatter plot is one full experiment. The lines represent the mean.

These results look very similar to the results we got using convolutional stride in sec. 5.1.4 above. The average ARI of the best model here (0.874)[‡] is however, slightly worse than the ARI of the best strided model (0.916)[§]. This seems counterintuitive since VGG is an iconic architecture famous for working well in image recognition. And indeed, looking closer at the results we can see that the VGG models have a lower MSE (0.0045 is the lowest average), than the strided models (0.00659 is the lowest average). This again reinforces our belief that we need to focus a bit more on the loss function and finding one that optimizes the ARI.

[‡]the best VGG model was: 2-vgg-block,8-filter, 3 filter-size
[§]the best strided model was: 2-conv-layer,16-filter, 5 filter-size

5.1.6 Final Choice of Architecture

Now we have tried a few different architectures and explored the effect of them on the MSE and ARI. While the number of hidden layers and features in our first trials seems to be directly connected to the ARI, the number of convolutional layers or VGG-like blocks didn't seem to have the same effect. Indeed, the model seemed to be most affected by having the convolutional layers at all, and not by their depth.

That said, to start exploring the effect of different loss functions on the cluster formation, we first need to choose an architecture. Both the VGG-like networks and strided convolutional networks performed well. Since the strided convolutional model seems to perform the best, and has 21303 trainable parameters compared to the vgg-model's 67775 parameters, based on occam's razor [¶] and the best ARI, we will explore the effect of different loss functions using a strided 2-conv-layer, 16-filter, 5 filter-size model.

[¶]Occam's razor is the theory that the simplest model/answer is usually the correct one, and therefore given two models with similar performance, we should choose the model with the least number of parameters.

5.2 MNIST - Choice of Loss Function and Regularization

Up until now we have been using MSE as our loss function. Here we compare the effect of different loss functions and regularizers on the clustering. To begin, we will look at a couple common choices that again are used to reduce the reconstruction error in autoencoders, namely binary cross entropy (BCE) and denoising. Then we will look at variational autoencoders (VAEs) with various priors on the latent space, which will allow us to try to directly optimize the latent space for clustering.

In the previous section we compared different architectures and settled on a 2-conv-layer, 16-filter, 5 filter-size, 3-hidden-layer model, so we will use that model for all experiments in this section. For reference, the model and its common hyperparameters are shown in fig. 5.13 and tab. 5.6:

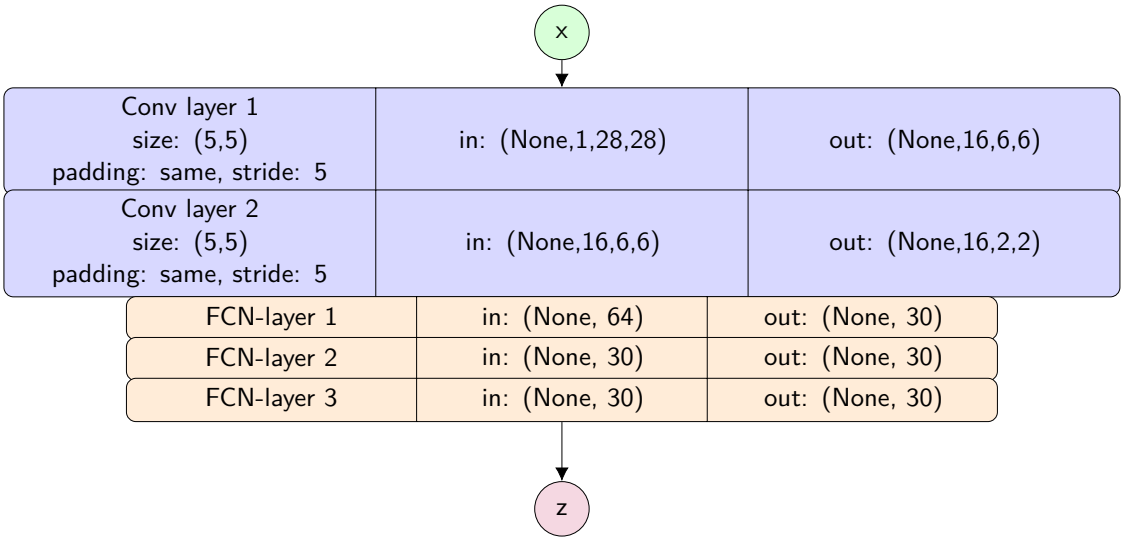


Fig. 5.13.: Final architecture to be used in loss function experiments.

Hyperparameter	Value
optimizer	adam
max epochs	500
early stopping	10
batch size	200
activation function	ReLU

Tab. 5.6.: Hyperparameters common to all the following loss function experiments

5.2.1 Mean Squared Error vs Binary Cross Entropy

To start, we look at the effect of changing the loss function to another popular choice for autoencoders, binary cross entropy (BCE). The hyperparameters for these experiments are shown in tab. 5.7.

Hyperparameter	Value
Experiments	loss function: MSE, BCE
No. runs each experiment	4
learning rate	0.001

Tab. 5.7.: Hyperparameters for BCE experiments

The full table of results can be found in tab. D.1, and are summarized in tab. 5.8.

Loss Function	Avg. Loss	No. Parameters	Mean ARI	Std ARI
BCE	0.10800	21300	0.889	0.0411
MSE	0.00963	21300	0.884	0.0635

Tab. 5.8.: **BCE Loss experiments** Summary of clustering results when comparing MSE and BCE loss functions

Although the results of the individual runs varies (from 0.824 ARI to 0.954), the average for BCE and MSE are very similar (0.889 and 0.884, respectively) and indicate that there is no significant effect of using BCE vs MSE on the cluster formation, given this architecture. We can assume this probably generalizes to other architectures as well, as both BCE and MSE are primarily concerned with producing good reconstructions, and theoretically have no direct effect on cluster formation in the latent space.

5.2.2 Denoising

Next, we will look at the effect of another popular approach used to reduce reconstruction error, namely denoising autoencoders.

Hyperparameter	Value
Experiments	gaussian noise: 0.2, 0.5, 1 sigma masking noise: 20, 50, 90 % masking noise
No. runs each experiment	4
learning rate	0.001
loss function	MSE

Tab. 5.9.: Hyperparameters for loss function experiments

The full results for these experiments can be found in D.2, and the summary below in 5.10.

Denoising Function	Sigma or P[a]	Mean Loss	No. Parameters	Mean ARI	Std ARI
No Noise	0.0	0.0122	21300	0.870	0.0900
Masking Noise	0.5	0.0103	21300	0.826	0.0965
Masking Noise	0.2	0.0146	21300	0.774	0.241
Gaussian Noise	0.1	0.0125	21300	0.740	0.164
Masking Noise	0.9	0.0150	21300	0.567	0.176
Gaussian Noise	0.2	0.0127	21300	0.552	0.255
Gaussian Noise	1.0	0.0224	21300	0.435	0.191
Gaussian Noise	0.5	0.0152	21300	0.306	0.113

Tab. 5.10.: **Denoising experiments** Summary of results showing the effect of denoising on cluster formation for our architecture

[a]For masking noise, this is P, the probability that any one pixel is masked, for gaussian noise this is Sigma, the standard deviation of the gaussian noise added to each pixel

From this we can gather that the denoising models which can improve normal autoencoder reconstruction, seem to hinder clustering, with the no-noise model having the highest average ARI, and large noise parameters reducing the ARI significantly.

5.2.3 Variational Autoencoders

Next, since VAEs have a loss function which directly evaluates the distribution of the latent space, we suspect they might be more successful in encouraging clustering than normal approaches to reducing reconstruction error in autoencoders.

To start investigating this, we look at a normal VAE with a single zero-mean gaussian prior. We then compare this with a single zero-mean laplacian prior and with GMM priors with two different initialization schemes for the means: Randomly initialized at the start and continually updated using k-means (see sec. 4.2.2).

Note on choosing latent space in variational models Before running the experiments, we should note that there are two ways of representing the latent space in variational models: since the output of the encoder gives the means and variances of the posterior which we then sample from, you could either choose to represent the latent space as the by this mean vector, or by all the samples sampled from the posterior. Initially, we choose to use all the samples sampled from the posterior, as at this point it isn't clear how we would represent the means for the GiMMPPy model of the posterior - if the posterior has equal Gaussian component weights, we might want to include all means for each input. On the other hand, if the Gaussian component weights are quite skewed - it always chooses one Gaussian component, then we would only want to include that mean in the latent space. Therefore, for simplicity, we use all the samples from the posterior. Later we will explore what the component weights look like and if we can justify using the means.

Hyperparameter	Value
Experiments	Distributions: Gaussian, Laplacian, GMM GMM initializations: Random, K-Means Continual
No. runs each experiment	3
learning rate	0.0001
loss function	ELBO, ELBO-laplace or GiMMP, based on disribution used

Tab. 5.11.: Hyperparameters for VAE cluster formation experiments

The full results for these experiments can be found in D.3, and the summary below in 5.12.

Prior Distribution	Initialization	Mean Loss	No. Parameters	Mean ARI	Std ARI
GMM	KMeans	1080	29022	0.946	0.00158
Laplace	Zero Mean	99	29022	0.885	0.0245
Gaussian	Zero Mean	124	29022	0.615	0.152
GMM	Random	1020	29022	0.599	0.0868

Tab. 5.12.: **Variational experiments** Summary of results showing the effect of different priors and prior initialization on cluster formation using VAE architecture

The results in tab. 5.12 are quite encouraging - by changing the prior distribution and prior initialization we can allow the model to encourage clustering. The GMM-KMeans model results are especially good, since they are consistently high (with a low standard deviation across different runs). The downside of this model is that it requires you to know how many clusters/gaussians there should be before running the model. In the case of MNIST we know exactly how many we are interested in, but other datasets are not so clear. The laplace model also performs rather well, which is particularily good because it does not require any assumptions about the number of clusters during training - it just naturally allows the model to spread out.

5.3 Maculinea Alcon - Experiments on Architecture

Finally, we have a solid basis to begin looking at the Maculinea Alcon dataset. We now have some insight on the effect of architecture and loss function on clustering from the trials with MNIST, and we can use these to direct the analysis on this dataset. That said, the tasks are not exactly the same, so we do start by repeating some of the experiments. Particularly, we repeat architecture experiments as our image complexity has now changed significantly, so we suspect the model will need to be larger to account for this.

Once again, we try to start with a simple solution - we start with the same architecture we found using MNIST. Then we will modify the architecture slightly and note the effect on the clustering. We first make some key choices:

1. **MSE Loss function**: Although MSE was not the best loss function from the MNIST trials, it was an order of magnitude faster to run, allowing us to complete many more trials. We also assume that the effect of the architecture on the clustering is mostly independant of the loss function, and this is reinforced by the fact that reconstruction loss is also used in the VAE model, so finding the MSE architecture with the best clustering should also translate to finding the VAE architecture with fairly good clustering.
2. **Start with MNIST architecture**: For simplicity, we start with the MNIST architecture. Unfortunately, to use this, we need to resize the butterfly images to be 28x28, which is really low resolution, but we explore adding convolutional layers to this architecture which we already know works somewhat well, to allow us to account for the larger image size. To keep more in line with the MNIST architecture, we keep the second last convolutional layer to have 28×28 input (see fig. 5.14).
3. **Gender as clustering criteria**: We decide to use gender as the clustering criteria. Gender is a simple prediction to start with, as it has three discrete groups (Female, Male, or Unknown), and we know from the biologists that sex-determination can be done effectively using the coloring on the top-side of the butterfly, meaning our model should be able to learn this if we use the top-side images. We also know from preliminary experiments that the autoencoder tends to naturally cluster the topside butterfly images by gender, so this is a natural choice.
4. **Batch size of 20**: We decrease the batch size from 200 for MNIST to 20, as we have much fewer training examples.
5. **Early stopping of 15**: We increase the early stopping criteria to 15 because we expect the model parameters to be more complex to learn and want to allow the model to have ample time to learn them.

5.3.1 Experiments on Convolutional Depth - Network 1

To start, we decide to first explore the effect of convolutional depth on the results, since we expect that we will need more convolutions to account for the increased resolution and added complexity. The architecture used in these experiments is shown in fig. 5.14.

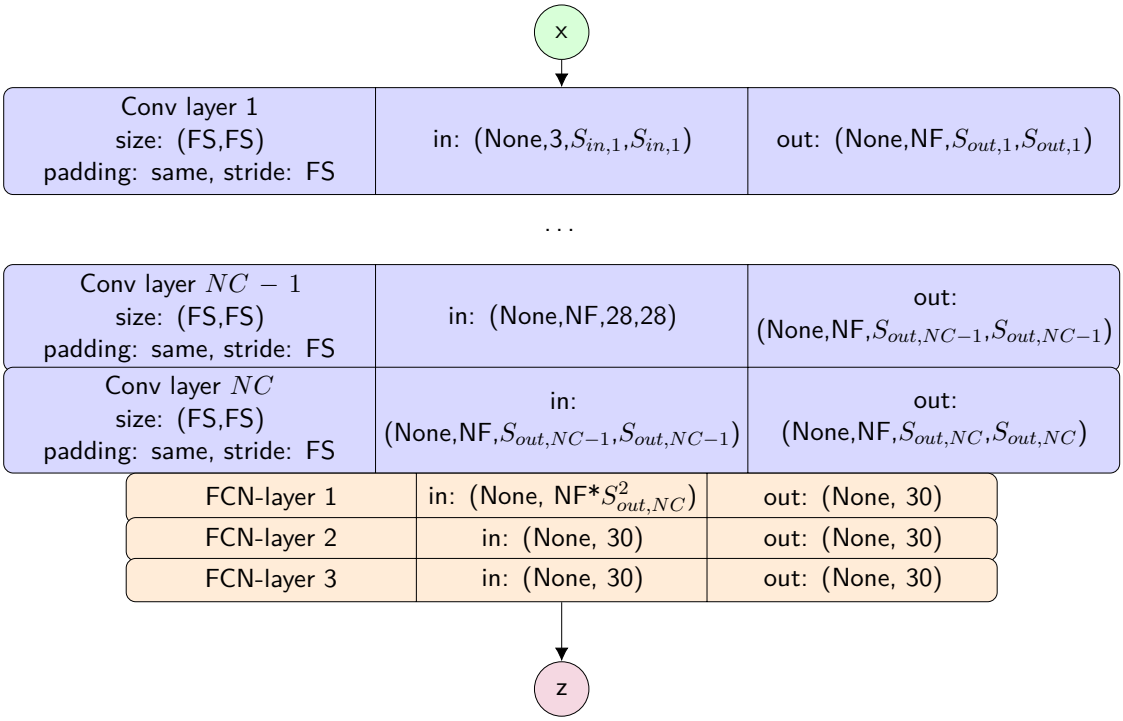
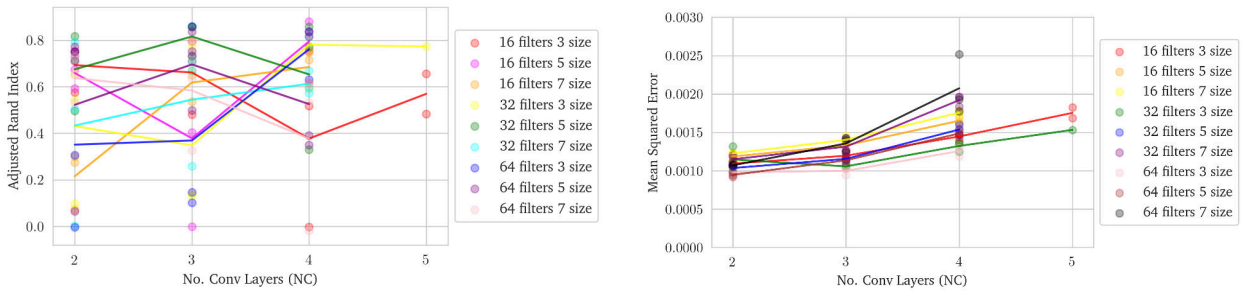


Fig. 5.14.: Initial architecture for convolutional Maculinea Alcon experiments. NC: number of convolutional layers, NF: number of filters, FS: filter size, S_{in} and S_{out} calculated using (5.2), such that conv layer NC-1 has an input size of 28×28,like in the MNIST experiments.

Hyperparameter	Value
Experiments	NC: 2,3,4 NF: 16, 32, 64 FS: 3,5,7
No. Runs each Experiment	3
Latent Dimensions	30
Optimizer	adam
Learning Rate	0.001
Loss Function	MSE
Max Epochs	500
Early Stopping	15
Batch Size	20

Tab. 5.13.: Hyperparameters for initial Maculinea Alcon experiments

The full table of results can be found in tab. E.1, and are summarized in fig. 5.15 and tab. 5.14.



(a) Adjusted rand index

(b) Mean squared error

Fig. 5.15.: Adjusted rand index (left) and Mean squared Reconstruction Error (right) on trials varying the number of convolutional layers, number of filters and with a filter size equal to stride. Each point on the scatter plot is one full experiment. The lines represent the mean.

Similarly to the convolutional experiments with MNIST, there are some models which seem to consistently perform better than others, however, there is no obvious trend relating to filter size, number of filters, or number of convolutional layers. That said, based on these results we choose to continue with a 3-conv-layer, 32-filter, 5-filter-size network as this performs consistently well. Some of the best and worst results are summarized in tab. 5.14 below.

NC	NF	FS	(S_{in},S_{in})	No. Parameters	Mean ARI	Std ARI
3	32	5	(136,136)	118921	0.816	0.0727
4	16	5	(676,676)	48569	0.795	0.0739
4	32	3	(244,244)	92233	0.781	0.0329
4	64	3	(244,244)	291305	0.761	0.113
3	64	5	(136,136)	438889	0.696	0.175
				...		
3	32	3	(82,82)	73737	0.321	0.393
2	16	7	(28,28)	34569	0.216	0.124

Tab. 5.14.: **Convolutional Alcon experiments** Summary of results showing the effect of convolutional layers on cluster formation for our architecture. NC: number of convolutional layers, NF: number of filters, FS: filter size, (S_{in},S_{in}) is input image size

5.3.2 Experiments on No. Hidden Layers - Network 2

Next, we try varying the number of fully connected layers and look at the result. We max the latent features out at 128 since this is the number of nodes in the last output of the convolutional layer. The architecture used in these experiments is shown in fig. 5.16.

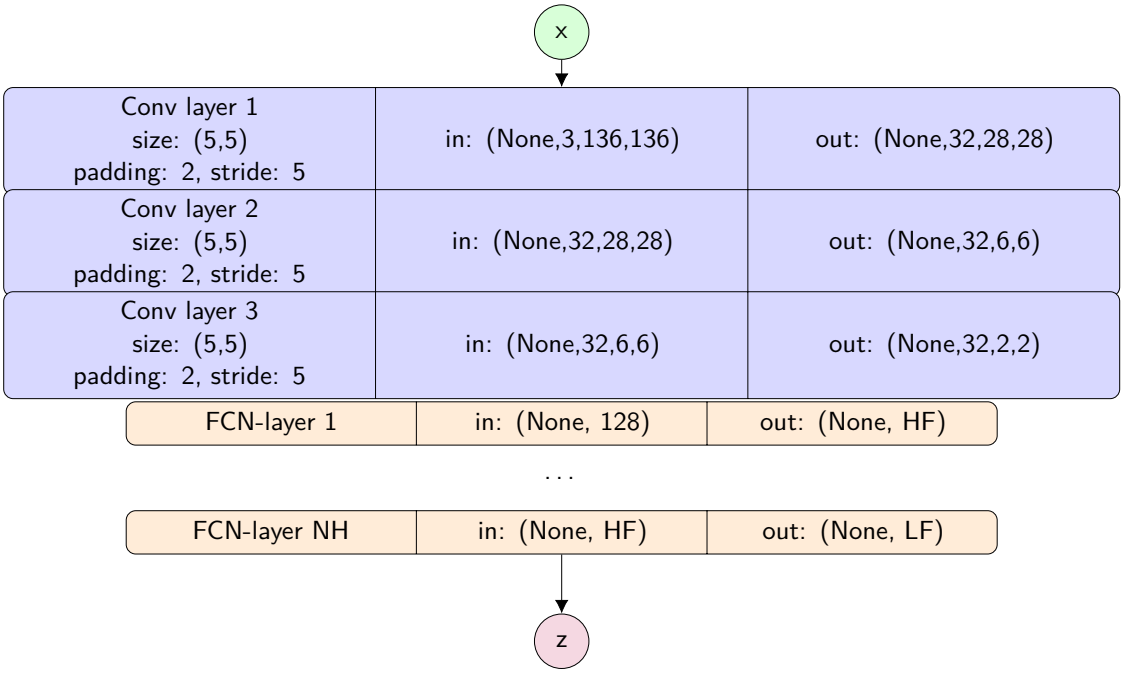
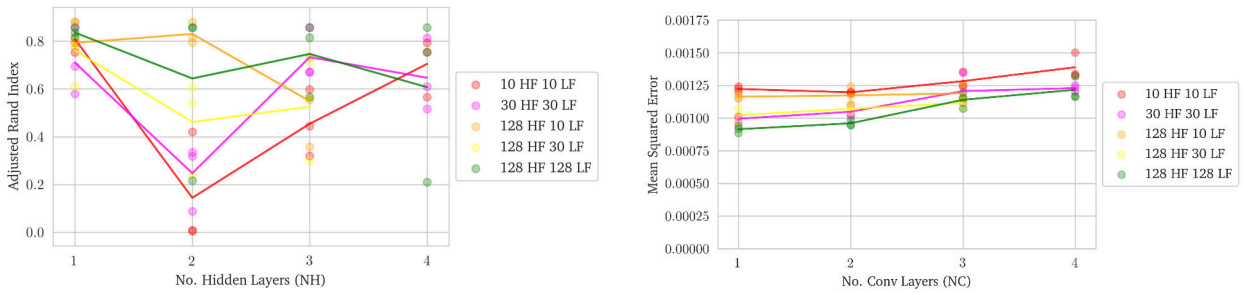


Fig. 5.16.: Architecture for Maculinea Alcon hidden layer experiments. NH: number of hidden layers, HF: number of hidden features, LF: number of latent features.

Hyperparameter	Value
Experiments	NH: 1,2,3,4 HF: 10,30,100,128 LF: 10,30,100,128
No. runs each experiment	3
Optimizer	adam
Learning Rate	0.001
Loss Function	MSE
Max Epochs	500
Early Stopping	15
Batch Size	20

Tab. 5.15.: Hyperparameters for initial Maculinea Alcon experiments

The full table of results can be found in tab. E.2, and are summarized in fig. 5.17 and tab. 5.16.



(a) Adjusted rand index

(b) Mean squared error

Fig. 5.17.: Adjusted rand index (left) and Mean squared Reconstruction Error (right) on trials varying the number of hidden layers. HF: Hidden Features, LF: Latent Features. Each point on the scatter plot is one full experiment. The lines represent the mean.

Although the results are quite varied, there seems to be one clear trend which is that the model performs best with only one hidden layer. This is quite surprising as we would expect more depth to improve the model results. However, greater depth can also make it more difficult for the model to learn, and this could be the case here. Some of the best and worst results are summarized in tab. 5.16 below.

NH	HF	LF	No. Parameters	Mean ARI	Std ARI
1	-	128	140387	0.836	0.0217
2	128	10	140085	0.830	0.0447
1	-	100	133191	0.821	0.0124
2	100	100	153391	0.815	0.0638
1	-	10	110061	0.808	0.0653
			...		
2	30	30	117061	0.247	0.139
2	10	10	110281	0.144	0.238

Tab. 5.16.: **Hidden Alcon experiments** Summary of results showing the effect of convolutional layers on cluster formation for our architecture. NC: number of convolutional layers, NF: number of filters, FS: filter size, (S_{in},S_{in}) is input image size

5.3.3 Experiments on Loss Function

Now that we have an architecture which appears to work well for clustering, we explore the effect of different loss function on clustering. Here we focus on comparing loss functions which produced good results on MNIST - namely MSE, BCE, the LiPPy loss function, and GiMMPPy loss function with EEK-means initialization. For these experiments, we use the best architecture determined by the previous experiments, reiterated in fig. 5.18. The hyperparameters for these experiments are listed in tab. 5.17

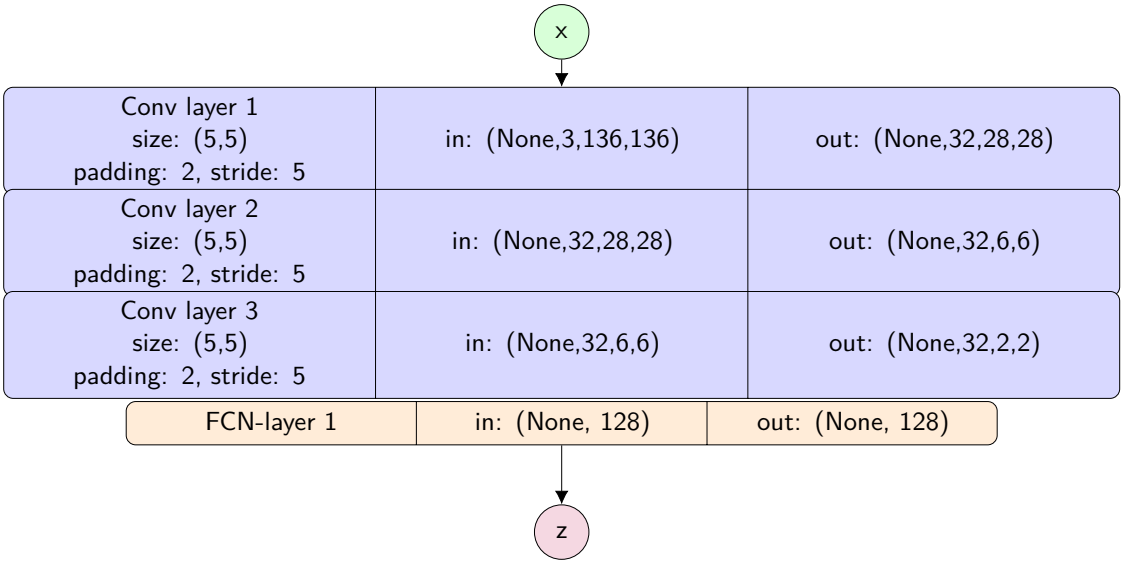


Fig. 5.18.: Architecture for Maculinea Alcon loss experiments.

Hyperparameter	Value
Experiments	MSE loss BCE loss LiPPy loss GiMMPPy EEK-means loss
No. runs each experiment	4
No. VAE samples	10
No. MC samples	100
optimizer	adam
learning rate	0.001 and 0.0001 for VAE
max epochs	500
early stopping	15
batch size	20

Tab. 5.17.: Hyperparameters for Maculinea Alcon loss experiments

The full table of results can be found in tab. E.3, and are summarized in tab. 5.18.

Loss	No. Parameters	Mean ARI	Std ARI
MSE	140387	0.836	0.0217
BCE	140387	0.759	0.0968
GiMMPPy EEK-means VAE	190181	0.734	0.0249
LiPPy VAE	157028	0.722	0.0105

Tab. 5.18.: **Alcon loss experiments** Summary of results showing the effect of loss function on cluster formation for our architecture.

The results of the loss experiments shown in tab. 5.18 are surprising as they indicate that the normal MSE autoencoder consistently outperforms our new LiPPy and GiMMPPy models, which performed (albeit marginally) better in the MNIST trials. This could indicate that the dataset is not as well described by a GMM, or it could indicate that the architecture we have chosen, with more convolutional layers and fewer fully connected layers does not work as well for the GiMMPPy and LiPPy loss functions. In any case, we want to further explore how these loss functions affect the clustering, and see if we can tweak them to be improved. To begin this exploration, let's return to the question of how we represent the latent space in variational models.

5.3.4 Choice of Variational Latent Space

Recall that the encoder of a variational model outputs a mean and variance of the posterior for each image, which is then sampled from and fed into the decoder. Up until now, we have been using all the samples from the posterior as our latent space. Now, we will see if we can use the mean of the posterior instead. This should be valid for our single-component loss functions, like the LiPPy loss function. However, for the GiMMPPy model which has multiple means, we need a way to first choose the correct mean, or somehow represent all the means, weighted by their probability of being chosen. So first, let's look at what the Gaussian component weights look like in our trained GiMMPPy models, as shown in tab. 5.19.

Loss	**ARI**	**Avg** w_1	**Avg** w_2	**Std** w_1	**Std** w_2
GiMMPPy	0.77	0.001	0.999	0.001	0.001
GiMMPPy	0.72	0.999	0.001	0.001	0.001
GiMMPPy	0.73	0.999	0.001	0.001	0.001
GiMMPPy	0.72	0.972	0.028	0.009	0.009

Tab. 5.19.: Mean and standard deviation of component weights for GiMMPPy models on Maculinea Alcon dataset

In table 5.19 we can see that in all of the models, one of the Gaussian component weights is consistently close to 1, and the other consistently close to 0. This suggests that the GMM posterior is collapsing into one Gaussian. This makes some sense, as we could expect that the latent space responsible for producing a single given image could be Gaussian distributed, instead of GMM-distributed, even while the prior on the whole dataset is a GMM. To take a deeper look at this, let's examine the weight distributions for the MNIST models, given in tab. 5.20.

Loss	ARI	Avg w_1	Avg w_2	Avg w_3	Std w_1	Std w_2	Std w_3
GiMMPPy	0.94	0.814	0.186	0.001	0.204	0.204	0.001
GiMMPPy	0.95	0.001	0.999	0.001	0.001	0.001	0.001
GiMMPPy	0.94	0.999	0.001	0.001	0.001	0.001	0.001
GiMMPPy	0.94	0.001	0.256	0.743	0.001	0.318	0.318
GiMMPPy	0.95	0.001	0.001	0.998	0.003	0.001	0.004

Tab. 5.20.: Mean and standard deviation of component weights for GiMMPPy models on MNIST dataset

In tab. 5.20 we can see that in three of five experiments the posterior collapsed into one, and in the other two experiments the three component posterior collapsed into two components, meaning that the posterior does not always collapse into one component. To explore the effect this has on the resulting ARI, we can run the GiMMPPy loss function with a single component posterior instead and compare. The results are summarized in tab. 5.21 below. From this we can see that having 3 gaussian components does seem to improve the clustering results, although only a little.

Loss	No. Posterior Components	Mean ARI	Std ARI
GiMMPPy	3	0.946	0.00506
GiMMPPy	1	0.939	0.00245

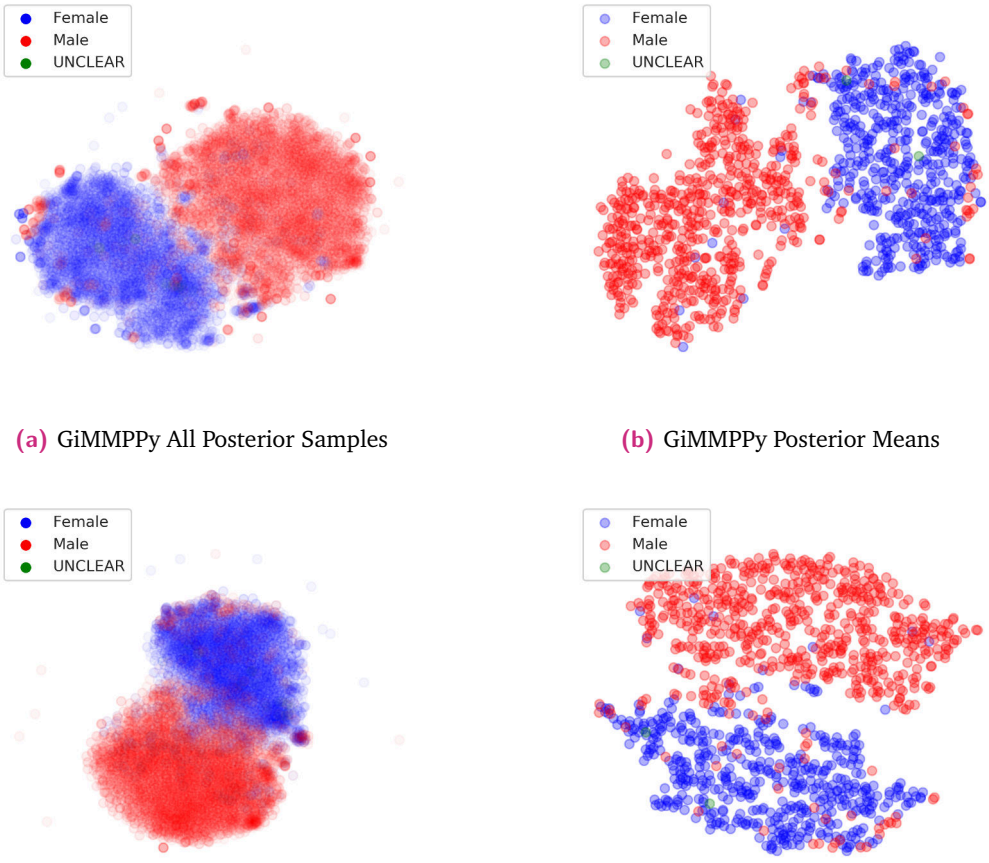
Tab. 5.21.: Average ARI for GiMMPPy models with 3 or 1 Gaussian components in the posterior for MNIST 1-2-9 subset with 3 ground truth clusters.

Now we know that the GMM posterior does not always collapse into one component, however, we also know that it did for all our Alcon experiments. Therefore for these experiments it should be valid to represent the latent space by the component mean with the highest component weight. Therefore let's do that and rerun our loss experiments on the Maculinea Alcon data, but this time using the mean of the gaussian component with the highest weight as the datapoint in latent space. We compare this with our results from tab. 5.18 in tab. 5.22.

Loss	Latent Space	No. Parameters	Mean ARI	Std ARI
MSE		140387	0.836	0.0217
GiMMPPy EEK-means VAE	Posterior Means	190181	0.832	0.0106
LiPPy VAE	Posterior Means	157028	0.816	0.0307
BCE		140387	0.759	0.0968
GiMMPPy EEK-means VAE	All Posterior Samples	190181	0.734	0.0249
LiPPy VAE	All Posterior Samples	157028	0.722	0.0105

Tab. 5.22.: **Alcon loss experiments** Summary of results showing the effect of loss function on cluster formation for our architecture.

Here we find that using the mean of the posterior with the highest component weight produces better results than using all the posterior samples. In fact, the results are almost as good as the best model, using MSE loss. This also makes some sense, as by using all the samples, we are essentially just adding more variance to the clusterings, and allowing them to spread out (and into eachother) more. We can then further explore this by using t-SNE to visualize the latent space, and thereby gain some intuition as to why this may be. The relevant plots are shown in fig. 5.19. Even though the left and right visualizations come from the same experiment, we can visually see that the right (latent space built from posterior means) creates more visually distinct clusters, and we assume this can be extrapolated to the higher dimensions as well.



(a) GiMMPPy All Posterior Samples

(b) GiMMPPy Posterior Means

(c) LiPPy All Posterior Samples

(d) LiPPy Posterior Means

Fig. 5.19.: 2D t-SNE visualization of latent space comparing having all posterior samples and only posterior means. (a) and (b) are from the same GiMMPPy experiment, (c) and (d) are from the same LiPPy experiment. Colors represent ground truth labellings, not classifications.

5.4 Maculinea Alcon Results - Pre analysis

Now that we have some models which seem capable of clustering the images in an unsupervised way, we can begin to interpret these results. First though, we want to complete a few more experiments - namely until now we have only been using the upperside (dorsal side) images of the butterflies for simplicity. We would like to see the effect of images of the underside (ventral side) of the butterfly, and also the results of combining the dorsal and ventral images into one 6-channel input.

Although we recognize that the MSE loss function performed best on gender, we continue to also use the LiPPy and GiMMPPy loss functions, partially because we want to explore the effect these loss functions have in more detail, and partially because results using the posterior means were comparable to that of MSE.

5.4.1 Dorsal, Ventral, and Combined Inputs

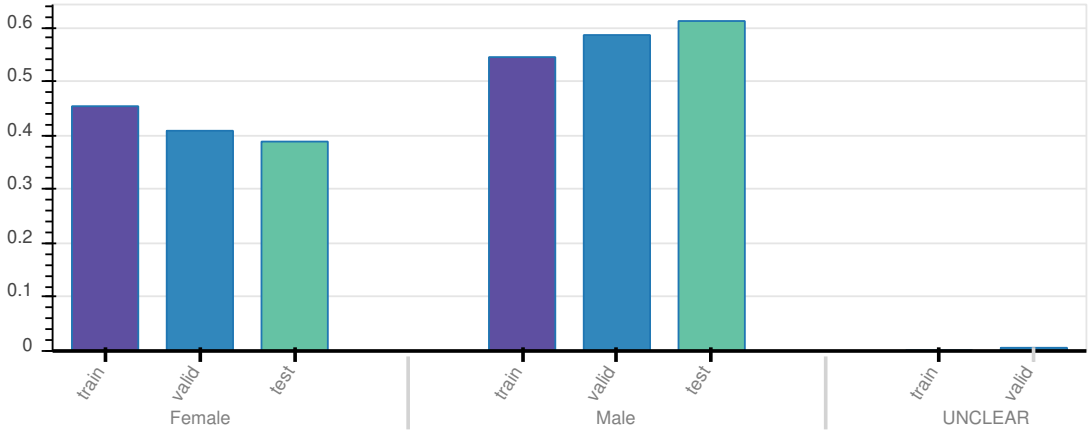
Let's look at the results of using different inputs to complete the clustering - namely using dorsal images only, ventral images only, and a 6-channel combination of both dorsal and ventral images. We also, now that we have finalized the model, finally take out our test set and see how our models perform on that. A full table of results is provided in tab. F.1 and the summarized results are displayed in tab. 5.23.

		Validation Set			Test Set		
Input	**Model**	**Mean ARI**	**Std ARI**	**Best ARI**	**Mean ARI**	**Std ARI**	**Best ARI**
Combined	MSE	0.892	0.0215	0.920	0.603	0.0404	0.635
Dorsal	MSE	0.836	0.0217	0.858	0.704	0.0216	0.711
Dorsal	GiMMPPy EEK	0.832	0.0105	0.837	0.700	0.0111	0.710
Dorsal	LiPPy	0.815	0.0306	0.837	0.705	0.0100	0.710
Combined	LiPPy	0.800	0.0498	0.831	0.559	0.0683	0.602
Combined	GiMMPPy EEK	0.689	0.227	0.809	0.559	0.166	0.654
Ventral	GiMMPPy EEK	0.00518	0.00262	0.0212	0.0180	0.00873	0.0212
Ventral	MSE	0.00403	0.0111	0.0204	0.0738	0.0153	0.0248
Ventral	LiPPy	0.00967	0.00698	0.0150	0.0147	0.00648	0.0144

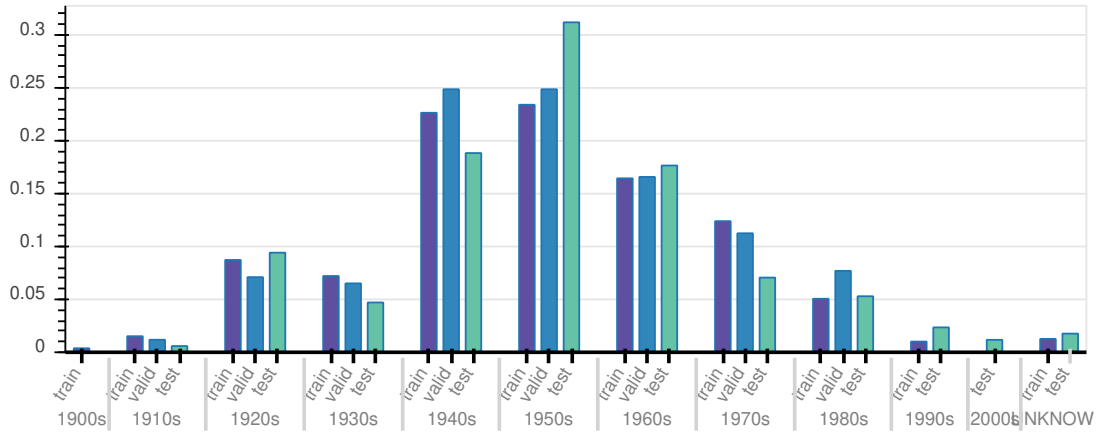
Tab. 5.23.: Dorsal/Ventral/Combined results on validation and test set: Summary of results showing the effect of using Dorsal/Ventral/Combined input images on gender-based cluster formation for our architecture on both the validation and test sets.

There are a few interesting results here. First, it appears that the **dorsal images are well correlated with gender**, but the ventral images do not seem to produce any obvious gender-related clusterings.

Next, we also notice that the **validation set and test set show signicantly different results**. This could indicate that we have overtrained our model on the validation set and/or that the test set and validation set have different distributions. When we first split the sets, we only randomly shuffled them, but did not ensure the distribution of gender, country etc in each set was as equal as possible. To investigate what is happening here, let's first look at the distribution of data across the different sets in fig. 5.20 to see if that is the problem.

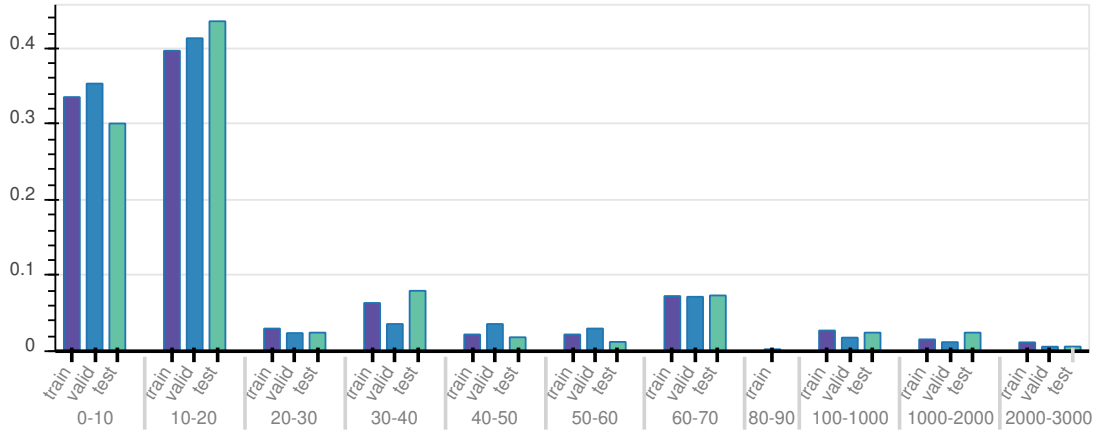


(a) Gender

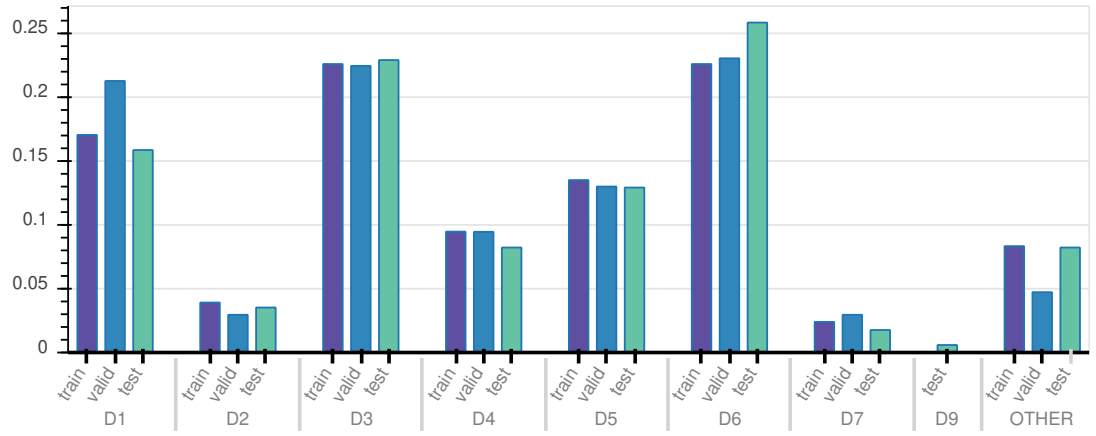


(b) Decade

Fig. 5.20.: Distribution of Maculinea Alcon dataset across training, validation and test sets for different classifications.



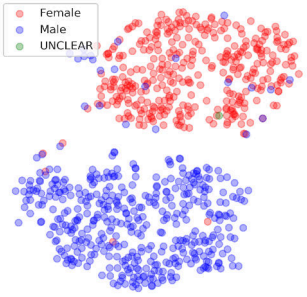
(c) Altitude



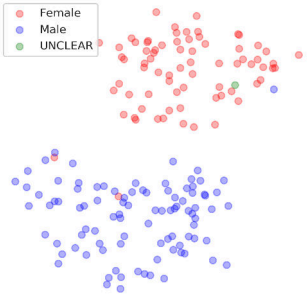
(d) Kaaber Region

Fig. 5.20.: Distribution of Maculinea Alcon dataset across training, validation and test sets for different classifications.

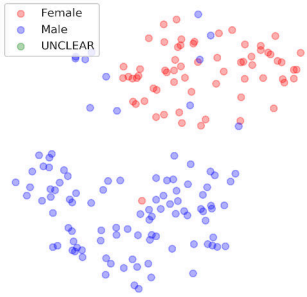
From fig. 5.20 we can see that the distributions are similar across datasets, at least with regards to the features we know. This indicates that there might be other factors which we are not accounting for - for example, the test set could contain an inordinate number of damaged specimens. However, perhaps we can get some idea by looking at the t-SNE visualization of the latent space across the different sets. This can be found in fig. 5.21.



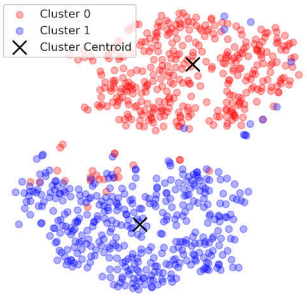
(e) Train Set - Ground Truth



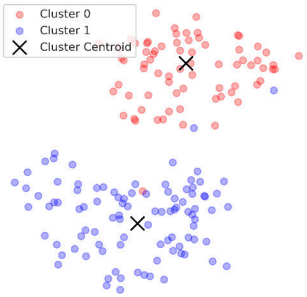
(f) Val Set - Ground Truth



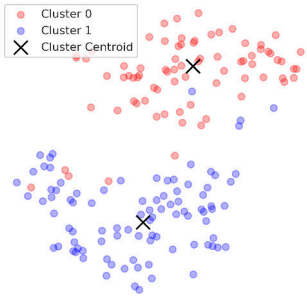
(g) Test Set - Ground Truth



(h) Train Set - K-Means Classification



(i) Val Set - K-Means Classification



(j) Test Set - K-Means Classification

Fig. 5.21.: t-SNE visualization of latent space for Combined MSE model. Top: ground truth gender classification. Bottom: K-means unsupervised cluster classification. For train (left), validation (middle) and test (right) sets. To make the plots comparable, t-SNE was completed first using the whole dataset, then the dataset was seperated to plot them individually.

From the t-SNE visualization in fig. 5.21 we can see that there are a number of male specimens, especially in the test and train sets, which the k-means misclassifies. This doesn't tell us too much though, so let's look specifically at the misclassifications in fig. 5.22.

Fig. 5.22.: Dorsal and Ventral images of specimens misclassified by gender in the test set using the MSE Combined Dorsal-Ventral model.

From fig. 5.22, we can indeed see that many of the misclassified specimens in the test set have oddities, such as very light wings, very small wings and bodies, discoloration (probably due to age of the specimen), or damage to the edges of the wings. It could be that these are randomly disproportionatly represented in the test datasets, which would help explain why the test set results are so poor. The best way to verify this would be to compare the number of such specimens here to the distribution across the dataset, but in the interests of time we do not do this.

5.5 Analyzing the data features

Since we now have an explanation as to why the test results and validation results differ so much, we can finally begin to look at the results themselves, and see if we can see any distinct correlations in the dataset. During this process, we decide to continue with comparing models with all 3 inputs (Dorsal, Ventral, Combined) and all 3 loss functions (MSE, LiPPy, GiMMPPy). For the variational models we also show the results using all samples, and using the posterior means.

Completing the exploration of the latent space is however, difficult, as it has 128 dimensions. Therefore we approach it in a few ways:

1. Visually inspect the results using **t-SNE** to give us some intuition about how the dataset is distributed in the latent space.
2. Train a simple **classifier/regressor** fully connected network on the latent space and the features of interest (gender, location, year, altitude), and see how well the network is able to learn from the models based on the input. If, for example, the network can easily classify latitude based on the dorsal images, this is an indication that the dorsal side contains some indication of latitude, even if this isn't obvious from the t-SNE.
3. Visually inspect the **reconstructions** made by taking the average latent space of a class of the dataset. For example, taking the average latent space of the Female specimens, then running this through the decoder and looking at the reconstruction, and comparing it with that of the other classes.

The full results for these (the ARI on unsupervised clustering of gender for all the models, the classification results per feature, and the t-SNE of the latent space for each model) are shown in appendix F. The reconstructions per feature are shown in appendix G. These results are discussed in more detail in the sections that follow.

5.5.1 Visual Inspection using t-SNE

T-SNE plots for the best experiments of each type for all features are provided in appendix F. Here we will highlight a few observations:

1. The models which appear to produce the most distinct clusters for gender are the Dorsal-input MSE-loss model (fig. F.1) and the Combined-input MSE-loss model (fig. F.3).
2. The Combined-input MSE-loss model appears to produce 4 distinct clusters (see fig. F.3), two sets of these are clearly based on gender, however, the distinguishing feature between the other two sets is not clear, and does not seem to be explained by the country, year, altitude, longitude or latitude.
3. For the variational models, the posterior means plots produce much more distinct clusters than the plots with all the samples from the posterior. (see, for example, figs. F.4 and F.5)
4. The Ventral plots show some gradient in longitude and latitude. See figures F.2,F.7,F.13
5. The country does not appear to be entirely random, with the international specimens seeming to appear more frequently on the outer edges of the clusters (see fig. F.1).
6. The ventral plots show no obvious clusterings.
7. The combined plots show features of both the dorsal and ventral plots - they usually produce two distinct clusters for male and female, but also show some variation gradient across the regions.

5.5.2 Classification/Regression Results

The classification/regression results give us a more objective view on what the models are able to learn. To complete this analysis, we split the features into two groups: those which are most easily represented as classification problems (gender and country), and those most easily represented as regression problems (year, altitude, longitude, latitude). The full results for these tests are shown in tables F.2 and F.3 respectively. Similar to the t-SNE results above, we only show the results from the best experiment of each model configuration. For these, a 2-layer FCN (shown in fig. 5.23) was used, trained with the hyperparameters given in tab. 5.24. Because the distribution of labels is quite skewed in many cases, we provide a 'random accuracy', which is the accuracy of randomly permutating the ground truth labels. We can then more easily see if the model is actually learning something, or is just learning the distribution of labels.

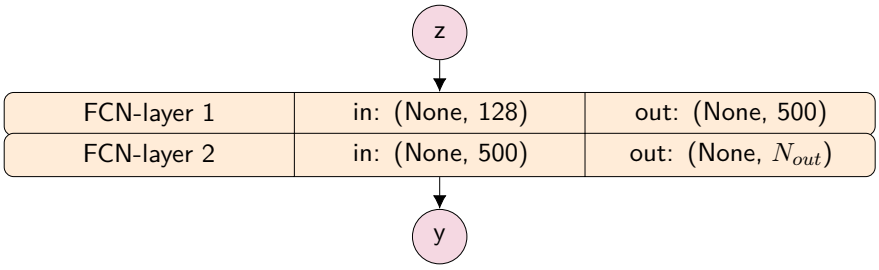


Fig. 5.23.: Fully Connected Classifier/Regressor architecture. If the FCN is a classifier, N_{out} is the number of classes, and softmax activation is used before the class prediction. If the FCN is a regressor, $N_{out} = 1$.

Hyperparameter	Value
No. runs each experiment	1
optimizer	adam
learning rate	0.001
loss function	MSE or CCE
max epochs	10000
early stopping	20
batch size	all
activation function	ReLU

Tab. 5.24.: Hyperparameters for FCN classifier or regressor. If classifier, then CCE (categorical cross entropy) is used as loss function, if regressor, then MSE is used.

Classification Results - Gender and Country

First, let's look at the classification results in tab. F.2. We can make some observations from these:

1. All the models have an easier time learning gender than country, when compared to the random assignment.
2. The Combined-input MSE-loss model is best at determining gender and the Dorsal-input MSE-loss model is best at determining country.
3. The Ventral models are consistently the worst for determining gender, however, they are still significantly better than random assignment (89% Accurate on the test set for the Ventral MSE model, and 52% accurate for the random model.). This suggests that there is some significant determination of gender which can be done from the ventral images. This is unexpected as we understood the gender is usually determined by the dorsal features, specifically the amount of blue present on the dorsal wings.
4. The MSE loss function performs best out of these models
5. The choice of latent space for the model does not seem to significantly affect the classification results.

Regression Results

Next, let's look at the regression results in tab. F.3. These are more difficult to directly compare and interpret, but we can make some observations:

1. **Longitude**: It seems that almost all the models found it easiest to learn the longitude, and the regression MSE of the models are all similar. However, this is slightly misleading, as there are two specimens from Russia which have very high longitudes compared to all the other specimens from Europe. These two specimens are in the training set for the Dorsal and Ventral models, so the random MSE and train MSEs are quite high, while the validation

and test MSEs are quite low. The Combined dataset is shuffled differently, and includes the two russian specimens in the test set instead, skewing the test MSE to look quite large. This is an unfortunate artifact of our dataloader which was discovered too late.

2. **Altitude**: The altitude seems to be best described by dorsal features. The models are all quite close here. The discrepancy between the Ventral models and other models is again probably due to the images being randomly shuffled differently in the dataloader for dorsal, ventral and combined inputs.
3. **Latitude**: The MSE models clearly perform better on this feature, with only minor differences between Dorsal, Ventral and Combined inputs.
4. **Year**: This seems to be the hardest feature for the models to predict. The MSE models with Dorsal and Ventral inputs seems to produce the best results here. What is interesting, is that the combined model performs significantly worse. This could be an artifact of us not running multiple trials, and perhaps the model did not converge, or perhaps because the model has to learn twice as much in the same 128 dimensional latent space, it does not learn to pick up some details in the combined model as well as it does in the ventral and dorsal models. This might indicate that a larger latent space should have been used for the combined model.
5. **Variational Latent Space**: The choice of latent space for the variational models (mean of the posterior or using all samples from the posterior) does not seem to significantly affect the regression results.

Now we have some quanitication of how well the model can learn to distinguish each of the classes.

Reconstructions

As a final analysis on the model results, let's look at the reconstructions created by running the average of the latent space of each class through the decoder. To do this, it is necessary to first discretize the numerical data so we can classify each image as a single class. To do this, we use the same discretizations used to make the histograms in fig. 2.2. For latitude and longitude, we use the Kaaber regions instead. The full set of reconstructions for all the different models can be found in appendix G. The most clear difference between reconstructions, is again in the dorsal-input averages per gender, which are clearly distinguishable. The country of origin and altitude also show quite some variation, however this could also be due to different distributions of male/female specimens from the different countries. The Kaaber regions show only minor variations.

Finally, we now have some indication of how the various models distribute the data, how the data looks in latent space, and what this translates to for the specimens. Below, as a last result, we will qualitatively evaluate the preprocessing of the images. Then we can move to the discussion where we explore these results in more depth.

5.6 Maculinea Alcon Preprocessing Evaluation

The preprocessing of the Maculinea Alcon dataset was mostly successful - we were able to take most of the original images and convert them to a somewhat standardized format. There were, however, some issues, which we will point out here. Perhaps the reader with an outside view may be able to take from this some ideas of how to improve the process.

In total, developing the methodology to standardize the images and executing it took over a month, which included many mistakes, code development, and manual checking of the results. The manual checking here is unfortunately key, and I see no way to create reliably good results without manual tweaks throughout the process. For any reader interested in recreating this process and wants to understand how much manual effort is involved, the manual tweaks that needed to be completed are listed here:

1. Initially the dataset was scoured for wrongly labelled images - 29 images renamed
2. During the segmentation process, approximately 180 images needed to be manually flagged as either missing parts of the butterfly after segmentation, or including too much of the background in the segmentation, or other weird features such as the butterfly being upside-down. Most of these could be aggregated though and fixed by using slightly different parameters in the felzenwalb segmentation. At this point, two images were deemed too difficult to segment, because they were overlapping with part of the label or color bar, and were thrown out of the dataset.
3. After the corners of the cm scale were found using the harris corner detector, the corners for each image were manually checked and corrected using the gui shown in fig. 4.12. I have no data here on how many corrections were made - this was completed at the same time as checking that the segments of the color bar were correctly found, and the whole process took approximately 10 hours. I retrospect, I think it might be possible to improve this by improving on the initial corner detection methodology or to automatically flag corners which do not seem square.
4. After finding the corners of all the butterflies rotating them, 32 images needed to be manually annotated, since the algorithm had difficulty finding the top/bottom of their wings, or centering them. This was especially true for butterflies that were missing wings. To do this manual annotation, VIA (VGG Image Annotator) was used [13].

5.6.1 Resamples

There is also the problem of a noticeable difference between the images sampled in the Copenhagen sub-dataset and the Aarhus sub-dataset, which can affect any further analysis. Since 30 specimens from the Aarhus sub-dataset were resampled with the Copenhagen setup, we can directly quantify the differences between the two setups. A visual comparison between the final Aarhus images and the ones resampled in the Copenhagen setup, is provided in appendix A.

6 Discussion

> *Questions you cannot answer are usually far better for you than answers you cannot question.*
>
> **— Yuval Noah Harari**
> 21 Lessons for the 21st Century

In the previous chapter we showed results from various experiments aimed at improving the cluster formation of images, and how the latent representations can be used to explore a dataset. Here we summarize those results and draw some conclusions from them.

First, in sec. 6.1 we summarize and compare the experiments involving architecture and loss functions.

Then, in secs. 6.2-6.3 we take a deeper look at the LiPPy and GiMMPPy models. Here we also show how the number of Monte Carlo samples affects runtime and model convergence for the GiMMPPy model.

Next, in sec. 6.4 we draw some conclusions about which loss function performs best.

Then, in sec. 6.5 we look at our final results from the Maculinea Alcon clustering, and superficially compare these results with Kaaber's analysis (we leave a real comparison to the biologists).

Finally, in sec. 6.6 we expand on some unanswered questions which have come up through this thesis, including blurring of the reconstructions, differences between the encoder and decoder filters, what would happen if we used the wrong number of clusters in the GiMMPPy model. These are all topics which we haven't been able to dive into in detail in this thesis. Here we present some new results, but due to time restrictions, we haven't run the models enough times to properly include them in the results section, but only use them to illustrate potential areas of interest for further research.

6.1 Effect of Architecture and Loss Function on Cluster Formation

In sections 5.1-5.3.3 we explored the effect of different architectures and loss functions on cluster formation in the latent space. We completed many different experiments, beginning with a simple 1-layer autoencoder network, to more complex convolutional networks. An overview of the best of each round of experiments on MNIST and on the Maculinea Alcon dataset are provided in tab. 6.1 and tab. 6.2 respectively. These are discussed in more detail in the sections that follow. For each experiment, please refer to the relevant section for details of the model.

Sec.	Experiment	Loss	Init.	Params	s/epoch[a]	Mean ARI	Std ARI
5.2.3	Loss	GiMMPPy	EEK	29022	61.667	0.946	0.003
5.3.4	1 Posterior	GiMMPPy	EEK	27162	61.492	0.939	0.005
5.1.4	Conv Str AE	MSE	-	21303	3.657	0.916	0.018
5.2.3	Loss	LiPPy	Zero	29022	33.311	0.893	0.026
5.2.1	Loss	BCE	-	21303	7.625	0.889	0.041
5.1.5	VGG-like AE	MSE	-	67775	4.270	0.874	0.054
5.2.2	Denoising AE	MSE	-	21303	9.600	0.825	0.096
5.1.2	Hidden AE	MSE	-	57154	6.178	0.720	0.139
5.1.3	Conv AE	MSE	-	6312483	5.054	0.682	0.090
5.2.3	VAE	ELBO Gaussian	Zero	29022	23.435	0.676	0.025
5.2.3	VAE	GiMMPPy	Random	29022	58.290	0.599	0.087
5.1.1	Simplest AE	MSE	-	47854	5.858	0.400	0.101

Tab. 6.1.: Summary comparison of best models from each set of experiments on MNIST 1-2-9 subset. The number of seconds per epoch here is previously unseen, and allows us to discuss if the runtime-accuracy tradeoff is worth it here.

[a] Seconds taken per epoch, on average during training

Sec.	Experiment	Loss	Init.	Latent	Params	s/epoch[a]	Mean ARI	Std ARI
5.3.2	Hidden Layers	MSE	-	-	140387	6.714	0.836	0.022
5.3.4	GiMMPPy Loss	GiMMPPy	EEK	Mean	190181	11.093	0.832	0.011
5.3.4	LiPPy Loss	LiPPy	Zero	Mean	157028	11.534	0.816	0.031
5.3.1	Convolutional	MSE	-	-	118921	6.793	0.816	0.073
5.3.3	BCE Loss	BCE	-	-	140387	8.644	0.759	0.097
5.3.3	GiMMPPy Loss	GiMMPPy	EEK	All	190181	11.093	0.734	0.025
5.3.3	LiPPy Loss	LiPPy	Zero	All	157028	11.534	0.722	0.010

Tab. 6.2.: Summary comparison of best models from each set of experiments on Maculinea Alcon dataset. Results are on the validation set.

[a] Seconds taken per epoch, on average during training

It should be noted that the higher runtime of the VAEs is at least partially due to halving the batch size, in order to allow it to run on my gpu without running into memory errors, as the

variational model requires more memory per batch to account for the number of samples taken per image (in our case, 10). It is also partially due to the monte carlo sampling approximation of the kl-divergence, which accounts for why both VAE GMM models take twice as long per epoch compared to the non-GMM models.

6.1.1 Effect of Architecture on Cluster Formation

Here we point out some learnings from the results of the architecture experiments.

1. **Better MSE $\neq$ Better ARI** Looking at the plots comparing the ARI and MSE for different models in sec. 5.1, the MSE does not appear to be closely correlated with model's ability to form well-separated clusters in the latent space, leading us to believe that a different loss function may be key to encouraging cluster formation
2. **Depth/Width of FCNs to increase ARI** Looking at the results from the MNIST hidden-layer experiments in sec. 5.1.2, it appears that deep thin fully connected networks generate distinct latent clusters better than deep wide networks or shallow networks, given the same latent dimensionality/bottleneck. This could be because non-linearity is more important for this specific task than including many variables, particularly when there are no convolutional layers to learn high level features.
3. **Adding convolutional layers** Looking at the results from the MNIST convolutional layers and convolutional strided layers in sections 5.1.3-5.1.4, the number of convolutional layers does not significantly affect the clustering results. In fact, relationship between convolutional depth and ARI appears quite random, suggesting that the depth is not so important. However, having convolutional layers, and the stride/parameter reduction of those layers does seem to. This could indicate that this model does not need very high level image features to be successful, but is sufficiently good with low-level edge detection, which matches our expectations well since the MNIST dataset is fairly simplistic compared to other image recognition tasks.
4. **Parameter reduction** Adding convolutionally strided layers allows us to significantly reduce the number of parameters in the model, while providing better results than simple fully connected networks. This is probably because the convolutional layers allow the network to share kernels across the image, so are less likely to overfit to the training set.

From the experimental results we were able to settle on a convolutionally strided network with 3 fully connected layers for the architecture to be used for MNIST clustering (for the exact network, see fig. 5.13). The experiments completed are by no means exhaustive, and there are plenty of additional neural network layers or tricks we did not explore, such as adding skip-connections. However, using this we created a solid foundation to begin testing different loss functions and have demonstrated how different architectures can affect clustering. With these conclusions, let's move on to exploring the effect of the loss function on cluster formation.

6.1.2 Effect of Loss Function on Cluster Formation

In sections 5.2 and 5.3.3 we explored the effect of different loss functions on the ability of the model generate distinct latent clusters. A comparison of some of the experiments is provided in tables 6.1 and 6.2.

We can take away some interesting learnings from these:

1. **Denoising**: Making the model a denoising model appears to have a negative effect on the cluster formation in the latent space. We hypothesize this might be because the model focuses on learning where to denoise instead of the underlying structure of the original data.
2. **Zero Mean Gaussian VAE vs GiMMPPy**: VAE architectures with a zero-mean gaussian prior are worse at latent space cluster formation in this case. This is probably because they push the latent space into a single gaussian, making it harder to distinguish the borders of the clusters in an unsupervised way.
3. **LiPPy vs MSE**: VAE architectures with a zero-mean laplacian prior appear to allow the latent space to spread out more, and consistently produce well-defined clusters, however, still do not demonstrate significant improvements on the non-variational MSE model (0.893 ARI vs 0.916 ARI), and the normal MSE model has almost a tenth of the runtime.
4. **GiMMPPy**: VAE architectures with a GMM prior, updated once per epoch using k-means, can consistently improve cluster formation, however this requires that you know the number of clusters before starting the experiment, or repeat the experiment many times with different numbers of clusters. The consistency here is key, as many of the other architectures have a high variance across experiments, but the standard deviation here is an order of magnitude lower.

From these, we can see that a VAE, with a well chosen prior can consistently encourage latent cluster formation, compared to a non-variational model. However, this comes at the cost of much higher runtimes, and you need to know the number of clusters expected beforehand. Also, the results of the best MSE model on cluster formation was still better than the best VAE model.

6.2 LiPPy Model

With the LiPPy model we showed that the choice of a different prior distribution for the variational autoencoder can improve the resulting clustering in the latent space. However, the question of whether this loss function we have proposed is really Laplacian remains to be determined. While we approximate the kl divergence using the sum of the 1D kl divergences, this, as shown in sec. 3.3.6, is probably not a particularily good approximation. That said, Monte Carlo sampling could probably be used in the same way we did for the GiMMPPy model to approximate the KL divergence, but we leave this for future work.

6.3 GiMMPPy Model

We explored using a GMM as posterior and prior in a variational autoencoder. In many cases this produced the best clustering, and with the lowest standard deviation in ARI, indicating it is much more consistent than using mean squared error. However, this comes at the cost of added complexity, computational time and the necessity of knowledge of approximately the number of clusters of interest, although, as we will explore in sec. 6.6, getting this wrong is not necessarily detrimental to the model.

Effect of no. Monte Carlo Samples on ARI in GiMMPPy model

We ran some trials on the effect of the number of Monte Carlo samples on the resulting ARI for the Maculinea Alcon dataset. Although these trials are by no means exhaustive, they serve to justify our usage of the low number of 100 samples per batch used in our experiments.

No. Monte Carlo Samples per Batch	Mean ARI	s/epoch
10	0.72	7.061
100	0.734	11.093
1000	0.73	33.830
10000	0.76	155.628

Tab. 6.3.: Runtime and cluster formation for dorsal-input GiMMPPy models with varying number of Monte Carlo samples to approximate the KL divergence between the prior and the batch's posterior

From tab. 6.3 we might be tempted to say that it is acceptable then to use 10 Monte Carlo samples. The trouble is, the numbers above only show the results from those models that converged, and the 10-sample Monte Carlo experiments has a higher tendency to diverge (3/5 models converged) compared to the 100 sample experiments (3/4 models converged).

6.4 So, what is the best loss function for cluster formation?

Although the GiMMPPy and LiPPy loss functions provide in many cases the best results, due to the tradeoff between added complexity of creating the models, added runtime and the necessity to know beforehand the number of clusters you are interested in, they are highly impractical. If I were to begin exploring a new dataset in a similar manner, I would at least start with MSE, and perhaps attempt to use GiMMPPy later to refine the results. That said, further exploration of different priors for variational models could certainly provide better and more interesting results,

and other papers [11, 24] show quite good results, so perhaps it works better when adapting the SGVB VAE instead of the AEVB VAE. Perhaps the LiPPy and GiMMPPy loss functions could still be useful, for example, to refine unsupervised clustering after some initial data exploration.

6.5 Maculinea Alcon Results

In this section we will scrutinize the results on the Maculinea Alcon dataset clustering. To start, we summarize the features of interest and the models which best captured these features (evaluated using the classification/regression results). See tab. 6.4.

Feature	Best Input	Best Loss Function	Val Acc./MSE	Test Acc./MSE	Rand Acc./MSE
Gender	Combined	MSE	0.976	0.953	0.490
Country	Dorsal	MSE/LiPPy/GiMMPPy	0.953	0.929	0.855
Longitude	Ventral	MSE/LiPPy/GiMMPPy	2.00	2.83	60.2
Altitude	Dorsal	LiPPy	17300	68800	190000
Latitude	Dorsal	GiMMPPy	2.85	5.29	12.6
Year	Dorsal/Comb/Ventral	LiPPy	310	270	544

Tab. 6.4.: Summary of inputs and models for each ground truth feature of interest which produced best classification accuracy / regression MSE using 2 layer FCN, and their resulting accuracies / mean squared errors on the validation and test sets compared to the accuracy of randomly shuffling the dataset. When the results for multiple models were close, all 'best' models are listed.

From this we can get an idea of which sides of the butterfly are most important for different classification/regression tasks, and which of our three final loss functions produced the best results on these. Interestingly here, the dorsal side seems to be the most important for most classifications. This could however, just be an artifact of the fact that the model produces quite blurred reconstructions which have a particularily hard time learning the reproduce the dark spots on the ventral side. Therefore, as we can also see in fig. **??**, the ventral reconstructions all look quite similar, even though they are quite distinct in the original images. The dorsal reconstructions, on the other hand, seem to have quite some variation, most obviously in the amount of blue present in the images, which is a significant indicator of gender. Kaaber's results, however, also indicated that the ventral side do not provide much indication of gender or location [25]. However, this has been challenged by recent work by David Nash and Philip Folman [16] who were able to show that the distibution in the location and size of the dark ventral spots varies by region.

Perhaps a key component missing here in order to be able to tell if the ventral side can be used as an indicator, is first having an autoencoder which was reproduce the ventral side better, including the darker spots.

6.6 Unanswered Questions

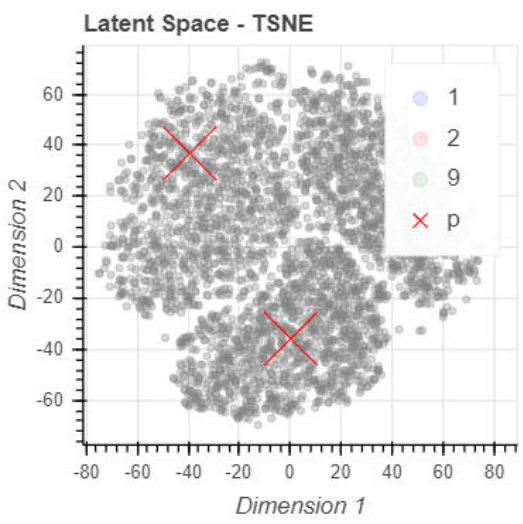
Here we are going to touch on some questions which we were unable to, or did not find time to answer in this thesis in the hopes that these might lead to future work, or inspire others.

6.6.1 Why does adding denoising not improve the model?

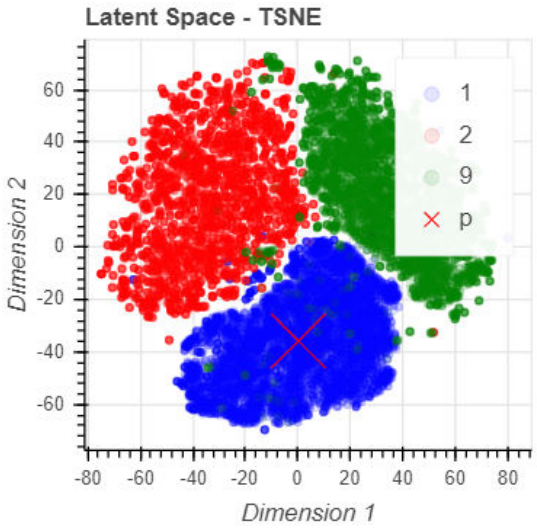
Intituitively we would expect that denoising, which improves the model's ability to generalize, should equally improve the latent space representation of the images, as it needs to learn more interesting and complex relationships between the images. However, as we saw in sec. 5.2.2, this was not the case. As we added more and more noise to the images, the latent representations produced worse and worse clusters. We hypothesize this might be because the autoencoder uses more of the latent space to determine which pixels to denoise instead of gaining a more general understanding of the image itself. However, we have not tested this in any way or explored it further.

6.6.2 Wrong number of Gaussian Components

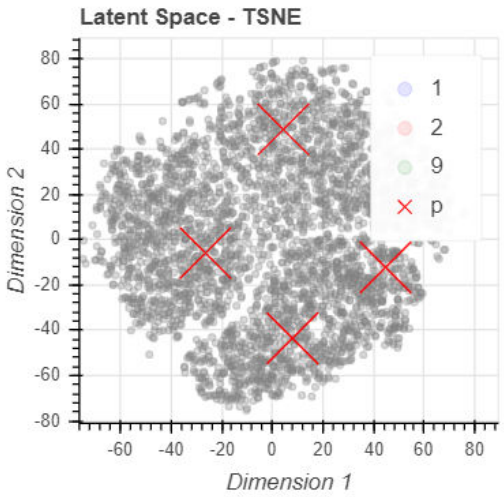
What happens if we set the GiMMPPy model up with the wrong number of gaussian components? Let's quickly do a couple of experiments on the 1-2-9 MNIST subset to see. First we set the number of Gaussian components to 2 instead of 3, and then to 4 instead of 3, and look at the final t-SNE as well as the location of the prior.



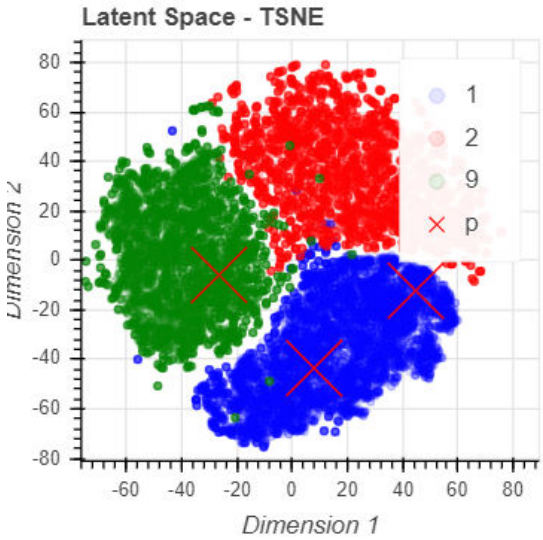
(a) Two Gaussian Components, Means of Prior shown with X



(b) Two Gaussian Components, Ground Truth Labels



(c) Four Gaussian Components, Means of Prior shown with X



(d) Four Gaussian Components, Ground Truth Labels

Fig. 6.1.: 2D t-SNE visualizations of experiments on the 1-2-9 MNIST subset with the wrong number of Gaussian components - 2 (left) and 4 (right).

From these we can see that using the wrong number of Gaussian components when setting up the GMM prior does not seem detrimental to the model. However, further experiments should be done to verify this.

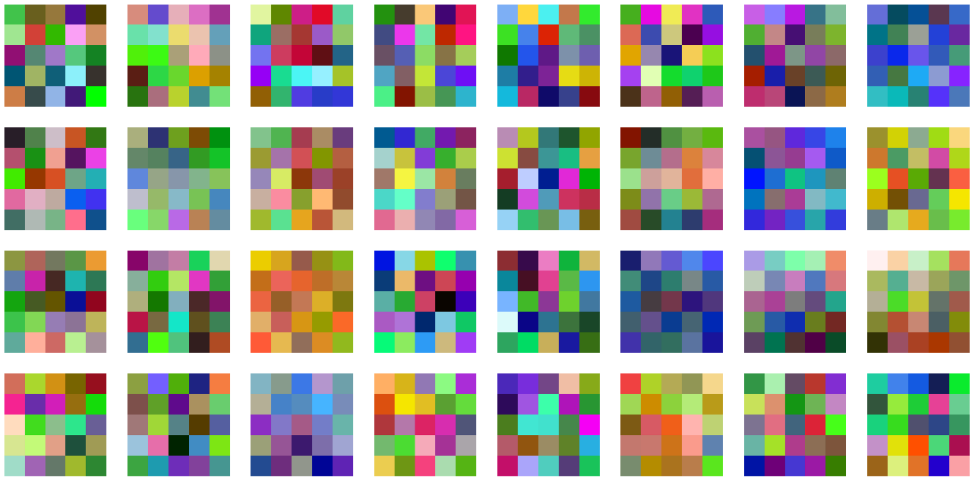
6.6.3 Blurring of Results

The output of all the Maculinea Alcon models used in this thesis look quite blurred - some examples are shown below in fig. **??**.

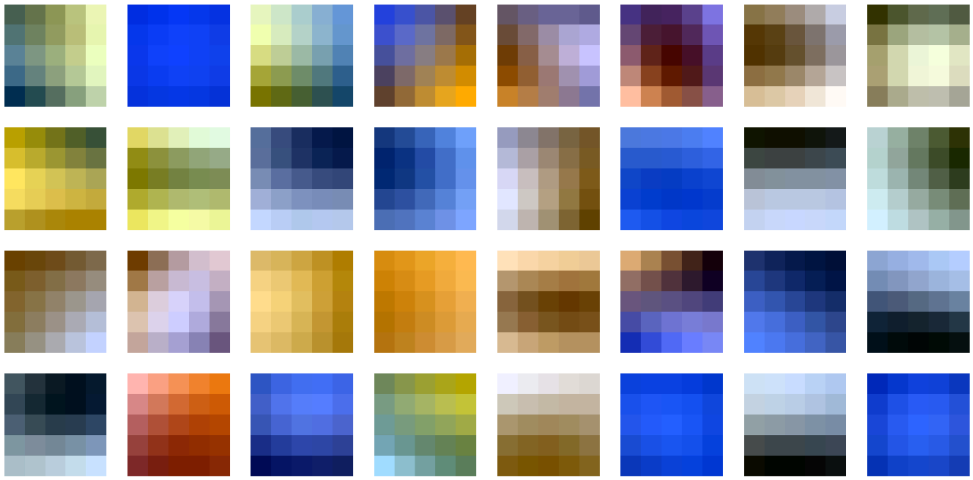
Fig. 6.2.: Example outputs from models. Originals (left), reconstructions (right). These exhibit blurring in the reconstructions.

Throughout the process of this thesis, we were unable to create an undercomplete model which did not have this extreme blurring. Although this was not the focus of this thesis, so we did not explore the reason for this in much detail, we might suspect that this means the latent space is not learning enough details, and this could also be the reason the ventral models generally produced worse results on the classification and regression - because the latent space could not learn the locations of the dark blobs well enough, and blurs them out instead. To try to understand this

better, we can quickly look at the convolutional filters learned by the encoder and decoder. There are shown in fig. **??**.



(a) Encoder Filters, first convolutional layer



(b) Decoder Filters, last convolutional layer

Fig. 6.3.: Comparison of learned convolutional filters of first layer of the encoder compared to last layer of the decoder for Dorsal-input MSE-loss model.

Here we notice that the decoder filters look much like we would expect - similar to edge detector and blob detector filters, however, the encoder filters look quite random. This suggests that the encoder may not be learning properly. This could, in some sense be due to the vanishing gradient, although this is unlikely as our model is rather shallow and uses the ReLU activation function. This could perhaps be removed by using tied weights. Answering this question is left to future work.

6.7 If I were to do it again - How would I approach it?

I would follow a very similar process;

1. I would begin by preprocessing the dataset in much the same way
2. Then I would instead focus on creating an undercomplete convolutional autoencoder with produces good reconstructions first, with a high focus on the ability to reconstruct the dark blobs on both the dorsal and ventral wings.
3. Then I would experiment with reducing the size of the latent space, to obtain a good tradeoff between dimensionality reduction and learning important visual features.
4. I would then again experiment with different loss functions.

7 Future Work

> *Tomorrow is often the busiest day of the week.*
>
> — **Spanish Proverb**

In this chapter we will list some areas which we think could be interesting for future work on this topic.

Maculinea Alcon Analysis

1. Determine why the Combined-input MSE-loss model produces 4 distinct clusters in t-SNE (see appendix F).
2. Compare our results with Kaaber's [25] and Philip's [16] results in more detail.

Investigate Blurring of Results

1. Explore architectures with a focus on producing sharper reconstructions, at least which show the distinct ventral dark spots, and see if this improves the classification / regression results on the ventral sides
2. Modify the models to use tied weights so we remove the random-looking encoder filters
3. Try other loss functions - L1 loss for example

Attempt to Improve Cluster Formation using VAEs

1. Test other heavy-tailed distributions such as student t distribution
2. Test using other distance measures for the k-means, such as Mahalnobis Distance - although since all Gaussian components in the prior have the same variance, 1, it may not be relevant.
3. Test how sensitive GiMMPPy Loss function is other other architectures

Attempt to Improve GiMMPPy VAE Results

1. Futher explore effect of number of Monte Carlo samples on model convergence and speed.
2. Identify and test other methods of intializing the means of the prior.
3. Test effect of using other methods listed in Hershey et al. [22] of approximating the KL divergence between two GMMs on model convergence and speed.
4. Compare directly with Jiang et al. [24]

5. Explore clustering results for larger number of clusters. Explore ways of 'pulling' the clusters apart - forcing them to be 6 standard deviations away from eachother.
6. Explore how the model chooses weight of posterior when using multi-posterior model
7. Figure out if the different Gaussian components in the posterior that collapse into two components in the MNIST model represent different classifications (1,2, or 9).

Attempt to Improve GiMMPPy VAE Results

1. Use Monte Carlo sampling approximate the Laplacian's KL divergence, instead of the 1D-sum approximation
2. Mathematically determine the KL divergence between two multivariate Laplacian distributions.

Pinned Insect Preprocessing/Digitization

1. Explore methods to remove horizontal banding from images, and/or ensure digitization stations do not produce this effect by increasing/decreasing the ISO speed on the camera, not using electronic shutter, or using lighting with higher/lower frequency
2. Explore other methods of color correction, and compare lighting setups for long-term digitization station usage, so specimens are evenly illuminated and between images can be better compared.
3. Explore how to improve digitization station so it is more image-processing friendly. For example, more easily computer-identifiable reference squares for measuring scale. Making the color target's more machine-readable. Improving segmentation of the specimen body.

8 Conclusions

> *There is no real ending. It's just the place where you stop the story.*
>
> — **Frank Herbert**

In this thesis we set out to acheive the following three goals:

1. Find and compare methods of clustering images using autoencoders
2. Explore the Maculinea Alcon dataset in a semi-supervised way and draw some conclusions about how the data therein is grouped
3. Produce a workflow for end-to-end semi-automatic processing of images of pinned insects to standardize the images for analyses that may involve color or size

Comparing methods of clustering images

For this goal, we focused on a subset of clustering techniques, specifically focusing on comparing architectures and loss functions for autoencoders. This analysis was by no means exhaustive, however, we were able to show that choice of architecture and particularily loss function can greatly affect the final cluster formation, and researchers should be careful to first experiment with different architectures beforehand. Although we did not conclude here as to one single 'best' architecture for every application, we showed that convolutional networks with fully connected layers can be successfully applied to cluster formation problems, and stride and/or max pooling are quite important to keep the number of trainable parameters low.

We also compared different loss functions. We concluded that denoising appears to negatively affect clustering, although we are not entirely sure why. We also showed results for two new posteriors and priors used in variational autoencoders, the Laplacian (LiPPy loss) and the Gaussian mixture model (GiMMPPy loss). We showed that a different choice of prior can greatly improve cluster formation for variational autoencoders, in particular choosing a prior with heavier tails, like the LiPPy loss, or a mixture model, although the mixture models have the downside that you need to know the number of clusters of interest before conducting the training. All the variational models have the additional downsides of added model complexity and runtimes, which make their usage difficult to justify, except in niche cases.

The Maculinea Alcon Dataset

We were able to create latent representations of the Maculinea Alcon dataset and compare these across inputs (Dorsal, Ventral and a 6-channel combination) and loss functions (MSE, LiPPy, GiMMPPy). We found that gender is easily distinguishable in all models, however, the other features we looked at (Danish or not, Longitude, Latitude, Year and Altitude) showed no obvious clusterings or gradients in the latent space, although we were able to train a classifier / regressor on the latent space and it could learn some underlying correlations. For these experiments, the dorsal side of the butterfly appeared to be the most useful in almost all determinations. However, this could also be because our model appeared to have trouble learning the distinct dark spots on the ventral side which it has been shown may be correlated to region. As for Kaaber's question of if the Maculinea Alcon are actually two species - the Maculinea Alcon and the Maculinea Rebeli - we find no obvious indication of this, however, we also have no experience in species delineation, and leave this instead to the biologists to debate.

Pinned Insect Image Processing Workflow

We presented a semi-automatic workflow for completing end-to-end image processing to segment, scale and color correct images of pinned insects for use in analysis. This workflow will probably not work for all images, however, especially images of specimens which still have labels under the specimen itself - this will lead to further problems with segmentation. There are some things we would have done differently next time, such as improving the process to find the reference square in the images. We hope that this may be of use as a basis for further researchers in the future.

Bibliography

[1] Acxiom. „Life Stage Clustering System: "PersonicX" An explanation of the Development of the PersonicX Clustering System". In: () (cit. on p. 12).

[2] Elie Aljalbout, Vladimir Golkov, Yawar Siddiqui, and Daniel Cremers. „Clustering with Deep Learning: Taxonomy and New Methods". In: *CoRR* abs/1801.07648 (2018). arXiv: 1801.07648 (cit. on p. 2).

[3] Javier Antoran and Antonio Miguel. „Disentangling in Variational Autoencoders with Natural Clustering". In: (Jan. 2019) (cit. on p. 3).

[4] David Arthur and Sergei Vassilvitskii. „K-Means++: The Advantages of Careful Seeding". In: vol. 8. Jan. 2007, pp. 1027–1035 (cit. on p. 20).

[5] A. Azzalini and A. W. Bowman. „A look at some data on the Old Faithful Geyser". In: *Applied Statistics* 39 (1990), pp. 357–365 (cit. on p. 13).

[6] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006 (cit. on pp. 13, 22, 23).

[7] Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. „Deep Clustering for Unsupervised Learning of Visual Features". In: *CoRR* abs/1807.05520 (2018). arXiv: 1807.05520 (cit. on p. 2).

[8] Computerphile. *How Blurs and Filters Work*. https://www.youtube.com/watch?v=C_zFhWdM4ic (cit. on pp. 29, 30).

[9] scikit-image developers. *scikit-image*. https://scikit-image.org/docs/dev/api/skimage.html (cit. on p. 3).

[10] scikit-learn developers. *Clustering*. https://scikit-learn.org/stable/modules/clustering.html (cit. on pp. 20, 123).

[11] Nat Dilokthanakul, Pedro A. M. Mediano, Marta Garnelo, et al. „Deep Unsupervised Clustering with Gaussian Mixture Variational Autoencoders". In: *CoRR* abs/1611.02648 (2016). arXiv: 1611.02648 (cit. on pp. 3, 45, 104).

[12] John Duchi. *Derivations for Linear Algebra and Optimization*. http://stanford.edu/~jduchi/projects/general_notes.pdf (cit. on p. 16).

[13] A. Dutta, A. Gupta, and A. Zissermann. *VGG Image Annotator (VIA)*. http://www.robots.ox.ac.uk/~vgg/software/via/. 2016 (cit. on p. 97).

[14] Charles Elkan. *Using the Triangle Inequality to Accelerate k-Means*. 2003 (cit. on p. 20).

[15]Raquel Florez and Juan Ramon. „Marketing Segmentation Through Machine Learning Models: An Approach Based on Customer Relationship Management and Customer Profitability Accounting“. In: *Social Science Computer Review - SOC SCI COMPUT REV* 27 (Apr. 2008), pp. 96–117 (cit. on p. 12).

[16]Philip Folman. *Patterns in Danish populations of Maculinea alcon*. 2019 (cit. on pp. 1, 6, 104, 111).

[17]Ali Ghodsi. *Ali Ghodsi, Lec : Deep Learning, Variational Autoencoder, Oct 12 2017 [Lect 6.2]*. `https://www.youtube.com/watch?v=uaaqyVS9-rM` (cit. on p. 34).

[18]Francis D Gibbons and Frederick P Roth. „Judging the quality of gene expression-based clustering methods using gene annotation“. In: 12 (10 Oct. 2002), pp. 1574–81 (cit. on p. 18).

[19]Manuel Gil. *On Renyi Divergence Measures for Continuous Alphabet Sources*. 2011 (cit. on p. 17).

[20]Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing (3rd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006 (cit. on p. 3).

[21]Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016 (cit. on pp. 12, 22).

[22]J. R. Hershey and P. A. Olsen. „Approximating the Kullback Leibler Divergence Between Gaussian Mixture Models“. In: *2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07*. Vol. 4. Apr. 2007, pp. IV-317-IV–320 (cit. on pp. 16, 17, 111).

[23]Roberta Hunt. *Workflows for Digitization of Insect Collections*. `https://github.com/diku-dk/image_gui/blob/master/Internship_Report___Digitization_Workflow__3_.pdf` (cit. on pp. 3, 10).

[24]Zhuxi Jiang, Yin Zheng, Huachun Tan, Bangsheng Tang, and Hanning Zhou. „Variational Deep Embedding: A Generative Approach to Clustering“. In: *CoRR* abs/1611.05148 (2016). arXiv: `1611.05148` (cit. on pp. 3, 45, 46, 104, 111).

[25]S. Kaaber. „Studies on Maculinea alcon (Schiff.) -rebeli (Hir.) (Lep. Lycaenidae) with Reference to the Taxonomy, Distribution, and Phylogeny of the Group“. In: *Entomologiske Meddelelser* 32 (1964), pp. 277–319 (cit. on pp. 1, 6, 8, 39, 104, 111).

[26]Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015. 2014 (cit. on p. 28).

[27]Diederik P Kingma and Max Welling. „Auto-Encoding Variational Bayes“. In: *arXiv e-prints*, arXiv:1312.6114 (Dec. 2013), arXiv:1312.6114. arXiv: `1312.6114 [stat.ML]` (cit. on pp. 16, 34, 35).

[28]Yann LeCun and Corinna Cortes. „MNIST handwritten digit database“. In: (2010) (cit. on p. 6).

[29]Weixian Liao, Yifan Guo, Xuhui Chen, and Pan Li. „A Unified Unsupervised Gaussian Mixture Variational Autoencoder for High Dimensional Outlier Detection“. In: Dec. 2018, pp. 1208–1217 (cit. on p. 3).

[30]Laurens van der Maaten and Geoffrey E. Hinton. „Visualizing Data using t-SNE“. In: 2008 (cit. on p. 37).

[31]E. Min, X. Guo, Q. Liu, et al. „A Survey of Clustering With Deep Learning: From the Perspective of Network Architecture“. In: *IEEE Access* 6 (2018), pp. 39501–39514 (cit. on p. 2).

[32]Sudipto Mukherjee, Himanshu Asnani, Eugene Lin, and Sreeram Kannan. „ClusterGAN : Latent Space Clustering in Generative Adversarial Networks“. In: *CoRR* abs/1809.03627 (2018). arXiv: `1809.03627` (cit. on p. 2).

[33]Augustus Odena. *Deconvolution and Checkerboard Artifacts*. `https://distill.pub/2016/deconv-checkerboard/` (cit. on p. 69).

[34]M. A. Ponti, L. S. F. Ribeiro, T. S. Nazare, T. Bui, and J. Collomosse. „Everything You Wanted to Know about Deep Learning for Computer Vision but Were Afraid to Ask“. In: *2017 30th SIBGRAPI Conference on Graphics, Patterns and Images Tutorials (SIBGRAPI-T)*. Oct. 2017, pp. 17–41 (cit. on pp. 29–31).

[35]Benjamin Price, Steen Dupont, Elizabeth Allan, et al. *ALICE: Angled Label Image Capture and Extraction for high throughput insect specimen digitisation*. Nov. 2018 (cit. on p. 3).

[36]Alec Radford, Jeff Wu, Rewon Child, et al. „Language Models are Unsupervised Multitask Learners“. In: (2019) (cit. on p. 12).

[37]Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. „Contractive Auto-encoders: Explicit Invariance During Feature Extraction“. In: *Proceedings of the 28th International Conference on International Conference on Machine Learning*. ICML'11. Bellevue, Washington, USA: Omnipress, 2011, pp. 833–840 (cit. on p. 43).

[38]Sebastian Ruder. „An overview of gradient descent optimization algorithms“. In: *CoRR* abs/1609.04747 (2016). arXiv: `1609.04747` (cit. on p. 27).

[39]Timur Sattarov and Mircea-Serban Pavel. „On the importance of preprocessing and initialization in k-means“. In: (Aug. 2015) (cit. on p. 18).

[40]Karen Simonyan and Andrew Zisserman. „Very Deep Convolutional Networks for Large-Scale Image Recognition“. In: *CoRR* abs/1409.1556 (2014) (cit. on pp. 31, 72).

[41]Pang-Ning Tan, Michael Steinbach, Anuj Karpatne, and Vipin Kumar. *Introduction to Data Mining (2Nd Edition)*. 2nd. Pearson, 2018 (cit. on pp. 18, 19).

[42]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*. ICML '08. Helsinki, Finland: ACM, 2008, pp. 1096–1103 (cit. on p. 32).

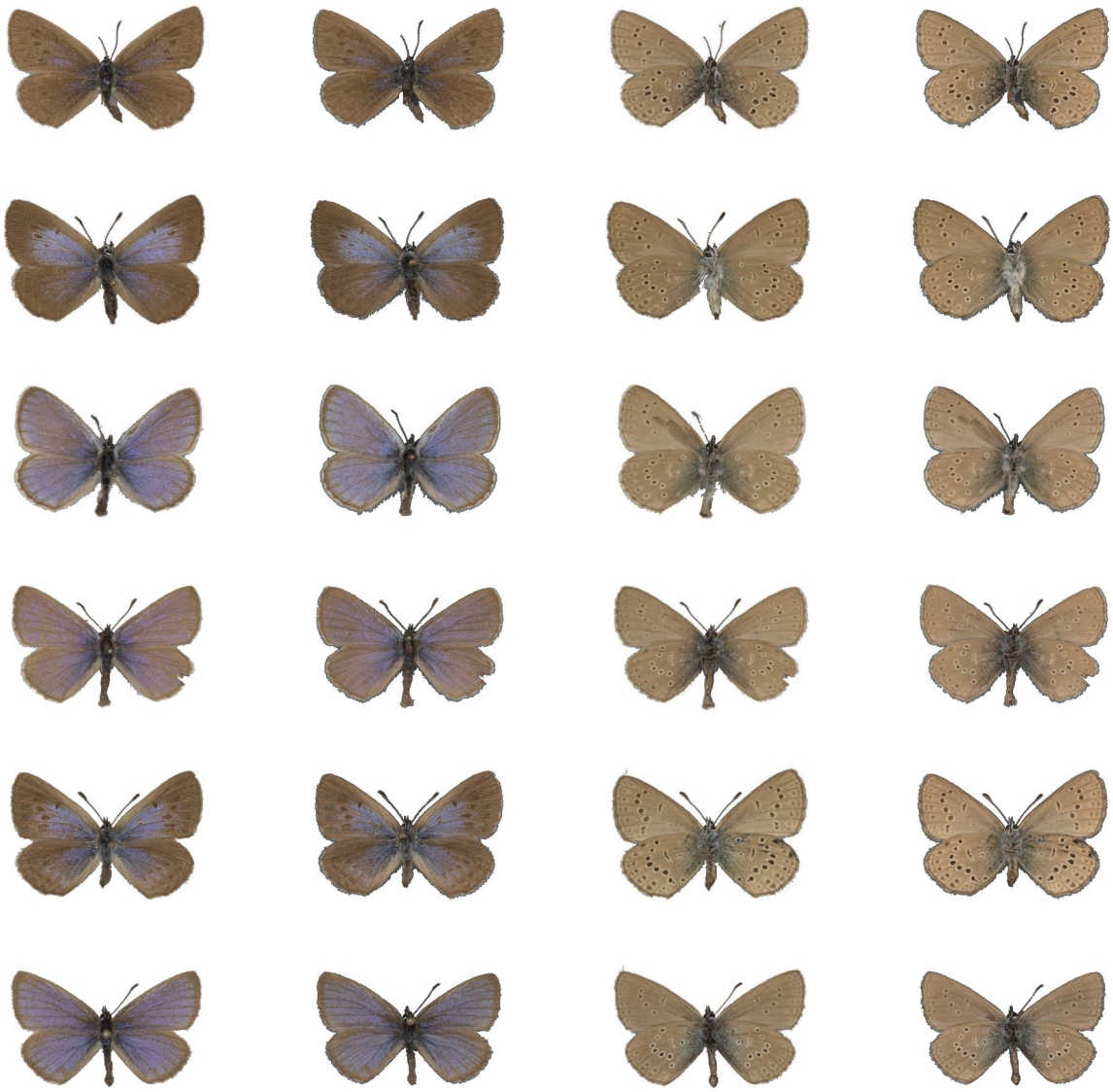
[43]Martin Wattenberg, Fernanda Viégas, and Ian Johnson. „How to Use t-SNE Effectively“. In: *Distill* (2016) (cit. on p. 37).

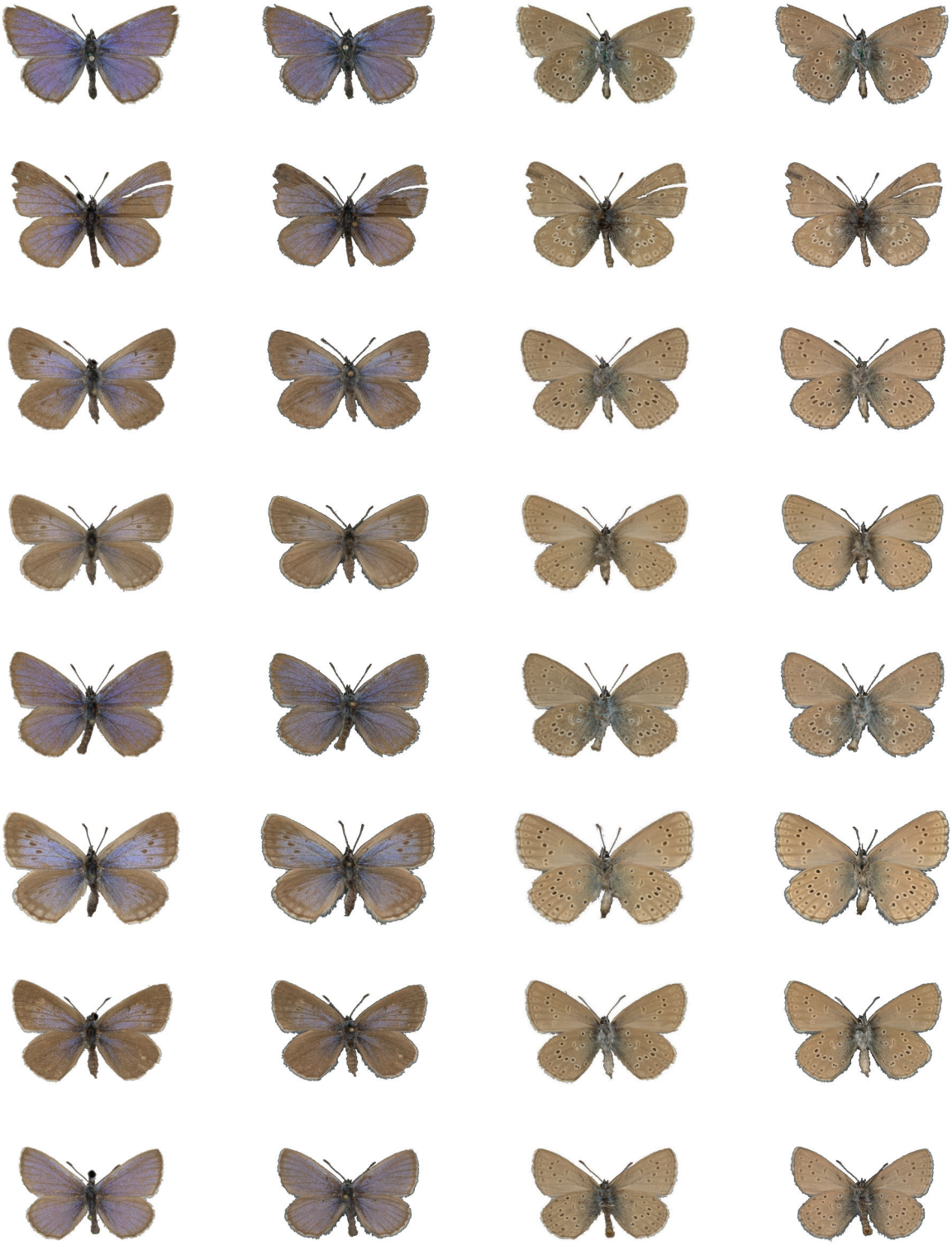
[44]Zhigang Xiong and Zhongneng Zhang. „A Data Preprocessing Method Applied to Cluster Analysis on Stock Data by Kmeans“. In: Jan. 2016 (cit. on p. 18).

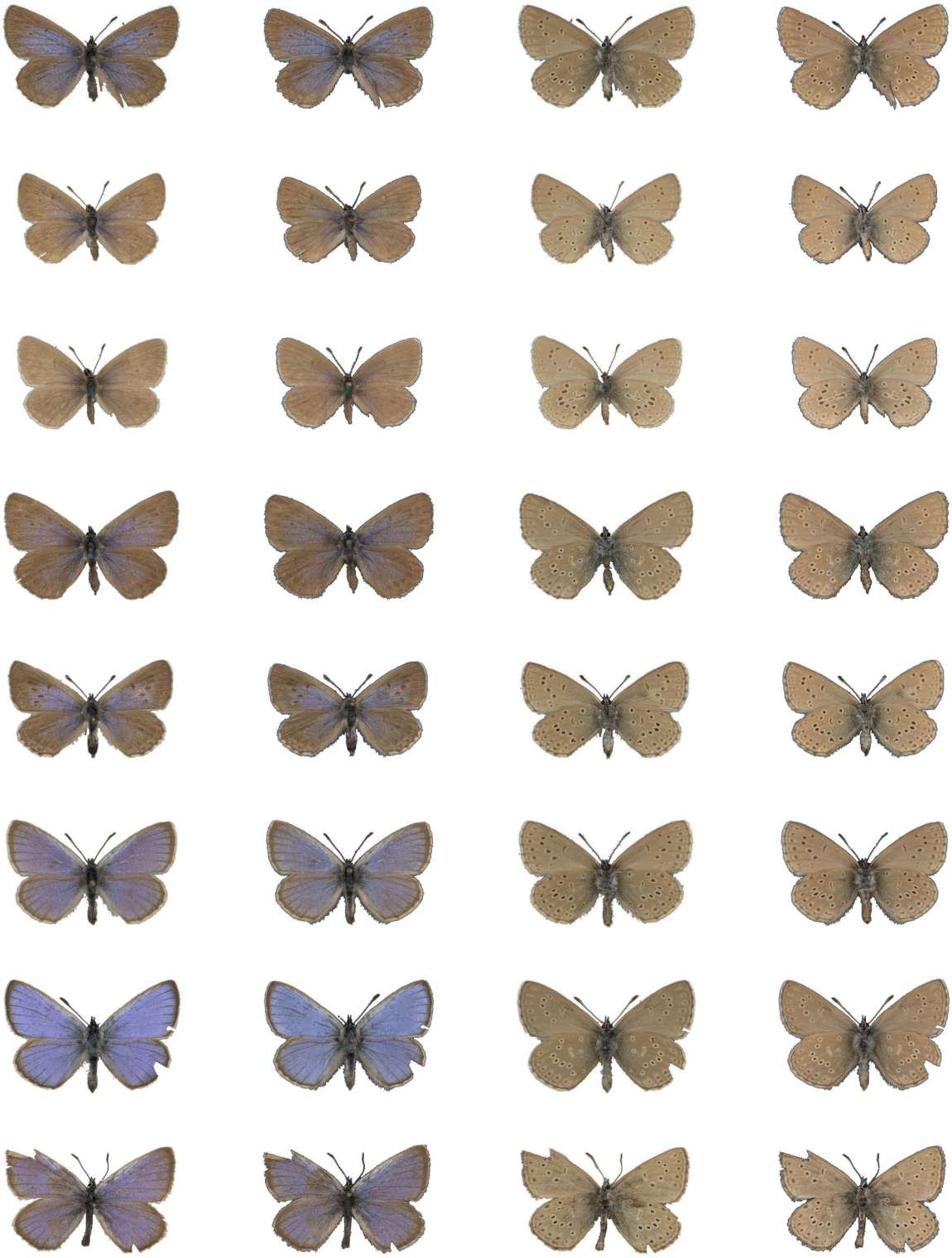
A

Comparison of Processed Maculinea Alcon Images between Copenhagen and Aarhus Sub-Datasets

In this appendix we show a comparison between the two datasets, and the effect of the imaging setup on the final results. From left to right there is: Aarhus Dorsal, Copenhagen Dorsal, Aarhus Ventral, Copenhagen Ventral. This is given for all resampled images.







B

Full Results of Cluster Formation using different 2-digit MNIST Combinations

Here we show the full results of the cluster formation on each of the MNIST 2-digit combintations using the simple AE architecture (see sec. 5.1.1). We also list the results on many different clustering criteria, including clustering criteria which do no require ground truth labellings. After these experiments however, it was decided to continue using only the Adjusted Rand Index (ARI) for simplicity. The others are shown here only to demonstrate that the results do not vary much across the different metrics. The metrics included are:

1. ARI: Adjusted Rand Index
2. AMI: Adjusted Mutual Information
3. NMI: Normalized Mutual Information
4. Homogenuity
5. Completeness
6. V-Score (A combination of homogenuity and Completeness)
7. Fowl: Fowlkes-Mallows Score
8. Sil: Silhouette Score
9. Dav: Davies-Bouldin Index

More information on the metrics can be found on sklearn's clustering page [10].

Subset	ARI	AMI	NMI	Homogenuity	Completeness	V Score
1-7	0.792	0.700	0.703	0.700	0.705	0.703
1-6	0.775	0.703	0.711	0.703	0.719	0.711
0-1	0.767	0.708	0.717	0.708	0.725	0.716
1-9	0.677	0.589	0.596	0.589	0.603	0.596
1-4	0.644	0.543	0.548	0.543	0.552	0.548
6-7	0.586	0.540	0.553	0.540	0.566	0.553
5-9	0.379	0.306	0.307	0.307	0.307	0.307
0-4	0.250	0.199	0.203	0.200	0.206	0.203
4-6	0.245	0.203	0.209	0.203	0.216	0.209
0-2	0.218	0.168	0.171	0.169	0.174	0.171
0-5	0.188	0.145	0.146	0.146	0.146	0.146
5-6	0.166	0.126	0.126	0.126	0.126	0.126
5-8	0.165	0.125	0.125	0.125	0.125	0.125
3-5	0.150	0.112	0.112	0.112	0.112	0.112
3-7	0.146	0.112	0.114	0.112	0.115	0.114
4-5	0.139	0.104	0.104	0.104	0.104	0.104
0-8	0.134	0.101	0.101	0.101	0.102	0.101
5-7	0.0889	0.0671	0.0675	0.0676	0.0674	0.0675
1-5	0.0783	0.0595	0.0601	0.0604	0.0598	0.0601
0-3	0.0776	0.0566	0.0570	0.0569	0.0570	0.0570
0-9	0.0703	0.0513	0.0517	0.0517	0.0517	0.0517
2-6	0.0632	0.0464	0.0467	0.0467	0.0467	0.0467
1-3	0.0335	0.0246	0.0250	0.0250	0.0249	0.0250
4-8	0.0276	0.0200	0.0204	0.0204	0.0204	0.0204
8-9	0.0267	0.0191	0.0195	0.0195	0.0196	0.0195
2-5	0.0244	0.0196	0.0200	0.0199	0.0200	0.0200
7-9	0.0234	0.0169	0.0173	0.0173	0.0173	0.0173
6-9	0.0220	0.0155	0.0160	0.0158	0.0161	0.0160
3-9	0.0214	0.0155	0.0159	0.0159	0.0159	0.0159
2-3	0.0200	0.0144	0.0148	0.0148	0.0148	0.0148
1-8	0.0148	0.009 54	0.009 90	0.009 88	0.009 92	0.009 90
7-8	0.0122	0.008 52	0.008 91	0.008 88	0.008 94	0.008 91
2-4	0.0110	0.008 12	0.008 47	0.008 47	0.008 48	0.008 47
2-8	0.0105	0.007 54	0.007 90	0.007 90	0.007 90	0.007 90
4-9	0.009 30	0.006 58	0.006 98	0.006 94	0.007 02	0.006 98
0-6	0.008 31	0.006 07	0.006 44	0.006 44	0.006 44	0.006 44
2-7	0.005 86	0.004 27	0.004 63	0.004 62	0.004 64	0.004 63
3-6	0.004 85	0.003 33	0.003 70	0.003 69	0.003 70	0.003 70
3-8	0.003 40	0.002 58	0.002 95	0.002 94	0.002 95	0.002 95
3-4	0.000 346	0.000 278	0.000 641	0.000 640	0.000 641	0.000 641
4-7	2.62×10^{-5}	3.98×10^{-5}	0.000 399	0.000 399	0.000 399	0.000 399
0-7	−0.000 177	−0.000 112	0.000 247	0.000 247	0.000 247	0.000 247
1-2	−0.000 456	−0.000 330	2.96×10^{-6}	2.96×10^{-6}	2.96×10^{-6}	2.96×10^{-6}
2-9	−0.000 484	−0.000 354	6.32×10^{-8}	6.30×10^{-8}	6.35×10^{-8}	6.32×10^{-8}
6-8	−0.000 516	−0.000 373	3.38×10^{-7}	3.38×10^{-7}	3.38×10^{-7}	3.38×10^{-7}

Tab. B.1.: MNIST combinations experiments Comparison of cluster formation for all 2-digit subsets of mnist, using simplest autoencoder (sec. 5.1.1), and many different cluster identification metrics.

Subset	Fowl	Sil	Dav
1-7	0.792	0.700	0.703
1-6	0.775	0.703	0.711
0-1	0.767	0.708	0.717
1-9	0.677	0.589	0.596
1-4	0.644	0.543	0.548
6-7	0.586	0.540	0.553
5-9	0.379	0.306	0.307
0-4	0.250	0.199	0.203
4-6	0.245	0.203	0.209
0-2	0.218	0.168	0.171
0-5	0.188	0.145	0.146
5-6	0.166	0.126	0.126
5-8	0.165	0.125	0.125
3-5	0.150	0.112	0.112
3-7	0.146	0.112	0.114
4-5	0.139	0.104	0.104
0-8	0.134	0.101	0.101
5-7	0.0889	0.0671	0.0675
1-5	0.0783	0.0595	0.0601
0-3	0.0776	0.0566	0.0570
0-9	0.0703	0.0513	0.0517
2-6	0.0632	0.0464	0.0467
1-3	0.0335	0.0246	0.0250
4-8	0.0276	0.0200	0.0204
8-9	0.0267	0.0191	0.0195
2-5	0.0244	0.0196	0.0200
7-9	0.0234	0.0169	0.0173
6-9	0.0220	0.0155	0.0160
3-9	0.0214	0.0155	0.0159
2-3	0.0200	0.0144	0.0148
1-8	0.0148	0.009 54	0.009 90
7-8	0.0122	0.008 52	0.008 91
2-4	0.0110	0.008 12	0.008 47
2-8	0.0105	0.007 54	0.007 90
4-9	0.009 30	0.006 58	0.006 98
0-6	0.008 31	0.006 07	0.006 44
2-7	0.005 86	0.004 27	0.004 63
3-6	0.004 85	0.003 33	0.003 70
3-8	0.003 40	0.002 58	0.002 95
3-4	0.000 346	0.000 278	0.000 641
4-7	2.62×10^{-5}	3.98×10^{-5}	0.000 399
0-7	−0.000 177	−0.000 112	0.000 247
1-2	−0.000 456	−0.000 330	2.96×10^{-6}
2-9	−0.000 484	−0.000 354	6.32×10^{-8}
6-8	−0.000 516	−0.000 373	3.38×10^{-7}

Tab. B.1 Continued: Comparison of cluster formation for all 2-digit subsets of mnist, using simplest autoencoder (sec. 5.1.1), and many different cluster identification metrics.

C

Full Results of MNIST Cluster Formation using Different Neural Network Architectures

Hidden Layers	Hidden Features	Parameters	MSE	ARI
4	30	57154	0.0146	0.946
5	30	59014	0.0186	0.835
4	30	57154	0.0128	0.787
3	30	55294	0.0123	0.765
1	10	17334	0.0249	0.751
2	30	53434	0.0185	0.748
4	10	17994	0.0361	0.703
1	30	51574	0.0119	0.700
3	10	17774	0.0240	0.692
3	10	17774	0.0253	0.684
3	30	55294	0.0129	0.680
3	30	55294	0.0115	0.665
1	10	17334	0.0358	0.665
2	500	1817814	0.009 61	0.657
2	30	53434	0.009 48	0.643
1	500	1316814	0.008 71	0.630
2	100	204214	0.005 95	0.594
6	30	60874	0.0209	0.590
1	100	184014	0.006 01	0.581
2	30	53434	0.009 48	0.551
2	10	17554	0.0210	0.549
1	100	184014	0.005 70	0.542
4	10	17994	0.0272	0.525
1	30	51574	0.0108	0.523
1	100	184014	0.005 14	0.512
4	30	57154	0.0129	0.477
1	10	17334	0.0281	0.470
0	0	47854	0.007 11	0.462

Tab. C.1.: Hidden layer experiments Effect of different neural network architectures with variable number of hidden layers and features on cluster identification accuracy (measured using ARI) for unsupervised MNIST clustering for subset 1-2-9. See sec. 5.1.2

Hidden Layers	Hidden Features	Parameters	MSE	ARI
0	0	47854	0.007 15	0.454
2	10	17554	0.0280	0.452
3	10	17774	0.0272	0.452
5	30	59014	0.0159	0.450
4	10	17994	0.0230	0.423
2	10	17554	0.0289	0.414
4	100	244614	0.0108	0.393
5	30	59014	0.0175	0.383
7	30	62734	0.0227	0.353
3	500	2318814	0.0133	0.352
4	100	244614	0.0137	0.344
2	500	1817814	0.004 89	0.340
3	500	2318814	0.007 11	0.336
4	100	244614	0.0110	0.327
4	500	2819814	0.0183	0.325
6	30	60874	0.0212	0.285
0	0	47854	0.007 17	0.284
1	30	51574	0.0104	0.278
7	30	62734	0.0228	0.269
3	100	224414	0.008 39	0.269
8	30	64594	0.0324	0.259
6	30	60874	0.0226	0.247
3	100	224414	0.008 40	0.245
2	100	204214	0.008 51	0.244
4	500	2819814	0.0172	0.231
9	30	66454	0.0244	0.213
7	30	62734	0.0314	0.210
2	100	204214	0.006 71	0.208
1	500	1316814	0.009 24	0.183
1	500	1316814	0.005 73	0.157
9	30	66454	0.0323	0.152
4	500	2819814	0.0178	0.136
2	500	1817814	0.0112	0.0884
10	30	68314	0.0514	0.0440

Tab. C.1 Continued: Hidden layer experiments Effect of different neural network architectures with variable number of hidden layers and features on cluster identification accuracy (measured using ARI) for unsupervised MNIST clustering for subset 1-2-9. See sec. 5.1.2

No. Conv. Layers	No. Conv. Filters	Filter Size	Parameters	MSE	ARI
2	16	3	12594379	0.008 60	0.867
2	32	3	25165083	0.0122	0.845
2	8	3	6312483	0.005 01	0.776
4	8	3	6314819	0.004 67	0.755
4	8	3	6314819	0.005 08	0.748
1	8	5	6311571	0.008 58	0.741
1	32	3	25146587	0.0160	0.686
1	16	5	12590251	0.0123	0.673
3	16	5	12615915	0.0123	0.671
2	8	3	6312483	0.008 58	0.671
1	16	3	12589739	0.0103	0.663
1	16	3	12589739	0.0123	0.660
2	8	5	6314787	0.0104	0.633
3	8	3	6313651	0.005 72	0.625
4	16	3	12603659	0.005 17	0.611
3	8	5	6318003	0.007 18	0.606
1	8	5	6311571	0.008 62	0.604
2	16	3	12594379	0.005 68	0.601
2	8	3	6312483	0.006 09	0.597
1	16	5	12590251	0.0224	0.555
3	16	3	12599019	0.008 47	0.551
1	8	5	6311571	0.0122	0.532
1	32	3	25146587	0.0130	0.530
4	16	3	12603659	0.0104	0.517
2	16	5	12603083	0.0102	0.512
1	16	3	12589739	0.0151	0.499
4	32	3	25202075	0.005 55	0.486
4	32	3	25202075	0.008 17	0.481

Tab. C.2.: Convolutional layer experiments Effect of different neural network architectures with variable number of convolutional layers and filters, with stride of 1 on cluster identification accuracy (measured using ARI) for unsupervised MNIST clustering for subset 1-2-9. See sec. 5.1.3

No. Conv. Layers	No. Conv. Filters	Filter Size	Parameters	MSE	ARI
3	32	3	25183579	0.008 26	0.453
2	32	3	25165083	0.005 48	0.450
4	16	3	12603659	0.0121	0.412
2	32	5	25198875	0.0102	0.409
3	16	3	12599019	0.008 36	0.399
3	32	3	25183579	0.006 20	0.389
2	32	3	25165083	0.005 50	0.380
2	16	5	12603083	0.008 46	0.378
2	16	3	12594379	0.0104	0.368
3	16	3	12599019	0.004 87	0.368
3	8	3	6313651	0.0224	0.366
3	8	5	6318003	0.0120	0.356
1	8	3	6311315	0.006 71	0.352
3	32	3	25183579	0.004 54	0.337
2	8	5	6314787	0.0181	0.331
1	16	5	12590251	0.0123	0.308
3	16	5	12615915	0.008 44	0.295
1	32	5	25147611	0.0128	0.280
1	8	3	6311315	0.005 62	0.248
4	32	5	25301403	0.0923	0.247
4	16	5	12628747	0.0152	0.240
4	8	3	6314819	0.0181	0.235
4	8	5	6321219	0.0922	0.187
2	16	5	12603083	0.005 32	0.148
3	8	3	6313651	0.005 90	0.139

Tab. C.2 Continued: Convolutional layer experiments Effect of different neural network architectures with variable number of convolutional layers and filters, with stride of 1 on cluster identification accuracy (measured using ARI) for unsupervised MNIST clustering for subset 1-2-9. See sec. 5.1.3

No. Conv. Layers	No. Conv. Filters	Filter Size/Stride	Parameters	MSE	ARI
2	32	5	64455	0.007 41	0.956
2	32	5	64455	0.006 38	0.945
3	16	3	17239	0.009 00	0.936
2	16	5	21303	0.008 70	0.935
3	8	3	8191	0.0167	0.931
2	16	3	24311	0.009 21	0.919
2	8	5	9327	0.0140	0.915
2	16	3	24311	0.008 80	0.915
1	16	5	39703	0.009 90	0.914
2	16	3	24311	0.009 14	0.912
2	16	5	21303	0.009 45	0.912
2	32	3	54087	0.006 93	0.909
3	32	3	49159	0.006 37	0.907
2	16	5	21303	0.009 56	0.900
3	16	3	17239	0.0117	0.896
3	32	3	49159	0.008 03	0.880
1	32	5	75655	0.0125	0.877
1	32	3	199559	0.0129	0.872
2	8	3	12879	0.0214	0.871
1	8	3	52703	0.009 04	0.864
1	8	5	21727	0.0115	0.862
2	32	3	54087	0.007 19	0.858
1	16	5	39703	0.009 13	0.853
3	8	3	8191	0.0167	0.847
2	8	3	12879	0.0173	0.839
2	32	5	64455	0.005 98	0.836
1	16	3	101655	0.008 68	0.836
1	16	5	39703	0.009 03	0.830

Tab. C.3.: Convolutional layer experiments with stride Effect of different neural network architectures with variable number of convolutional layers and filters, with stride equal to filter size on cluster identification accuracy (measured using ARI) for unsupervised MNIST clustering for subset 1-2-9. See sec. 5.1.4

No. Conv. Layers	No. Conv. Filters	Filter Size/Stride	Parameters	MSE	ARI
1	32	3	199559	0.005 86	0.809
1	8	3	52703	0.0108	0.807
1	32	5	75655	0.006 69	0.796
1	16	3	101655	0.008 15	0.791
1	16	3	101655	0.0107	0.786
1	32	3	199559	0.0116	0.749
1	8	5	21727	0.0128	0.733
3	32	3	49159	0.006 67	0.714
1	32	5	75655	0.008 32	0.693
1	8	5	21727	0.0145	0.644
2	8	5	9327	0.0136	0.430
3	8	3	8191	0.0183	0.425
2	8	5	9327	0.0142	0.423
1	8	3	52703	0.008 61	0.422
3	16	3	14311	0.0743	0.372
2	32	3	30663	0.0740	0.219

Tab. C.3 Continued: Convolutional layer experiments with stride Effect of different neural network architectures with variable number of convolutional layers and filters, with stride equal to filter size on cluster identification accuracy (measured using ARI) for unsupervised MNIST clustering for subset 1-2-9. See sec. 5.1.4

No. VGG Blocks	No. Conv. Filters	Filter Size	Parameters	MSE	ARI
2	8	3	67775	0.005 89	0.934
3	16	3	55879	0.007 25	0.920
1	16	3	491639	0.008 09	0.910
2	32	3	301319	0.004 61	0.882
3	8	3	24055	0.007 69	0.879
2	16	3	138711	0.005 94	0.879
1	32	3	988743	0.0112	0.870
2	8	3	67775	0.006 28	0.861
3	8	3	24055	0.0106	0.855
2	32	3	301319	0.004 51	0.834
2	8	3	67775	0.008 11	0.828
1	8	3	246543	0.005 67	0.803
1	16	3	491639	0.006 03	0.792
2	16	3	138711	0.009 48	0.787
2	16	3	138711	0.004 63	0.784
1	16	3	491639	0.005 17	0.747
1	8	3	246543	0.006 72	0.734
1	8	3	246543	0.005 38	0.725
1	32	3	988743	0.008 10	0.724
2	32	3	301319	0.004 41	0.666
3	16	3	55879	0.007 43	0.647
1	32	3	988743	0.005 68	0.640
3	8	3	24055	0.0128	0.617
3	16	3	55879	0.007 30	0.550

Tab. C.4.: VGG block experiments Effect of different neural network architectures with variable number of vgg blocks and filters, with stride equal to 1 on cluster identification accuracy (measured using ARI) for unsupervised MNIST clustering for subset 1-2-9. See sec. 5.1.5

D Full Results of MNIST Cluster Formation using Different Loss Functions and Regularization

Loss Function	Loss	No. Parameters	ARI
mse	0.009 84	21 300	0.954
bce	0.108	21 300	0.932
mse	0.009 33	21 300	0.921
bce	0.109	21 300	0.901
bce	0.105	21 300	0.890
mse	0.009 54	21 300	0.836
bce	0.111	21 300	0.834
mse	0.009 80	21 300	0.824

Tab. D.1.: **BCE Loss experiments** Clustering results when comparing MSE and BCE loss functions

Denoise Function	Sigma or P[a]	Loss	No. Parameters	ARI
Masking Noise	0.2	0.0115	21 300	0.936
Gaussian Noise	0.0	0.0141	21 300	0.925
Gaussian Noise	0.0	0.008 47	21 300	0.912
Masking Noise	0.5	0.0130	21 300	0.910
Masking Noise	0.5	0.009 14	21 300	0.907
Gaussian Noise	0.0	0.009 14	21 300	0.906
Gaussian Noise	0.1	0.0111	21 300	0.894
Masking Noise	0.2	0.0139	21 300	0.889
Gaussian Noise	0.2	0.0110	21 300	0.886
Gaussian Noise	0.1	0.009 12	21 300	0.868
Masking Noise	0.5	0.009 12	21 300	0.760
Masking Noise	0.9	0.0160	21 300	0.745
Gaussian Noise	0.0	0.0172	21 300	0.735
Masking Noise	0.5	0.009 87	21 300	0.726
Gaussian Noise	1.0	0.0226	21 300	0.720
Gaussian Noise	0.1	0.0178	21 300	0.620
Gaussian Noise	0.2	0.0148	21 300	0.580
Gaussian Noise	0.1	0.0120	21 300	0.578
Masking Noise	0.9	0.0154	21 300	0.563
Masking Noise	0.2	0.0184	21 300	0.498
Gaussian Noise	0.2	0.0110	21 300	0.468
Gaussian Noise	0.5	0.0119	21 300	0.463
Masking Noise	0.9	0.0135	21 300	0.393
Gaussian Noise	1.0	0.0219	21 300	0.356
Gaussian Noise	1.0	0.0226	21 300	0.349
Gaussian Noise	1.0	0.0222	21 300	0.316
Gaussian Noise	0.5	0.0161	21 300	0.313
Gaussian Noise	0.2	0.0140	21 300	0.275
Gaussian Noise	0.5	0.0105	21 300	0.238
Gaussian Noise	0.5	0.0225	21 300	0.212

Tab. D.2.: Denoising experiments Clustering results from experiments with different levels of denoising using gaussian noise and masking noise.

[a]For masking noise, this is P, the probability that any one pixel is masked, for gaussian noise this is Sigma, the standard deviation of the gaussian noise added to each pixel

Distribution	Initialization	Loss	No. Parameters	ARI
GMM	KMeans	1080	29 000	0.939
GMM	KMeans	1050	29 000	0.938
GMM	KMeans	1120	29 000	0.936
Laplace	Zero Mean	101	29 000	0.912
Laplace	Zero Mean	95.2	29 000	0.879
Laplace	Zero Mean	101	29 000	0.864
Gaussian	Zero Mean	111	29 000	0.711
Gaussian	Zero Mean	120	29 000	0.678
Gaussian	Zero Mean	115	29 000	0.676
Gaussian	Zero Mean	114	29 000	0.673
GMM	Random	1030	29 000	0.661
Gaussian	Zero Mean	113	29 000	0.641
GMM	Random	1000	29 000	0.538
Gaussian	Zero Mean	173	29 000	0.308

Tab. D.3.: **VAE experiments** Clustering results from experiments with different latent distributions and initializations for VAE.

E

Full Results of Maculinea Alcon Cluster Formation using Different Neural Network Architectures

Conv Layers	No. Filters	Filter Size	Parameters	MSE	ARI
4	16	5	48569	0.001 70	0.880
3	64	3	217449	0.001 02	0.859
4	32	5	170185	0.001 60	0.858
3	32	5	118921	0.001 14	0.858
3	32	5	118921	0.001 13	0.858
4	64	3	291305	0.001 19	0.837
3	64	5	438889	0.001 16	0.836
4	64	5	643817	0.001 49	0.836
4	32	3	92233	0.001 25	0.816
2	32	5	67657	0.001 08	0.816
4	64	3	291305	0.001 29	0.815
3	64	7	829609	0.001 40	0.794
3	16	3	29529	0.001 15	0.794
2	32	7	115561	0.001 07	0.793
4	32	3	92233	0.001 37	0.775
3	32	3	73737	0.001 06	0.774
5	32	3	110729	0.001 53	0.773
2	64	5	233961	0.000 942	0.771
4	32	5	170185	0.001 54	0.771
4	16	5	48569	0.001 62	0.754
3	64	5	438889	0.001 14	0.752
2	16	3	24889	0.001 11	0.752
2	64	3	143593	0.001 00	0.751
4	32	3	92233	0.001 35	0.751
2	16	3	24889	0.001 11	0.751
4	16	5	48569	0.001 63	0.750
4	16	7	84809	0.001 72	0.749
3	32	5	118921	0.001 19	0.732

Tab. E.1.: Convolutional Layer Experiments Effect of different neural network architectures with variable number of convolutional layers, filters and filter size on cluster identification accuracy (measured using ARI) for unsupervised Maculinea Alcon clustering by Gender. See sec. 5.3

Conv Layers	No. Filters	Filter Size	Parameters	MSE	ARI
4	16	7	84809	0.001 76	0.715
2	32	5	67657	0.001 05	0.712
2	64	5	233961	0.000 918	0.711
3	16	3	29529	0.001 16	0.710
2	16	5	22905	0.001 13	0.710
3	32	7	215977	0.001 26	0.707
2	16	5	22905	0.001 22	0.674
4	32	7	316393	0.001 96	0.669
3	16	7	59689	0.001 46	0.669
3	32	7	215977	0.001 42	0.668
5	16	3	38809	0.001 68	0.656
2	32	3	55241	0.001 12	0.653
2	64	7	428073	0.001 07	0.651
3	16	7	59689	0.001 36	0.650
4	16	3	34169	0.001 40	0.635
3	64	7	829609	0.001 43	0.632
4	64	3	291305	0.001 30	0.631
4	64	7	1231145	0.001 93	0.617
4	32	7	316393	0.001 84	0.615
2	16	5	22905	0.001 21	0.593
4	16	7	84809	0.001 79	0.592
2	16	3	24889	0.001 06	0.576
4	32	7	316393	0.001 96	0.573
2	32	3	55241	0.001 32	0.542
3	16	7	59689	0.001 39	0.535
4	64	7	1231145	0.001 78	0.533
2	64	7	428073	0.001 06	0.532
4	16	3	34169	0.001 47	0.518
2	32	7	115561	0.001 17	0.504
3	64	5	438889	0.001 10	0.499
2	32	5	67657	0.000 980	0.497
5	16	3	38809	0.001 82	0.483
3	16	3	29529	0.001 27	0.480
3	16	5	35737	0.001 39	0.436
4	32	5	170185	0.001 47	0.374
4	64	5	643817	0.001 53	0.364
3	64	7	829609	0.001 24	0.355
4	64	5	643817	0.001 43	0.349

Tab. E.1 Continued: Convolutional Layer Experiments Effect of different neural network architectures with variable number of convolutional layers, filters and filter size on cluster identification accuracy (measured using ARI) for unsupervised Maculinea Alcon clustering by Gender. See sec. 5.3

Conv Layers	No. Filters	Filter Size	Parameters	MSE	ARI
2	16	7	34569	0.001 23	0.274
3	32	7	215977	0.001 26	0.258
3	64	3	217449	0.001 03	0.147
3	32	3	73737	0.001 08	0.134
3	64	3	217449	0.000 943	0.103
2	32	3	55241	0.000 998	0.0979
2	16	7	34569	0.001 23	0.0730
2	64	5	233961	0.000 976	0.0651
3	32	3	73737	0.001 03	0.0572
2	32	7	115561	0.001 20	0.003 79
3	16	5	35737	0.001 22	−0.000 732
4	16	3	34169	0.001 47	−0.001 81
2	64	3	143593	0.000 947	−0.003 51
4	64	7	1231145	0.002 52	−0.0174

Tab. E.1 Continued: Convolutional Layer Experiments Effect of different neural network architectures with variable number of convolutional layers, filters and filter size on cluster identification accuracy (measured using ARI) for unsupervised Maculinea Alcon clustering by Gender. See sec. 5.3

Hidden Layers	No. Hid. Features	No. Lat. Features	Parameters	MSE	ARI
2	128	10	143085	0.001 17	0.880
1	128	30	115201	0.001 02	0.880
1	10	10	110061	0.001 20	0.880
2	100	100	153391	0.000 959	0.880
3	100	100	173591	0.001 10	0.859
4	128	128	239459	0.001 17	0.858
3	128	128	206435	0.001 20	0.858
2	128	128	173411	0.000 952	0.858
1	128	128	140387	0.000 941	0.858
3	30	30	118921	0.001 12	0.858
2	128	128	173411	0.000 945	0.857
1	30	30	115201	0.001 01	0.857
1	128	128	140387	0.000 886	0.836
1	100	100	133191	0.000 967	0.836
2	128	10	143085	0.001 24	0.816
3	128	128	206435	0.001 07	0.815
2	100	100	153391	0.000 940	0.815
1	128	128	140387	0.000 917	0.814
1	100	100	133191	0.000 918	0.814
1	300	300	184591	0.000 879	0.814
1	100	100	133191	0.000 925	0.814
4	30	30	120781	0.001 20	0.814
1	128	10	110061	0.001 15	0.814
2	128	10	143085	0.001 10	0.794
4	10	10	110721	0.001 34	0.794
1	128	10	110061	0.001 18	0.794
1	10	10	110061	0.001 24	0.793
1	128	30	115201	0.001 07	0.792
1	128	10	110061	0.001 16	0.771
4	128	128	239459	0.001 32	0.754
4	10	10	110721	0.001 33	0.754
1	10	10	110061	0.001 22	0.753
2	100	100	153391	0.000 983	0.752
3	128	10	176109	0.001 26	0.735
3	128	30	181249	0.001 13	0.714
4	100	100	193791	0.001 34	0.709
1	30	30	115201	0.000 968	0.695
3	30	30	118921	0.001 16	0.671

Tab. E.2.: Hidden Layer Experiments Effect of different neural network architectures with variable number of hidden layers, hidden features and latent features on cluster identification accuracy (measured using ARI) for unsupervised Maculinea Alcon clustering by Gender. See sec. 5.3

Hidden Layers	No. Hid. Features	No. Lat. Features	Parameters	MSE	ARI
1	128	30	115201	0.000 973	0.611
4	30	30	120781	0.001 23	0.610
2	128	30	148225	0.001 11	0.603
3	10	10	110501	0.001 24	0.598
4	100	100	193791	0.001 33	0.590
1	30	30	115201	0.001 01	0.580
3	128	128	206435	0.001 15	0.566
4	10	10	110721	0.001 50	0.566
3	128	30	181249	0.001 14	0.562
3	128	10	176109	0.001 15	0.554
2	128	30	148225	0.001 06	0.543
4	100	100	193791	0.001 34	0.538
4	30	30	120781	0.001 25	0.517
3	100	100	173591	0.001 15	0.465
3	10	10	110501	0.001 36	0.445
2	10	10	110281	0.001 21	0.419
3	128	10	176109	0.001 16	0.358
2	30	30	117061	0.001 10	0.335
3	10	10	110501	0.001 26	0.319
2	30	30	117061	0.001 03	0.319
3	128	30	181249	0.001 08	0.301
2	128	30	148225	0.001 05	0.239
2	128	128	173411	0.000 984	0.217
4	128	128	239459	0.001 16	0.210
2	30	30	117061	0.001 02	0.0872
2	10	10	110281	0.001 19	0.008 46
2	10	10	110281	0.001 20	0.004 13

Tab. E.2 Continued: Hidden Layer Experiments Effect of different neural network architectures with variable number of hidden layers, hidden features and latent features on cluster identification accuracy (measured using ARI) for unsupervised Maculinea Alcon clustering by Gender. See sec. 5.3

Loss Function	**Parameters**	**ARI**
BCE	140387	0.814
BCE	140387	0.814
GiMMPPy EEK	190181	0.770
LiPPy	157028	0.736
GiMMPPy EEK	190181	0.726
LiPPy	157028	0.724
GiMMPPy EEK	190181	0.722
GiMMPPy EEK	190181	0.716
LiPPy	157028	0.714
LiPPy	157028	0.706
BCE	140387	0.650

Tab. E.3.: Loss Function Experiments Effect of different neural network architectures with variable number of hidden layers, hidden features and latent features on cluster identification accuracy (measured using ARI) for unsupervised Maculinea Alcon clustering by Gender. See sec. 5.3.3

F

Maculinea Alcon Results and Visualization of Ground Truth Clusterings

In this appendix we show the t-SNE results on the latent space of the different models using different inputs. Specifically, we show models trained on three different input image sets: dorsal-only, ventral-only and a 6-channel combined dorsal-ventral input. We compare results on three different loss functions: MSE, GiMMPPy EEK-means, LiPPy. For the variational loss functions (LiPPy and GiMMPPy), we show both the distribution of all the samples from the posterior, and the distribution of the means of the posterior in latent space.

Model	Loss Function	Valid ARI	Test ARI	Valid ARI - MP[a]	Test ARI - MP
Combined	MSE	0.920	0.598	0.	0.
Combined	MSE	0.897	0.598	0.	0.
Combined	MSE	0.875	0.655	0.	0.
Combined	MSE	0.875	0.562	0.	0.
Dorsal	MSE	0.858	0.731	0.	0.
Dorsal	MSE	0.836	0.711	0.	0.
Dorsal	MSE	0.814	0.692	0.	0.
Combined	LiPPy	0.804	0.608	0.831	0.617
Combined	LiPPy	0.783	0.593	0.810	0.580
Combined	GiMMPPy	0.779	0.611	0.789	0.617
Dorsal	GiMMPPy	0.770	0.702	0.837	0.711
Combined	GiMMPPy	0.758	0.624	0.810	0.655
Combined	GiMMPPy	0.750	0.606	0.810	0.655
Combined	LiPPy	0.744	0.575	0.831	0.580
Dorsal	LiPPy	0.736	0.694	0.837	0.711
Dorsal	GiMMPPy	0.730	0.667	0.837	0.690
Dorsal	LiPPy	0.724	0.687	0.816	0.711
Dorsal	GiMMPPy	0.722	0.678	0.816	0.692
Dorsal	GiMMPPy	0.716	0.680	0.837	0.711
Dorsal	LiPPy	0.712	0.678	0.772	0.690
Dorsal	LiPPy	0.706	0.676	0.837	0.711
Combined	GiMMPPy	0.350	0.354	0.376	0.312
Ventral	MSE	0.0244	0.103	0.	0.
Ventral	GiMMPPy	0.0212	0.0122	0.008 10	0.0220
Ventral	LiPPy	0.0148	0.0181	0.006 48	0.0197
Ventral	LiPPy	0.0131	0.0214	0.0305	0.0298
Ventral	LiPPy	0.0111	0.0226	0.003 31	0.0215
Ventral	GiMMPPy	0.007 17	0.0114	−0.002 06	0.007 97
Ventral	GiMMPPy	0.003 33	0.008 84	0.002 89	0.003 04
Ventral	GiMMPPy	0.003 19	0.0149	−0.003 44	0.0186
Ventral	MSE	0.000 229	0.0443	0.	0.
Ventral	LiPPy	−0.000 341	−0.000 477	0.004 13	0.0351
Ventral	MSE	−0.000 528	0.128	0.	0.
Combined	LiPPy	−0.000 681	0.003 83	0.726	0.460
Ventral	MSE	−0.004 00	0.0738	0.	0.

Tab. F.1.: Dorsal, Ventral, Combined Experiments: Results of experiments with final Maculinea Alcon architecture, on gender feature, comparing different inputs - Dorsal images only, Ventral images only and a 6-channel combination image.

[a]MP: Mean of the Posterior. This is 0 for MSE because it has no posterior.

Model	Loss	Latent[a]	Feature	Train Acc	Val Acc	Test Acc	Rand Acc[b]	Test/Rand[c]
Combined	MSE	-	Gender	0.980	0.976	0.953	0.490	1.94
Dorsal	MSE	-	Gender	0.984	0.970	0.971	0.507	1.91
Dorsal	LiPPy	Mean	Gender	0.958	0.953	0.953	0.510	1.87
Dorsal	GiMMPPy	All	Gender	0.952	0.952	0.942	0.504	1.87
Combined	LiPPy	All	Gender	0.956	0.963	0.938	0.507	1.85
Combined	LiPPy	Mean	Gender	0.959	0.964	0.947	0.513	1.84
Dorsal	GiMMPPy	Mean	Gender	0.977	0.964	0.976	0.530	1.84
Dorsal	LiPPy	All	Gender	0.961	0.947	0.928	0.507	1.83
Combined	GiMMPPy	All	Gender	0.940	0.952	0.924	0.506	1.83
Ventral	MSE	-	Gender	0.903	0.893	0.888	0.524	1.70
Ventral	GiMMPPy	Mean	Gender	0.721	0.692	0.729	0.489	1.49
Ventral	LiPPy	All	Gender	0.712	0.670	0.724	0.505	1.44
Ventral	LiPPy	Mean	Gender	0.717	0.657	0.718	0.509	1.41
Ventral	GiMMPPy	All	Gender	0.628	0.609	0.630	0.507	1.24
Combined	GiMMPPy	Mean	Gender	0.549	0.556	0.615	0.508	1.21
Dorsal	MSE	-	Country	0.919	0.953	0.929	0.855	1.09
Dorsal	LiPPy	Mean	Country	0.919	0.953	0.929	0.856	1.09
Dorsal	GiMMPPy	Mean	Country	0.919	0.953	0.929	0.860	1.08
Dorsal	GiMMPPy	All	Country	0.919	0.953	0.929	0.862	1.08
Ventral	LiPPy	Mean	Country	0.932	0.964	0.929	0.864	1.08
Dorsal	LiPPy	All	Country	0.919	0.953	0.929	0.864	1.08
Ventral	MSE	-	Country	0.932	0.959	0.918	0.860	1.07
Combined	LiPPy	Mean	Country	0.924	0.947	0.911	0.854	1.07
Ventral	GiMMPPy	Mean	Country	0.923	0.953	0.912	0.855	1.07
Ventral	LiPPy	All	Country	0.928	0.956	0.915	0.860	1.06
Combined	LiPPy	All	Country	0.929	0.947	0.913	0.862	1.06
Combined	GiMMPPy	All	Country	0.924	0.947	0.911	0.861	1.06
Ventral	GiMMPPy	All	Country	0.923	0.953	0.912	0.863	1.06
Combined	MSE	-	Country	0.924	0.947	0.911	0.863	1.06
Combined	GiMMPPy	Mean	Country	0.924	0.947	0.911	0.867	1.05

Tab. F.2.: Classification results: Results of best experiment for each model-loss combination on classification accuracy for a given feature of interest. Here country is actually only two choices: Denmark or International. Rand represents the accuracy of a random permutation of the dataset.

[a] Kind of latent space used for variational models. Mean means using the mean of the posterior as the latent space, all means using all the samples from the posterior as the latent space

[b] Random accuracy created by randomly shuffling the whole dataset, and assigned the labels as gt labels. This allows us to account for the distribution being uneven across all classes.

[c] This is the test accuracy divided by the random accuracy, to give an idea how much better the model is than random. Higher numbers here are better.

Model	Loss	Latent[a]	Feature	Train MSE	Val MSE	Test MSE	Rand MSE[b]	Test/Rand[c]
Ventral	MSE	-	Longitude	41.4	2.58	2.36	60.2	0.0391
Ventral	GiMMPPy	Mean	Longitude	41.5	2.54	2.37	59.2	0.0401
Ventral	LiPPy	Mean	Longitude	41.5	2.62	2.40	58.9	0.0407
Ventral	GiMMPPy	All	Longitude	41.6	2.61	2.46	59.4	0.0414
Ventral	LiPPy	All	Longitude	41.6	2.87	2.64	59.3	0.0444
Dorsal	MSE	-	Longitude	41.4	2.00	2.83	59.8	0.0473
Dorsal	LiPPy	Mean	Longitude	41.5	2.05	2.90	60.7	0.0477
Dorsal	GiMMPPy	All	Longitude	41.6	2.15	2.95	59.5	0.0497
Dorsal	LiPPy	All	Longitude	41.6	2.22	3.08	59.7	0.0516
Dorsal	GiMMPPy	Mean	Longitude	42.4	2.94	3.73	59.0	0.0633
Dorsal	LiPPy	Mean	Altitude (m)	118 000	17 300	68 800	190 000	0.362
Dorsal	LiPPy	All	Altitude (m)	118 000	17 700	69 100	191 000	0.362
Dorsal	MSE	-	Altitude (m)	118 000	18 200	69 300	192 000	0.362
Dorsal	GiMMPPy	All	Altitude (m)	118 000	18 600	69 500	187 000	0.371
Combined	GiMMPPy	All	Altitude (m)	118 000	18 100	72 200	190 000	0.379
Combined	LiPPy	All	Altitude (m)	118 000	18 700	72 500	191 000	0.381
Combined	LiPPy	Mean	Altitude (m)	118 000	18 400	72 400	189 000	0.384
Combined	GiMMPPy	Mean	Altitude (m)	118 000	18 800	72 600	187 000	0.389
Combined	MSE	-	Altitude (m)	118 000	18 300	72 400	173 000	0.418
Dorsal	LiPPy	Mean	Latitude	7.11	2.91	5.33	12.6	0.424
Dorsal	GiMMPPy	Mean	Latitude	7.13	2.85	5.29	12.4	0.426
Dorsal	GiMMPPy	Mean	Altitude (m)	118 000	18 700	69 600	163 000	0.427
Dorsal	MSE	-	Latitude	7.18	3.04	5.51	12.5	0.441
Ventral	LiPPy	Mean	Latitude	6.55	4.43	6.31	13.0	0.486
Ventral	GiMMPPy	Mean	Latitude	6.57	4.45	6.34	12.9	0.491
Combined	LiPPy	Mean	Latitude	6.98	2.91	6.10	12.4	0.493
Dorsal	LiPPy	Mean	Year	277	310	270	544	0.497
Combined	GiMMPPy	Mean	Latitude	6.97	2.90	6.10	12.3	0.497
Ventral	MSE	-	Latitude	6.61	4.51	6.40	12.2	0.523
Combined	MSE	-	Latitude	7.55	3.73	6.79	12.6	0.539
Combined	LiPPy	Mean	Year	290	274	278	511	0.545
Combined	GiMMPPy	All	Latitude	7.70	3.63	6.82	12.5	0.545
Ventral	LiPPy	Mean	Altitude (m)	98 700	58 200	116 000	194 000	0.598
Ventral	LiPPy	Mean	Year	276	248	314	519	0.606
Ventral	LiPPy	All	Altitude (m)	98 700	58 200	116 000	191 000	0.608
Ventral	GiMMPPy	All	Altitude (m)	98 500	57 800	116 000	187 000	0.620
Dorsal	GiMMPPy	All	Latitude	9.24	5.18	7.70	12.2	0.630
Ventral	GiMMPPy	Mean	Altitude (m)	98 600	57 900	116 000	182 000	0.636

Tab. F.3.: Regression results: MSE of best experiment for each model-loss combination for a given feature of interest.

[a] Kind of latent space used for variational models. Mean means using the mean of the posterior as the latent space, all means using all the samples from the posterior as the latent space

[b] Random MSE created by randomly shuffling the whole dataset, and assigned the labels as gt labels. This allows us to account for the distribution being uneven across all classes.

[c] This is the test MSE divided by the random MSE, to give an idea how much better the model is than random. Lower numbers here are better.

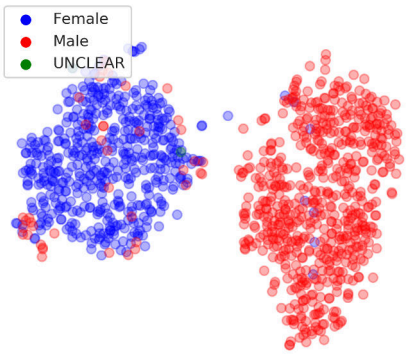
Model	Loss	Latent[a]	Feature	Train MSE	Val MSE	Test MSE	Rand MSE[b]	Test/Rand[c]
Combined	LiPPy	All	Latitude	10.7	7.78	11.2	12.4	0.900
Dorsal	MSE	-	Year	373	426	517	535	0.967
Dorsal	LiPPy	All	Latitude	14.0	11.3	13.8	12.4	1.12
Ventral	LiPPy	All	Latitude	13.9	14.0	16.1	12.4	1.30
Combined	MSE	-	Longitude	22.0	2.07	94.7	60.4	1.57
Combined	LiPPy	Mean	Longitude	22.0	2.05	94.6	59.9	1.58
Combined	GiMMPPy	All	Longitude	22.0	2.07	94.6	59.6	1.59
Combined	LiPPy	All	Longitude	22.0	2.15	94.9	59.6	1.59
Combined	GiMMPPy	Mean	Longitude	22.2	2.14	95.2	59.5	1.60
Combined	MSE	-	Year	943	1120	1010	532	1.89
Ventral	GiMMPPy	All	Latitude	33.7	40.5	40.9	12.1	3.38
Combined	LiPPy	All	Year	5810	6380	6630	536	12.4
Ventral	GiMMPPy	Mean	Year	6720	6850	7740	567	13.6
Dorsal	GiMMPPy	Mean	Year	11 200	11 200	9570	557	17.2
Ventral	GiMMPPy	All	Year	9070	8740	10 100	536	18.8
Dorsal	LiPPy	All	Year	8370	10 400	10 700	548	19.5
Ventral	LiPPy	All	Year	9070	11 800	12 000	528	22.7
Dorsal	GiMMPPy	All	Year	14 100	13 700	12 400	530	23.3
Combined	GiMMPPy	Mean	Year	15 100	13 200	13 800	511	27.0
Combined	GiMMPPy	All	Year	17 100	15 100	15 700	539	29.2

Tab. F.3 Continued: **Regression results:** MSE of best experiment for each model-loss combination for a given feature of interest.

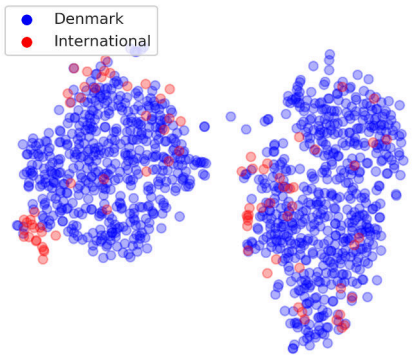
[a] Kind of latent space used for variational models. Mean means using the mean of the posterior as the latent space, all means using all the samples from the posterior as the latent space

[b] Random MSE created by randomly shuffling the whole dataset, and assigned the labels as gt labels. This allows us to account for the distribution being uneven across all classes.

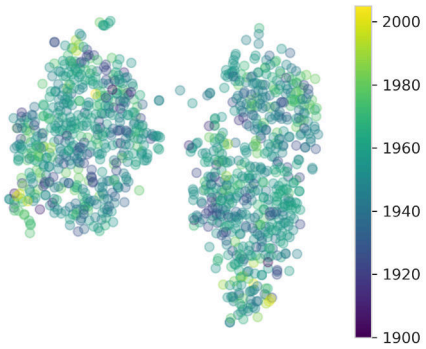
[c] This is the test MSE divided by the random MSE, to give an idea how much better the model is than random. Lower numbers here are better.



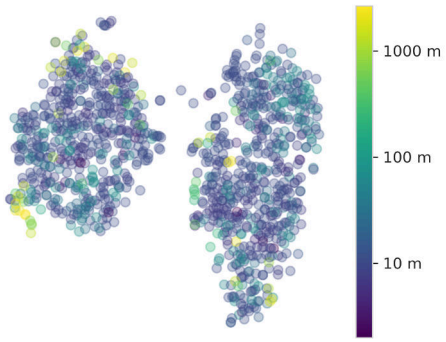
(a) Gender



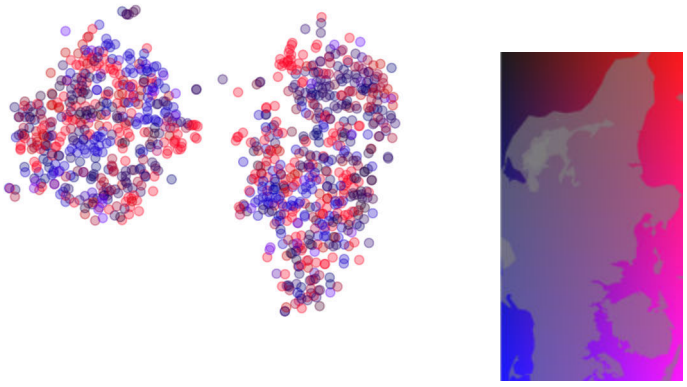
(b) Country



(c) Year Collected

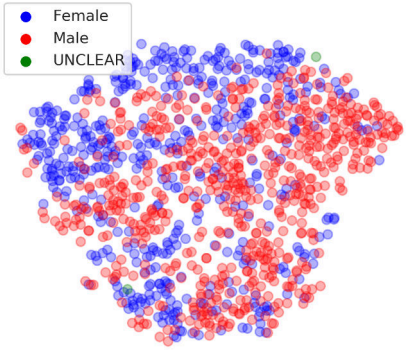


(d) Altitude

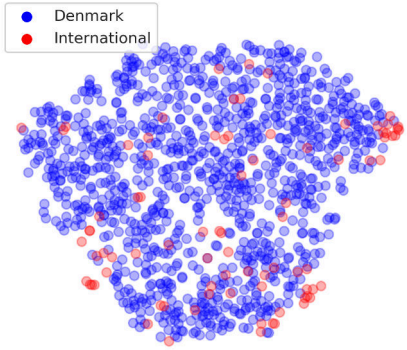


(e) Latitude and Longitude

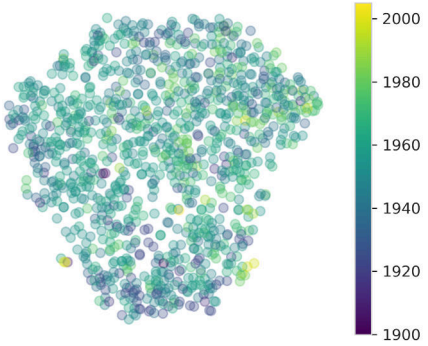
Fig. F.1.: Dorsal MSE: 2D t-SNE visualizations of distribution of dataset in latent space.



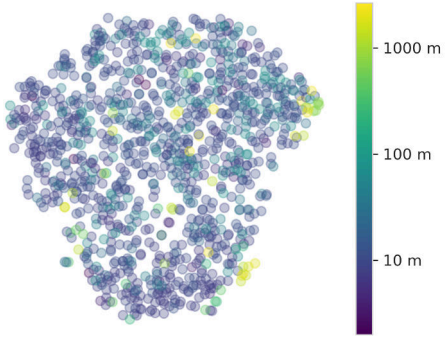
(a) Gender



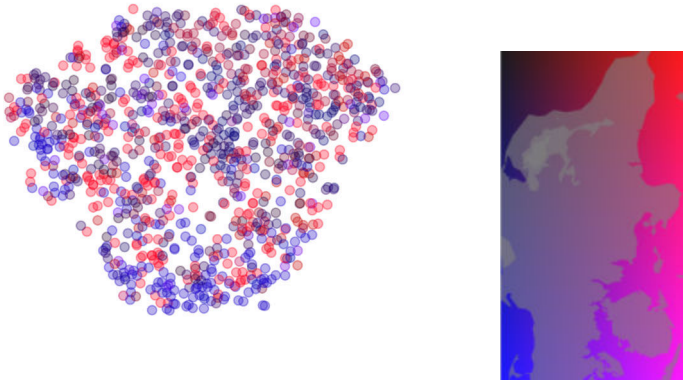
(b) Country



(c) Year Collected

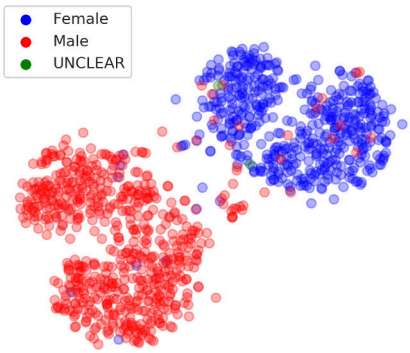


(d) Altitude

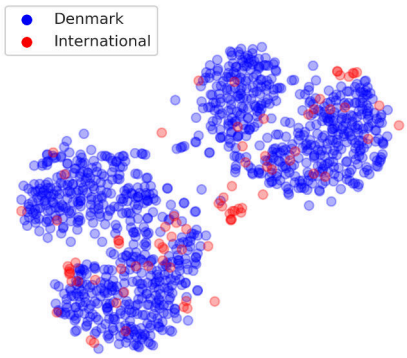


(e) Latitude and Longitude

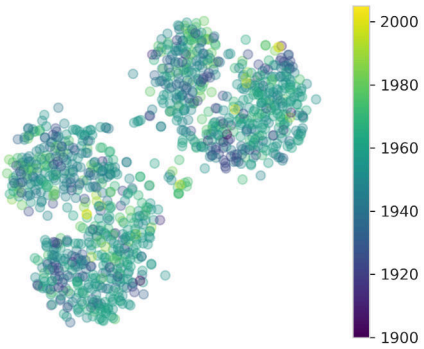
Fig. F.2.: **Ventral MSE:** 2D t-SNE visualizations of distribution of dataset in latent space.



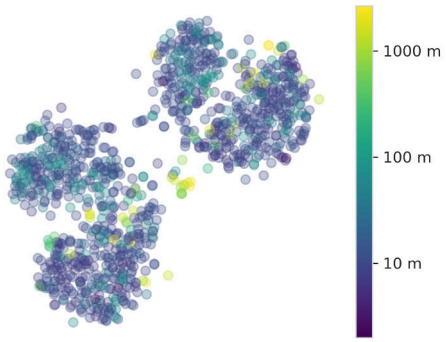
(a) Gender



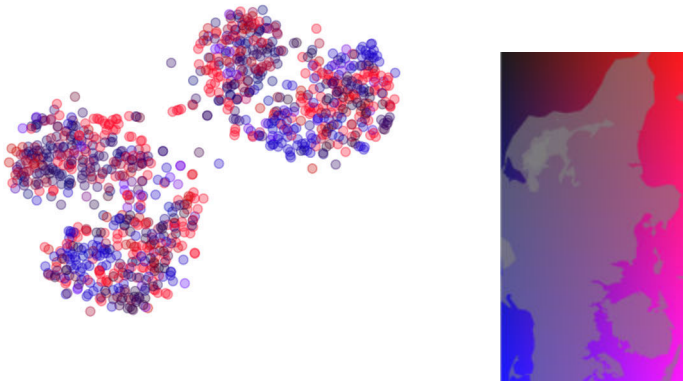
(b) Country



(c) Year Collected

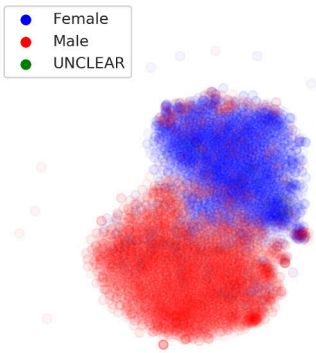


(d) Altitude

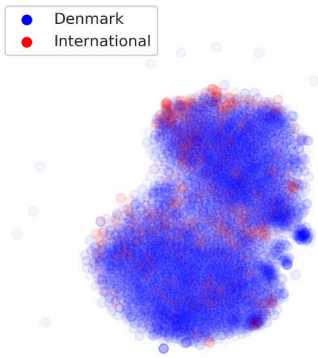


(e) Latitude and Longitude

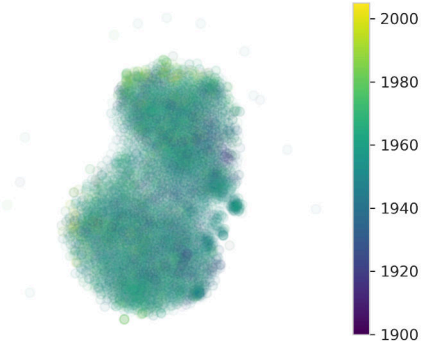
Fig. F.3.: **Combined MSE:** 2D t-SNE visualizations of distribution of dataset in latent space.



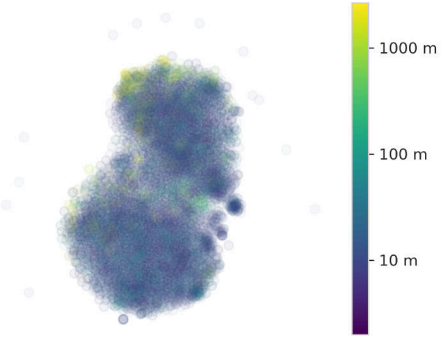
(a) Gender



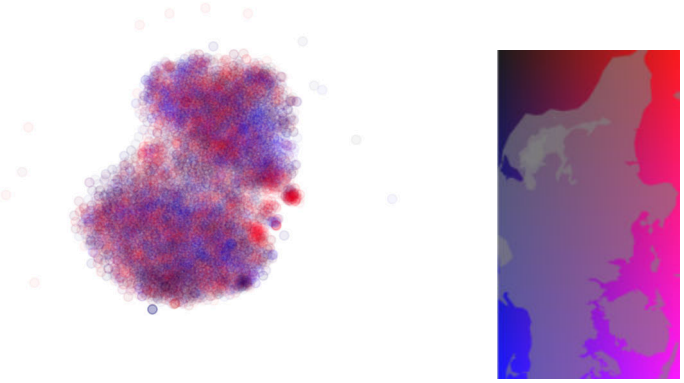
(b) Country



(c) Year Collected

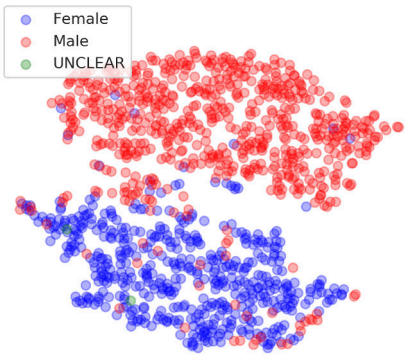


(d) Altitude

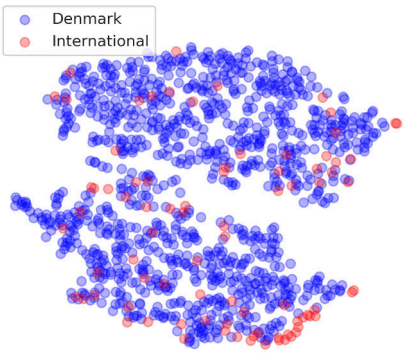


(e) Latitude and Longitude

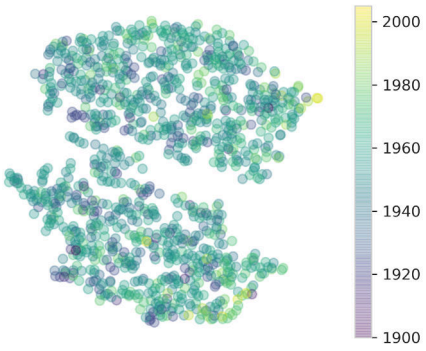
Fig. F.4.: Dorsal LiPPy: 2D t-SNE visualizations of samples from posterior of dataset in latent space.



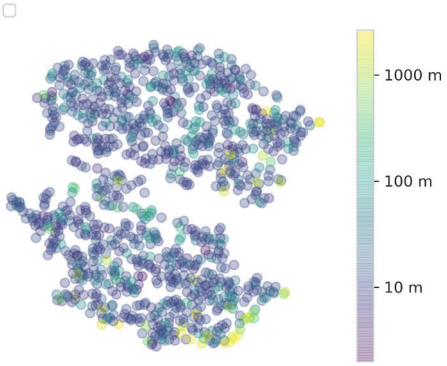
(a) Gender



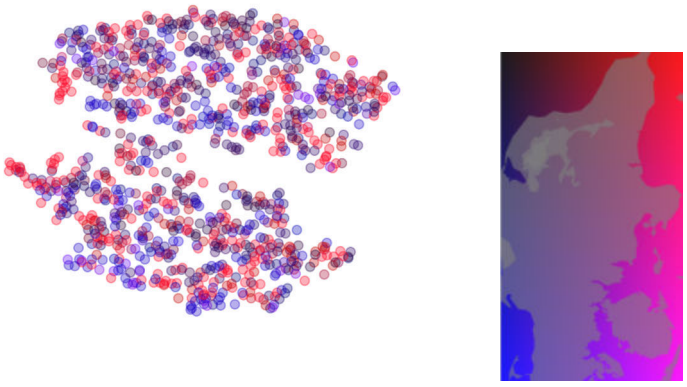
(b) Country



(c) Year Collected

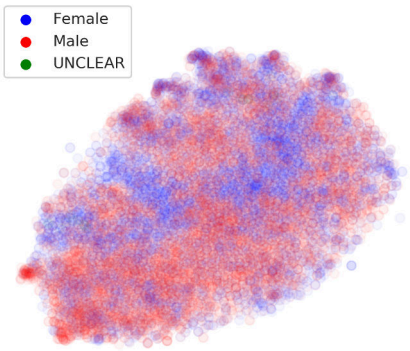


(d) Altitude

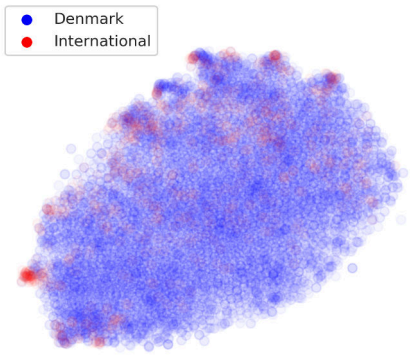


(e) Latitude and Longitude

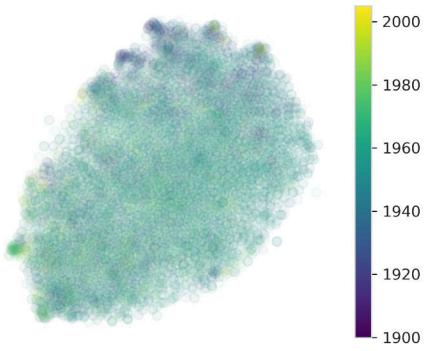
Fig. F.5.: Dorsal LiPPy Means: 2D t-SNE visualizations of distribution of means of posterior of dataset in latent space.



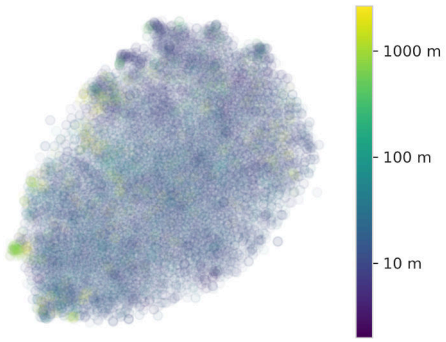
(a) Gender



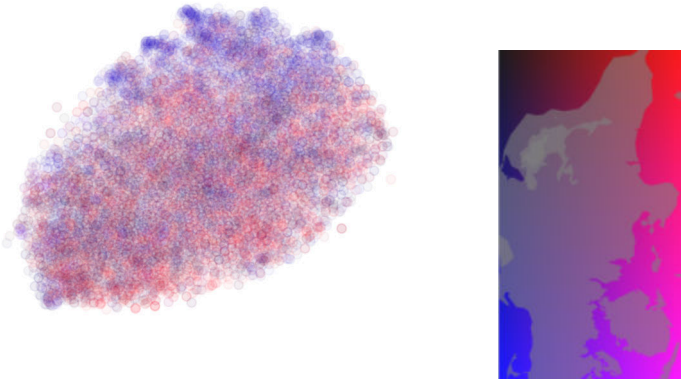
(b) Country



(c) Year Collected

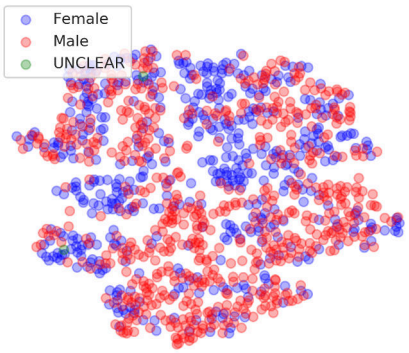


(d) Altitude

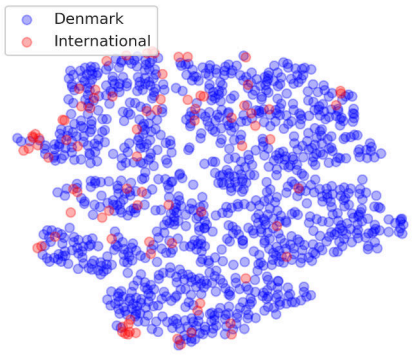


(e) Latitude and Longitude

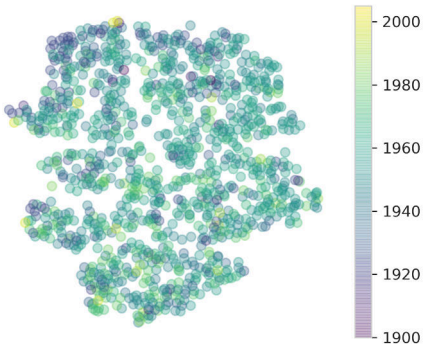
Fig. F.6.: Ventral LiPPy: 2D t-SNE visualizations of samples from posterior of dataset in latent space.



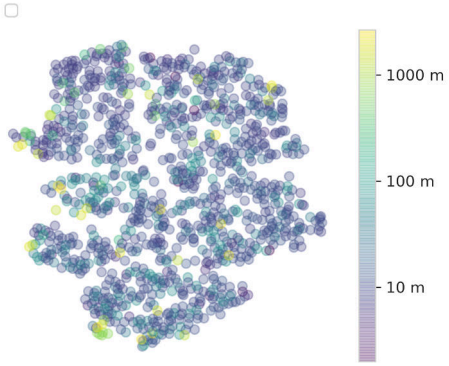
(a) Gender



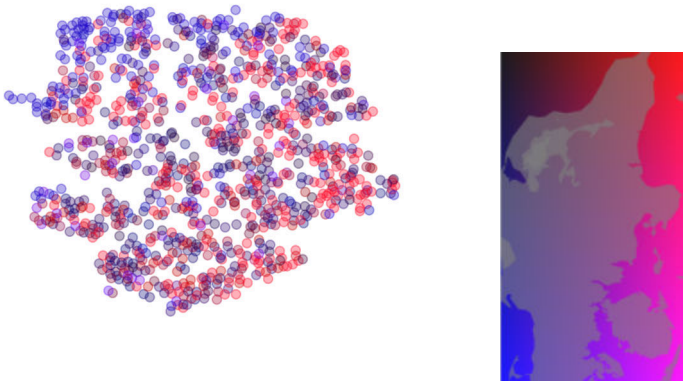
(b) Country



(c) Year Collected

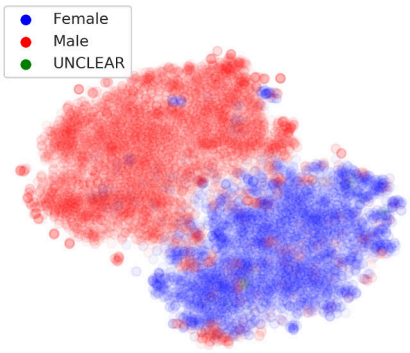


(d) Altitude

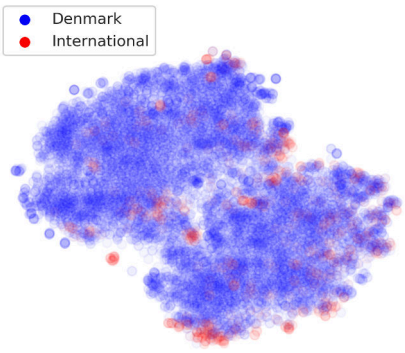


(e) Latitude and Longitude

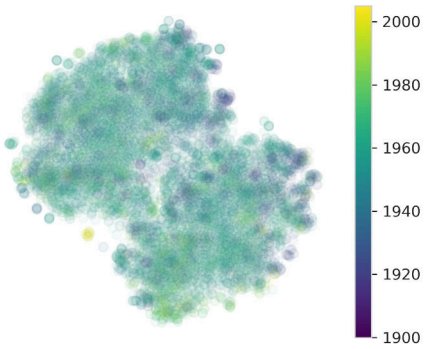
Fig. F.7.: Ventral LiPPy Means: 2D t-SNE visualizations of distribution of means of posterior of dataset in latent space.



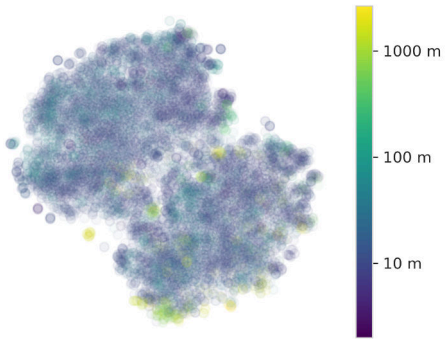
(a) Gender



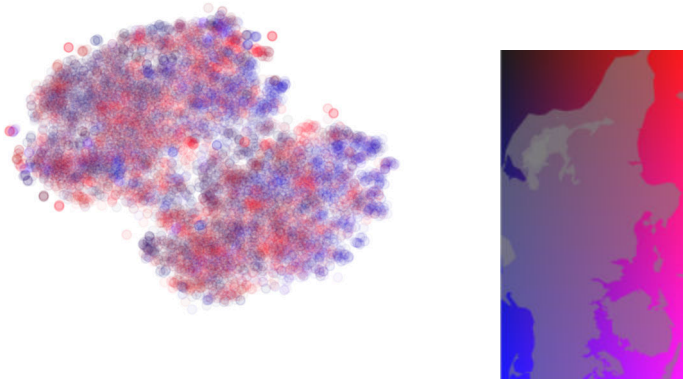
(b) Country



(c) Year Collected

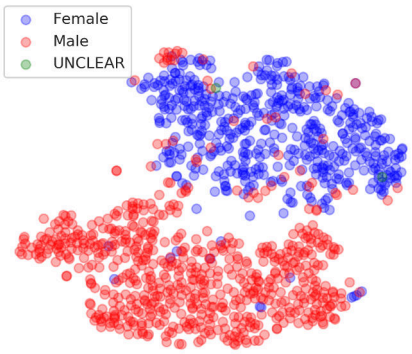


(d) Altitude

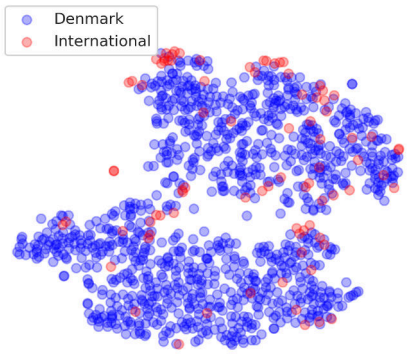


(e) Latitude and Longitude

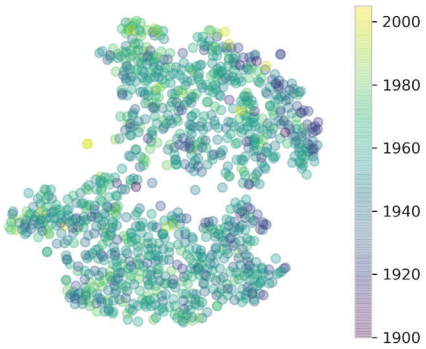
Fig. F.8.: Combined LiPPy: 2D t-SNE visualizations of samples from posterior of dataset in latent space.



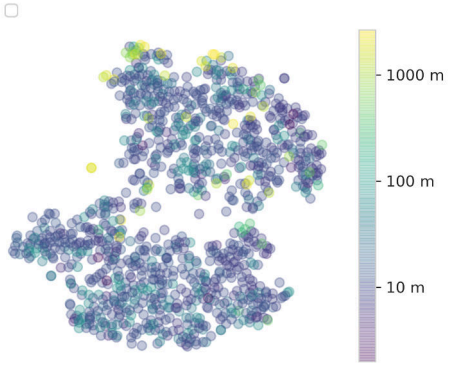
(a) Gender



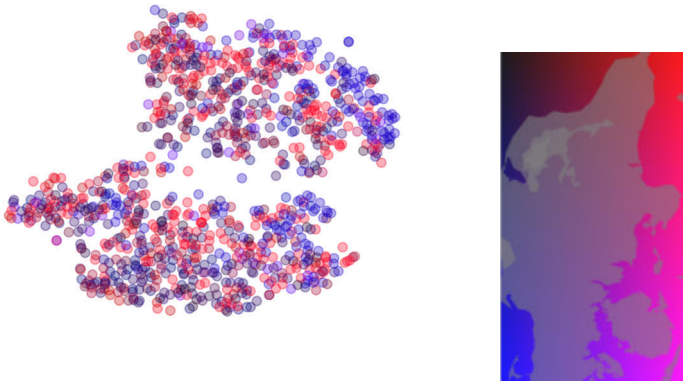
(b) Country



(c) Year Collected

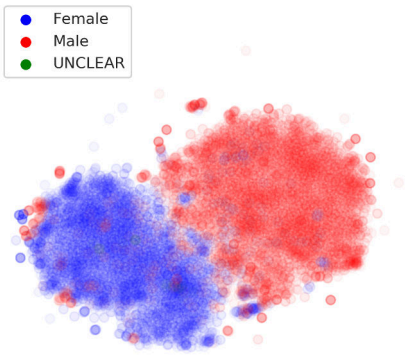


(d) Altitude

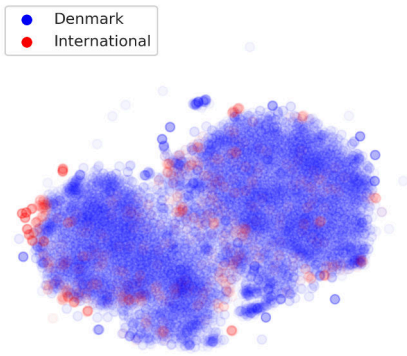


(e) Latitude and Longitude

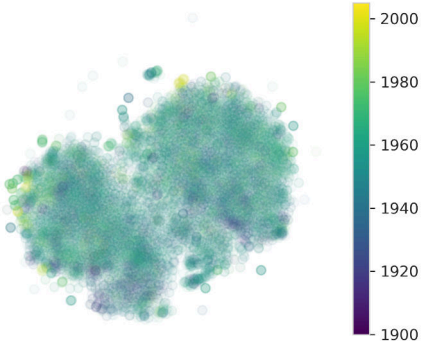
Fig. F.9.: Combined LiPPy Means: 2D t-SNE visualizations of distribution of means of posterior of dataset in latent space.



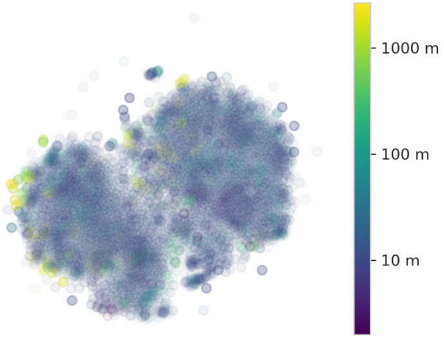
(a) Gender



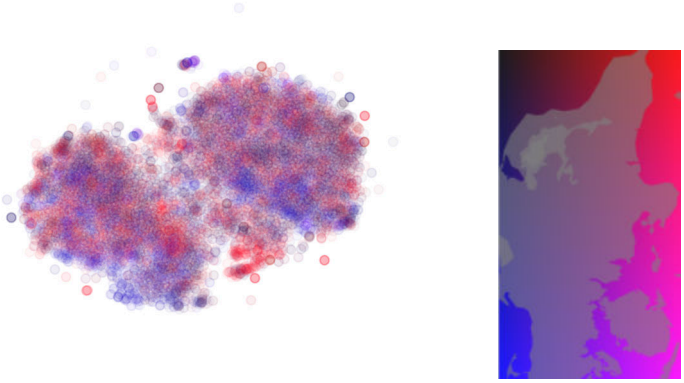
(b) Country



(c) Year Collected

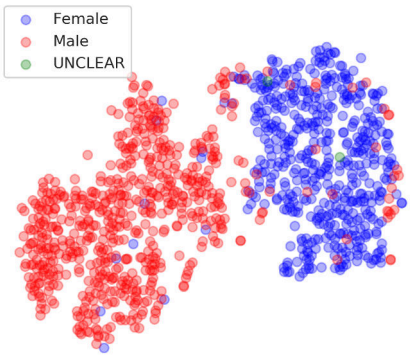


(d) Altitude

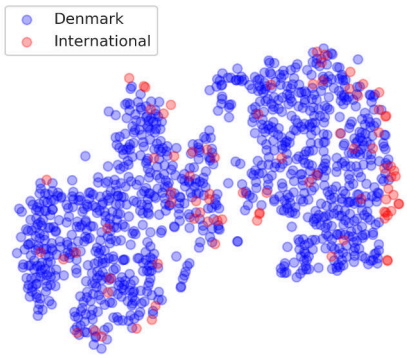


(e) Latitude and Longitude

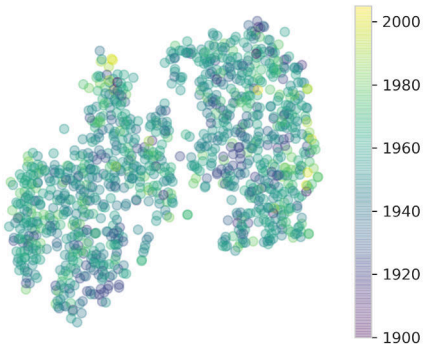
Fig. F.10.: Dorsal GiMMPPy EEK-means: 2D t-SNE visualizations of samples from posterior of dataset in latent space.



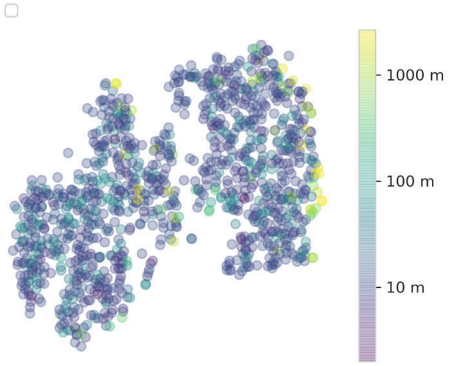
(a) Gender



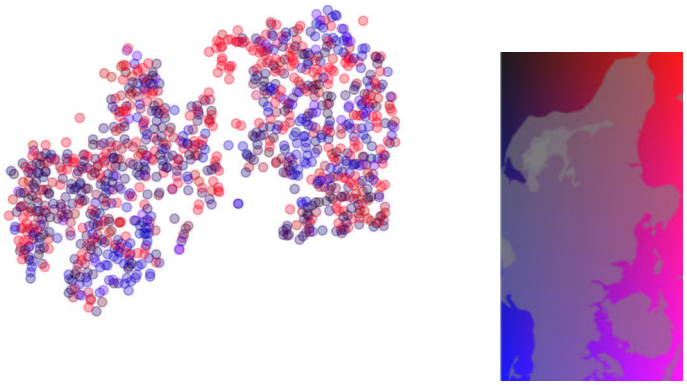
(b) Country



(c) Year Collected

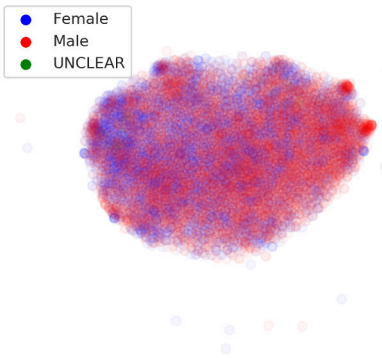


(d) Altitude

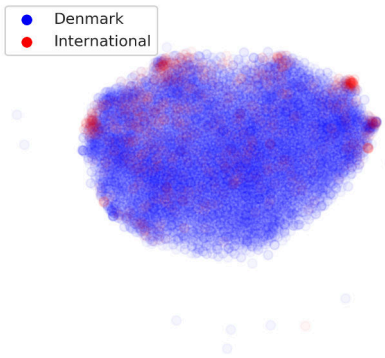


(e) Latitude and Longitude

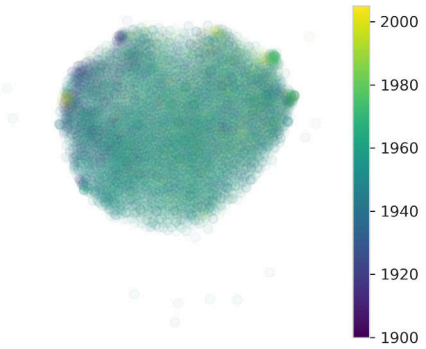
Fig. F.11.: Dorsal GiMMPPy EEK-means Means: 2D t-SNE visualizations of means of posterior of dataset in latent space.



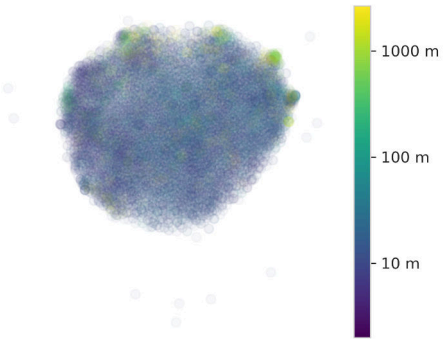
(a) Gender



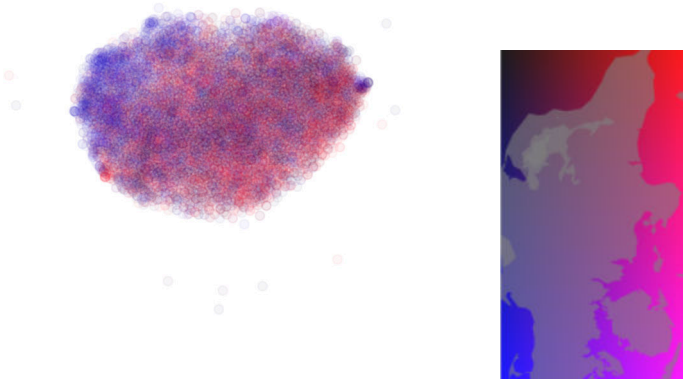
(b) Country



(c) Year Collected

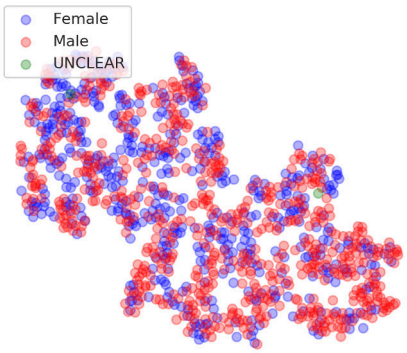


(d) Altitude

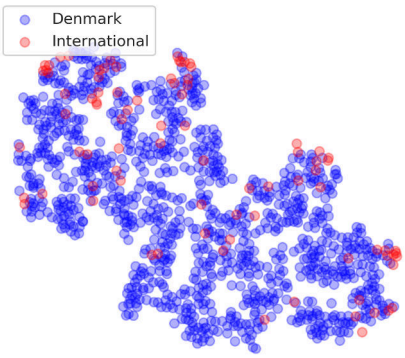


(e) Latitude and Longitude

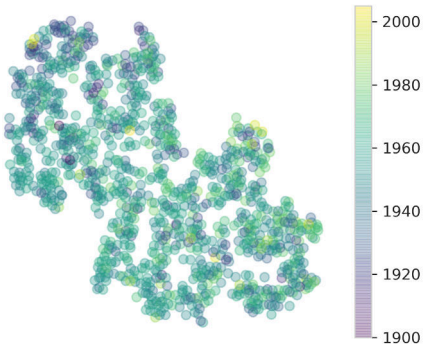
Fig. F.12.: Ventral GiMMPPy EEK-means: 2D t-SNE visualizations of samples from posterior of dataset in latent space.



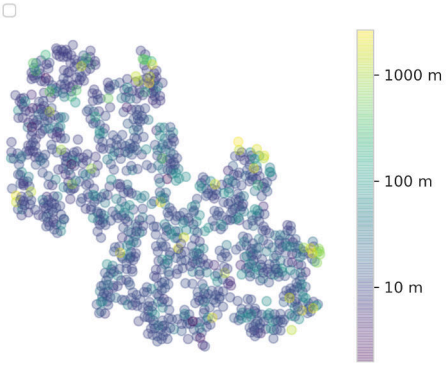
(a) Gender



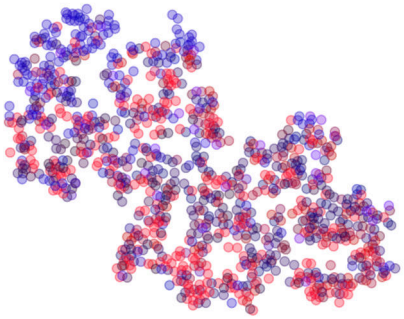
(b) Country



(c) Year Collected

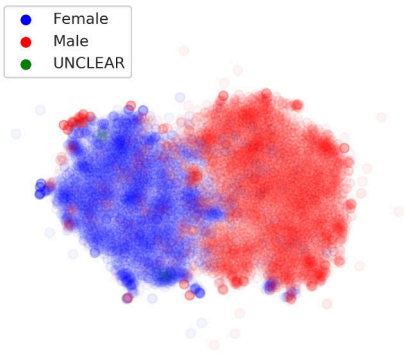


(d) Altitude

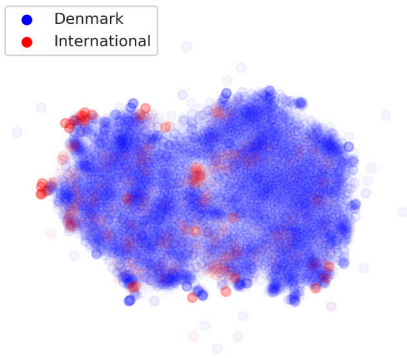


(e) Latitude and Longitude

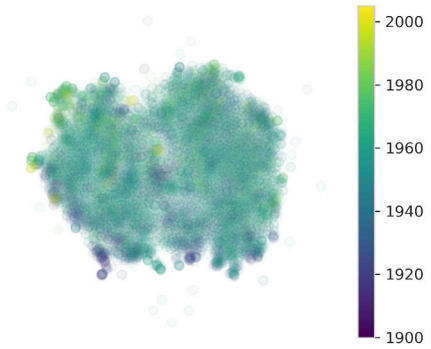
Fig. F.13.: Ventral GiMMPPy EEK-means Means: 2D t-SNE visualizations of samples from posterior of dataset in latent space.



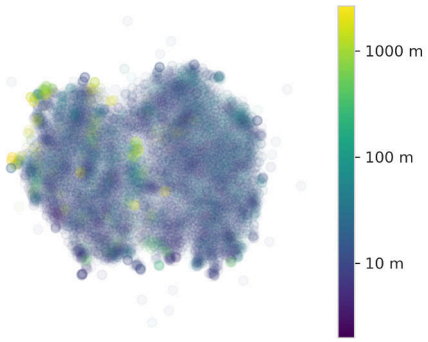
(a) Gender



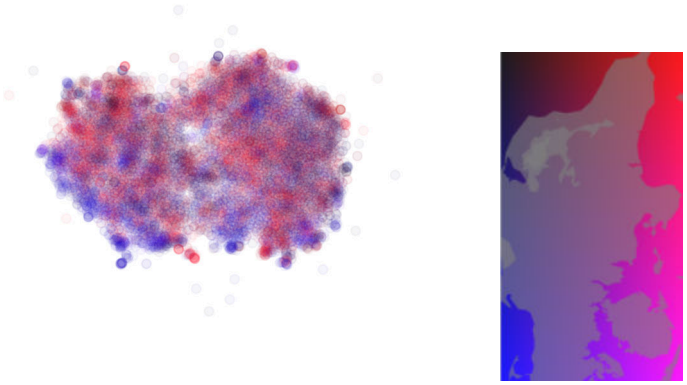
(b) Country



(c) Year Collected

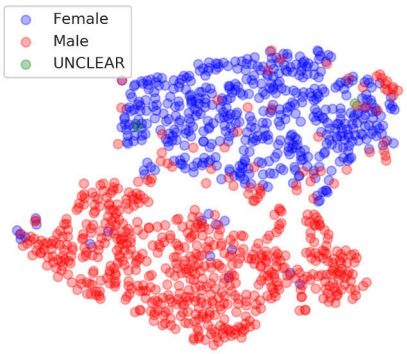


(d) Altitude

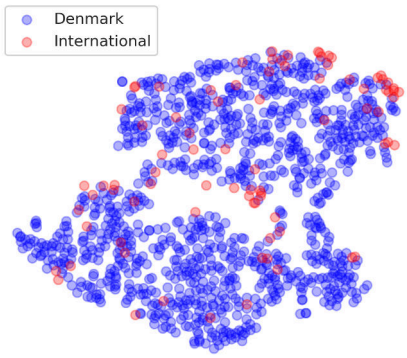


(e) Latitude and Longitude

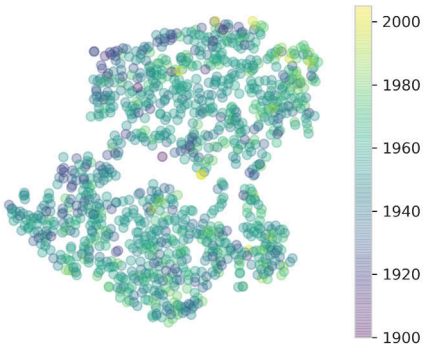
Fig. F.14.: Combined GiMMPPy EEK-means: 2D t-SNE visualizations of samples from posterior of dataset in latent space.



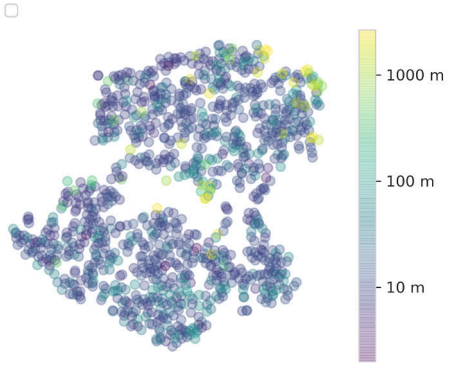
(a) Gender



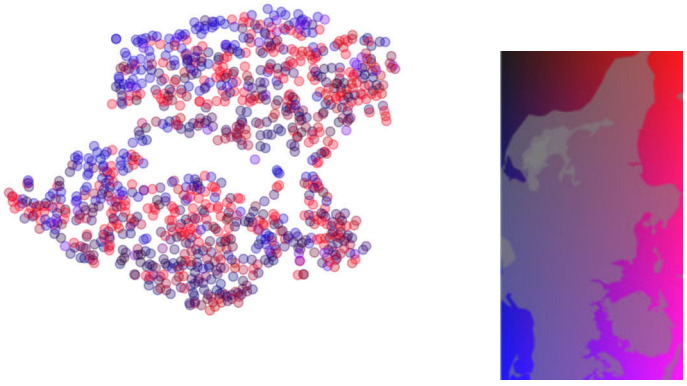
(b) Country



(c) Year Collected



(d) Altitude



(e) Latitude and Longitude

Fig. F.15.: Combined GiMMPPy EEK-means Means: 2D t-SNE visualizations of means of posterior of dataset in latent space.

G

Reconstructions of different groupings from different models

In this appendix we show the reconstructions of the 'average' image in latent space for each model, excluding groupings with four or fewer examples.

G.1 Gender

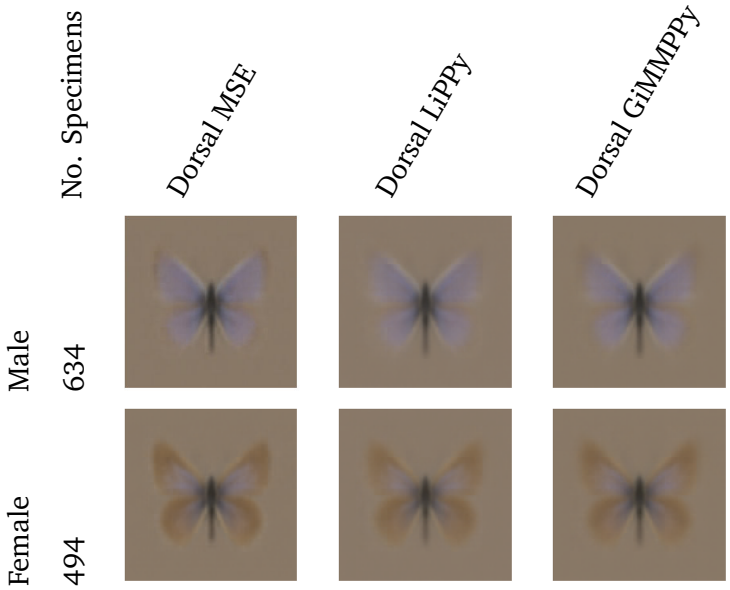


Fig. G.1.: Dorsal-input, Gender: Reconstructions made using average latent space per gender from models with dorsal-input.

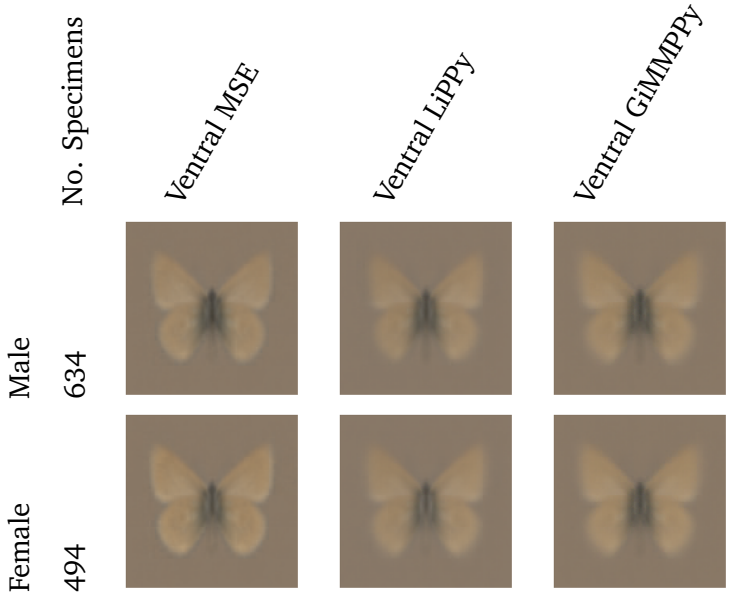


Fig. G.2.: Ventral-input, Gender: Reconstructions made using average latent space per gender from models with ventral-input.

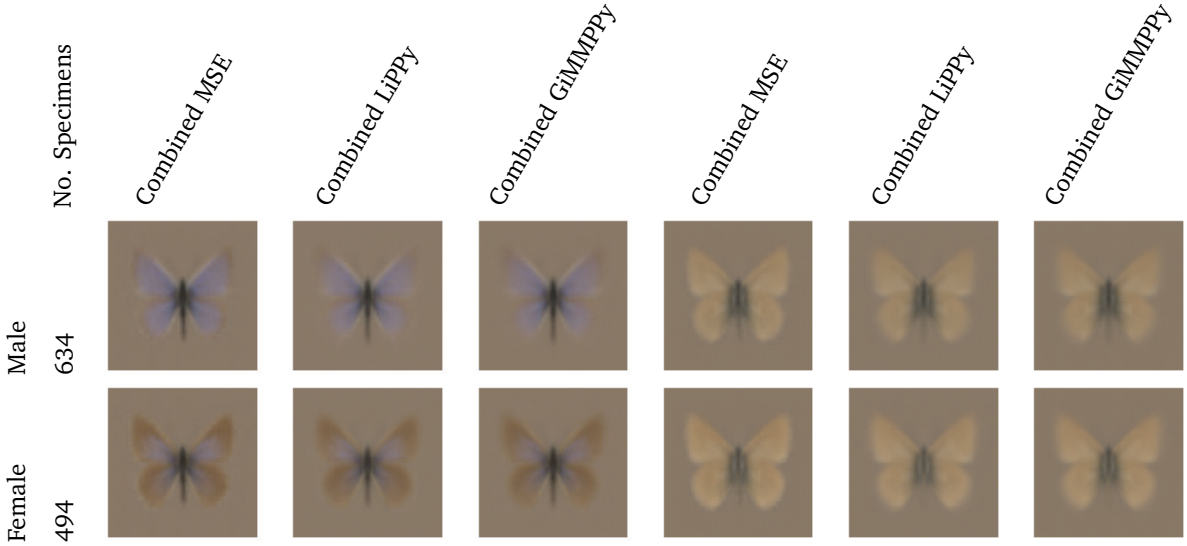


Fig. G.3.: Combined-input, Gender: Reconstructions made using average latent space per gender from models with combined-input.

G.2 Kaaber Region

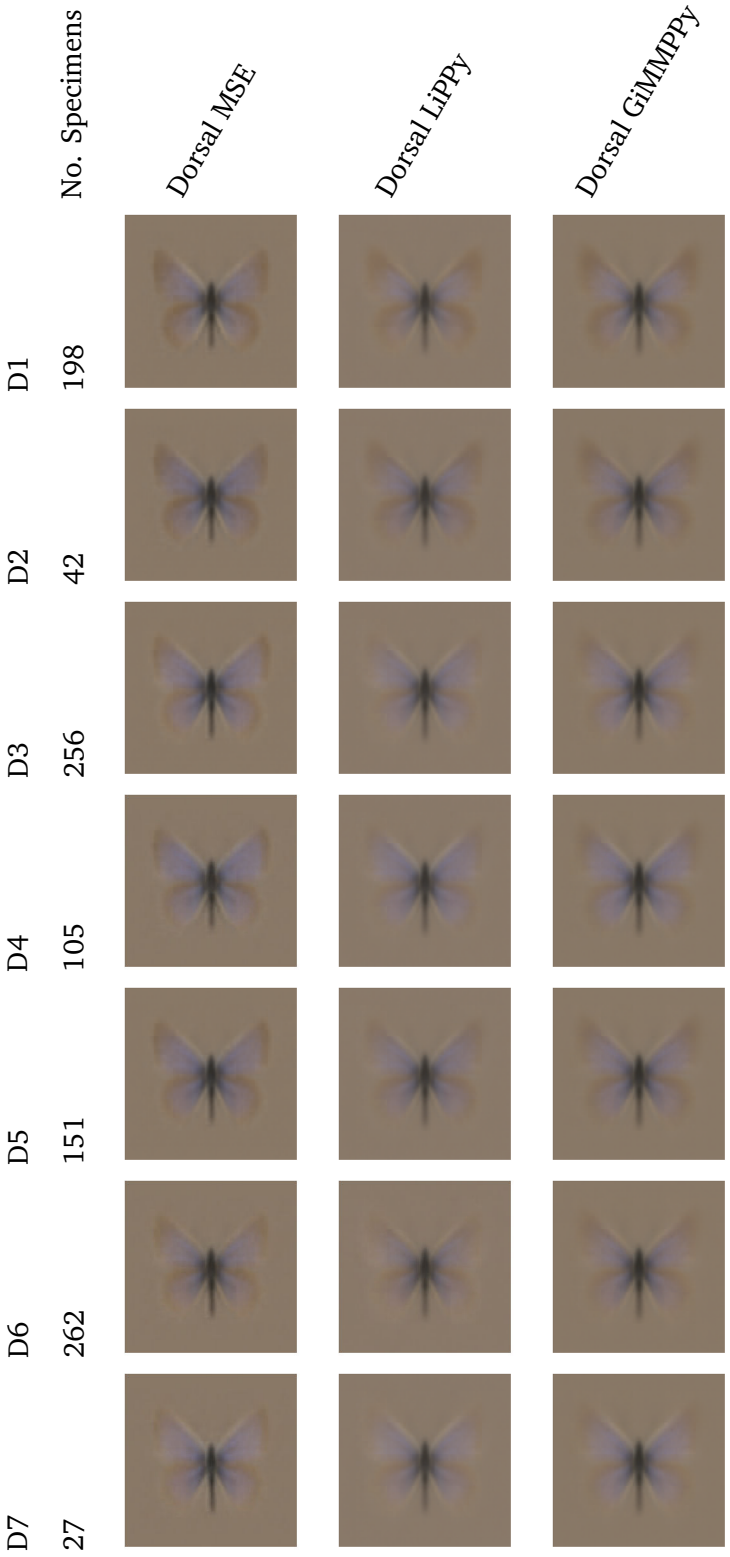


Fig. G.4.: Dorsal-input, Kaaber Region: Reconstructions made using average latent space per Kaaber Region from models with dorsal-input.

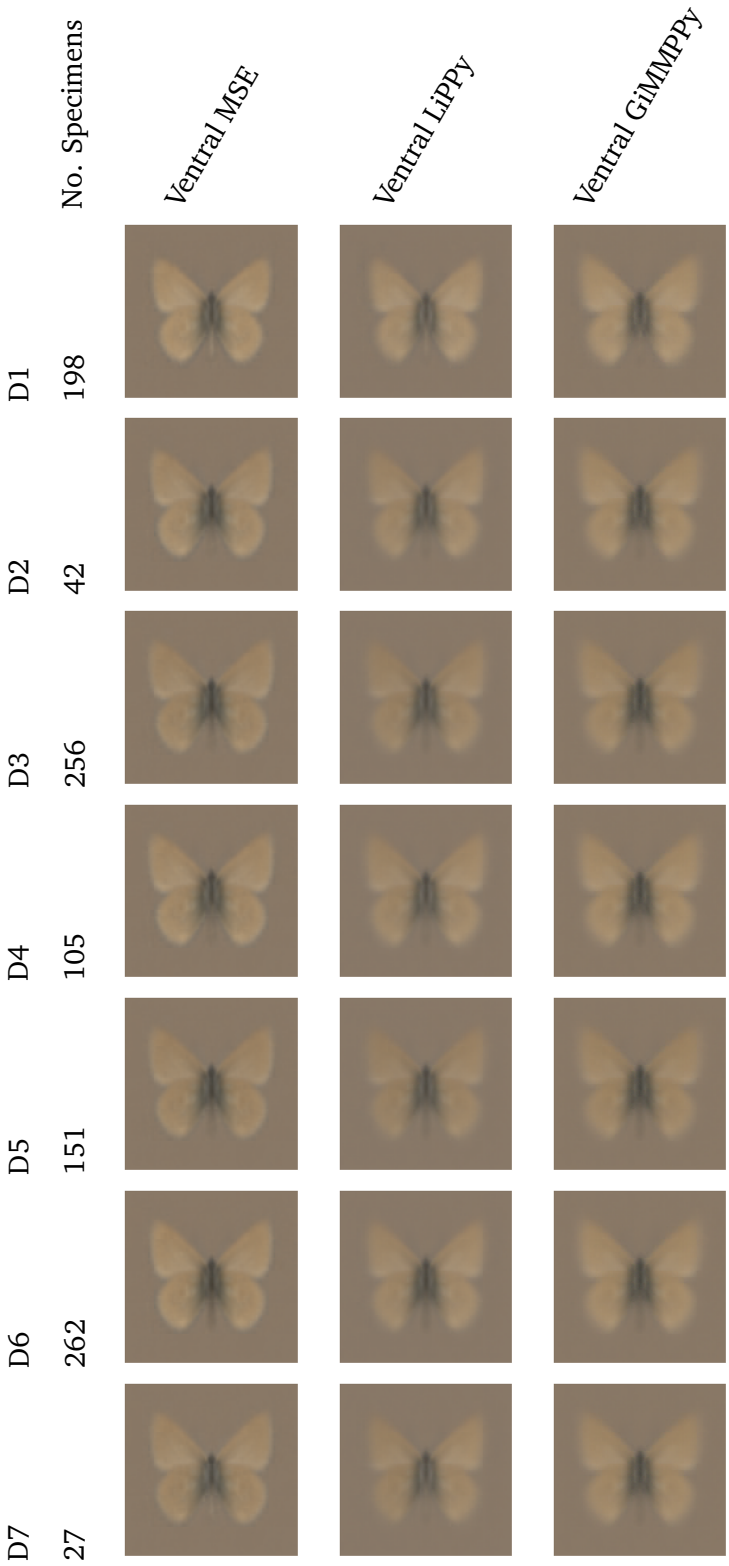


Fig. G.5.: Ventral-input, Kaaber Region: Reconstructions made using average latent space per Kaaber Region from models with ventral-input.

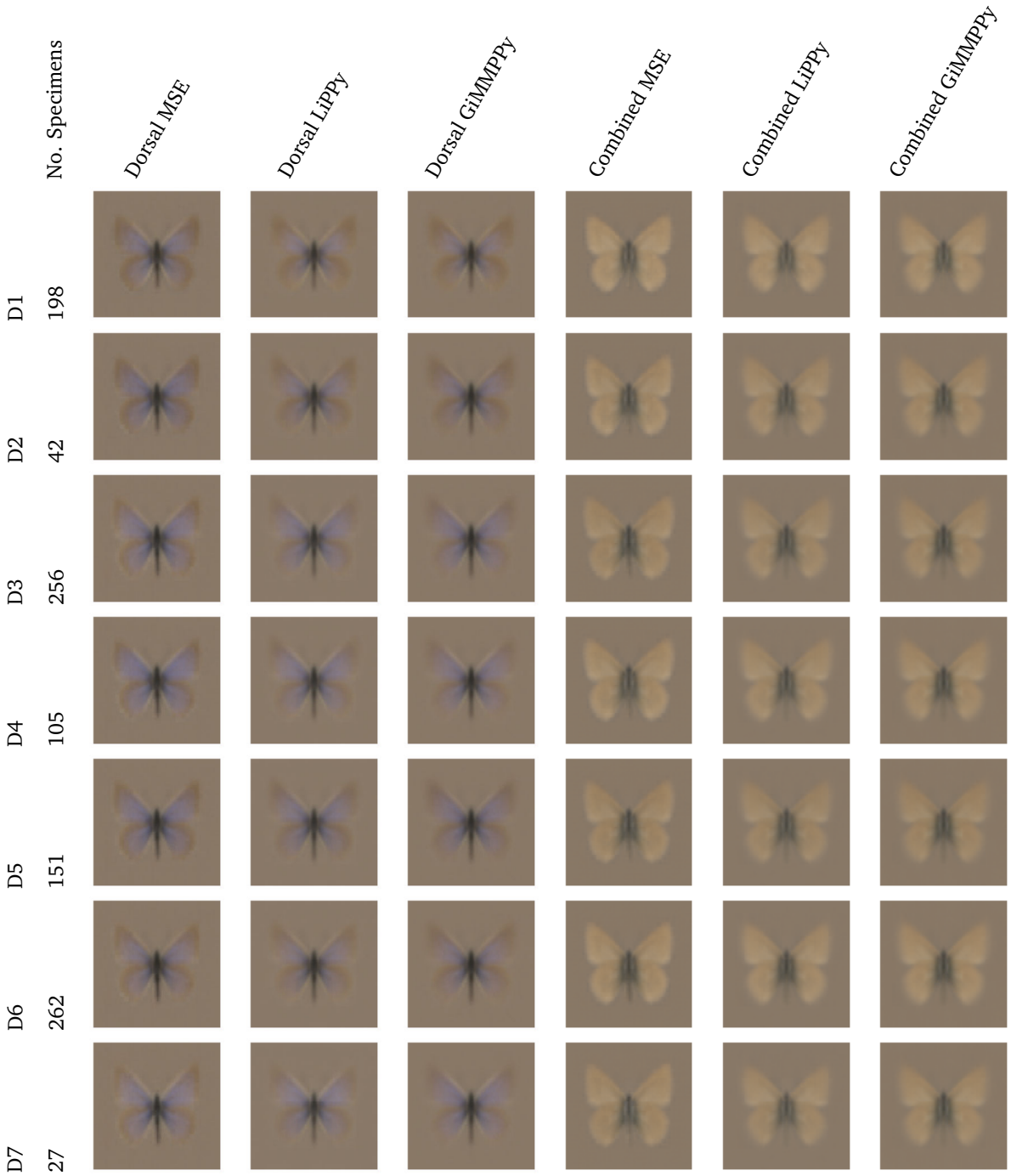


Fig. G.6.: Combined-input, Kaaber Region: Reconstructions made using average latent space per Kaaber Region from models with combined-input.

G.3 Country

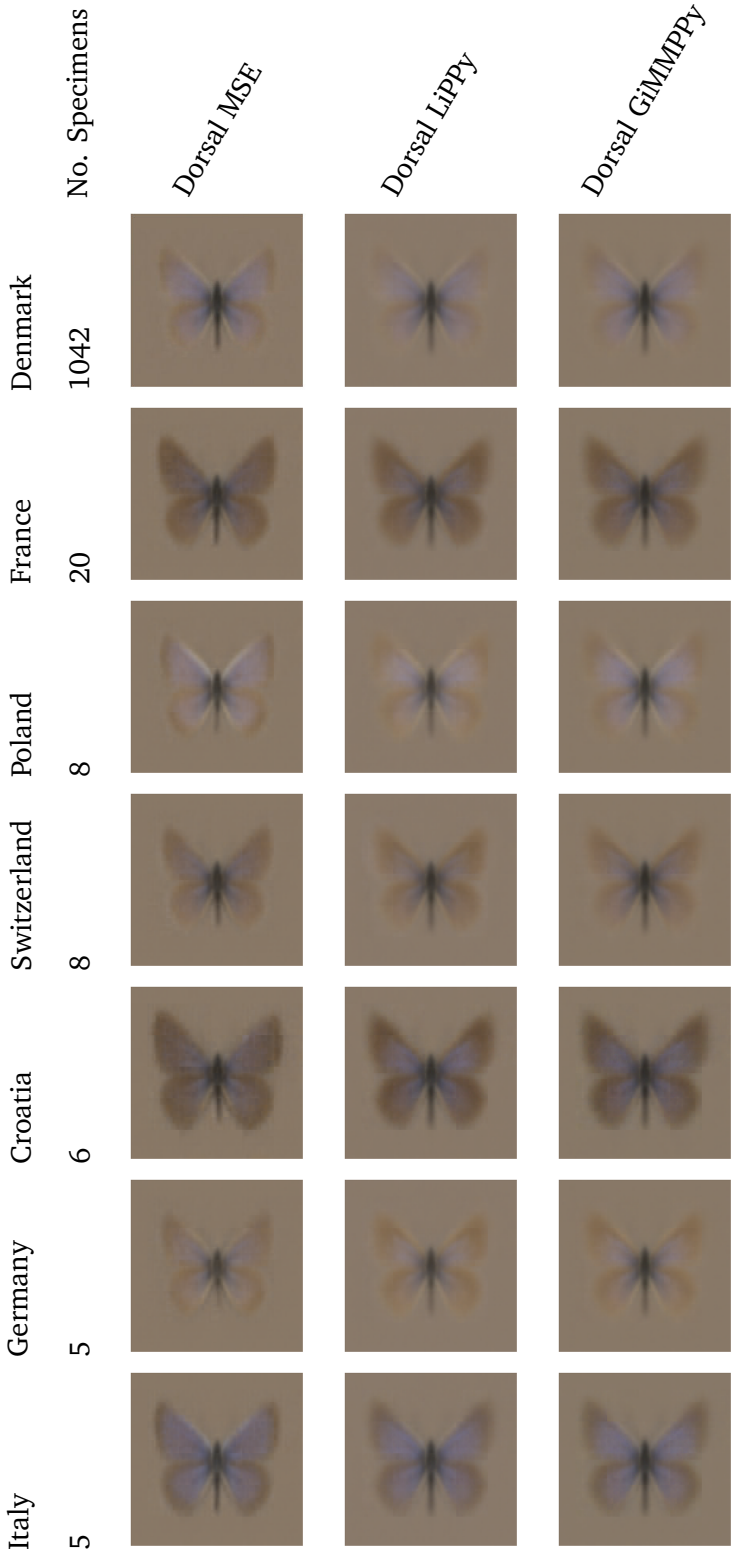


Fig. G.7.: Dorsal-input, Country: Reconstructions made using average latent space per country from models with dorsal-input.

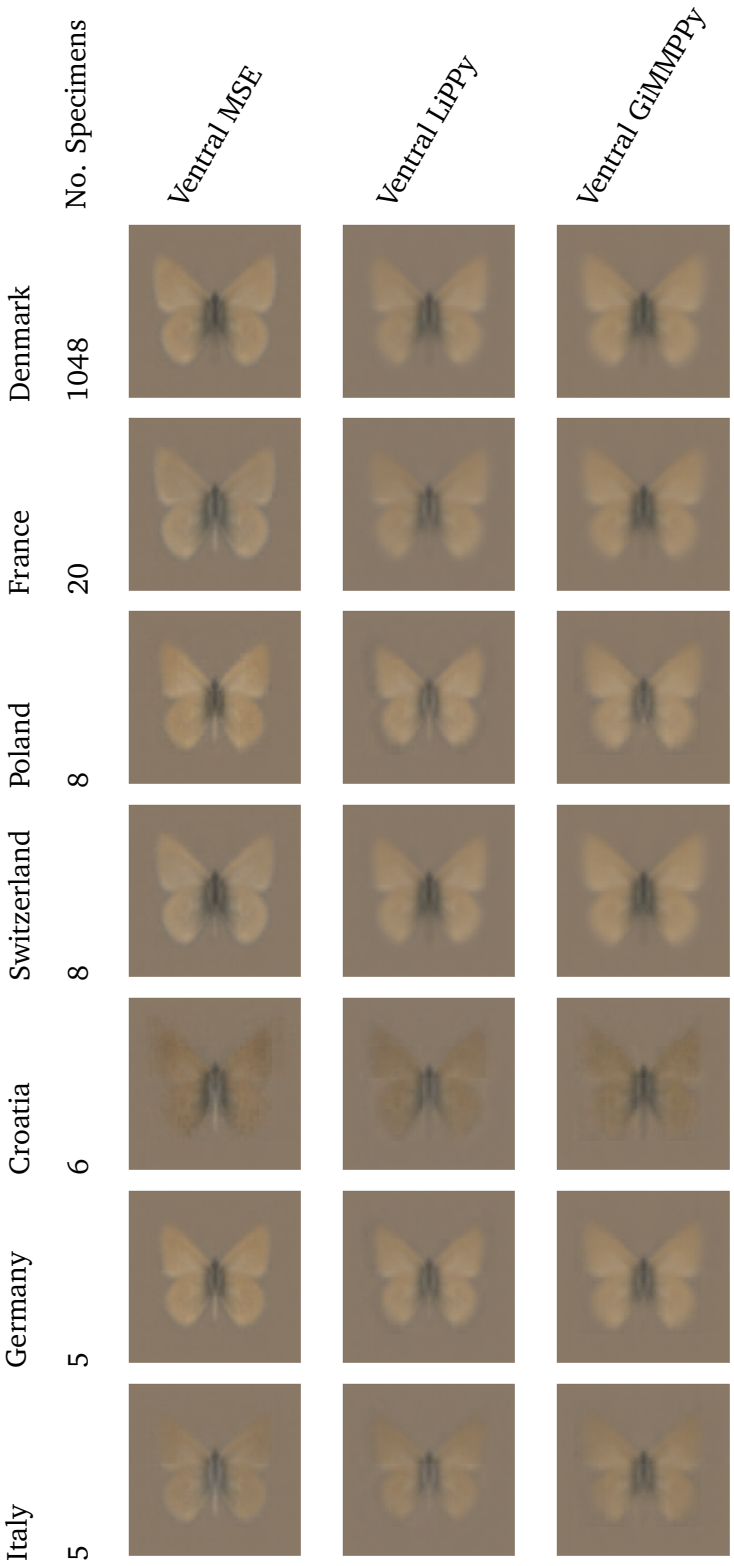


Fig. G.8.: Ventral-input, Country: Reconstructions made using average latent space per country from models with ventral-input.

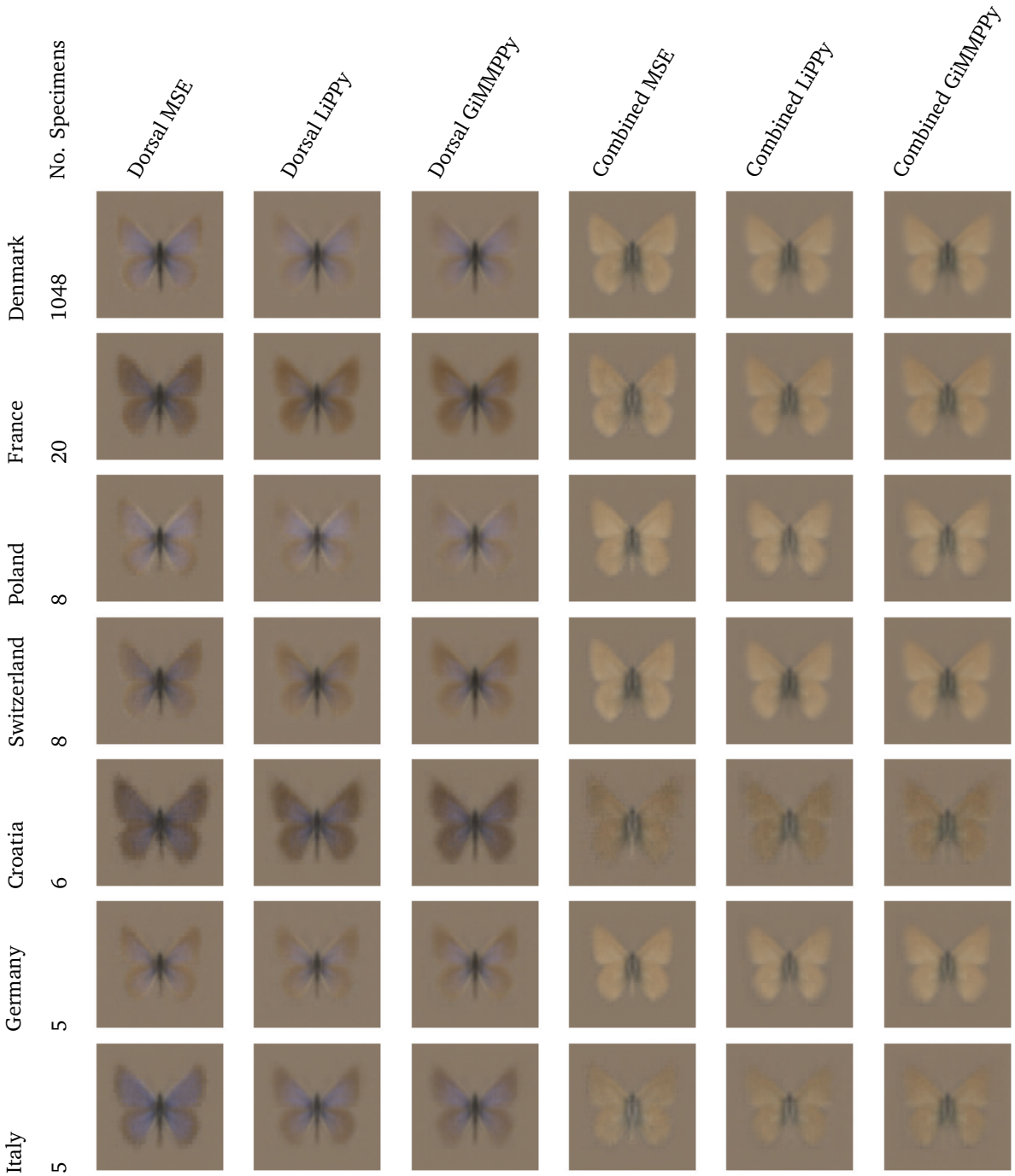


Fig. G.9.: Combined-input, Country: Reconstructions made using average latent space per country from models with combined-input.

G.4 Decade

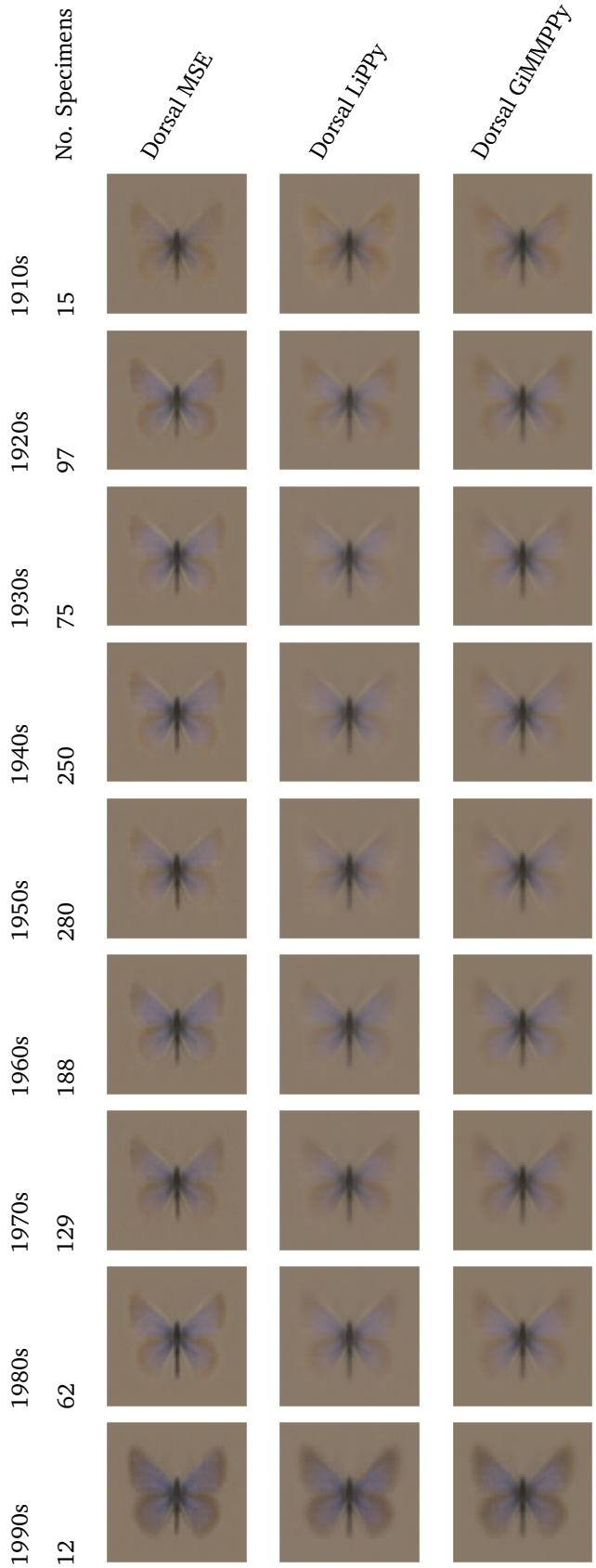


Fig. G.10.: Dorsal-input, Decade: Reconstructions made using average latent space per decade from models with dorsal-input.

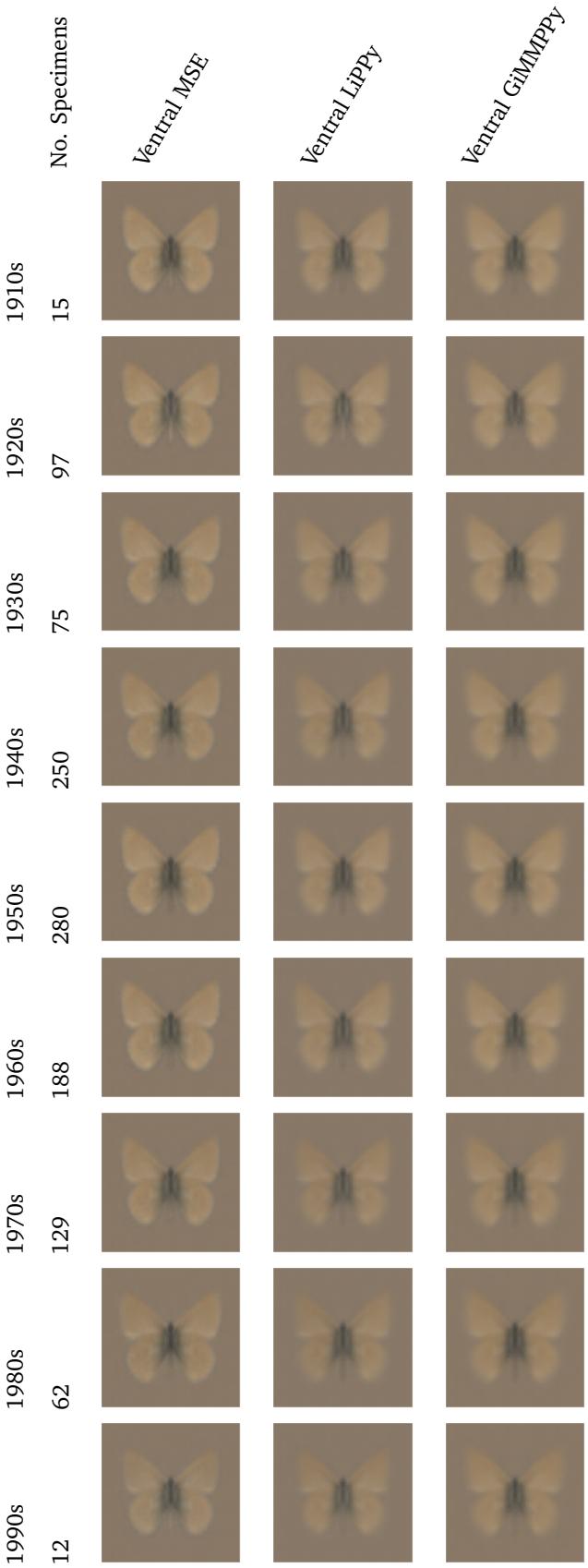


Fig. G.11.: **Ventral-input, Decade:** Reconstructions made using average latent space per decade from models with ventral-input.

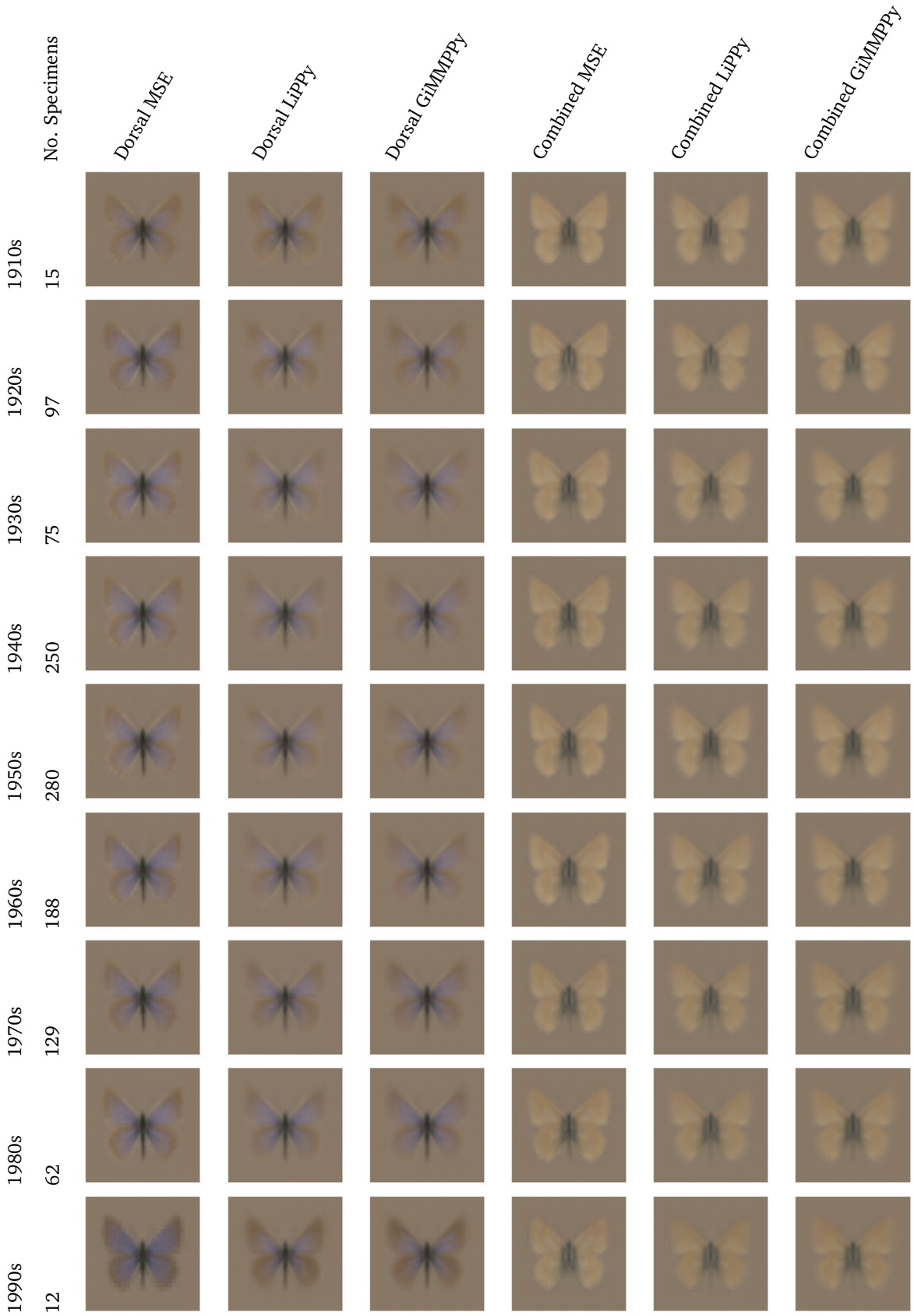


Fig. G.12.: Combined-input, Decade: Reconstructions made using average latent space per decade from models with combined-input.

G.5 Altitude

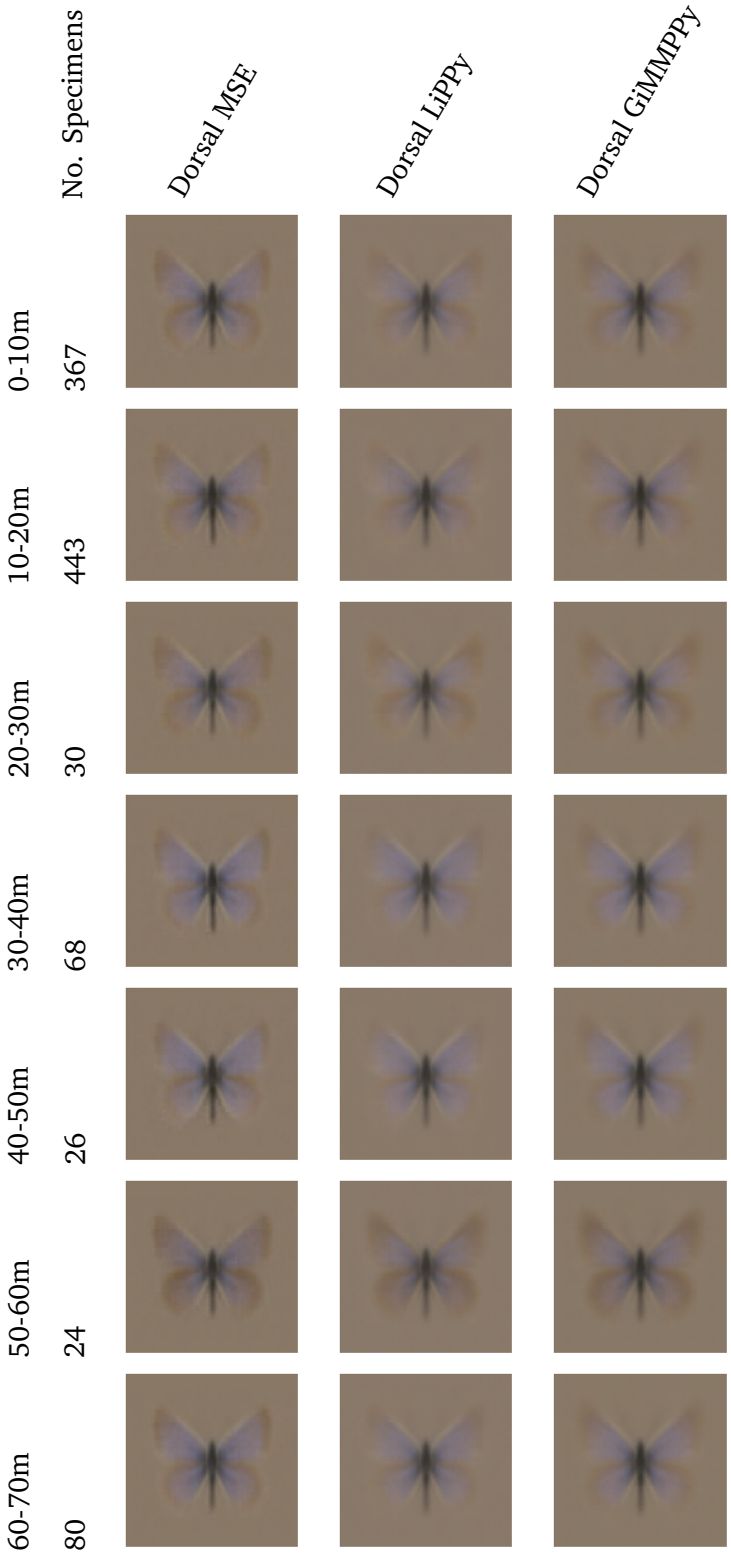


Fig. G.13.: Dorsal-input, Altitude: Reconstructions made using average latent space per altitude from models with dorsal-input.

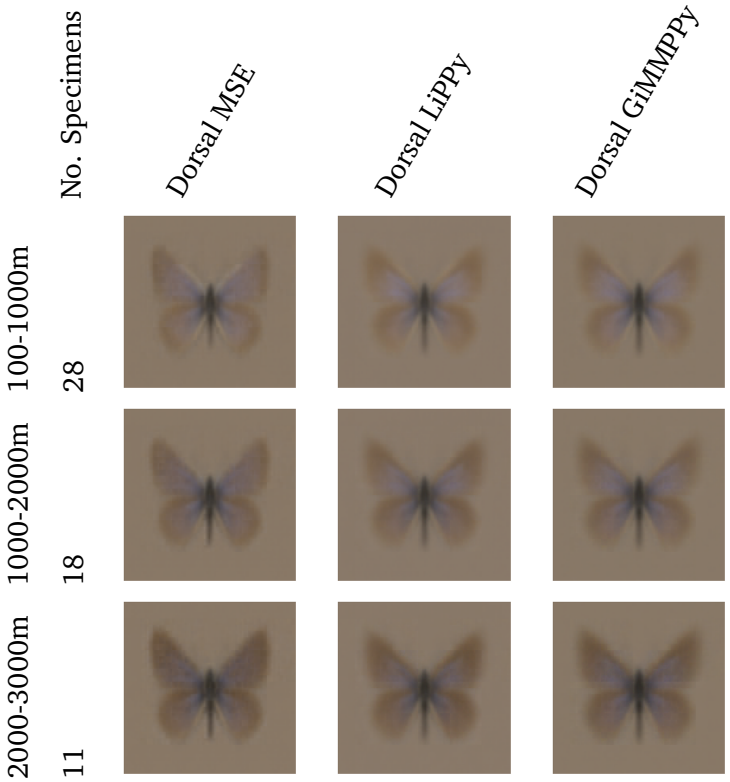


Fig. G.14.: Dorsal-input, Altitude: Reconstructions made using average latent space per altitude from models with dorsal-input.

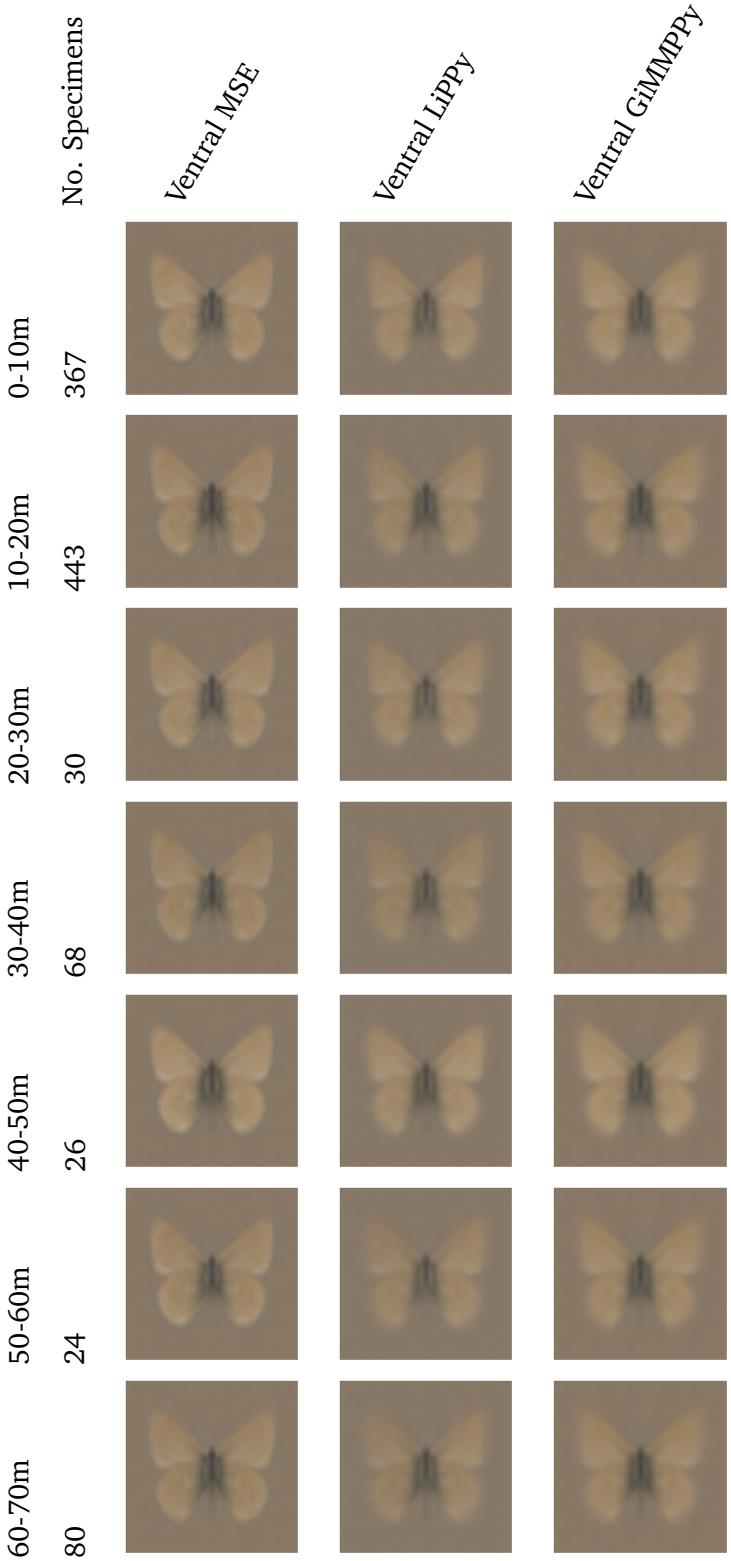


Fig. G.15.: Ventral-input, Altitude: Reconstructions made using average latent space per altitude from models with ventral-input.

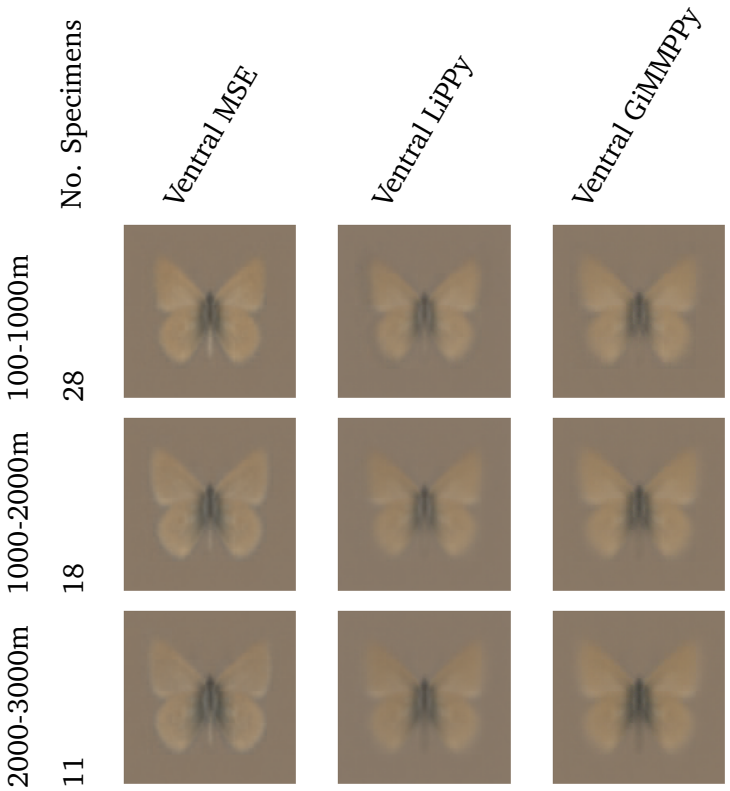


Fig. G.16.: Ventral-input, Altitude: Reconstructions made using average latent space per altitude from models with ventral-input.

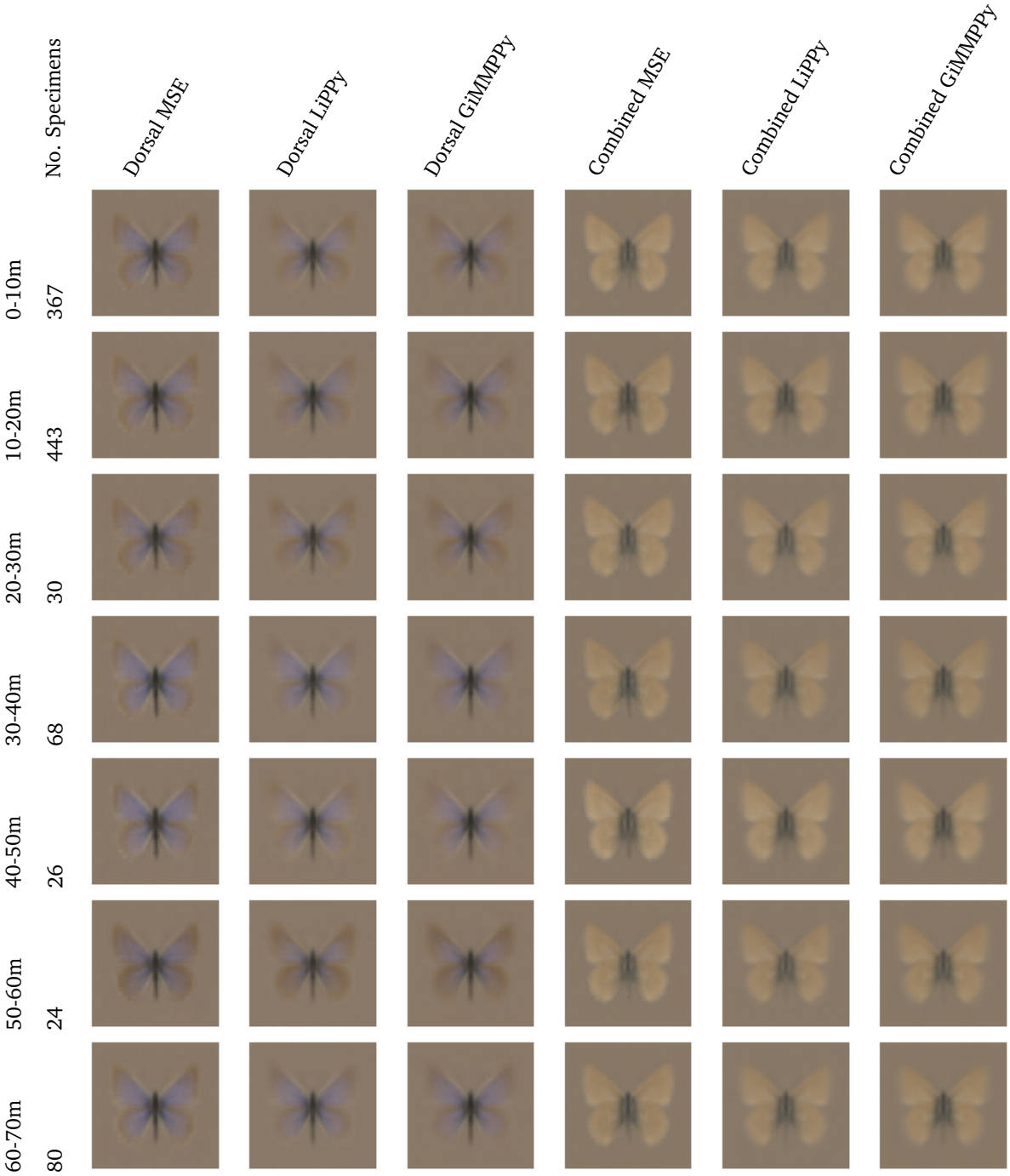


Fig. G.17.: Combined-input, Altitude: Reconstructions made using average latent space per altitude from models with combined-input.

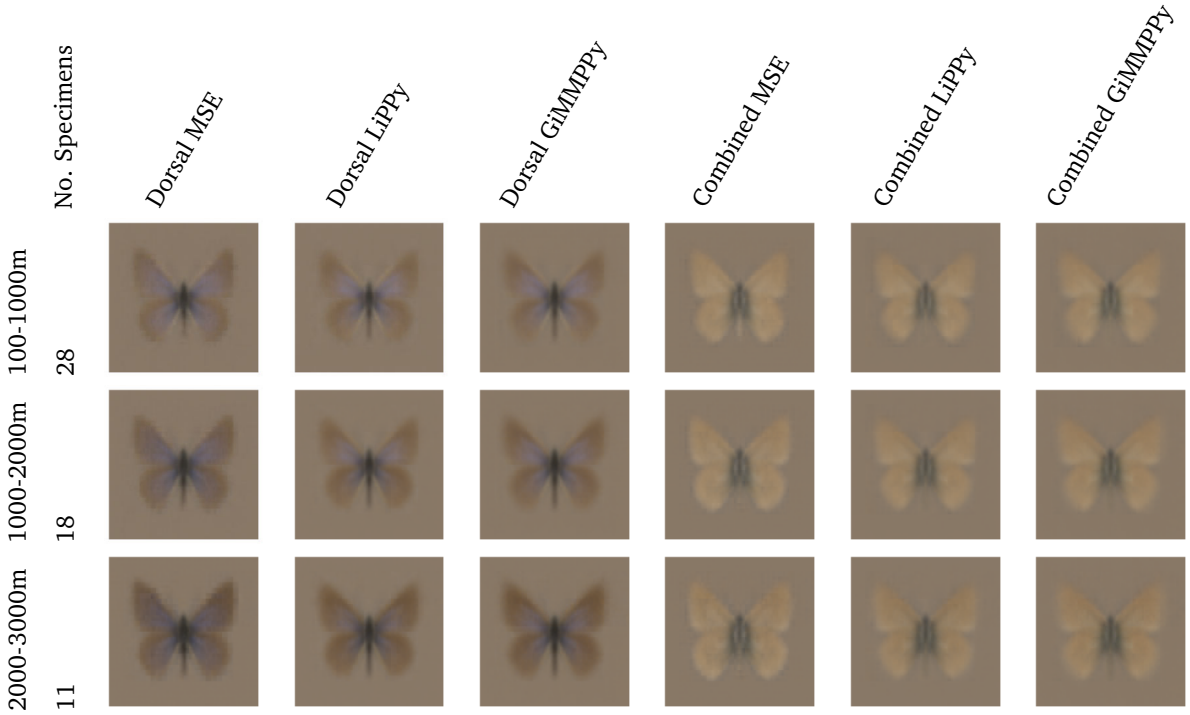


Fig. G.18.: Combined-input, Altitude: Reconstructions made using average latent space per altitude from models with combined-input.