



UiT The Arctic University of Norway

Faculty of Science and Technology
Department of Physics and Technology

Leveraging Explainability Maps for Group-unsupervised Robustness to Spurious Correlations

Adrian Henrik de Sena Sletten

FYS-3941 Master's thesis in applied physics and mathematics 30 SP - August 2023

“Never attribute to high-level abilities that which can be adequately explained
by shortcut learning”
–Geirhos et al. [35]

Abstract

Shortcut learning, the tendency for models to rely on spurious correlations, is a widespread issue in deep learning. Although being a known issue, uncovering the shortcuts present in a dataset can be a difficult task. Over the last few years, explainability methods have been leveraged to find previously unknown shortcuts within mainstream datasets. However, how to best mitigate a model's reliance on shortcuts, is not a matter that is widely agreed on.

Recently, the concept of group robustness has appeared as a potential way for mitigating shortcuts. In group robustness, the data in each class is divided into subclasses where some contain the shortcut features while some other subclass does not. By optimising a model to increase the worst group performance, the model learns to perform well across groups, mitigating reliance on shortcuts.

One notable limitation that exists for current group robustness methods is their reliance on group labels to guarantee performance improvements. The issue with this is that acquiring these additional labels is a difficult and time-consuming task. Therefore we propose, eXplainability-based Feature Reweighting (XFR), a group-unsupervised group robustness method. Our proposed method leverages the clustering of explainability heatmaps to estimate pseudo-labels for groups in a dataset and afterwards uses these labels to improve group robustness.

In our results, we show that XFR clearly improves group robustness compared to standardly trained models (ERM). We also show that performance is on par with, and sometimes even surpasses, methods that partially or fully utilise group labels.

Acknowledgements

First of all, I would like to thank my main supervisor, Michael Kampffmeyer, and my co-supervisor, Riddhi Chakraborty, for their guidance, support and continuous encouragement throughout the journey of this master's thesis. Their expertise and feedback have been instrumental in shaping the direction of this work. I am especially grateful to my co-supervisor for engaging in thought-provoking and enriching discussions during our lunch breaks.

I would also like to thank my fellow students whose dedication and work ethic has inspired me to work harder. You helped make the countless hours we spent at the office not just bearable, but also fun. Thank you.

Next, I would like to extend my gratitude to my family and friends for their continuous support and presence throughout my studies.

Lastly, I would like to express my heartfelt thanks to Hana, for her unwavering belief in me and her endless encouragement. Her continuous support has been a driving force behind my accomplishments, and I am truly fortunate to have her as a source of strength.

Adrian Henrik de Sena Sletten,
Tromsø, August 2023.

Contents

Abstract	iii
Acknowledgements	v
List of Figures	xi
List of Tables	xv
List of Abbreviations	xvii
1 Introduction	1
1.1 Deep Learning Today	1
1.2 Shortcut learning: An issue	2
1.3 Group Robustness: A possible solution	3
1.3.1 Limitations with current group robustness methods	4
1.4 Contributions	4
1.5 Outline of thesis	5
I Background	7
2 Deep Learning	9
2.1 Overview of deep learning	9
2.1.1 Deep learning approaches	10
2.1.2 Main building blocks	11
2.2 Multilayer Perceptron	12
2.2.1 Perceptron	12
2.2.2 From one to many layers	14
2.2.3 Forward pass	15
2.2.4 Activation functions	16
2.2.5 Loss functions	19
2.2.6 Backward pass	20
2.2.7 Gradient descent variants and optimisation algorithms	22
2.2.8 General training procedure	23

2.3	Convolutional neural networks	25
2.3.1	Convolution operation	25
2.3.2	The convolutional layer	27
2.3.3	Pooling	28
2.3.4	Example of simple CNN architecture	29
3	Explainability in Deep Learning	31
3.1	Deep learning models are blackboxes	31
3.2	XAI	32
3.2.1	Transparent methods	32
3.2.2	Post-hoc methods	33
3.3	Layer-wise Relevance Propagation	33
3.3.1	Propagation mechanisms	34
3.3.2	Different propagation rules	35
3.3.3	Best practice	35
4	Clustering	37
4.1	Uncovering underlying structures in the data	37
4.2	Clustering algorithms	38
4.2.1	Hierarchical clustering	39
4.2.2	K-means	39
4.2.3	Deep clustering	40
4.2.4	Spectral clustering	40
4.2.5	Deciding the number of clusters	42
5	Shortcuts in deep learning	43
5.1	What are shortcuts?	43
5.2	A simple example	44
5.3	Shortcuts are present in all learning systems	45
5.4	The origin of shortcuts	46
5.4.1	Ambiguity and lack of constraints	46
5.4.2	Biases	46
5.5	Shortcuts in computer vision tasks	47
5.5.1	Artefacts	47
5.5.2	Context and background reliance	48
5.5.3	Texture-shape cue conflicts	48
5.6	Mitigation of shortcuts	48
5.6.1	Group robustness	50
II	Method	53
6	Proposed method	55
6.1	Explainability-based feature reweighting (XFR)	55

6.1.1	Unsupervised Group Discovery	56
6.1.2	Group Balanced Training	56
6.1.3	Variants	57
6.2	Building blocks of XFR for experiments	58
6.2.1	Spectral Relevance Analysis	59
6.2.2	Deep feature reweighting	59
6.2.3	XFR setup	60
III	Experiments	61
7	Experimental setup	63
7.1	Group robustness datasets	63
7.1.1	Colored-MNIST	64
7.1.2	Waterbirds	66
7.1.3	CelebA	66
7.2	Metrics	67
7.3	Models	67
7.3.1	External Methods	67
7.3.2	Implemented Methods	68
8	Main experiment and analysis	71
8.1	Results	71
8.1.1	Explainability improvements from XFR	73
8.2	Exploring the group estimation method	74
9	Ablation studies	83
9.1	Changing the downsampling size	83
9.1.1	Effect of eigengap heuristic on L-XFR performance	84
9.2	The effect of eigengap on the main results	85
9.3	Using 1 cluster	86
9.4	Retraining on the training set instead of the validation set	87
9.5	Removing minority groups	88
9.6	Merging of clusters	88
IV	Conclusion and future work	91
10	Conclusion	93
11	Future work	95
	References	97

List of Figures

1.1	Example from [12] showing how background and context are used as a shortcut feature to classify the class "cow". (A) showcases an image of a cow in a normal context, while (B) and (C) showcase images of cows in unusual contexts. The top-5 predictions for each image are presented, and only in (A) is "cow" predicted.	3
2.1	Visualisation of the perceptron. Inputs are multiplied by weights and added up to create output.	13
2.2	The forward pass. Illustration of the feed-forward process for layer r of an MLP.	15
2.3	The sigmoid function and its derivative.	18
2.4	2D convolution. Can be interpreted as sliding the kernel over all the possible positions of the image.	26
2.5	2D pooling. Pooling performed with window size 2x2 and stride 2.	28
2.6	Simple deep CNN example: the standard VGG-16 network architecture [108].	29
3.1	Example of an LRP heatmap from a VGG-16 model showcasing the parts of the input that are relevant for the prediction "golden retriever". Red and blue represent positive and negative relevance respectively. Here we have utilised the following propagation rules: LRP- $\alpha\beta$ with $\alpha = 2$ for the convolutional layers, and LRP- ϵ for the fully-connected layers.	34
4.1	Overview of clustering methods described. The visualisation is based on [51].	38
4.2	Here (a) shows an instance of a Blob dataset with 5 blobs, and (b) shows the 10 smallest eigenvalues of the Laplacian matrix. There is a gap in the eigenvalues after the fourth one, which suggests 4 clusters. However, note that the largest eigengap is after the fifth eigenvalue. This reflects our prior knowledge of the data that there are 5 blobs.	42

5.1	Simple shortcut learning example. Classification task of cats and dogs with the provided training set and labels (black font). For the test sets blue font indicates the label a human would give, while red font is the label given by a model trained on the given training set. The i.i.d test set contains similar examples to the training set, while the o.o.d test set contains different examples. Testing on different data shows that the model uses a different decision rule than expected.	44
5.2	Shortcut learning example from [65]. Shows that the model has learned to rely on the presence of a watermark to classify an image as class 'horse'.	48
5.3	Example of background reliance in caption generation task. The presence of a green hillside is spuriously correlated with the class sheep.[104]	49
5.4	Model predictions for 3 types of images. (a) shows only texture, (b) shows only the content, and (c) is a combination of the two images. Note how the model still classifies (c) by texture.[36]	49
8.1	The first 12 eigenvalues of the Laplacian matrix when performing the 3 runs of spectral clustering needed to perform L-XFR (y_0 and y_1) and G-XFR (all). The eigengap heuristic uses the biggest gap among the 10 smallest eigenvalues as an estimate for the best number of clusters to use.	76
8.2	The average relevance for each cluster of $y = 0$	78
8.3	The average relevance for each cluster of $y = 1$	78
8.4	The average relevance for each cluster of all observations.	78
8.5	Similarities using Pearson's correlation coefficient (see Equation 4.1), between the average heatmaps from each cluster in the 3 clustering runs of CelebA. A reminder that -1 or 1 indicates a complete negative or positive correlation, while 0 indicates no correlation. Values close to 0 thereby indicate little similarity.	80
8.6	Similarities using Pearson's correlation coefficient (see Equation 4.1), between the average heatmaps from each cluster in the 3 clustering runs of C-MNIST. A reminder that -1 or 1 indicates a complete negative or positive correlation, while 0 indicates no correlation. Values close to 0 thereby indicate little similarity.	80
8.7	t-SNE plot of CelebA heatmaps in 2D coloured with ground truth groups. It is impossible to recover the ground truth labels here.	81

- 8.8 t-SNE plot of C-MNIST heatmaps projected down to 2D. G-XFR is able to capture the ground truth minority groups in both cases - the reason for its good performance in Table 8.1. Colours are consistent in the columns, meaning that pink (and also grey) was found as part of the same cluster. . . . 81

List of Tables

7.1	Data splits in the Colored-MNIST dataset.	64
7.2	Data splits in the Waterbirds dataset.	64
7.3	Data splits in the CelebA dataset.	65
8.1	Worst group and mean accuracy on the test sets of the different datasets. The Group Info column showcases for each method whether group labels are used for that split of the data (\times = does not use group labels, \checkmark = uses group labels, $\checkmark\checkmark$ = validation set group labels is used for training and finetuning, and $\times\times$ = validation set without group labels is used for training and finetuning). The results for JTT, LfF, and Group DRO were gathered from [71], GEORGE results are from [111], while the ones below the separation line are our own results. For our results that use last layer retraining we report mean \pm std over 5 runs after selecting the hyperparameter.	72
8.2	Examples showcasing the improvements in the explainabilities. Images on the left are the original images, in the middle are the LRP heatmaps for ERM, and on the right are the LRP heatmaps for G-XFR. All images showcase "waterbirds on land background" that were misclassified by ERM, but classified correctly by G-XFR.	73
8.3	LRP example from the different datasets. The 1st column showcases the original image, the 2nd column showcases the aggregated LRP heatmaps, and the remaining 3 columns showcase each of the RGB components of the heatmap. This example clearly illustrates the additional information gained by utilising the RGB heatmap.	75
8.4	Image and heatmap examples from largest clusters. For these images, the explanations are focused on the core features needed to perform the classification task.	79

8.5	Image and heatmap examples from the smaller clusters. The explanations for these images all focus on the top-left corner of the image. The potential shortcut feature that the model is utilising in the corner could be the presence of a single colour there (in most cases the colour is black). The single colour might come from segmentation of the original image, or from cropping or extending the original image.	79
9.1	Results for G-XFR and L-XFR for different downsizing of the heatmaps compared to ERM and DFR baselines. Best results across the resolutions are bolded.	84
9.2	Comparing the performances when using a fixed number of clusters vs. the eigengap heuristic that gave 2 clusters for $y = 0$, and 9 clusters for $y = 1$. Here we have presented the weighted (to the group distribution of the training set) mean instead of the normal mean over the test set.	85
9.3	Eigengap heuristic C-MNIST: (6,2), 8. Eigengap heuristic for Waterbird: (3,2), 3	86
9.4	Here we compare "DFR (class)" that uses the classes directly as groups, with other methods.	87
9.5	Comparing the difference from using the validation vs the training set to improve group robustness.	87
9.6	Comparison of main results with majority cluster results. Here we perform retraining only utilising the largest group for each class. Bolded values are the overall best results.	88
9.7	Comparison of main results with supercluster results	89

List of Abbreviations

AI artificial intelligence

CNN convolutional neural network

DFR deep feature reweighting

DL deep learning

DNNS deep neural networks

ERM empirical risk minimisation

G-XFR global-XFR

GEORGE

group DRO group distributionally robust optimisation

JTT Just Train Twice

L-XFR local-XFR

LfF Learn from Failure

LRP layer-wise relevance propagation

ML machine learning

MLP multilayer perceptron

SGD stochastic gradient descent

SPRAY Spectral Relevance Analysis

t-SNE t-Stochastic Neighbourhood Embedding

WGA worst group accuracy

XAI eXplainable AI

XFR Explainability-based feature reweighting



Introduction

1.1 Deep Learning Today

Deep learning (DL) is a subfield of machine learning (ML) and artificial intelligence (AI), that has had a large surge in popularity lately. In DL, models are designed and trained to tap into the wealth of information present in large datasets. Through training, these models unveil and learn hidden patterns and insights that can be leveraged to perform a variety of different tasks in both ordinary and critical domains. As such, DL has played and will continue to play, a role in reshaping the landscape of technological possibilities and applications.

Today, some ordinary everyday applications of DL are image and video filters, speech recognition [98], automatic caption generation [10], machine translation [91], personalised advertising [126], music recommendation [126], chatbots [89], bots for playing games like chess [107], battery management of your phone, weather prediction [101], etc. Generative technologies enable the generation of images [79], sounds, videos [109], and text. These applications enhance convenience, entertainment, and communication in daily life without posing significant safety risks or high-stakes consequences.

Presently DL is also used in safety-critical, high-stakes, and socially-impactful domains such as medical diagnostics [13], autonomous driving [50], energy management [85], environmental monitoring [124] and conservation, eligibility for loans [41], fraud detection [75], automatic monitoring and removal of

harmful content [26], hiring [23], and much more. These applications of DL play a crucial role in safeguarding lives, improving public safety, mitigating global challenges, and addressing critical societal issues, where errors could lead to substantial consequences on human well-being, the environment, and the functioning of crucial systems.

As DL is utilised in more and more critical domains, it is imperative that the models used can be trusted and relied on. Trust and reliability can be increased by utilising models that perform well, and that are robust to differences from its testing ground to the real world. A different approach for increasing trust is to provide explanations for why a DL model made a certain decision. In this way, wrong decisions may be pinpointed and dealt with. However, although explanations are useful they are usually not provided since DL models are considered to be black box models - we have no knowledge of how its decisions are made [83]. Considerable work [38] has been done over the last years to create good and interpretable explanation methods for deep learning models.

1.2 Shortcut learning: An issue

A clear issue standing in the way of reliable and trustworthy deep learning models is the concept of shortcut learning [35]. Shortcut learning describes a model's tendency of learning spurious correlations or superficial cues to solve a problem in an unintended way. While resulting in models with high test accuracies, these models fail in real-world applications where the shortcut features may not be present.

In Figure 1.1, we showcase a simple example of shortcut learning, where we see a classification model utilising the context or background in its decision-making. The model is given images of cows in different contexts and reports the top-5 most likely predictions, and we observe that the model only makes the correct prediction in the normal context. From the dataset the model was trained on, it learned that the detection of "grass" or "field" is a viable strategy for achieving high accuracy in predicting the class "cow". Strategies like this often go unnoticed if models are not extensively tested.

This example highlights the difficulty of dealing with shortcut learning. Supervised deep learning methods are trained to learn underlying patterns in the dataset by simply using the information provided by a class label. The fact that models are not taught what to focus on when learning from the data opens the possibility for shortcut learning opportunities, such as the presence of grass for classifying cows.

But to deal with this issue is not as simple as just improving our datasets. The



Figure 1.1: Example from [12] showing how background and context are used as a shortcut feature to classify the class "cow". (A) showcases an image of a cow in a normal context, while (B) and (C) showcase images of cows in unusual contexts. The top-5 predictions for each image are presented, and only in (A) is "cow" predicted.

grass shortcut, for instance, emerged because probably a majority of training images for the model featured cows in grassy environments — precisely where cows are predominantly found. To construct more accurate datasets, we would need to break free from the reliance on the existing distribution of images and gather examples that showcase the varieties that exist in the real world. However despite our efforts, there is no guarantee that in our attempt to improve datasets we will not inadvertently introduce some other shortcut opportunity that can be exploited.

Despite its challenges, finding ways of overcoming shortcut learning is an important step towards robust, trustworthy and reliable DL models, as it directly impacts their ability to generalize across diverse situations.

1.3 Group Robustness: A possible solution

Group robustness is a recent approach aimed at mitigating shortcut learning, and it aims to balance a model's performance across groups in the data to remove the reliance on the shortcut features. This can e.g. be done by dividing each class into 2 groups where one group contains the spurious correlations, while the other does not. In the cow example, we would have a group that contains cow images featuring grass, and another group with cow images and no grass. By balancing the performance across the groups, i.e. making the model do well in both groups, we would ideally guide the model to learn the core feature and not utilise the spurious attribute.

The key metrics used to quantify group robustness are the mean accuracy over all observations and the Worst group accuracy (WGA). Utilising the WGA is useful as it captures the performance difference that is often present between groups, while the mean accuracy tells us how effective the model is in general. For example, traditional models, trained using empirical risk minimization ERM often yield high mean accuracy, but low WGA. The goal of group robustness methods is to close the gap between WGA and the mean accuracy while trying to keep the mean accuracy as close as possible to that of the ERM model. In other words, we wish to improve robustness while giving up as little average performance as possible.

1.3.1 Limitations with current group robustness methods

Several group robustness methods have been proposed [28], [58], [71], [86], [87], [92], [99] and all have shown improvements in WGA. However, despite their improvements nearly all methods suffer from a common limitation - the use of group labels. Some methods utilise group labels for the entire dataset to both train and tune their hyperparameters [99]. Various other methods try to reduce the number of required group labels, and e.g. use the group information from just the validation set to tune their hyperparameters [28], [71], [86], [92], or utilise the group labelled validation set to train [58] or estimate pseudo-labels [87]. The only exception we could find that claims it does not utilise group labels [111], performs quite poorly compared to the previously mentioned approaches.

The reason why the need for group labels is a huge limitation is because labels are expensive and time-consuming to get a hold of, and may sometimes even be infeasible to acquire. Furthermore, the use of group labels also requires a definition of sensible groups, a task that could be especially difficult to do for large-scale datasets, as it necessitates underlying knowledge of the shortcut opportunities in the dataset.

1.4 Contributions

In order to address the limitations above, we propose Explainability-based feature reweighting (XFR), an approach for improving group robustness without the use of group labels. Our method builds on the hypothesis that models process different groups differently, and that these underlying strategies can be learned from the explanations of the models.

The core pipeline for our method is as follows: We apply an explainability

method on a standardly trained model to acquire the heatmaps showcasing the relevant regions for the model's decision. By clustering these heatmaps we find clusters with similar explanations (or decision rules), and we utilize these to define sensible groups of the data. Once these sensible groups have been found, we can perform optimisation to balance the performance across them.

We propose two different methods for defining sensible groups from the clustering results, each with its own strengths and weaknesses. And we show that the optimisation over these groups leads to group robustness improvements when evaluating using the ground-truth groups.

One approach, named Global-XFR (G-XFR), clusters all explanations to find global similarities across different classes and once the clusters are found, divides them class-wise to acquire the groups. By leveraging global information, this approach can capture shortcut opportunities present across different classes, with the disadvantage being the need to cluster explanations from all classes (might be many).

The other approach, named Local-XFR (L-XFR), performs class-wise clustering of the explanations where the clusters can be directly viewed as the groups. This approach is more efficient as it only requires clustering of explainabilities from one class, and might better be able to capture within-class shortcuts than the global approach. The disadvantage is the lack of global information which can be useful in finding small groups within a class that have a large presence globally.

1.5 Outline of thesis

The thesis is structured into parts. First is the background part which contains chapters about Deep Learning (Chapter 2), Explainability in Deep learning (Chapter 3), Clustering (Chapter 4), and finally Shortcut Learning in Deep Learning (Chapter 5). This is followed by the method part where the Proposed Method (Chapter 6) is presented. Next is the experiment part where first an overview of the Experimental Setup (Chapter 7) is given, before the Main Experiment (Chapter 8) and Ablations (Chapter 9) are presented. Lastly, in the final part the Conclusion (Chapter 10) is given and potential ideas for Future Work (Chapter 11) are presented.

Part I

Background

/2

Deep Learning

This part covers fundamentals regarding the field of deep learning that are relevant to the thesis. The first section gives a brief overview of deep learning, while the later sections go into detail about fully-connected neural networks and convolutional neural networks. Parts of this chapter were first presented in my project thesis [110].

2.1 Overview of deep learning

Deep learning (DL) is a subfield of Machine learning (ML), which again is a small subset of Artificial intelligence (AI). It is based on artificial neural networks, which is a rough model of the structure and function of the human brain. Deep learning algorithms enable machines to learn from data, automatically improving their performance through experience, without being explicitly programmed [2].

Deep learning models leverage a hierarchical structure of multiple layers of neurons to process and learn from the input data. Each layer receives input from the previous layer, processes it, and passes it on to the next layer until the final output is produced. The training process most often involves adjusting the weights of these connections between neurons to minimize the error between the predicted output and the actual output.

In 2012 in the ILSVRC (ImageNet [24] Large Scale Visual Recognition Challenge), which is a large-scale image classification task, a deep neural network [60] won the competition outperforming other handcrafted techniques for this type of task. Ever since then, the use of DL in the field of computer vision has exploded. Today DL has achieved state-of-the-art results in a wide range of applications, not only in computer vision for tasks like classification [17], segmentation [82] and object detection [7], but also in other applications like speech recognition [98], machine translation [91], art [79], medical imaging [13], robotics [112], bioinformatics [53], natural language processing [89], cybersecurity [115], and many others [2].

Despite its impressive successes, deep learning still faces many challenges. One such challenge is to understand and explain why models make certain choices, and why they fail. Knowledge of this can help us alleviate the models' shortcomings and help us build better and more trustworthy ones. Another challenge is to make the models more robust so that they handle the noise and messiness of real-world data. Overcoming these challenges is increasingly important as DL is steadily used in more safety-critical, high-stakes, and socially-impactful domains, all meanwhile models become bigger and more complex, making their decisions harder to comprehend.

2.1.1 Deep learning approaches

Deep learning can be divided into different approaches or learning paradigms. Some of the relevant approaches to this thesis are described below.

Supervised learning is about learning from labelled data, or more specifically, it concerns problems where the task is to learn a mapping from input to output given a set of input-output examples (X, Y) , also called labelled data. Most problems can be said to be either classification or regression problems [3]. Classification is about learning class/group assignments, e.g. telling apart what is present in an image, what kind of object something is, or even determining if a person is eligible for a loan based on their personal and financial history [41]. Regression is about being able to estimate function values. This can be used to estimate coordinates of bounding boxes in object detection, estimate lane geometry and drive path for self-driving cars [50], and so on.

In *unsupervised learning* [3] only unlabeled data is used. No supervisor has instructed what the desired output should be. Hence, unsupervised learning is often used to find and learn underlying patterns in the data, as seen in various clustering and dimensionality reduction methods. These learned patterns can then be used for new insights, classification, or for data generation [102].

Semi-supervised learning is about using both labelled and unlabeled data [40]. Similarly to supervised learning, we may wish to learn a mapping from input to output using the labelled data, but this mapping can now be guided by the patterns present in the unlabeled data. This approach is especially useful in cases where a lot of data is available, but where labels are difficult or expensive to create [102].

Self-supervised learning (SSL) is a sub-field of unsupervised learning that has an approach similar to supervised learning [40], [102]. The goal is to find a mapping between input and output, and hence learn a good representation of the data, using only unlabeled data. To do this, some simple labels are created by e.g. exploiting the data structure. This process is often referred to as training using a pretext task [1]. Examples of SSL are autoencoders where the label is the input image itself, training of large language models by trying to predict the next word [121] and several others. In computer vision, there are many models that have been trained on such SSL pretext tasks where labels are easy to generate [1]. Examples of such tasks are: rotating an image and predicting its rotation, mixing an image like a puzzle and predicting its permutation, and so on. Again to reiterate, the goal is not these tasks directly, but rather for the models to extract some useful fundamental information from the unlabeled data when solving them, which may be applicable in other tasks.

2.1.2 Main building blocks

The main building blocks of most DL methods today that are relevant for the thesis (we will not be using the transformer architecture [117]), are fully connected layers and convolutional layers. These are the main building blocks that makeup Deep neural networks, which includes MLPs [116] and CNNs [39].

The Multilayer perceptron, otherwise known as a feed-forward neural network, is a deep hierarchical collection of neurons in layers, where the output of one layer is given as input to the next. Each neuron is (fully) connected to all inputs to that layer. In the neuron, a linear combination of the inputs is made and then passed through a non-linear transformation. This very flexible architecture allows Multilayer perceptron (MLP)s to do very complex tasks, and in fact, it has been shown that they theoretically can be seen as universal function approximators [49].

The Convolutional neural network (CNN), is a deep hierarchical collection of convolution operations, down-sampling operations, and non-linear transformations. This architecture is specialised for grid-like data such as images and videos. By exploiting the spatial structure of this type of data, Convolutional

neural network (CNN)s are able to outperform MLPs, being far faster and only using a fraction of the resources.

2.2 Multilayer Perceptron

This section concerns Multilayer perceptrons (MLPs) [116], also known as feed-forward neural networks, a building block used in most deep learning models today. We start off by looking at a single perceptron, the building block of an MLP, and how these can learn. We then go on to using multiple perceptrons together and putting them into layers, and introduce activation functions that make it all possible. We then go briefly through the mathematics involved in a so-called forward pass, and how the input data flows through the network. In the following sections common loss functions are introduced along with the backward pass where gradients are computed. We also give a glimpse of how optimisation algorithms use the gradients to update the model parameters. Finally, the last section covers general training procedures.

2.2.1 Perceptron

The perceptron, also known as the McCulloch-Pitts neuron [78][43], is the main building block in a multilayer perceptron (MLP). It is a simple linear model that maps an input \mathbf{x} to an output \hat{y} ,

$$\hat{y} = f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0 = \sum_{i=1}^N w_i x_i + w_0 \quad (2.1)$$

Geometrically the perceptron can be interpreted as a hyperplane defined by $f(\mathbf{x}) = 0$, where the value and sign of y indicate the distance and side, respectively, that the point \mathbf{x} is in regards to the hyperplane [3]. The hyperplane divides the input space into two parts, and it is then obvious that a perceptron can be used to perform binary classification of two linearly separable classes.

To perform this classification, one needs a method of fitting the parameters to the given data belonging to two linearly separable classes, as it is infeasible to do this for high dimensional input spaces due to the large number of possible combinations. The *pocket algorithm* [32] is a well-known way to do this, where the problem is defined as a loss function that when minimised will give parameters that completely separate the two classes. The loss function in this algorithm is minimized iteratively using gradient descent, and (guaranteed) to

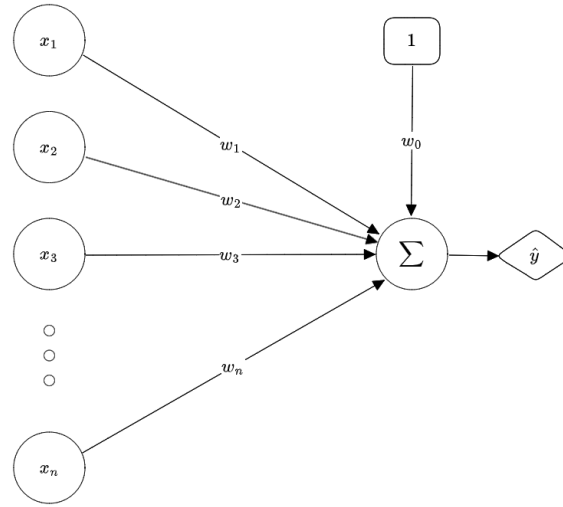


Figure 2.1: Visualisation of the perceptron. Inputs are multiplied by weights and added up to create output.

give a solution given that the classes are linearly separable. Note however that this method does not guarantee a unique solution, as there may be an infinite number of hyperplanes that separates the two classes.

Given data and corresponding labels $\{\mathbf{x}_i, y_i\}_{i=1}^N$ where $y_i = \pm 1$, and the model's prediction for each datapoint $\{\hat{y}_i\}_{i=1}^N$, the loss in the perceptron algorithm is given by:

$$\mathcal{L} = \sum_{i: \hat{y}_i \neq y_i} \hat{y}_i \cdot \text{sign}(\hat{y}_i) \quad (2.2)$$

where the magnitude of each misclassification is added up. By minimising this loss to 0, all data points are correctly classified. The hope is then that this learned hyperplane can also be used to say something about other unseen data.

Gradient descent is a method used to iteratively find a minimum of a function. Given a starting point on the function, the gradient in regard to the function parameters is calculated. The function is minimised by stepping in the negative gradient direction, or in other words, using the negative gradient to update the function parameters. Below are the update functions for the parameters used in the perceptron algorithm:

$$\mathbf{w}_{new} = \mathbf{w}_{old} - \eta \nabla_{\mathbf{w}_{old}} \mathcal{L}(\mathbf{w}_{old}) = \mathbf{w}_{old} - \eta \sum_{i: y_i \neq r_i} \mathbf{x}_i \cdot \text{sign}(y_i) \quad (2.3)$$

$$w_{0,new} = w_{0,old} - \eta \nabla_{w_{0,old}} \mathcal{L}(w_{0,old}) = w_{0,old} - \eta \sum_{i: y_i \neq r_i} \text{sign}(y_i) \quad (2.4)$$

By repeatedly alternating between performing prediction of the data points and applying the update rules (until the loss is 0), a hyperplane that separates the data is found.

2.2.2 From one to many layers

The perceptron works fine in cases where the data is linearly separable. Unfortunately, this is usually not the case, as illustrated by the famously simple XOR problem [81]. This problem requires two hyperplanes to correctly separate the two classes. An intuitive way to deal with this would be to combine the output of two perceptrons. Note however that directly combining the output of two perceptrons in a new perceptron does not have the intended behaviour, and results in only a single hyperplane. That is because a linear combination of linear combinations is still linear as demonstrated in Equation 2.5.

Assume two perceptrons with outputs \hat{y}_1 and \hat{y}_2 , that we wish to combine to create output \hat{y} .

$$\begin{aligned}\hat{y} &= w_a \hat{y}_1 + w_b \hat{y}_2 + w_0 = w_a(\mathbf{w}_1^T \mathbf{x} + w_{10}) + w_b(\mathbf{w}_2^T \mathbf{x} + w_{20}) + w_0 \\ &= (w_a \mathbf{w}_1 + w_b \mathbf{w}_2)^T \mathbf{x} + (w_0 + w_a w_{10} + w_b w_{20}) \\ &= \mathbf{w}^{*T} \mathbf{x} + w_0^*\end{aligned}\tag{2.5}$$

Note how the output of such an arrangement is also linear.

To solve this problem we introduce what is called an activation function, which is tasked with bringing in non-linearities to our models. Activation functions have turned out to be a vital discovery for deep learning enabling deep neural networks (DNNs), where layers of neurons are connected one after the other to approximate complex non-linear functions [116]. The most used activation functions are presented in Subsection 2.2.4.

The layers in an MLP are usually divided into 3 classes. The input layer consists of the input data to the model, while the output layer consists of the last neuron layer and its output. Hidden layers is the name given to all layers between these two. As an example, the perceptron has an input and an output layer, while the model described in Equation 2.5 additionally has 1 hidden layer. An MLP is usually considered shallow if it has 1 hidden layer, and deep if it has more than that [54].

The Universal Approximation Theorem [49], states that a network with one hidden layer with an infinite number of neurons can approximate any continuous function to an arbitrary precision, as long as an activation function is used. In reality, this is infeasible, and many problems are in fact not continuous.

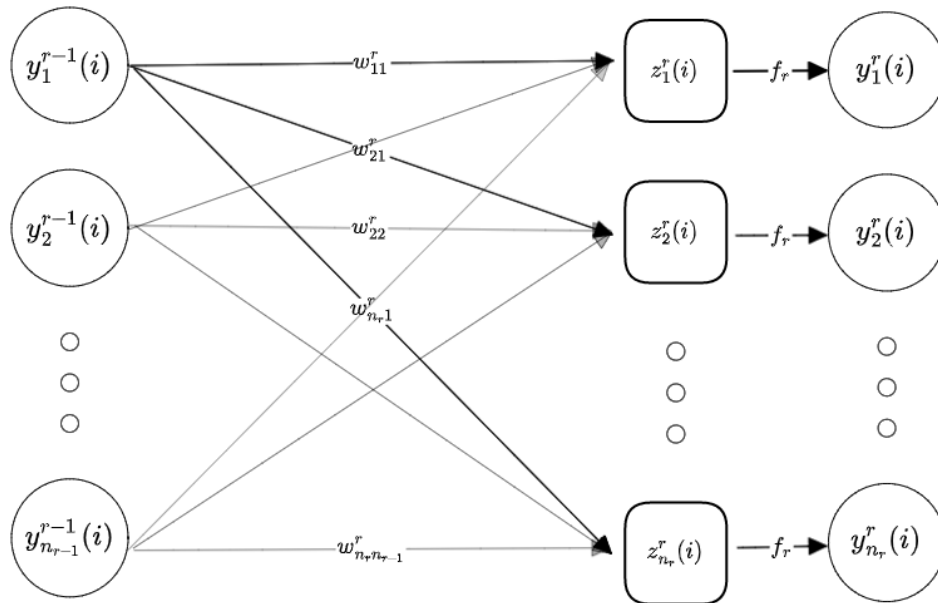


Figure 2.2: The forward pass. Illustration of the feed-forward process for layer r of an MLP.

However, it has also been shown that this network property can be attained by using a finite number of neurons if multiple layers are used [77].

2.2.3 Forward pass

In this section, we cover some notation and use it to describe the forward pass (forward propagation, feed-forward process) of an MLP. The forward pass is an expression used to describe the process of the input moving through the layers (neurons and activations) resulting in the model output. To describe this process we will use a notation similar to that used in [116].

- We have a dataset containing N data points $\mathbf{x}(i)$, where i ranges from 1 to N . Each data point is of dimensionality n_0 .
- The neural network has layers labeled as $r = 1, 2, \dots, L$. The initial layer, $r = 0$, serves as the input layer.
- For each layer r , there are n_{r-1} inputs and n_r outputs. The number of neurons in a layer is the number of outputs.
- The inputs to the network are defined as $x_j(i) = y_j^0(i)$, where j ranges

from 1 to n_0 .

- The j outputs from the previous layer $y_j^{r-1}(i)$, together with a bias term $y_0^r(i) = 1$, serve as inputs for layer r .
- We let w_{kj}^r describe the weight connecting the input j to the neuron k in layer r .
- The pre-activation output of neuron j in layer r is denoted as $z_j^r(i)$, where j ranges from 1 to n_r .
- The post-activation output of neuron j in layer r is represented as $y_j^r(i) = f(z_j^r(i))$ for j from 1 to n_r , where f is the activation function.
- The network's final output or prediction is indicated using the notation $\hat{y}_j(i) = y_j^L(i)$, where the output neurons are denoted by j ranging from 1 to n_L .
- If available, the actual data labels are represented as $y_j(i)$.

Note that these expressions may be converted to vectors and matrices representing each layer.

$$\mathbf{z}^r(i) = \begin{bmatrix} z_1^r(i) \\ \vdots \\ z_{n_r}^r(i) \end{bmatrix}, \mathbf{y}^r(i) = \begin{bmatrix} 1 \\ y_1^r(i) \\ \vdots \\ y_{n_r}^r(i) \end{bmatrix}, \mathbf{w}_k^r = \begin{bmatrix} w_{k0}^r \\ w_{k1}^r \\ \vdots \\ w_{kn_r}^r \end{bmatrix}, W^r = \begin{bmatrix} (\mathbf{w}_1^r)^T \\ (\mathbf{w}_2^r)^T \\ \vdots \\ (\mathbf{w}_{n_r}^r)^T \end{bmatrix} \quad (2.6)$$

Using these vectorised representations the forward pass through a hidden layer can be described in the following way

$$\mathbf{y}^r(i) = \begin{bmatrix} 1 \\ f_r(\mathbf{z}^r(i)) \end{bmatrix} = \begin{bmatrix} 1 \\ f_r(W^r \mathbf{y}^{r-1}(i)) \end{bmatrix} \quad (2.7)$$

where f_r is the activation function for layer r . For the output layer, the 1 is not needed:

$$\hat{\mathbf{y}}(i) = f_L(\mathbf{z}^L(i)) = f_L(W^L \mathbf{y}^{L-1}(i)) \quad (2.8)$$

2.2.4 Activation functions

The importance of activation functions was discussed in Subsection 2.2.2. Here we chose to present the most commonly used activations; their qualities, ad-

vantages and disadvantages, along with their derivatives which are used in the backward pass to update model parameters.

Linear

Linear activation, otherwise known as the identity activation, is the same as not using an activation. As previously discussed, not having an activation between layers of the network is counterproductive and equivalent to just using the final layer. Then, why even address linear activation? Because this type of activation is useful for the output layer of models used for regression where the output can take any value.

$$f(x) = x, \quad f'(x) = 1 \quad (2.9)$$

ReLU

The rectified linear unit (ReLU) [84][52] is the go-to activation function for hidden layers in most deep learning models. It is a piece-wise function often denoted by $\max(0, x)$, which is linear for positive values and 0 for negative values. The ReLU is a simple and resource-efficient function and has been shown to increase the effectiveness of training deep models.

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{else} \end{cases}, \quad f'(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{else} \end{cases} \quad (2.10)$$

Note that for negative values the output and gradient are 0. This is a problem for learning because if a neuron is only capable of outputting negative values it will never fire and will thereby never be updated. This problem has been dubbed the dying ReLU problem [73], and caused the emergence of many variations of the ReLU with non-zero value outputs for negative values (e.g. Leaky ReLU [76], ELU [19], GeLU [46], Swish [93], ...).

Sigmoid

The sigmoid [20], a type of logistic function, is a continuous monotonically increasing function with horizontal asymptotes at $\{0, 1\}$ for inputs $x \rightarrow \pm\infty$. It can be seen as a smooth, and hence differentiable, version of the standard step function. Hence it maps input onto the range 0 to 1. The Sigmoid is used as an activation function in hidden layers (if the network has few layers), as an output layer if a value between 0 and 1 is required, and in recurrent neural networks (GRU and LSTM).

$$f(x) = \frac{1}{1 + e^{-x}}, \quad f'(x) = f(x) (1 - f(x)) \quad (2.11)$$

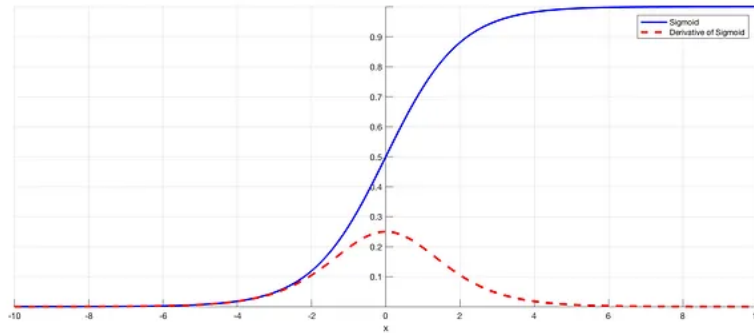


Figure 2.3: The sigmoid function and its derivative.

Note in Figure 2.3, that the derivative of the sigmoid is positive and has a maximum value of 0.25. This causes the so-called *vanishing gradient problem* [11] when using the sigmoid since DNNs require repeated multiplications by the gradient of the activation function to update model parameters. If the sigmoid is used as the activation function in a deep model only the last layers will be updated, as the gradients will be too small in the earlier layers to make significant updates.

Hyperbolic tangent

The hyperbolic tangent [40], usually just denoted \tanh , is a logistic function similar to the sigmoid function (in fact it can be defined in terms of the sigmoid). The \tanh is similarly s-shaped but has horizontal asymptotes at $\{-1,1\}$ for inputs $x \rightarrow \pm\infty$.

$$f(x) = 2 \cdot \text{sigmoid}(x) - 1 = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \quad f'(x) = 1 - f(x)^2 \quad (2.12)$$

Note that the derivatives of the \tanh are in the range $(0, 1)$, which also causes vanishing gradients.

Softmax

The softmax activation function [15] is used to normalise the outputs of a layer so that they are all positive and sum to 1. In this way, the outputs of the layer may be interpreted as a probability distribution, a useful property for e.g. the output layer of models used for classification where the outputs can be interpreted as the confidence of the model.

$$f(\mathbf{x})_k = \frac{e^{x_k}}{\sum_j e^{x_j}}, \quad \frac{\partial f(\mathbf{x}_k)}{\partial x_m} = f(\mathbf{x}_k)(1[m = k] - f(\mathbf{x}_m)) \quad (2.13)$$

where $1[m = k]$ is 1 if $m = k$ and 0 otherwise. Note that the post-activation output of a neuron is affected by all neurons in that layer.

2.2.5 Loss functions

A loss function is a function used during the training of DL models to measure the deviation between the model's current prediction and an ideal solution [40]. The goal of training is to minimize this loss (for the model to learn), and hence make the predictions more like the ideal solutions. Loss functions are task-specific, and within tasks, there exist many variations [119]. Their definitions obviously influence what the model learns, but they also influence how the model learns and may introduce some extra constraints (e.g. small or sparse parameters).

The basic setup is usually a differentiable function of this form

$$\mathcal{L} = \sum_{i=1}^N \varepsilon(i) \quad (2.14)$$

where each datapoint i gives rise to an error term ε , which are then added together.

Here are listed some of the most basic loss functions used when labels for the data are available.

The *binary cross-entropy loss* is used in binary classification, usually when MLP has one neuron in the last layer. It is derived from the negative log-likelihood function and has therefore desirable properties [40]. Using labels $y_1(i) \in \{0, 1\}$ the expression for binary cross-entropy loss is given as:

$$\mathcal{L} = - \sum_{i=1}^N ((1 - y_1(i)) \log(1 - \hat{y}_1(i)) + y_1(i) \log(\hat{y}_1(i))) \quad (2.15)$$

The *cross-entropy loss* is a generalisation of the binary cross-entropy to more classes and is used in classification when an MLP has more than one neuron in the last layer. Using labels in a one-hot encoding ($\mathbf{y}(i)$ has one entry of 1 and is otherwise 0), the expression for the cross-entropy loss is given by:

$$\mathcal{L} = - \sum_{i=1}^N \sum_{j=1}^C [y_j(i) = 1] \log(\hat{y}_j(i)) \quad (2.16)$$

The *Mean absolute error loss* is often used for regression problems since it penalises the absolute difference between the predicted values and the true

values. Its expression is simply given by

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^N \sum_{j=1}^C |y_j(i) - \hat{y}_j(i)| \quad (2.17)$$

The *Mean squared error loss* is also used for regression problems as it punishes differences between predicted values and true values. However, in contrast to mean absolute error, mean squared error penalises large deviations more than small ones since it uses the squared deviation. The expression for this loss is:

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^N \sum_{j=1}^C (y_j(i) - \hat{y}_j(i))^2 \quad (2.18)$$

These are just a few common loss functions. Many variants exist, reflecting the desired properties. Some loss functions include regularisation terms to better control the learning process. An example of such a regularisation method is the addition of a weight decay term, which encourages the model to learn smaller weights. This can be achieved by adding an L1 term $\lambda \sum_k |w_k|$, or an L2 term $\frac{\lambda}{2} \sum_k w_k^2$, where the weight decay hyperparameter λ governs the strength of the regularisation. L2 regularisation is the most commonly used type of weight decay and has been shown to be effective in many different types of models [61].

2.2.6 Backward pass

After a forward pass through a network, a prediction is acquired $\hat{\mathbf{y}}(i)$ for each datapoint i . Depending on the task, this prediction may be combined with the data label $\mathbf{y}(i)$ to calculate a loss value. Then similar to the case with the perceptron, gradients with regard to the different network parameters can be calculated. These can then be used through gradient descent to update the network weights to decrease the loss, and hence improve the network. This process of moving backwards from the loss value to calculate gradients and update parameters is called back-propagation [97], or the backward pass of the model. Below is the mathematics describing the process of calculating the gradients.

To calculate the gradient of the loss function with regards to the weights of a neuron k in the last layer L we utilise the chain rule:

$$\nabla_{\mathbf{w}_k^L} \mathcal{L} = \frac{\partial \mathcal{L}}{\partial \mathbf{w}_k^L} = \sum_i \frac{\partial \varepsilon(i)}{\partial \mathbf{w}_k^L} = \sum_i \frac{\partial \varepsilon(i)}{\partial \hat{y}_k(i)} \frac{\partial \hat{y}_k(i)}{\partial z_k^L(i)} \frac{\partial z_k^L(i)}{\partial \mathbf{w}_k^L} \quad (2.19)$$

In the final expression, the first term depends on the choice of the loss function, the second term on the choice of activation function, and the third term is:

$$\frac{\partial z_k^L(i)}{\partial \mathbf{w}_k^L} = \frac{\partial [(\mathbf{w}_k^L)^T \mathbf{y}^{L-1}(i)]}{\partial \mathbf{w}_k^L} = \mathbf{y}^{L-1}(i) \quad (2.20)$$

By combining Equation 2.19 and Equation 2.20, and introducing δ to simplify notation, the gradient is given by:

$$\nabla_{\mathbf{w}_k^L} \mathcal{L} = \sum_i \frac{\partial \varepsilon(i)}{\partial \hat{y}_k(i)} \frac{\partial \hat{y}_k(i)}{\partial z_k^L(i)} \mathbf{y}^{L-1}(i) = \sum_i \delta_k^L(i) \mathbf{y}^{L-1}(i) \quad (2.21)$$

To be able to calculate the gradient of the loss for earlier layers we need to utilise the following:

$$\frac{\partial z_k^r(i)}{\partial y_j^{r-1}(i)} = w_{kj}^r. \quad (2.22)$$

Note that the neuron j in layer $L - 1$ affects all neurons in the following layers so when calculating the gradient need to put together the effects of all these neurons. Following a process similar to Equation 2.19, can show that:

$$\nabla_{\mathbf{w}_j^{L-1}} \mathcal{L} = \sum_i \left(\sum_{k=1}^{n_L} \delta_k^L(i) w_{kj}^L \right) \frac{\partial y_j^{L-1}(i)}{\partial z_j^{L-1}(i)} \mathbf{y}^{L-2}(i) \quad (2.23)$$

$$= \sum_i \delta_j^{L-1}(i) \mathbf{y}^{L-2}(i) \quad (2.24)$$

This process for finding gradients can be generalised for any layer and is described below.

Gradient calculation

The gradients for parameters in any layer can be found with:

$$\nabla_{\mathbf{w}_j^r} \mathcal{L} = \sum_{i=1}^N \delta_j^r(i) \mathbf{y}^{r-1}(i) \quad (2.25)$$

starting from

$$\delta_j^L(i) = \frac{\partial \varepsilon(i)}{\partial \hat{y}_j(i)} \frac{\partial \hat{y}_j(i)}{\partial z_j^L(i)} \quad (2.26)$$

and then using the following recursive formula:

$$\delta_j^{r-1}(i) = \left(\sum_{k=1}^{n_r} \delta_k^r(i) w_{kj}^r \right) \frac{\partial y_j^{r-1}(i)}{\partial z_j^{r-1}(i)}, \text{ for } r = L, L - 1, \dots, 2 \quad (2.27)$$

Once the gradients have been found the model parameters can be updated through gradient descent:

$$\mathbf{w}_j^r(\text{new}) = \mathbf{w}_j^r(\text{new}) + \eta \nabla_{\mathbf{w}_j^r} \mathcal{L} \quad (2.28)$$

2.2.7 Gradient descent variants and optimisation algorithms

The gradient descent method in Equation 2.28 requires the use of all available training data, an entire batch. This is known as batch learning [66]. This can be a problem in cases where there is a lot of data, or where the size of each data point is large, as there might not be enough memory to process it all. A natural solution to this problem is to randomly divide the data into chunks, or mini-batches, and pass these individually through the network and update the network weights through so-called stochastic learning. This allows the model to be updated multiple times per complete pass of the training set, known as epoch, which apart from reducing memory demands also has been linked to increased performance [66][14].

The technique mentioned above is usually referred to as mini-batch Stochastic gradient descent (SGD), and is the most widely used gradient descent method, and is performed by doing Equation 2.28 for each mini-batch. The mini-batch size governs the trade-off between doing many noisy steps with approximations of the true gradient and doing fewer larger steps with more accurate approximations. The exact size of the mini-batches is different for different applications, and in many cases is often set as high as the memory capacity allows.

Several ways of optimising SGD to find the local minimum of the loss function faster and more efficiently have been proposed [95]. Here we present two of these methods.

SGD with momentum

Gradient descent with momentum was introduced to decrease the number of oscillations experienced by vanilla SGD. This method accelerates SGD convergence by adding part from the previous gradient to the current update, so as to keep momentum, or steps moving in the same direction. In this way, parameters whose gradients point in the same direction receive large updates, while parameters whose gradient change direction receive smaller updates [95].

The update function is performed in two steps. First, a momentum term v is updated for each parameter using the current gradients, then this term is used to update the model parameters.

$$v_t = \gamma v_{t-1} + \eta \nabla_{\theta} \mathcal{L} \quad (2.29)$$

$$\theta_t = \theta_{t-1} - v_t \quad (2.30)$$

The momentum term γ is usually set to 0.9.

Adam

Adaptive moment estimation [56], or simply Adam, is a method that computes adaptive learning rates for each parameter. In Adam, the current gradient for each parameter g_t is used to calculate exponentially decaying averages, of both previous gradients m_t and squared previous gradients v_t . These are estimates of the first and second moment respectively (hence the name "Adam").

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \quad (2.31)$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \quad (2.32)$$

The authors of [56] argue that since m_t and v_t are initialised as vectors of 0s, this causes the moment estimates to be biased, especially during initial timesteps and when the decay rates are small, meaning the β -values are close to 1. Therefore they do a bias correction of the moments.

$$\hat{m}_t = \frac{m_t}{1 - (\beta_1)^t} \quad (2.33)$$

$$\hat{v}_t = \frac{v_t}{1 - (\beta_2)^t} \quad (2.34)$$

Then using these values the update function is given by:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \quad (2.35)$$

The authors of [56] propose to use the following hyperparameters: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$.

2.2.8 General training procedure

To train a deep learning model, we need data we wish to learn from. For supervised learning the data ought to be divided into 3 non-overlapping sets:

training, validation and test set, for example consisting of 75-15-15 percent of the entire dataset respectively. The training set will be used by the model to learn. The validation set is used to check the performance of the model during training and to choose hyperparameters, and can also be used to stop training when the model's performance starts decreasing. Meanwhile, the test set is used only for the final evaluation of the model.

To build a model, we need to specify the task and choose the architecture, including the number of layers, neurons per layer, and activation functions. We also need to select the loss function, optimiser, how weights are initiated, in addition to selecting several hyperparameters like mini-batch size, the number of epochs, optimisation parameters, loss parameters, and so on. These decisions are critical to the model's performance and should be carefully considered based on the problem's complexity and the available data.

Once all the model parameters have been set, training can begin. The training process involves iterating over the entire training dataset for the specified number of epochs. During each epoch, the training dataset is split into random mini-batches of the chosen size. For each mini-batch, the following steps are performed: do a forward pass, calculate loss, do a backward pass to find gradients, and finally update the model weights using the chosen optimisation method.

Loss on the training set should go down over time. To combat the model overfitting or memorising the training set, performance on the validation set should also be tracked. This can be done at the end of every epoch or every few epochs by doing a forward pass with the validation set and calculating its loss. The validation loss indicates what the model is actually learning. Usually during training the model weights that achieved the lowest validation loss are stored, so even if the model trains for too long the best performing version is kept.

Once the model has finished training, the best weights are selected based on the validation set performance. Note that earlier it was mentioned that the validation set can be used to select hyperparameters, this is done by repeating the training procedure for a different selection of hyperparameters and finally choosing the best model among them with regard to the validation set performance. The final model's performance is then evaluated by doing a forward pass using the test set. This is the model's true performance on this task since both the training and validation sets affected the final model.

2.3 Convolutional neural networks

MLPs are not perfect, one shortcoming is that when inputs are very large they require many weights (easier to overfit). Examples of such inputs are images. Images are also grid-like data structures, something that is not exploited by MLPs, as they will take in all the images as input. Natural to think that pixels are most related to what is immediately around them compared to all pixels in the image. The use of convolutional layers connects only close pixels in regions together, and solves the many weights issue by using weight sharing between the "neurons". Convolutional neural networks (CNNs) are networks that use convolutions in at least one of its layers instead of the usual matrix multiplication [40]. Usually divided into two parts, a feature extractor and a head. The feature extractor is usually made up of convolutional layers, pooling and activations, while the head can be anything like e.g. a classification network.

In this section, we will focus on the feature extractor part of CNNs. The first subsection introduces convolutions, then the next ones look into the main building blocks of the feature extractors, convolutional layers and pooling layers, and how forward pass is done for these. Then we present an existing CNN architecture.

2.3.1 Convolution operation

A convolution is a mathematical function that takes in two functions. The functions can in principle be discrete or continuous, but will here focus on discrete signals. The usual setup is a 1D, 2D or 3D discrete signal \mathbf{x} and a (small) filter or kernel signal \mathbf{w} . The convolution can be thought of as sliding the filter across the signal, computing the element-wise multiplication of the elements and returning a value. Doing this process for all possible positions in the signal will give the result, a filtered version of the signal (see Figure 2.4).

The 1D discrete convolution can be described as a weighted moving average,

$$\mathbf{y}_i = (\mathbf{w} * \mathbf{x})_i = \sum_l w_l x_{i-l} \quad (2.36)$$

The 2D discrete convolution can be thought of as a sweeping window [39],

$$\mathbf{y}_{i,j} = (\mathbf{w} * \mathbf{x})_{i,j} = \sum_l \sum_h w_{l,h} x_{i-l,j-h} \quad (2.37)$$

The 3D convolution can be thought of as a moving cuboid. What is usually used

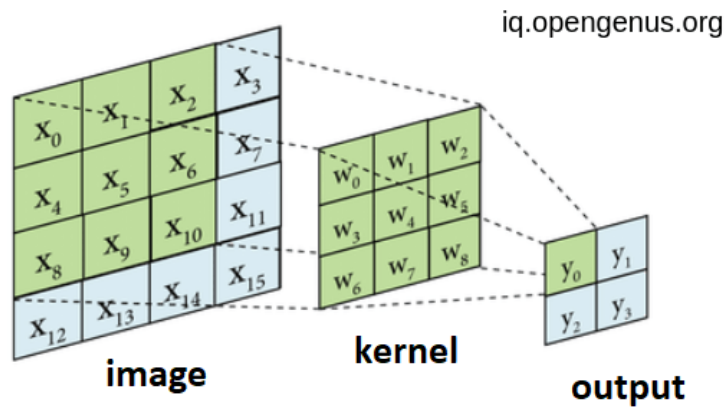


Figure 2.4: 2D convolution. Can be interpreted as sliding the kernel over all the possible positions of the image.

in CNNs are cuboids with a selected height and width and the same depth as the input. Hence the relevant 3D convolutions are akin to stacking several 2D. An example of a 3D signal is RGB images which have 3 channels one for each color (red, green, blue). In this case, the filter is as deep (has as many channels) as the signal.

$$\mathbf{y}_{i,j} = (\mathbf{w} * \mathbf{x})_{i,j} = \sum_l \sum_h \sum_c w_{l,h,c} x_{i-l,j-h,c} \quad (2.38)$$

Note that the input to the convolution is 3D but that the output is 2D. By parallelizing the convolutions the output can be made into 3D. For example, if k convolutions are done in parallel, their output can be concatenated so as to create a multi-channelled output. This concept is key to the convolutional layers in a CNN.

Note that in Figure 2.4 the filter is never centred over the edge values and that the output is of a different size than the input. The size difference depends on the kernel width F , and $\lfloor F/2 \rfloor$ of each edge is lost when doing a convolution. The fact that the output shrinks in size can sometimes be troublesome, especially when building CNNs since we wish to stack many convolutions after one another.

We introduce padding, adding extra values of width $\lfloor F/2 \rfloor$ around each edge so that the resulting output is of the same size as the input. Different types of padding exist varying in what values are used for the extensions. Zero-padding is quite common, or padding by some other constant values, duplication of edges, and wrapping of edges (pad with a copy of the opposite edge).

2.3.2 The convolutional layer

A convolutional layer receives input \mathbf{x} , e.g. a RGB image, consisting of elements $x_{i,j,k}$. The layer has weights W , in the shape of a cuboid. The cuboid is often called a kernel, and has usually equal and odd-sized sides for the two first dimensions, while the depth is the same as that of the input. The forward pass in a convolutional layer, where \mathbf{x}^* denotes the padded version of the input, is given by:

$$z_{i,j} = \sum_l \sum_h \sum_c w_{l,h,c} x_{i-l,j-h,c}^* + b \quad (2.39)$$

Note that in the first part, we are doing element wise-multiplication between the kernel and a part of the input (a cuboid of the same size as the kernel) and that these are then added together with a bias. Notice that if the cuboids in the first part are flattened out, this expression can be expressed as an inner product. With a little more work this calculation can look just like Equation 2.7.

If the 0-th element is the bias and $F = LxHxC$ is the number of elements in the kernel, then a flattened kernel can be expressed as such

$$\mathbf{w}_1 = [w_{10}, w_{11}, \dots, w_{1F}]^T \quad (2.40)$$

We can also flatten out each cuboid for each input n . The cuboids, those that will be multiplied with the sliding window, can be given index $r = 1, 2, \dots, i \cdot j$. A flattened cuboid with index r belonging to input n can be flattened out as such:

$$\mathbf{x}_r(n) = [1, x_{r1}(n), x_{r2}(n), \dots, x_{rF}(n)]^T \quad (2.41)$$

Observe then that the convolution from Equation 2.38 can be expressed as:

$$y_{1r}(n) = \mathbf{w}_1^T \mathbf{x}_r(n) \quad (2.42)$$

And that the convolution of the whole input n can be written as:

$$\mathbf{y}_1(n)^T = \mathbf{w}_1^T X = \mathbf{w}_1^T [\mathbf{x}_1(n), \mathbf{x}_2(n), \dots, \mathbf{x}_{i \cdot j}(n)] \quad (2.43)$$

Bear in mind that the output vector must be reshaped back into the appropriate shape (we must undo the reshaping made to the input) once the calculation is completed. Note that we can easily generalise from this example to see how multiple kernels can be added. If a matrix W is created with each kernel as a column, then the final expression for a forward pass in a convolutional layer may be written as:

$$I(n) = \text{reshape}(Y(n)) = \text{reshape}(W^T X(n)) \quad (2.44)$$

Here the output $I(n)$, if padded appropriately, has the same height and width as the input, and a number of channels equal to the number of kernels used.

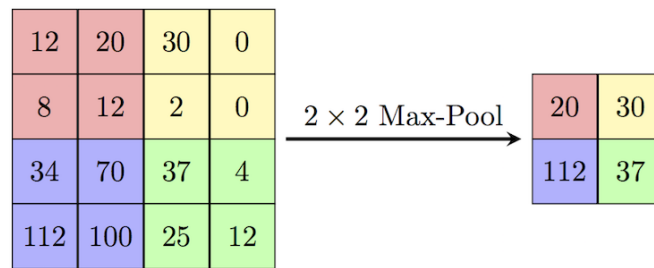


Figure 2.5: 2D pooling. Pooling performed with window size 2x2 and stride 2.

With this new expression, we can see that the backward pass for convolutional layers can be found in a similar way as those for the fully connected layers, just requiring some reshaping before and after each layer. The first reshape is to undo the last forward pass reshape, and hence create the matrices W , X and Δ (the equivalent of δ). The second reshape is to get these matrices into the correct shape. The W matrix can simply be reshaped to match the dimension of the kernel, while the gradients that will be passed to earlier layers will need to be uncubed (undo sliding window cubification) and reshaped to the size of the input to this layer. Note that the cuboids from the forward pass were overlapping, so the gradient cuboids now also will be. This is dealt with by adding together their contributions.

2.3.3 Pooling

Pooling is a downsampling operation used in CNNs performed as a sliding window for each channel. The most used pooling setup halves the input's width and height by utilising a window of size 2x2 and a stride of 2, which essentially divides the image into non-overlapping 2x2 windows (see Figure 2.5). Stride defines the amount the sliding windows move, for example in Figure 2.4 the stride was 1, since the windows slides 1 unit at the time.

There are different types of pooling, differentiated by what mathematical operation they implement. The two most common types are max-pooling, which returns the maximum value in the window, and average-pooling, which returns the average of the window.

Pooling is usually applied after one convolutional layer or a block of convolution layers. An activation function, such as the ReLU, is usually also applied after the pooling layer.

Pooling has become common to use as the last operation when going from convolution layers to fully connected ones. Known as global average pooling,

the kernel size is set to be the input size, so that each channel gets averaged down to a single number. What we are left with is a vector that can be given as input to a MLP. The advantage of using global average pooling is that the input to the CNN does not need to be of a fixed size, just above a certain minimum size.

2.3.4 Example of simple CNN architecture

In Figure 2.6 is shown an example of a simple CNN architecture, the VGG-16 network [108]. The figure illustrates how CNNs are usually structured and how the data is transformed throughout the network. The network is divided into a feature extractor part (blue and red) and a classification head (green). In the feature extractor, there are several convolution layers with ReLU activation, that are sometimes followed by pooling layers to reduce the spatial resolution. Note that 64 parallel convolutions are performed in each of the first two layers and that the number of convolutions increases as the spatial resolution decreases. In the classification head, the output of the feature extractor is flattened and passed through several fully connected layers to ultimately result in the model output, a vector of size 1000. Each of the values in this output represents a class of the dataset the model is trained on (in this case the 1000 classes of ImageNet [24]), and the index of the largest value represents the model prediction.

To summarize, Figure 2.6 illustrates the process of how an RGB image of size 224×224 can be made into a prediction by a CNN. Obviously, for this to perform well the many weights of the network (approx. 138 million of them) must first be adjusted through training.

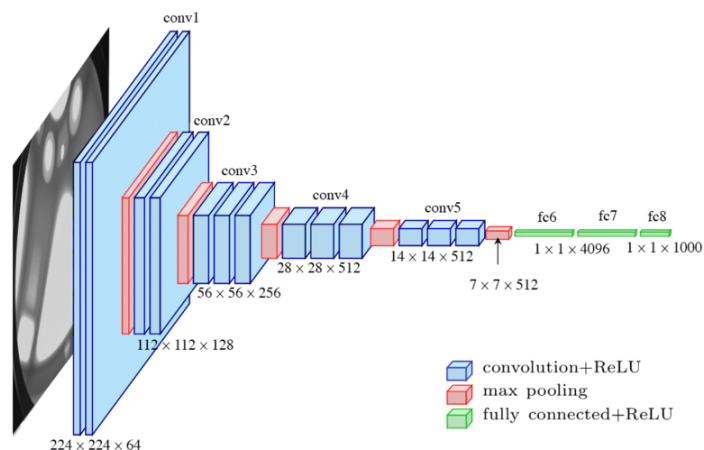


Figure 2.6: Simple deep CNN example: the standard VGG-16 network architecture [108].

/ 3

Explainability in Deep Learning

In this chapter, we will present the concept of explainability within deep learning. We first lay some groundwork for why we need explainability, and why it is important. We then give some overview of different types of explainability methods before closely explaining Layer-wise relevance propagation (LRP) [8], which we will use later in our proposed method.

3.1 Deep learning models are blackboxes

Imagine you are using the CNN presented in the previous chapter (Figure 2.6) to classify images of cats and dogs. After the model has made its prediction, consider the challenge of explaining why the model arrived at this particular decision. This task is not trivial, as it was a collection of mathematical operations that were performed in series that lead to the prediction. This is a problem that affects not only our simple example but most deep learning applications. For this reason, deep learning models are often referred to as "black box" models, as we have no insight into what is happening within them [38].

Not understanding or being able to explain why a decision was made can be an issue in certain fields [4], [55]. Imagine for a second that the task is no longer

"tell apart cats from dogs", but instead by looking at some medical imaging to determine if a person suffers from some deadly disease. It is obvious that before assigning the patient to some treatment, the doctor would want to be sure that what the model is saying is true. Deep learning by itself lacks this form of explainability and traceability, which makes it hard to rely on and trust [123]. This lack of grounding for decisions has led to slower adoption of deep learning algorithms in safety-critical, high-stakes, and socially-impactful domains [37], [45], [106]. It is also noteworthy to mention, that some governments have put in place regulations to limit the scope of AI and require some explainability when it is used for decision-making (see e.g. GDPR [30] Art. 22, Art. 15 (1) h., and art.13 (2) f.).

3.2 XAI

To address the shortcomings related to "black box" models, the concept of EXplainable AI (XAI) has emerged. XAI aims to make decisions made by ML and DL algorithms transparent, interpretable and comprehensible, with the goal of elevating a user's trust, confidence and understanding when interacting with AI models [38].

Explainability can be divided into two categories: local and global. Local explainability [70] entails explaining individual decisions, providing insight into the reasoning behind a prediction or action. Meanwhile, global explainability [100] considers the overall behaviour of the models and how it behaves across all instances. Both of these categories are important and have given rise to several methods [48].

Usually, a different categorisation of explainability methods is used. Rather than categorising based on the scope of the explanations, we instead categorise by how the explanations are created. In the following subsections we will describe these: transparent and post-hoc methods.

3.2.1 Transparent methods

Transparent methods refer to the use of ML methods that are straightforward, simple, and easily interpretable. These methods, such as logistic regression, decision trees, and k-nearest neighbours, are explainable due to their simple nature [3]. While they might not match the performance of complex deep learning models, they provide insights into their decision-making. In some cases where explainability is a priority, transparent methods are used despite having worse accuracy than deep learning models in the same task [96]. Because of

this trade-off, some recent effort has been put into the creation of transparent deep learning models. Some examples of such models are ProtoPNet [18] and ProtoVAE [33].

3.2.2 Post-hoc methods

Post-hoc methods are used with black box models to explain the complex relationship between features that have been learned. These methods start off with a trained model and create explanations for it. Such methods are usually divided into model-agnostic and model-specific methods.

Model-agnostic methods provide explanations that are not tied to a specific model type. These methods usually rely purely on input-output pairs to create explanations, meaning they assume the model to be a black box - allowing them to be used with any model. Some well-known methods that fall within this category are Local Interpretable Model-Agnostic Explanations (LIME) [94], and SHapley Additive exPlanations (SHAP) [74].

In contrast, **model-specific** methods are designed to explain the decisions of a specific model architecture. These methods utilise the specific internal structure and architecture of the model to generate explanations. Intuitively, by utilising the internal information of the models it should be easier to create explanations. The drawback of such methods is their reduced flexibility as they must be created for specific models. An example of such a method is LRP [8], which we will cover in the following section.

3.3 Layer-wise Relevance Propagation

Layer-wise relevance propagation (LRP) [8] is a technique aimed at explaining the importance of each input to the final output $f(x)$. The method begins from the output layer assigning relevance to the part of the output we wish to explain, e.g. the model's prediction. This relevance is then backpropagated using some special propagation rules until the input is reached. The result from this is a heatmap that showcases the importance of each input value to the result in the last layer. This heatmap, therefore, serves as an explanation of the model's decision, a local explanation (see example in Figure 3.1).

In the next subsection, we present the core principle in LRP and the relationships between the relevances of different layers. In the following subsection, we showcase a selection of popular propagation rules that we will utilise later in the thesis. Finally, we comment on how these rules are being used in prac-

tice.

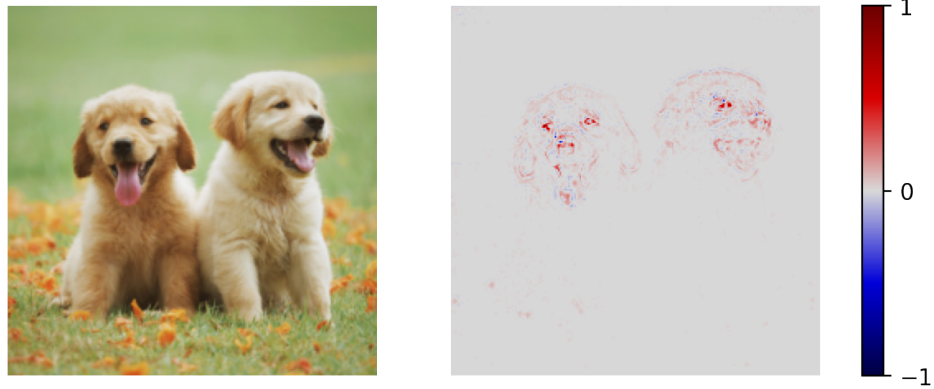


Figure 3.1: Example of an LRP heatmap from a VGG-16 model showcasing the parts of the input that are relevant for the prediction "golden retriever". Red and blue represent positive and negative relevance respectively. Here we have utilised the following propagation rules: LRP- $\alpha\beta$ with $\alpha = 2$ for the convolutional layers, and LRP- ϵ for the fully-connected layers.

3.3.1 Propagation mechanisms

The core principle of LRP is the layer-wise conservation rule which states that the total relevance throughout each layer remains constant:

$$f(x) = \dots = \sum_{i \in l+1} R_i^{(l+1)} = \sum_{i \in l} R_i^{(l)} = \dots = \sum_i R_i^{(1)} \quad (3.1)$$

where we use the notation $R_i^{(l)}$ to denote the relevance of neuron i in layer l . This rule ensures that the resulting heatmap is consistent with regard to the output we are trying to explain.

Following from the conservation rule, and the fact that LRP is performed layer-wise, the relevance $R_i^{(l)}$ must be describable by the relevances in the subsequent layer $R_j^{(l+1)}$. This relationship, which is integral for backpropagation, can be expressed as

$$R_i^{(l)} = \sum_j R_{i \leftarrow j}^{(l,l+1)}, \quad (3.2)$$

where the notation $R_{i \leftarrow j}^{(l,l+1)}$ denotes the relevance messages sent from neuron j in layer $l + 1$ to neuron i in layer l . Naturally, a corresponding relationship exists in the reverse direction:

$$R_j^{(l+1)} = \sum_i R_{i \leftarrow j}^{(l,l+1)}. \quad (3.3)$$

3.3.2 Different propagation rules

A variety of LRP propagation rules exists [83], and we will here present some of the most popular ones. These rules essentially govern how the relevance messages $R_{i \leftarrow j}^{(l, l+1)}$ in Equation 3.2 are calculated.

The simplest of the propagation rules referred to as LRP-0, is defined as the ratio between local and global pre-activations:

$$R_{i \leftarrow j}^{(l, l+1)} = \frac{z_{ij}}{z_j} \cdot R_j^{(l+1)} \quad (3.4)$$

where z_{ij} is the contribution from neuron i to the activation of neuron j (for a linear layer this would e.g. be $z_{ij} = w_{ij}x_i$), and $z_j = \sum_i z_{ij}$ represents the pre-activation value of neuron j .

A problem exists with Equation 3.4: If z_j is small, then $R_{i \leftarrow j}^{(l, l+1)}$ will be numerically unstable. To mitigate this issue, a small and positive stabilizer is introduced with a value close to 0:

$$R_{i \leftarrow j}^{(l, l+1)} = \begin{cases} \frac{z_{ij}}{z_j + \epsilon} \cdot R_j^{(l+1)} & z_j \geq 0 \\ \frac{z_{ij}}{z_j - \epsilon} \cdot R_j^{(l+1)} & z_j < 0 \end{cases} \quad (3.5)$$

This propagation rule is referred to as LRP- ϵ . Note that utilising this rule for stabilization leads to the breaking of the conservation rule since some of the relevance will be absorbed by ϵ .

An alternative stabilization approach is the LRP- $\alpha\beta$ rule. This approach manages positive (controlled by α) and negative (controlled by β) pre-activations separately. Using the constraint $\alpha + \beta = 1$ to satisfy the conservation rule (since $z_j = z_j^- + z_j^+$), the propagation rule is given as follows:

$$R_{i \leftarrow j}^{(l, l+1)} = R_j^{(l+1)} \cdot \left(\alpha \cdot \frac{z_{ij}^+}{z_j^+} + \beta \cdot \frac{z_{ij}^-}{z_j^-} \right) \quad (3.6)$$

where

$$z_{ij}^+ = \begin{cases} z_{ij}, & z_{ij} \geq 0 \\ 0, & \text{else} \end{cases}, \quad z_{ij}^- = \begin{cases} 0, & \text{else} \\ z_{ij}, & z_{ij} < 0 \end{cases}, \quad z_j^\pm = \sum_i z_{ij}^\pm.$$

Often a small stabiliser is also introduced for this rule.

3.3.3 Best practice

It has been observed that applying a single rule for all layers in a model often yields suboptimal explanations of model behaviour [83]. It has been observed

that LRP-0 and LRP- ϵ work well on shallow networks [63], [114], but struggle on deeper ones [5]. LRP- $\alpha\beta$ on the other hand, appears to do well but yields similar explanations independent of class [42], [83].

Recent implementations have therefore switched over to using a composite of the rules [64], [65], [83], resulting in closer to optimal explanations. Common guidelines ([59], [83]) recommend the use of LRP- ϵ for the dense layers near the output of models with small epsilon ($\epsilon \ll 1$), followed by LRP- $\alpha\beta$ with $\alpha \in \{1, 2\}$ for the convolutional layers. Some implementations also use a different rule for the first layer only [9], [65], usually referred to as LRP-flat (LRP-0 with $z_{ij} = 1$), that results in more granular heatmaps as the relevance is distributed equally to the input.

/4

Clustering

In this chapter, we present clustering and provide a high-level overview of several popular directions within clustering, both classic and deep ones. Finally, we present spectral clustering, the clustering algorithm which is considered in Chapter 6.

4.1 Uncovering underlying structures in the data

Clustering is a fundamental technique in unsupervised machine learning that is utilised to uncover underlying patterns and structures within data [3]. The main goal of clustering is to divide the available data points into different clusters (groups). Unlike supervised learning, where labels are used, clustering operates without labels relying purely on the structure of the dataset. There are various definitions of clustering that exist, however, the core concept is usually the same: Data points in the same cluster should have high similarity and be dissimilar to data points in other clusters [122].

The foundation of clustering heavily relies on the idea of distance (dissimilarity) and similarity measures. These measures quantify the relationships between data points, forming the basis for grouping them into meaningful clusters. An

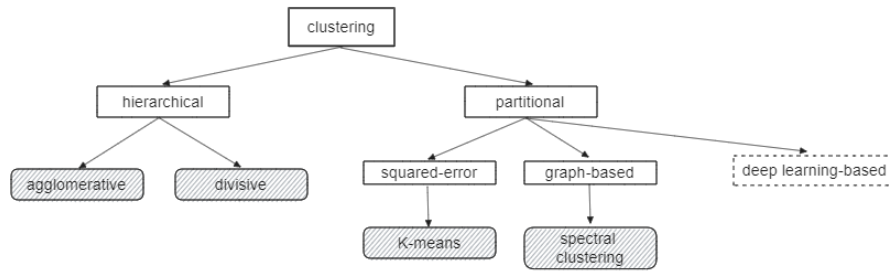


Figure 4.1: Overview of clustering methods described. The visualisation is based on [51].

example of a distance metric is the *Minkowski distance*

$$d_p(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^l |x_i - y_i|^p \right)^{1/p} \quad (4.1)$$

where for $p = 1$ we get the Manhattan distance, and for $p = 2$ we get the Euclidean distance. An example of a similarity metric is the *Pearson's correlation coefficient* [57]

$$r_{\text{Pearson}}(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x}_d^T \mathbf{y}_d}{\|\mathbf{x}_d\| \|\mathbf{y}_d\|} \quad (4.2)$$

where $\mathbf{x}_d = [x_1 - \bar{x}, \dots, x_l - \bar{x}]^T$ and $\bar{x} = \frac{1}{l} \sum_{i=1}^l x_i$. Note that this metric takes the value -1 and 1 when the vectors are perfectly negatively and positively correlated respectively, and values close to zero when the vectors are not correlated.

4.2 Clustering algorithms

Throughout the years a large variety of clustering algorithms have been proposed, each with its strengths and weaknesses depending on the data and the desired use case. Our purpose is not to give a thorough survey of these, but rather to look at some core and classic algorithms (shown in Figure 4.1). For a more complete survey of clustering approaches the interested reader is referred to [51], [122], [128].

4.2.1 Hierarchical clustering

Hierarchical clustering [116] is a valuable technique as it both clusters and organises the data in an intuitive and hierarchical manner. It is usually divided into two different approaches: agglomerative (bottom-up) and divisive (top-down). In agglomerative, each data point starts as an individual cluster. As we move up the hierarchy we merge the closest clusters into a new cluster - a process that is repeated iteratively until a single cluster remains. On the other hand, divisive clustering begins with all data points in one cluster, which is then successively divided into subclusters as we move down the hierarchy. The result of hierarchical clustering is a tree-like structure, a dendrogram, that visualises the complex relationships between all the data points. Note that this structure allows us to easily change the number of desired clusters, without needing to recompute the clustering.

The choices that are most influential for the structure of the dendrogram are the choice of distance metric between two data points, and the strategy for comparing two clusters that consist of several data points. There are several strategies, usually referred to as linkage methods, for comparing 2 clusters. Some of these are:

- **Single-link:** The smallest distance between clusters. Finds the closest pair of points (one from each cluster), and uses the distance between them as the distance measure between the clusters. This link type often results in elongated clusters.
- **Average-link:** The average distance between clusters. Find the pairwise distance between all points (one from each cluster), and use the average of this as the distance between the clusters. This link type often leads to compact and evenly distributed clusters.
- **Complete-link:** The largest distance between clusters. Similar to single-link but use the pair of points (one from each cluster) that are farthest apart as the cluster distance. This link-type results in spherical clusters.

4.2.2 K-means

K-means is one of the most popular clustering methods, as it is simple, fast, efficient and easily interpretable [43]. In K-means, each cluster is characterised by its centre (the centroid), and data points are assigned to the cluster whose centroid is closest. Because of this definition, which is essentially to minimise the within-cluster distance (or the squared-error in regards to the centroids), the clusters found by K-means are spherical in nature.

The standard algorithm is quite simple: given a number of clusters N , first make some initial guess for the cluster centroids, this could e.g. be N randomly chosen data points from the dataset [31]. Step 1) is to assign data points to the centroid they are closest to in Euclidean distance, and Step 2) is to recalculate the cluster centroid based on the currently estimated cluster arrangement. These two steps are repeated until some criteria are met, e.g. convergence or a maximum number of repeats has been done.

4.2.3 Deep clustering

Deep clustering involves the application of deep learning techniques to learn and identify meaningful clusters in the data [21]. While some approaches combine classical clustering methods with deep learning, others utilise solely deep learning techniques to both learn representation and cluster [128]. Utilising deep learning allows us to discover more complex patterns and to better explore high-dimensional data, compared to classical clustering methods [113]. If applied correctly, deep clustering can help boost and improve performance compared to classical methods.

As it is outside the scope of the thesis we will not focus on various deep clustering methods, but rather just acknowledge their existence and potential usefulness. We leave the exploration of whether these methods can be utilised with the work presented in this thesis as future work.

4.2.4 Spectral clustering

Spectral clustering is a graph-based technique that significantly stands out from methods such as K-means where the clusters are usually forced to have a certain shape. Clusters found by spectral clustering can have any shape, as the main goal of the method is to conserve the local structures (often denoted as neighbourhoods) present in the data.

Core procedure

There exists a variety of ways of doing spectral clustering. Below we present the common procedure used by many of these methods:

1. Create a symmetric similarity matrix: $S = \{s_{ij}\}$ where s_{ij} denotes the similarity between data points x_i and x_j .
2. Construct the Laplacian matrix: $L = D - S$ where D is the diagonal degree

matrix with elements $d_{ii} = \sum_j s_{ij}$.

3. create the normalized graph Laplacian matrix $L^* = D^{-1/2}LD^{-1/2}$, which is symmetric and semi-definite. This is a useful transformation as this matrix has nonnegative eigenvalues and orthogonal eigenvectors.
4. Perform eigenvalue decomposition: $L^*\boldsymbol{v} = \lambda\boldsymbol{v}$, to acquire the pairs $\{(\lambda_1, \boldsymbol{v}_1), (\lambda_2, \boldsymbol{v}_2), \dots\}$ where $\lambda_1 \leq \lambda_2 \leq \dots$. Not all pairs need to be found just the k ones with the smallest eigenvalues. This is for when we cluster into k clusters.

Different implementations use different similarity measures. A commonly used approach is the Gaussian kernel with Euclidean distance, $s_{ij} = \exp(-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2/2\sigma^2)$ for $i \neq j$ and otherwise 0. A different approach that may lead to a sparser similarity matrix is to utilise the graph of the k -nearest neighbours W with elements $w_{ij} = 1$ if \boldsymbol{x}_j is one of the k -nearest neighbours of \boldsymbol{x}_i . This matrix can then be made into the symmetric similarity matrix like so: $S = \frac{1}{2}(W + W^T)$.

Assigning cluster labels

In one common formulation of spectral clustering (SM [105], or discretization method), we can imagine clustering being done by finding a way of cutting the graph in 2 in such a way that the sum of the edges that need to be removed is the minimum. This (semi-)optimal cut is performed as such: let each element of the next smallest eigenvector represent a data point, use thresholding on these values, and assign elements larger than the threshold to one cluster and the rest to the other. This method can be applied recursively using larger eigenvectors, to find the k clusters we are after.

In a different formulation (NJW [88]), we utilise the core procedure to acquire some of the smallest eigenvectors that we then use to project the data down into a lower dimensional space. This transformation preserves local neighbourhoods allowing traditional clustering algorithms such as K-means to perform quite well. The clusters found by K-means in the lower dimensional space are utilised as the spectral clustering results in the original space.

A recent alternative to the mentioned approaches, known as cluster-QR [22], is to extract the cluster assignments from the eigenvectors using column-pivoted QR-factorization [29]. This method has no tuning parameters and runs no iterations, yet may outperform both the K-means and the discretization approach in terms of both quality and speed.

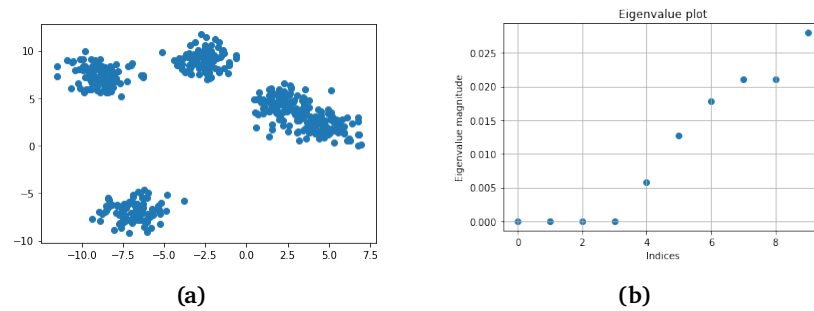


Figure 4.2: Here (a) shows an instance of a Blob dataset with 5 blobs, and (b) shows the 10 smallest eigenvalues of the Laplacian matrix. There is a gap in the eigenvalues after the fourth one, which suggests 4 clusters. However, note that the largest eigengap is after the fifth eigenvalue. This reflects our prior knowledge of the data that there are 5 blobs.

4.2.5 Deciding the number of clusters

Choosing the number of clusters k for spectral clustering is not a trivial task, similar to other clustering methods. Various approaches have been proposed, but perhaps the most popular is the *eigengap* heuristic [118]. This heuristic looks for the largest gap between the consecutive smallest eigenvalues, using the number of eigenvalues before the gap as the estimate for the number of clusters k . An intuition behind this heuristic in the graph-cutting mindset is that the eigenvalues represent the amount of connectivity lost in the graph by cutting it. Hence a significant drop in connectivity is a good place to stop partitioning the data. See Figure 4.2 for an example of this heuristic in action.

/5

Shortcuts in deep learning

Here, we present an introduction to shortcut learning, and its effect on the field. We will look at what shortcuts are through a simple example, see that shortcut learning seems to be a common feature present in all learning systems, and then try to understand where they come from. Next, we will give some more examples of shortcut learning in computer vision tasks, before finally going into group robustness, an approach to mitigate shortcuts. Note that large parts of this chapter were first presented in my project thesis [110].

5.1 What are shortcuts?

Shortcuts, sometimes also referred to as spurious correlations or superficial cues, are often seen as unintended ways of solving a task that makes the model perform well on specific tests but not general ones. Some feature in the data is spuriously correlated to the concept we are actually trying to learn, and the model learns to take advantage of this correlation, perhaps because it is easier (a shortcut) than to learn the core feature we intend the model to find [35].

Shortcut learning thereby usually materialises in the form of the model learning the wrong decision rule. This concept may be referred to as *generalisation failure* which does not mean failure to generalise, but instead generalisation in an unintended direction [35].

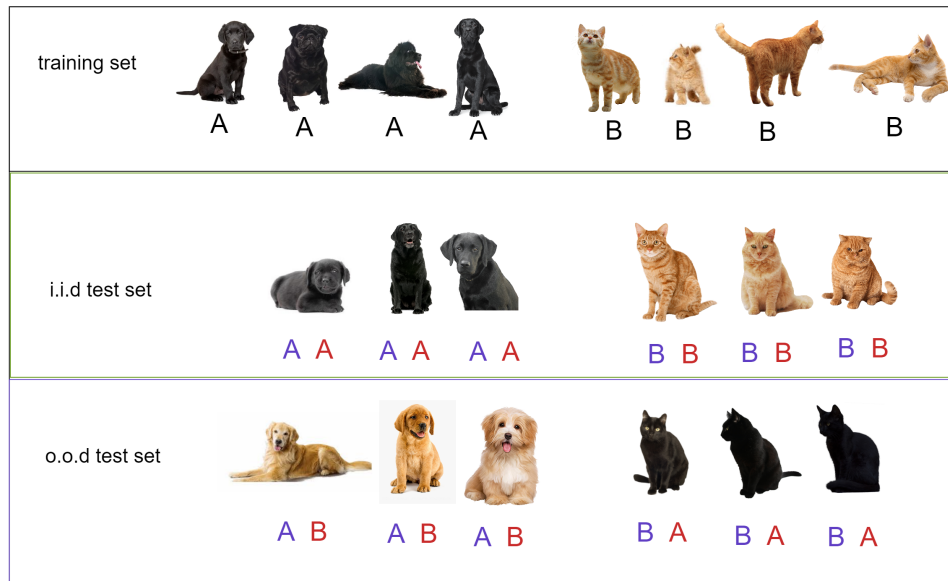


Figure 5.1: Simple shortcut learning example. Classification task of cats and dogs with the provided training set and labels (black font). For the test sets **blue font** indicates the label a human would give, while **red font** is the label given by a model trained on the given training set. The i.i.d test set contains similar examples to the training set, while the o.o.d test set contains different examples. Testing on different data shows that the model uses a different decision rule than expected.

5.2 A simple example

Perhaps it's easy to understand this process through a simple example. In Figure 5.1 we provide a simple example of shortcut learning for a deep neural network. The task concerns classifying images of cats and dogs given a limited training set. When humans and the model classify images from a test set that has similar images to the training set (i.i.d.), they behave the same. However when the test data is different from the training set (o.o.d.), the human and model behaviour is different. Had we only used the i.i.d. test it would have seemed like the model learned the intended strategy, classifying cats and dogs, however by also testing on o.o.d. data we reveal that the model learned a far easier strategy, to classify based on colour.

In this simple example, it is straightforward to notice what the shortcut feature is, but this is not usually the case. The shortcut features tend to be much more deceptive, and the dataset and problem are much larger and more complex, leading to the shortcuts going unnoticed.

5.3 Shortcuts are present in all learning systems

In the previous section, it seems perhaps obvious to us what the model should have done, and what the "intended" solution should be. The model however, knows nothing about what the intended solution should be, it lacks the human bias we have. Having said that, it is not just artificial learning systems that make use of shortcuts, as there are countless examples of even humans making use of shortcuts. Think of how many students use memorisation to pass tests, instead of taking time to actually learn and understand the subject. The goal of the tests is to get a measure of how much the students understand the subject, but memorisation has proven to be an effective shortcut strategy to do well on the tests but has fewer benefits for the students in the long run.

A well-known example of shortcut learning in the animal world is the Clever Hans example [47]. Hans was a horse from the early 1900s who was famous for his intellectual abilities and was able to do things like simple arithmetic, tell the time, and remember peoples' names. The usual process would be that Hans' handler would ask him a question, which Hans would answer by tapping his hooves a certain number of times or in a certain way (the handler had taught him a simple hoof language to communicate). It turned out however, that Hans was not as intelligent as it seemed, or perhaps it is better to say, that he was not intelligent in the way people thought he was intelligent. Through various experiments, it was shown that while answering questions Hans' handler, or others around him, were giving him cues as to what they wanted him to answer. For example, if asked to add numbers Hans would continue to tap his hoof until he noticed subtle visual cues in his handler's body language or face, which indicated that it was time to stop tapping his hoof. Hans was not able to answer correctly to questions if the people asking were hidden away, or if they did not know the correct answer [90].

These are just a few examples showing that biological learning systems also utilise shortcuts during learning. The examples shown so far indicate that shortcuts seem to be prevalent in both biological and modern deep learning systems and that it is something we must be aware of during the learning processes. The last example also demonstrates that shortcuts are not always necessarily bad. Hans was not displaying the abilities he was thought to have, but his actual ability of noticing near imperceptible, and often involuntary, changes in people is quite an impressive ability.

5.4 The origin of shortcuts

Some ([35]) argue that learning systems follow the so-called "principle of least effort", where they learn what is easiest to achieve their goal. The problem is that there are often easier ways available apart from the intended way to achieve the goal. Learning systems making use of these easier unintended ways is what we characterise as shortcut learning. But what exactly makes one way easier than another? What creates these easier ways? We will refer to these easier ways as shortcut opportunities from here on out.

5.4.1 Ambiguity and lack of constraints

One reason why shortcut opportunities might be created is due to our inability to accurately enough formulate and constrain the problem we wish to solve. An example of this is shown in Figure 5.1 where the task is not accurately enough formulated, and the solution found by the model could objectively be seen as a valid solution.

Across deep learning, problems are formulated as the minimisation of loss functions. This is an ambiguous way of framing a problem allowing for possible shortcut opportunities because there is no guarantee that the model will actually learn what we intend it to learn. However, we are required to continue to use loss functions since we are unable to formulate the highly complex problems we wish to solve in a more precise manner. We might be able to introduce changes to the currently used loss functions to improve their behaviour. However, it seems unlikely that we will be able to completely get rid of the ambiguities. Therefore it might be that there is no absolute solution to shortcut learning.

5.4.2 Biases

Different types of biases may create shortcut opportunities when solving a problem. There may be some dataset bias, some correlations or imbalance present in the data that is taken advantage of. The act of simply scaling up the dataset might not be sufficient to get rid of the shortcut opportunities. Shortcut opportunities may be created by some sort of systematic bias/error from data collection or data annotation. Lastly, we mention inductive bias, which is the collection of all the assumptions incorporated by a model such as architecture, training data, loss function and optimisation (see many examples of such assumptions in Chapter 2). It is clear that the selection of these, and the assumptions made in their selection will have a great effect on the types of shortcut opportunities available for models to find and rely on. Great work has

been done in regards to try and minimise the shortcut opportunities created by these biases (see Section 5.6).

5.5 Shortcuts in computer vision tasks

There are many examples of shortcut learning all across deep learning [16], [23], [68]. However, we will here focus on computer vision-related examples since these are the most relevant for this thesis. In the following subsections, we present some examples of shortcut learning where the solution found by the models deviated from the expectations.

5.5.1 Artefacts

Sometimes models learn to take advantage of other features or objects than what we want them to find. This is often what is thought of as shortcuts, and the field of explainable AI has been able to uncover several such shortcuts as we will see below.

In the paper "Unmasking Clever Hans predictors and assessing what machines really learn" [65] by Lapuschkin et. al, the authors show an example of an image classification task where a majority of the images belonging to the class 'horse' contained a watermark in the bottom left corner. Through their investigation (see Figure 5.2) they showed that the model learned this spurious correlation and that if the watermark was inserted on a different image, the model would still classify it as 'horse'. The same paper also discovered that even a synthetic artefact like padding, which is used all across computer vision tasks, can be used by a model as a shortcut feature.

Another example is the classification of pneumonia, where the dataset contains x-rays of healthy and sick patients' chests. Some models learn to differentiate between images from different hospitals, by looking for identification tags, or by telling how different hospitals process their images, and use that information for classification [125]. Some hospitals may have contributed with many positive, or negative cases, so simply classifying based on the hospital is a viable and easier method to perform well on tests. This is obviously not what we wish the model to learn.

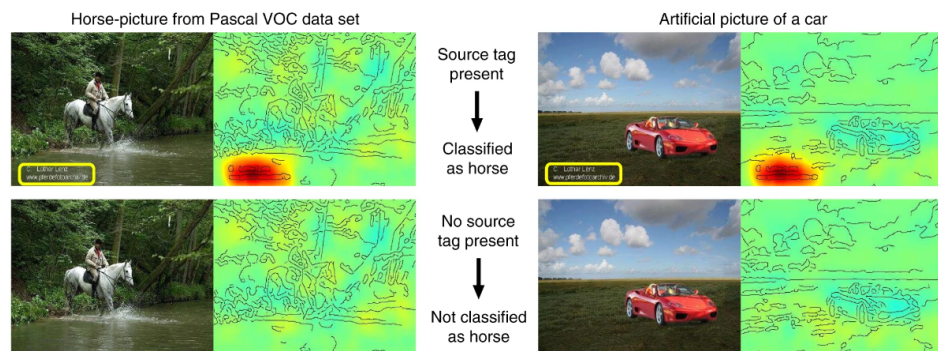


Figure 5.2: Shortcut learning example from [65]. Shows that the model has learned to rely on the presence of a watermark to classify an image as class 'horse'.

5.5.2 Context and background reliance

There are many examples where models have been shown to rely on the background or context of images to make predictions. This may lead to strange results, like the one shown in Figure 5.3, where a model is asked to create a caption for the image. The model has picked up a spurious correlation between 'greenhill' and 'sheep', and imagines the presence of sheep in the image.[104]

5.5.3 Texture-shape cue conflicts

In the paper [36] a dataset (GST) is introduced which contains texture-shape cue conflict, meaning that the shape and texture of the images come from different classes of objects (see figure Figure 5.4). The point of this dataset was to study if humans and ImageNet-trained CNN models are shape or texture biased by allowing them to classify these images. If the shape is more often predicted correctly, one is called shape-biased, while if the texture is more correctly classified one is called texture-biased. Using this dataset it was shown that humans are shape-biased, while CNNs are texture biased, which went against previous assumptions that CNNs worked similarly to the human visual system. Thereby this can be seen as a shortcut.

5.6 Mitigation of shortcuts

Numerous works ([34]–[36], [67], [80],...) have been put forward to try to mitigate the various shortcuts presented in Section 5.5. One such work is the promising new approach of *group robustness* [99].



A herd of sheep grazing on a lush green hillside
 Tags: grazing, sheep, mountain, cattle, horse

Figure 5.3: Example of background reliance in caption generation task. The presence of a green hillside is spuriously correlated with the class sheep.[104]

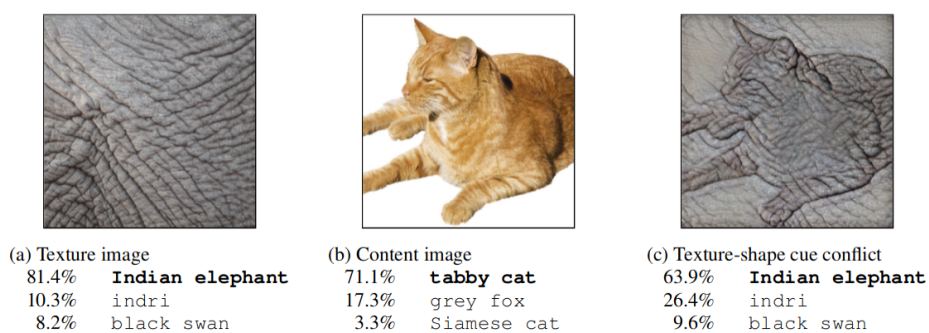


Figure 5.4: Model predictions for 3 types of images. (a) shows only texture, (b) shows only the content, and (c) is a combination of the two images. Note how the model still classifies (c) by texture.[36]

5.6.1 Group robustness

Models that rely on learned shortcuts often underperform when those shortcuts are not present [35]. To address this issue, the concept of group robustness has emerged. Group robustness [99] refers to a model's ability to generalise and maintain consistently high performance across different groups of data points [27].

The fundamental principle behind group robustness is simple. First, the dataset is categorized into groups that either include or exclude the shortcut features. Then, using the group information the model is trained to perform well across the groups. By performing the optimization in this way, the model is steered towards learning the core features present across the groups, instead of relying purely on the shortcut features [99].

Group definition

The group robustness setting [99] assumes that the data distribution consists of several groups $g \in \mathcal{G}$ which form a disjoint set. Typically, these groups are defined as a collection of the class label $y \in \mathcal{Y}$ and some spurious attribute label $s \in \mathcal{S}$ - that is $g = (y, s)$. The spurious attribute label operates similarly to the class label, indicating the presence of some feature that may be learned as a shortcut. Examples of such features could be: the presence of some artefact, the background type, gender, or the colour used (see Section 7.1 for examples).

Optimising using groups

Once the groups have been defined and acquired for the different observations $x \in \mathcal{X}$, we need to utilise them to increase group robustness. Here we will present what is done by group DRO [99], the method that kick-started the pursuit of group robustness.

We start off by recapping the standard training (ERM) loss,

$$\mathcal{L}^{ERM}(\theta) = \frac{1}{N} \sum_{i=1}^N l(x_i, y_i; \theta) \quad (5.1)$$

which simply averages the loss from each observation. By minimising this loss we ensure that the average accuracy of the model increases. Note that there is no guarantee that the models trained with this loss will perform similarly across different groupings of this data.

Group DRO [99] is pitched as a method for making performance across different groupings of the data more similar. Its loss function is defined as such,

$$\mathcal{L}^{GDRO}(\theta) = \max_{g \in \mathcal{G}} \frac{1}{N_g} \sum_{i: g_i = g} l(x_i, y_i; \theta) \quad (5.2)$$

where N_g is the number of observations with group $g_i = g$. By continuously optimising with regard to the worst group we ensure that the loss or performance across the groups becomes uniform.

Issues

The main issues for group robustness are 1) defining sensible groups and 2) acquiring group annotations for real-world datasets.

The first issue revolves around defining meaningful groups. To deal with this issue we need to answer several questions, one of them being: "What exactly is the model learning as a shortcut?", something we have seen in earlier sections is not a question we can provide a simple answer to. Another question is "How should we deal with cases where multiple shortcut features exist in a dataset?", as nearly all of the research on group robustness has concentrated on situations featuring a sole shortcut feature. The only exception to this is [69], where it was shown that current group robustness methods do not do well in cases where there are multiple shortcuts, and in fact, a dynamic akin to a whac-a-mole game was observed, where decreased reliance on one shortcut feature amplified the reliance on a different shortcut feature.

The second issue, assuming some sensible group definition was found, is the non-trivial task of creating the group labels for the dataset which is required by nearly all currently proposed group robustness methods. This is a task that costs both time and money, and that may require special knowledge or the use of a specialist. Labelling large datasets is already a momentous task, and now needing to label other features as well may prove difficult or even impossible.

Although group DRO requires that the entire dataset is group labelled, several recent works have proposed methods that require orders of magnitude less group labels [58], [71], [86], [87]. One proposed method [111] also improves on group robustness without needing group labels, although its performance is lacking compared to other methods.

There is therefore a need for a group robustness method that is group-unsupervised and that performs comparably to methods that utilise group labels.

Part II

Method

/6

Proposed method

The goal of this chapter is to present, Explainability-based feature reweighting (XFR), our proposed group-unsupervised method for improving group robustness. We start off by introducing the core ideas of our proposed method, before diving into its two main components: *unsupervised group discovery* and *group balanced training*. We then present two variations of Explainability-based feature reweighting (XFR) differentiated by how the unsupervised group discovery is performed. Lastly, we put forward an instance of our approach that uses SPRAY [65] and DFR [58] as its components.

6.1 Explainability-based feature reweighting (XFR)

In Subsection 5.6.1 we brought to light several issues that currently plague group robustness methods, these issues mainly being the definition of meaningful groups and the acquisition of group labels using these definitions.

As a possible solution to these issues, we hypothesise that explainabilities can help uncover a model's underlying strategies for processing different groups. More specifically, that groupings of similar explanations may serve as proxies to the group labels, and that these may then be utilised to improve group robustness without the use of group labels.

Building upon these ideas we propose our approach, **Explainability-based feature reweighting (XFR)**, which leverages explainabilities to improve group robustness without the use of group labels.

Similar to existing group robustness methods [71], [86], [87], [111], our approach leverages a two-step approach: 1) *unsupervised group discovery*, where we estimate pseudo-labels for the groups, and 2) *group balanced training*, where we optimise for group robustness using the acquired pseudo-labels.

In the following subsections, we give an overview of the two main components of XFR, before presenting the two variants, local-XFR and global-XFR, that do the group discovery in slightly different manners.

6.1.1 Unsupervised Group Discovery

Unsupervised group discovery is the first component of XFR and is performed over two steps. Given a traditionally trained model (ERM) we first find the heatmap explanations for the data. Next, we perform clustering of the heatmaps in a particular way (see Subsection 6.1.3) to find the groupings in the data which represent the underlying strategies.

A large number of possible configurations exist for this component of XFR due to its general nature. One could in principle use any explanation method that yields a heatmap with it, and also any type of clustering algorithm. This generality makes this component of XFR quite flexible.

6.1.2 Group Balanced Training

The second component of XFR is to utilise the group labels found by the first component to train a group robust model. In principle, any existing group robustness method that uses group labels can be used for this component.

The choice of the method used here will determine what data we need to estimate group labels for. As an example, group DRO [99], a well-known group robustness method, requires group labels for both the training and validation set. Meanwhile, a method like JTT [71], requires only group labels for the validation set.

6.1.3 Variants

An important question we must answer is: "How should the clustering results be used to create pseudo-labels for groups?". Here we propose two variations of XFR that are differentiated by how this question is answered. These methods are in part based on the definition of groups from Subsection 5.6.1, and attempt to solve the core issue of defining sensible groups.

Some important differences that XFR introduces compared to existing methods that ought to be mentioned, is the allowance of an arbitrary number of groups, and that different classes are not required to have the same number of groups. These properties are easy to introduce by utilising clustering and should ideally allow for more flexibility in capturing multiple shortcuts - a possible limitation holding back existing methods that utilise these assumptions (e.g. methods that use the labels or assume a certain number of groups).

Global-XFR: Clusters as pseudo-labels for the spurious attributes

This first variant of XFR follows the typical group definition, where a group is characterised by the class label and spurious attribute label, $g = (y, s)$. In this definition, all combinations of $y \in \mathcal{Y}$ and $s \in \mathcal{S}$ are possible defining the space $\mathcal{G} = \mathcal{Y} \times \mathcal{S}$. In other words, this means that the same spurious attribute might be present across all classes in the dataset. As we assume access to the class labels, we require a way of estimating the spurious attributes so that these properties are maintained.

A possible way of finding similar attributes across different classes is to perform the clustering of explanations in a global sense, that is, to cluster all explanations of the dataset. The resulting clusters from this process will have the properties we are looking for, and we therefore, assume that they can serve as pseudo-labels for the spurious attributes present in the data.

Ideally using this approach, if a model is using e.g. some artefacts as a shortcut to perform classification, then this artefact ought to be present in the explanation. By performing the clustering as described above, we should find all explanations in the dataset that utilise the artefact. Following this, by creating the groups using the class labels and the clustering labels, we should end up with groups for each class that do and do not contain the artefact. Training a model using these groups should hence, following the reasoning behind group robustness, guide the model to learn the core features present across the groups.

The advantage of this variant of XFR is that it should perform well in cases

where different classes utilise the same spurious attribute, since the global information about this attribute can be used to find and locate observations that have similar explanations. The drawback is the potential difficulty of clustering observations from all classes, especially when the number of classes is large.

Local-XFR: Class-wise clusters as pseudo-labels for groups

The second variant of XFR deviates from the assumption $\mathcal{G} = \mathcal{Y} \times \mathcal{S}$ - that the same spurious attribute exists across classes. For this approach, clustering is performed independently for each class (locally), and the clusters found are used as pseudo-labels for the groups directly. Performing clustering class-wise has the benefit of decreasing the complexities related to clustering all explanations at once.

The reasoning behind L-XFR is the following observation: a model that has learned to utilise shortcut features will perform differently across different groupings within a class; performance on a group containing the shortcut feature will be high, and on the rest, it will be low. Additionally, one ought to expect there to be different explanations for observations that use shortcuts and observations that do not. By optimising over these different groupings, we again should expect the model to learn the core features present across the groups, instead of relying on the shortcut features.

Naturally, the pseudo-groupings derived using this approach are not guaranteed to be similar across classes (unlike in G-XFR), although it is possible. Performing the pseudo-labelling in this manner should be more efficient, and flexible to variations between explanations in a class. In a sense, it can be said that this approach helps with class-specific shortcuts. However, it might struggle in cases where the same spurious attribute is utilised across classes and the global information is helpful for e.g. finding groups with a small number of samples (as we will later see in Figure 8.8).

6.2 Building blocks of XFR for experiments

In the previous section, the conceptual groundwork was laid for XFR. In this section, we will describe conceptually the XFR setup used for our experiments, and in particular, the methods chosen as the building blocks of XFR.

To do the unsupervised group discovery we leverage the methodology proposed by Spectral Relevance Analysis (SPRAY) [65] where spectral clustering is used

to cluster LRP heatmaps. This method is chosen because it has already proven its capability to find clusters that contain different strategies (also shortcut strategies). Thus we do not need (for now) to explore different combinations of explainability methods and clustering methods.

To do the group balanced training we will utilise Deep feature reweighting (DFR) [58], which performs retraining of the last layer using the validation set. We choose this method because it requires only pseudo-labels for the validation set, and it is fast and efficient while still performing well. We note that in principle, any group robustness method can be used.

6.2.1 Spectral Relevance Analysis

Spectral Relevance Analysis (SPRAY) [65] is an explanation technique that allows for efficient investigation of classifier behaviour on large datasets. SPRAY applies spectral clustering on a dataset of LRP explanations to identify the decision rules of a model and presents these results in a concise and interpretable manner. The results can then be inspected to uncover shortcuts present in the dataset that the model has learned to utilise.

To perform SPRAY we require a dataset, a model trained on this dataset, and some data we wish to analyse (can also be the dataset or some subset of it). Once these are acquired, SPRAY is performed through these 5 steps, where the fifth is an optional step. Step 1 consists of acquiring the relevance maps for the data we wish to analyse. The relevance maps are LRP heatmaps that show which part of the data the model sees as important for its decision. Step 2 consists of downsizing the heatmaps and making them uniform in shape and size to speed up the following steps. Step 3 is to do spectral clustering of the heatmaps, which uses structures in the heatmaps to find clusters. Step 4 is to identify interesting clusters by using the eigengap heuristic. This allows us to select the best number of clusters so that we are left with the most separable ones. Thus ideally if done correctly, each cluster should contain a single decision rule that should be distinguishable from that present in other clusters. Step 5, which is optional, is to visualise the clustering using t-Stochastic Neighbourhood Embedding (t-SNE) to show how SPRAY is working.

6.2.2 Deep feature reweighting

Deep feature reweighting (DFR) [58], is a group robustness method that revolves around retraining the last layer of a network using a group-balanced set (often the validation set). The authors of the paper that proposed DFR, Kirichenko et al., argue that the feature extractor trained with ERM learns both core and shortcut features but that the shortcut features are overly weighted

in the last layer. By doing last layer retraining on a balanced set they show that the model can learn to use the core features, resulting in improved worst group accuracy sometimes rivalling oracle methods such as group DRO [99].

Assume we have a dataset $\mathcal{D} = \{x_i, y_i\}$, and another (usually smaller) dataset $\hat{\mathcal{D}}$ where the groups are balanced. We will refer to \mathcal{D} as the training dataset and $\hat{\mathcal{D}}$ as the reweighting dataset. Note here that the reweighting dataset can be a subset of the training dataset or some different dataset. Bear in mind, that when we say that retraining is done on e.g. the validation set, we do not mean that the whole set is used, but rather that a group-balanced version of it is. The group balancing is performed by subsampling all groups to be the size of the smallest group.

To do DFR we first need a base model. If we do not already have a model, we train one in the usual way (ERM) using the training dataset. Assuming our model is made up of a feature extractor and a final classification layer, we proceed by keeping the feature extractor and training a new classification layer using the reweighting dataset. Finally, we use this new model to classify the test data.

6.2.3 XFR setup

Since DFR is used, we will need pseudo-group labels for the validation set. We start off with a model we want to improve, which could be a model that was trained with standard Empirical risk minimisation (ERM) on the training set. Images are then made to the same size. Next, we acquire the LRP heatmaps for all the validation set images. Then, we perform local and global clustering using spectral clustering. Here we leverage the eigengap heuristic to determine the number of clusters. After this step, the pseudo-labels for the groups are acquired and DFR can be run.

Note that we partly do not utilise "step 2" in spray, since allowing downsizing introduces a parameter, which might need to be optimized. We have not mentioned "step 5" here, but it will be used later to analyse the clusters.

Part III

Experiments



Experimental setup

In this chapter, we present the overall experimental setup used. First, the group robustness datasets are presented, followed by the metrics we will use in the experiments. Next, the models used, and their implementation is presented.

7.1 Group robustness datasets

There are 3 datasets used in this thesis. The first is a synthetic dataset based on the MNIST dataset [25]. The second and third are natural image-based datasets that are commonly used within group robustness. Each dataset has three splits for training, validation and testing. The amount of images for each split, along with example images, is given in Table 7.1, Table 7.2 and Table 7.3 for each dataset.

The general setup for the datasets used in this thesis is that each of the datasets has 2 classes given by $y = \{0, 1\}$ and 2 spurious attributes $s = \{0, 1\}$. Combining the classes and the spurious attributes we end up with 4 groups $g = \{0, 1, 2, 3\}$. In each dataset the frequency of each group is different, but in all cases, at least one of the groups can be considered a minority group, as they only make up a small fraction of the total. We will henceforth denote the frequency of groups as the group distribution of the data. Note that the group distribution may vary across the dataset splits (see Table 7.2).

Note that the only difference between $g_0 = (y = 0, s = 0)$ and $g_1 = (y = 0, s = 1)$ is the spurious attribute. Our goal is for our models to not be reliant on spurious attributes, hence our goal is for the model to perform equally well across the groups.



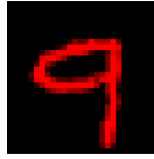

Split	Total Data	Group 0 ($y=0, s=0$)	Group 1 ($y=0, s=1$)	Group 2 ($y=1, s=0$)	Group 3 ($y=1, s=1$)
Train	50,000	254	25,284	24,231	231
Val	10,000	45	5,013	4,893	49
Test	10,000	48	5,091	4,815	46
Examples:					

Table 7.1: Data splits in the Colored-MNIST dataset.

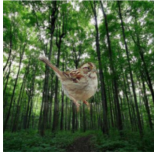



Split	Total Data	Group 0 ($y=0, s=0$)	Group 1 ($y=0, s=1$)	Group 2 ($y=1, s=0$)	Group 3 ($y=1, s=1$)
Train	4,795	3,498	184	56	1,057
Val	1,199	467	466	133	133
Test	5,794	2,255	2,255	642	642
Examples:					

Table 7.2: Data splits in the Waterbirds dataset.

7.1.1 Colored-MNIST

We create a dataset where we have control of the number of elements in each group and what the spurious attribute is.

The Colored-MNIST dataset is a synthetic dataset based on the well-known MNIST [25] dataset. The MNIST dataset is a collection of several thousands of examples of handwritten digits (0-9). The images are single-channelled (black and white) and have a size of 28x28 pixels, and are accompanied by a label giving the ground truth.

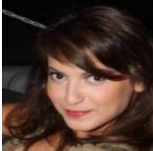



Split	Total Data	Group 0 ($y=0, s=0$)	Group 1 ($y=0, s=1$)	Group 2 ($y=1, s=0$)	Group 3 ($y=1, s=1$)
Train	162,770	71,629	66,874	22,880	1,387
Val	19,867	8,535	8,276	2,874	182
Test	19,962	9,767	7,535	2,480	180
Examples:					

Table 7.3: Data splits in the CelebA dataset.

We use the original data split, 60000 train and 10000 test. Since the original dataset does not have a validation set, we use the last 10000 images of the training set as the validation set.

We convert the dataset into a 2 class problem by modifying the task. This is done by simply going over to classify the numbers as smaller or equal to 4 ($y = 0$: value ≤ 4), and larger than 4 ($y = 1$: value > 4). To create the spurious attributes we make use of colors. Red is used as the first spurious attribute ($s = 0$: RGB = (255, 0, 0)), and green is used as the second spurious attribute ($s = 1$: RGB = (0, 255, 0)). Naturally, the images will need to be made 3-channeled to account for this change.

As we are interested in combating spurious correlations we create the dataset in a way such that there are correlations between the classes and spurious attributes. We use 99% correlation. That means that 99% of images from one class will have the same colour, while the remaining 1% will have the other colour. The amount of correlation was deliberately chosen so that ERM worst group accuracy is low. Table 7.1 shows the number of images in each group for each split, as well as example images from each of the 4 groups.

The goal of creating this dataset was to have control of the number of images in each group, and to be able to set the spurious attributes ourselves. As an unexpected consequence, C-MNIST highlights the strength of G-XFR and its clustering of all data points since the spurious attribute are more easily found by it.

7.1.2 Waterbirds

Waterbirds [99] is a synthetic dataset created with the purpose of testing a model's reliance on background. The dataset consists of RGB images depicting different types of birds on different types of backgrounds. The different types of birds are divided into 2 classes, landbirds ($y = 0$) and waterbirds ($y = 1$). The different backgrounds are also divided into 2 and represent the spurious attributes of this dataset: land background ($s = 0$) and water background ($s = 1$). The group distributions across the different splits are presented in Table 7.2.

The Waterbirds dataset is created by using 2 other datasets, the Caltech-UCSD Birds-200-2011 (CUB) dataset [120] and the Places dataset [127]. The CUB dataset contains images of birds labelled by species and their segmentation masks. To construct the Waterbirds dataset the labels in the CUB dataset are split into 2 groups, where waterbirds are made up of seabirds (albatross, auklet, cormorant, frigatebird, fulmar, gull, jaeger, kittiwake, pelican, puffin, or tern) and waterfowls (gadwall, grebe, mallard, merganser, guillemot, or Pacific loon), while the remaining classes are labelled as landbirds. The birds are cropped using the pixel-level segmentation masks and pasted onto a water background (categories: ocean or natural lake) or land background (categories: bamboo forest or broadleaf forest) from the Places dataset.

The official train-test split of the CUB dataset is used, and 20% of the training set is used to create the validation set. The group distribution for the training set is such that most images (95%) depict bird types with corresponding backgrounds, to represent a distribution that may arise from real-world data. This distribution turns the background into a spurious feature. Take note that there is a distribution shift from the training split to the validation and test splits which are both more balanced, and include many more elements for the minority group. The creators of the dataset argue that they do this to more accurately gauge the performance of the minority groups, something that might be difficult if there are too few examples. They also do this to allow for easier hyperparameter tuning.

7.1.3 CelebA

CelebA here is a reference to a part of the CelebA celebrity face dataset [72] that was introduced by [99] as a group robustness dataset. From the original dataset, the feature *Blond_Hair* is used as the class, meaning that the images are divided into people who are not blonde ($y = 0$) and blonde ($y = 1$). Meanwhile, as a spurious attribute, we use the feature *Male* from the original dataset, which divides into female ($s = 0$) and male ($s = 1$). The official train-val-test split of

the CelebA dataset is used. Note in Table 7.3, that the splits are likely randomly created, which results in equally group-distributed splits. Across all splits the group (blonde, male) is the smallest.

This dataset tests for model reliance on strongly correlated features in a real-world dataset. Observe in Table 7.3 that $g_3 = (y = 1, s = 1)$ which represents blonde males is severely underrepresented compared to the other groups, hence we expect the model to learn gender as a spurious feature for the class blonde.

7.2 Metrics

As in previous group robustness work ([58], [71], [99]), we will utilise the metrics, worst group accuracy (WGA) and mean accuracy, to evaluate group robustness.

The WGA is given by

$$\text{WGA} = \min_g \frac{1}{N_g} \sum_{i:g=g_i} (f(x_i) = y_i) \quad (7.1)$$

the smallest of the group accuracies.

Like previous work [99], we report mean accuracy by weighting the group accuracies according to their prevalence in the training data. This ensures that the mean accuracies are correct in regard to the training distribution, and is therefore not affected by the distribution shift that might be present in the test set (see Table 7.2).

7.3 Models

In this section, we present all models used in the experiment section. We give an overview of what each model does and how they utilise group labels. For the models we run ourselves, we additionally provide some implementation details.

7.3.1 External Methods

These are the group robustness methods whose results were gathered from their respective papers.

Oracle

Group distributionally robust optimisation (**group DRO**) [99] is regarded as an oracle method since it utilises group labels for both the training and validation set. This method uses the training group labels to minimise the worst group loss during training, and tunes its hyperparameters using the validation set group labels. Theoretically, this method should perform best among all methods since it fully exploits the group labels.

Group labels for finetuning

Just Train Twice (JTT) [71] is a 2-stage method. In the first stage, a traditional model is trained for a few epochs on the training set, and then it is evaluated on the training set. In the second stage the misclassifications from the first stage are upsampled, and a new model is trained using the original data and the upsampled data points. Group labels from the validation set are used to tune how early to stop the training of the first model, and to determine how much upsampling to do. The intuition behind Just Train Twice (JTT) is that models that are trained for a short period will learn the easy cases first, and struggle with the hard ones (the minority groups).

Learn from Failure (LfF) [86] is also a 2-stage method. In the first stage, a model is trained using a special loss that intentionally biases it to do well on the easy cases. In the second stage, the performance from both the first and the second stage models are used to automatically upweight difficult examples. Also here the group labels of the validation set are used to tune hyperparameters.

No group labels

GEORGE [111], is an unsupervised method that clusters the output of the feature extractor to estimate pseudo-labels for groups, and then uses **group DRO** on these.

7.3.2 Implemented Methods

Here we present the methods that were implemented by us. The base ERM model is utilised for both **DFR** and **XFR**.

No group labels: base model

Empirical risk minimisation (ERM) is conventional training without group information. In our experiments we have finetuned ResNet-50 [44] models pre-trained on ImageNet-1k [24] on the Waterbirds and CelebA dataset, and trained a ResNet-18 [44] on the C-MNIST dataset.

For ERM training/finetuning, we use default hyperparameters from DFR implementation: learning rate 10^{-3} , weight decay 10^{-4} , and use of augmentation (horizontal flip, and random crop). The only change was the batch size used, which was set to as large a value as possible.

The models are trained for 100 epochs, and the checkpoint that yielded the best mean accuracy is used throughout.

Group labels on validation set

Deep feature reweighting (DFR), retrains the last layer using a group-balanced validation set. The group-balanced validation set is obtained by subsampling the groups to be the same size as the smallest group.

DFR is performed as follows: From a base model, the outputs of the feature extractor (the penultimate layer) are acquired for the entire validation set, and normalised. Retraining is then performed using logistic regression with L1-regularization. The strength of the regularization is chosen among the values $C \in \{1, 0.7, 0.3, 0.1, 0.07, 0.03, 0.01\}$.

For the retraining step, we need to fit some hyperparameters. Therefore we split the retraining step into a finetuning step and an evaluation step.

During the finetuning step, we randomly split the reweighting set in half, and perform a sweep of the hyperparameters while training on the first half and evaluating on the second half. The hyperparameter that yields the best WGA is chosen.

For the evaluation step, the whole reweighting set is used along with the selected hyperparameter to retrain for a final time. This final retrained layer is then evaluated using the test set.

Note that due to the subsampling to create a group-balanced reweighting set, the performance values for DFR may vary a lot. Therefore in the experiments, we rerun the finetuning step 5 times, choosing the overall best hyperparameter across these runs. In the evaluation step we also repeat the subsampling, this time 20 times. The weights for the logistic regression are aggregated over these 20 runs just like in [58], before the model is finally evaluated on the test set.

For our results that utilise last layer retraining we report mean \pm std of 5 runs of the evaluations step (that is, using the same hyperparameter we do the 20 runs 5 times).

Unsupervised group discovery

Here we report the steps taken to perform the first part of XFR, as steps for the second part were given above.

We utilise a composite of LRP rules to get the explainability heatmaps as recommended by [59], [83]. Following their recommendations we use LRP- ϵ for the dense layers near the output of the model with small epsilon ($\epsilon \ll 1$), followed by LRP- $\alpha\beta$ with $\alpha = 2$ for the convolutional layers.

For the spectral clustering, we use the affinity matrix created from the k-nearest neighbours with the estimate of $k = \lfloor \log(N) \rfloor$ [118] where N is the number of points being clustered. As it was explained in Chapter 4 we utilise cluster-QR [22] to perform the clustering.

The eigengap heuristic is applied to the 10 smallest eigenvalues of the Laplacian matrix to select the number of significant clusters to use.

/ 8

Main experiment and analysis

In this chapter, we present the main results of this thesis and analyse the results from our proposed method. Using the datasets in Section 7.1, we present the metrics from Section 7.2, for the models in Section 7.3. These results are compared and discussed. Next, we analyse the behaviour of Explainability-based feature reweighting (XFR) to better understand what it is doing.

8.1 Results

In Table 8.1 our main results are given. Note that for the reported XFR results we use full-sized heatmaps (28x28) for C-MIST, while Waterbirds and CelebA use half-sized heatmaps (112x112) due to memory limitations.

From the main results, we make the following observations:

- ERM has the largest mean accuracy across the board. This is a common phenomenon reported in many places [58], [99], etc.: Some average accuracy must be traded to improve robust accuracy. This is why our main enfaces will be on the WGA.

Methods	Group Info	C-MNIST		Waterbirds		CelebA	
		Train/Val	WGA(%)	mean(%)	WGA(%)	mean(%)	WGA(%)
Group DRO	✓/✓	-	-	91.4	93.5	88.9	92.9
JTT	✗/✓	-	-	86.7	93.3	81.1	88.0
LfF	✗/✓	-	-	78.0	91.2	77.2	85.1
GEORGE	✗/✗	-	-	76.2 \pm 2.0	95.7 \pm 0.6	53.7 \pm 1.3	94.6 \pm 0.2
Base (ERM)	✗/✗	39.6	99.3	76.8	98.1	41.1	95.9
DFR	✗/✓/✓	74.2 \pm 2.1	93.7 \pm 0.2	92.1 \pm 0.2	94.6 \pm 0.1	86.9 \pm 0.4	91.1 \pm 0.1
L-XFR (ours)	✗/✗✗	24.8 \pm 1.2	99.2 \pm 0.0	92.6 \pm 0.3	94.3 \pm 0.3	83.1 \pm 0.2	89.6 \pm 0.4
G-XFR (ours)	✗/✗✗	63.3 \pm 1.0	96.8 \pm 0.2	84.7 \pm 1.0	88.5 \pm 1.3	77.8 \pm 0.9	92.8 \pm 0.0

Table 8.1: Worst group and mean accuracy on the test sets of the different datasets. The Group Info column showcases for each method whether group labels are used for that split of the data (✗= does not use group labels, ✓= uses group labels, ✓✓= validation set group labels is used for training and finetuning, and ✗✗= validation set without group labels is used for training and finetuning). The results for JTT, LfF, and Group DRO were gathered from [71], GEORGE results are from [111], while the ones below the separation line are our own results. For our results that use last layer retraining we report mean \pm std over 5 runs after selecting the hyperparameter.

- XFR improves WGA compared to ERM by a large amount (the exception case will be discussed later). On C-MNIST we see a WGA improvement of approx. 23%, on Waterbirds we see an improvement of 8% or more, and in CelebA we see an improvement of 36% or more. These results show XFR’s viability as a method for improving group robustness.
- XFR vastly outperforms GEORGE [111], the other group-unsupervised method, by improving WGA by 8% or more in the Waterbirds dataset and 24% or more on the CelebA dataset.
- XFR’s performance is comparable with existing methods that use group labels, and in one place even performs better than group DRO.
- XFR has performance similar to or worse than DFR. In cases where XFR is worse, it is still significantly closing the gap between ERM and DFR. This behaviour makes sense since DFR is essentially an oracle version of XFR since it utilises the group labels.

L-XFR performance on C-MNIST highlights a failure case for this variation of the proposed method, as it performs even worse than ERM. We will look more closely at this later, but for now, we can explain this performance from the dataset distribution: the classes have a very high (99%) correlation with the spurious attributes, so clustering purely by class makes it very difficult to capture the minority groups. On the other hand, G-XFR performs quite well as

it can leverage the global information to capture the global spurious attributes (see Figure 8.8).

8.1.1 Explainability improvements from XFR

In Table 8.2, we compare the LRP heatmaps from ERM and G-XFR. As can be seen from the heatmap examples, G-XFR has made the model focus on the core features of the images.

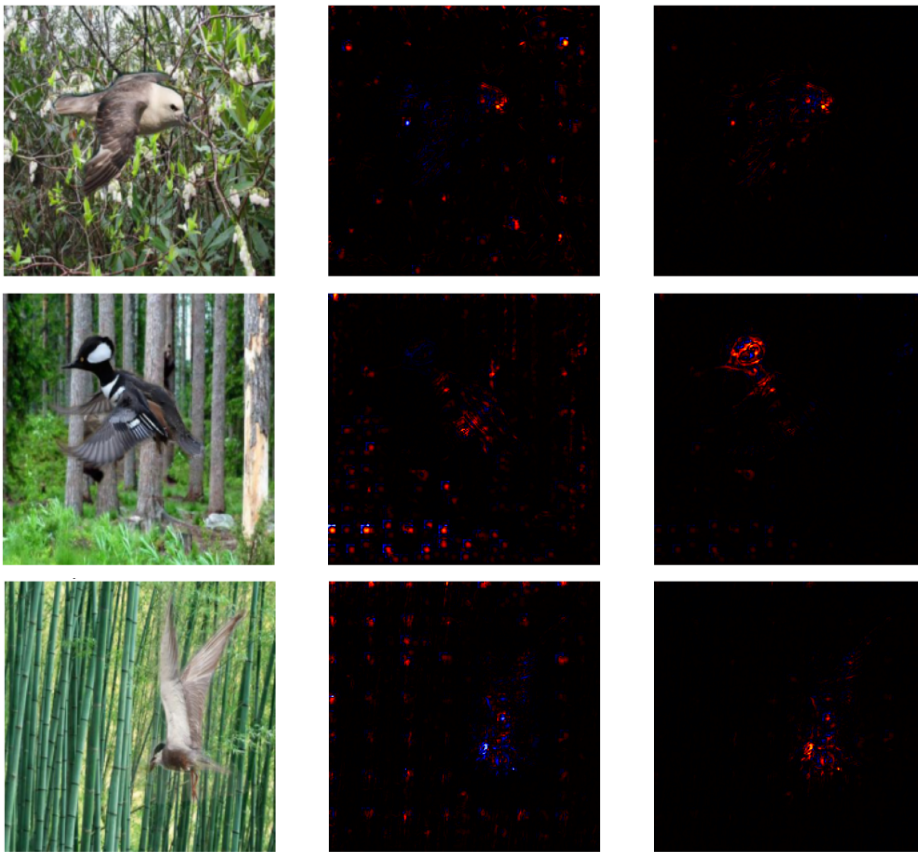


Table 8.2: Examples showcasing the improvements in the explainabilities. Images on the left are the original images, in the middle are the LRP heatmaps for ERM, and on the right are the LRP heatmaps for G-XFR. All images showcase "waterbirds on land background" that were misclassified by ERM, but classified correctly by G-XFR.

8.2 Exploring the group estimation method

In this section, we take a closer look at the group estimation part of XFR.¹ We provide examples of LRP heatmaps and showcase the eigenvalues onto which we apply the eigengap heuristic to estimate the number of clusters. Next, we look at the clusters for CelebA in particular, to figure out what kind of explanations the clusters are capturing. We will find that the clusters can be divided into clusters that focus on the core features and clusters that capture shortcut strategies. We then showcase examples from these two types of clusters and try to give an explanation of what the shortcut features leveraged are.

LRP heatmaps

We utilise the LRP heatmaps with 3 channels, one for each channel in the original image, because it led to better performance during testing than using the aggregated relevance of each pixel. Doing this provides heatmaps that not only show the important regions (positions) for the explanation but also the important colours. In Table 8.3, we show some heatmap examples and the additional information present in the RGB channels.

Eigenvalues and the eigengap heuristic

In Figure 8.1 we present the smallest eigenvalues of the Laplacian matrices in the spectral clustering runs. The eigengap heuristic is applied to these to determine the ideal number of clusters in each setting. For CelebA, 2 clusters are chosen for the clustering of images with class $y = 0$, 9 clusters are chosen for clustering class $y = 1$, and 3 clusters are chosen to cluster all data points. For C-MNIST the numbers selected are 6 for class $y = 0$, 2 for class $y = 1$, and 8 for clustering all points.

What are the clusters capturing?

In Figure 8.2, Figure 8.3 and Figure 8.4 are showcased the average relevances for each of the clusters for CelebA, and the number of data points in each cluster. In all the 3 cases, there is one cluster which contains a majority of the data points and whose relevance appears to focus on various facial features like the eyes and hair region. These clusters seem to be capturing the core features we

1. we will not look too deeply into Waterbirds in this section due to some technical difficulties in acquiring its results. Also because of the difficulty, we will be presenting a different run of CelebA than that showcased in Table 8.1 with slightly degraded performance.

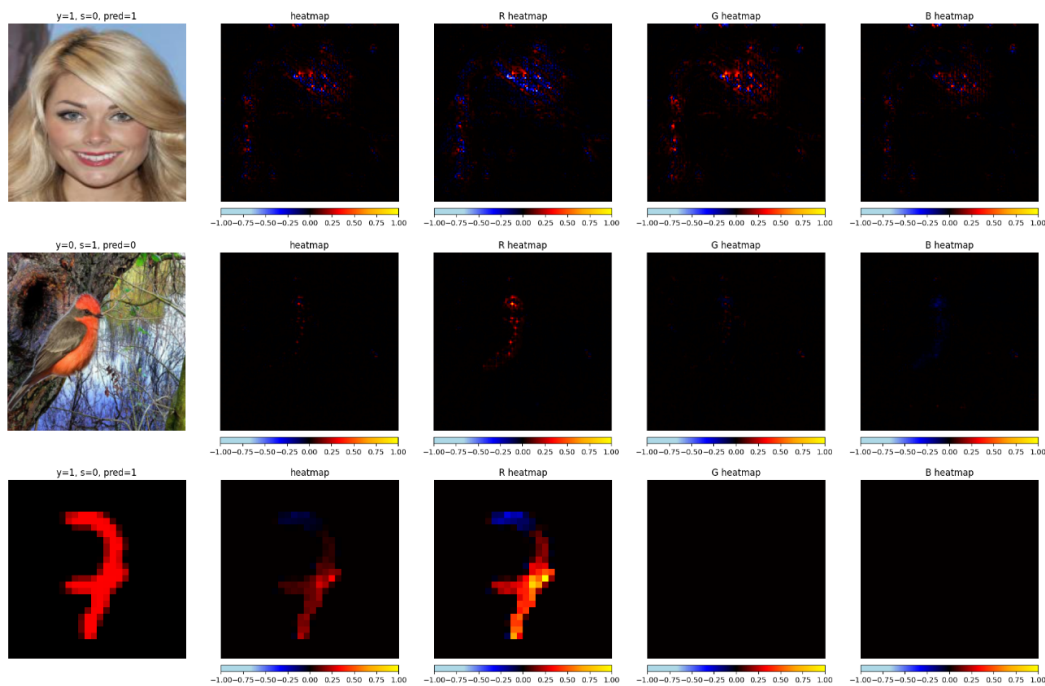


Table 8.3: LRP example from the different datasets. The 1st column showcases the original image, the 2nd column showcases the aggregated LRP heatmaps, and the remaining 3 columns showcase each of the RGB components of the heatmap. This example clearly illustrates the additional information gained by utilising the RGB heatmap.

are after. Meanwhile, the remaining clusters all focus on the top-left corner of the images, most likely a shortcut feature.

To support or claim about the clusters we provide some examples from each of the types. In Table 8.4 are examples of image-and-explanation pairs picked from the largest clusters. As can be clearly seen from these examples, the explanations are pointing at the expected locations in the image, since the relevances are on the hair or face.

Meanwhile, in Table 8.5 we see several image-and-explanation pairs picked from the smaller clusters whose heatmaps light up in the top-left corner only. From the provided examples and our own examination, it seems like the explanations are focused on this corner only in cases where it is a single colour - usually black, and in a few cases white or grey. In most of the examined cases, the single colour seems to be caused by either cropping or extension of the image, or from cutting out the background and replacing it with black (like in the bottom-left example in Table 8.5).

Although we can use the clusters to find the shortcuts utilised, we cannot think of a reasonable explanation as to how using this shortcut can be useful for

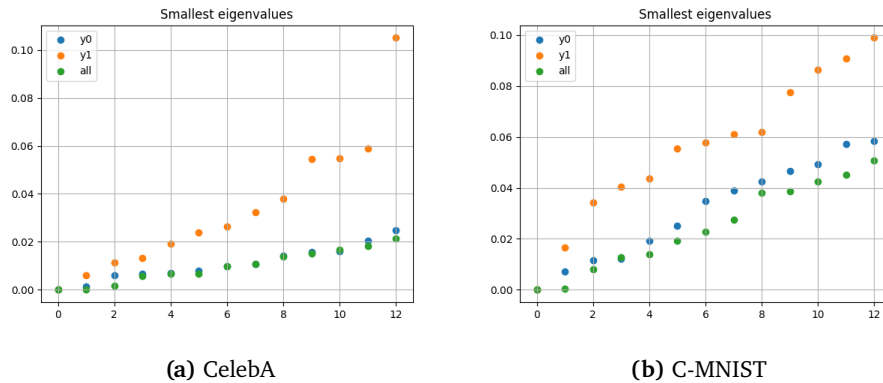


Figure 8.1: The first 12 eigenvalues of the Laplacian matrix when performing the 3 runs of spectral clustering needed to perform L-XFR (y_0 and y_1) and G-XFR (all). The eigengap heuristic uses the biggest gap among the 10 smallest eigenvalues as an estimate for the best number of clusters to use.

doing classification. Perhaps a deeper exploration of the dataset is required for further understanding.

Cluster similarity

We look into if the different types of clusters, clusters that contain core features and clusters that contain shortcut features, can be found using a similarity metric. We utilise Pearson's correlation coefficient (see Equation 4.1) to determine the similarity between the averages of each cluster - an estimate of the actual similarity between the clusters. We present the similarities of CelebA in Figure 8.5, and the similarities of C-MNIST in Figure 8.6. For CelebA we do in fact observe that the last cluster is the least similar to the other clusters, which supports our division of different types of clusters. For C-MNIST we see an equivalent separation only for the global clustering (clusters 5 and 7 are different from the rest). The fact that the similarities of the clusters found by local clustering in C-MNIST are not separable into two types is yet another hint for why L-XFR performs badly in Table 8.1.

Are the clusters capturing the original groups?

To explore if the clusters are capturing the original groups we leverage t-SNE visualisations. As t-SNE and spectral clustering leverage similar concepts, the clusters found by spectral clustering should be reasonably separable in the t-SNE plot. Hence, since it is impossible to cluster the t-SNE projections of the CelebA heatmaps in Figure 8.7 in a way that captures the original group labels,

it means that our group estimation method is not necessarily retrieving the original group labels. An explanation for why our group estimation helps improve group robustness can perhaps be explained by the fact that the clusters we find can be divided into ones that use shortcut and core features.

In the simple case of the C-MNIST dataset, we see that the original groups are separable in the t-SNE projections, which makes it possible for our group estimation method to retrieve the original groups. This is precisely what happens for G-XFR and explains its good performance in Table 8.1.

Exploration summary

In this section, we have seen that when our group estimation method leads to improvements in group robustness it is because either the original minority groups have been retrieved, or because the estimated groups are separable into two types, one type that utilises core features and another that utilises some shortcut feature. We have shown that the existence of the different types of clusters can be found by human inspection, and that it can also be estimated from a similarity measure between the clusters. By leveraging the original labels, we have utilised t-SNE plots to analyse what our proposed group estimation method is doing compared to the ground truth. This has granted us insights regarding the inner workings of our proposed method, and some of its failure modes.

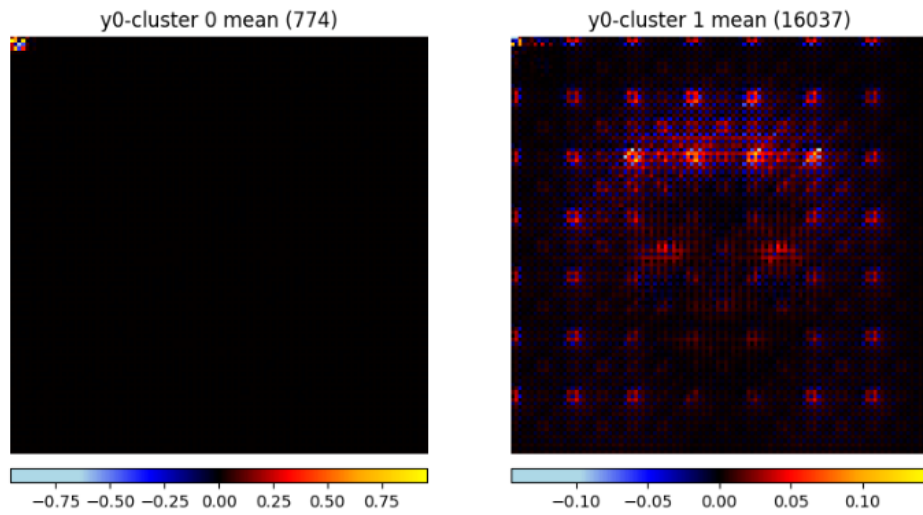


Figure 8.2: The average relevance for each cluster of $y = 0$.

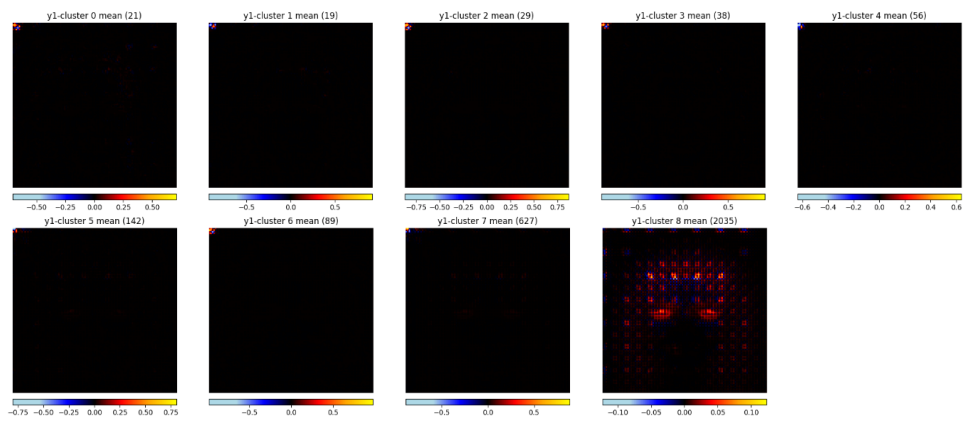


Figure 8.3: The average relevance for each cluster of $y = 1$.

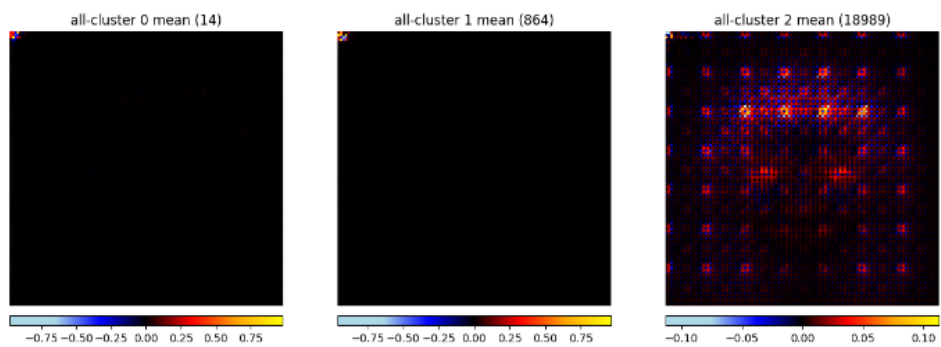


Figure 8.4: The average relevance for each cluster of all observations.

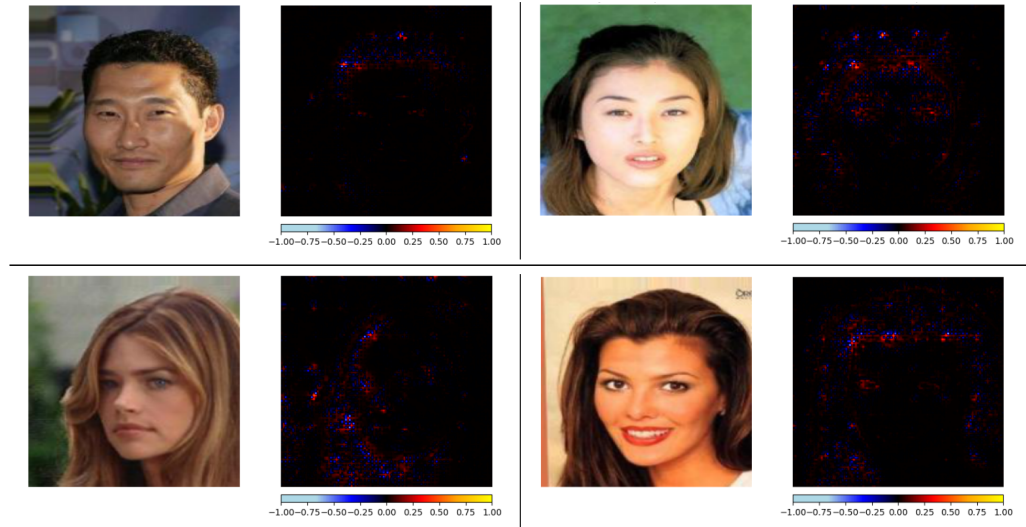


Table 8.4: Image and heatmap examples from largest clusters. For these images, the explanations are focused on the core features needed to perform the classification task.

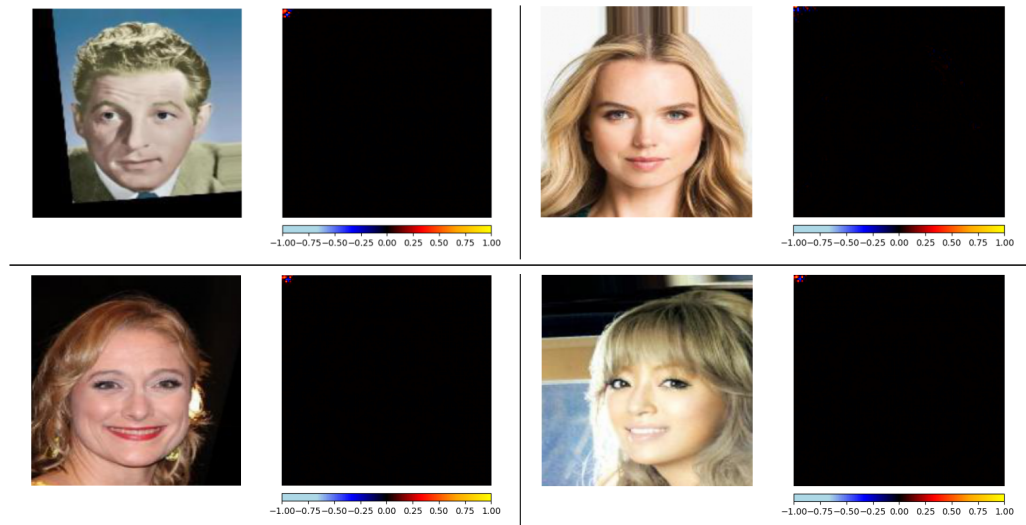


Table 8.5: Image and heatmap examples from the smaller clusters. The explanations for these images all focus on the top-left corner of the image. The potential shortcut feature that the model is utilising in the corner could be the presence of a single colour there (in most cases the colour is black). The single colour might come from segmentation of the original image, or from cropping or extending the original image.

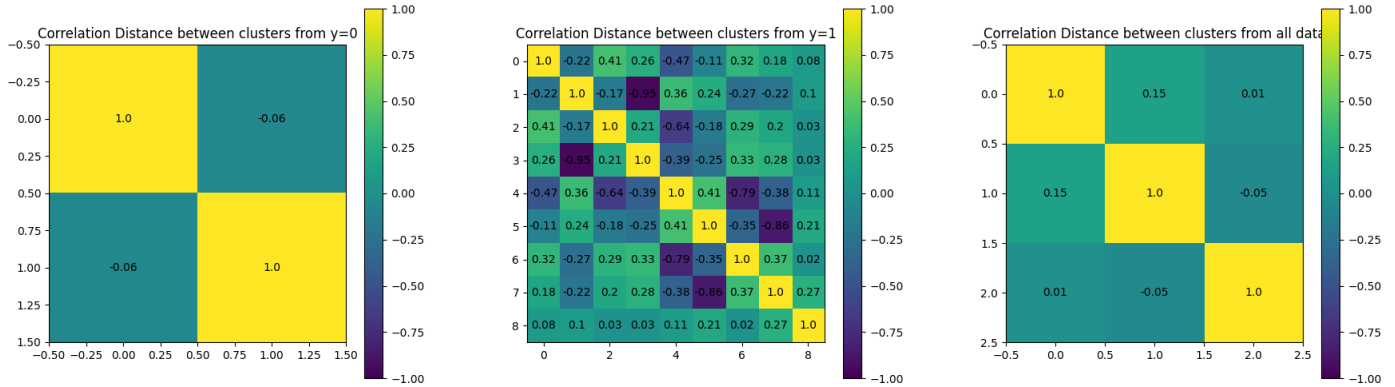


Figure 8.5: Similarities using Pearson's correlation coefficient (see Equation 4.1), between the average heatmaps from each cluster in the 3 clustering runs of CelebA. A reminder that -1 or 1 indicates a complete negative or positive correlation, while 0 indicates no correlation. Values close to 0 thereby indicate little similarity.

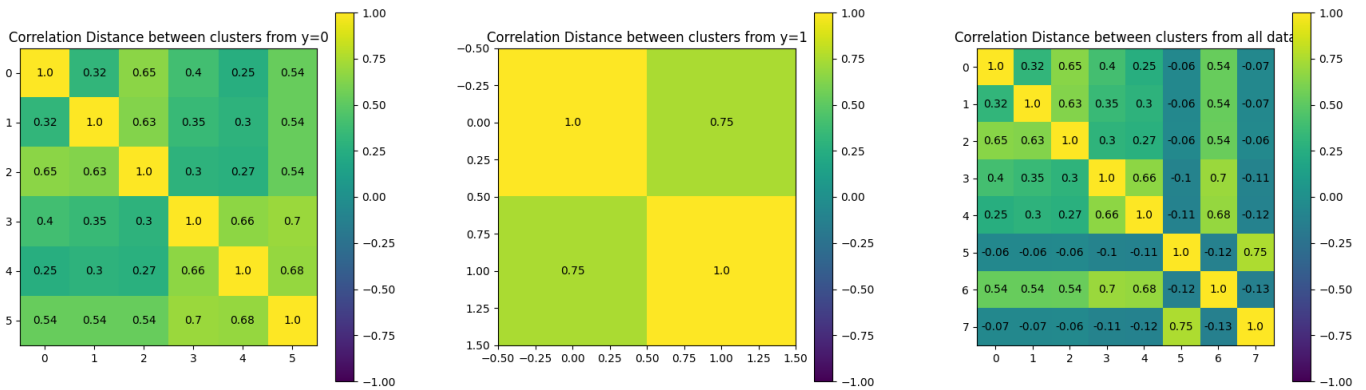


Figure 8.6: Similarities using Pearson's correlation coefficient (see Equation 4.1), between the average heatmaps from each cluster in the 3 clustering runs of C-MNIST. A reminder that -1 or 1 indicates a complete negative or positive correlation, while 0 indicates no correlation. Values close to 0 thereby indicate little similarity.

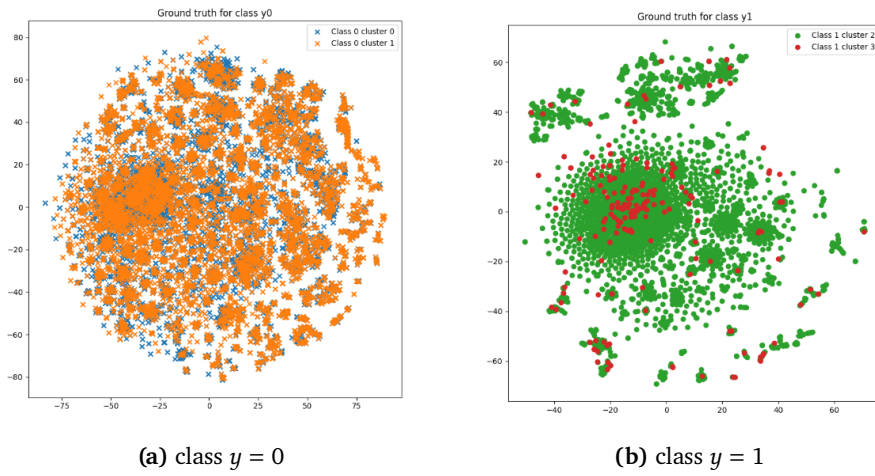


Figure 8.7: t-SNE plot of CelebA heatmaps in 2D coloured with ground truth groups. It is impossible to recover the ground truth labels here.

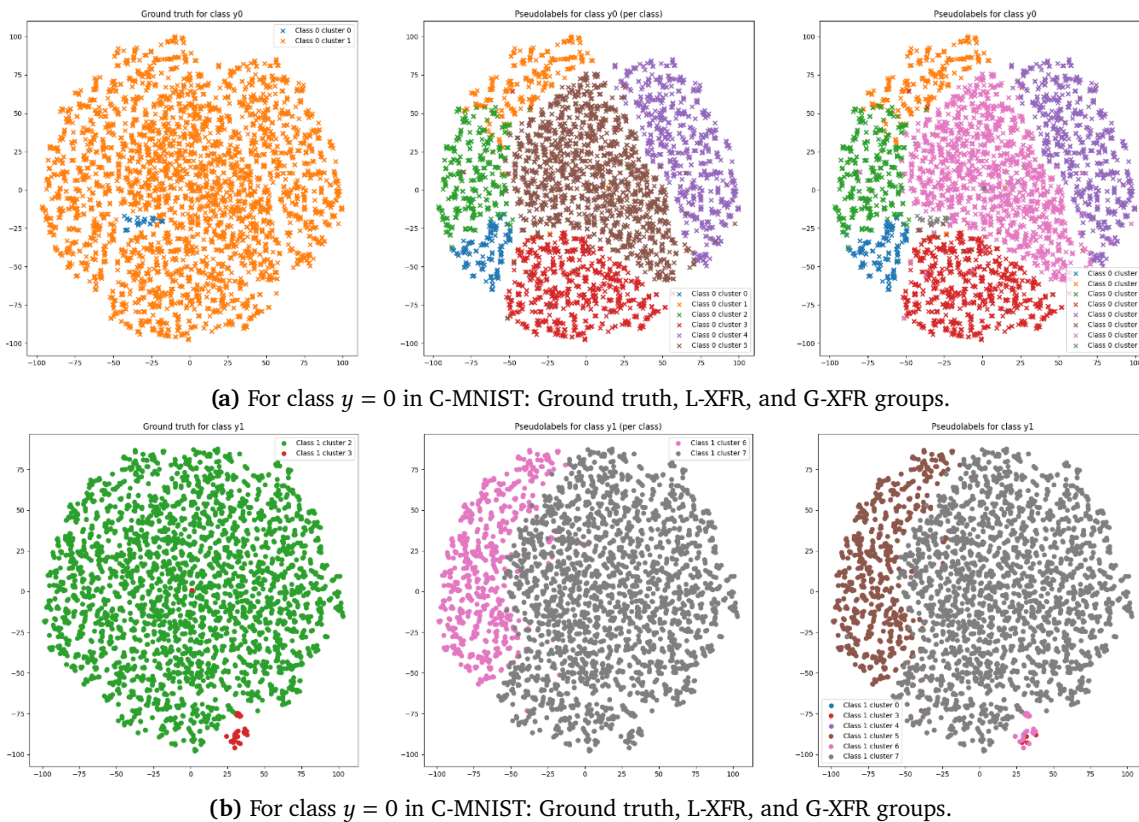


Figure 8.8: t-SNE plot of C-MNIST heatmaps projected down to 2D. G-XFR is able to capture the ground truth minority groups in both cases - the reason for its good performance in Table 8.1. Colours are consistent in the columns, meaning that pink (and also grey) was found as part of the same cluster.

/9

Ablation studies

In this chapter we present a variety of different ablation studies. This is to explore several different ideas. These are sometimes only performed on some of the datasets.

9.1 Changing the downsampling size

In our method, we have utilised the heatmaps with as close to the initial resolution as we could (sometimes needed to use half-size for memory reasons). Here we will explore varying the resolution of the heatmaps to see their effect on XFR. The results for fractions of the initial resolution are presented in Table 9.1.

Performance swings a lot depending on the resolution, need a way of finding a good resolution to use. Ideally would want a smaller size to speed up clustering. G-XFR already performs well for smaller resolutions. The problem is L-XFR, whose performance on Waterbirds decreases a lot, even going lower than WGA of ERM, when smaller heatmaps are utilised.

G-XFR	Waterbirds		CelebA	
	WGA(%)	mean(%)	WGA(%)	mean(%)
Resolutions				
112x112	84.7 \pm 1.0	88.5 \pm 1.3	77.8 \pm 0.9	92.8 \pm 0.0
56x56	86.4 \pm 1.1	90.9 \pm 0.8	73.3 \pm 2.6	83.0 \pm 1.6
28x28	88.2 \pm 0.6	91.7 \pm 0.5	77.0 \pm 1.0	83.6 \pm 0.7
14x14	86.3 \pm 0.5	91.6 \pm 0.3	81.1 \pm 1.0	91.7 \pm 0.2
7x7	90.6 \pm 0.3	93.8 \pm 0.2	75.4 \pm 0.4	93.4 \pm 0.0
ERM	76.8	98.1	41.1	95.9
DFR	92.1 \pm 0.2	94.6 \pm 0.1	86.9 \pm 0.4	91.1 \pm 0.1

L-XFR	Waterbirds		CelebA	
	WGA(%)	mean(%)	WGA(%)	mean(%)
Resolutions				
112x112	92.6 \pm 0.3	94.3 \pm 0.4	83.1 \pm 0.2	89.6 \pm 0.4
56x56	90.8 \pm 0.3	94.0 \pm 0.2	72.4 \pm 1.9	83.4 \pm 1.1
28x28	91.1 \pm 0.6	93.9 \pm 0.3	81.4 \pm 0.4	87.6 \pm 0.3
14x14	76.9 \pm 0.5	84.4 \pm 0.3	82.5 \pm 0.3	88.6 \pm 0.2
7x7	73.4 \pm 0.5	80.0 \pm 0.6	76.8 \pm 0.4	93.3 \pm 0.0
ERM	76.8	98.1	41.1	95.9
DFR	92.1 \pm 0.2	94.6 \pm 0.1	86.9 \pm 0.4	91.1 \pm 0.1

Table 9.1: Results for G-XFR and L-XFR for different downsizing of the heatmaps compared to ERM and DFR baselines. Best results across the resolutions are bolded.

9.1.1 Effect of eigengap heuristic on L-XFR performance

It’s possible that it is not the downsampling size by itself that causes the large dip for L-XFR, but instead the eigengap heuristic. Therefore we look at the performance of L-XFR for the waterbirds dataset for the resolution 7x7 (where the WGA was worse than ERM) without using the eigengap heuristic. Instead, we sweep over different numbers of clusters. For simplicity, we utilise the same number of clusters for both classes.

We observe that the eigengap which ended up choosing 2 clusters for class 0, and 9 clusters for class 1, underperforms compared to any fixed size. This might indicate that eigengap might not be the best measure to be used, at least not for downsized waterbirds explanations.

L-XFR nr. clusters	Waterbirds (7x7)	
	WGA(%)	w. mean(%)
2	86.3 \pm 0.8	94.5 \pm 0.6
3	88.0 \pm 1.3	95.0 \pm 1.0
4	85.5 \pm 1.2	93.7 \pm 1.2
5	87.1 \pm 0.3	95.1 \pm 0.2
6	86.5 \pm 0.5	95.2 \pm 0.3
7	87.3 \pm 1.0	95.0 \pm 0.8
8	90.4 \pm 0.6	95.6 \pm 0.3
9	91.1 \pm 0.2	96.2 \pm 0.2
10	91.0 \pm 0.5	96.6 \pm 0.2
(2,9) eigengap	73.4 \pm 0.5	80.0 \pm 0.6

Table 9.2: Comparing the performances when using a fixed number of clusters vs. the eigengap heuristic that gave 2 clusters for $y = 0$, and 9 clusters for $y = 1$. Here we have presented the weighted (to the group distribution of the training set) mean instead of the normal mean over the test set.

9.2 The effect of eigengap on the main results

Following the results in Table 9.2 where the eigengap heuristic underperforms for a certain resolution of heatmaps, we now explore how the main results are affected by it. Similarly to before, we sweep over different numbers of clusters and compare these results with the ones utilising the eigengap. Our results are presented in Table 9.3.

Eigengap seems to be a nice guess for the best number of clusters (Waterbirds L-XFR and C-MNIST G-XFR), but does not always perform the best (C-MNIST L-XFR and Waterbirds G-XFR). Exploring other methods for choosing the best number of clusters without utilising group labels should definitely be done in follow-up work.

Before moving on, we would like to comment on the jump in WGA seen for L-XFR on C-MNIST. As discussed earlier, we hypothesise that the class-wise clustering has difficulty finding the differently coloured examples in C-MNIST since there are few examples of the minority groups present. We speculate that the performance improvement for L-XFR when using many clusters finally enables these observations to be in their own cluster.

L-XFR	Waterbirds (112x112)		C-MNIST (28x28)		
	nr. clusters	WGA(%)	mean(%)	WGA(%)	mean(%)
2	86.6 \pm 1.1	91.7 \pm 0.8	-	-	
3	89.4 \pm 0.6	92.8 \pm 0.7	20.0 \pm 1.0	99.3 \pm 0.0	
4	90.8 \pm 0.5	94.8 \pm 0.4	18.3 \pm 0.8	99.3 \pm 0.0	
5	86.4 \pm 1.3	88.6 \pm 1.0	19.6 \pm 1.0	99.2 \pm 0.0	
6	86.5 \pm 1.5	89.1 \pm 1.6	17.5 \pm 1.7	98.8 \pm 0.0	
7	88.9 \pm 0.3	91.2 \pm 0.5	16.7 \pm 0.0	98.9 \pm 0.0	
8	85.5 \pm 0.5	90.9 \pm 0.6	15.8 \pm 1.7	98.9 \pm 0.0	
9	87.1 \pm 0.6	92.0 \pm 0.9	50.4\pm1.6	98.3\pm0.0	
10	88.1 \pm 0.6	94.0 \pm 0.8	49.2 \pm 1.7	98.8 \pm 0.0	
eigengap	92.6\pm0.3	94.3\pm0.3	24.8 \pm 1.2	99.2 \pm 0.0	
G-XFR	Waterbirds (112x112)		C-MNIST (28x28)		
	nr. clusters	WGA(%)	mean(%)	WGA(%)	mean(%)
2	77.2 \pm 1.2	83.5 \pm 1.7	-	-	
3	84.7 \pm 1.0	88.5 \pm 1.3	65.0 \pm 2.4	95.7 \pm 0.5	
4	87.4 \pm 0.9	91.1 \pm 1.5	55.4 \pm 2.8	97.4 \pm 0.2	
5	89.1 \pm 0.9	91.2 \pm 1.0	60.4 \pm 1.9	96.4 \pm 0.1	
6	89.5 \pm 0.4	91.5 \pm 0.7	63.3\pm1.7	97.3\pm0.2	
7	87.9 \pm 2.2	90.2 \pm 1.9	63.3 \pm 1.0	96.8 \pm 0.1	
8	88.6 \pm 1.5	90.8 \pm 1.5	63.3 \pm 1.0	96.8 \pm 0.2	
9	90.3\pm0.4	93.4\pm0.5	62.5 \pm 2.3	95.9 \pm 0.2	
10	89.5 \pm 0.6	91.8 \pm 1.1	57.9 \pm 2.4	96.2 \pm 0.2	
eigengap	84.7 \pm 1.0	88.5 \pm 1.3	63.3\pm1.0	96.8\pm0.2	

Table 9.3: Eigengap heuristic C-MNIST: (6,2), 8. Eigengap heuristic for Waterbird: (3,2), 3

9.3 Using 1 cluster

Following the work on the effect of the eigengap, we thought of exploring what happens when only one cluster is used, i.e. clustering is not performed and we use the classes directly as labels. We denote this method as "DFR (class)". The result is presented in Table 9.4, and it shows that retraining using classes as groups improves group robustness. This is an interesting behaviour that has also been pointed out in a concurrent work [62]. The improvement of WGA seen for the Waterbirds dataset is thought to be attributed to the class imbalance. Note that some of the improvements by XFR can probably partially be explained by "DFR (class)".

Methods	C-MNIST		Waterbirds		CelebA	
	WGA(%)	w. mean(%)	WGA(%)	w. mean(%)	WGA(%)	w. mean(%)
ERM	39.6	99.3	76.8	98.1	41.1	95.9
L-XFR	24.8 \pm 1.2	99.2 \pm 0.0	92.6 \pm 0.3	94.3 \pm 0.3	83.1 \pm 0.2	89.6 \pm 0.4
G-XFR	63.3 \pm 1.0	96.8 \pm 0.2	84.7 \pm 1.0	88.5 \pm 1.3	77.8 \pm 0.9	92.8 \pm 0.0
DFR	74.2 \pm 2.1	93.7 \pm 0.2	92.1 \pm 0.2	94.6 \pm 0.1	86.9 \pm 0.4	91.1 \pm 0.1
DFR (class)	31.2\pm0.0	99.3\pm0.0	91.8\pm0.1	94.5\pm0.1	70.9\pm0.3	93.9\pm0.0

Table 9.4: Here we compare "DFR (class)" that uses the classes directly as groups, with other methods.

9.4 Retraining on the training set instead of the validation set

In our approach, we followed the process done by DFR [58], where retraining is done on the validation set. Here we will explore what happens if retraining is performed on utilising the training set. Due to the large size of the training set for CelebA, we will utilise a subsample of it (10 000 images, approx. 6% of the training set).

Methods	C-MNIST		Waterbirds		CelebA	
	WGA(%)	w. mean(%)	WGA(%)	w. mean(%)	WGA(%)	w. mean(%)
ERM	39.6	99.3	76.8	98.1	41.1	95.9
DFR (val)	74.2 \pm 2.1	93.7 \pm 0.2	92.1 \pm 0.2	94.6 \pm 0.1	86.9 \pm 0.4	91.1 \pm 0.1
L-XFR (val)	24.8 \pm 1.2	99.2 \pm 0.0	92.6 \pm 0.3	94.3 \pm 0.3	83.1 \pm 0.2	89.6 \pm 0.4
G-XFR (val)	63.3 \pm 1.0	96.8 \pm 0.2	84.7 \pm 1.0	88.5 \pm 1.3	77.8 \pm 0.9	92.8 \pm 0.0
DFR (train)	54.2 \pm 1.3	99.0 \pm 0.0	89.2 \pm 0.3	97.6 \pm 0.0	88.3 \pm 0.3	90.7 \pm 0.2
L-XFR (train)	45.8 \pm 1.3	99.3 \pm 0.0	85.3 \pm 0.3	93.5 \pm 0.0	74.1 \pm 0.8	92.0 \pm 0.2
G-XFR (train)	56.7 \pm 3.1	98.8 \pm 0.0	85.8 \pm 0.4	93.2 \pm 0.1	74.0 \pm 2.4	91.8 \pm 0.2

Table 9.5: Comparing the difference from using the validation vs the training set to improve group robustness.

In general, we observe a decrease in WGA when utilising the training set. Therefore we would recommend XFR to be applied on the validation set. It is possible that the reduced performance is caused by the fact that the data used for retraining was already used to train the base model. Some possible future work could be to explore splitting the training set (similar to what is done in [92]) to use one part for training the base model and the other part to perform retraining.

9.5 Removing minority groups

In our exploration of the clusters found, we observed that some were quite small. In fact, often there was one large cluster containing most instances and several smaller clusters containing only a few instances. It is possible that these small clusters are outliers and are affecting the performance in a negative way, so we explore what happens when they are removed. To get the results in Table 9.6 we do the standard XFR procedure, but do retraining of the last layer only on the largest group for each class, ignoring the minority groups.

In general, we see that removing the minority groups worsens the performance, indicating that they are more than outliers and hold important information.

Methods	Waterbirds		CelebA	
	WGA(%)	mean(%)	WGA(%)	mean(%)
L-XFR	92.6 \pm 0.3	94.3 \pm 0.3	83.1 \pm 0.2	89.6 \pm 0.4
L-XFR (major)	91.8 \pm 0.2	94.1 \pm 0.1	76.9 \pm 0.9	93.4 \pm 0.0
G-XFR	84.7 \pm 1.0	88.5 \pm 1.3	77.8 \pm 0.9	92.8 \pm 0.0
G-XFR (major)	92.5 \pm 0.1	94.8 \pm 0.1	71.2 \pm 0.2	93.9 \pm 0.0

Table 9.6: Comparison of main results with majority cluster results. Here we perform retraining only utilising the largest group for each class. Bolded values are the overall best results.

9.6 Merging of clusters

Motivated by the high similarity between some of the clusters in Figure 8.5 and Figure 8.6, we will try to combine the most similar clusters so that there are only 2 clusters for each class. We denote these as the "superclusters". We explore this to try to conform to the knowledge that there are 2 groups per class. The intuition is that similar clusters might have similar shortcuts, and the merging into superclusters might create a group with shortcuts and a group without. In Table 9.7 we observe only marginal changes when using superclusters, and sometimes big drops in WGA.

The merging of clusters was performed by using agglomerative clustering with average-link on the absolute value of the correlation between cluster centres (Figure 8.5 and Figure 8.6). By using the absolute value we merge clusters that are highly positive or negatively correlated. It is possible that a better merging algorithm can be used that yields better results, but we will leave it as potential

future work.

Methods	C-MNIST		Waterbirds		CelebA	
	WGA(%)	mean(%)	WGA(%)	mean(%)	WGA(%)	mean(%)
L-XFR	24.8 \pm 1.2	99.2 \pm 0.0	92.6 \pm 0.3	94.3 \pm 0.3	83.1\pm0.2	89.6 \pm 0.4
L-XFR (super)	31.2 \pm 0.0	99.3 \pm 0.0	92.6 \pm 0.3	94.3 \pm 0.4	72.2 \pm 0.4	93.7 \pm 0.0
G-XFR	63.3\pm1.0	96.8 \pm 0.2	84.7 \pm 1.0	88.5 \pm 1.3	77.8 \pm 0.9	92.8 \pm 0.0
G-XFR (super)	21.7 \pm 2.8	99.3 \pm 0.0	86.6 \pm 0.4	91.6 \pm 0.3	78.6 \pm 0.9	92.6 \pm 0.2

Table 9.7: Comparison of main results with supercluster results

Part IV

Conclusion and future work

/10

Conclusion

Shortcut learning, the tendency for models to rely on spurious correlations, is a major issue standing in the way of reliable and trustworthy Deep Learning models. This is because they utilise the principle of least effort, and focus on features that are not relevant to the learning task.

Group robustness has emerged as one of the recent proposed approaches to mitigating shortcut learning and have proven to be useful. However, they require group labels to have good performance. Unfortunately, the acquisition of group labels is a daunting task, requiring time, money, and a deep understanding of the underlying features of the dataset we are interested in.

To solve this issue we proposed a new group-unsupervised group robustness method, Explainability-based feature reweighting (XFR), which consists of two steps: "unsupervised group discovery" and "group balanced training". In the first step, the method finds underlying strategies used by a trained model (by finding groups of its explainability heatmaps), and uses these strategies as pseudo-labels for groups which can be used in the second step to increase group robustness.

The main results show that group robustness was increased by XFR compared to the base model (ERM), and that the proposed method performs better than certain baselines that utilize group labels. Performance sometimes even approaches that of a method using the full group labels. We highlight that despite XFR outperforming other methods it is still outperformed by standard DFR, which may be regarded as an oracle version of our method. This is an expected result, as we are not using any group labels.

In the analysis section, we look into the pseudo-groups found by the unsupervised group estimation. We show that our improved group robustness results are not necessarily caused by capturing the underlying (labelled) group structure. We find that the presence of dissimilar pseudo-groups seems to indicate improvements in group robustness performance, although this would need to be explored further.



Future work

In this thesis, we have presented the first steps towards an unsupervised group robustness method that performs comparably to existing methods that use group labels. The proposed framework due to its general nature, opens up several future directions which can be explored. Below are some possible future directions divided into three groups.

Exploring the unsupervised group discovery component

- Assess the performance of different clustering algorithms, such as k-means and deep clustering methods.
- Explore the use of a different method for producing heatmaps, such as Grad-CAM [103].
- Explore the use of different measures to determine the number of clusters, e.g. like in [6] that use the Fisher Discriminant Analysis (FDA), or like in [111] where the Silhouette (SIL) criterion is used.
- Explore if clusters need to be able to be divided into dissimilar groups for there to be robustness improvements.

Exploring the group balanced training

- Explore the effectiveness of the pseudo-labels for other group robustness methods.

Expanding assessment

- Test performance in the presence of multiple shortcuts. Assess if the proposed method suffers from the whac-a-mole dynamic [69].
- In this thesis we focused on image data, however, group robustness is also usually evaluated on text datasets. Explainability heatmaps can also be acquired for text data, where performance can also be assessed.
- Explore the observation efficiency - How many observations are needed to run the process? The ablation where training is done on a portion of the training set showed good performance using 5% of the training set. What proportions are helpful?
- Develop a way to optimally tune parameters, e.g. number of clusters and resolution of heatmaps, to gain further improvement.
- Find limitations of the framework. One possible limitation is in regard to the positions of features in images. Conceptually, it seems like XFR should cluster similar observations based on the spacial position of the shortcut. This could be better studied with e.g. a synthetic dataset where a shortcut artefact is placed in different positions of the image.

References

- [1] S. Albelwi, “Survey on self-supervised learning: Auxiliary pretext tasks and contrastive learning methods in imaging,” *Entropy*, vol. 24, no. 4, p. 551, 2022.
- [2] M. Z. Alom, T. M. Taha, C. Yakopcic, *et al.*, “A state-of-the-art survey on deep learning theory and architectures,” *Electronics*, vol. 8, no. 3, p. 292, Mar. 2019, ISSN: 2079-9292. DOI: 10.3390/electronics8030292. [Online]. Available: <http://dx.doi.org/10.3390/electronics8030292>.
- [3] E. Alpaydin, *Introduction to Machine Learning, third edition* (Adaptive Computation and Machine Learning series). MIT Press, 2014, ISBN: 978-0-262-02818-9.
- [4] J. Amann, A. Blasimme, E. Vayena, D. Frey, V. I. Madai, and t. P. consortium the, “Explainability for artificial intelligence in healthcare: A multidisciplinary perspective,” *BMC Medical Informatics and Decision Making*, vol. 20, no. 1, p. 310, Nov. 2020, ISSN: 1472-6947. DOI: 10.1186/s12911-020-01332-6. [Online]. Available: <https://doi.org/10.1186/s12911-020-01332-6>.
- [5] M. Ancona, E. Ceolini, C. Öztireli, and M. Gross, “Gradient-based attribution methods,” *Explainable AI: Interpreting, explaining and visualizing deep learning*, pp. 169–191, 2019.
- [6] C. J. Anders, L. Weber, D. Neumann, W. Samek, K.-R. Müller, and S. Lapuschkin, “Finding and removing clever hans: Using explanation methods to debug and improve deep models,” *Information Fusion*, vol. 77, pp. 261–295, 2022.
- [7] P. Azevedo, *Object detection state of the art 2022*, Jun. 2022. [Online]. Available: <https://medium.com/@pedroazevedo6/object-detection-state-of-the-art-2022-ad750e0f6003>.
- [8] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek, “On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation,” *PloS one*, vol. 10, no. 7, e0130140, 2015.
- [9] S. Bach, A. Binder, K.-R. Müller, and W. Samek, “Controlling explanatory heatmap resolution and semantics via decomposition depth,” in *2016 IEEE International Conference on Image Processing (ICIP)*, IEEE, 2016, pp. 2271–2275.

- [10] S. Bai and S. An, “A survey on automatic image caption generation,” *Neurocomputing*, vol. 311, pp. 291–304, 2018.
- [11] S. Basodi, C. Ji, H. Zhang, and Y. Pan, “Gradient amplification: An efficient way to train deep neural networks,” *Big Data Mining and Analytics*, vol. 3, no. 3, pp. 196–207, 2020. DOI: 10.26599/BDMA.2020.9020004.
- [12] S. Beery, G. Van Horn, and P. Perona, “Recognition in terra incognita,” in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 456–473.
- [13] M. Biswas, V. Kuppili, L. Saba, *et al.*, “State-of-the-art review on deep learning in medical imaging,” *Frontiers in Bioscience-Landmark*, vol. 24, no. 3, pp. 380–406, 2019.
- [14] L. Bottou, “Stochastic gradient descent tricks,” in *Neural Networks: Tricks of the Trade: Second Edition*, G. Montavon, G. B. Orr, and K.-R. Müller, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 421–436, ISBN: 978-3-642-35289-8. DOI: 10.1007/978-3-642-35289-8_25. [Online]. Available: https://doi.org/10.1007/978-3-642-35289-8_25.
- [15] J. Bridle, “Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters,” in *Advances in Neural Information Processing Systems*, D. Touretzky, Ed., vol. 2, Morgan-Kaufmann, 1989. [Online]. Available: <https://proceedings.neurips.cc/paper/1989/file/0336dcbab05b9d5ad24f4333c7658a0e-Paper.pdf>.
- [16] J. Buolamwini and T. Gebru, “Gender shades: Intersectional accuracy disparities in commercial gender classification,” in *Conference on fairness, accountability and transparency*, PMLR, 2018, pp. 77–91.
- [17] A. Byerly, T. Kalganova, and R. Ott, “The current state of the art in deep learning for image classification: A review,” in *Science and Information Conference*, Springer, 2022, pp. 88–105.
- [18] C. Chen, O. Li, D. Tao, A. Barnett, C. Rudin, and J. K. Su, “This looks like that: Deep learning for interpretable image recognition,” *Advances in neural information processing systems*, vol. 32, 2019.
- [19] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, *Fast and accurate deep network learning by exponential linear units (elus)*, 2015. DOI: 10.48550/ARXIV.1511.07289. [Online]. Available: <https://arxiv.org/abs/1511.07289> (visited on 02/02/2023).
- [20] G. Cybenko, “Approximation by superpositions of a sigmoidal function,” *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.
- [21] P. Dahal, “Deep clustering: Using deep neural networks for clustering,” *parasdahal.com*, Apr. 2019. [Online]. Available: <https://parasdahal.com/deep-clustering>.
- [22] A. Damle, V. Minden, and L. Ying, “Simple, direct and efficient multi-way spectral clustering,” *Information and Inference: A Journal of the*

- IMA*, vol. 8, no. 1, pp. 181–203, Mar. 2019, ISSN: 2049-8772. DOI: 10.1093/imaiai/iay008. [Online]. Available: <https://doi.org/10.1093/imaiai/iay008>.
- [23] J. Dastin, *Amazon scraps secret ai recruiting tool that showed bias against women*, Oct. 2018. [Online]. Available: <https://www.reuters.com/article/us-amazon-com-jobs-automation-insight-idUSKCN1MK08G>.
- [24] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE conference on computer vision and pattern recognition*, Ieee, 2009, pp. 248–255.
- [25] L. Deng, “The mnist database of handwritten digit images for machine learning research,” *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.
- [26] A. Dhoka, S. Pachauri, C. Nigam, and S. Chouhan, “Machine learning and speech analysis framework for protecting children against harmful online content,” in *2023 Second International Conference on Electronics and Renewable Systems (ICEARS)*, IEEE, 2023, pp. 1420–1424.
- [27] N. Drenkow, N. Sani, I. Shpitser, and M. Unberath, *A systematic review of robustness in deep learning for computer vision: Mind the gap?* 2021. DOI: 10.48550/ARXIV.2112.00639. [Online]. Available: <https://arxiv.org/abs/2112.00639> (visited on 03/12/2023).
- [28] J. Duchi, T. Hashimoto, and H. Namkoong, *Distributionally robust losses for latent covariate mixtures*, 2020. DOI: 10.48550/ARXIV.2007.13982. [Online]. Available: <https://arxiv.org/abs/2007.13982> (visited on 03/01/2023).
- [29] H. Engler, “The behavior of the qr-factorization algorithm with column pivoting,” *Applied Mathematics Letters*, vol. 10, no. 6, pp. 7–11, 1997.
- [30] European Parliament and Council of the European Union. “Regulation (EU) 2016/679 of the European Parliament and of the Council,” of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). (May 4, 2016), [Online]. Available: <https://data.europa.eu/eli/reg/2016/679/oj>.
- [31] E. Forgy, “Cluster analysis of multivariate data: Efficiency versus interpretability of classification,” *Biometrics*, vol. 21, no. 3, pp. 768–769, 1965.
- [32] S. Gallant, “Perceptron-based learning algorithms,” *IEEE Transactions on Neural Networks*, vol. 1, no. 2, pp. 179–191, 1990. DOI: 10.1109/72.80230.
- [33] S. Gautam, A. Boubekki, S. Hansen, *et al.*, “Protovae: A trustworthy self-explainable prototypical variational model,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 17 940–17 952, 2022.
- [34] Y. Ge, Y. Xiao, Z. Xu, X. Wang, and L. Itti, *Contributions of shape, texture, and color in visual recognition*, 2022. DOI: 10.48550/ARXIV.2207.09510.

- [Online]. Available: <https://arxiv.org/abs/2207.09510> (visited on 02/05/2023).
- [35] R. Geirhos, J. Jacobsen, C. Michaelis, *et al.*, “Shortcut learning in deep neural networks,” *CoRR*, vol. abs/2004.07780, 2020. arXiv: 2004.07780. [Online]. Available: <https://arxiv.org/abs/2004.07780> (visited on 12/19/2022).
- [36] R. Geirhos, P. Rubisch, C. Michaelis, M. Bethge, F. A. Wichmann, and W. Brendel, “Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness,” *CoRR*, vol. abs/1811.12231, 2018. arXiv: 1811.12231. [Online]. Available: <http://arxiv.org/abs/1811.12231> (visited on 12/19/2022).
- [37] F. Giuste, W. Shi, Y. Zhu, *et al.*, “Explainable artificial intelligence methods in combating pandemics: A systematic review,” *IEEE Reviews in Biomedical Engineering*, 2022.
- [38] P. Gohel, P. Singh, and M. Mohanty, “Explainable ai: Current status and future directions,” *arXiv preprint arXiv:2107.07045*, 2021.
- [39] R. Gonzalez and R. Woods, *Digital Image Processing, Global Edition*. Pearson Education, 2018, ISBN: 9781292223070.
- [40] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, <http://www.deeplearningbook.org>.
- [41] B. O. GREECE, *A robust machine learning approach for credit risk analysis of large loan level datasets using deep learning and extreme gradient boosting*. [Online]. Available: https://www.bis.org/ifc/events/ifc_9thconf/Petropoulos.pdf.
- [42] J. Gu, Y. Yang, and V. Tresp, “Understanding individual decisions of cnns via contrastive backpropagation,” in *Computer Vision—ACCV 2018: 14th Asian Conference on Computer Vision, Perth, Australia, December 2–6, 2018, Revised Selected Papers, Part III 14*, Springer, 2019, pp. 119–134.
- [43] A. Håkansson and R. L. Hartung, *Artificial Intelligence: Concepts, areas, techniques and applications*. Lund: Studentlitteratur AB, 2020, ISBN: 978-91-44-12599-2.
- [44] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
- [45] W. D. Heaven, *Hundreds of ai tools have been built to catch covid. none of them helped*. Aug. 2022. [Online]. Available: <https://www.technologyreview.com/2021/07/30/1030329/machine-learning-ai-failed-covid-hospital-diagnosis-pandemic/>.
- [46] D. Hendrycks and K. Gimpel, *Gaussian error linear units (gelus)*, 2016. DOI: 10.48550/ARXIV.1606.08415. [Online]. Available: <https://arxiv.org/abs/1606.08415> (visited on 02/02/2023).
- [47] E. T. Heyn, “Berlin’s wonderful horse; he can do almost everything but talk – how he was taught,” Sep. 4, 1904. [Online]. Available: <https://>

- [//timesmachine.nytimes.com/timesmachine/1904/09/04/101396572.pdf](https://timesmachine.nytimes.com/timesmachine/1904/09/04/101396572.pdf) (visited on 02/26/2023).
- [48] A. Holzinger, A. Saranti, C. Molnar, P. Biecek, and W. Samek, “Explainable ai methods-a brief overview,” in *International Workshop on Extending Explainable AI Beyond Deep Models and Classifiers*, Springer, 2020, pp. 13–38.
- [49] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989, ISSN: 0893-6080. DOI: [https://doi.org/10.1016/0893-6080\(89\)90020-8](https://doi.org/10.1016/0893-6080(89)90020-8). [Online]. Available: <https://www.sciencedirect.com/science/article/pii/0893608089900208>.
- [50] Y. Huang and Y. Chen, “Autonomous driving with deep learning: A survey of state-of-art technologies,” *arXiv preprint arXiv:2006.06091*, 2020.
- [51] A. K. Jain, M. N. Murty, and P. J. Flynn, “Data clustering: A review,” *ACM Comput. Surv.*, vol. 31, no. 3, pp. 264–323, Sep. 1999, ISSN: 0360-0300. DOI: [10.1145/331499.331504](https://doi.org/10.1145/331499.331504). [Online]. Available: <https://doi.org/10.1145/331499.331504>.
- [52] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, “What is the best multi-stage architecture for object recognition?” In *2009 IEEE 12th International Conference on Computer Vision*, Sep. 2009, pp. 2146–2153. DOI: [10.1109/ICCV.2009.5459469](https://doi.org/10.1109/ICCV.2009.5459469).
- [53] J. Jumper, R. Evans, A. Pritzel, *et al.*, “Highly accurate protein structure prediction with alphafold,” *Nature*, vol. 596, no. 7873, pp. 583–589, 2021.
- [54] D. E. Kim and M. Gofman, “Comparison of shallow and deep neural networks for network intrusion detection,” in *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*, 2018, pp. 204–208. DOI: [10.1109/CCWC.2018.8301755](https://doi.org/10.1109/CCWC.2018.8301755).
- [55] T. W. Kim and B. R. Routledge, “Why a right to an explanation of algorithmic decision-making should exist: A trust-based approach,” *Business Ethics Quarterly*, vol. 32, no. 1, pp. 75–102, 2022. DOI: [10.1017/beq.2021.3](https://doi.org/10.1017/beq.2021.3).
- [56] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2014. DOI: [10.48550/ARXIV.1412.6980](https://doi.org/10.48550/ARXIV.1412.6980). [Online]. Available: <https://arxiv.org/abs/1412.6980> (visited on 02/01/2023).
- [57] “Pearson’s correlation coefficient,” in *Encyclopedia of Public Health*, W. Kirch, Ed. Dordrecht: Springer Netherlands, 2008, pp. 1090–1091, ISBN: 978-1-4020-5614-7. DOI: [10.1007/978-1-4020-5614-7_2569](https://doi.org/10.1007/978-1-4020-5614-7_2569). [Online]. Available: https://doi.org/10.1007/978-1-4020-5614-7_2569.
- [58] P. Kirichenko, P. Izmailov, and A. G. Wilson, *Last layer re-training is sufficient for robustness to spurious correlations*, 2022. DOI: [10.48550/ARXIV.2204.02937](https://doi.org/10.48550/ARXIV.2204.02937). [Online]. Available: <https://arxiv.org/abs/2204.02937> (visited on 02/07/2023).

- [59] M. Kohlbrenner, A. Bauer, S. Nakajima, A. Binder, W. Samek, and S. Lapuschkin, “Towards best practice in explaining neural network decisions with lrp,” in *2020 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2020, pp. 1–7.
- [60] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017, ISSN: 0001-0782. DOI: 10.1145/3065386. [Online]. Available: <https://doi.org/10.1145/3065386>.
- [61] A. Krogh and J. Hertz, “A simple weight decay can improve generalization,” in *Advances in Neural Information Processing Systems*, J. Moody, S. Hanson, and R. Lippmann, Eds., vol. 4, Morgan-Kaufmann, 1991. [Online]. Available: <https://proceedings.neurips.cc/paper/1991/file/8eefcfd5990e441f0fb6f3fad709e21-Paper.pdf>.
- [62] T. LaBonte, V. Muthukumar, and A. Kumar, *Saving a split for last-layer retraining can improve group robustness without group annotations*, Jul. 2023. [Online]. Available: <https://openreview.net/forum?id=IGPde2wLse>.
- [63] S. Lapuschkin, A. Binder, G. Montavon, K.-R. Müller, and W. Samek, “The lrp toolbox for artificial neural networks,” *Journal of Machine Learning Research*, vol. 17, no. 114, pp. 1–5, 2016. [Online]. Available: <http://jmlr.org/papers/v17/15-618.html>.
- [64] S. Lapuschkin, A. Binder, K.-R. Müller, and W. Samek, “Understanding and comparing deep neural networks for age and gender classification,” in *Proceedings of the IEEE international conference on computer vision workshops*, 2017, pp. 1629–1638.
- [65] S. Lapuschkin, S. Wäldchen, A. Binder, G. Montavon, W. Samek, and K.-R. Müller, “Unmasking clever hans predictors and assessing what machines really learn,” *Nature Communications*, vol. 10, no. 1, Mar. 2019. DOI: 10.1038/s41467-019-08987-4. [Online]. Available: <https://doi.org/10.1038/s41467-019-08987-4>.
- [66] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, “Efficient backprop,” in *Neural Networks: Tricks of the Trade: Second Edition*, G. Montavon, G. B. Orr, and K.-R. Müller, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 9–48, ISBN: 978-3-642-35289-8. DOI: 10.1007/978-3-642-35289-8_3. [Online]. Available: https://doi.org/10.1007/978-3-642-35289-8_3.
- [67] S. Lee, I. Hwang, G.-C. Kang, and B.-T. Zhang, “Improving robustness to texture bias via shape-focused augmentation,” in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2022, pp. 4322–4330. DOI: 10.1109/CVPRW56347.2022.00478. [Online]. Available: https://openaccess.thecvf.com/content/CVPR2022W/HCIS/papers/Lee_Improving_Robustness_to_Texture_Bias_via_Shape-Focused_Augmentation_CVPRW_2022_paper.pdf (visited on 02/20/2023).

- [68] J. Lehman, J. Clune, D. Misevic, *et al.*, “The surprising creativity of digital evolution: A collection of anecdotes from the evolutionary computation and artificial life research communities,” *Artificial life*, vol. 26, no. 2, pp. 274–306, 2020.
- [69] Z. Li, I. Evtimov, A. Gordo, *et al.*, *A whac-a-mole dilemma: Shortcuts come in multiples where mitigating one amplifies others*, 2022. DOI: 10.48550/ARXIV.2212.04825. [Online]. Available: <https://arxiv.org/abs/2212.04825>.
- [70] Y. Liang, S. Li, C. Yan, M. Li, and C. Jiang, “Explaining the black-box model: A survey of local interpretation methods for deep neural networks,” *Neurocomputing*, vol. 419, pp. 168–182, 2021.
- [71] E. Z. Liu, B. Haghighi, A. S. Chen, *et al.*, *Just train twice: Improving group robustness without training group information*, 2021. DOI: 10.48550/ARXIV.2107.09044. [Online]. Available: <https://arxiv.org/abs/2107.09044> (visited on 02/21/2023).
- [72] Z. Liu, P. Luo, X. Wang, and X. Tang, “Deep learning face attributes in the wild,” in *Proceedings of International Conference on Computer Vision (ICCV)*, Dec. 2015.
- [73] L. Lu, “Dying ReLU and initialization: Theory and numerical examples,” *Communications in Computational Physics*, vol. 28, no. 5, pp. 1671–1706, Jun. 2020. DOI: 10.4208/cicp.oa-2020-0165. [Online]. Available: <https://doi.org/10.4208%5C%2Fcicp.oa-2020-0165> (visited on 02/02/2023).
- [74] S. M. Lundberg and S.-I. Lee, “A unified approach to interpreting model predictions,” *Advances in neural information processing systems*, vol. 30, 2017.
- [75] X. Ma, J. Wu, S. Xue, *et al.*, “A comprehensive survey on graph anomaly detection with deep learning,” *IEEE Transactions on Knowledge and Data Engineering*, 2021.
- [76] A. L. Maas, A. Y. Hannun, A. Y. Ng, *et al.*, “Rectifier nonlinearities improve neural network acoustic models,” in *Proc. icml*, Atlanta, Georgia, USA, vol. 30, 2013, p. 3. [Online]. Available: https://ai.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf (visited on 02/02/2023).
- [77] V. Maierov and A. Pinkus, “Lower bounds for approximation by mlp neural networks,” *Neurocomputing*, vol. 25, no. 1, pp. 81–91, 1999, ISSN: 0925-2312. DOI: [https://doi.org/10.1016/S0925-2312\(98\)00111-8](https://doi.org/10.1016/S0925-2312(98)00111-8). [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0925231298001118>.
- [78] W. McCulloch and W. Pitts, “A logical calculus of ideas immanent in nervous activity,” *Bulletin of Mathematical Biophysics*, vol. 5, pp. 127–147, 1943.
- [79] Midjourney, *Midjourney*. [Online]. Available: <https://www.midjourney.com/home/?callbackUrl=%2Fapp%2F>.

- [80] M. Minderer, O. Bachem, N. Houlsby, and M. Tschannen, “Automatic shortcut removal for self-supervised representation learning,” *CoRR*, vol. abs/2002.08822, 2020. arXiv: 2002.08822. [Online]. Available: <https://arxiv.org/abs/2002.08822> (visited on 12/19/2022).
- [81] M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry*. MIT Press, 1969.
- [82] Y. Mo, Y. Wu, X. Yang, F. Liu, and Y. Liao, “Review the state-of-the-art technologies of semantic segmentation based on deep learning,” *Neurocomputing*, vol. 493, pp. 626–646, 2022.
- [83] G. Montavon, A. Binder, S. Lapuschkin, W. Samek, and K.-R. Müller, “Layer-wise relevance propagation: An overview,” *Explainable AI: interpreting, explaining and visualizing deep learning*, pp. 193–209, 2019.
- [84] V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines,” in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.
- [85] T. A. Nakabi and P. Toivanen, “Deep reinforcement learning for energy management in a microgrid with flexible demand,” *Sustainable Energy, Grids and Networks*, vol. 25, p. 100413, 2021.
- [86] J. Nam, H. Cha, S. Ahn, J. Lee, and J. Shin, *Learning from failure: Training debiased classifier from biased classifier*, 2020. DOI: 10.48550/ARXIV.2007.02561. [Online]. Available: <https://arxiv.org/abs/2007.02561>.
- [87] J. Nam, J. Kim, J. Lee, and J. Shin, *Spread spurious attribute: Improving worst-group accuracy with spurious attribute estimation*, 2022. DOI: 10.48550/ARXIV.2204.02070. [Online]. Available: <https://arxiv.org/abs/2204.02070> (visited on 02/21/2023).
- [88] A. Y. Ng, M. Jordan, Y. Weiss, *et al.*, “On spectral clustering: Analysis and an algorithm,” *Proceedings of IEEE Neural Information Processing Systems (NIPS)*, 2002.
- [89] OpenAI, *Chatgpt*. [Online]. Available: <https://openai.com/blog/chatgpt>.
- [90] O. Pfungst, *Clever Hans (The Horse of Mr. Von Osten) A contribution to experimental animal and human psychology*, trans. by C. L. Rahn. 1911. [Online]. Available: <https://www.gutenberg.org/files/33936/33936-h/33936-h.htm> (visited on 02/26/2023).
- [91] M. Popel, M. Tomkova, J. Tomek, *et al.*, “Transforming machine translation: A deep learning system reaches news translation quality comparable to human professionals,” *Nature communications*, vol. 11, no. 1, p. 4381, 2020.
- [92] S. Qiu, A. Potapczynski, P. Izmailov, and A. G. Wilson, “Simple and fast group robustness by automatic feature reweighting,” *arXiv preprint arXiv:2306.11074*, 2023.
- [93] P. Ramachandran, B. Zoph, and Q. V. Le, “Searching for activation functions,” *CoRR*, vol. abs/1710.05941, 2017. arXiv: 1710.05941. [On-

- line]. Available: <http://arxiv.org/abs/1710.05941> (visited on 02/02/2023).
- [94] M. T. Ribeiro, S. Singh, and C. Guestrin, ““ why should i trust you?” explaining the predictions of any classifier,” in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 1135–1144.
- [95] S. Ruder, “An overview of gradient descent optimization algorithms,” *CoRR*, vol. abs/1609.04747, 2016. arXiv: 1609.04747. [Online]. Available: <http://arxiv.org/abs/1609.04747> (visited on 04/02/2023).
- [96] C. Rudin, “Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead,” *Nature Machine Intelligence*, vol. 1, no. 5, pp. 206–215, May 2019, ISSN: 2522-5839. DOI: 10.1038/s42256-019-0048-x. [Online]. Available: <https://doi.org/10.1038/s42256-019-0048-x>.
- [97] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, pp. 533–536, 1986.
- [98] A. Sable, *End to end automatic speech recognition: State of the art*, Sep. 2022. [Online]. Available: <https://blog.paperspace.com/end-to-end-automatic-speech-recognition-state-of-the-art/>.
- [99] S. Sagawa, P. W. Koh, T. B. Hashimoto, and P. Liang, *Distributionally robust neural networks for group shifts: On the importance of regularization for worst-case generalization*, 2019. DOI: 10.48550/ARXIV.1911.08731. [Online]. Available: <https://arxiv.org/abs/1911.08731> (visited on 03/01/2023).
- [100] R. Saleem, B. Yuan, F. Kurugollu, A. Anjum, and L. Liu, “Explaining deep neural networks: A survey on the global interpretation methods,” *Neurocomputing*, 2022.
- [101] A. G. Salman, B. Kanigoro, and Y. Heryadi, “Weather forecasting using deep learning techniques,” in *2015 international conference on advanced computer science and information systems (ICACSIS)*, Ieee, 2015, pp. 281–285.
- [102] L. Schmarje, M. Santarossa, S.-M. Schröder, and R. Koch, “A survey on semi-, self-and unsupervised learning for image classification,” *IEEE Access*, vol. 9, pp. 82146–82168, 2021.
- [103] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, “Grad-cam: Visual explanations from deep networks via gradient-based localization,” in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 618–626.
- [104] J. C. Shane, *Do neural nets dream of electric sheep?* Mar. 2018. [Online]. Available: <https://www.aiweirdness.com/do-neural-nets-dream-of-electric-18-03-02/>.

- [105] J. Shi and J. Malik, “Normalized cuts and image segmentation,” *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, no. 8, pp. 888–905, 2000.
- [106] E. H. Shortliffe and M. J. Sepúlveda, “Clinical Decision Support in the Era of Artificial Intelligence,” *JAMA*, vol. 320, no. 21, pp. 2199–2200, Dec. 2018, ISSN: 0098-7484. DOI: 10.1001/jama.2018.17163. eprint: https://jamanetwork.com/journals/jama/articlepdf/2713901/jama_shortliffe_2018_vp_180139.pdf. [Online]. Available: <https://doi.org/10.1001/jama.2018.17163>.
- [107] D. Silver, T. Hubert, J. Schrittwieser, *et al.*, “Mastering chess and shogi by self-play with a general reinforcement learning algorithm,” *arXiv preprint arXiv:1712.01815*, 2017.
- [108] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.
- [109] U. Singer, A. Polyak, T. Hayes, *et al.*, “Make-a-video: Text-to-video generation without text-video data,” *arXiv preprint arXiv:2209.14792*, 2022.
- [110] A. Sletten, “An overview of shortcut learning in deep learning,” *UiT project thesis*, 2023.
- [111] N. Sohoni, J. Dunnmon, G. Angus, A. Gu, and C. Ré, “No subclass left behind: Fine-grained robustness in coarse-grained classification problems,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 19 339–19 352, 2020.
- [112] M. Soori, B. Arezoo, and R. Dastres, “Artificial intelligence, machine learning and deep learning in advanced robotics, a review,” *Cognitive Robotics*, 2023.
- [113] M. Steinbach, L. Ertöz, and V. Kumar, “The challenges of clustering high dimensional data,” in *New Directions in Statistical Physics: Econophysics, Bioinformatics, and Pattern Recognition*, L. T. Wille, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 273–309, ISBN: 978-3-662-08968-2. DOI: 10.1007/978-3-662-08968-2_16. [Online]. Available: https://doi.org/10.1007/978-3-662-08968-2_16.
- [114] I. Sturm, S. Lapuschkin, W. Samek, and K.-R. Müller, “Interpretable deep neural networks for single-trial eeg classification,” *Journal of neuroscience methods*, vol. 274, pp. 141–145, 2016.
- [115] H. Team, *State-of-the-art ai approaches to cybersecurity*, Jul. 2022. [Online]. Available: <https://medium.com/hackless/state-of-the-art-ai-approaches-to-cybersecurity-17cb17ae9373>.
- [116] S. Theodoridis and K. Koutroumbas, *Pattern Recognition, Fourth Edition*. Academic Press, 2009, ISBN: 9781597492720.
- [117] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, “Attention is all you need,” *Advances in neural information processing systems*, vol. 30, 2017.
- [118] U. Von Luxburg, “A tutorial on spectral clustering,” *Statistics and computing*, vol. 17, pp. 395–416, 2007.

- [119] Q. Wang, Y. Ma, K. Zhao, and Y. Tian, “A comprehensive survey of loss functions in machine learning,” *Annals of Data Science*, pp. 1–26, 2020.
- [120] P. Welinder, S. Branson, T. Mita, *et al.*, “Caltech-ucsd birds 200,” 2010. [Online]. Available: <https://api.semanticscholar.org/CorpusID:7138640>.
- [121] “What’s the next word in large language models?” *Nature Machine Intelligence*, vol. 5, no. 4, pp. 331–332, Apr. 2023, ISSN: 2522-5839. DOI: 10.1038/s42256-023-00655-z. [Online]. Available: <https://doi.org/10.1038/s42256-023-00655-z>.
- [122] D. Xu and Y. Tian, “A comprehensive survey of clustering algorithms,” *Annals of Data Science*, vol. 2, no. 2, pp. 165–193, Jun. 2015, ISSN: 2198-5812. DOI: 10.1007/s40745-015-0040-1. [Online]. Available: <https://doi.org/10.1007/s40745-015-0040-1>.
- [123] T. Xu, *Ai makes decisions we don’t understand. that’s a problem*. Jul. 2021. [Online]. Available: <https://builtin.com/artificial-intelligence/ai-right-explanation>.
- [124] Q. Yuan, H. Shen, T. Li, *et al.*, “Deep learning in environmental remote sensing: Achievements and challenges,” *Remote Sensing of Environment*, vol. 241, p. 111716, 2020.
- [125] J. R. Zech, M. A. Badgeley, M. Liu, A. B. Costa, J. J. Titano, and E. K. Oermann, “Variable generalization performance of a deep learning model to detect pneumonia in chest radiographs: A cross-sectional study,” *PLOS Medicine*, vol. 15, no. 11, pp. 1–17, Nov. 2018. DOI: 10.1371/journal.pmed.1002683. [Online]. Available: <https://doi.org/10.1371/journal.pmed.1002683>.
- [126] S. Zhang, L. Yao, A. Sun, and Y. Tay, “Deep learning based recommender system: A survey and new perspectives,” *ACM computing surveys (CSUR)*, vol. 52, no. 1, pp. 1–38, 2019.
- [127] B. Zhou, À. Lapedriza, J. Xiao, A. Torralba, and A. Oliva, “Learning deep features for scene recognition using places database,” in *NIPS*, 2014. [Online]. Available: <https://api.semanticscholar.org/CorpusID:1849990>.
- [128] S. Zhou, H. Xu, Z. Zheng, *et al.*, “A comprehensive survey on deep clustering: Taxonomy, challenges, and future directions,” *arXiv preprint arXiv:2206.07579*, 2022.

