NEURAL NETWORKS FOR ITERATIVE FEATURE IMPORTANCE ANALYSIS OF DEEP LEARNING MODELS

A thesis submitted to the University of Manchester for the degree of
DOCTOR OF PHILOSOPHY
in the Faculty of Science and Engineering

2023

Maksymilian A. Wojtas

The School of Engineering
Department of Computer Science

Contents

Word count: 30050

Abstract

Deep neural networks established themselves as a powerful tool used for multiple real-world applications and research purposes. On the other hand, the exact nature that gives them the power to discover high-level, non-linear dependencies also comes with a major disadvantage: a lack of interpretability.

This drawback is eminent for many domains which require explanations of predictions made by a model. The domains this applies to include self-driving cars, medicine, credit rating prediction or even the stock market. Additionally, an interpretable model that provides explanations can help the user to discover new rules governing the data by looking at how the model makes decisions.

In this thesis, we will study model explainability and interpretability through the lens of feature selection via feature importances. First, we will try to understand what explainable AI (XAI) is and then present a background of different methods used for Feature Selection (FS) that can be applied to Machine Learning (ML) and Deep Learning (DL) models. Then, we will formally state the feature importance problem in a model-independent way. Furthermore, we will propose two dual-net (each of them assembled from two neural networks) models that can be applied for feature selection and feature importance estimation where one of the models can only be applied for a given size of the optimal feature subset while the second one is able to determine the global optimum during training. Also, we will present a comparison of the task performance and feature selection results between these two models and other commonly used methods in this field, which shows that the novel models presented here are able to achieve supreme performance through the process of online feature selection. Finally, we will show results for possible extensions to these models, like group-based feature selection or feature subset encodings.

Declaration

Copyright

Acknowledgements

I would like to begin by expressing my heartfelt appreciation to my supervisor, Ke Chen, for his invaluable guidance, support, and mentorship throughout my PhD program. His wisdom, expertise, and unwavering commitment to my success have been instrumental in shaping my research work and professional development. I am truly grateful for the opportunity to work with such a brilliant and inspiring mentor.

I would also like to extend my deepest gratitude and thanks to my wife, Margaret, for her unwavering support and encouragement throughout my PhD journey. She has been my rock, my sounding board, and my source of motivation during both the good times and the tough times. Her love and belief in me have been instrumental in getting me to where I am today.

Finally, I would like to thank my parents for their endless love, encouragement, and support throughout my life and especially during my PhD program. Their unwavering belief in me has been a constant source of inspiration and motivation. I am forever grateful for their sacrifices and dedication to my education.

I am truly blessed to have such amazing individuals in my life. Thank you all from the bottom of my heart.

Acronyms

CI	Confidence Interval.
DL	Deep Learning.
FIE	Feature Importance Estimation.
FIR	Feature Importance Rankings.
FS	Feature Selection.
IFS	Instance-wise (Local) Feature Selection.
MI	Mutual Information.
ML	Machine Learning.
MSE	Mean Squared Error function.
NLP	Natural Language Processing.
NN	Neural Network.
PFS	Population-wise (Global) Feature Selection.
ReLU	Rectified Linear Unit activation function.
SGD	Stochastic Gradient Descent.
SVM	Support Vector Machine.
XAI	Explainable Artificial Intelligence.

Chapter 1

Introduction

In machine learning, feature importance ranking (FIR) refers to a task that measures the contributions of individual input features (variables) to the performance of a supervised learning model. FIR has become one of the powerful tools in explainable/interpretable AI [1] to facilitate understanding of decision-making by a learning system and discovery of critical factors in a specific domain, e.g., in medicine, what genes are likely leading causes of cancer [2].

Feature selection (FS) is a similar concept that aims to find the best subset of features for a particular task. Especially today, in the domain of big data with thousands of high-dimensional samples, most of the 'old' FS methods prove to be unusable, mainly due to low performance, high computational burden or both [3]. Due to the existence of correlated/dependent and redundant features to targets in high-dimensional real data, one goal of feature selection [4] is to address the well-known curse of dimensionality challenge and to improve the generalization of a learning system by maximizing the performance of the system. Feature selection may be conducted at either population (PFS, global FS, population-wise) or instance level (IFS, local FS, instance-wise); the population-wise methods would find out an optimal feature subset collectively for all the instances in a population, while the instance-wise ones tend to uncover a subset of salient features specific to a single sample. Both methods can show us which variables are deemed important by the model in general and which ones were used by the model for the current prediction, which is one of the goals of explainable AI [1].

Deep learning (DL) has become extremely powerful in intelligent system development, allowing it to achieve the best results in many areas (time-series analysis, video detection, and image classification). Due to its performance compared to other, less accurate methods, one can conjecture that it can use the information embedded in datasets to the fullest. On the other hand, its purported "black box" nature makes it extremely difficult to apply to tasks demanding explainability/interpretability, thus preventing extraction of the feature importance from DL models. In this thesis, we plan to use the power of DL in order to maximize the performance of other models through iterative feature selection, also increasing the understanding of the underlying data.

1.1 Population-wise Feature Selection

The Population-wise Feature Selection (PFS), called global feature selection, may be divided into three categories. **Filter methods** focus on selecting the features before the model is trained (CCM [5], mRMR [6], RFE [2]). **Embedded methods** are used to extract feature rankings from a trained model(LASSO [7], Random Forests [8]). **Wrapper methods** focus on retraining a model with different feature subsets iteratively, which is why they are rarely used due to the high computational burden [9].

In the context of DL, only several methods can be classified as PFS when compared to the multitude of IFS methods. The scarcity of PFS DL-related methods might be caused by the innate inexplicability of population-wise variable importances for DL models and the computational burden needed to retrain them.

1.2 Instance-wise Feature Importance Ranking

In the context of one sample, it makes more sense to talk about instance-wise feature importance ranking (IFIR) rather than feature selection, also called local feature selection. The usefulness of IFIR focuses on the explainability of models, which is especially important for the applicability of DL. The most known work in the context of DL focuses on calculating input gradients of the trained model with respect to the given output: SmoothGrad, Vargrad, Integrated Gradients, "vanilla" gradient backprop [10, 11, 12, 13, 14, 15]. A lot of work also shows how the methods mentioned above fail logical tests [16, 17, 18]. Especially one of them [17] shows that the rankings developed from most of the above methods perform *worse* than random rankings when the models are appropriately retrained using new subsets of variables. It is worth noting that similar thinking can be applied to PFS in order to validate the results. As most of these methods are tested on pictorial datasets due to the easiness of visual examination, the datasets consist of a significant degree of associated variables. The authors of [17] shows the failure of the above methods but presents no explanations. I conjecture that these redundancies are the underlying reason for the failure of these methods.

1.3 Group Population-wise Feature Selection

Group Population-wise Feature Selection (GPFS) can be used to take advantage of prior knowledge about features when these features can be grouped. A good example of a typical use case is a pictorial dataset, where one pixel corresponds to three features (one per colour channel) or gene expression data where each gene has to be explained by several features. It is obvious that removing only one channel of a pixel makes no sense: the whole pixel would have to be removed. Another example would be signal-processing data with different frequency bands. One of the most common methods is Group Lasso [19] and its variations. Unfortunately, most IFS and PFS methods don't have a group extension, which limits their performance in this domain. Using the prior grouping and structure should increase the interpretability of the results and allow the method to be used in a wider scope of applications.

1.4 Scope Of The Thesis

Deep learning is a field where most of the discoveries and developments were made in the last decade, which resulted in a lot of practical applications but fewer published theoretical approaches. The avenues of research in the DL field are many, and each raises its own set of questions [20]. Below, we will summarise some avenues of research that were researched for this thesis, as well as some that are clearly **out** of the scope of this thesis.

1. Firstly, we want to find a way to evaluate feature subsets with an underlying connection to interpretability through performance changes and feature importance.

2. We want to find an optimal (according to the point above) feature subset, thus maximising the performance of a model.

3. We want to work with feature subsets of variable size, enabling us to perform a global search over the feature powerset. This also includes the ability to perform GPFS.

4. Finally, we look for a method that enables us to visualize the feature subsets and how they relate to one another.

Out of scope Starting with the adversarial domain, we have adversarial features selection [21] or detection of adversarial inputs [22]. Another important field is the stability of DL approaches, where small perturbations in the input data can diametrically change the model predictions [23] or stability of general ML feature selection algorithms [24]. Furthermore, different models can have different optimal subsets of features due to the predictive capabilities of a model; as an extreme example, one just has to compare linear models with NNs, where linear models will rarely use non-linear dependencies to their fullest potential. Weight interpretation is another approach where much of the research is done [25]. Furthermore, there is much work being done in order to speed up NNs speed, both for training and prediction purposes, which involves node pruning [26, 27] or knowledge distillation [28]. Finally, instance-wise (local) feature selection is also out of the scope of this work, but many insights developed for PFS also apply to IFS, which include minimum requirements that any FS method should fulfil [17, 29].

1.5 Contributions Of The Thesis

The main contributions of this work are written down below and correspond to the research questions raised in Section 1.4:

1. First, we propose a performance-driven feature importance metric, which can be used for any model and is expressive enough to be easily understood and give insight into the way a model makes predictions and the nature of a dataset.

2. We develop a DL-based framework of two models that can accurately find an optimal subset of a given size and produce feature rankings for any data. We carry out a comprehensive comparison with other PFS methods, with both performance and analysis of produced feature rankings. We also produce results across different sizes of subsets to compare performances with other methods in different environments. Finally, we produce visually compelling results for FIR and FIE using pictorial datasets.

3. The third contribution is based on another DL framework that is a major upgrade to the one described in the previous point. A major hurdle was overcome by allowing the framework to look for optimal subsets for different subset sizes while maintaining the performance-driven nature and broadening the expressive abilities of FIE. Again, we carry out a comprehensive comparison with other PFS methods, including more methods than in the previous analysis. The comparison is based mainly on the performance while keeping in mind the correctness of FIE results. We show that it is possible to extend this algorithm to GPFS and present results for such a setting.

4. We propose an initial analysis of a novel way of interacting with datasets through feature subset encodings that help to visualize the complex interactions inside the datasets that are detected by DL models. Due to the innovative properties of this method, there is no real comparison possible, so we are satisfied with presenting the results and proving the method's effectiveness for a synthetic dataset where the ground truth is known.

1.6 Publications Included In The Thesis

Published Papers

1. **Maksymilian Wojtas** and Ke Chen. Feature importance ranking for deep learning. In Advances in Neural Information Processing Systems, volume 33, pages 5105–5114. Curran Associates, Inc., 2020 [30] [1]

The article published under "1." correlates with Section 4. While Maksymilian Wojtas carried out the research, coding, literature review and writing, the

[1]As the paper was limited in size due to conference limitations, we have also created supplementary material. The contents of this additional text are also included in this thesis.

supervisor, Ke Chen, helped with the writing and literature review.

Chapter 2

Background & Related Work

In this section, we will discuss the motivations for explainable AI (XAI) and discuss different approaches that can be taken in order to explain AI models. Although this is not the main contribution of the thesis, the underlying motivation for FIE is gaining an understanding of the data/model, which is also connected to interpretability. Then, we will dive deeper into the topic of explaining AI through feature importance estimates (FIE), and feature importance rankings (FIR) and finally discuss different feature selection methods, as well as their categorization. However, let us start with a short clarification of the nomenclature below.

2.1 Motivation For Explainable AI

First, let us discuss the differences in nomenclature that are often confused by many papers from the XAI field, that is the interchangeable usage of **interpretability** and **explainability**. The former can be viewed as a passive function of a model, where one can look at the inner workings of a model, such as architecture or parameters and discover why the model behaves in a particular way. Another term commonly used in this context is transparency. On the other hand, explainability is an active part of a model that is able to produce explanations that are easily understandable by a human and provide useful information about the behaviour of the model.

This thesis focuses on providing a model that passively shows which features are being used, meaning that we focus on an interpretable approach. We carry out performance-based research with additional visual validation for FIE while

keeping in mind that gaining an understanding of the broader context of feature importance is important to fully understand the motivations behind our research. Explainable AI is a relatively new topic where much of the work done was recent [31]. The reason behind such an amount of work done is simple: due to recent rises in computational power, the DL solutions have become more accessible to the public [32]. Now, when decisions made by Deep Neural Networks start to affect human lives, it is necessary to produce the reasoning behind the decisions taken. Unfortunately, the number of parameters needed to train the deepest of neural networks approaches hundreds of millions, and it is simply not feasible to be able to process such a complex model and try to explain its predictions. That is why, before we start to use DL models in areas such as medicine, self-driving cars or law, we need first to handle the black-box nature of neural networks.
Moreover, humans stay clear of techniques they do not fully understand [33], or in other words, techniques that are not interpretable or explainable.
On the contrary, black-box models' supreme performance is partially due to their complexity and ability to grasp highly nonlinear interactions, so making such a model explainable, which can be seen as a criterion that a model must fulfil, will make it deteriorate in performance [34].
Still, the criterion of explainability can bring other practical advantages, thus increasing the training performance, as observed by [31]:

- Detection of bias in the training data, which can be followed by a correction applied to counteract the bias.
- Identification of potential adversarial perturbations that affect the predictions can be helpful in increasing the model performance on the testing data.
- Confirmation of causality between the important input variables and the prediction target.

This thesis will focus on the last mentioned point, which is a natural consequence of feature importance analysis, which is often done as a part of a feature selection process.
Finally, as we mentioned before, users also need explainable models for other, less

practical purposes. Below we have gathered several typical purposes for which XAI can be used:

1. **Causality**, the main focus of this thesis, is often found to be the reason behind creating an explainable model. It helps to grasp insights from available data and allows to discover hidden variable dependencies [35].

2. **Fairness**, discovering the biases in the data as well as making sure that the AI models' behaviour is ethical (non-discriminating) is an important aspect of XAI that was deemed as a requirement by the EU in their GDPR regulation [36].

3. **Confidence**, sometimes called **Trustworthiness**, is a measure of the reliability of a model for a given task. XAI should convince the user that the model is performing as projected [37], as well as show the possible bounds for the model's correct operation. This is related to the model stability [38], as unstable models rarely inspire confidence.

4. **Informativeness** is a major reason behind developing explainable models [31]. It allows the user to discover the inner workings of a model and explain why the model made a particular decision.

To facilitate XAI, different ways of showing explanations were investigated. These include *visual explanations*, *text explanations*, *rule-based explanations* (or in other words *simplifications*), *explaining by example* and finally, the main focus of this thesis, *feature importance*, sometimes also called *feature relevance*.

2.2 XAI For DL Through Feature Importance

Feature importance or feature relevance is a metric that allows the user to find out how the model responds to different features or discover the complex feature dependencies that are already present in the data. Feature importance is often considered an indirect way to achieve model explainability.
The categorization of the types of feature importance [39] can be understood by providing four different examples, one per category:

1. **Global (population-wise) model importance** is a setting that is used to determine, in general, which of the variables are used by a model to make predictions and how much predictive power each of them has in relation to others. This setting can be used to validate the framework created by the model, increasing the **trustworthiness** as well as providing information about different biases in the dataset when compared to subject knowledge (**fairness**).

2. **Global (population-wise) data importance** allows us to find complex feature dependencies that are understood by a model and gain new knowledge about the phenomena that we are trying to predict (**causality**).

3. **Local (instance-wise) model importance** is another scenario that can help with XAI, as it provides explanations for individual samples in the context of a prediction made by a model. It is a major source of **informativeness** for the model.

4. **Local (instance-wise) data importance** is a rarely researched aspect of feature importance [39] that tries to answer the question of why a particular result came into being. One possible reason for the complexity of this approach is the differentiation between data and model explanations on the basis of one data point.

Obviously, there is, and there should be, an intersection between the model and data feature importances, as increasing the performance of a model can be understood as it finding some new "knowledge", thus allowing us to assume that the difference between data and model feature importances would be zero for a task with an ideal model. This is the main reason why we pursue the path of maximizing the model performance while measuring FIE, as we understand that better-performing models can achieve greater performance by using the same features in different ways (compared to the lower-performing models), which leads to different feature importance estimates. It is worth mentioning that this assumption does not hold [39] when dealing with redundant variables, as the choice between which variables are being used by the model might be initialization-dictated and random, even in an ideal case. Nevertheless, the gap between model and data importance measures

diminishes when a model approaches ideal performance. That is why our work is based on achieving **Global (population-wise) data importance** and **Global (population-wise) model importance** through performance maximization.
It was also proven that using the expected value of instance-wise methods for providing population-wise explanations should not be used [39]. That is why we have not included methods that aggregate the values of local FIE to create global FIE in the second half of our work.

2.3 Population Feature Selection Methods

In this subsection, we cover different methods of Feature Selection and Feature Importance Estimation. It is worth pointing out that not all the methods that enable FS provide FIE.

2.3.1 Wrapper Methods

Wrapper methods are iterative methods that use the feedback from the model to evaluate different feature subsets and choose a new perspective optimal subset. When a stopping criterion is reached, the framework returns a proposal for an optimal subset. One of their disadvantages is the computational burden required to evaluate different feature subsets, as most methods need to be retrained for every subset. On the other hand, the produced results are tuned to a selected predictive model.

RFE [2] , or Recursive Feature Elimination, is a wrapper method that can be employed for different models that are able to produce feature rankings internally. It is a popular choice of FS for biological data and is often used in tandem with SVM. While allowing for many models to be used in tandem, the model's repeated retraining makes it computationally expensive.

Backward and Forward FS [40] are basic wrapper algorithms that compare the performance of models when features are added/removed until a stopping criterion is reached. They are powerful tools but suffer from extensive computation burdens, even more than RFE. On the other hand, they are able to include the associations between features due to their exhaustive search.

BAHSIC [41] is a method combining backward feature selection with Hilbert Schmidt Independence Criterion (HSIC) that can be applied to regression, binary of multilabel analysis. It still suffers from the same problems as the vanilla backwards feature selection.

2.3.2 Filter Methods

Filter methods are model-independent and are used before the main model is used to create predictions as a part of a two-step pipeline (during the second step, the features chosen in the first step are used by a model to generate predictions). Their advantage is their relative speed compared to wrapper methods and model independence which allows them to be flexible and easily applied to any analysis. However, the same model independence prevents them from tuning the proposed feature subset to the inductive model.

CCM [5] is a kernel method that is based on a trace of the conditional covariance operator that produces an optimal subset of a given size. It does not produce FIE scores. While certainly fast, the memory requirements for several bigger datasets made it impossible to run with normal equipment.

MRMR [6] is a widely popular FS method based on mutual information metric. It uses an iterative process to determine a subset of variables that are minimally correlated with each other and maximize the correlation with the target variable. Its basis in information theory makes it an understandable and powerful method, though the computational power requirements make it sometimes run longer than a model run afterwards(which counters the advantage of the filter methods).

2.3.3 Embedded Methods

The embedded methods try to integrate the advantages of filter and wrapper methods without increasing the computational burden. For this group of methods, the model can "delete" features during training through regularization or pruning so that in the end we are left with a model trained with an optimal subset of features. Usually, the user is also left with FIE that are used to measure the performances of different features. In this subsection, we will use $\boldsymbol{w}$ to mark the vector of weights that is associated with variable importances that are held by the model.

LASSO [42] is a widely popular method of regularized least squares minimization that includes the penalty criterion, where λ is a hyperparameter guiding the relative strength of the penalty compared to the actual loss of the model:

$$\text{penalty} = \lambda||\boldsymbol{w}||. \tag{2.1}$$

An obvious disadvantage is the fact that LASSO can only handle linear correlations, but is a quick method to run.

ElasticNet [43] is an extension of LASSO that prevents the model from deleting most of the features while providing relevant FIE by adding L2 penalty term:

$$\text{penalty} = \lambda_1||\boldsymbol{w}|| + \lambda_2||\boldsymbol{w}||_2, \tag{2.2}$$

where λ_1, λ_2 perform a similar role to λ for LASSO.

DeepPINK [44] is a DL-focused method that uses Deep Knockoffs [45] to measure the model's response to original features and knock-off features. The responses are then compared in order to provide FIE finally. The NN uses L1 regularization and provides an estimate of the false discovery rate (FDR). The feature weight estimation also involves reading the network weights which is a process often found when other DL-focused methods try to extract FIE. Unfortunately, it often fails for really deep networks with nonlinear activation functions.

DFS [46] or Deep Feature Selection is another DL-based method that uses L1 regularization in its first layer (which is set to use one-to-one connections) to find FIE. While certainly easy in implementation and readability of results, we found that it rarely gave the best results.

AvGrad [47] is a method that aggregates input gradients (which are usually connected to local FIE) of a NN in order to create a global FIE. We included this method only for completeness as it was already proven that methods that take the expected value of local FIE cannot create correct global FIE [39].

2.4 Local Feature Selection Methods

In this section, we will provide only a brief description of IFS with a focus on DL, as the main focus of our work is PFS. It is however worth noting the approaches used by different researchers that allowed them to discover the inner nature of NNs because the ideas behind them can also be used to build PFS methods.

LIME [48] is a technique that probes the local neighbourhood of a prediction that was created by a trained model and trains another interpretable ML model on the created data. The interpretable nature of the second model allows the user to extract information about the way in which features are being used to produce original predictions. It is a widely used, model-independent method that leans on its linear nature to provide explanations.

SmoothGrad [10] is a more advanced method of getting FIE that perturbs the input sample several times before measuring the input gradients of a NN. Then, the results are aggregated across perturbations in order to create the final result. We included this method, as well as the other gradient-based methods from below as they are one of the most widely used for deep learning. On the other hand, the validity of input gradients as a method to measure feature importance is heavily disputed.

PatternNet, LRP [29, 49] investigate methods that use signals and attribution instead of salience maps (input gradients) to determine local FIE. They also distractors to obfuscate the input data through which the feature importance estimates are derived.

Integrated Gradients [14] is a method that aggregates the measured input gradients between the given sample and so-called *baseline input*, which can be understood as a null input, like a black image or text vector with no words encoded. This requirement is usually tailored towards the visual domain, as rarely the baseline-input is hard to decide on.

ChatGPT explanations [50] , a very recent method developed only for language synthesis models. It uses the advanced ChatGPT-4 model to explain other language models in three simple steps. This is a method tailored specifically to these NLP models and included here mainly due to their recent success.

Chapter 3

Problem Statement

Ideally, a FIR approach should be able to:

1. Detect any functional dependence between input features and targets;
2. Rank the importance of all the selected features to reflect their contributions to the learning performance;
3. Preserve the detected functional dependence and the feature importance ranking in test/validation data.

From a more practical standpoint, it would be ideal if such a measure would return positive values for actual predictors and negative values for null (noisy) features. In this section, we will discuss many problems that arise when one tries to measure feature importance. We will start with conceptual obstacles and move towards more practical problems.

3.1 Defining Optimal Feature Subset Of Constant Size

Suppose $\mathcal{D} = \{\mathcal{X}, \mathcal{Y}\}$ is a dataset used for supervised learning. In this data set, $(\boldsymbol{x}, \boldsymbol{y})$ is a training example, where $\boldsymbol{x} \in \mathcal{X}$ is a vector of d features and $\boldsymbol{y} \in \mathcal{Y}$ is its corresponding target. Let $\boldsymbol{m} \in \mathcal{M}$ denote a d-dimensional binary mask vector of 0/1 elements, where $||\boldsymbol{m}||_0 = s$, $s < d$ and $|\mathcal{M}| = \binom{d}{s}$. Thus, we can use such a mask vector to indicate a feature subset: $\{\boldsymbol{x} \otimes \boldsymbol{m}\}_{\boldsymbol{x} \in \mathcal{X}}$, where $\otimes$ denotes Hadamard product that yield a subset of s features for any instance $\boldsymbol{x} \in \mathcal{X}$. Assume that $\mathcal{Q}(\boldsymbol{x}, \boldsymbol{m})$ quantifies the instance-level performance of an ideal learning system trained on $\mathcal{D}$ via a feature subset, $\{\boldsymbol{x} \otimes \boldsymbol{m}\}_{\boldsymbol{x} \in \mathcal{X}}$.

Let us define a value function $v : 2^d \rightarrow \mathbb{R}$ that assigns each of the masks $\boldsymbol{m}$ a scalar indicating the population-wise performance of a subset of features.

$$v(\boldsymbol{m}) = \sum_{\boldsymbol{x} \in \mathcal{X}} \mathcal{Q}(\boldsymbol{x}, \boldsymbol{m}). \tag{3.1}$$

The optimal subset $\boldsymbol{m}^*$ can be then formulated as follows:

$$\boldsymbol{m}^* = \underset{\boldsymbol{m} \in \mathcal{M}}{\operatorname{argmax}}\, v(\boldsymbol{m}). \tag{3.2}$$

3.2 Defining Optimal Feature Subset Of Variable Size

For an optimal feature subset of any size, we can relax the condition $||\boldsymbol{m}||_0 = s$ to $0 \geq ||\boldsymbol{m}||_0 \geq d$ which changes the number of possible feature subsets from $|\mathcal{M}| = \binom{d}{s}$ to power set of all the features $|\mathcal{M}| = 2^d \ = \sum_{i=0}^{d} \binom{d}{i}$.
In that case, it is worth noticing that Equation 3.2 is poorly formed as an ideal model should be able to handle any number of noisy features. That is why we added a regularization term to the value function v to make the optimal subset unique:

$$v_\lambda(\boldsymbol{m}, \lambda) = \sum_{\boldsymbol{x} \in \mathcal{X}} \mathcal{Q}(\boldsymbol{x}, \boldsymbol{m}) - \lambda * ||\boldsymbol{m}||_0 = v(\boldsymbol{m}) - \lambda * ||\boldsymbol{m}||_0, \tag{3.3}$$

where $\lambda > 0$ is a regularization factor. The uniqueness arises from the penalization of subsets with null features, efficiently reducing the optimal solution to the one without any null features.
Notice that for real-world scenarios we do not have access to infinite data, which adds additional meaning to λ and is further discussed in Section 3.4.

3.3 Defining Feature Importance

Most of today's methods overlook the associations between the features when discussing the feature selection / feature importances, as [9] says: *"Most existing algorithms of feature selection (...) assume that data is independent"*. In our opinion, this is a mistake which allowed our models to attain better results than other methods.
If one wants to include feature associations in their work, it forces measuring the feature importance of a given feature $f \in \mathcal{F}$, where $\mathcal{F}$ is the full set of features,

only when considering it as a part of a subset $\mathcal{S} \subset \mathcal{F}$ (for now, let us **not** make an assumption that $f \in \mathcal{S}$).

The motivation behind that reasoning is the fact that the other features from $\mathcal{S}$ might have redundant or synergistic information when compared with f. That makes it clear that our definition of feature importance will also depend on the subset of features $\mathcal{S}$ (or for our models mask $\boldsymbol{m}$) in order to correctly incorporate the interactions between the feature f and features from subset $\mathcal{S}$.

Notice that the subset $\mathcal{S}$ can also be defined by a mask $\boldsymbol{m}$ and in this work we will often use these two symbols alternatively (in order to facilitate a more understandable notation of adding/removing a feature from a feature subset).

The main motivation behind choosing the following feature importance metric is the fact that feature selection is a performance-driven process: in the end, *the user wants to achieve the best possible performance by not using the noisy features and finding out which of the used features are the ones bringing the most value to the model.* In other words, the optimal subset should achieve the best performance and feature importance should vary according to the performance of a subset with and without the feature.

That is why we define the feature importance as follows:

$$\phi(\mathcal{S}, f) = \begin{cases} v(\mathcal{S} \cup \{f\}) - v(\mathcal{S}) & \text{if } f \notin \mathcal{S}, \\ v(\mathcal{S}) - v(\mathcal{S} \setminus \{f\}) & \text{otherwise.} \end{cases} \tag{3.4}$$

Let $\phi(\mathcal{S}, f)$ return the importance of feature f when measured in tandem with the feature subset $\mathcal{S}$. For consistency, let $\Phi(\mathcal{S}, \mathcal{F})$ quantify the importance vector of all the features in $\mathcal{F}$. When analysing the formula we can notice that, for ideal models, $\phi(\mathcal{S}, f) > 0$ for a feature f that brings predictive power to the model when considered with the features already present in $\mathcal{S}$.

Let us check what that would be the value of feature importance for a noisy feature f_{noise} when using an ideal model. In that case, they would have a feature importance score $\phi(\mathcal{S}, f_{noise}) = 0$, no matter the $\mathcal{S}$, as an ideal model would not be affected by additional, noisy features. As it stands, we rarely have access to ideal models and we must do with models whose **validation/test** performance deteriorates when they are given noisy features (the opposite effect happens for the training data). If that assumption holds, then $\phi(\mathcal{S}, f_{noise}) < 0$, again no matter the $\mathcal{S}$, if we adjust correctly the v function (Eq. 3.1) to use validation data subset for measuring subset performance (while the model weights are being tuned on training data).

To sum up, if measured correctly, the feature importance measure $\phi(\mathcal{S}, f)$ should be positive for features which hold predictive power towards the target and negative for null (noise) features. That also means that for an optimal subset $\boldsymbol{m}^*$ (and corresponding $\mathcal{S}^*$) the following condition must hold:

$$\phi(\mathcal{S}^*, f) \begin{cases} \leq 0 & \text{if } f \notin \mathcal{S}^*, \\ > 0 & \text{otherwise.} \end{cases} \tag{3.5}$$

Unfortunately, in the real world, even noisy features can sometimes have a nominal positive influence on the test performance which is due to finite test data. A way to counteract this effect is discussed in the next section.

3.4 Regularizing The Feature Subsets

Most scientific research often uses CI to produce the probability of a true value of an estimate lying between the given interval. The width of such an interval is inversely proportional to the number of samples used for a given estimate. For example, for linear scenarios, correlation is a perfect measure of variable dependence. Even if the true value of correlation between two independent variables is 0, the estimate of correlation will vary for a finite sample size. Then, a trained linear model like OLS can use the perceived correlation to make predictions about one variable using the values of another. Moreover, if the direction of the estimated correlation is the same for validation data, removing the first variable from the feature set will actually result in a drop in validation performance. This would result in positive feature importance for a variable that is independent of the target according to our definition of feature importance (Eq. 3.4). To sum up, even for an ideal model for a given scenario, it is possible to measure positive feature importance for a null feature due to a finite sample size of a dataset.

To counteract this effect, we can use the λ parameter in Equation 3.3 to set the minimum level of contribution level per feature towards the value of its importance for it to be qualified as a not-null feature. If we use v_λ instead of v in Equation 3.4:

$$\phi_\lambda(\mathcal{S}, f, \lambda) = \begin{cases} v_\lambda(\mathcal{S} \cup \{f\}, \lambda) - v_\lambda(\mathcal{S}, \lambda) & \text{if } f \notin \mathcal{S}, \\ v_\lambda(\mathcal{S}, \lambda) - v_\lambda(\mathcal{S} \setminus \{f\}, \lambda) & \text{otherwise.} \end{cases} \tag{3.6}$$

and after using Eq. 3.3 to expand v_λ we are left with:

$$\phi_\lambda(\mathcal{S}, f, \lambda) = \begin{cases} v(\mathcal{S} \cup \{f\}) - v(\mathcal{S}) - \lambda & \text{if } f \notin \mathcal{S}, \\ v(\mathcal{S}) - v(\mathcal{S} \setminus \{f\}) - \lambda & \text{otherwise.} \end{cases} \tag{3.7}$$

which we can shorten to:

$$\phi_\lambda(\mathcal{S}, f, \lambda) = \phi(\mathcal{S}, f) - \lambda. \tag{3.8}$$

Keep in mind our earlier motivations: we want the feature importance score to be positive when the feature brings predictive power and negative, or equal to zero, otherwise. If we wish for a feature to be counted as a useful predictor using our adjusted feature importance measure then $\phi_\lambda(\mathcal{S}, f, \lambda) > 0$ and according to equation 3.8 that means $\phi(\mathcal{S}, f) > \lambda$. In other words, the regularization parameter λ corresponds to a minimum vanilla feature importance a feature must have to be counted as a useful predictor according to the new, adjusted measure. That gives us some flexibility when it comes to issues discussed at the beginning of this section, as we can adjust λ to make sure that accidental predictive power (which is measured according to $\mathcal{Q}$) has to be bigger than λ for a feature to be counted as 'useful'.

3.5 Defining Performance Metric

Continuing with our trend of moving to more and more real-world problems, another topic we have ignored for now is the exact process of calculating $v(\boldsymbol{m})$ and what is $\mathcal{Q}$ from Equation 3.1. The only thing previously mentioned regarding this issue was in Section 3.3, where we discussed using validation data to circumvent the fact that noisy features improve **training** performance of a model.

A good candidate for $\mathcal{Q}$ can be a function returning values inversely-proportional to the loss of a model. This measure seems viable as:

1. Loss is the definition of performance, and we want our feature importance measurements to be performance-driven,

2. Loss function is always chosen with respect to the data and the type of analysis: it changes whether we want to use regression, binary classification, multi-label classification,

3. It is already calculated by DL models for training purposes, so reusing it for feature importance scores will save time and reduce model complexity.

To make the chosen $\mathcal{Q}$ inversely proportional to the loss function, we decided to define $\mathcal{Q}$ as the opposite of the loss (other possibilities include reciprocals or more exotic functions, which usually must be tuned for a given loss due to the spread of possible loss values across different problems).

3.6 Other Problems

It is worth noting that finding an optimal subset is NP-hard, mainly due to the fact that each subset needs to be investigated on its own in order to take into account all possible feature associations. This is especially prevalent for deep learning models. To accurately measure the performance of a subset of features, one needs to retrain models [17] instead of just setting them to some constant value in order to accommodate for changing the distributions of input data. Even by today's standards, retraining neural networks 2^d times for each possible feature subset for a dataset with tens of features would take a prohibitively long time.
Additionally, even though the feature selection measure that we proposed in Equation 3.8 seems like a comprehensive take on feature importance, it still lacks the ability to explain some of the more complex feature associations (with more than 2 variables).
The final problem has to do with real-world data: for most of the datasets, we do not know the ground truth, as we simply do not always know which features are important. One can use their knowledge in a given field to validate the results produced by feature importance rankings or create an artificial dataset for which the ground truth is known. A common attempt in the former case is using pictorial data (like face classification) to inspect if the algorithm creates a visually compelling feature importance map, which is what we did for our work.

3.7 Group Feature Selection

Group feature selection was developed for problems when we can use our prior knowledge to group variables in sets that have to be taken as a whole for the final result. A good example of such a setting would be pixels in colourful images, where the RGB channels for each can be grouped. Another example would be grouping the momentum XYZ components together for analysis of physical phenomena.

In more formal language, instead of using $\mathcal{F} = \{f_0, f_1, f_2, ..., f_{d-1}\}$ for a dataset with d features, we have to assume that we have d groups $\mathcal{G} = \{G_0, G_1, G_2, ..., G_{d-1}\}$ where each group G_j is composed of d_j features: $G_j = \{f_{j,0}, f_{j,1}, f_{j,2}, ..., f_{j,d_j-1}\}$ and the total number of features in the dataset is $d_{total} = \sum_{j=0}^{d-1} d_j$.

Our task here is selecting an optimal subset of **groups** symbolized by $\boldsymbol{m}^*$ that is a d-dimensional vector of 0/1.

3.8 Feature Subset Encodings

The goal of producing feature subset encodings is to provide the user insights into the structure of variables in the dataset. Formally, let us define a similarity distance function $d_S : 2^d \times 2^d \to \mathbb{R}$ that produces a distance for a given pair of feature subsets:

$$d_S(\boldsymbol{m}_1, \boldsymbol{m}_2) = f_S(\{\mathcal{Q}(\boldsymbol{x}, \boldsymbol{m}_1)\}_{\boldsymbol{x}\in\mathcal{X}}, \{\mathcal{Q}(\boldsymbol{x}, \boldsymbol{m}_2)\}_{\boldsymbol{x}\in\mathcal{X}}) \tag{3.9}$$

where $f_S : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is a function measuring the statistical distance between two distributions, like Jensen-Shannon distance (square root of Jensen-Shannon divergence [51]).

The encodings would be k-dimensional vectors $\boldsymbol{z}$ produced for l sets of feature subsets $M = \{\boldsymbol{m}_0, \boldsymbol{m}_1, \boldsymbol{m}_2, ..., \boldsymbol{m}_{l-1}\}$ by a model f_z with weights φ:

$$\boldsymbol{z} = f_z(\varphi, \boldsymbol{m}), \tag{3.10}$$

so that the Euclidean distances between them would try to conserve the similarity distance measured using function f_S between the vector of predictions. We can set l to be 2^d or corresponding to the feature powerset or any other set of subsets. As a reminder, the formula for Euclidean distance between two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ is defined as:

$$d_{EU}(\boldsymbol{a},\boldsymbol{b}) = ||\boldsymbol{a}-\boldsymbol{b}||_2. \tag{3.11}$$

Now, we can formulate the problem as an optimization problem where we are trying to minimize the following:

$$\min_{\varphi} \sum_{j=0}^{l} \sum_{i=0}^{l} \big(d_{EU}(f_z(\varphi,\boldsymbol{m}_i), f_z(\varphi,\boldsymbol{m}_j)) - d_S(\boldsymbol{m}_i,\boldsymbol{m}_j)\big)^2. \tag{3.12}$$

Chapter 4

Feature Importance Rankings With Constant Subset Size

This section includes the work done in the first half of the course and focuses on overcoming the problem stated in Section 3.1 as well as trying to find feature rankings; as they can be loosely understood as a type of feature importance that is defined in Section 3.3. The chapter describes the method used in our previous work [30] with **minimal** formatting alternations that the PhD thesis format necessitates. This approach does not solve all the problems stated in Chapter 3.

4.1 Model Description

To tackle the FIR problem stated in Eq.(3.2) effectively with three criteria described in Section 3.1, we propose a deep learning model of dual nets, *operator* and *selector*, as shown in Fig. 4.1.

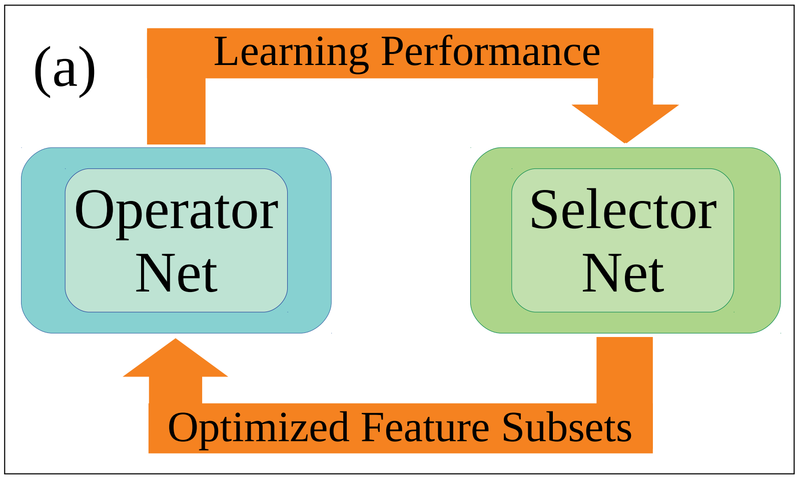


Figure 4.1: Dual-net architecture: workflow of the model.

The operator net is employed to accomplish a supervised learning task, e.g., classification or regression, on a given feature subset provided by the selector net, while the selector net is designated to learn finding out an optimal feature subset based on the performance feedback of the operator net working on optimal feature subset candidates during learning. Both the operator and the selector nets are trained jointly in an alternate manner (c.f. Section 4.2) to reach a synergy for the FIR.

Technically, the operator is carried out with a deep neural network parameterized with θ, $f_O(\theta; \boldsymbol{x}, \boldsymbol{m})$, for a given task, e.g., multi-layer perceptron (MLP) or convolution neural network (CNN). This net is trained on $\mathcal{D}$ based on different feature subsets to learn $f_O \colon \mathcal{X} \times \mathcal{M} \to \mathcal{Y}$. After learning (c.f. Section 4.2), the trained operator net, $f_O(\theta^*; \boldsymbol{x}, \boldsymbol{m}^*)$, is applied to the test data for prediction, where θ^* is the optimal parameters of the operator net and $\boldsymbol{m}^*$ is generated. During the training process, the selector net helps to iteratively provide candidate feature subsets.

In our method, the selector is implemented with an MLP parameterized with φ, $f_S(\varphi; \boldsymbol{m})$. As defined in Eq.(3.2), a selected optimal feature subset should maximize the averaging performance of the operator quantified by $\mathcal{Q}(\boldsymbol{x}, \boldsymbol{m})$ for all $\boldsymbol{x} \in \mathcal{X}$. Thus, we want the selector net to learn to predict the averaging performance of the operator net on different feature subsets; i.e., $f_S \colon \mathcal{M} \to \mathbb{R}$. After being trained properly (c.f. Section 4.2), we can use an algorithm working on the trained selector net of the optimal parameters φ^*, $f_S(\varphi^*; \boldsymbol{m})$, to generate an optimal feature subset indicated by $\boldsymbol{m}^*$ and rank feature importance to achieve $\Phi(\boldsymbol{m}^*)$ (c.f. Section 4.3).

4.2 Learning Algorithm

In essence, the FIR defined in Eq.(3.2) is a combinatorial optimization problem. According to the no free lunch theory for optimization [52], no algorithm can perform better than a random strategy in expectation in the setting of combinatorial optimization. Therefore, our learning algorithm is developed by leveraging learning with a stochastic local search procedure enhanced by injecting noise [53] on a small number of candidate feature subsets, $\mathcal{M}' \subset \mathcal{M}$, to avoid the exhaustive

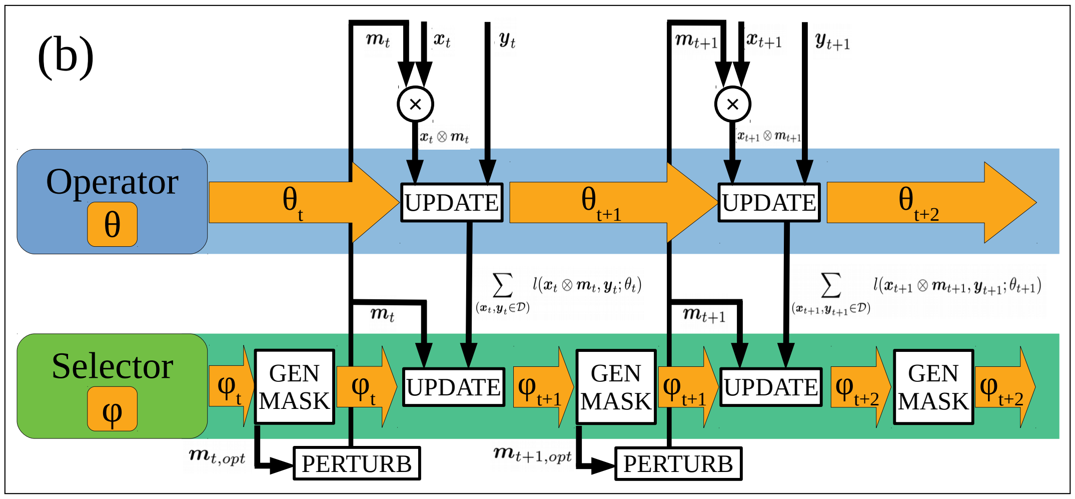

(b)
Operator
θ
Selector
φ
m_t
x_t
y_t
$x_t \otimes m_t$
UPDATE
GEN MASK
PERTURB
$m_{t,opt}$
$m_{t+1,opt}$
$\sum_{(x_t,y_t\in\mathcal{D})} l(x_t \otimes m_t, y_t; \theta_t)$
$\sum_{(x_{t+1},y_{t+1}\in\mathcal{D})} l(x_{t+1} \otimes m_{t+1}, y_{t+1}; \theta_{t+1})$

Figure 4.2: Dual-net architecture: diagram showing parameter updates for each of the models.

search.

For a training data set, $\mathcal{D} = \{\mathcal{X}, \mathcal{Y}\} = \{(\boldsymbol{x}, \boldsymbol{y})\}_{\boldsymbol{x}\in\mathcal{X},\boldsymbol{y}\in\mathcal{Y}}$, a mask subset, $\mathcal{M}'$, converts each training example $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{D}$ into $|\mathcal{M}'|$ examples: $\{(\boldsymbol{x}\otimes\boldsymbol{m}, \boldsymbol{y})\}_{\boldsymbol{m}\in\mathcal{M}'}$. Thus, the loss functions on $\mathcal{M}'$ (changing during learning) for the operator and the selector nets are defined respectively as follows:

$$\mathcal{L}_O(\mathcal{D}, \mathcal{M}'; \theta) = \frac{1}{|\mathcal{M}'||\mathcal{D}|} \sum_{\boldsymbol{m}\in\mathcal{M}'} \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} l(\boldsymbol{x}\otimes\boldsymbol{m}, \boldsymbol{y}; \theta), \tag{4.1a}$$

$$\mathcal{L}_S(\mathcal{M}'; \varphi) = \frac{1}{2|\mathcal{M}'|} \sum_{\boldsymbol{m}\in\mathcal{M}'} \Big(f_S(\varphi; \boldsymbol{m}) - \frac{1}{|\mathcal{D}|} \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} l(\boldsymbol{x}\otimes\boldsymbol{m}, \boldsymbol{y}; \theta)\Big)^2. \tag{4.1b}$$

Here, $l(\boldsymbol{x}\otimes\boldsymbol{m}, \boldsymbol{y}; \theta)$ is an instance-level cross-entropy/categorical cross-entropy loss for binary/multi-class classification or the mean square error (MSE) loss for regression. In Eq.(4.1b), we utilize the loss of the operator net, $l(\boldsymbol{x}\otimes\boldsymbol{m}, \boldsymbol{y}; \theta)$, to characterize its learning performance, $\mathcal{Q}(\boldsymbol{x}, \boldsymbol{m})$, since maximizing $\mathcal{Q}(\boldsymbol{x}, \boldsymbol{m})$ is equivalent to minimizing $l(\boldsymbol{x}\otimes\boldsymbol{m}, \boldsymbol{y}; \theta)$.

As described in Section 4.1, during learning, the operator net relies on the mask generator to provide an optimal subset of masks (through guidance of the selector net), $\mathcal{M}'$, indicating different optimal feature subset candidates, while the selector net requires the performance feedback from the operator net, $l(\boldsymbol{x}\otimes\boldsymbol{m}, \boldsymbol{y}; \theta)$ for all $\boldsymbol{m} \in \mathcal{M}'$. Two nets in our learning model hence have to be trained alternately. Below, we present the main learning steps in our learning algorithm of two phases, while the pseudo-code can be found in Section 4.4.

Phase I: Initial Operator Learning via Exploration. At first, we start training the operator net by using a small number of random feature subsets for several epochs until it can yield different performances on different feature subsets stably. Technically, in each epoch, we randomly draw a subset of different masks, $\mathcal{M}_1'$, from $\mathcal{M}$; i.e., $\mathcal{M}_1' = \{\boldsymbol{m}_i | \boldsymbol{m}_i = \text{Random}(\mathcal{M}, s)\}_{i=1}^{|\mathcal{M}'|}$, where Random$(\mathcal{M}, s)$ is a function that randomly draws a d-dimensional mask of s one-elements and $d-s$ zero-elements from $\mathcal{M}$. If θ is trained by stochastic gradient decent (SGD), then it is updated by $\theta'' \triangleq \theta' - \eta \nabla_\theta \mathcal{L}_O(\mathcal{D}, \mathcal{M}_1'; \theta)|_{\theta=\theta'}$[1] where η is a learning rate. After E_1 epochs, we set $\theta_1 = \theta''(E_1)$ and $\boldsymbol{m}_{1,opt}' = \text{argmin}_{\boldsymbol{m} \in \mathcal{M}_1'} \sum_{(\boldsymbol{x},\boldsymbol{y}) \in \mathcal{D}} l(\boldsymbol{x} \otimes \boldsymbol{m}, \boldsymbol{y}; \theta_1)$ to be used at the beginning of Phase II-A; i.e., $t = 1$ as shown in Fig. 4.2.

Phase II-A: Selector Learning via Operator's Feedback. As illustrated in Fig. 4.2, the operator provides training examples for the selector at step t: $\{(\boldsymbol{m}, \frac{1}{|\mathcal{D}|} \sum_{(\boldsymbol{x},\boldsymbol{y}) \in \mathcal{D}} l(\boldsymbol{x} \otimes \boldsymbol{m}, \boldsymbol{y}; \theta_t))\}_{\boldsymbol{m} \in \mathcal{M}_t'}$. By using the SGD with initializing φ_1 randomly, the parameters in the selector net, φ, are updated by $\varphi_{t+1} \triangleq \varphi_t - \eta \nabla_\varphi \mathcal{L}_S(\mathcal{M}_t'; \varphi)|_{\varphi=\varphi_t}$. Then, we adopt an *exploration-exploitation* strategy to generate a new mask subset, $\mathcal{M}_{t+1}'$, for the operator learning at step $t+1$. Thus, $\mathcal{M}_{t+1}'$ is divided into two mutually exclusive subsets: $\mathcal{M}_{t+1}' = \mathcal{M}_{t+1,1}' \cup \mathcal{M}_{t+1,2}'$. Motivated by the role of noise in stochastic local search [53], $\mathcal{M}_{t+1,1}'$ is generated via exploration to avoid overfitting: $\mathcal{M}_{t+1,1}' = \{\boldsymbol{m}_i | \boldsymbol{m}_i = \text{Random}(\mathcal{M}, s)\}_{i=1}^{|\mathcal{M}_{t+1,1}'|}$. Motivated by the input gradient idea [47], $\mathcal{M}_{t+1,2}'$ is generated by exploitation of the selector net, $f_S(\varphi_{t+1}; \boldsymbol{m})$, as follows:

a) **Generation of an optimal subset**. Starting with d-dimensional $\boldsymbol{m}_0 = (\frac{1}{2}, \cdots, \frac{1}{2})$, meaning that every feature has an equal chance to be selected, we have $\boldsymbol{\delta}_{\boldsymbol{m}_0} = \frac{\partial f_S(\varphi_{t+1}; \boldsymbol{m})}{\partial \boldsymbol{m}}|_{\boldsymbol{m}=\boldsymbol{m}_0}$. As input features of the larger gradients contribute more to the learning performance of the operator, we can find top s features based on their gradients by $(\boldsymbol{m}_{opt}, \bar{\boldsymbol{m}}_{opt}) = \text{argsort}(\boldsymbol{\delta}_{\boldsymbol{m}_0}, s)$ where $\boldsymbol{m}_{opt}$ is the mask to indicate top s features and $\bar{\boldsymbol{m}}_{opt}$ is the mask for the remaining $d-s$ features. To ensure the optimality of $\boldsymbol{m}_{opt}$, we come up with a three-step *validation* procedure:

[1] Parameters are actually updated on a batch $\mathcal{B}$ randomly drawn from $\mathcal{D}$, hence $\frac{|\mathcal{D}|}{|\mathcal{B}|}$ times in an epoch.

i) Re-evaluate the contributions of top s features by $(\boldsymbol{m}_{opt}, \bar{\boldsymbol{m}}_{opt}) = \text{argsort}(\boldsymbol{\delta}_{\boldsymbol{m}_{opt}}, s)$ where $\boldsymbol{\delta}_{\boldsymbol{m}_{opt}} = \frac{\partial f_S(\varphi_{t+1};\boldsymbol{m})}{\partial \boldsymbol{m}}|_{\boldsymbol{m}=\boldsymbol{m}_{opt}}$;

ii) Replace a feature of negative gradient in $\boldsymbol{m}_{opt}$ with the one of the largest gradient in $\bar{\boldsymbol{m}}_{opt}$ if there exists;

iii) Further, check the optimality via a function $(\boldsymbol{m}'_{opt}, \bar{\boldsymbol{m}}'_{opt}) = \text{swap}(\boldsymbol{m}_{opt}, \bar{\boldsymbol{m}}_{opt})$ that yields $\boldsymbol{m}'_{opt}$ by swapping between the feature of least gradient in $\boldsymbol{m}_{opt}$ and the one of the largest gradient in $\bar{\boldsymbol{m}}_{opt}$. Repeat (i)-(iii) until $f_S(\varphi_{t+1};\boldsymbol{m}_{opt}) \leq f_S(\varphi_{t+1};\boldsymbol{m}'_{opt})$. After going through the validation procedure, $\boldsymbol{m}_{t+1,opt}$ is obtained for step $t+1$.

b) Generation of optimal subset candidates via perturbation. As the optimal subset $\boldsymbol{m}_{t+1,opt}$ might be a local optimum, we would further inject noise to generate more optimal subset candidates by a perturbation function $\text{Perturb}(\boldsymbol{m}_{opt}, s_p)$. For $s_p < s$, $\text{Perturb}(\boldsymbol{m}_{opt}, s_p)$ randomly flips s_p different elements in $\boldsymbol{m}_{opt}/\bar{\boldsymbol{m}}_{opt}$ from 1/0 to 0/1 and swaps between changed elements in $\boldsymbol{m}_{opt}$ and $\bar{\boldsymbol{m}}_{opt}$. Applying $\text{Perturb}(\boldsymbol{m}_{opt}, s_p)$ repeatedly leads to multiple optimal subset candidates;

c) Formation of optimal subset candidates. Assembling **a)** and **b)** leads to $\mathcal{M}'_{t+1,2} = \{\boldsymbol{m}_{t,best}\} \cup \{\boldsymbol{m}_{t+1,opt}\} \cup \big\{\boldsymbol{m}_i | \boldsymbol{m}_i = \text{Perturb}(\boldsymbol{m}_{t+1,opt}, s_p)\big\}_{i=1}^{|\mathcal{M}'_{t+1,2}|-2}$. Here, we always include $\boldsymbol{m}_{t,best}$, the subset that leads to the best learning performance of the operator net in the last step (step t), as the most important subset candidate in the current step (step $t+1$) in order to make the operator learning progress steadily. Note that $\boldsymbol{m}_{t,best}$ may not be $\boldsymbol{m}_{t,opt}$.

Phase II-B: Operator Learning via Optimal Subset Candidates from Selector. After completing the training of Phase II-A at step t , the selector net provides the optimal subset candidates, $\mathcal{M}'_{t+1} = \mathcal{M}'_{t+1,1} \cup \mathcal{M}'_{t+1,2}$, for the operator net, as illustrated in Fig. 4.2. At step $t+1$, the operator net is thus trained based on $\mathcal{M}'_{t+1}$ with SGD: $\theta_{t+1} \triangleq \theta_t - \eta \nabla_\theta \mathcal{L}_O(\mathcal{D}, \mathcal{M}'_{t+1}; \theta)|_{\theta=\theta_t}$.

As shown in Fig. 4.1, our alternate algorithm enables the operator and the selector nets to be trained jointly in Phase II until a pre-specified stopping condition is satisfied.

4.3 Deployment

After the learning described in Section 4.2 is accomplished, we obtain the optimal parameters of the operator and the selector nets, θ^* and φ^*.

By using the trained selector net, $f_S(\varphi^*; \boldsymbol{m})$, we find out an optimal feature subset with the same procedure used in Phase II-A as follows:

1. starting with $\boldsymbol{m}_0 = \left(\frac{1}{2}, \cdots, \frac{1}{2}\right)$, calculate the gradient $\boldsymbol{\delta}_{\boldsymbol{m}_0} = \frac{\partial f_S(\varphi^*; \boldsymbol{m})}{\partial \boldsymbol{m}}|_{\boldsymbol{m}=\boldsymbol{m}_0}$ (we chose $\frac{1}{2}$ as starting value as it is in the middle between two possible values of 0 and 1);

2. finding top s features by $(\boldsymbol{m}^*, \bar{\boldsymbol{m}}^*) = \text{argsort}(\boldsymbol{\delta}_{\boldsymbol{m}_0}, s)$, where $\boldsymbol{m}^*$ indicates the optimal subset of top s features;

3. going through the validation procedure described in Phase II.A to ensure the optimality of $\boldsymbol{m}^*$. Thus, feature importance ranking on the final $\boldsymbol{m}^*$ is done by setting $\Phi(\boldsymbol{m}^*) = \frac{\partial f_S(\varphi^*; \boldsymbol{m})}{\partial \boldsymbol{m}}|_{\boldsymbol{m}=\boldsymbol{m}^*}$ and sorting the input gradients of selected features.

During testing, for a test instance, $\hat{\boldsymbol{x}}$, the trained operator net, $f_O(\theta^*; \boldsymbol{x}, \boldsymbol{m})$, can be used to make a prediction, $f_O(\theta^*; \hat{\boldsymbol{x}}, \boldsymbol{m}^*)$, via $\hat{\boldsymbol{x}} \otimes \boldsymbol{m}^*$, which allows a supervised learning task to be done based on only the optimal feature subset, $\boldsymbol{m}^*$, found out with our proposed approach.

4.4 Pseudo Code

In this section, we describe the implementation of our alternate learning algorithm used to train our proposed dual-net neural architecture for feature importance ranking underlying feature selection. The pseudo code[2] in Algorithm 1 carries out the alternate learning algorithm as described in Section 4.2 . The pseudo code in Algorithm 2 implements a subroutine used in line 10 of Algorithm 1 to

[2]Our source code and all the related information regarding the experimental settings are available online: https://github.com/maksym33/FeatureImportanceDL.

generate an optimal feature subset in the current step as described in Phase II.A (c.f. Section 4.2). In Algorithm 1, lines 1-7 corresponds to Phase I, lines 9-12 carry out Phase II.A and lines 13-18 implement Phase II.B. Phase II.A and II.B alternate until the early stopping condition is satisfied as implemented by the loop from line 8 to line 23.

Algorithm 1 Alternate Learning Algorithm

```
Require: loss function of operator net, $l(\boldsymbol{x} \otimes \boldsymbol{m}, \boldsymbol{y}; \theta)$, selector net, $f_S(\varphi, \boldsymbol{m})$
Require: feature set/subset size $d$ and $s$, fraction of random masks $f$, perturbation factor $s_p$
Require: number of optimal subset candidates $|\mathcal{M}'|$, number of batches $E_1$ in Phase I.
Require: mask weight vector used in the weighted selector loss $\boldsymbol{w}_S$ of $|\mathcal{M}'|$ elements.
 1: for $e \leftarrow 0$ to $E_1$ do
 2:     $\mathcal{M}'_1 = \{\boldsymbol{m}_i | \boldsymbol{m}_i = \text{Random}(\mathcal{M}, s)\}_{i=1}^{|\mathcal{M}'|}$   ▷ create a random batch of masks
 3:     $\mathcal{L}_O(\mathcal{D}, \mathcal{M}'_1; \theta) = \frac{1}{|\mathcal{M}'||\mathcal{D}|} \sum_{\boldsymbol{m} \in \mathcal{M}'} \sum_{(\boldsymbol{x},\boldsymbol{y}) \in \mathcal{D}} l(\boldsymbol{x} \otimes \boldsymbol{m}, \boldsymbol{y}; \theta)$   ▷ calculate operator loss
 4:     $\theta'' \triangleq \theta' - \eta \nabla_\theta \mathcal{L}_O(\mathcal{D}, \mathcal{M}'_1; \theta)|_{\theta=\theta'}$   ▷ update the parameters in operator
 5:     $\mathcal{L}_S(\mathcal{M}'; \varphi) = \frac{1}{2|\mathcal{M}'|} \sum_{\boldsymbol{m} \in \mathcal{M}'} \Big( f_S(\varphi; \boldsymbol{m}) - \frac{1}{|\mathcal{D}|} \sum_{(\boldsymbol{x},\boldsymbol{y}) \in \mathcal{D}} l(\boldsymbol{x} \otimes \boldsymbol{m}, \boldsymbol{y}; \theta) \Big)^2$   ▷ calculate MSE loss of selector net
 6:     $\varphi'' \triangleq \varphi' - \eta \nabla_\varphi \mathcal{L}_S(\mathcal{M}'_1; \varphi)|_{\varphi=\varphi'}$   ▷ update the weights in selector
 7: end for
 8: for $t \leftarrow 0$ to inf do
 9:     $\mathcal{M}'_{t+1,1} = \{\boldsymbol{m}_i | \boldsymbol{m}_i = \text{Random}(\mathcal{M}, s)\}_{i=1}^{(1-f)|\mathcal{M}'|}$   ▷ create a random batch of masks
10:     $m_{t+1,opt} \Leftarrow \mathit{generateOptimalMask}(f_{SN}(\varphi_t))$   ▷ implemented in Algorithm 2
11:     $\mathcal{M}'_{t+1,2} = \{\boldsymbol{m}_{t,best}\} \cup \{\boldsymbol{m}_{t+1,opt}\} \cup \{\boldsymbol{m}_i | \boldsymbol{m}_i = \text{Perturb}(\boldsymbol{m}_{t+1,opt}, s_p)\}_{i=1}^{f|\mathcal{M}'|-2}$
        ▷ collect the best mask from last step, the current optimal mask and those perturbed optimal masks
12:     $\mathcal{M}'_{t+1} = \mathcal{M}'_{t+1,1} \cup \mathcal{M}'_{t+1,2}$   ▷ form new subset candidates for operator
13:     $\mathcal{L}_O(\mathcal{D}, \mathcal{M}'_1; \theta) = \frac{1}{|\mathcal{M}'||\mathcal{D}|} \sum_{\boldsymbol{m} \in \mathcal{M}'} \sum_{(\boldsymbol{x},\boldsymbol{y}) \in \mathcal{D}} l(\boldsymbol{x} \otimes \boldsymbol{m}, \boldsymbol{y}; \theta_t)$   ▷ calculate operator loss
14:     $\theta_{t+1} \triangleq \theta_t - \eta \nabla_\theta \mathcal{L}_O(\mathcal{D}, \mathcal{M}'_1; \theta)|_{\theta=\theta}$   ▷ update parameters in operator
15:     $\mathcal{L}_S(\mathcal{M}'; \varphi) = \frac{1}{2|\mathcal{M}'|} \sum_{i=0}^{|\mathcal{M}'|} w_{S,i} \Big( f_S(\varphi; \boldsymbol{m}) - \frac{1}{|\mathcal{D}|} \sum_{(\boldsymbol{x},\boldsymbol{y}) \in \mathcal{D}} l(\boldsymbol{x} \otimes \boldsymbol{m}, \boldsymbol{y}; \theta) \Big)^2$   ▷ calculate the weighted MSE loss of selector
16:     $\varphi_{t+1} \triangleq \varphi_t - \eta \nabla_\varphi \mathcal{L}_S(\mathcal{M}'_1; \varphi)|_{\varphi=\varphi_t}$   ▷ update the parameters in selector
17:     $\boldsymbol{m}_{t+1,best} = \text{argmin}_{\boldsymbol{m} \in \mathcal{M}'_{t+1}} \Big( \sum_{(\boldsymbol{x},\boldsymbol{y}) \in \mathcal{D}} l(\boldsymbol{x} \otimes \boldsymbol{m}, \boldsymbol{y}; \theta) \Big)$   ▷ record the best performed mask
18:     $\mathcal{L}_{t,m_{opt}} \leftarrow \big( \sum_{(\boldsymbol{x},\boldsymbol{y}) \in \mathcal{D}} l(\boldsymbol{x} \otimes \boldsymbol{m}, \boldsymbol{y}; \theta) \big)[(1-f)+1]$   ▷ record loss of the $\boldsymbol{m}_{opt}$, $(1-f)+1$ should be its index
19:     if $\mathit{checkEarlyStopping}(\mathcal{L}_{t,m_{opt}})$ then
20:         $\theta_t = \mathit{restoreBestWeights}()$   ▷ stopping condition is met
21:         break
22:     end if
23: end for
```

Algorithm 2 Generation of Optimal Feature Subset

```
Require: selector net $f_S(\varphi, \boldsymbol{m})$
Require: input feature set size $d$, selected subset size $s$
 1: $\boldsymbol{m}_0 \leftarrow (\frac{1}{2}, \frac{1}{2}, ..., \frac{1}{2})$
 2: $\boldsymbol{\delta}_{m_0} = \frac{\partial f_S(\varphi, \boldsymbol{m})}{\partial \boldsymbol{m}}|_{\boldsymbol{m}=\boldsymbol{m}_0}$    ▷ calculate input gradient
 3: $\boldsymbol{m}_{opt} \leftarrow (0, 0, ..., 0)$
 4: $(\boldsymbol{i}_{unmasked}, \boldsymbol{i}_{masked}) \leftarrow \text{argsort}(\boldsymbol{\delta}_{m_0}))$   ▷ determine indexes with 1s (unmasked,
    top $s$ biggest gradients) and 0s (masked, the smallest gradients)
 5: $\boldsymbol{m}_{opt}[\boldsymbol{i}_{unmasked}] \leftarrow (1, 1, ..., 1)$    ▷ complete creating $\boldsymbol{m}_{opt}$
 6: $\boldsymbol{\delta}_{m_{opt}} = \frac{\partial f_{SN}(\varphi, \boldsymbol{m})}{\partial \boldsymbol{m}}|_{\boldsymbol{m}=\boldsymbol{m}_{opt}}$    ▷ reclalculate the gradient at $\boldsymbol{m} = \boldsymbol{m}_{opt}$
 7: $i_{min} \leftarrow \text{argmin}(\boldsymbol{\delta}_{m_{opt}}[i_{unmasked}])$    ▷ index of minimum unmasked gradient
 8: $i_{max} \leftarrow \text{argmax}(\boldsymbol{\delta}_{m_{opt}}[i_{masked}])$    ▷ index of maximum masked gradient
 9: $\boldsymbol{i}_{neg} \leftarrow \text{argwhere}(\boldsymbol{\delta}_{m_{opt}}[i_{unmasked}] < 0)$  ▷ create a set of unmasked indices that
    have negative gradients
10: for $i$ in $\boldsymbol{i}_{neg}$ do
11:     $\boldsymbol{m}'_{opt} \leftarrow \boldsymbol{m}_{opt}$    ▷ **Validation step 1**
12:     $\boldsymbol{m}'_{opt}[i] \leftarrow 0$    ▷ mask the negative, previously unmasked, input
13:     $\boldsymbol{m}'_{opt}[i_{max}] \leftarrow 1$    ▷ unmask the biggest (gradient-wise), previously masked
        input
14:     if $f_S(\varphi, \boldsymbol{m}''_{opt}) < f_S(\varphi, \boldsymbol{m}_{opt})$ then
15:         $\boldsymbol{m}_{opt} \leftarrow \boldsymbol{m}'_{opt}$    ▷ replace $\boldsymbol{m}_{opt}$ and restart the validation
16:         recalcualte $\boldsymbol{i}_{unmasked}$ and $\boldsymbol{i}_{masked}$
17:         goto step 6
18:     end if
19: end for
20: $\boldsymbol{m}''_{opt} \leftarrow \boldsymbol{m}_{opt}$    ▷ **Validation step 2**
21: $\boldsymbol{m}''_{opt}[i_{min}] \leftarrow 0$    ▷ mask the smallest (gradient-wise), previously unmasked,
    input
22: $\boldsymbol{m}''_{opt}[i_{max}] \leftarrow 1$    ▷ unmask the biggest (gradient-wise), previously masked
    input
23: if $f_S(\varphi, \boldsymbol{m}''_{opt}) < f_{SN}(\varphi, \boldsymbol{m}_{opt})$ then
24:     $\boldsymbol{m}_{opt} \leftarrow \boldsymbol{m}''_{opt}$    ▷ replace $\boldsymbol{m}_{opt}$ and restart the validation
25:     recalculate $\boldsymbol{i}_{masked}$ and $\boldsymbol{i}_{masked}$
26:     goto step 6
27: end if
```

Chapter 5

Experiments - Constant Subset Size

5.1 Results

In this section, we focus on presenting the results of our experiments. We divided the results/datasets into several major categories: synthetic, visual and benchmark. We decided to use synthetic data as this is the only type of dataset when the ground truth is known. We use visual datasets to present visually convincing results, that can be validated by any human. We chose to use some benchmark datasets to compare the performance of our trained Operator to the ones provided by other methods.

In this section, we will visit each of these data categories, where we will give a brief summary of the datasets in the category. Then, we will present our results and compare them to other methods if possible.

Finally, we summarize the experimental setup for our framework as well as the other methods.

5.1.1 Synthetic Data

Our first evaluation employs 3 synthetic datasets in literature [5, 54] for feature selection regarding regression and binary/multiclass classification as follows:

XOR as 4-way classification [5]. Group 8 corners of the cube, $(v_0, v_1, v_2) \in \{-1, +1\}^3$, by the tuples (v_0v_2, v_1v_2), leading to 4 sets of vectors paired with their negations $\{v^{(c)}, -v^{(c)}\}$. For a class c, a point is generated from the mixture distribution: $\frac{1}{2}[N(v^{(c)}, 0.5I_3) + N(-v^{(c)}, 0.5I_3)]$. Then, form a 10-D feature vector for each example by adding 7 standard noise features, $(X_3, \cdots, X_9) \sim N(0, I_7)$.

Nonlinear regression [5]. $Y = -2\sin(2X_0) + \max(X_1, 0) + X_2 + \exp(-X_3) + \epsilon$, where $(X_0, \cdots, X_9) \sim N(0, I_{10})$ and $\epsilon \sim N(0,1)$, leading to a 10-D feature vector for each example.

Binary classification [54]. To generate examples, set $Y = -1$ when $(X_0, \cdots, X_9) \sim N(0, I_{10})$ and $Y = +1$ when X_0 through X_3 are standard normal conditioned on $9 \leq \sum_{i=0}^{3} X_i^2 \leq 16$ and $(X_4, \cdots, X_9) \sim N(0, I_6)$, resulting in a 10-D feature vector for each example.

For each dataset, we randomly generate 512 and 1024 examples, respectively, for training and testing. With our problem formulation described in Section 3.1, our experiment on synthetic data simulates an application scenario that selects s out of d features where s is larger than the number of features relevant to the target in a dataset. As there are up to 4 relevant features in the above 3 datasets, we choose $s = 5$ in our experiment and compare with the following methods DFS [55], AvGrad [47], FS [54] based on MLP, LASSO [42], RF [8], RFE [2], BAHSIC [56, 41], mRMR [6] and CCM [5]. According to a taxonomy [4], DFS, AvGrad, RF and ours are embedding methods, FS is a wrapper method and all the others are filtering methods. For those filtering methods, we use the exact same kernel SVM/SVR described in those papers [2, 5, 6, 41, 56] and an MLP on LASSO for classification/regression. While DFS, AvGrad, LASSO and RF work on FIR for all 10 features, all other methods work with the same setting as ours by finding out the top 5 features and FIR.

Fig. 5.1 shows the feature selection and FIR results yielded by different methods regarding the top 5 features on 3 synthetic datasets where the FIE are normalized for each method (shown on the y-axis). In the case of filter methods, an equal FIE is set to all the features selected by those methods. Additionally, a grey rectangle is overlaid on each graph to mark the truly important features. We use blue bars to mark if the feature was selected in every CV fold of the analysis and red if it was not (stability analysis).

It is observed from Fig. 5.1 that our approach always finds out those relevant features in all 5 folds and does FIR properly by assigning negative scores (gradients), meaning unimportant, to irrelevant features. For the 4-way classification, DFS, RF, RFE, BAHSIC and CCM also find 3 relevant features in all 5 folds but others fail as shown in Fig. 5.1(a) although mRMR and CCM cannot yield FIR scores. In terms of accuracy, ours outperforms all other 9 methods despite the fact that DFS, AvGrad and RF work directly on the full feature set. For the nonlinear regression, FS, RF, RFE and CCM also select 4 relevant features in all 5 folds but ours yields the least MSE as shown in Fig. 5.1(b). For the binary classification, all the methods apart from LASSO find 4 relevant features in all 5 folds, as shown in Fig. 5.1(c). For this dataset, those state-of-the-art filtering methods yield better accuracy than others and the accuracy resulting from ours is slightly worse but comparable to those. In terms of FIR on all relevant features, ours is entirely consistent with those yielded by RF but performs significantly better than RF on 3 datasets. In comparison to the existing FIR methods for deep learning, ours always outperforms DFS, AvGrad and FS on 3 datasets in terms of both FIR and learning performance.

On the other hand, our method outputs negative importance for the regression task for the 2nd feature. This feature is useful only half of the time (corresponds to the term $\max X_1, 0)$ and it was perceived as deeply negative on one of the folds, resulting in an overall slightly negative score. It is worth pointing out that some of the other methods (RF, BAHSIC, AvGrad) also have trouble with correctly including this feature.

5.1.2 Benchmark Data

We further evaluate our approach on several well-known benchmark datasets from two different perspectives; i.e., explainability of FIR (MNIST) and learning performance on supervised feature selection (all).

MNIST Dataset

To demonstrate the explainability of FIR via visual inspection, we employ an MNIST[57] subset of hard-to-distinguish digits “3” and “8” for binary classification. The information on this subset is summarized in Table I. For comparison, we also apply 3 embedding methods, DFS, AvGrad and RF to this subset. To see the explainability of FIR, we adopt the same fully-connected MLP instead of CNN in DFS, AvGrad and the operator net in ours ($s = 85, d = 784$). The setting ensures that no other mechanisms like convolution/pooling layers can help a model automatically extract salient features for FIR. As a result, the accuracies yielded by DFS, AvGrad, RF and **ours** on the test data are $97.42 \pm 0.30\%$, $99.27 \pm 0.04\%$,

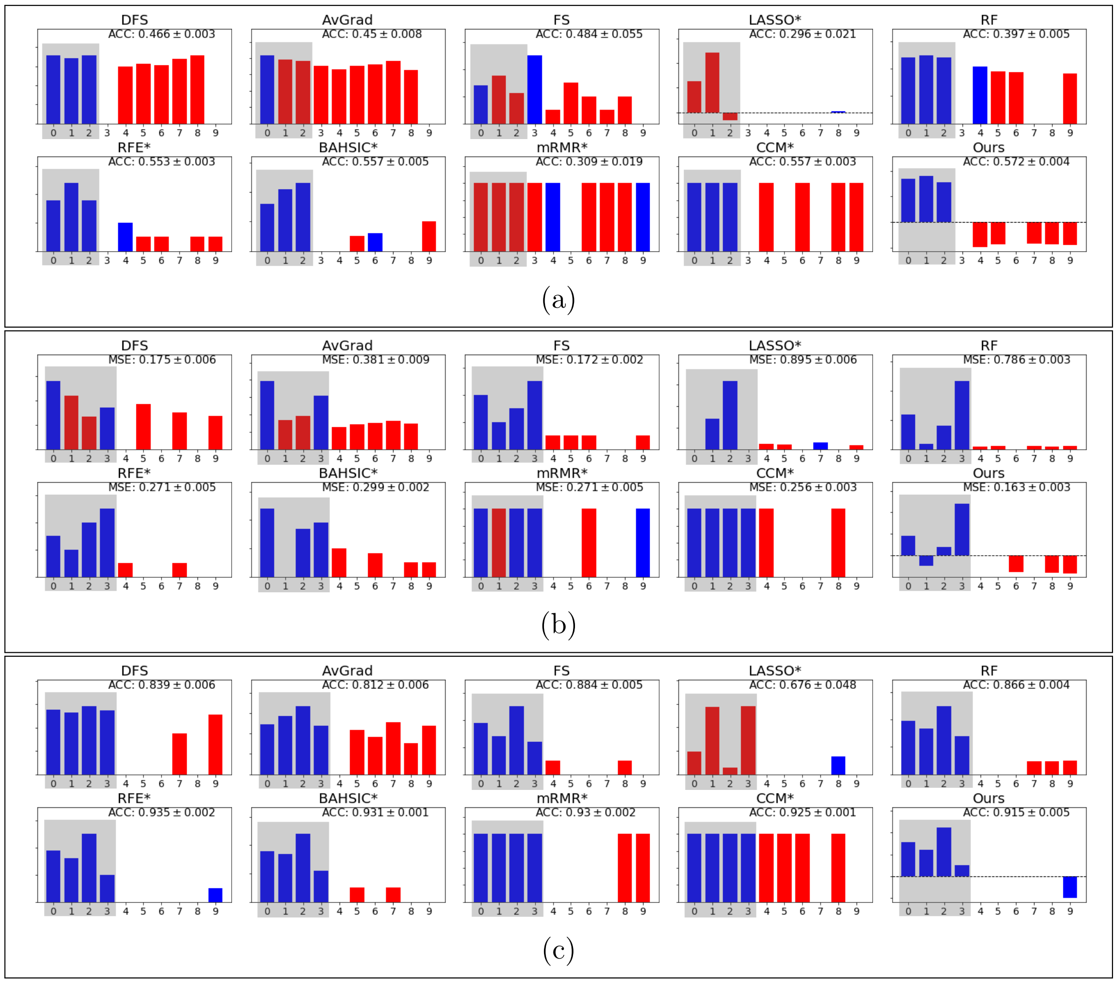


Figure 5.1: 5-fold cross-validation results (mean$\pm$std) on synthetic datasets ($s = 5, d = 10$). (a) XOR 4-way classification. (b) Nonlinear regression. (c) Binary classification. * refers to a filtering method, and blue/red colours indicate a feature selected in all 5 folds/fewer folds, respectively. Grey rectangles were overlaid on the important features.

$98.84 \pm 0.03\%$ and $\mathbf{99.31 \pm 0.08\%}$, respectively, where ours and DFS use 85 and 212 features, respectively, but AvGrad and RF need all 784 features. For visual inspection, which is described in more detail in Section 5.1.3 we normalize the FIR scores achieved by different methods to the same range and illustrate typical feature importance maps produced by 4 methods in a fold in Fig. 5.2. It is observed from Fig. 5.2(a),(b) that DFS and AvGrad, two FIR methods for deep learning, do not produce explainable maps. In contrast, it is evident from Fig. 5.2(d-f) that ours yields a meaningful map where those features (pixels) that distinguish between "3" and "8" images are vividly highlighted in terms of their importance. Again, ours yields a map similar to that of RF (c.f. Fig. 5.2(c)) but outperforms this off-the-shelf FIR method.

Table I: Information on benchmark and real-world datasets used in our experiments.

Data Set	MNIST	glass	vowel	TOX-171	yale	Enhancer–Promoter
#Features	784	10	10	5784	1024	102
#Classes	2	6	11	4	15	3
#Training	11,982	150	742	137	132	5,756
#Testing	1,984	64	248	34	33	2,878

Main Benchmark Data

In our comparative study, we choose four challenging benchmark datasets for feature selection evaluation. As reported in [5], the state-of-the-art feature selection methods, including the latest strong ones, do not perform well on the following datasets.

Glass dataset[1] The Glass is a famous UCI benchmark dataset for the task of predicting a type of glass based on its chemical composition. Glass dataset usually contains nine chemical features and the ID for each instance that is generally not treated as a feature. In the experimental setting of CCM [5], they treated the ID as a new feature, so ten features were used in their experiments. Due to the fact that the instances in the data file are arranged in a non-shuffled manner according to their class labels, the ID feature turns up to be one of the most important features so that CCM and other robust feature selection methods yield very high accuracy, e.g., CCM achieves 86% on average [5]. In our experiment, we follow this setting so that our approach yields 90%+ accuracy (c.f. Figure 4 in the main text). Without the ID feature, however, all the methods, including ours, yield considerably lower accuracies, although ours still outperforms those methods used in our comparative study. In the 5-fold cross-validation, the accuracy of our approach drops to the levels of 75%-80%, quite close to the known top accuracy of 80% on the OpenML platform[2].

Vowel Dataset[3] The Vowel is yet another famous UCI benchmark dataset for predicting English vowels from acoustic features. Following the same setting used in CCM [5], we use a newer version of this dataset so that we can make a fair

[1][online]: https://archive.ics.uci.edu/ml/machine-learning-databases/glass/
[2][online]: https://www.openml.org/t/3815
[3][online]: https://www.openml.org/d/307

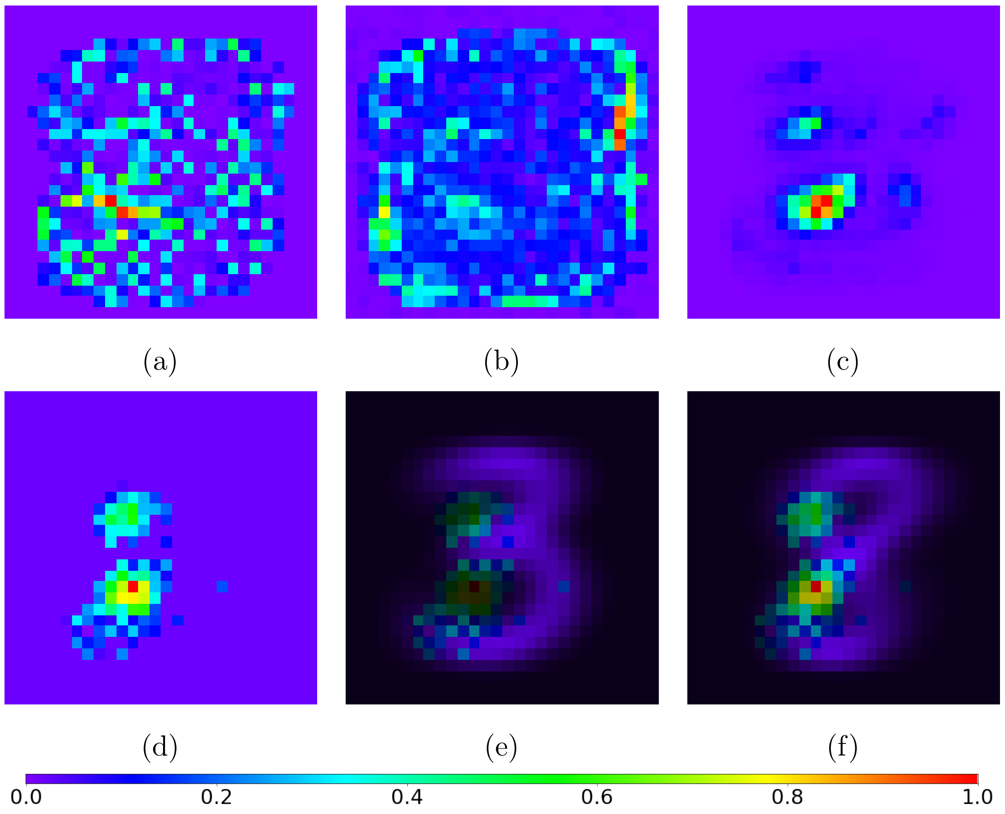


Figure 5.2: Feature importance maps yielded by different FIR methods. (a) DFS. (b) AvGrad. (c) RF. (d-f) Ours and our map superimposed on the mean images of “3” and “8”, respectively, for clarity.

comparison to those feature selection methods used in our comparative study.

TOX-171 dataset[4] The TOX-171 is a biological microarray dataset with only 43 instances/class but 5,784 features. The nature of this dataset makes a deep-learning model very prone to overfitting.

Yale Dataset[5] The YALE faces is a well-known facial image benchmark dataset. There are 15 individual subjects, and 11 images of different facial expressions, e.g., wink, happy and sad, were collected from each individual. When this dataset is used for face recognition, a random split of this dataset could lead to a certain degree of covariant shift; the instances in training and validation/test sets may be subject to different distributions, but their distributions conditional on the label are the same. This causes difficulty for all learning models without covariant shift adaptation.

Feature Selection Benchmark. We further conduct the evaluation in feature selection. As our approach has the same setting as used in the supervised feature

[4][online]: http://featureselection.asu.edu/old/datasets.php
[5][online]: http://www.cad.zju.edu.cn/home/dengcai/Data/FaceData.html

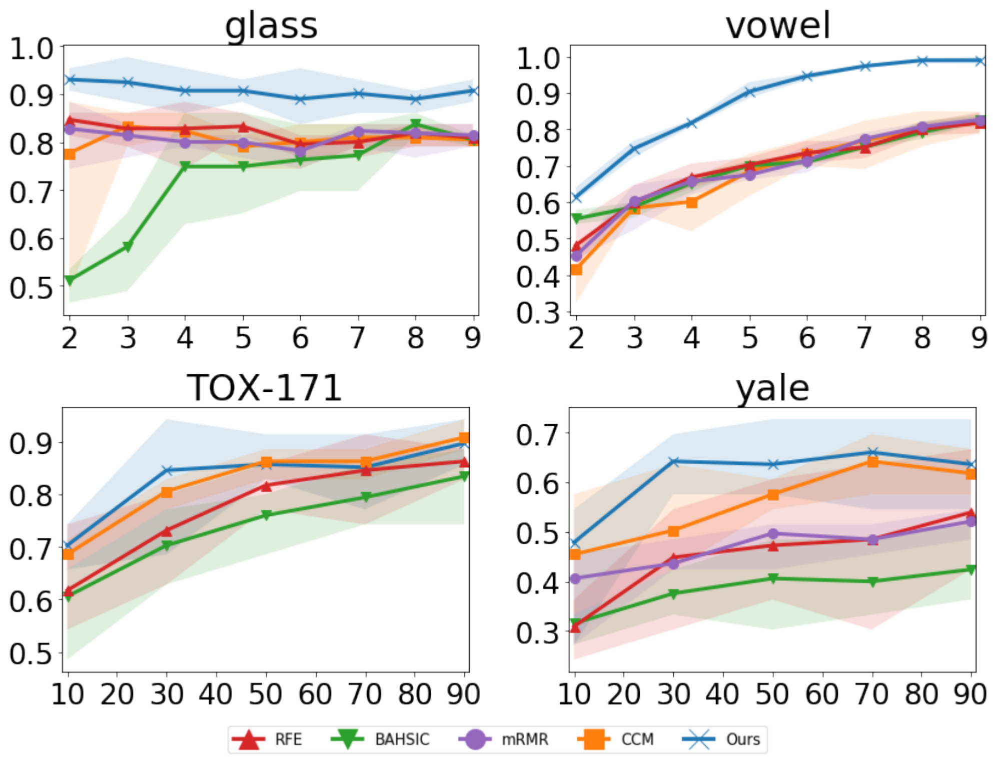


Figure 5.3: Classification accuracies (vertical axis) yielded by the supervised feature selection methods and ours for different numbers of selected features (horizontal axis) on 4 benchmark datasets.

selection methods, we compare ours to those strong supervised feature selection methods, RFE, BAHSIC, mRMR and CCM, on four benchmark datasets: `glass` [58], `vowel` [58], `TOX-171` [9] and `yale` [59], as summarized in Table I. For our model, we employ MLPs to implement the operator for `glass`, `vowel` and `TOX-171` but a CNN to carry out the operator for `yale` to demonstrate the flexibility of our dual-net architecture. By following the setting used in [5], we employ kernel SVMs for the classification on features selected by 4 filtering methods. It is evident from Fig. 5.3 that ours substantially outperforms all others on `glass`, `vowel` and `yale` with a large margin. Overall, ours yields results comparable to the strongest performer, CCM, on `TOX-171` where there are 5,700+ features but only 109 training examples for parameter estimate in each of 5 folds, which is very challenging for deep learning.

Enhancer–Promoter data

To evaluate our approach on real-world data, we adopt the GM12878 cell line (a lymphoblastoid cell line) dataset [55]. This is the dataset for which the *deep* *feature selection* (DFS) method [55] is especially proposed. Therefore, we follow their setting by using only annotated DNA regions of the GM12878 cell line (200 bp).

In the original dataset, there are 7 classes and 102 features, each class has 3,000 instances apart from one that has only 2878 instances. The 7 classes are ***active promoter, active enhancer***, *active exon, inactive promoter, inactive enhancer, inactive exon* and *unknown regions*. The main interest in medicine is classifying the function of DNA sequences into enhancer, promoter and background since non-coding gene regulatory enhancers are essential to transcription in mammalian cells [60].

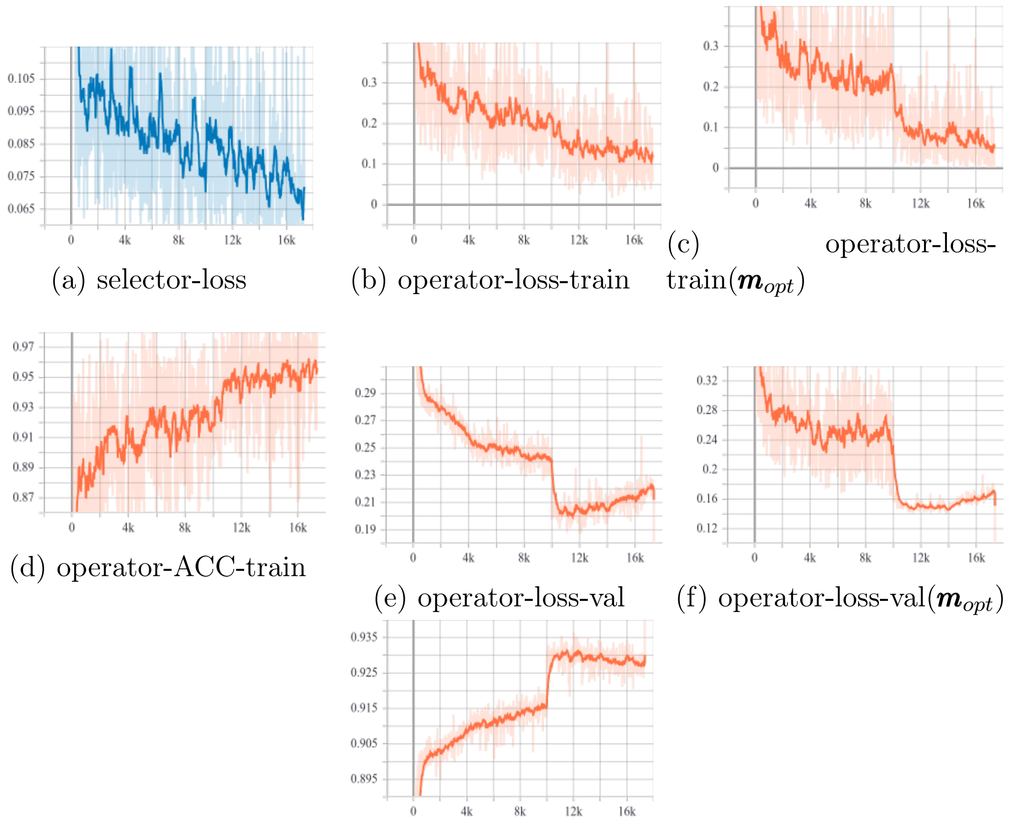
Following the suggestion in [60, 55], we merge inactive enhancers, inactive promoters, active exons, and unknown regions into a background class. Thus, we have a 3-class imbalanced dataset as the background class has roughly 5 times more instances than the other two classes: active promoter and active enhancer. We follow a preprocessing method consisting of two steps: 1) making the dataset balanced by down-sampling so that each of 3 classes has 2878 samples; 2) overcoming the natural skewness of biological outcome by taking the logarithm on the input. To avoid the zero-value issue in the logarithm, we append a small positive number to each feature, $x \leftarrow x + 0.01$, in our experiments. Note that step 2) is not described in the DFS paper but we believe that this is important for such a data distribution.

It is also worth clarifying that we see some discrepancies between the data presented in the authors' repository[6] and their article [55]. Two main differences include 1) 102 features in the repository but 92 features stated in the article; 2) at least 2,878 instances per class in the repository but only 2,156 features mentioned in the article. While we use the dataset in their repository, we have done our best by keeping all the settings suggested in their article for a fair comparison in our experiments.

Due to the limited space in the main text, we only report the result of our approach for a subset mask size, $s = 35$. Here, we report more results of our approach and other methods used in a comparative study on this dataset.

We first illustrate the learning behaviour of our dual-net model on this real-

[6][online]: `https://github.com/yifeng-li/DECRES/tree/master/data`



(a) selector-loss (b) operator-loss-train (c) operator-loss-train($\boldsymbol{m}_{opt}$)

(d) operator-ACC-train (e) operator-loss-val (f) operator-loss-val($\boldsymbol{m}_{opt}$)

(g) operator-ACC-val

Figure 5.4: **Enhancer-Promoter Dataset**. Evolution of the operator and the selector losses in Phase II ($d = 102, s = 35$). The `x-axis` corresponds to the number of batches and `y-axis` refers to the loss statistics of 5 folds. (a) The selector loss. (b) The operator loss on the training set. (c) The operator loss on the training set with $\boldsymbol{m}_{opt}$ only. (d) The classification accuracy evaluated on the training set. (e) The operator loss on the validation set. (f) The operator loss on the validation set with $\boldsymbol{m}_{opt}$ only. (g) The classification accuracy evaluated on the validation set. Note that Phase II starts when the operator net has been trained for 10,000 batches in Phase I. The final datapoint at $x \approx 18000$ in (e)-(g) corresponds to performance being evaluated on the test set with a 'virtual' batch number around 18k.

world dataset in Figure 5.4. As shown in Figure 5.4(a), the averaged selector loss has the typical behaviour as described in Section 5.2. The loss fluctuation and the loss reduction trend in the loss evolution vividly exhibit how the stochastic local search strategy works in finding out optimal subset masks. As shown in Figures 5.4(b)-5.4(g), the learning progresses steadily as evident in the evolution of the averaged operator losses and the averaged classification accuracies on the training and the validation sets. Also, it is clearly seen in Figures 5.4(b)-5.4(g), that overfitting occurs once the optimal subset mask is identified at around 10.5k batches. Once again, this observation provides solid evidence to support early stopping with the operator training/validation losses measured on the optimal subset masks ($\boldsymbol{m}_{opt}$). Furthermore, it is also seen in Figures 5.4(e)-5.4(g) (at the maximum batches) that the averaged test losses and the averaged classification accuracy yielded by the trained operator net on 5 folds are superior to their counterparts on the validation set. Once again, this suggests that our alternate learning leads to favourable generalization performance although an earlier stopping may yield better accuracy.

Apart from the comparative study specified in the main text, we have conducted further experiments on this dataset for comparison to two recent state-of-the-art methods [61, 62] that obtain the population-wise FIR by aggregating the instance-wise FIR. In [61], the global aggregation method workable on this dataset is the homogeneity-weighted importance, which is the same as the global LIME importance proposed in [48]. In our experiment, we use an MLP of the architecture: 102-300-200-50-3 and `n_samples=500` to achieve the LIME importance on the validation set [48]. In the SAN [62], the population-wise FIR is obtained via either instance-level aggregation (SAN_AGGR) or global attention layer (SAN_GLOBAL). In our experiments, we use the same settings suggested by the authors [62][7] with the following hyperparameters: $k = 1$, `p_dropout=20%`, `epochs=32`, `batch_size=32`, `n_1_1=128` (number of hidden neurons in SAN). In terms of feature selection, both methods fall into the filtering category. Therefore, we employ an MLP of the architecture: s-300-200-50-3 to be a classifier trained on those selected feature subsets of $s = 5, 25, 35, 45, 55$, respectively, for this 3-class classification task.

We report the accuracies yielded by 6 different methods for different subset sizes. As shown in Figure 5.6, it is evident that our approach yields slightly better accuracies than the DFS [55], a method especially proposed for this biological dataset, when the subset size of selected features is larger than 15. Also, our approach outperforms RFE [2], a state-of-the-art feature selection method especially effective in gene selection for cancer classification, and RF [8], a famous off-the-shelf ensemble learning model. In contrast, it is evident from Figure 5.6 that ours along with DFS also outperforms those methods of using the aggre-

[7]We do this experiment with the authors' code: https://gitlab.com/skblaz/attentionrank.

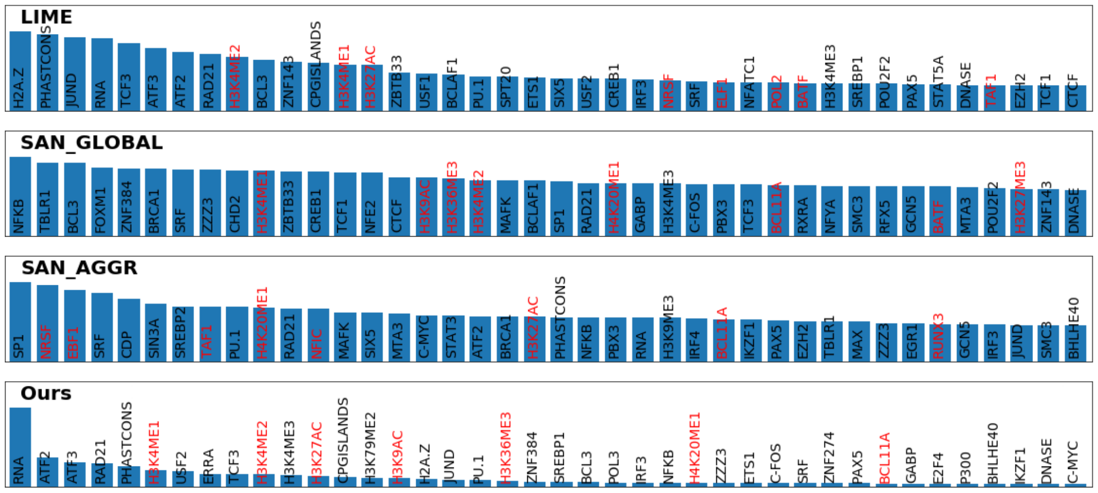


Figure 5.5: Feature importance ranking (FIR) scores yielded by LIME, SAN and ours for top 55 features ($s = 55$) on the Enhancer-Promoter dataset: GM12878 Cell line (200 bp).

gation of instance-wise FIR to obtain population-wise one at all different subset sizes ranging from 15 to 55.

For feature importance ranking (FIR), we show the FIR scores yielded by two instance-wise aggregation-based methods and ours for $s = 55$ in Figure 5.5. It is observed that the two methods and ours yield different FIR results and different settings in the SAN also result in different FIR for the top 55 features. Due to a lack of ground truth, it is difficult to draw an affirmative conclusion but the experimental results suggest that the population-wise FIE is an extremely challenging problem for real-world data. Nevertheless, we have highlighted the variables suspected to be important [55] in red.

We further show the FIR scores for the top 40 features produced by DFS, RF, RFE and ours in Figure 5.7. The FIR scores of the RFE and the RF are generated based on the RFE feature importance estimator [2] and the out-of-bag errors [8]. The FIR score of the DFS is achieved based on the magnitude of shrunk weights between input and the first hidden layer introduced in the DFS method [55]. From Figure 5.7, it is observed that our approach yields relatively consistent FIR results when different subset sizes are used given the fact that the importance ranking order of top features only varies for one or two. Also, our approach is the only one to rank “RNA” and two important genes “ATF2"” and “ATF3” consistently among the most important features regardless of feature subset sizes. In comparison, the DFS also selects those two genes but does not rank them at the top. On the other

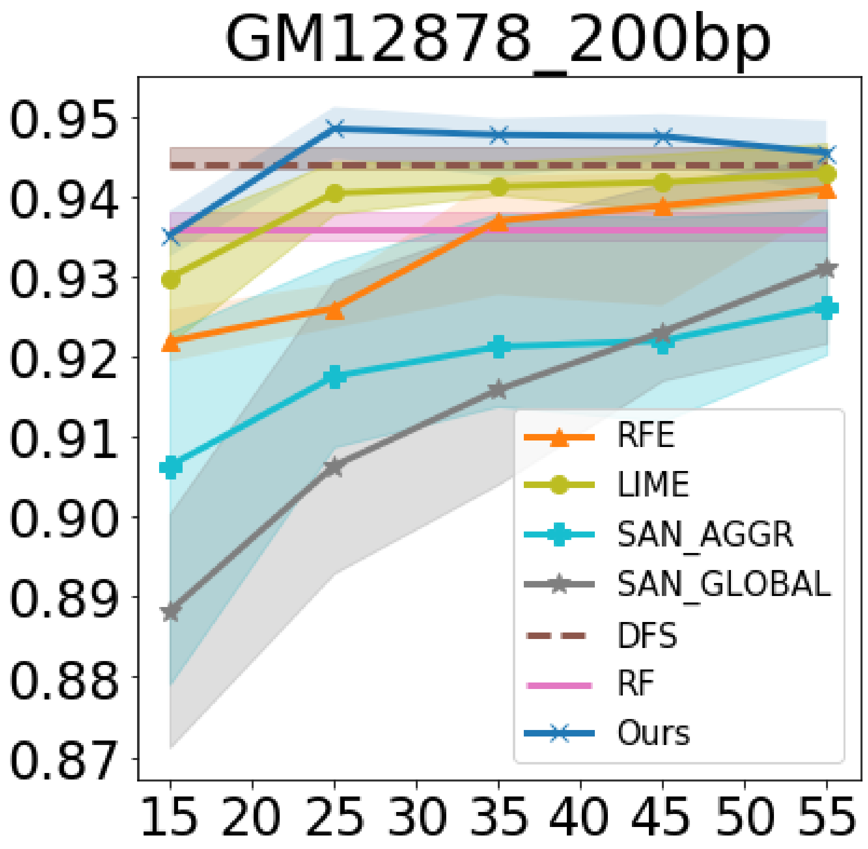


Figure 5.6: Classification accuracies yielded by different methods on the Enhancer-Promoter dataset: cell line GM12878 (200dp). The shadowed regions refer to the performance range between the minimum and the maximum accuracies on 5 folds. While DFS and RF yield only one result with all the 102 features, other methods produce the results at different subset sizes for $s = 15, 25, 35, 45, 55$.

hand, the RFE chooses other genes, "RAD21", "“PGISLANDS" and "H3K4ME3", as the most important features irrespective of feature subset sizes. It is also be seen in Figure 5.7 that the DFS and ours, two deep learning models rank the importance of features similarly but differently from the RFE and the RF that yield similar FIR scores. Our experimental results on this real-world dataset suggest that deep learning models may lead to different results from the existing state-of-the-art and off-the-shelf machine learning models for FIR. Thus, learning models of different types should be considered simultaneously and their results can be fused at the discretion of domain experts in such real-world applications.

To evaluate the efficiency, we record the averaging training time on this dataset in terms of 5-fold cross-validation. our experiments are conducted on a server of the specification and the environment: Intel Core i7-5930K CPU (3.50GHz), NVIDIA GeForce GTX TITAN X GPU, 64 GB RAM and CentOS 7. In summary, our algorithm takes around 1,100 sec while RF, SAN, DFS, LIME and RFE take around 2.5, 35, 90, 540 and 1,700 sec, respectively. The dual-net architecture along with the alternate learning is responsible for a high computational load in our approach.

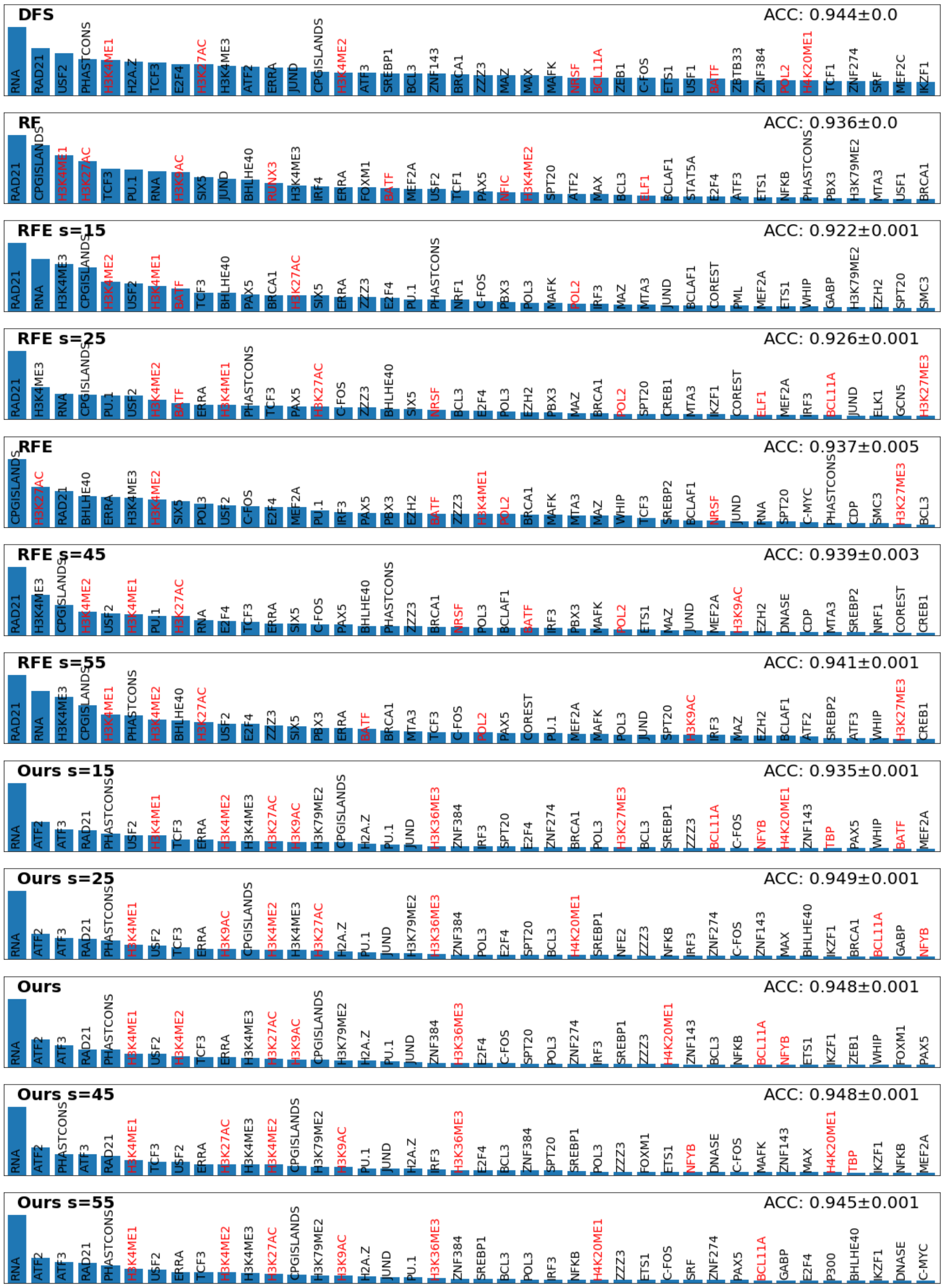


Figure 5.7: Accuracy and feature importance ranking (FIR) scores yielded by different methods on the Enhancer-Promoter dataset: GM12878 Cell line (200 bp). While DFS and RF yield only one result with all the 102 features, RFE and ours produce the results at different subset sizes for $s = 15, 25, 35, 45, 55$. Note that the results yielded RFE and ours for $s = 35$ above are not specified deliberately with the subset size to indicate that those have been reported in the main text.

RNA-seq Data

Apart from the comparative study on the Enhancer-Promoter dataset, we have further applied our approach to the UCI gene expression cancer RNA-Seq data set[8], to evaluate our approach on a dataset of many features.

The gene expression cancer RNA-Seq dataset is part of The Cancer Genome Atlas Pan-cancer Analysis Project. The original data set is maintained by the cancer genome atlas pan-cancer analysis project. Gene expression data are composed of DNA microarray and RNA-seq data. Therefore, microarray data analysis facilitates the clarification of biological mechanisms and the development of drugs toward a more predictable future. In comparison to hybridization-based microarray technology, RNA-seq has a larger range of expression levels and contains more information. RNA-Seq is a random extraction of gene expression of patients with five different types of tumours including BRCA (breast), KIRC (kidney), COAD (colon), LUAD (lung) and PRAD (prostate). The dataset contains 801 samples, each of which has 20,531 biological features or genes. The data set is imbalanced and there are 300, 146, 78, 141 and 136 samples for BRCA, KIRC, COAD, LUAD and PRAD, respectively.

Unlike other methods, e.g., [63], we do not pre-process the imbalanced data in our experiment apart from the removal of 267 constant features. In other words, we use only 20,264 features in each sample to train our dual-net model. All the data are standardised with zero mean and unit standard deviation. The dataset is randomly split into two subsets, training and test, where there are 600 and 201 samples in the training and the test subset. The four-fold cross-validation (for fair comparison) working on the training subset is used for parameter estimate and hyperparameter tuning for our dual-net model. To make a fair comparison to the best performer on this dataset as reported in [63], we use the identical setting by using $s = 49$ in our experiment. The information on the dual-net architecture and optimal hyperparameter values used in this experiment is provided in Tables III and IV, respectively.

Table II shows the existing results of several feature selection methods [63, 64, 65, 66, 67, 68, 69] on this dataset with various settings although most of the

[8][online]: https://archive.ics.uci.edu/ml/datasets/gene+expression+cancer+RNA-Seq

Table II: Accuracy yielded by different methods on RNA-seq dataset (adapted from Table 7 in [63]).

Method	Samples	Features	Classes	Selected Features	Classifiers	Accuracy
[64]	96	4026	9	<60	1	0.9730
[65]	62	6000	2	15	1	0.9677
[66]	97	24481	2	7	7	0.9381
[66]	102	12600	2	4	7	0.9706
[67]	175	1072	2	-	110	0.9500
[68]	215	1047	4	-	20	0.9860
[69]	569	32	2	24	1	0.9877
[63]	801	20531	5	49	5	0.9881
Ours	**801**	**20531**	**5**	**49**	**1**	**0.9938**

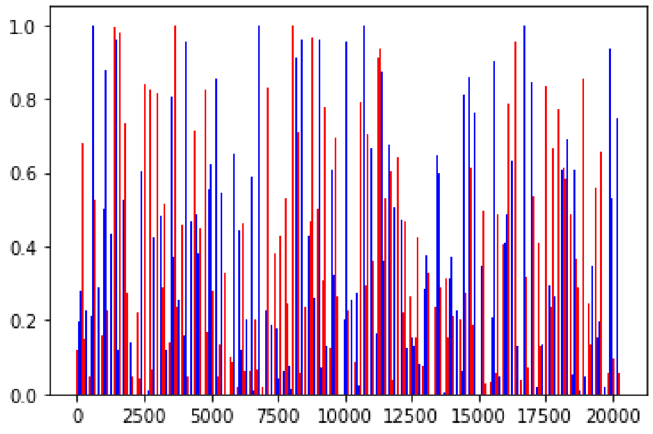


Figure 5.8: Feature importances for the RNA-seq dataset. Red lines indicate truly important features while blue lines correspond to negative impotance.

existing methods do not work on the entire dataset. From Table II, it is clearly seen that, under the same settings, our approach outperforms the best performer on this dataset in literature. Additionally, in Figure 5.8 we provide the feature importance estimates, where the red/blue corresponds to important/noisy features (positive/negative importance). The experimental result on this nontrivial real-world dataset demonstrates that our approach works well for a data set of many features as long as there are enough training examples required by deep learning. Thus, we firmly believe that our approach will be applicable to a large data set of many features, e.g., images where there are millions of pixels. We shall look into the scalability of our approach in our ongoing work.

In summary, our experimental results manifest that leveraged with deep learning, our approach outperforms a number of state-the-art FIR and feature selection methods on two biological datasets. This suggests that our approach would be a strong candidate for feature selection and feature importance ranking in real-world biological applications.

5.1.3 Visual Validation

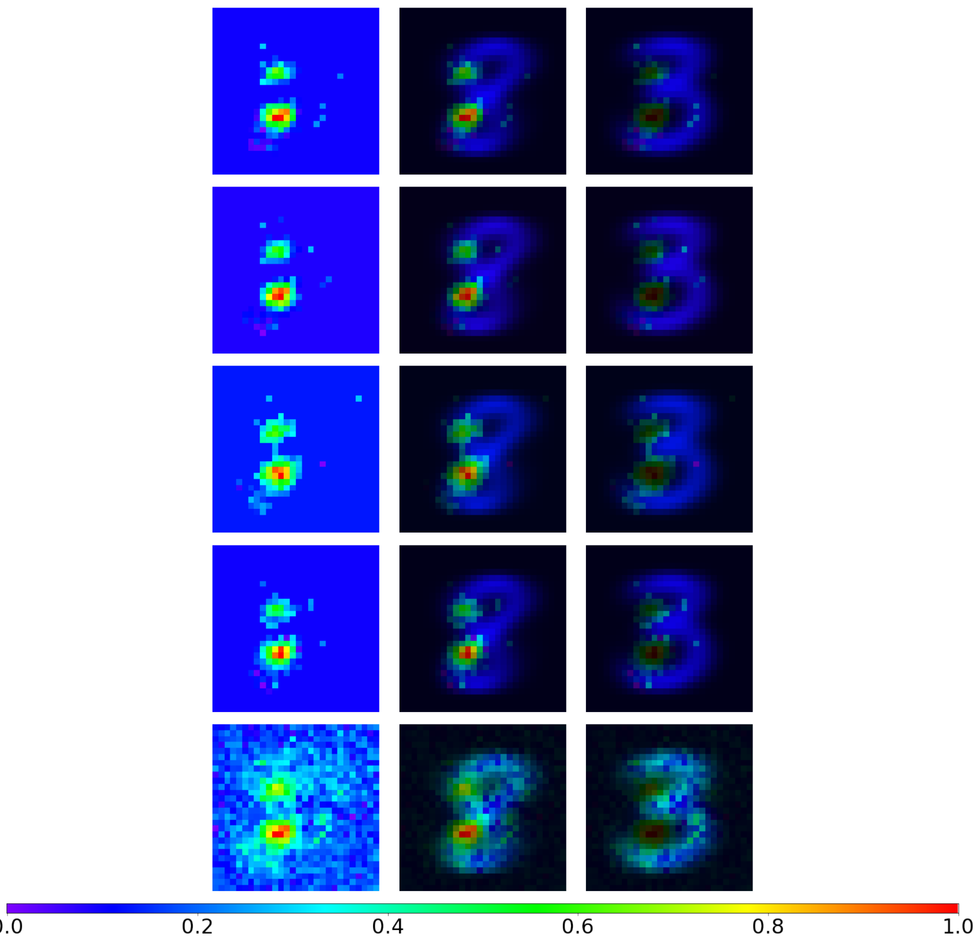


Figure 5.9: **MNIST Subset**. Feature importance maps ($d = 784, s = 85$) generated with the method described in Section 5.1.3. From top to bottom, the first 4 rows correspond to feature importance maps achieved from different folds. The bottom row is the full feature importance map corresponding to the feature importance map shown in Figure 4 of the main text.

Below, we show feature importance maps achieved from other folds on the MNIST subset to prove the stability of the algorithm and those yielded by our approach on the Yale dataset.

To obtain the superimposed feature importance maps on the background image (the mean of raw images), we apply a method as follows. A blank image is first created in the HSV (hue-saturation-value) colour format. The hue used in the [0, 270] range corresponds to the importance, and the saturation is set to 1.0

to encode the mean background image from the dataset. Due to the feature importance ranking (FIR) scores being normalised, no negative FIR scores are shown to ensure unselected features have the background colour.

Figure 5.9 shows different feature importance maps achieved from the other 4 folds. As our FIR approach described in Section 4.3 of the main text measures the FIR scores based on the input gradient, it can achieve the input gradient for all the features regardless of whether a feature is selected or not. To this end, we can generate a full feature importance map as well. It is observed from 5.9 that the feature importance maps achieved from different folds are very much consistent and the full feature importance map provides a clearer picture in terms of explainability/interpretability.

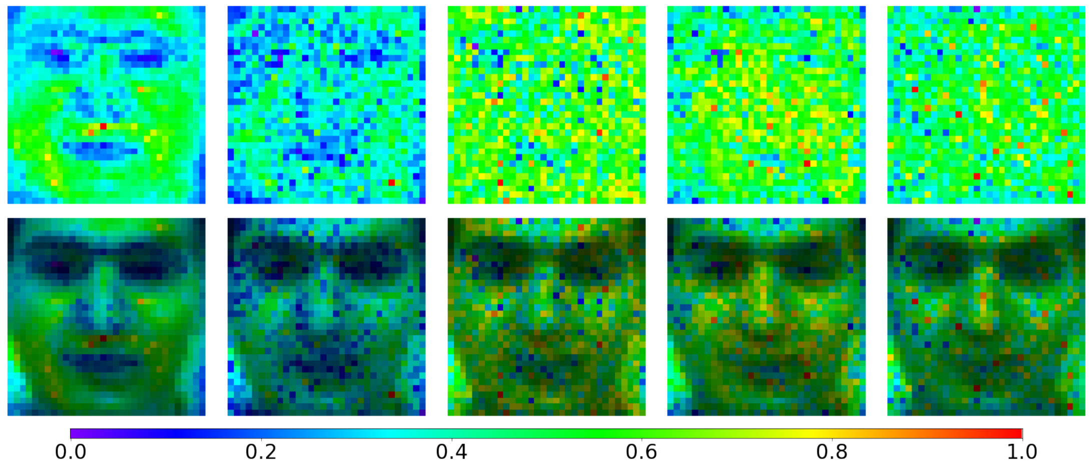


Figure 5.10: **Yale Dataset**. Feature importance maps achieved with different subset mask sizes: $s = 10, 30, 50, 70, 90$ (from left to right) out of $d = 32 \times 32$. The second row corresponds to the images generated by superimposing the feature importance maps to the background image, i.e., the mean face image averaged on 11 images collected from an individual.

As Yale is a facial image dataset, we can also illustrate the feature importance maps in Figure 5.10 for visual inspection. The visual inspection reveals that increasing the mask size s results in a less clear visual representation of feature importance. In comparison, the best-performing mask size of 30 clearly selects several meaningful yet discriminative features, e.g., pixels near the lip, nose and eyes, and ranks their importance properly, as shown in the 2nd column in Figure 5.10. As this dataset has limited instances (7 training examples/class on average), we reckon that the use of a large subset mask size is likely to cause overfitting, as revealed by their feature importance maps shown in Figure 5.10.

5.1.4 Experimental Setup

In this section, we describe the details of the experimental settings used in our experiments. In our experiments, we always use the *grid search* along with *5-fold cross-validation* on a training set to find out optimal hyperparameters involved in different learning methods. Below, we first present the detailed setup in our approach, then describe all the technical details of other learning methods used in our comparative studies on different datasets.

In our implementation of the operator net, we have to consider an issue concerning the differentiation between *a selected feature of which value is zero* and *any removed features masked with zero* due to the use of the binary masks in our work. Thus, we design an operator net architecture shown in Figure 5.11. Instead of feeding only the selected features, $\boldsymbol{x} \otimes \boldsymbol{m}$, to the first hidden layer, we concatenate the mask, $\boldsymbol{m}$, used to indicate the selected features, and the selected features themselves, $\boldsymbol{x} \otimes \boldsymbol{m}$, to form the input fed to the first hidden layer as illustrated in Figure 5.11. Thus, the dimension of the input to the first hidden layer is $2d$ rather than the d features previously described. It is worth mentioning that we had investigated other manners to tackle the aforementioned “zero-value" issue, e.g., stipulating a value beyond the range of any features for a removed feature in $\boldsymbol{x} \otimes \boldsymbol{m}$. However, neither of those yields a better performance than the architecture presented above.

Moreover, there are specific settings in our approach due to technical reasons; e.g., the loss function used to train the selector net and the subtle technical details related to Phase II in our alternate learning algorithm, as described in Section 4.2.

In our experiments, the loss function used to train the selector net presented in Eq.4.1b is actually replaced by a *weighted* loss as follows:

$$\mathcal{L}_S(\mathcal{M}'; \varphi) = \frac{1}{2|\mathcal{M}'|} \sum_{\boldsymbol{m} \in \mathcal{M}'} w_{\boldsymbol{m}} \Big(f_S(\varphi; \boldsymbol{m}) - \frac{1}{|\mathcal{D}|} \sum_{(\boldsymbol{x},\boldsymbol{y}) \in \mathcal{D}} l(\boldsymbol{x} \otimes \boldsymbol{m}, \boldsymbol{y}; \theta) \Big)^2, \quad (5.1)$$

where $w_{\boldsymbol{m}} = 10, 5, 1$ is set to:

- **high value** when $\boldsymbol{m} = \boldsymbol{m}_{t,best}$ (the best=performing subset found in the last step),

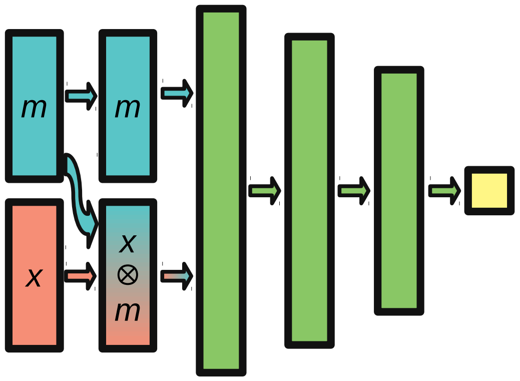


Figure 5.11: The actual implementation of the operator net in our experiments to overcome the "zero-value" representation issue. As a result, both the selected features, $\boldsymbol{x} \otimes \boldsymbol{m}$, and the mask, $\boldsymbol{m}$, used to indicate those selected features are concatenated as the input to the first hidden layer.

Table III: The optimal architectural hyperparameters of our dual-net learning model in our experiments. **Yale***: uses convolutional layers.

Data Set	**Operator Net**	**Selector Net**
4-way Classification	$20 \rightarrow 60 \rightarrow 30 \rightarrow 20 \rightarrow 4$	$10 \rightarrow 100 \rightarrow 50 \rightarrow 10 \rightarrow 1$
Nonlinear Regression	$20 \rightarrow 100 \rightarrow 50 \rightarrow 25 \rightarrow 1$	$10 \rightarrow 100 \rightarrow 50 \rightarrow 10 \rightarrow 1$
Binary Classification	$20 \rightarrow 60 \rightarrow 30 \rightarrow 20 \rightarrow 1$	$10 \rightarrow 100 \rightarrow 50 \rightarrow 10 \rightarrow 1$
MNIST Subset	$1568 \rightarrow 500 \rightarrow 250 \rightarrow 100 \rightarrow 1$	$784 \rightarrow 300 \rightarrow 200 \rightarrow 100 \rightarrow 1$
Glass	$20 \rightarrow 50 \rightarrow 25 \rightarrow 10 \rightarrow 6$	$10 \rightarrow 500 \rightarrow 250 \rightarrow 100 \rightarrow 1$
Vowel	$20 \rightarrow 50 \rightarrow 25 \rightarrow 10 \rightarrow 11$	$10 \rightarrow 500 \rightarrow 250 \rightarrow 100 \rightarrow 1$
TOX-171	$11568 \rightarrow 100 \rightarrow 50 \rightarrow 20 \rightarrow 4$	$5784 \rightarrow 500 \rightarrow 250 \rightarrow 100 \rightarrow 1$
Yale*	$32{\times}32 \rightarrow 16{\times}16{\times}32 \rightarrow 8{\times}8 \times 64 \rightarrow 4{\times}4{\times}128 \rightarrow 30 \rightarrow 30 \rightarrow 15$	$1024 \rightarrow 500 \rightarrow 250 \rightarrow 100 \rightarrow 1$
Enhancer–Promoter	$204 \rightarrow 300 \rightarrow 200 \rightarrow 50 \rightarrow 3$	$102 \rightarrow 500 \rightarrow 250 \rightarrow 100 \rightarrow 1$
RNA-seq	$40528 \rightarrow 1000 \rightarrow 500 \rightarrow 200 \rightarrow 5$	$20264 \rightarrow 500 \rightarrow 250 \rightarrow 100 \rightarrow 1$

- **middle value** when $\boldsymbol{m} = \boldsymbol{m}_{t+1,opt}$ (the optimal subset generated in the current step),
- **low value** for any other subsets in $\mathcal{M}'_{t+1}$, respectively (c.f. Phase II-A in Section 4.2).

In our experiments, we set these **high**/**middle**/**low** values to 10/5/1 respectively. The weighted selector loss exploits what has been learned so far in order to facilitate the stochastic local search in tackling the combinatorial optimization problem more effectively.

In our experiments, the 3-step validation procedure used for *the generation of the optimal subset* would ensure its optimality within those feature subset

Table IV: Other optimal hyperparameters of our dual-net learning model in our experiments. E_1 is the number of SGD training batches (instead of epoches) in Phase I of the alternative learning. In Phase II.A, $|\mathcal{M}'_t|$ refers to the number of different optimal subset candidates used in a single batch during the SGD learning. $f = \frac{|\mathcal{M}'_{t,1}|}{|\mathcal{M}'_{t,2}|}$ is the fraction that governs the exploitation-exploration trade-off in the selector learning, and s_p indicates the number of elements perturbed. For details of the alternative learning algorithm, see Section 4.2.

Data Set	E_1	$\lvert\mathcal{M}'_t\rvert$	f	s_p
4-way Classification	$6,000$	32	0.5	2
Nonlinear Regression	$6,000$	32	0.5	2
Binary Classification	$6,000$	32	0.5	2
MNIST Subset	$10,000$	32	0.5	5
Glass	$10,000$	128	0.5	2
Vowel	$6,000$	128	0.5	2
TOX-171	$1,500$	128	0.5	5
Yale	$4,500$	128	0.5	5
Enhancer–Promoter	$10,000$	64	0.5	5
RNA-seq	8000	32	0.5	5

candidates (c.f. Phrase II.A in Section 4.2). However, the condition to exit from the loop of repeating steps i)-iii) may not always be satisfied. In our experiments, we hence set the maximum number of repetitions in this test to *5* iterations so that the subset optimality validation procedure always ends up with *five* iterations. In addition, the parameter update of the operator and the selector nets in Phase II is done in different frequencies in the alternate learning; i.e., the parameters in the operator net are updated once on each batch in Phase II.B, while the parameters in the selector net are updated once on every 8 batches in Phase II.A.

We employ MLPs (CNNs) of the sigmoid (ReLu) neurons to carry out the operator net and MLPs with the sigmoid neurons for the selector net in our dual-net architecture. For training MLPs (CNNs), we adopt the *Adam optimizer* (Adam with Nestrov momentum for the operator net) [70] via the stochastic gradient descent (SGD) procedure. Early stopping is used based on the losses evaluated on the validation data[9]. All the optimal hyperparameters used in our experiments are summarized in Table IV. The used architectures are shown in Table III, where we display the numbers of neurons used in each dense layer. A special

[9]In our alternate learning procedure, we use the operator loss incurred by the optimal subset, $\boldsymbol{m}_{t,opt}$, on the validation set for early stopping. For clarity and details, see Section 4.6.

Table V: Optimal regularization hyperparameters, λ, used in LASSO on different datasets in our experiments.

Data Set	Fold-1	Fold-2	Fold-3	Fold-4	Fold-5
4-way classification	0.055	0.056	0.046	0.008	0.042
Nonlinear Regression	0.001	0.0	0.053	0.001	0.0
Binary Classification	0.025	0.07	0.081	0.02	0.039

case is the Yale dataset, where we used convolutional layers for the Operator so that the displayed architecture corresponds to the shapes of internal layers. Each convolutional layer has zero padding, max pooling and kernels of sides 5, 3, and 3 respectively. Two dense layers follow the convolutional layers.

5.1.5 Optimal Hyperparameters In Other Methods

Table VI: Optimal hyperparameters (#tree, depth) for RF on different datasets in our experiments.

Data Set	Fold-1	Fold-2	Fold-3	Fold-4	Fold-5
4-way classification	(80, 14)	(70, 11)	(90, 15)	(150, 14)	(150, 12)
Nonlinear Regression	(50, 15)	(60, 15)	(50, 14)	(60, 10)	(50, 8)
Binary Classification	(60, 14)	(50, 10)	(90, 8)	(90, 15)	(160, 13)
MNIST	(200, 21)	(200, 20)	(210, 21)	(200, 21)	(200, 20)
Enhancer-Promoter	(120, 11)	(120, 12)	(100, 15)	(150, 13)	(60, 14)

Other 9 methods have also been employed for a comparative study on different datasets. We strictly follow the original settings described in those papers. We implement deep learning algorithms by ourselves with Tensorflow 2 [71] and Keras [72]. For other methods, we use the existing code in the Python scikit-Learn library [73] for FS, LASSO, RF, RFE or the authors' project website for BAHSIC[10], mRMR[11] and CCM [12]. Below, we summarize the actual optimal hyperparameters pertaining to those methods used in our comparative study as well as brief descriptions of each used method.

[10]BAHSIC webpage: https://www.cc.gatech.edu/~lsong/code.html
[11]PyMRMR library: https://pypi.org/project/pymrmr/
[12]CCM repository: https://github.com/Jianbo-Lab/CCM

Deep Feature Selection (DFS) [55]. For DFS, we use the MLPs of the architectures as same as that of the operator net in our dual-net architecture apart from the input layers for a given task, as shown in Table III. Instead of having the concatenation of the selected features and the mask indicating the selected features in our operator net, the DFS appends an additional one-to-one layer between the input and the first hidden layer. Similarly, the sigmoid neurons are used in their modified MLPs and the Adam optimizer [70] is adopted for training MLPs via the SGD. The optimal regularization hyperparameter is $\lambda = 0.01$ for 3 synthetic datasets and the MNIST subset after a grid search from a large range of λ. For the Enhance-Promoter dataset, the optimal hyperparameter is $\lambda = 0.008$. The rest of the parameters are kept the same as suggested in [55], which is $\lambda_2 = 1, \alpha_1 = 0.0001, \alpha_2 = 0$. Note that we implement the DFS code by ourselves with Tensorflow 2.0 and Keras since the authors' code is merely applicable to a specific dataset.

Average Input Gradient (AvGrad) [47]. It is simply a post-processing method for feature importance ranking (FIR) based on a trained MLP, we employ the same MLP architecture as that of our operator net apart from the input and the Adam optimizer [70] via the SGD on 3 synthetic datasets and the MNIST subset, Table III.

Forward Selection (FS) [4]. For FS, we employ the MLPs as the base learner in this wrapper method and the training procedure is identical to those used in the AvGrad on 3 synthetic datasets. For FIR, FS always ranks the importance of an early-selected feature higher than that of others selected later in the forward subset selection procedure.

LASSO [42]. We use the grid search to find out the optimal regularization hyperparameter, λ, in LASSO. The optimal hyperparameters found in 5 folds are listed in Table V.

Random Forest (RF) [8]. We use the grid search to find out the optimal hyperparameters: number of decision trees and depth of the trees. We search from a range from 50 to 220 trees and between 7 and 24 in depth. The optimal hyperparameters found in 5 folds are listed in Table VI.

Recursive Feature Estimation (RFE) [2]. We use 1 step for all the datasets apart from TOX-171 and Yale datasets where 5 steps are used. In our experiments, we adopt the default values for underlying estimators (linear SVM) with $C = 1$ and $\gamma = \frac{1}{n_{features} * \text{var}(X)}$.

Backward Elimination using HSIC (BAHSIC) [56, 41]. A default hyperparameter regarding the fraction of removed features in each iteration is set to 0.1 as suggested in their papers. In our experiments, we adopt the inverse kernels suggested in their papers and the BAHSIC webpage.

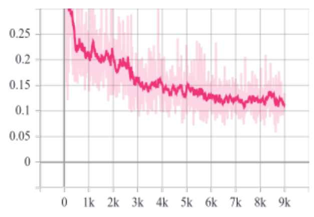
Minimal Redundancy Maximal Relevance Criterion (mRMR) [6]. No hyperparameter needs to be tuned in this method. In our experiments, we adopt the "MIQ" option suggested in the `PyMRMR` library.

Conditional Covariance Minimization (CCM) [5]. We use $\epsilon = 0.001$ for two synthetic classification datasets, 4-way and binary classification, and 4 benchmark datasets, Glass, Vowel, TOX-171 and Yale. For the nonlinear regression dataset, we use $\epsilon = 0.1$. As all 7 datasets were used in the paper, we adopt the optimal hyperparameters reported in the paper and suggested in the CCM repository.

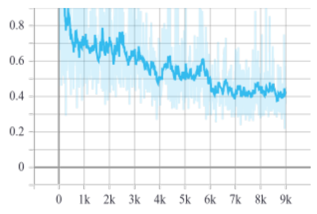
As CCM, RFE, BAHSIC, mRMR and LASSO are filtering methods for feature selection, we need to measure their performance based on another learning model. For CCM, RFE, BAHSIC and mRMR, we adopt the same setting used in [5], i.e., SVM/SVR with a Gaussian kernel of optimal hyperparameters: $C = 1$ and $\gamma = \frac{1}{n_{features} * \text{var}(X)}$. For LASSO, we use the same MLPs used in deep learning models, i.e., DFS, AvGrad and ours (operator net).

5.2 Learning behaviour

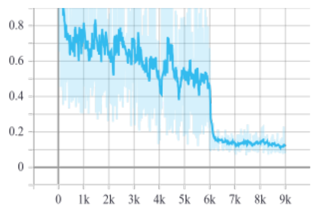
As described in Section 4.2, our alternate learning algorithm trains two learning models, operator and selector, simultaneously in an alternate manner; i.e.,



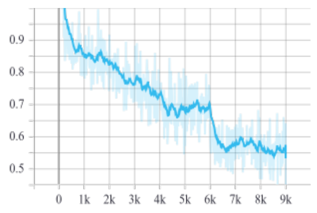
(a) selector-loss



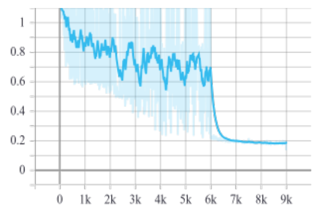
(b) oper.-loss-train



(c) oper.-loss-train($\boldsymbol{m}_{opt}$)



(d) oper.-loss-val



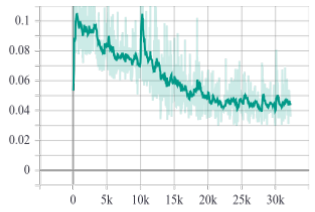
(e) oper.-loss-val($\boldsymbol{m}_{opt}$)

Figure 5.12: **Synthetic nonlinear regression dataset**. Evolution of the operator and the selector losses in Phase II ($d = 10, s = 5$). The `x-axis` corresponds to the number of batches, and `y-axis` refers to the loss statistics of 5 folds. (a) The selector loss. (b) The operator loss on the training set. (c) The operator loss on the training set with $\boldsymbol{m}_{opt}$ only. (d) The operator loss on the validation set. (e) The operator loss on the validation set with $\boldsymbol{m}_{opt}$ only. Note that Phase II starts when the operator net has been trained for 6,000 batches in Phase I.

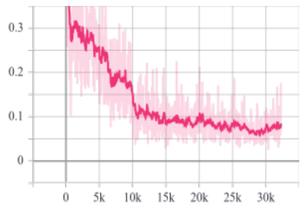
in Phase II, the learning behaviours of the operator and the selector nets are mutually affected by each other in each batch during the SGD learning. This is different from most of the existing deep learning algorithm that involves only a deep neural network to be trained. Therefore, we need to investigate how our proposed learning model behaves. Below, we exhibit the typical learning behaviour in our alternative learning on 3 datasets, *synthetic nonlinear regression*, *MNIST subset* and *Yale*.

Figure 5.12 illustrates the learning behaviour of the operator and the selector in terms of losses in Phase II on the synthetic *nonlinear regression* dataset. It is observed from Figure 5.12(a) that the trained operator in Phase I provides informative training examples so that the averaged selector loss of 5 folds decreases monotonically as required. As evident in Figures 5.12(b) and 5.12(d), the averaged operator loss on training and validation sets further decreases steadily as the selector keeps offering more "promising" optimal mask candidates achieved by the stochastic local search for combinatorial optimization. It is clearly seen in Figures 5.12(b) and 5.12(d) that at the beginning of Phase II (up to 1k batches), operator loss on both training and validation sets sharply decreases once the selector has been involved. Also, the loss may be reduced substantially when an optimal mask is identified, as shown in Figure 5.12(d) (between 6k and 7k batches). Given the fact that at the end of Phase II. For each iteration, we always achieve an optimal mask, $\boldsymbol{m}_{t,opt}$. Thus, we can apply such optimal masks only to measure the operator loss. As a result, Figures 5.12(c) and 5.12(e) illustrate the evolution of the operator loss evaluated with $\boldsymbol{m}_{t,opt}$ only on training and validation sets. In contrast to the operator loss with all optimal mask (subset) candidates shown in Figure 5.12(d), the abrupt loss drop resulting from the identified optimal mask is much more visible in Figure 5.12(e). Therefore, early stopping in our alternate learning algorithm is based on the operator loss evaluated with $\boldsymbol{m}_{t,opt}$ only. Overall, Figure 5.12 demonstrates that our alternate learning algorithm works well and eventually converges for this regression task.

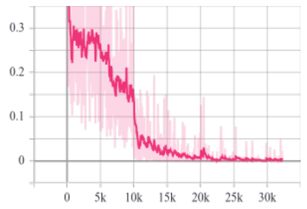
Next, Figure 5.13 illustrates the learning behaviour of the operator and the selector in terms of losses in Phase II on the MNIST benchmark subset, a *binary classification* task. It is seen from Figure 5.13(a) that the evaluation of the averaged selector loss of 5 folds has a reduction trend as the number of batches is increased although the averaged loss no longer drops monotonically. The sharp selector loss increase at around 10k batches is typical and reflects the nature of our stochastic local search procedure in tackling the combinatorial optimization issue. The sharp increase is likely caused by the fact that the optimal mask identified leads to the sharp operator loss reduction and the selector net did not have such training examples before this moment. This analysis is manifested by all the results at round 10k batches shown in other plots in Figure 5.13. As evident in Figures 5.13(b) and 5.13(c), the averaged operator loss on training further decreases in general. Using an alternative performance index, we also show the averaged classification accuracy measured on the training set in Figure 5.13(d), allowing one to see the learning performance vividly. Likewise, we illustrate the averaged operator loss and accuracy on the validation set in Figures 5.13(e)-(g). Once again, we can see our alternate learning algorithm works very well. Once again, the operator validation loss evaluated with $\boldsymbol{m}_{opt}$ only provides solid evidence for early stopping. In general, the learning behaviour on this binary classification dataset very much resembles that of the nonlinear regression dataset (c.f. Figure 5.12). After the alternate learning is completed, we can evaluate the performance



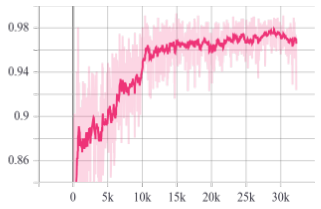
(a) selector-loss



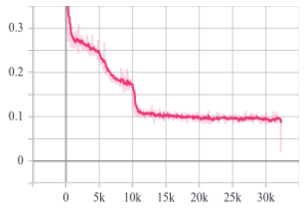
(b) oper.-loss-train



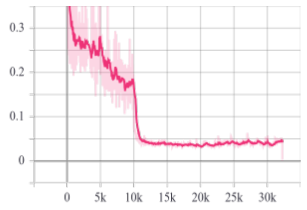
(c) oper.-loss-train($\boldsymbol{m}_{opt}$)



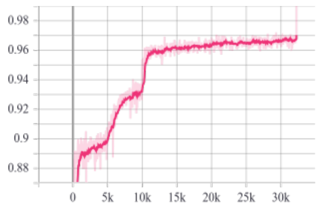
(d) oper.-ACC-train



(e) oper.-loss-val



(f) oper.-loss-val($\boldsymbol{m}_{opt}$)

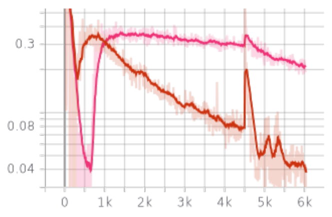


(g) oper.-ACC-val

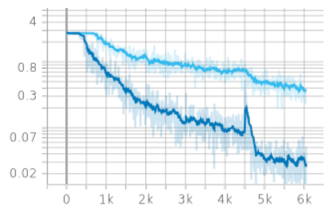
Figure 5.13: **MNIST Benchmark Subset**. Evolution of the operator and the selector losses in Phase II ($d = 784, s = 85$). The `x-axis` corresponds to the number of batches and `y-axis` refers to the loss statistics of 5 folds. (a) The selector loss. (b) The operator loss on the training set. (c) The operator loss on the training set with $\boldsymbol{m}_{opt}$ only. (d) The classification accuracy evaluated on the training set. (e) The operator loss on the validation set. (f) The operator loss on the validation set with $\boldsymbol{m}_{opt}$ only. (g) The classification accuracy evaluated on the validation set. Note that Phase II starts when the operator net has been trained for 10,000 batches in Phase I. The final datapoint at $x \approx 32000$ in (e)-(g) corresponds to performance being evaluated on the test set with a 'virtual' batch number around 32k.

of the trained operator net on the test set in the same manner. To show the test performance, we depict the averaged loss evaluated on the test set with all the optimal mask candidates and the optimal mask as well as the accuracy based on the optimal mask at the maximum batch in Figures 5.13(e)-(g). Interestingly, it is seen in Figures 5.13(e)-(g) that the test performance is significantly better than the validation performance in terms of both the losses and the accuracy. This suggests that our alternate learning algorithm yields a favourable generalization performance on this benchmark dataset.

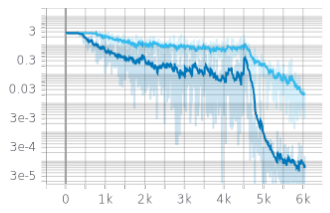
Finally, Figure 5.14 shows the learning behaviour of the operator and the



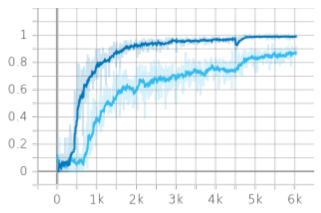
(a) selector-loss



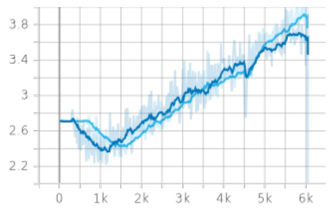
(b) oper.-loss-train



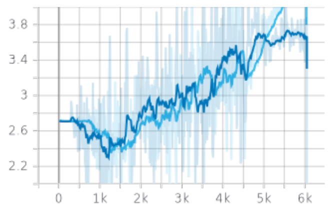
(c) oper.-loss-train($\boldsymbol{m}_{opt}$)



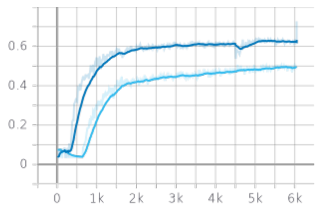
(d) oper.-ACC-train



(e) oper.-loss-val



(f) oper.-loss-val($\boldsymbol{m}_{opt}$)



(g) oper.-ACC-val

Figure 5.14: **Yale Benchmark Dataset**. Evolution of the operator and the selector losses in Phase II ($d = 784, s = 10, 30$). The `x-axis` corresponds to the number of batches and `y-axis` refers to the loss statistics of 5 folds. The light/dark colours correspond to $s = 10/s = 30$, respectively. (a) The selector loss. (b) The operator loss on the training set. (c) The operator loss on the training set with $\boldsymbol{m}_{opt}$ only. (d) The classification accuracy evaluated on the training set. (e) The operator loss on the validation set. (f) The operator loss on the validation set with $\boldsymbol{m}_{opt}$ only. (g) The classification accuracy evaluated on the validation set. Note that Phase II starts when the operator net has been trained for 4,500 batches in Phase I. The final datapoint at $x = 6000$ in (e)-(g) corresponds to performance being evaluated on the test set with a 'virtual' batch number 6k.

selector in terms of losses in Phase II on the Yale benchmark dataset, a *multiclass classification* task. For this facial image dataset, we employ a convolutional neural network described in Table III to carry out the operator net. To understand the learning behaviour better, we compare the situations of the alternate learning for different subset sizes, $s = 10$ and $s = 30$. It is observed from Figure 5.14(a) that the averaged selector loss for different subset sizes behaves quite differently. For $s = 10$, the selector loss sharply decreases at the first few hundred batches and then sharply increases. The limited amount of information carried in 10 out of 1024 features may be accountable for this phenomenon. In contrast, the evolution of selector loss for $s = 30$ is similar to that shown in Figures 5.12(a) and 5.13(a).

Figures 5.14(b)-(d) suggest that the averaged operator loss for different subset sizes keeps decreasing and the accuracy remains increasing on the training set. In contrast, the trend of the averaged operator loss for different subset sizes increases on the validation set after 1.5k batches as shown in Figures 5.14(e) and (f). This looks like a typical overfitting scenario. As seen in Figure 5.14(g), however, the averaged classification accuracy on the validation set generally keeps increasing regardless of different subset sizes. Furthermore, for $s = 30$, the averaged operator test losses and the test accuracy shown in Figures 5.14(e)-(g) (at 6k batches) also provide evidence for good generalization performance. Surprisingly, the alternate learning behaviour on this benchmark dataset contradicts or is inconsistent with the normal behaviour of a learning system. While we do not fully understand such learning behaviour, our preliminary analysis implies that this phenomenon could be caused by the *covariant shift* nature of this facial image dataset and limited training data. In the Yale dataset, the images of an individual subject correspond to different facial expressions. Since there are only limited training examples and the selector learning is constrained by the operator training performance, the stochastic local search from Phase II.A may have to do a lot of exploration in order to find out the "genuine" optimal subset (mask). This can be observed by the fluctuated operator validation loss as shown in Figures 5.14(e) and (f). Thanks to our stochastic exploration-exploitation strategy, some sub-optimal subsets may still direct the learning towards the learning performance at an acceptable level.

In summary, we exhibit typical yet different learning behaviour of our dual-net architecture trained by the alternate learning algorithm in Figures 5.12-5.14. In most of the situations including the one reported in Section 5.1.2 and others not reported here, we can use the operator validation loss evaluated with the optimal mask only for early stopping. On some occasions, however, we encounter some "strange" learning behaviour, as exemplified in Figure 5.14. In such occasions, we might have to use the validation classification accuracy (or validation MSE in regression) for early stopping. Thus, we are going to investigate such "strange" learning behaviour in our ongoing work.

5.3 Discussion

In general, our idea is motivated by RF [8] and the dropout regularization [74]; our exploration-exploitation strategy (c.f. Sect. 4.2) allows for the simultaneous use of different feature subsets and dropout of input "nodes" randomly during learning. Also, we want to make a connection to evolutionary computation (EC) regarding feature selection [75]. In our approach, a single learning model, the operator, works on different feature masks simultaneously during learning to carry out the functionality of a population of learning models in EC. Instead of purely stochastic operations on population in EC, our selector uses a more efficient gradient-guided local stochastic search strategy.

Our approach can be applied to the generic population-wise feature selection problem that needs to find out an optimal feature subset from $\sum_{s=1}^{d-1} \binom{d}{s}$ subsets for a feature set of d features. Instead of a direct search of the entire subset space, we adopt a strategy that makes our model work in parallel on different subset sizes, the same as used in the state-of-the-art supervised feature selection methods, e.g., CCM [5]. To this end, however, our approach might have a higher computational burden than those kernel-based methods in learning. Also, our approach is extensible to group-based FIR and feature selection by introducing the group feature constraints to our stochastic local search procedure (c.f. Sect. 4.2), which would overcome the limitation of linear models, e.g., group LASSO [19].

In conclusion, we propose a dual-net neural architecture along with an alternate learning algorithm to enable deep learning to work effectively for FIR and feature selection. A thorough evaluation manifests that our approach outperforms several state-of-the-art FIR and supervised feature selection methods. In our ongoing work, we would extend our approach to instance-wise FIR, group and unsupervised feature selection scenarios and explore its potential in challenging real applications.

Chapter 6

Feature Importance Estimates With Variable Subset Size

When we realized we needed a new, better model to use for the process of FS, there were several requirements that we wanted to satisfy in comparison to the model described in the previous section. We managed to satisfy all of them.

1. Make the framework work on finding the global optimal subset size so it does not require a chosen subset size as a hyperparameter.

2. Scale down the number of arbitrary solutions in the training/deployment process, which includes things like weighting the training samples or creating more approachable code.

3. Switch from using the input gradients as a measure of importance as it was repeatedly proven (Section 2.3.3) that they are not an ideal tool for that purpose and are costly to compute.

4. Keep the performance-driven FS and FIE nature of the model.

5. Adjust the FIE measurements so that they are more in line with the previously mentioned definition from Section 3.3.

Since this framework and the framework mentioned in Section 4 have similar high-level features, we will continue to call both of the networks Operator and Selector for consistency.

6.1 Model Description

The general framework is identical to the one mentioned in Section 4:

- The framework consists of two NNs working in tandem and trained alternatively.
- One of the networks, called Operator, is used for the supervised, base task for which a given dataset was prepared, while the second one, called Selector, is used to measure FIE and provide the current optimal mask.
- The optimal mask is iteratively enhanced using the knowledge extracted from the Selector.

Obviously, even though the general outlook remains similar to the one from Section 4, the behaviour, training and extraction of FIE are completely different. Below we summarise the main differences in behaviour and provide the particular motivation and consequences of the used solutions:

Variable subset size. The Selector is granted the ability to change the size of the optimal subset. If nothing else were to be changed, the Selector would constantly add features to the optimal subsets. The underlying reason for that behaviour is the fact that increasing the number of features for **the training** set leads to increased **training** performance with a corresponding decrease in **test** performance. To counteract this process, we use the validation performance of the Operator to guide and train the Selector (instead of the training performance).

Decreasing complexity. In order to decrease complexity, we managed to trim the general learning behaviour while keeping the performance. Firstly, we cut the process of weighting the samples going into the Selector. The FIE extraction process was also greatly streamlined (see paragraph below). Finally, we cut back on the validation process of the optimal subset that was previously mentioned in Phase II-A. This resulted in faster and more comprehensible training procedure.

Extraction of FIE. To facilitate the discovery of FIE, we had to change the nature of the Selector's output. Previously, the singular output was based on approximating the function v from Section 3.1. Now, we decided to use the Selector to approximate directly ϕ from Section 3.3, which resulted in the Selector having d outputs, one per feature, which translates to $f_S : \mathcal{M} \to \mathbb{R}^d$ instead of the previous $f_S : \mathcal{M} \to \mathbb{R}$. To smooth the Selector's behaviour (counteract the addition of null features that are accidentally associated with the target) we used the FIE measure of ϕ_λ, as discussed in Section 3.4. The process described in this paragraph also helps with keeping the model **performance-driven** while maintaining a close connection with the **definition of FIE**.

6.2 Learning Algorithm

In this part of the text, we will analyse the new training pipeline. A general overview of the process is displayed in Figure 6.1. Compared to the similar diagram from Chapter 4, we decided to add an additional element, marked in yellow (topmost), that symbolizes the object that generates new mask batches and holds the current optimal mask. To address the changes made in Section 6.1, one can

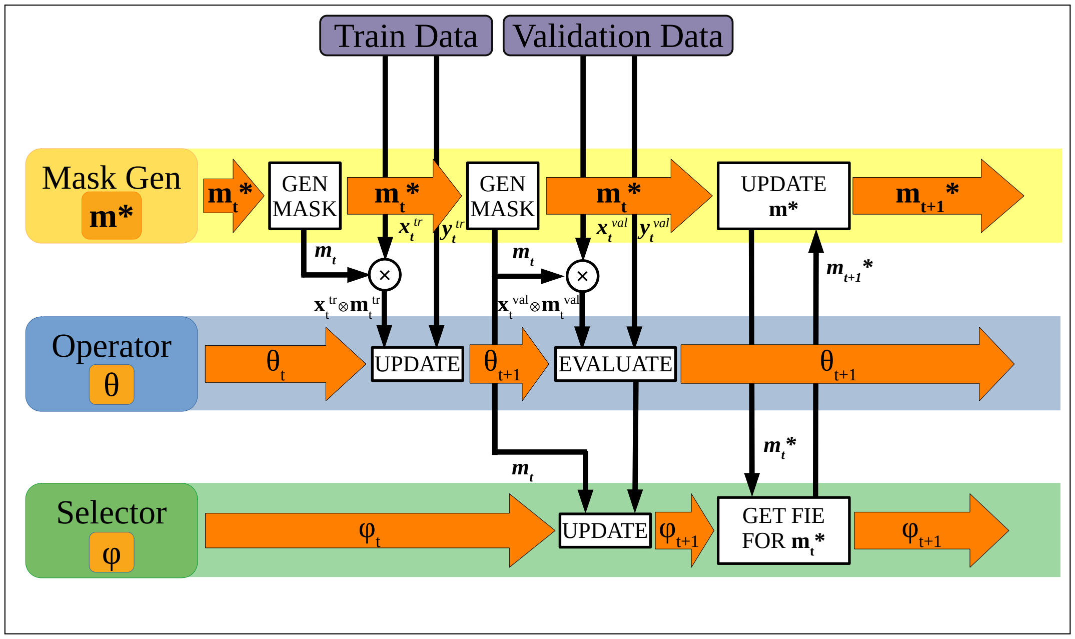


Figure 6.1: Dual-net architecture: diagram showing parameter updates for each of the models. We included an object symbolizing a mask generator which holds the current optimal mask in order to increase clarity.

see on the diagram that the Selector uses validation data during its weight updates to help with the **variable subset size**. Previously, the sensitivity of the Selector's output had to be measured to find the exact feature importance, while now, the Selector outputs the feature importances directly - all done in order to **decrease complexity** and **extract FIE** easier. Finally, we streamlined the mask generation algorithm (the yellow Mask Gen object on the Figure 6.1) to, again, **decrease complexity**.

Below, we summarize the new algorithm in more detail, briefly going through the unchanged Operator, streamlined mask generation, different choices for selector loss and easy feature importance extraction. This is the order of operations when looking from the left to the right of Figure 6.1. Finally, we perform a theorethical analysis of the expected optimal feature subset size to understand the meaning behind the λ hyperparameter.

6.2.1 Operator Loss

The description below summarizes the working of the Operator, which did not change from the description provided in 4, apart from minute changes in notation. As a reminder, we have a dataset $\mathcal{D} = \{\mathcal{X}, \mathcal{Y}\} = \{(\boldsymbol{x}, \boldsymbol{y})\}$ that can be split into disjointed sets of training data $\mathcal{D}_{tr} = \{\mathcal{X}_{tr}, \mathcal{Y}_{tr}\}$ and validation data $\mathcal{D}_{val} = \{\mathcal{X}_{val}, \mathcal{Y}_{val}\}$. We also use a number of feature subsets $\mathcal{M}'$ which are then used to create $|\mathcal{M}'|$ samples for each $\boldsymbol{x} \in \mathcal{X}$ by applying each mask to every datapoint: $\{(\boldsymbol{x} \otimes \boldsymbol{m}, \boldsymbol{y})\}_{\boldsymbol{m} \in \mathcal{M}'}$. The loss function of the operator stays the same and is defined as:

$$\mathcal{L}_O(\mathcal{D}, \mathcal{M}'; \theta) = \frac{1}{|\mathcal{M}'||\mathcal{D}|} \sum_{\boldsymbol{m} \in \mathcal{M}'} \sum_{(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{D}} l(\boldsymbol{x} \otimes \boldsymbol{m}, \boldsymbol{y}; \theta). \quad (6.1)$$

where l corresponds to task-specific (dictated by the dataset) loss function. We decided on using commonly accepted losses like MSE for the regression tasks and cross-entropy for binary and categorical classification tasks.

6.2.2 Choosing Feature Subsets To Train On

Summary: To briefly summarize the section below: the first phase is a pretraining phase so that the Operator outputs can stabilize to not affect the Selector so much. Then, during the main phase, the selector provides new mask batches from three sources:

1. best from the last batch,
2. randomly explore the neighbourhood of the mask space,

3. exploit the selector to predict the prospective changes to available masks.

We also provide a mechanism that regulates the temperature of the exploration-exploitation algorithm.

Pretraining: During the main training loop, the Selector uses its predictions to discern which features should be added/removed. At the very beginning of the training process, both the Operator and the Selector have little knowledge about the nature of data which might result in removing useful features at the beginning of the training. Obviously, the removed features might be added later on, but in order to increase the stability of training and streamline the process we have chosen to include a **pretraining** phase for the Operator, which usually takes around a few hundred training steps. During this phase, the feature subsets through which the inputs of the Operator are masked are chosen randomly using a binomial distribution $B(n = 1, p = 0.7)$, and not decided by the selector. The p was chosen arbitrarily as choosing too small p would result in the Operator not finding any useful predictors for hard datasets (especially for datasets like XOR where the synergy between the features is the only useful predictor) while choosing p too large would result in little variation between the masks. We found that the pretraining process is important in jump-starting the Selector's training, but its length is not something that requires any tuning and is usually kept at around 100 training steps.

Notation: The mask batch $\mathcal{M}'_t$ at time t is made out of $M = |\mathcal{M}'_t|$ masks, where:

$$M \in \{a|(\exists b \in \mathbb{N})[a = 4b]\} \tag{6.2}$$

and is set as $M = 64$ for most experiments (unless stated otherwise). As a reminder: $\mathcal{M}' = \{\boldsymbol{m}_i \in \{0,1\}^d \mid 0 \leq i < M\}$. For the clarity of explanations we divide $\mathcal{M}'$ into three sets:

$$\begin{aligned}
Q_1 &= \{\boldsymbol{m}_i \in \mathcal{M}' \mid i \in \{0, 1, ..., \frac{M}{4} - 1\}\} \\
Q_2 &= \{\boldsymbol{m}_i \in \mathcal{M}' \mid i \in \{\frac{M}{4}, \frac{M}{4} + 1, ... \frac{M}{2} - 1\}\} \\
H_2 &= \{\boldsymbol{m}_i \in \mathcal{M}' \mid i \in \{\frac{M}{2}, \frac{M}{2} + 1, ..., M - 1\}\}
\end{aligned}$$

where Q_1^t, Q_2^t, H_2^t correspond to the first quarter, second quarter and the second half of $\mathcal{M}'_t$ at step t.

Mask Batch Generation: We find Q_1^t, Q_2^t, H_2^t at step t from:

$$\begin{aligned}
Q_1^t &= \{f_{comp}(\boldsymbol{m}_i, \boldsymbol{m}_{i+\frac{M}{4}}, \mathcal{L}_O(\mathcal{D}, \boldsymbol{m}_i, \theta_t), \mathcal{L}_O(\mathcal{D}, \boldsymbol{m}_{i+\frac{M}{4}}, \theta_t)) \mid \boldsymbol{m} \in \mathcal{M}'_{t-1} \wedge 0 \le i < \frac{M}{4}\} \\
Q_2^t &= \{f_{exploit}(\boldsymbol{m}, f_S(\boldsymbol{m}, \phi_t) - \lambda \boldsymbol{m}, n_\lambda) \mid \boldsymbol{m} \in Q_1^t\} \\
H_2^t &= \{f_{explore}(\boldsymbol{m}, p_{explore}) \mid \boldsymbol{m} \in Q_1^t \cup Q_2^t\}
\end{aligned}$$

where λ, n_λ and $p_{explore}$ are scalar hyperparameters, θ_t, ϕ_t are weights of the Operator and Selector respectively and

$$\begin{aligned}
f_{comp} &: \{0,1\}^d \times \{0,1\}^d \times \mathbb{R} \times \mathbb{R} \to \{0,1\}^d, \\
f_{exploit} &: \{0,1\}^d \times \{0,1\}^d \times \mathbb{N} \to \{0,1\}^d, \\
f_{explore} &: \{0,1\}^d \times \mathbb{R} \to \{0,1\}^d
\end{aligned}$$

are functions used to construct ($f_{exploit}, f_{explore}$) new feature subsets. They alter the mask fed as input or choose (f_{comp}) one of the two masks based on the Operator's performance.

In the case of $t = 0$; when $\mathcal{M}'_{t-1}$ is not accessible for generating Q_1^t we start with masks generated during pretraining phase:

$$Q_1^t = \{\boldsymbol{m}_i \in \{0,1\}^d \mid i \in \{0, 1, ... \frac{M}{4} - 1\} \ \wedge m_{i,j} \sim B(1, 0.7)\}. \tag{6.3}$$

The functions $f_{comp}, f_{explore}$ are defined as:

$$f_{comp}(\boldsymbol{m}_A, \boldsymbol{m}_B, l_A, l_B) = \begin{cases} \boldsymbol{m}_A, & \text{if } l_A < l_B \\ \boldsymbol{m}_B, & \text{otherwise.} \end{cases} \tag{6.4}$$

$$f_{explore}(\boldsymbol{m}, p_{explore}) = (c_i : c_i = m_i * (1 - a_i) + (1 - m_i) * a_i) \tag{6.5}$$

where a_i are realizations of p.d.f. $A \sim B(1, p_{explore})$ and m_i corresponds to i-th component of $\boldsymbol{m}$.

To define $f_{exploit}(\boldsymbol{m}, \boldsymbol{y}, n_\lambda)$ (where we used $\boldsymbol{y} = f_S(\boldsymbol{m}, \phi_t) - \lambda\boldsymbol{m}$ as a vector of outputs from the Selector diminished by λ for the components where the feature is present in $\boldsymbol{m}$) we first need to define Y^{sorted} as a reindexed sequence of $\boldsymbol{y}$ with monotonically increasing values. Then we define the set of n_λ smallest values $Y_{small} = \{Y_i^{sorted} \mid 0 \leq i < n_\lambda\}$. Next, we construct a binary vector $\boldsymbol{b}$ of length d that shows whether the corresponding component of $\boldsymbol{y}$ is in Y_{small}:

$$b_i = \begin{cases} 1, & \text{if } y_i \in Y_{small} \\ 0, & \text{otherwise.} \end{cases}$$

Finally we can define the output mask of $f_{exploit}$ as a vector $\boldsymbol{d} = (d_i : d_i = m_i * (1 - b_i) + (1 - m_i) * b_i)$. In other words, we add/remove n_λ features that are perceived by the Selector's output (regularized with λ) to be the most/least useful.

Summary so far: The motivation and meaning behind the process described above can be summarized in 4 steps for at time t:

1. First, we pick a quarter of masks, Q_1^t, from the output of step 4. from step $t-1$. If $t = 0$, then we sample the masks from a binomial distribution similar to the **pretraining** phase.

2. Secondly, we **exploit** what the Selector learned to try and find the best prospective masks laying n_λ l_1-distance away from Q_1^t to construct Q_2^t.

3. Next, we **explore** the neighbourhood of Q_1^t and Q_2^t using the probability $p_{explore}$ to remove/add every feature.

4. Finally, we **compare** the validation results obtained from the Operator for Q_1^t and Q_2^t to construct the Q_1^{t+1} for the next iteration.

As described above, our strategy includes the **exploration-exploitation dilemma** that is widely known to the reinforcement learning community [76]. It is common to use different approaches to this problem, like ϵ-greedy strategy, to establish a necessary balance between exploration and exploitation and even change it during

execution. The pretraining phase described at the very beginning of this section can be thought of as a phase that is fully dedicated towards the exploration of all possible masks, with no exploitation of the Selector's knowledge.
The whole process is written down in a pseudo-code format in Algorithm 3, where the comments mark the parts corresponding to $f_{compare}, f_{exploit}, f_{explore}$.

Adaptive Exploration The algorithm described above has proven to bring results that are good enough, but we noticed that the Q_1^t quickly converges to a set of identical masks. That also resulted in feature subsets in Q_2^t being identical, as $f_{exploit}$ that is used for its creation is fully deterministic. We found that decreasing the exploitation by introducing some randomness into the $f_{exploit}$ increased the quality of results - this can be thought of as probing the local neighbourhood of Q_1^t in the direction that is not fully random but mainly conditioned by the Selector (in comparison, $f_{explore}$ used to create H_2^t is probing in completely random directions). While the results for these approaches are identical when the fraction of useful features in the dataset is relatively big (around 5%), the probability-based approach shows its strength when dealing with a much smaller signal-to-noise ratio (around 1% of useful features). In the actual implementation of the process described above, instead of sorting the indices of $\boldsymbol{y} = f_s(\boldsymbol{m}, \phi_t) - \lambda\boldsymbol{m}$, we use $\boldsymbol{y}$ to randomly chose indices according to the probability of $\text{Softmax}(-\boldsymbol{y})$.
This algorithm can only be applied in a safe manner when the f_s outputs are scaled to a valid range. As we will find out later, that is only possible for **classification-based** loss that is described in Section 5.2.4.

6.2.3 Operator Performance Evaluation

Due to the batch creation process described above, at the end of the training, we end up with several optimal subsets in Q_t^1 which sometimes differ amongst themselves by one or two features. At first glance, it might seem prudent to choose the one with the best validation performance as the optimal one, but we found out that using all of them as an ensemble through the aggregation of logit values (or just the linear activations for regression) gives slightly better results in

Algorithm 3 Generation of Optimal Feature Subset for $t > 0$

```
Require: selector net $f_S(\varphi, \boldsymbol{m})$,
Require: operator net's loss $\mathcal{L}_O(\mathcal{D}, \boldsymbol{m}, \theta_t)$
Require: number of exploitation changes $n_\lambda$,
Require: feature subset size regularization parameter $\lambda$,
Require: probability of exploration changes $p_{exploration}$,
Require: first half of the mask batch from step $t-1$: $Q_1^{t-1}$ and $Q_2^{t-1}$,
Require: number of masks in the batch $M$,
                                                        ▷ $f_{compare}$
1: for $i$ in range($\frac{M}{4}$) do
2:     if $\mathcal{L}_O(\mathcal{D}, Q_{1,i}^{t-1}, \theta_t) > \mathcal{L}_O(\mathcal{D}, Q_{2,i}^{t-1}, \theta_t)$ then
3:         $Q_{1,i}^t \leftarrow Q_{2,i}^{t-1}$
4:     else
5:         $Q_{1,i}^t \leftarrow Q_{1,i}^{t-1}$
6:     end if
7: end for
8: $Q_2^t \leftarrow Q_1^t$                                  ▷ $f_{exploit}$
9: for $i$ in range($\frac{M}{4}$) do
10:    if $random\_exploitation$ then
11:        $\boldsymbol{y}_{softmax} \leftarrow \text{Softmax}(f_S(\varphi, Q_{1,i}^t) - \lambda Q_{1,i}^t)$
12:        $idcs \leftarrow \text{random\_choice}(\text{range}(d), n_{choices} = n_\lambda, p = \boldsymbol{y}_{softmax})$
13:    else
14:        $idcs \leftarrow \text{argsort}(f_S(\varphi, Q_{1,i}^t) - \lambda Q_{1,i}^t)[: n_\lambda]$
15:    end if
16:    $Q_{2,i}^t[idcs] \leftarrow \text{logical\_not}(Q_{2,i}^t[idcs])$
17: end for
18: $\boldsymbol{m}_{pert} \leftarrow \text{Binomial}(1, p_{explore})$          ▷ $f_{explore}$
19: $H_2^t \leftarrow \text{Concatenate}([Q_1^t, Q_2^t], axis = 0)$
20: $H_2^t \leftarrow H_2^t * (1 - \boldsymbol{m}_{pert}) + (1 - H_2^t) * \boldsymbol{m}_{pert}$
21: $\mathcal{M}_t' \leftarrow \text{Concatenate}([Q_1^t, Q_2^t, , H_2^t], axis = 0)$     ▷ Build final mask batch
```

general.

6.2.4 Selector Loss

For the selector, we tested four different losses. To be able to understand the differences between them, below we describe different approaches to the task of FIE for the Selector.

Classification vs Regression Firstly, the task of the Selector can be achieved by both Regression or Classification. The regression approach focuses on trying to predict the loss difference if a given feature was added to/removed from the input feature subset. On the other hand, the classification approach tries to predict whether adding/removing the feature would result in an increase of Operator's performance. Both of these methods are closely tied to the definition of FIE from Section 3.3, but while the regression is trying to predict ϕ directly, the classification tries to predict its sign. We use MSE for regression and Binary Cross-Entropy (Equation 6.6) for classification to measure the appropriate errors. The Binary Cross-Entropy is defined as:

$$L_H(\boldsymbol{y}^{true}, \boldsymbol{y}^{pred}) = \frac{1}{|N|} \sum_{i=0}^{|N|} y_i^{true} \log y_i^{pred} + (1 - y_i^{true}) \log (1 - y_i^{pred}), \tag{6.6}$$

where $\boldsymbol{y}^{true}, \boldsymbol{y}^{pred}$ are vectors of labels and predictions correspondingly, both of length N, such that $\boldsymbol{y}^{true} \in \{0,1\}^N$ and $\boldsymbol{y}^{pred} \in [0,1]^N$. For comparison, the MSE loss is defined as

$$L_{MSE}(\boldsymbol{y}^{true}, \boldsymbol{y}^{pred}) = \frac{1}{2|N|} \sum_{i=0}^{|N|} (y_i^{true} - y_i^{pred}), \tag{6.7}$$

where $\boldsymbol{y}^{true} \in \mathbb{R}^N$ and $\boldsymbol{y}^{pred} \in \mathbb{R}^N$.

One-to-one vs Many-to-many Additionally, the feature subsets (masks), and the corresponding Operator's performances, can be presented in two different ways to the Selector. The masks can be fed to the Selector without any modifications so that the second half of the batch is composed of the first half with added binomial noise (more on that in Section 6.2.2) so that there is a clear one-to-one correspondence between each mask and its noisy version. However, assuming that there are N masks in a batch, we would only get N different relative loss measurements for the Selector to learn from. We will refer to this approach as **one-to-one**. Now, if instead we measure the relative difference between every mask in a batch we will end up with $N^2 - N$ relative loss measurements, which is $N-1$ times more than in the case of the one-to-one approach. We will call this new approach **many-to-many**. To fully understand why the many-to-many approach might suffer in comparison to one-to-one, we will first need to understand one of the other techniques we used to be able to train the Selector on datasets with hundreds of features, which is outlined below.

Aggregation of FIE For now, we only discussed subsets which differ by one feature. While it is possible to feed the Selector masks that differ by only one feature in a one-to-one scenario, the number of Selector's outputs that are affected (through which we backpropagate the gradients) stays at one per sample. This approach is sufficient for tasks with tens of features but starts to suffer when we reach hundreds of features and fails when the number of features goes into thousands, as we need to remember that number of output neurons for the selector is d, and the more frequently we update their weights the better Selector's performance will be. To counteract these sparse weight updates, we decided to make it possible for the mask pairs to differ by more than one feature, even though it allows the model to 'overlook' some feature associations. This allows us to arrive at the final formula for the Selector's losses. For regression-based loss, we have:

$$\mathcal{L}_S^{reg}(\mathcal{M}';\varphi) = \frac{1}{|\mathcal{M}^*|} \sum_{(\boldsymbol{m}^j,\boldsymbol{m}^k)\in\mathcal{M}^*} L_{MSE}\Big(\mathcal{L}_O(\mathcal{D}';\{\boldsymbol{m}^{\text{k}}\};\theta) - \mathcal{L}_O(\mathcal{D}';\{\boldsymbol{m}^{\text{j}}\};\theta); \sum_{i=0}^{d} f_S(\varphi;\boldsymbol{m}^j)_i * (m_i^k - m_i^j)\Big), \quad (6.8)$$

where $\mathcal{M}^*$ is the set of pairs of masks $(\boldsymbol{m}^j,\boldsymbol{m}^k)$ fed into the Selector and dependent on the **one-to-one** or **many-to-many** approach, $\mathcal{D}'$ is a batch of pairs $(\boldsymbol{x},\boldsymbol{y})$ extracted from the actual dataset and index i corresponds to different elements of $\boldsymbol{m}$.

For the classification approach, we have the following:

$$\mathcal{L}_S^{clf}(\mathcal{M}';\varphi) = \frac{1}{|\mathcal{M}^*|} \sum_{(\boldsymbol{m}^j,\boldsymbol{m}^k)\in\mathcal{M}^*} L_H\Big(\text{sgn}\big(\mathcal{L}_O(\mathcal{D}';\{\boldsymbol{m}^{\text{k}}\};\theta) - \mathcal{L}_O(\mathcal{D}';\{\boldsymbol{m}^{\text{j}}\};\theta)\big); \sum_{i=0}^{d} f_S(\varphi;\boldsymbol{m}^j)_i * (m_i^k - m_i^j)\Big), \quad (6.9)$$

where $\text{sgn}(x)$ is the signum function.

For the **one-to-one** approach, we define the set of pairs:

$$\mathcal{M}^* = \{(\mathcal{M}'_j, \mathcal{M}'_k) \mid j \in \{0,1,2,...,M-1\}; k = j + \frac{M}{2} \mod M\} \quad (6.10)$$

given mask batch $\mathcal{M}'$ being treated as a vector of length M, so that each mask is paired with another msk whose index is $\frac{M}{2}$ greater/lesser.

For the **many-to-many** approach, we have:

$$\mathcal{M}^* = \{(\boldsymbol{m}^j,\boldsymbol{m}^k) \mid \boldsymbol{m}^j \in \mathcal{M}'; \boldsymbol{m}^k \in \mathcal{M}'; j \neq k\}. \quad (6.11)$$

In this case, every mask is paired with every other mask in a batch.

Now, when comparing these approaches, the masks for the **one-to-one** approach differs, by average, by $p_{explore} * n$. That is not the case for **many-to-many**, where there is no such rule.

6.2.5 Extracting Feature Importance

FIE extraction is a straightforward process and is given by the Selector outputs. The input of the selector is the batch of masks from Q_t^1. The feature importance may differ slightly if the masks in Q_t^1 are different so to obtain the final result we average over Q_t^1.

6.2.6 Expected Optimal Feature Subset Size

The purpose of this section is an estimation of the number of null features that are included in the optimal mask, be it due to accidental association or **Adaptive Exploration**, where the optimal mask extraction is probability-based.
The mechanism that outputs the optimal subset (detailed in the sections above) also usually makes the optimal mask contain null features. They are easy to identify because their importance might be slightly above 0 - as mentioned before, some features are accidentally associated with the target variable (for both training and validation data). We regulate the number of null features present by changing the λ parameter. As a reminder, this parameter regularizes the optimal mask. The probability vector of removing/adding each feature for **Adaptive Exploration** is given by Softmax$(-f_S(\boldsymbol{m}, \phi_t) + \lambda \boldsymbol{m})$. The probability p of adding/removing feature j at time t is equal to:

$$p(j, t, \boldsymbol{m}) = \frac{e^{-f_S(\boldsymbol{m}, \phi_t)_j + \lambda m_j}}{\sum_{i=0}^{d} e^{-f_S(\boldsymbol{m}, \phi_t)_i + \lambda m_i}}, \tag{6.12}$$

where the subscripts i, j mark the i-th and j-th elements of a vector.

Now, let us calculate the expected size of the optimal subset when the system reaches equilibrium, the Selector stops learning, and the size of the optimal mask stops changing:

$$\sum \boldsymbol{m}_t = \mathbb{E}(\sum \boldsymbol{m}_{t+1} | d, \phi_t, \lambda), \tag{6.13}$$

where d corresponds to the number of features in the dataset. Using linearity of expectation:

$$\sum \boldsymbol{m}_t = \sum_{i=0}^{d} \mathbb{E}(m_{i,t+1} | d, \phi_t, \lambda). \tag{6.14}$$

Before we continue, let us assume that we are using the predictions of an ideal Selector. In that case, the output of the Selector for the essential features would be $-\infty$, knowing that the feature is present in the optimal mask ($-\infty$ corresponds to a probability of 0% of removing that feature from the optimal mask). We do not make any assumptions regarding the presence of the null features, as depending on the sampled batch, they can be considered helpful, as outlined below.

It can be useful to calculate the lower bound of the association of the sampled null features, which can be understood as the standard error of Pearson's correlation:

$$\sigma_r = \sqrt{\frac{1-r^2}{n-2}}, \tag{6.15}$$

where r is the correlation (assume $r \approx 0$), and n is the number of samples on which the correlation was measured. In our case, the true correlation coefficient $\rho = 0$ for the null features. We can then approximate the correlation error, knowing that the typical batch size in our analysis is 32 and get $\sigma_r \approx 0.1826$.

Getting back to our main analysis, let us assume without a loss of generality that the features present in $\boldsymbol{m}_t$ are sorted according to their perceived importance calculated by the ideal Selector: starting from n_{true} useful features that have to be present in $\boldsymbol{m}_t$ (ideal Selector), $n^1_{null} = |\boldsymbol{m}_t| - n_{true}$ null features present in $\boldsymbol{m}_t$ and finally $n^0_{null} = d - |\boldsymbol{m}_t|$ features that are not present in the optimal mask. Now, expanding on Equation 6.14:

$$\begin{aligned}
|\boldsymbol{m}_t| &= \sum_{i=0}^{d} \mathbb{E}(m_{i,t+1}|d,\phi_t,\lambda) \\
&= \sum_{i=0}^{n_{true}} \mathbb{E}(m_{i,t+1}|d,\phi_t,\lambda) + \sum_{i=n_{true}}^{|\boldsymbol{m}_t|} \mathbb{E}(m_{i,t+1}|d,\phi_t,\lambda) + \sum_{i=|\boldsymbol{m}_t|}^{d} \mathbb{E}(m_{i,t+1}|d,\phi_t,\lambda) \\
&= n_{true}(1-p_{true}(d,\phi_t,\lambda)) + n^1_{null}(1-p^1_{null}(d,\phi_t,\lambda) + n^0_{null} * p^0_{null}(d,\phi_t,\lambda) \\
&= n_{true} + n^1_{null} - n^1_{null} * p^1_{null}(d,\phi_t,\lambda) + n^0_{null} * p^0_{null}(d,\phi_t,\lambda) \\
&= |\boldsymbol{m}_t| - n^1_{null} * p^1_{null}(d,\phi_t,\lambda) + n^0_{null} * p^0_{null}(d,\phi_t,\lambda)
\end{aligned} \tag{6.16}$$

where the $p_{true}, p^1_{null}, p^0_{null}$ are the probabilities of removing/adding the corresponding feature during the next iteration. Additionally, if we use the ideal Selector, then $p_{true}(d,\phi_t,\lambda) = 0$ to ensure that the useful features are included in the next iteration. Finally, we can cancel the $|\boldsymbol{m}_t|$ terms from both sides to obtain the equilibrium condition:

$$\begin{aligned} n^1_{null} * p^1_{null}(d, \phi_t, \lambda) &= n^0_{null} * p^0_{null}(d, \phi_t, \lambda) \\ \frac{|\boldsymbol{m}_t| - n_{true}}{d - |\boldsymbol{m}_t|} &= \frac{p^0_{null}(d, \phi_t, \lambda)}{p^1_{null}(d, \phi_t, \lambda)} \end{aligned} \tag{6.17}$$

The next step is an approximation of p^1_{null}, p^0_{null}, which are given by the following expression in the Softmax case:

$$\begin{aligned} \frac{p^0_{null}(d, \phi_t, \lambda, \boldsymbol{m}_t)}{p^1_{null}(d, \phi_t, \lambda, \boldsymbol{m}_t)} &= \frac{\frac{\mathbb{E}(e^{-f_S(\boldsymbol{m}_t, \phi_t)_j} | m_{j,t}=0)}{\mathbb{E}(\sum_{i=0}^d e^{-f_S(\boldsymbol{m}_t, \phi_t)_i + \lambda m_{i,t}})}}{\frac{\mathbb{E}(e^{-f_S(\boldsymbol{m}_t, \phi_t)_j + \lambda} | m_{j,t}=1)}{\mathbb{E}(\sum_{i=0}^d e^{-f_S(\boldsymbol{m}_t, \phi_t)_i + \lambda m_{i,t}})}} \\ &= \frac{\mathbb{E}(e^{-f_S(\boldsymbol{m}_t, \phi_t)_j} | m_{j,t} = 0)}{\mathbb{E}(e^{-f_S(\boldsymbol{m}_t, \phi_t)_j + \lambda} | m_{j,t} = 1)} \\ &= \frac{\mathbb{E}(e^{-f_S(\boldsymbol{m}_t, \phi_t)_j} | m_{j,t} = 0)}{\mathbb{E}(e^{-f_S(\boldsymbol{m}_t, \phi_t)_j} | m_{j,t} = 1)} * e^{-\lambda} \end{aligned} \tag{6.18}$$

For the null feature to be useful, it has to randomly display the same dependence on the target variable in the training and validation sets. Additionally, the model has to be sensitive enough to use the provided information. It is out of the scope of this work to calculate the probabilities through the above analysis, even in the linear case. To approximate the probabilities, we notice that for a big enough number of training and validation examples and for the ideal Selector:

$$\mathbb{E}(f_S(\boldsymbol{m}_t, \phi_t)_j | m_{j,t} = 0) \approx \mathbb{E}(f_S(\boldsymbol{m}_t, \phi_t)_j | m_{j,t} = 1) \tag{6.19}$$

so that we have:

$$\frac{p^0_{null}(d, \phi_t, \lambda, \boldsymbol{m}_t)}{p^1_{null}(d, \phi_t, \lambda, \boldsymbol{m}_t)} = \frac{\mathbb{E}(e^{-f_S(\boldsymbol{m}_t, \phi_t)_j} | m_{j,t} = 0)}{\mathbb{E}(e^{-f_S(\boldsymbol{m}_t, \phi_t)_j} | m_{j,t} = 1)} * e^{-\lambda} \approx e^{-\lambda} \tag{6.20}$$

Using Equation 6.20 we can solve Equation 6.17 for $|\boldsymbol{m}_t|$:

$$\begin{aligned} \frac{|\boldsymbol{m}_t| - n_{true}}{d - |\boldsymbol{m}_t|} &= e^{-\lambda} \\ |\boldsymbol{m}_t| - n_{true} &= (d - |\boldsymbol{m}_t|) * e^{-\lambda} \\ |\boldsymbol{m}_t| + |\boldsymbol{m}_t| * e^{-\lambda} &= d * e^{-\lambda} + n_{true} \end{aligned}$$

$$|\boldsymbol{m}_t| = \frac{d * e^{-\lambda} + n_{true}}{1 + e^{-\lambda}} \tag{6.21}$$

or in terms of the fraction of the total mask size:

$$\frac{|\boldsymbol{m}_t|}{d} = \frac{e^{-\lambda} + \frac{n_{true}}{d}}{1 + e^{-\lambda}} \tag{6.22}$$

where $\frac{n_{true}}{d}$ is the fraction of the useful features. Figure 6.2 illustrates the derived dependency as a function of λ. We can see that the optimal mask size converges to n_{true} as λ increases since:

$$\lim_{\lambda \to \inf} \frac{|\boldsymbol{m}_t|}{d} = \frac{n_{true}}{d} \tag{6.23}$$

This analysis summarizes why we use $\lambda = 3$, which is a value for which around

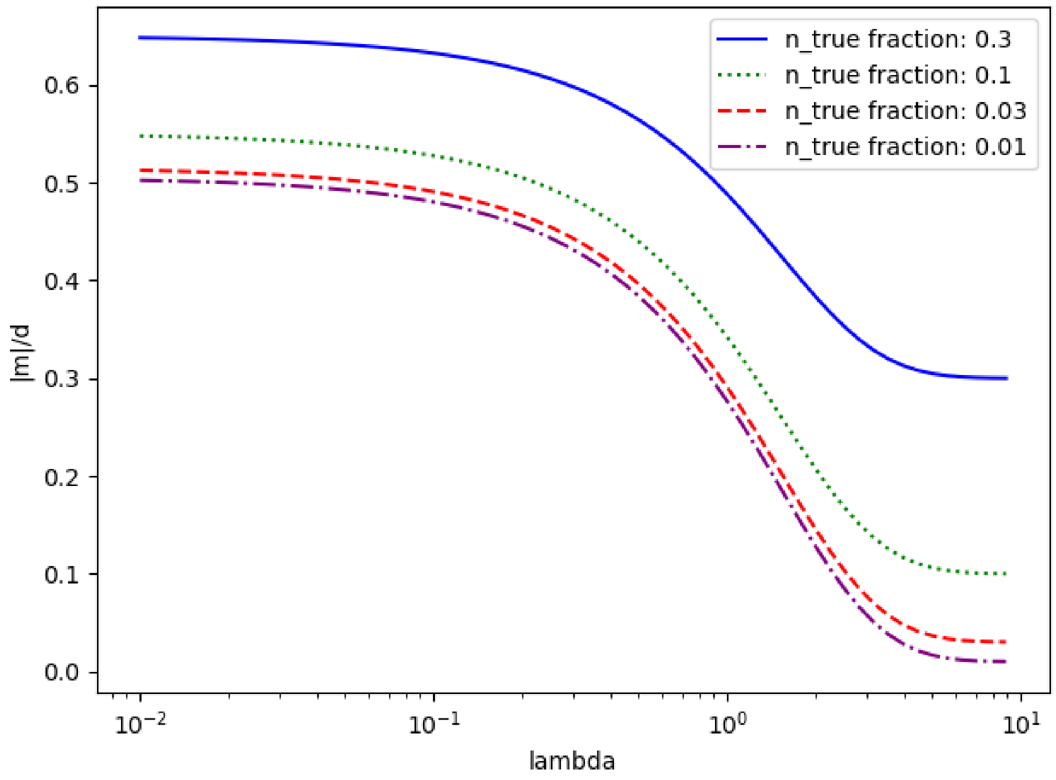


Figure 6.2: The fractional size of the optimal mask as a function of λ for different fractions of useful features.

90% of the null features should be removed. In other words, this hyperparameter allows us to **control the FDR** (false discovery rate).

6.3 Results

In this section, similarly to the previous method, we evaluate our approach on synthetic, benchmark and real-world datasets.

Firstly, we will focus on results from three types of datasets: **synthetic**, **visual** and **benchmark**. We used the same datasets as described in Section 5.1 - be it synthetic, visual or benchmark data. One of the differences in our analysis is the

change of the focus of **synthetic** data, where we extend the datasets to thousands of null features to examine which of the Selector loss functions from the ones mentioned in Section 6.2.4. We use **visual** data to validate the FIE results produced by the Selector and finally measure the framework's performance on **banchmark** data against other methods.
Then, we will give a summary of the experimental details that are used across all experiments. That includes the value of the λ parameter, used architectures, and hyperparameter choices.
Ultimately, we will analyse the learning behaviour of the system for any irregularities, as well as perform a stability study between different folds of the dataset.

6.3.1 Synthetic Datasets - Choosing Selector Loss

Loss Function Performance Comparison

To showcase model capabilities as well as to compare different Selector's loss function approaches (**many-to-many** vs **one-to-one** and **classification** vs **regression**), we decided to design an experiment with synthetic data where the number of null features is geometrically increased, which decreases the signal-to-noise ratio. We used the same three synthetic datasets (XOR, OrangeSkin, SynthRegression) as in Chapter 4, but we were able to generate more null features than 7. We tested four different loss function configurations:

1. **One-to-one classification**,
2. **Many-to-many classification**,
3. **One-to-one regression**,
4. **Many-to-many regression**.

We varied the total number of features for each dataset by doubling it at every step: 10,20,...,320. We found that $N_{explore} = 100$ is enough for the Operator and the Selector to achieve satisfying results and tuning it does not affect the performance. During the cross-validation procedure, we test several Operator and Selector architectures with no more than three hidden layers and different values of $\lambda \in \{0.03, 0.1, 0.03, 0.1, 0.3, 1.0, 3.0\}$. We set the early stopping to $N_{explore}$ epochs.

We used a grid-search strategy to obtain the optimal hyperparameters based on the validation performance.

The figures 6.3, 6.4, 6.5 show the results of the experiment by comparing the test accuracy/MSE for each dataset and loss function for a different number of input features.

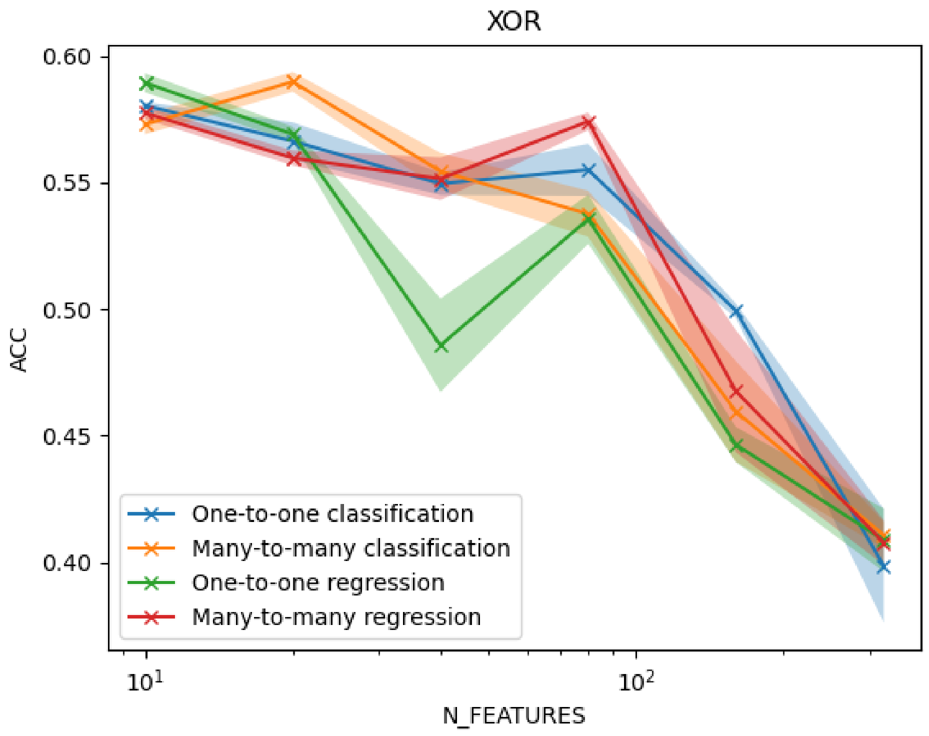


Figure 6.3: XOR dataset results. Y-axis corresponds to test accuracy, while the X-axis displays the number of features used for each analysis.

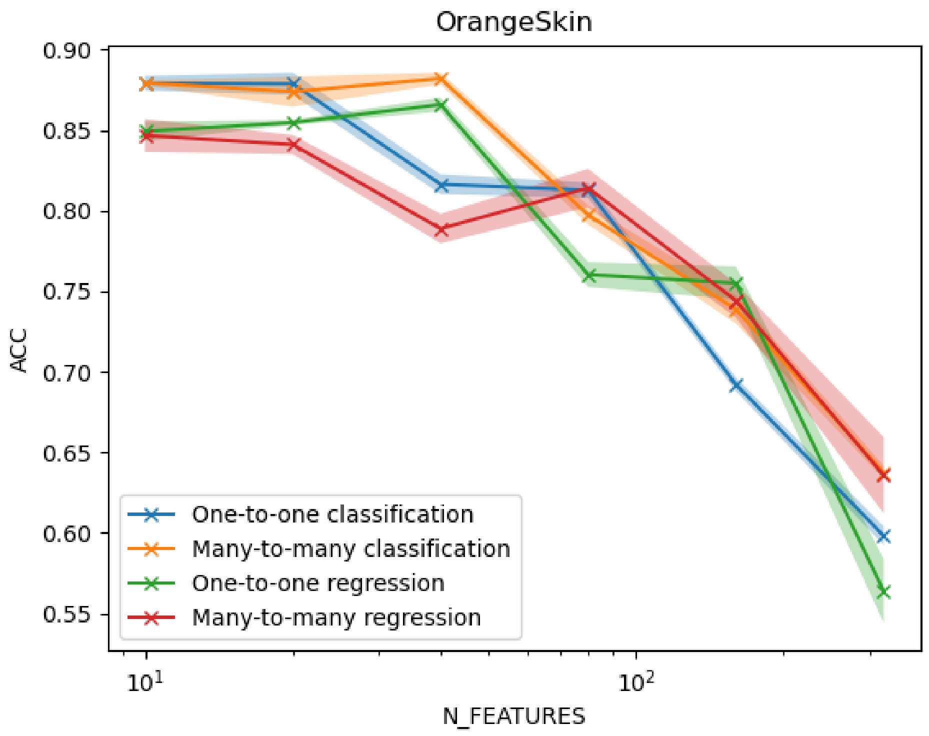


Figure 6.4: Orange Skin dataset results. Y-axis corresponds to test accuracy, while the X-axis displays the number of features used for each analysis.

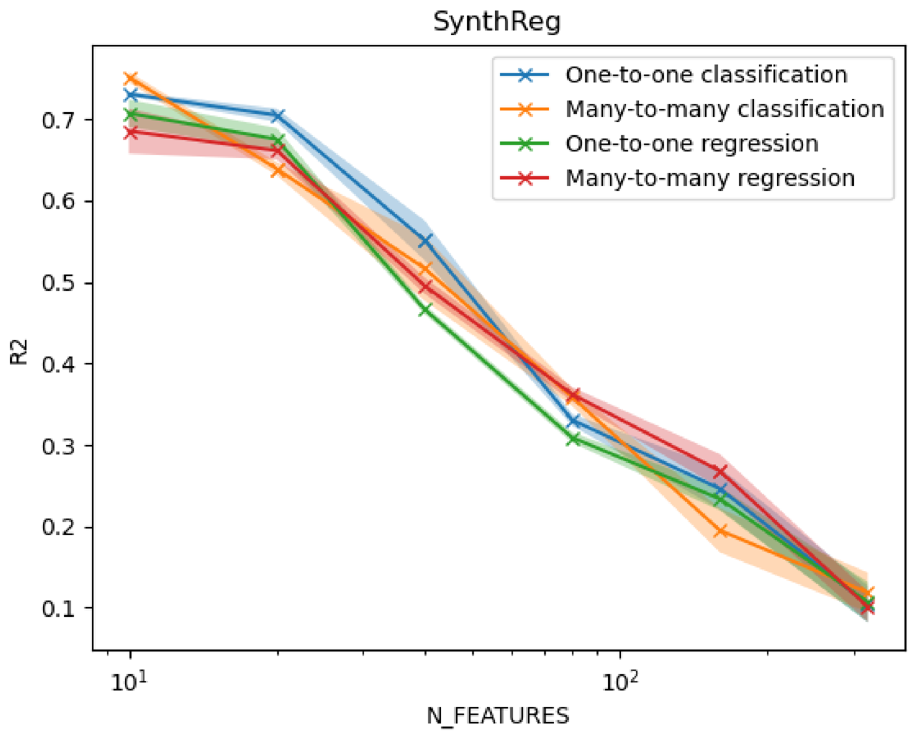


Figure 6.5: Synthetic Regression dataset results. Y-axis corresponds to test R^2, while the X-axis displays the number of features used for each analysis.

It is clear that the **classification-based** loss function produces better results, with a slight advantage towards the **one-to-one** option, which is why we will use this setting in future analyses.

Additionally, we found that the **classification-based** loss provides a clear interpretation of the λ value so that it does not have to be tuned as a hyper-parameter for a given dataset, as seen in Section 6.2.6. On the contrary, the **regression-based** loss cannot use **Adaptive Exploration** without careful loss scaling and fine-tuning of λ to produce reasonable results.

Loss Function Feature Ranking Comparison

The next test designed to compare different loss functions included comparing the average rank of the important features for each dataset. The rank is averaged over the 5-fold cross-validation process and recorded for different numbers of features, similar to the previous subsection. We decided against displaying the feature importances directly for all the features due to the exponentially scaling number of features used in our experiments.

The figures 6.6, 6.7, 6.8 show the measurements of the average rank of the important features for each dataset. Additionally, a theoretical minimum rank that depends solely on the number of important features for each dataset was added.

The first observation based on the presented results is the fact that every dataset favours a different type of loss. Firstly, the XOR dataset shows a big gap in average rank between the regression-based (red and green) and classification-based (orange and blue) losses. Additionally, it is clear that the **one-to-one classification** loss manages to minimize the average rank of the important features the most. Secondly, the Orange Skin dataset shows the advantage of the **one-to-one** losses compared to **many-to-many** losses. We also find it impressive that both **one-to-one** losses achieved nearly minimal average rank for essential features when the number of features is 80. Finally, the Synthetic Regression dataset from Figure 6.8 shows no significant difference between the choice of a loss function. This dataset was, surprisingly, the hardest for the algorithm to succeed. We found that the algorithm relatively often overlooked one of the features, X_1, which contributes to the loss with the term $max(X_1, 0)$. To summarize, the average rank metric shows the discrepancies between **one-to-one** and **many-to-many** losses, favouring the former setting. Additionally, we found that the regression-based losses perform worse for some datasets than the classification-based losses. These observations confirmed our decision to use **one-to-one classification** loss in the future.

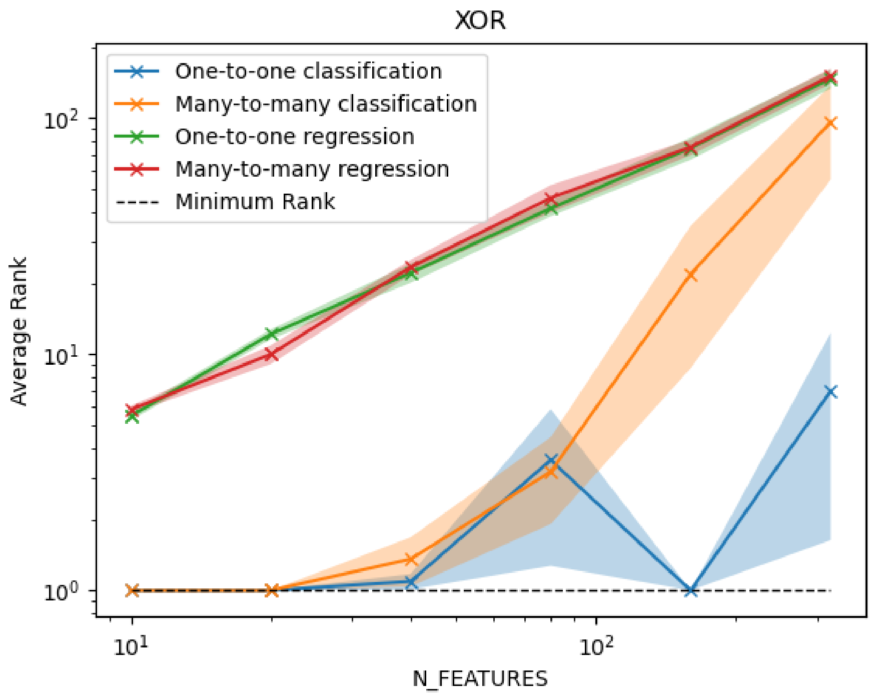


Figure 6.6: XOR dataset results. Y-axis corresponds to the average rank of important features, while the X-axis displays the number of features used for each analysis.

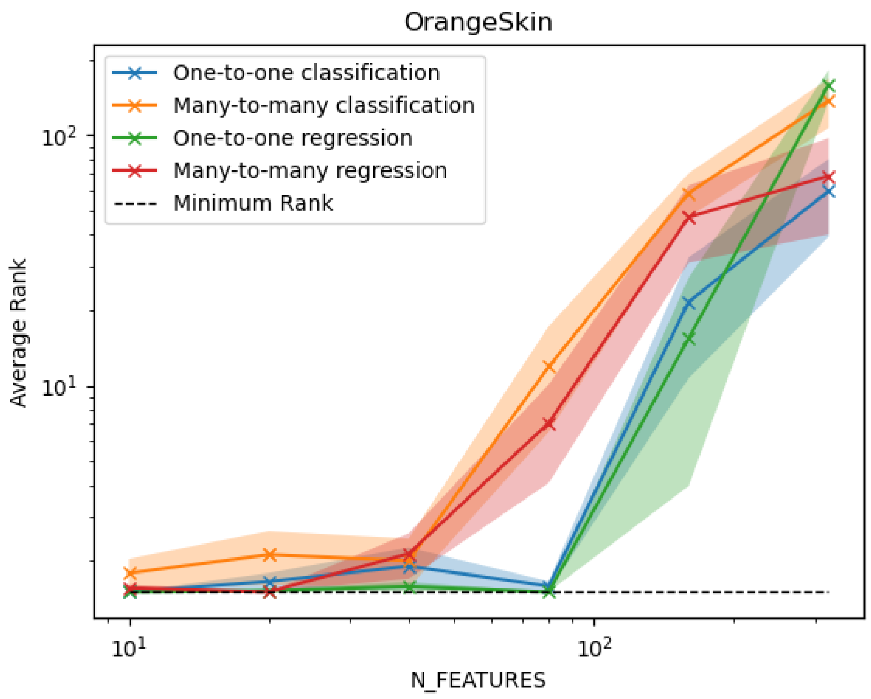


Figure 6.7: Orange Skin dataset results. Y-axis corresponds to the average rank of important features, while the X-axis displays the number of features used for each analysis.

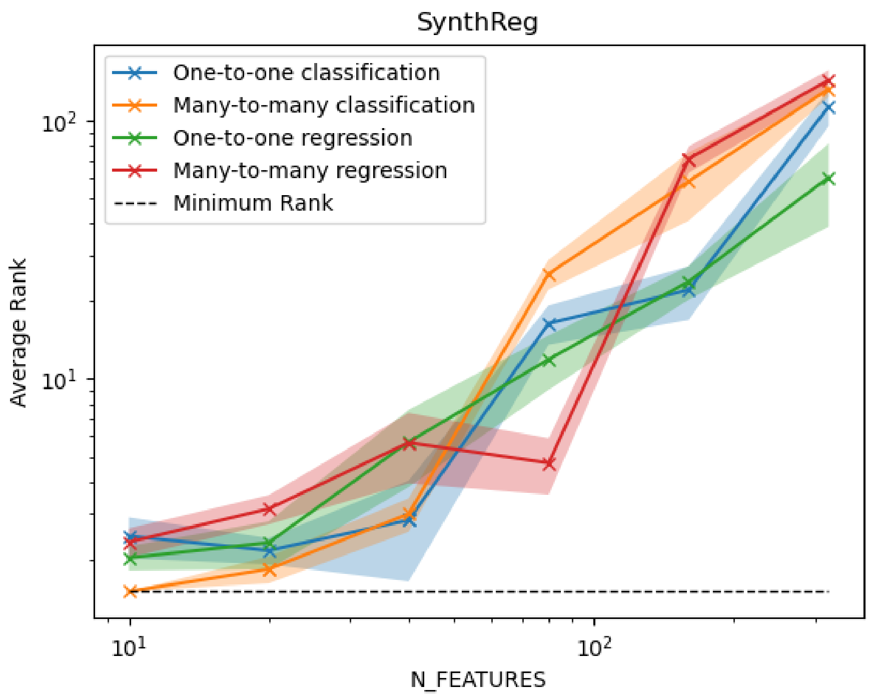


Figure 6.8: Synthetic Regression dataset results. Y-axis corresponds to the average rank of important features, while the X-axis displays the number of features used for each analysis.

6.3.2 Feature Importance Visualization

MNIST

We decided to visually inspect the MNIST datasets, similar to the one from Section 5.1.2. As a reminder, we use digits "3" and "8" from the MNIST dataset to create a supervised task of binary classification. We presented the results of the analysis, for one of the cross-validation folds, in Figure 6.9. The average size of the optimal mask was approximately two times bigger (179.2 ± 5.5) than the one set in Section 5.1.2 (85). The performance of the classifier was kept at the same level of $99.25\% \pm 0.19$. The results are visually consistent with the ones obtained in Section 5.1.2, even though they were created **without** setting up the optimal subset size.

All the images in Figure 6.9 show averaged feature importances with different information overlaid on top. While Figure 6.9.a shows raw importance, Figure 6.9.b displays only the importance of the features that were accepted into the optimal mask. Next, Figures 6.9.c and 6.9.d use the alpha channel to present the average digit "3" and "8" correspondingly. The final row again uses the alpha channel to present the difference in average digits: "8"-"3" (Figure 6.9.e) and "3"-"8" (Figure 6.9.f). We presented the results in this way to bring to attention that the optimal subset consists mainly of two groups of features: the ones that are usually present in "8" and NOT in "3" (Figure 6.9.e) and the ones that are usually present in "3" and NOT in "8" (Figure 6.9.f). The other features that are not in both of these groups have low importance scores on average.

YALE

The inherent difficulty of the YALE faces dataset lies in the scarcity of the number of training samples as well as the number of classes. The dataset (see more detail in Section 5.1.2) in a 5-fold validation setting has 98 training samples, 25 validation samples and 42 testing samples. The 15 subjects that are photographed correspond to different classes, which means that the validation set does not even have 2 images of each subject.

Even then, we were able to obtain satisfactory results through careful hyperparameter tuning, which resulted in $81.0 \pm 4.7\%$ accuracy. Finally, it is worth noting

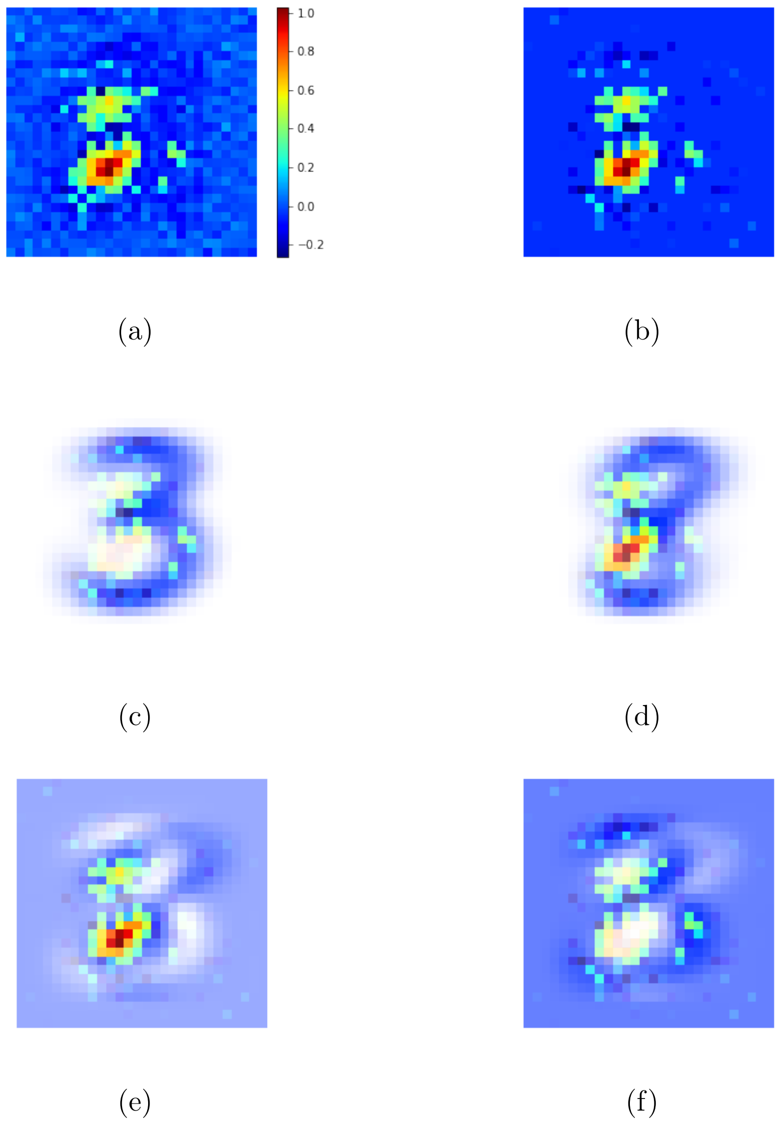


Figure 6.9: Feature importance map yielded by the Selector. (a) Raw feature importance. (b) Masked feature importance. (c) Masked feature importance superimposed on mean digit "3". (d) Masked feature importance superimposed on mean digit "8". (e) Masked feature importance superimposed on the difference between mean digits "8" and "3". (f) Same as (e) but with an inverted difference.

that the usual dense Operator architecture will fail to obtain good performance on this dataset, which pushed us to use the convolutional layers for this dataset, just like we did for the YALE dataset from Section 4.

Figure 6.10 shows the obtained feature importances calculated by the Selector. It is easy to notice from Figure 6.10.c that the majority of important features are located in the area around the nose of the subjects, while the detrimental features are centred around the neck of the subjects.

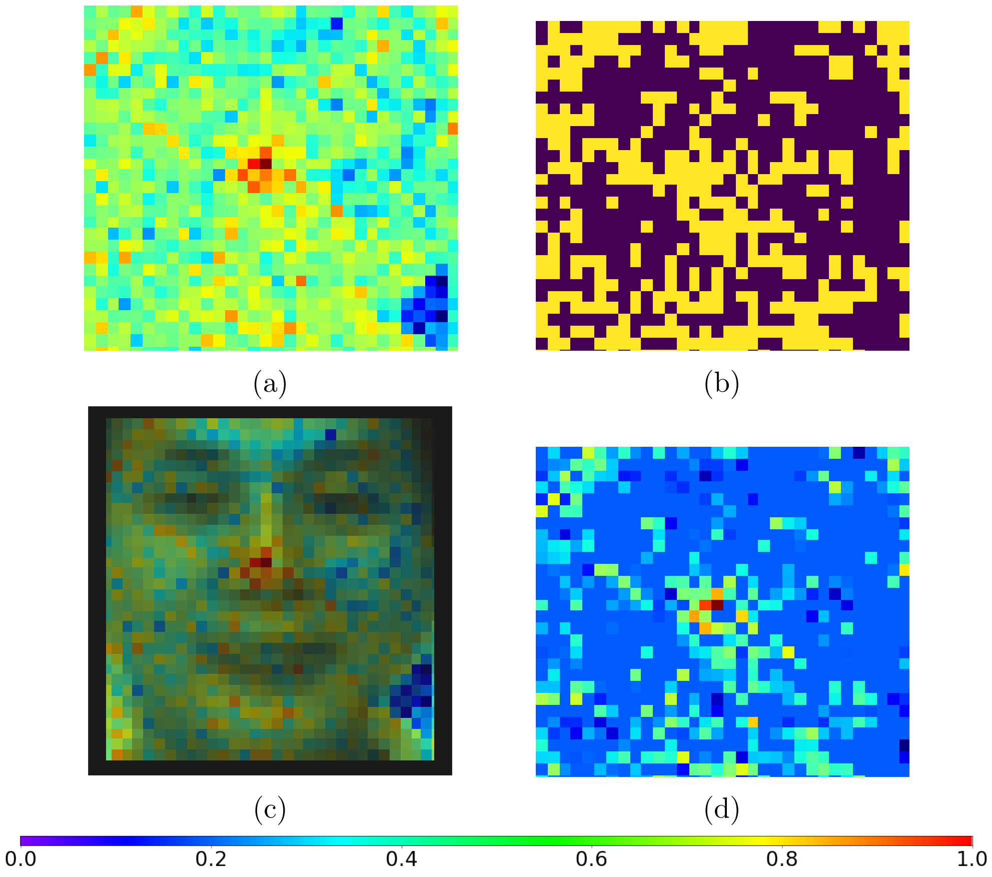


Figure 6.10: Feature importances for the YALE faces dataset. Subfigure (a) shows raw feature importances, (b) displays the optimal mask obtained during training, (c) uses the average face of one of the subjects (one class, chosen randomly) to better visualize the meaning behind the feature importances and (d) shows the feature importances only for the optimal subset.

6.3.3 Benchmark Datasets

Finally, we compare our method to other feature selection pipelines across 4 different datasets. It is important to stress that two of the datasets, YALE and TOX-171, are datasets where $d >> n$ with a relatively low number of training samples.

From Table VII we can see that the approach discussed in this section provides the best results across most of the datasets. While our approach from Section 4 shows a minor improvement on the Glass dataset, the result of 81% accuracy on YALE datasets shows a major advantage of the newest Selector/Operator dual architecture. The missing mRMR result for the TOX-171 dataset points to one of the advantages of this method: low scalability in high-dimensional datasets. In other words, the algorithm struggles to provide results for datasets with thousands

Table VII: Performance of different Feature Selection approaches measured on benchmark datasets. The displayed performance metric is Accuracy, measured in %.

Data Set	Glass	Vowel	TOX-171	YALE
Ours, Section 4	**92.0 ± 1.1**	98.9 ± 0.1	90.2 ± 5.6	66.9 ± 5.3
Ours, Section 6	91.5 ± 1.5	**99.0 ± 0.2**	**90.6 ± 2.7**	**81.0 ± 4.7**
CCM	84.4 ± 1.0	82.5 ± 0.5	90.1 ± 3.2	65.4 ± 4.8
RFE	86.6 ± 0.8	81.8 ± 0.4	84.1 ± 2.7	54.9 ± 5.9
mRMR	82.8 ± 2.3	79.8 ± 0.3	-	52.5 ± 2.8
DeepPink	91.9 ± 0.8	98.5 ± 0.1	90.9 ± 2.5	77.0 ± 3.2

Table VIII: Number of features selected by different Feature Selection approaches measured on benchmark datasets.

Data Set	Glass	Vowel	TOX-171	YALE
Ours, Section 4	**3**	8	83	30
Ours, Section 6	8	**2**	**90**	**38**
CCM	4	9	90	70
RFE	3	9	90	90
mRMR	2	9	-	50
DeepPink	3	8	88	34

of features.

For completeness, we provide a table summarizing the number of features chosen by each algorithm on Table VIII. The bolded values correspond to the best result from Table VII.

6.3.4 Experimental Details

Alternate Learning

To reach the best possible results, the Operator and the Selector must be trained alternately. As a reminder, the Selector is trained using the validation data results of the Operator. To maintain a balance between the networks while keeping the targets of the Selector relatively stable, we decided to divide the training data for each epoch into four chunks. After the Operator is trained on a chunk, the Selector is trained on the validation data. Although the number of chunks might seem arbitrary, it is connected to the 5-fold CV procedure. This approach means that training data is four times larger than validation, so throughout the whole training process, the number of training steps is the same for the Selector and the

Operator. From now on, we use the term "epoch" to mark the Operator training on one chunk of data and the Selector being trained on the validation data. Do note that this means that the data used during one "epoch" corresponds to $\frac{1}{4}$ of the whole training dataset or $\frac{1}{5}$ of the deployment data.

Hyperparameters

We have chosen the Adam optimizer for both the Selector and the Operator, with a lower Operator's learning rate of $lr_O = 1e - 4$ and the Selector's learning rate of $lr_S = 1e - 3$. We set the probability of perturbation $p_{exploration} = 0.1$, and the number of exploitation changes $n_\lambda = 2$.

Changing λ during training

We also decided to increase slowly λ throughout the training from 0 to $\lambda_{final} = 3$ over the period of $2N_{explore}$ epochs. This can be understood as steadily decreasing the temperature of an exploration-exploitation framework. In our case, low λ allows for more stable training that does not converge to a possible optimal mask too fast, thus not getting stuck in a local optimum.
The number of pretraining epochs is also set to $N_{explore}$, as both values need to be tuned to the number of samples of a particular dataset. The exact tuning of $N_{explore}$ is not necessary, as its only purpose is to ensure several hundred training steps for both the Operator (pretraining phase) and the Selector (increasing λ) for exploration purposes. We use the terms data batch size $|\mathcal{D}'|$ and mask batch size $|\mathcal{M}'|$ to mark the number of data points sampled from training/validation data and the number of unique masks used for the training step. For the Operator, that means that the number of actual samples it sees during one training step is given by $|\mathcal{D}'| * |\mathcal{M}'|$ as the Operator's batch is given by $\{(\boldsymbol{x} \otimes \boldsymbol{m}, \boldsymbol{y})\}_{\boldsymbol{m} \in \mathcal{M}', (\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{D}'}$. The Selector's actual batch size is given by $|\mathcal{M}'|$ or $|\mathcal{M}'| * (|\mathcal{M}'| - 1)$, depending on the **many-to-many** or **one-to-one** approach.

Architectures

We used ReLU activations in the hidden layers of both the Selector and the Operator with Softmax/Sigmoid/Linear activations for binary classification/multilabel and multiclass classification/regression tasks. We decided against using Dropout in the input layer as the masks already create sparse inputs, although it is possible to use it for hidden layers. For most of the tasks, we decided against using more complicated layers in order to limit the complexity of the solutions while showcasing that even vanilla architectures are able to outperform other, similar methods. For the output layer for the Selector, we use linear activation for **regression-based** loss and sigmoid for **classification-based** loss.

Early stoppping

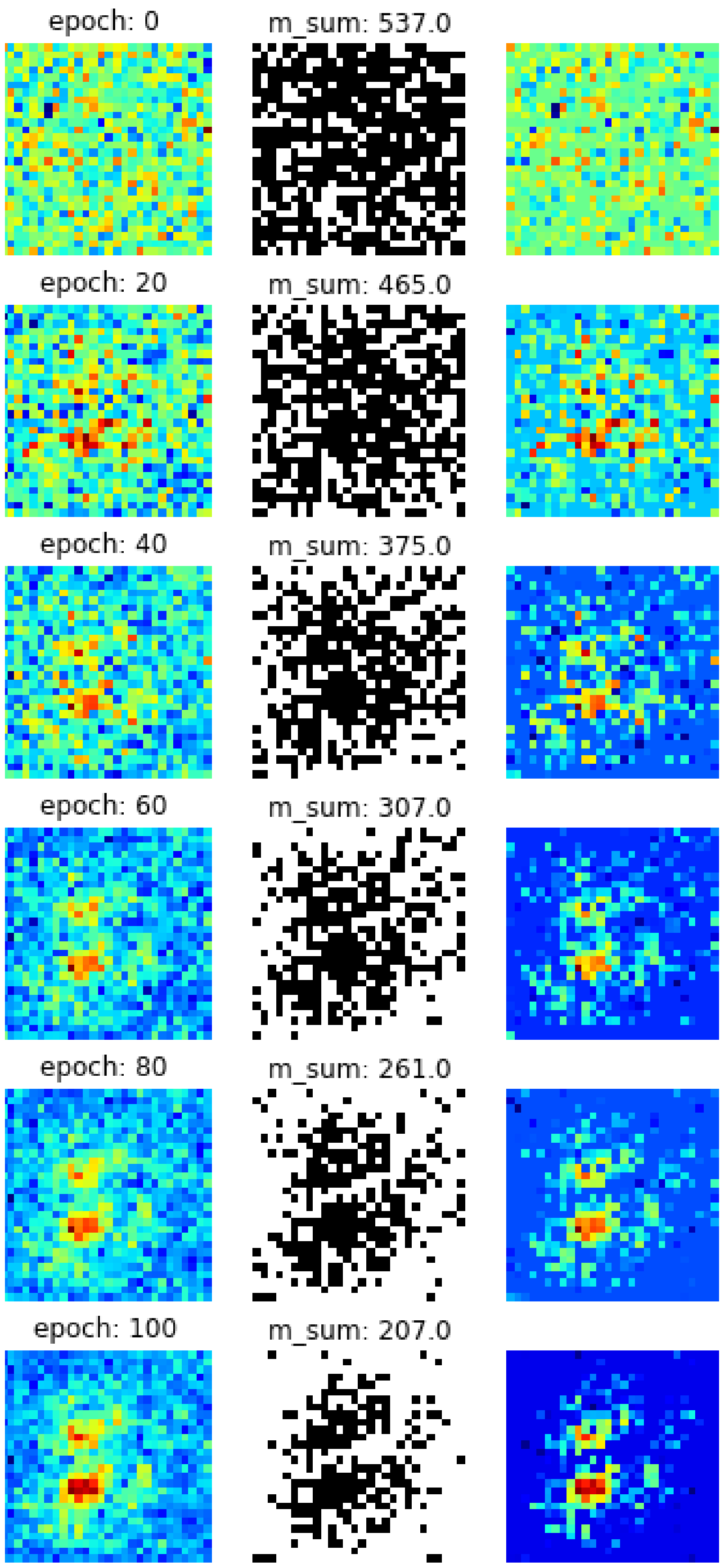
We use the early stopping procedure to maximize the Operator's performance. The monitored metric is the Operator's validation loss with a patience period of $N_{explore}$. After the stopping condition is reached, both the Operator and Selector weights are regressed to the time when they achieved the best performance. The optimal mask is also extracted from that time.

6.3.5 Training Behaviour

Selector's Training State

In this section, we will display the learned feature importances and optimal feature subsets for the Selector from Section 6 for datasets which allow visual examination. The state of the Selector (one of the optimal masks and the corresponding feature importances) was saved every N-th epoch, where N differs between datasets, due to their variable size. Finally, the epoch count does NOT correspond to the widely used epoch meaning: a period of time during which the model sees the whole dataset once. Instead, due to the particularities of our implementation, the epoch counter used by us corresponds to $\frac{1}{4}$ of the original epoch.

MNIST Figure 6.11 shows the evolution of feature importances during MNIST digit recognition training (for digits "3" and "8") throughout the whole training process which lasted around 230 epochs. We can see that the Selector updates its optimal mask and the FIE continuously during the whole training period. We also notice that the FIE map becomes more and more homogenous,


epoch: 0
m_sum: 537.0
epoch: 20
m_sum: 465.0
epoch: 40
m_sum: 375.0
epoch: 60
m_sum: 307.0
epoch: 80
m_sum: 261.0
epoch: 100
m_sum: 207.0

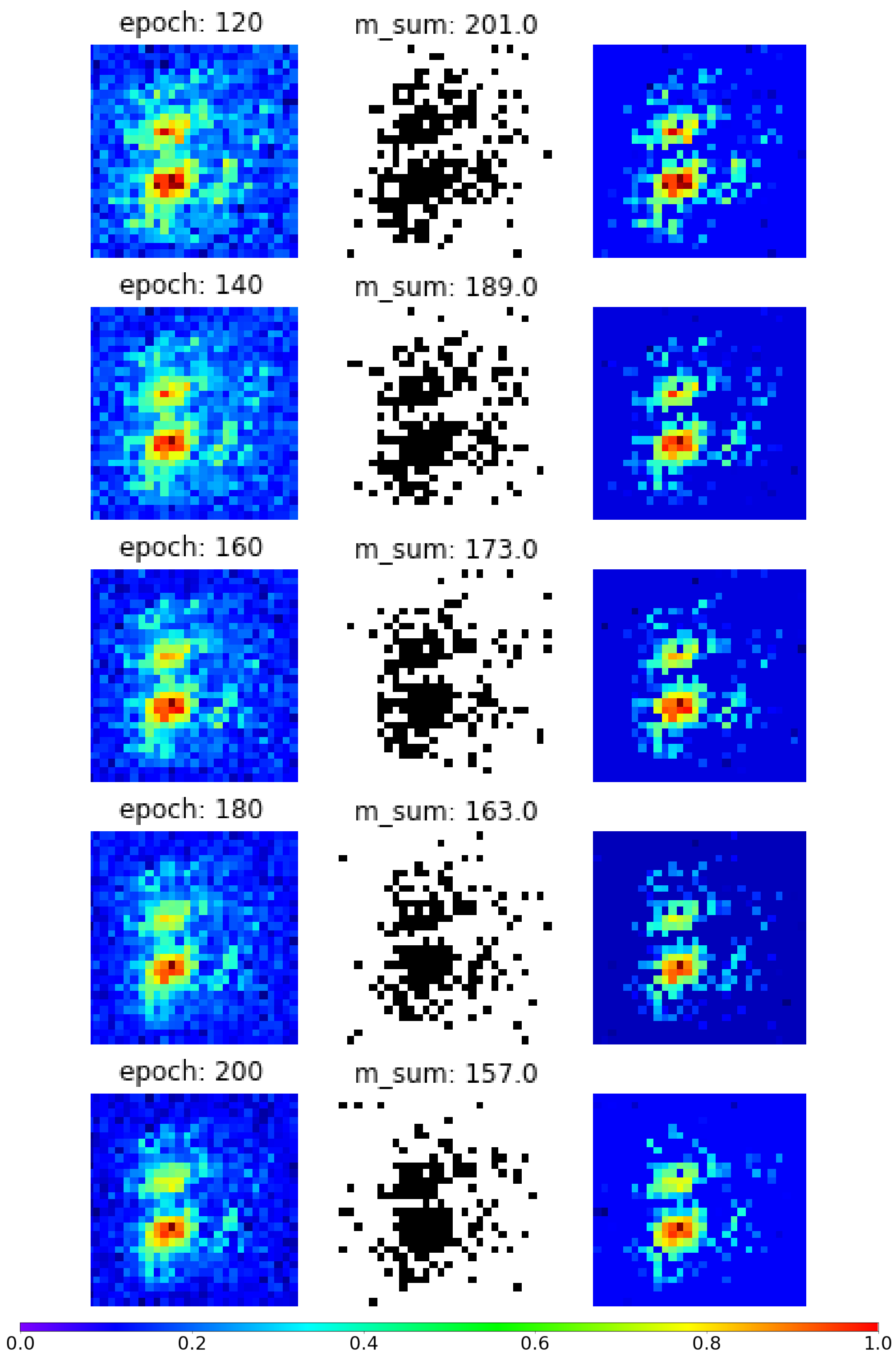
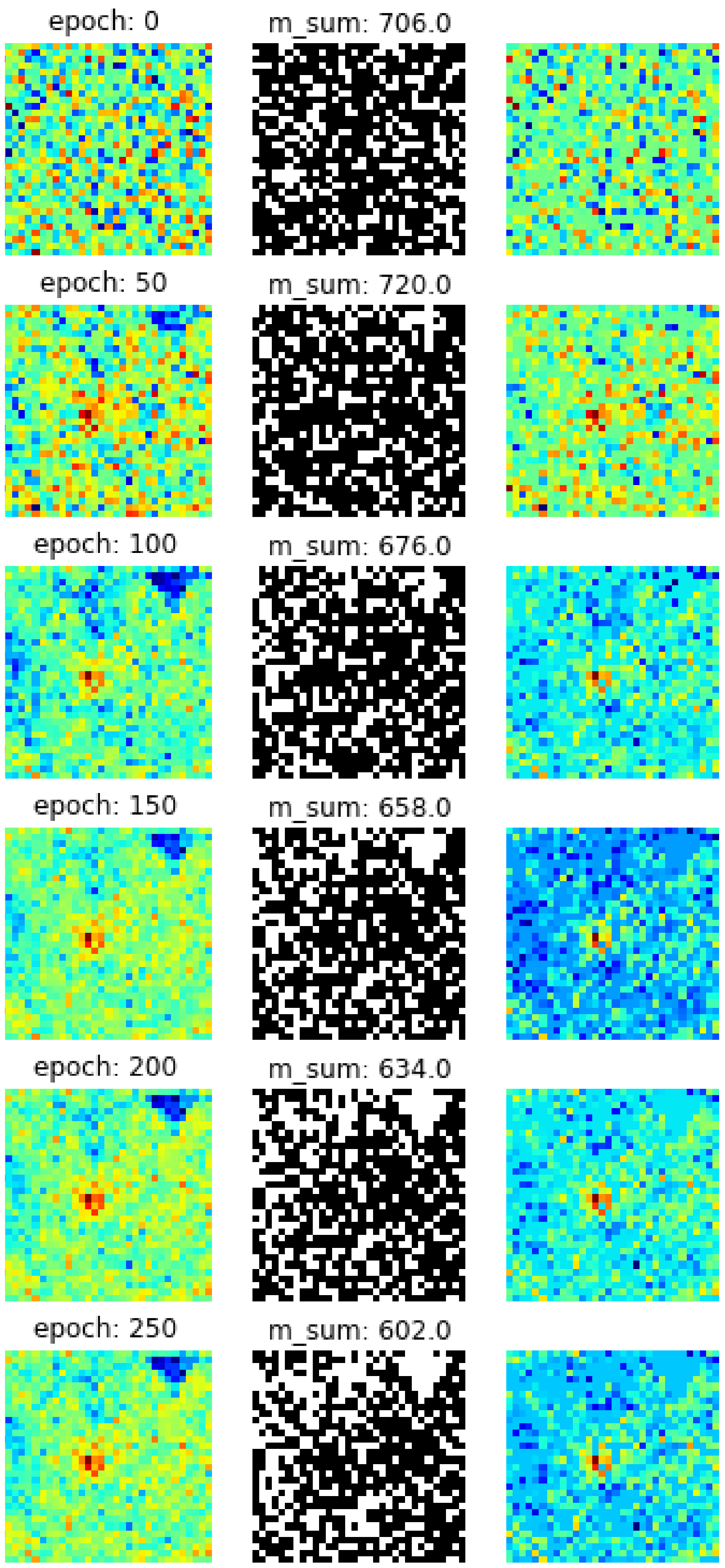


Figure 6.11: Evolution of feature importances for the MNIST digit dataset with digits "3" and "8". Each row corresponds to a different epoch. The left column shows the raw feature importances, the middle column displays the optimal mask (black for accepted features), and the right column presents only the feature importances for the optimal subset.

YALE Figure 6.12 shows the progress of the Selector for the YALE dataset. We show only the first 500 epochs as further progress shows only the reduction of the size of the optimal mask, where the final size is 412. We can see that after the first 150 epochs, the Selector converges to a rough estimate of future feature importances, and the rest of the epochs mainly reduce the size of the optimal mask and fine-tune the feature importances. Contrary to MNIST data, the additional epochs do not lead to more homogeneous results, even though the Operator continues to train and converges towards its final weights as well. The actual training took 1550 epochs, with the best Operator's validation performance recorded around the 1350-th epoch.


epoch: 0
m_sum: 706.0
epoch: 50
m_sum: 720.0
epoch: 100
m_sum: 676.0
epoch: 150
m_sum: 658.0
epoch: 200
m_sum: 634.0
epoch: 250
m_sum: 602.0

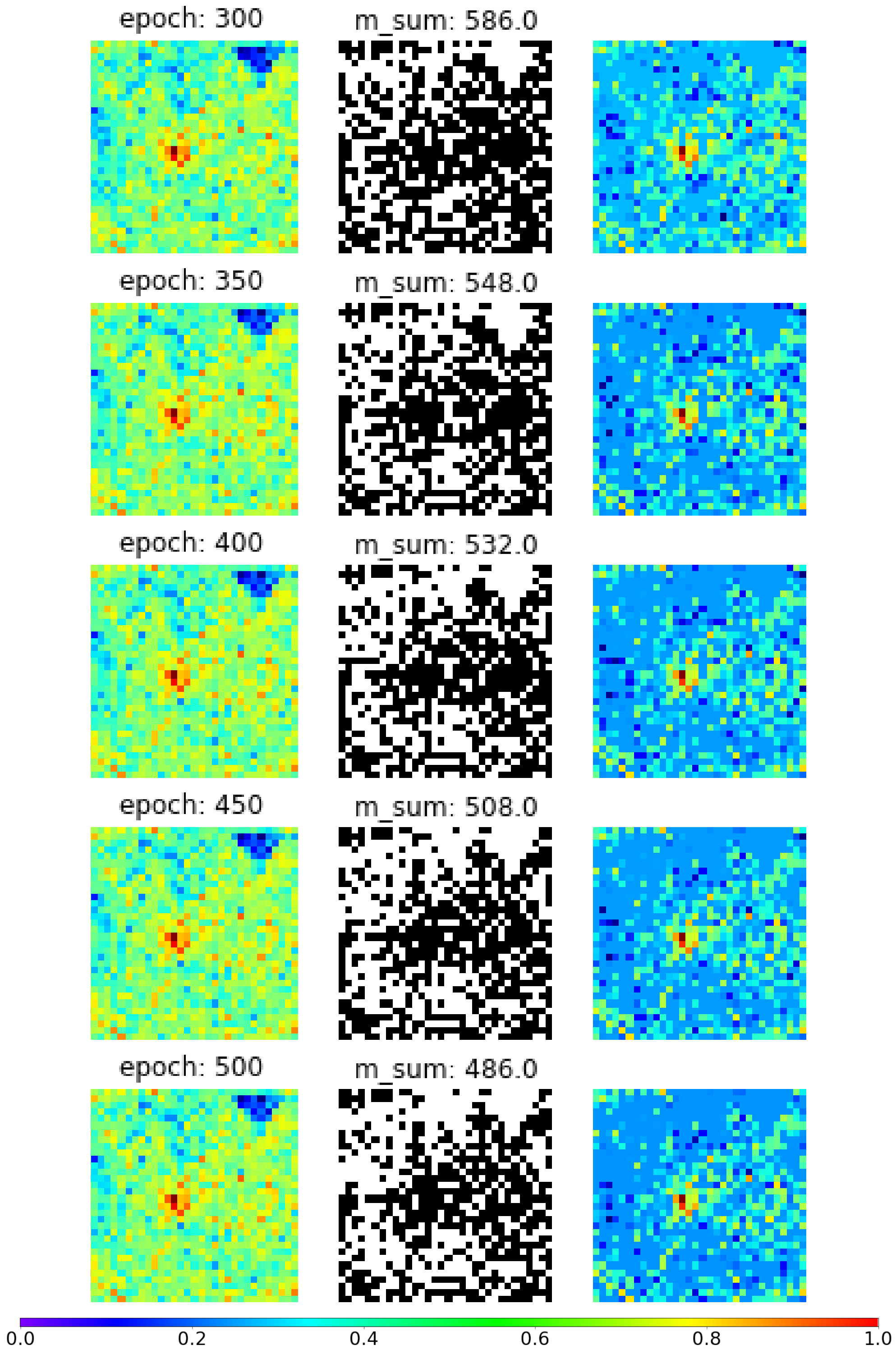


Figure 6.12: Feature importances for the YALE faces dataset. Each row corresponds to a different epoch. The left column shows the raw feature importances, the middle column displays the optimal mask (black for accepted features), and the right column presents only the feature importances for the optimal subset. The figures are rotated 90° clockwise relative to the ones from Section 6.3.2 as the original, raw dataset is presented in a rotated fashion as well.

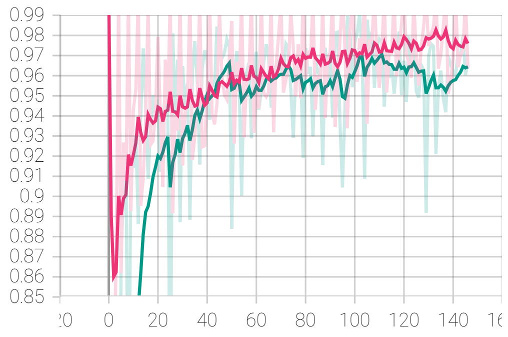
MNIST Training Curves

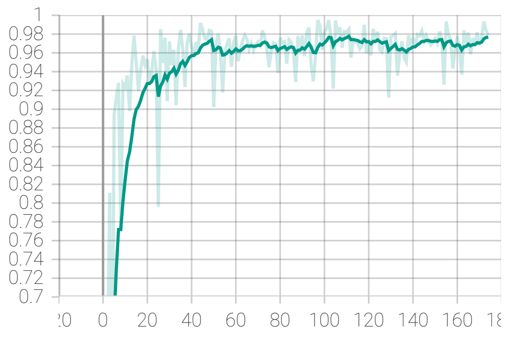
To visualize the performance of the Selector and the Operator, we decided to include the training curves for a randomly chosen fold of the MNIST digit recognition task. Figure 6.13 displays five plots that illustrate the progress of the Separator, Operator and the average optimal subset size.

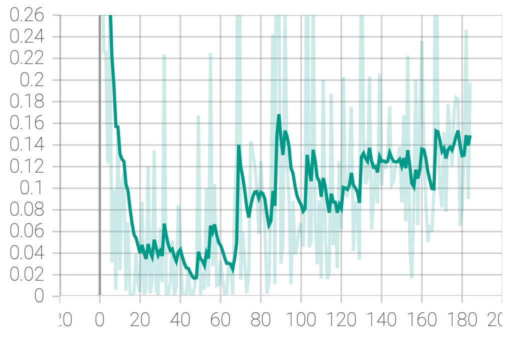
The first noticeable thing is the periodic nature of the magenta curve in subfigure (a). This corresponds to the Operator's training accuracy. The period of this trend is equal to four epochs, but we are not certain about the exact reason for this oscillation. We made sure that changing the optimal mask is not the underlying cause, as this happens every epoch and the lack of any granular changes is visible in subfigure (e). We hypothesise that this trend is caused by the division of the whole training dataset into 4 distinct parts and calling training on each of them an "epoch", as mentioned in 6.3.5.

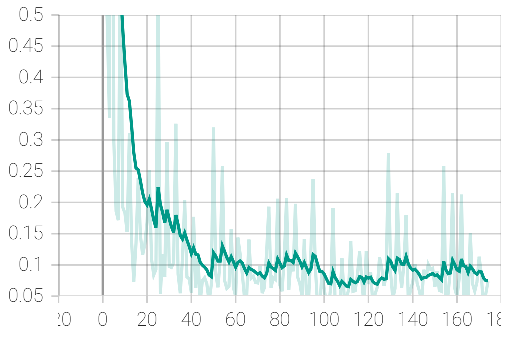
Additionally, both figures (a), (b) and (d) display the normal supervised training curve, where the training performance is higher than the validation performance and the loss/accuracy is steadily decreasing/increasing. When comparing the validation accuracies of subfigures (a) and (b), we can see that they are not identical. That is because the accuracy visible in subfigure (b) is calculated only for the optimal subsets Q_1, while the curve from (a) is calculated for the full mask batch, which also includes random masks. The fact that the curve from (b) achieves higher accuracy confirms that explanation.

Moving on to the Selector's progress, in subfigures (c) and (e) we can see the Selector's loss and the average subset size of the Q_1. The Selector's loss, after the initial drop, starts to increase with big variations. This upturn is caused by a feedback loop that, even though it seems detrimental, helps the Selector by providing masks that differ mainly in the "interesting" regions. The loop is based on the increasing performance of the Operator. It starts to "use" more and more of the available features to produce its predictions, leading to an increase in the complexity of the Selector's task. This, in turn, leads to new optimal subsets being proposed, which are usually smaller in size and contain more significant information, which again improves the Operator's performance. Subfigure (e) shows that even though the other curves vary from epoch to epoch, the size of the optimal subset is steadily decreasing and flattens out at the end of the training procedure.


(a) Operator's accuracy (train+val)


(b) Operator's accuracy for Q_1 (val)


(c) Selector's loss


(d) Operator's loss for Q_1 (val)

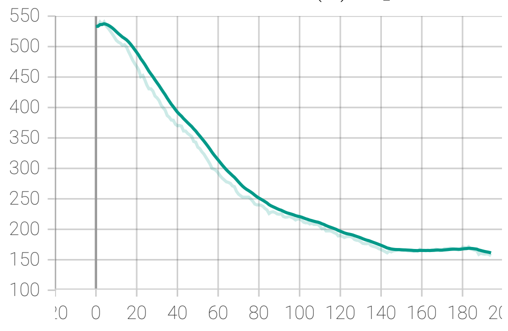

(e) Average subset size of Q_1

Figure 6.13: Training and validation curves for MNIST dataset. Magenta colour corresponds to training data while green corresponds to validation data.

6.3.6 Stability

Figure 6.14 illustrates the stability of our approach. We decided to show the differences in the Operator's performance, the progression of the optimal feature subsets and the differences in the optimal subsets obtained at the beginning of the training. We performed this analysis across the five folds of the cross-validation procedure.

From subfigures (a) and (b) we can see that there are no major differences in the

Operator's loss across the runs and that the progression of the average subset size is not dependent on starting conditions or lucky seed. Finally, we added all the optimal masks calculated across the five runs and divided them by their amount, creating an average optimal subset in subfigure (c). Thus, the features with the corresponding value of 1.0 are present in **every** calculated optimal subset while the ones with 0.0 are never present in any of the calculated subsets. We can see that the optimal masks consist of two clusters of features that are always present and can be deemed the most important (there is also a single pixel, located to the right of the bottom cluster, that is always present in the optimal subsets). We are convinced that the differences between the optimal subsets are caused by the redundancy between the various pixel regions.

We also found that even though the importance scores were not directly calculated for subfigure (c), it is similar to the feature importance estimates obtained in Figure 6.11. We conjecture that the frequency with which each feature is present in the optimal subset can be used as an impromptu feature importance score.

6.4 Extensions

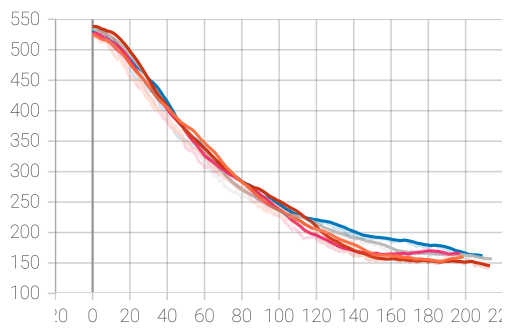
6.4.1 Group Feature Importance Rankings

In this subsection, we will focus on a mechanism allowing our algorithm to perform group feature selection, as well as presenting results for one of the group feature selection benchmark datasets [77].

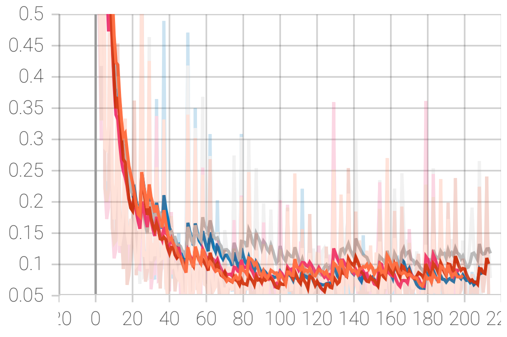
Method

Adding group-based data comprehension to our method is a straightforward process. The only thing that was changed was the masking process, which used to be a simple multiplication (application of binary mask) $f_{mask}(\boldsymbol{x}, \boldsymbol{m}) = \boldsymbol{x} \cdot \boldsymbol{m}$. The dimensionality of both $\boldsymbol{m}$ and $\boldsymbol{x}$ was also equal to d. We did not use the masking function notation (f_m) not to introduce unnecessary complexity.

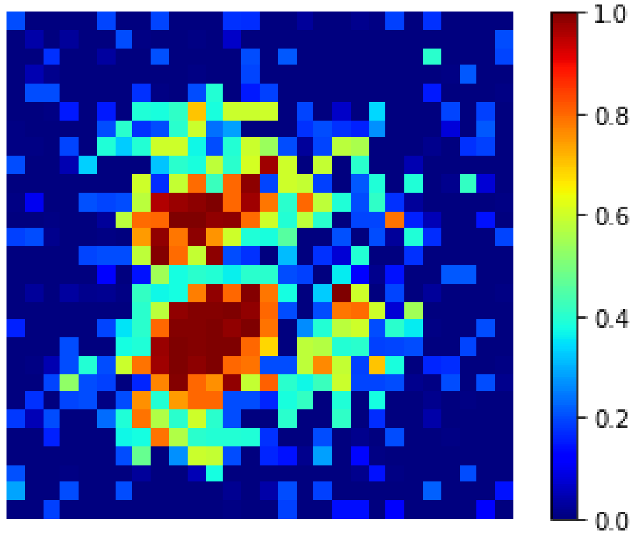
For group feature rankings, we can understand each group G_k as a subset of features characterised by a mask $\boldsymbol{g}_k$ which is a binary vector of length d. Given a set of d_G groups $\mathcal{G} = \{G_1, G_2, G_3, ..., G_{d_G}\}$ (see Section 3.7) for a dataset with d



(a) Average subset size of Q_1



(b) Operator's loss for Q_1 (val)



(c) Average optimal subset

Figure 6.14: Stability comparison for MNIST 5-fold CV. For that purpose, different curve colours in subfigures (a) and (b) correspond to different folds.

features we can define the group masking function:

$$f^G_{mask}(\boldsymbol{x}, \boldsymbol{m}, \mathcal{G}) = \boldsymbol{x} \cdot \dot{(}m_0\boldsymbol{g}_0 + m_1\boldsymbol{g}_1 + m_2\boldsymbol{g}_2 + ... + m_{d_G}\boldsymbol{g}_{d_G}) \tag{6.24}$$

where the dimensionality of $\boldsymbol{m}$ is d_G. This approach allows us to use the algorithm without any other changes, as the Selector now works on groups rather than singular features.

To clarify, one can think of $\boldsymbol{g}_k$ as pre-defined (appropriately for each task) binary vectors that group features together. Added together (with the OR operator), they should add up to the full vector of ones, so that each feature is accounted for in one of the groups. Then, the mask $\boldsymbol{m}$, instead of affecting each feature on its own, affects groups of features all at once - so the features in one group are all either masked or not.

Results

We decided to use the yeast cell cycle dataset[1], for which the number of groups $d_G = 20$ consisting of $d = 106$ features and $n = 542$ samples. The groups were decided following the procedure first used by [78] and then followed also by [77], and is based on first clustering the features using K-means and optimizing the Gap statistic [79].

The dataset [80] [81] is used to benchmark the identification of transcription factors (TFs) which control the rate of DNA to mRNA transcription. The target variable is the mRNA level of the genes taken at 28 min during a cell cycle, and the input variables are the measured binding information from the TFs. It has been confirmed [82] that 21 of the TFs are what truly drives the target variable. The other group feature selection methods we compare are:

- Deep-gKnock [77], is a method similar to deep knockoffs that integrate the power of DL and knockoff technique in a group feature selection setting.
- DeepPink is another knockoff-based technique that uses DL to produce feature importances.
- group-SLOPE [83] is a method that allows for the adaptive selection of variables in a high-dimensional setting through convex optimization.

Table IX: Performance of different Group Feature Selection approaches measured on yeast cell cycle dataset.

Method	# features chosen	# important features chosen
Ours	28	**14**
DeepPink	5	3
group-SLOPE	40	13
Deep-gKnock	21	11

All the methods mentioned above focus on approximating the FDR (false discovery rate), or more precisely, gFDR (group false discovery rate). We find that our method manages to find most of the important features while keeping the number of chosen features low.

[1]The dataset is available at SPLS library CRAN repository: `https://search.r-project.org/CRAN/refmans/spls/html/00Index.html` and was extracted using `pyreadr` package from raw `.rda` files.

6.4.2 Feature Subset Encodings for Feature Selection and Explainability

Motivation

In the course of our work we grew curious about possible visualizations of the feature space. The goal was to understand the interactions between different subsets. Following that objective, we created a novel framework that allows the user to gain insights into the dataset by approximating the similarity between different feature subsets.

Method

The created space uses the similarity of results to determine the relative Euclidean distances between embeddings of different feature subsets. We found the best results with Jensen-Shannon divergence (JSD) as the similarity measure, which started to become more popular in the ML community due to the well-known Generative Adversarial Network architecture [84]. To be more precise, we used the square root of the Jensen-Shannon divergence, called Jensen-Shannon distance [85], which for distributions X, Y is given by:

$$D_{JSD}(X,Y) = \sqrt{\mathrm{JSD}(X||Y)} = \sqrt{\frac{\mathrm{KL}(X||Z) + \mathrm{KL}(Y||Z)}{2}} \tag{6.25}$$

where Z is the mixture $Z = \frac{X+Y}{2}$ and KL is the Kullback-Leibler divergence. One of the reasons behind using the aforementioned distance is the fact that it is symmetric, strictly positive and zero for equal distributions. Contrary to JSD, it also obeys triangle inequality. Additionally, it is bounded in the 0-1 range, which allows easy comparison between different datasets with different numbers of features.

To obtain the embedding predictions, we added an additional output head $f_S^{emb}(\boldsymbol{m}, \phi^{emb})$ to the Selector that outputs n_{emb}-dimensional embeddings. The loss associated with this output head is minimizing the differences in relative distances between the feature subsets:

$$\mathcal{L}(\mathcal{X}_{val}; \mathcal{M}'; \theta; \phi^{emb}) = \sum_{\boldsymbol{m}_i \in \mathcal{M}} \sum_{\boldsymbol{m}_j \in \mathcal{M}} D_{EUC}(f_S^{emb}(\phi^{emb}, \boldsymbol{m}_i), f_S^{emb}(\phi^{emb}, \boldsymbol{m}_j)) - D_{JSD}(f_O(\theta, \mathcal{X}_{val}, \boldsymbol{m}_i), f_O(\theta, \mathcal{X}_{val}, \boldsymbol{m}_j)), \tag{6.26}$$

where we used the simplified notation of $f_O(\theta, \mathcal{X}_{val}, \boldsymbol{m}_j)$ to mark the vector of results from f_O measured on the dataset $\mathcal{X}_{val}$ and D_{EUC} marks the Euclidean distance.

Another option was to use a dimensionality reduction algorithm like TSNE [86] to analyze the distances between the final prediction results $f_O(\theta, \mathcal{X}_{val}, \boldsymbol{m}_j)$.

Results

We decided to visualize the results on the XOR dataset as it presents the biggest problem for many feature selection algorithms due to its synergistic nature. As a reminder, the XOR dataset consists of 10 features where the first four (with indices $1, 2, 3, 4$) are not null and decide the ground-truth class of each sample. We set $n_{emb} = 3$ in order to be able to visualize the feature subset embeddings. After training the Selector using our pipeline from Section 6, we predicted all embeddings for the feature powerset (1024 feature subsets). The results are shown in Figure 6.15 as a scatter plot, where the marker colour corresponds to different configurations of the not-null features.

At first glance, it is clear that the feature subsets are clustered in certain regions of the embedding space, where each cluster can be characterized by one colour (label). This points towards a high degree of similarity between the feature subsets in each cluster, which is confirmed by the method with which the dataset was synthesised. The only difference between the points with the same colour is a different configuration of null features. The second observation can be made that the cluster with no important features (label "None") is directly opposite to the cluster with all the important features (label "1,2,3,4"). Between the two extremes, all the other clusters are ordered so that the clusters with fewer useful features are closer to the "None" cluster.

6.4.3 Alternative Masking Function

A final addition to our algorithm allows it to handle sparse data. Previously, that was one of the weaknesses of both of the Operators: if trained on sparse data, like bag-of-words encodings of NLP datasets, the masking process would result in even higher sparsity and would prevent the Operator from finding usable gradients.

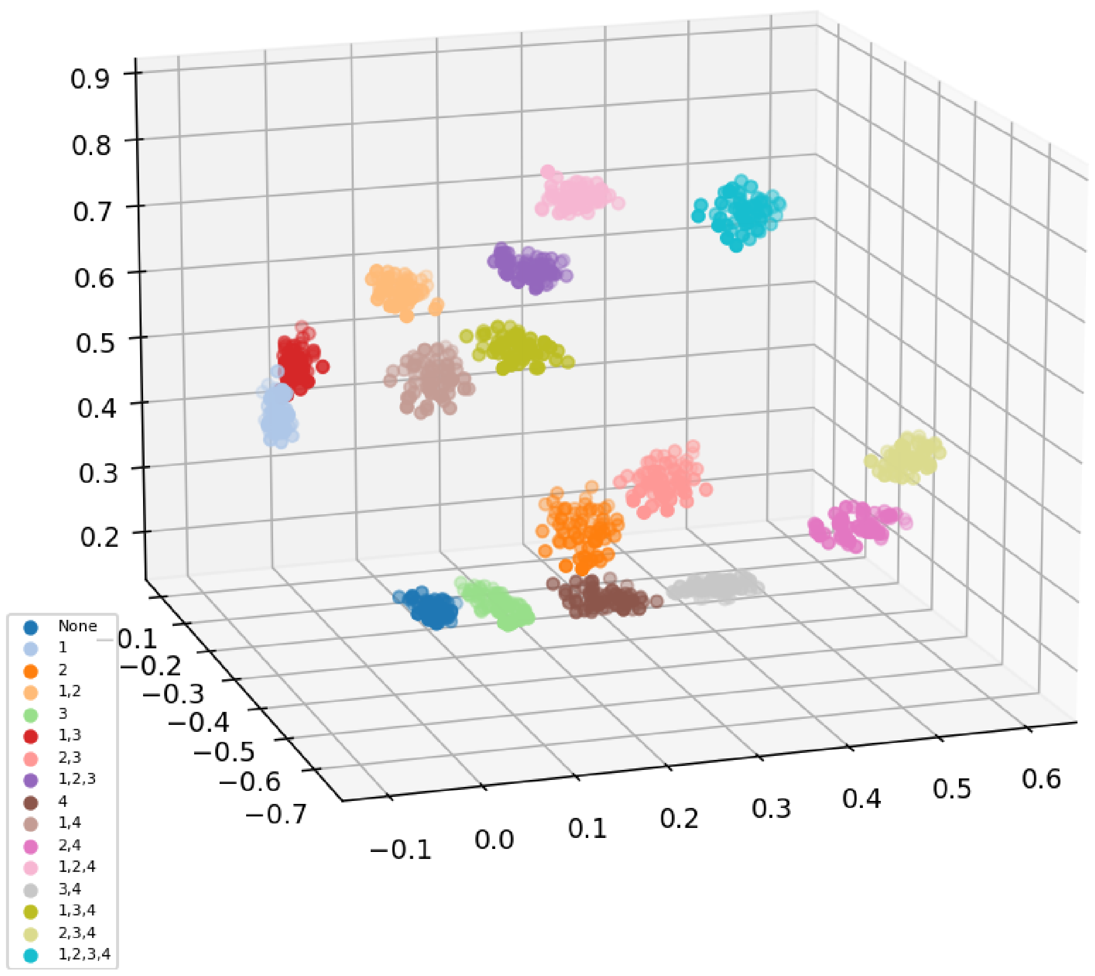


Figure 6.15: 3D visualization of the powerset of feature subset embeddings for the XOR dataset.

To counteract this problem, we found another masking function f_{mask}^{sparse} that does not increase the data sparsity. Instead of simply multiplying $\boldsymbol{x} \cdot \boldsymbol{m}$, we shuffle the unused features in random order.

We found that this approach hinders the Operator from performing well on non-sparse datasets but allows the method to be run on sparse datasets with mixed results.

6.5 Discussion

The first discovery made was regarding the optimal loss function. Before the experiment, we hypothesised that the **many-to-many classification** would be the best-performing option. We hypothesised that the **classification**-like structure would allow for less extreme Selector's targets. Additionally, the **many-to-many** nature would result in a several-fold increase in training samples provided to the Selector, leading to increased training performance. On the contrary, we discovered that while our conjecture seemed true in the case of **classification** vs **regression** comparison, the **many-to-many** approach proved to be worse than the **one-to-one**.

We hypothesise that the reason behind this result is the prevalence of mask-to-mask comparisons for the **many-to-many** method, where the masks differ by a larger degree. As a reminder, the factor that decides how the first half of the mask batch differs from the second half depends on $p_{explore}$. For the **many-to-many** method, the difference between some of the mask comparisons can be greater. The big differences between paired masks might result in the signal about precisely which feature resulted in different performances between masks being lost. This is counterintuitive to our expectations, but we still decided to use **one-to-one classification** loss according to our motivation of using performance-driven feature importance estimation.

Secondly, during our visual examination experiments (MNIST and YALE datasets), we found that our method produces meaningful feature importance estimates. We verified that the MNIST "3" vs "8" classification task feature importance estimations are consistent with the ones obtained in Section 5.1.2. We noticed that the majority of the features present in the optimal subset (especially those with the highest feature importance) clearly show up on the plots showing the difference between the average digit "3" and "8" (and the opposite). Additionally, for the YALE faces dataset, we noticed that the most important features are centred around the nose area of the subjects, thus proving the consistency and real-world applications of our approach. On the other hand, the noisy background of the YALE results shows the difficulties in obtaining valid results for such an unbalanced dataset (1024 features and only 123 training samples spread over 15 classes).

Next, for benchmark datasets, we notice that our method consistently outperforms other approaches when it comes to testing accuracy. We were also surprised with the achieved performance on the YALE dataset, which is usually highly unstable due to the small number of samples.

Then we analysed how the feature importance estimations, the corresponding optimal subsets, and the models' performances change across the training procedure. We noticed that, depending on the size of the dataset, the optimal mask is formed (similar to the final one) at different relative times. For larger datasets, this process takes more relative time (the optimal mask is still changing for the late epochs). We also checked and verified the performance curves' validity and showed the convergence of the optimal feature subset size to its final value.

We also analysed the stability of our framework. This translated into comparing the differences in performance curves, the progression of the optimal subset and the final optimal subset. We showed that our algorithm produces stable and reliable results across runs and that the chosen features for MNIST dataset consist of two main clusters of the features that are present in each optimal subset. Additionally, the frequency of the presence of each feature in the optimal mask can be treated as an impromptu feature importance estimate.

On the other hand, we found some interesting artefacts in the training curves that show up during training on all datasets, which are especially visible in Figure 6.13. Finally, we wanted to also bring to light one other issue that is not clearly visible from the results presented so far. That is the fact that the difference between validation and test performances is much larger than usual (for vanilla networks). That is caused by the fact that the validation data is used to fine-tune the optimal subset, which results in overfitting the validation data. That is usually not the case when tuning the hyperparameters as the number of choices is relatively small, but for the case of optimal subset, the number of possible subsets scales as 2^d. This indicates one of the problems of our method is the validation data overfitting.

Chapter 7

Conclusions

While powerful, deep learning requires the users' experience to employ its capabilities fully. Especially in light of recent developments in the space of generative networks (Stable Diffusion [87], Giga GAN [88] or ChatGPT-4.0 [89]), the interest in neural networks is peaking. These two factors require providing the new user base with a toolkit that can handle the increasing complexity of DL solutions while remaining approachable and flexible towards different use cases. The benefit of such a tool would be to provide much-needed clarity to neural networks, increasing their interpretability and explainability. These two properties would allow the users to gain confidence in the produced predictions and possibly extend the applications towards areas where the explanations are legally required. Additionally, it is possible to extract additional knowledge about the data by using explanations and interpretable methods.

First, we introduced the topic in a comprehensive way and presented a review of the most common methods used in the area. Then, we stated the problem of defining what feature importance is and proposed a definition that was the defining principle behind the proposed solutions. We discussed the methods that were developed in the scope of this work, compared their performance to other commonly used methods and discussed the results. Below, we will summarize the contents of this work and our contributions, as well as discuss the possible future avenues of research.

7.1 Thesis Summary

Recently, many approaches have been developed that try to provide explainable or interpretable AI. The goal is to provide meaningful, precise and clear explanations of networks' predictions and gain a deeper understanding of DL's inner workings.

In Chapter 2, we briefly explained the differences between interpretability and explainability, categorized typical motivations for XAI and provided different types of feature importances that can be extracted from a model. We presented the most common and cutting-edge feature selection methods.

Chapter 3 starts with providing definitions of an optimal feature subset of constant and variable sizes as well as our take on feature importance, which is a highly debatable topic. By giving a new feature importance definition, we accomplish one of the research questions specified in Section 1.4. While the provided definition is performance-based and provides a clear view of the performance of each feature, its complexity in the sense of dependency on the current subset makes it harder to use out-of-the-box and understand for first-time users.
Then, we provide a basis for why we chose to use validation data for our feature importance measurements and how it allows us to counteract the impact of overfitting. We show the impact of regularizing feature subsets by explaining how it affects the optimal subset and helps with issues of finite sample sizes. We provide a summary of other problems with feature importance measurements and the validity of the provided information. Finally, we focus on possible extensions of feature selection like group feature selection or feature subset encodings.

We present our first framework that provides feature importance estimates in Chapter 4. It is a dual-net system of the Operator and Selector that are trained jointly in an alternate manner. Their synergistic relationship is meant to not only provide feature importance but also to improve the performance of the Operator. In broad terms, the Operator is meant to accomplish the original, supervised task dictated by the dataset and the user. At the same time, the Selector uses the

Operator's response to different feature subsets to find the optimal subset. We analyse its performance on synthetic datasets, perform a visual check of the produced feature importances on the pictorial datasets and compare the Operator's performance to other feature importance methods on benchmark datasets. The main disadvantage of such an approach is the necessity of the hyperparameters that dictate the required size of the optimal feature subset, as well as the relative required power of such a framework. We found it a powerful tool, but the number of tunable hyperparameters makes it much harder for wider use.

Next, in Chapter 6, we identify the disadvantages of our approach from the previous Chapter and provide clear goals that the next approach must have. The crucial difference between the two approaches is the removal of the constant feature subset size constraint and a major simplification of the algorithm. We show the motivation behind the algorithm and explain how the goals stated above were obtained. During the research process, we found several prospective loss functions that can be defined for the Selector Net. We test them on the same synthetic datasets as in Chapter 4 but with a twist of geometric increase in the number of null features. In the process, we compare both the Operator's performance and the validity of found feature subsets. We found that the best-performing loss function was a **one-to-one classification** loss function, and we chose this approach for future experiments. Then, we produce visual representations of feature importances on the MNIST and YALE faces datasets, both pictorial datasets. We decide that the resulting feature importance maps provide a clear and concise representation of the features used by the Operator. We also investigate how the feature importances are produced throughout the training. Finally, we compare the performances of this novel approach to our previous dual-net architecture and other methods, which shows a distinct advantage of our framework.

We found that the developed method satisfies the two main points of interest that we raised when starting our research: providing a meaningful, performance-based feature selection method that allows for variable-size feature subsets that also provides FIE. It is simple to launch and synergizes with the model deployed for the original task, making it achieve greater performance. On the other hand, it requires more computing power, as well as more GPU VRAM to run two DL models in parallel.

During the research process, one often finds fascinating applications or extensions of their work to neighbouring domains. After the work outlined above, we suggest possible extensions of our framework: Feature Subset Encodings and Group Feature Selection. We briefly discuss the motivation for both extensions and present experiments we performed to test them. We plotted the resulting feature subset encodings and explained the provided insights, which support the underlying structure of the synthetic XOR dataset. Then, we use one of the Group Feature Benchmarks from the literature to measure the relative performance of our methods. The new method of feature subset visualization is certainly powerful and provides a novel way to understand the data. On the other hand, the possible number of data points that can be visible (powerset of all the features) can be hard to plot and understand for datasets with a large number of features.

In general, we find that our work of pursuing XAI through feature importance estimation was a good course to study. After all, it is a quantity that can be measured for every model. Certainly, many issues remain, like the exact definition of feature importance, its dependence on synergistic/redundant variables or bridging the gap between what is truly important to the data and the model. Some of these issues can be addressed in future work, like getting help from domain experts for more interpretability-focused research.

7.2 Future Work

While work provides possible solutions for both interpretable and explainable AI, some research avenues remain open in the context of broader explainability.

1. **Local Feature Importance (Explanations)**
   It is possible to use a similar framework for local feature importance measurements. This technique would result in performance-driven explanations of singular data points. We found initial success with preliminary experiments, but this is out of the scope of this work.

2. **Unsupervised and Reinforcement Learning**

This thesis focused on the domain of supervised learning. The counterparts to this approach are unsupervised and reinforcement learning. One of the major hurdles would be modifying the underlying technology to work with more exotic loss functions (unsupervised) or with sparse rewards (reinforcement learning).

3. **Visual Explanations**

   While our work with feature subset embeddings or producing feature importance maps for pictorial datasets provides visual cues about the inner workings of the Operator, a similar understanding would be harder to reach for data with multiple not-null features or data without clear pictorial representation. Creating a framework that would provide the user with a visual representation of the inner working of a neural network would be a huge step forward for XAI.

4. **Datasets**

   We want to broaden the spectrum of the datasets that we test our method on. That would correspond to other domains, like NLP or sensor data.

5. **Domain Experts**

   We want to investigate the interpretability aspect of our work in more detail by discussing the results of our method for different datasets with their respective experts. That would usually correspond to gene expression or physical sensor data.

7.3 Limitations

In this section, we want to briefly summarize the limitations that were already mentioned in the previous sections. We will refer to the final version of our algorithm from Section 6.

1. **Processing Power** - our method still requires more processing power than other, less powerful solutions, like LASSO. On the other hand, this processing power is used to understand the FIE in a more complex manner, which results in different requirements and expectations for each method.

2. **GPU Memory** - due to using two models, the memory requirements are doubled. Currently, in the DL field, memory is the resource to maximize, allowing for faster training and better performance. That is counterproductive with the use of our method. Contrary to this, we think that the increased performance and additional insights into the data at least partially compensate for this downside.

3. **Flexibility** - while already investigated, our method is not ideal for sparse datasets, where the masking mechanism makes the data even more sparse.

Bibliography

[1] Wojciech Samek, Thomas Wiegand, and Klaus-Robert Müller. Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models. *arXiv preprint arXiv:1708.08296*, 2017.

[2] Isabelle Guyon, Jason Weston, and Stephen Barnhill. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46(2):389–422, 2002.

[3] Jundong Li, Kewei Cheng, Suhang Wang, Fred Morstatter, Robert P. Trevino, Jiliang Tang, and Huan Liu. Feature selection. *ACM Computing Surveys*, 50(6):1–45, Jan 2018.

[4] Isabelle Guyon and Andre Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3(3):1157–1182, 2003.

[5] Jianbo Chen, Mitchell Stern, Martin Wainwright, and Michael Jordan. Kernel feature selection via conditional covariance minimization. In *Advances in neural information processing systems*, 2017.

[6] Hanchuan Peng, Fuhui Long, and Chris Ding. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8):1226–1238, 2005.

[7] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

[8] L Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[9] Jundong Li, Kewei Cheng, Suhang Wang, Fred Morstatter, Robert P Trevino, Jiliang Tang, and Huan Liu. Feature selection: A data perspective. *ACM Computing Surveys*, 50(6):94, 2018.

[10] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. Smoothgrad: removing noise by adding noise. *arXiv preprint arXiv:1706.03825*, 2017.

[11] Avanti Shrikumar, Peyton Greenside, Anna Shcherbina, and Anshul Kundaje. Not just a black box: Learning important features through propagating activation differences. *arXiv preprint arXiv:1605.01713*, 2016.

[12] David Baehrens, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert MÃžller. How to explain individual classification decisions. *Journal of Machine Learning Research*, 11(Jun):1803–1831, 2010.

[13] Julius Adebayo, Justin Gilmer, Ian Goodfellow, and Been Kim. Local explanation methods for deep neural networks lack sensitivity to parameter values. *arXiv preprint arXiv:1810.03307*, 2018.

[14] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3319–3328. JMLR. org, 2017.

[15] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[16] Pieter-Jan Kindermans, Sara Hooker, Julius Adebayo, Maximilian Alber, Kristof T Schütt, Sven Dähne, Dumitru Erhan, and Been Kim. The (un) reliability of saliency methods. In *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*, pages 267–280. Springer, 2019.

[17] Sara Hooker, Dumitru Erhan, Pieter-Jan Kindermans, and Been Kim. Evaluating feature importance estimates. *arXiv preprint arXiv:1806.10758*, 2018.

[18] Julius Adebayo, Justin Gilmer, Michael Muelly, Ian Goodfellow, Moritz Hardt, and Been Kim. Sanity checks for saliency maps. In *Advances in Neural Information Processing Systems*, pages 9505–9515, 2018.

[19] Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *J. R. Stat. Soc. Series B Stat. Methodol.*, 68(1):49–67, 2006.

[20] Rene Vidal, Joan Bruna, Raja Giryes, and Stefano Soatto. Mathematics of deep learning, 2017.

[21] Fei Zhang, Patrick P. K. Chan, Battista Biggio, Daniel S. Yeung, and Fabio Roli. Adversarial feature selection against evasion attacks. 2020.

[22] Zhihao Zheng and Pengyu Hong. Robust detection of adversarial attacks by modeling the intrinsic properties of deep neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[23] Stephan Zheng, Yang Song, Thomas Leung, and Ian Goodfellow. Improving the robustness of deep neural networks via stability training. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[24] Utkarsh Mahadeo Khaire and R. Dhanalakshmi. Stability of feature selection algorithm: A review. *Journal of King Saud University - Computer and Information Sciences*, 34(4):1060–1073, 2022.

[25] Eoin M. Kenny and Mark T. Keane. Twin-systems to explain artificial neural networks using case-based reasoning: Comparative tests of feature-weighting methods in ann-cbr twins for xai. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 2708–2715. International Joint Conferences on Artificial Intelligence Organization, 7 2019.

[26] Jianchang Mao, K. Mohiuddin, and A.K. Jain. Parsimonious network design and feature selection through node pruning. In *Proceedings of the 12th*

IAPR International Conference on Pattern Recognition, Vol. 3 - Conference C: Signal Processing (Cat. No.94CH3440-5), volume 2, pages 622–624 vol.2, 1994.

[27] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, 2015.

[28] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015.

[29] Pieter-Jan Kindermans, Kristof T. Schütt, Maximilian Alber, Klaus-Robert Müller, Dumitru Erhan, Been Kim, and Sven Dähne. Learning how to explain neural networks: Patternnet and patternattribution, 2017.

[30] Maksymilian Wojtas and Ke Chen. Feature importance ranking for deep learning. In *Advances in Neural Information Processing Systems*, volume 33, pages 5105–5114. Curran Associates, Inc., 2020.

[31] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-López, Daniel Molina, Richard Benjamins, Raja Chatila, and Francisco Herrera. Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai, 2019.

[32] Peter Norvig Stuart J. Russell. *Artificial intelligence : a modern approach.* 2021.

[33] Jichen Zhu, Antonios Liapis, Sebastian Risi, Rafael Bidarra, and G. Michael Youngblood. Explainable ai for designers: A human-centered perspective on mixed-initiative co-creation. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8, 2018.

[34] Filip Karlo Došilović, Mario Brčić, and Nikica Hlupić. Explainable artificial intelligence: A survey. In *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 0210–0215, 2018.

[35] Pramila Rani, Changchun Liu, Nilanjan Sarkar, and Eric Vanman. An empirical study of machine learning techniques for affect recognition in human–robot interaction. *Pattern Analysis and Applications*, 9, 05 2006.

[36] Bryce Goodman and Seth Flaxman. European union regulations on algorithmic decision-making and a "right to explanation". 2016.

[37] Kaylee Burns, Lisa Anne Hendricks, Kate Saenko, Trevor Darrell, and Anna Rohrbach. Women also snowboard: Overcoming bias in captioning models, 2018.

[38] Bin Yu. Stability. *Bernoulli*, 19(4):1484 – 1500, 2013.

[39] Nimrod Harel, Ran Gilad-Bachrach, and Uri Obolski. Inherent inconsistencies of feature importance, 2022.

[40] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *J. Mach. Learn. Res.*, 3(null):1157–1182, mar 2003.

[41] Le Song, Alex Smola, Arthur Gretton, Justin Bedo, and Karsten Borgwardt. Feature selection via dependence maximization. *Journal of Machine Learning Research*, 13(5):1393–1434, 2012.

[42] R. Tibshirani. Regression shrinkage and selection via the LASSO. *J. R. Stat. Soc. Series B Stat. Methodol.*, 58(2):267–288, 1996.

[43] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the royal statistical society: series B (statistical methodology)*, 67(2):301–320, 2005.

[44] Yang Young Lu, Yingying Fan, Jinchi Lv, and William Stafford Noble. Deeppink: reproducible feature selection in deep neural networks, 2018.

[45] Yaniv Romano, Matteo Sesia, and Emmanuel J. Candès. Deep knockoffs. 2018.

[46] Yifeng Li, Chih-Yu Chen, and Wyeth W Wasserman. Deep feature selection: Theory and application to identify enhancers and promoters. In *International*

Conference on Research in Computational Molecular Biology, pages 205–217. Springer, 2015.

[47] Yotam Hechtlinger. Interpretation of prediction models using the input gradient. *ArXiv: 1611.07634*, 2016.

[48] Marco Tulio Ribeiro, Sameer Singh, and Carlos 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*, pages 1135–1144. ACM, 2016.

[49] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140, 2015.

[50] OpenAI. Chatgpt. https://openai.com/research/language-models-can-explain-neurons-in-language-models, 2023.

[51] D. M. Endres and J. E. Schindelin. A new metric for probability distributions. *IEEE Trans. Inf. Theory.*, 49(7):1858–1860, 2003.

[52] David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[53] O. J. Mengshoel. Understanding the role of noise in stochastic local search: Analysis and experiments. *Artificial Intelligence*, 172(2):955–990, 2008.

[54] J. Friedman, T. Hastie, and R. Tibshirani. *The Element of Statistical Learning*. Springer, Berlin, 2001.

[55] Yifeng Li, Chih-Yu Chen, and Wyeth Wasserman. Deep feature selection: Theory and application to identify enhancers and promoters. *Journal of Computational Biology*, 23(5):322–326, 2016.

[56] Le Song, Alex Smola, Arthur Gretton, Karsten M. Borgwardt, and Justin Bedo. Supervised feature selection via dependence estimation. In *International conference on Machine learning*, pages 823–830, 2007.

[57] Yann LeCun, C. Cortez, and C. Burges. The MNIST Handwritten Digit Database. http://yann.lecun.com/exdb/mnist/.

[58] Dheeru Dua and Casey Graff. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml.

[59] Yale Face Database. http://vision.ucsd.edu/content/yale-face-database.

[60] R. Anderson, C. Gebard, I. Miguel-Escalada, et al. An atlas of active enhancers across human cell types and tissues. *Nature*, 507:455–461, 2014.

[61] I. van der Linden, H. Haned, and E. Kanoulas. Global aggregations of local explanations for black box models. In *Proceedings of SIGIR Workshop on Fairness, Accountability, Confidentiality, Transparency, and Safety*, 2019.

[62] B. Skrlj, S. Dzeroski, N. Lavrac, and M. Petkovic. Feature importance estimation with self-attention networks. In *Proceedings of 24th European Conference on Artificial Intelligence*, 2020.

[63] Pilar García-Díaz, Isabel Sánchez-Berriel, Juan A. Martínez-Rojas, and Ana M. Diez-Pascual. Unsupervised feature selection algorithm for multiclass cancer classification of gene expression rna-seq data. *Genomics*, 112(2):1916 – 1925, 2020.

[64] H. Peng C. Ding. Minimum redundancy feature selection from microarray gene expression data. *Journal of bioinformatics and computational biology*, 3(2):185 – 205, 2005.

[65] Pritha Mahata and Kaushik Mahata. Selecting differentially expressed genes using minimum probability of classification error. *Journal of Biomedical Informatics*, 40(6):775 – 786, 2007. Intelligent Data Analysis in Biomedicine.

[66] Huawen Liu, Lei Liu, and Huijie Zhang. Ensemble gene selection for cancer classification. *Pattern Recognition*, 43(8):2763 – 2772, 2010.

[67] Myron G. Best, Nik Sol, Irsan Kooi, Jihane Tannous, Bart A. Westerman, François Rustenburg, Pepijn Schellen, Heleen Verschueren, Edward Post, Jan

Koster, Bauke Ylstra, Najim Ameziane, Josephine Dorsman, Egbert F. Smit, Henk M. Verheul, David P. Noske, Jaap C. Reijneveld, R. Jonas A. Nilsson, Bakhos A. Tannous, Pieter Wesseling, and Thomas Wurdinger. Rna-seq of tumor-educated platelets enables blood-based pan-cancer, multiclass, and molecular pathway cancer diagnostics. *Cancer Cell*, 28(5):666 – 676, 2015.

[68] Yongjun Piao, Minghao Piao, and Keun Ho Ryu. Multiclass cancer classification using a feature subset-based ensemble from microrna expression profiles. *Computers in Biology and Medicine*, 80:39 – 44, 2017.

[69] Ahmet Saygili. Classification and diagnostic prediction of breast cancers via different classifiers. 2:48–56, 12 2018.

[70] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of International Conference for Learning Representations*, pages 585–592, 2015.

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

[72] François Chollet et al. Keras. https://keras.io, 2015.

[73] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[74] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinv. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(8):1929–1958, 2014.

[75] B. Xue, M. Zhang, W. N. Browne, and X. Yao. Survey on evolutionary computation approaches to feature selection. *IEEE Transactions on Evolutionary Computation*, 20(4):606–626, 2016.

[76] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018.

[77] Guangyu Zhu and Tingting Zhao. Deep-gknock: Nonlinear group-feature selection with deep neural networks. *Neural Networks*, 135:139–147, 2021.

[78] Shuangge Ma, Xiao Song, and Jian Huang. Supervised group lasso with applications to microarray data analysis. *BMC bioinformatics*, 8:1–17, 2007.

[79] Robert Tibshirani, Trevor Hastie, Mike Eisen, Doug Ross, David Botstein, Pat Brown, et al. Clustering methods for the analysis of dna microarray data. *Dept. Statist., Stanford Univ., Stanford, CA, Tech. Rep*, 1999.

[80] Zhang MQ Spellman PT, Sherlock G. Comprehensive identification of cell cycle-regulated genes of the yeast saccharomyces cerevisiae by microarray hybridization. *Mol Biol Cell.*, pages 3273–3297, 1998.

[81] Robert F Lee TI, Rinaldi NJ. Transcriptional regulatory networks in saccharomyces cerevisiae. *Science*, pages 799–804, 2002.

[82] Chen G. Wang L. Group scad regression analysis for microarray time course gene expression data. *Bioinformatics*, page 1486–1494, 2007.

[83] Brzyski D, Gossmann A, Su W, and Bogdan M. Group slope - adaptive selection of groups of predictors. *J Am Stat Assoc.*, pages 419–433, 2019.

[84] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[85] Endres and Schindelin. A new metric for probability distributions, 2003.

[86] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.

[87] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2021.

[88] Minguk Kang, Jun-Yan Zhu, Richard Zhang, Jaesik Park, Eli Shechtman, Sylvain Paris, and Taesung Park. Scaling up gans for text-to-image synthesis, 2023.

[89] OpenAI. Gpt-4 technical report, 2023.