AN ABSTRACT OF THE THESIS OF

Saeed Khorram for the degree of Master of Science in Computer Science presented on March 18, 2020.

Title: Toward Disentangling the Activations of the Deep Networks via Low-dimensional Embedding and Non-negative Factorization

Abstract approved: _____

Fuxin Li

In this thesis, we introduce a novel Explanation Neural Network (XNN) to explain the predictions made by a deep network. The XNN works by embedding a high-dimensional activation vector of a deep network layer non-linearly into a low-dimensional explanation space while retaining faithfulness i.e., the original deep learning predictions can be constructed from the few concepts extracted by our explanation network. We then visualize such concepts for humans to learn about the high-level concepts that deep learning is using to make decisions. We propose an algorithm called Sparse Reconstruction Autoencoder (SRAE) for learning the embedding to the explanation space. SRAE aims to reconstruct only parts of the original feature space while retaining faithfulness. A pull-away term is applied to SRAE to make the explanation space more orthogonal. A visualization system is then introduced for human understanding of the features in the explanation space. The proposed method is applied to explain CNN models in image classification tasks. We conducted a human study, which shows that the proposed approach outperforms a saliency map baseline, and improves human performance on a difficult classification task. Also, several novel metrics are introduced to evaluate the performance of explanations quantitatively without human involvement.

Further, we propose DeepFacto where a factorization layer similar to non-negative matrix factorization (NMF) is added to the intermediate layer of the network and showcase its capabilities in supervised feature disentangling. Jointly training an NMF decomposition

with deep learning is highly non-convex and cannot be addressed by the conventional backpropagation and SGD algorithms. To address this obstacle, we also introduce a novel training scheme for training DNNs using ADMM called Stochastic Block ADMM which allows for simultaneous leaning of non-differentiable decompositions. Stochastic Block ADMM works by separating neural network variables into blocks, and utilizing auxiliary variables to connect these blocks while optimizing with stochastic gradient descent. Moreover, we provide a convergence proof for our proposed method and justify its capabilities through experiments in supervised learning and DeepFacto settings.

Toward Disentangling the Activations of the Deep Networks via Low-dimensional Embedding and Non-negative Factorization

by

Saeed Khorram

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented March 18, 2020
Commencement June 2020

Master of Science thesis of Saeed Khorram presented on March 18, 2020.

APPROVED:

Major Professor, representing Computer Science

Head of the School of Electrical Engineering and Computer Science

---

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Saeed Khorram, Author

ACKNOWLEDGEMENTS

I would like to sincerely thank my parents and my sister for their unconditional support and love. I would also like to thank my advisor, Fuxin Li, for his constructive mentoring during my presence at Oregon State University. Futher, I would like to thank Zhongang Qi and Xiao Fu for the joint discussions and collaborations. At the end, I would like to thank my friends, and those who were by side on this journey.

"هر چیز که در جستن آنی، آنی." ــ مولانا

"What you seek is seeking you." — Rumi

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

Figure Page

LIST OF FIGURES (Continued)

Figure Page

LIST OF FIGURES (Continued)

Figure Page

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF APPENDIX FIGURES

LIST OF APPENDIX TABLES

Chapter 1: Introduction

In the past couple of years, *deep neural networks* (DNNs) have shown impressive performance in many domains such as computer vision [40, 80, 39]. On the whole, they process data by cascading information through multiple non-linear mappings. This gives them the high capacity to capture complex structures in the data. Yet, this comes at the cost of obstructed *interpretability* as the decision making process of the DNNs is still a *black-box* to human understanding. Interpretability is a crucial part of advancing the artificial intelligence (AI) systems, and lack of interpretability hinders the efforts to trust, refine, or learn from them [21]. There typically exists a trade-off between the interpretability and the performance of these systems. Classical machine learning methods such as *rule-based* methods [100], despite having relatively weak performance, are highly interpretable. On the other end of the spectrum, deep learning methods suffer from lack of interpretability while even outperforming humans in certain domains [89].

Further, *adversarial* examples [96, 35] have shown that the decision making of the DNNs is indeed very fragile — by only adding a small amount of intended perturbations to their input, they can be readily fooled. This suggests that solely relying on the confidence of the DNNs predictions, that may have been backed up with high performance on benchmarks, is not wise. This clearly shows the need for explanations in decisions of the DNNs, especially in the domains that they are constantly making critical decisions such as in self-driving cars, medical diagnosis, etc. Simply put, in order to establish *trust*, human needs to understand *how* deep learning makes decisions. Such understanding can help the human to gain additional insights into human-machine interactions — for humans to enhance the current algorithms and to potentially learn from them in tasks they show superior performance.

Interpretable AI, also knows as explainable AI (XAI), is the line of research that attempts to mitigate the lack of interpretability in the AI systems, particularly in deep learning. In the recent years, different directions have been taken toward interpretability in DNNs such as visualizing particular filters in the convolutional neural networks (CNNs) [91, 117], aligning the individual activation units in the CNNs with a set of (human-

labeled) semantic concepts [5], generating attention maps that backtrack a decision to its relevant areas in the input [85, 77, 122]. These methods are often nice and quite informative, but they work on individual images and do not provide high-level concepts that can be broadly applicable to many images simultaneously, nor can we believe they are complete explanations of deep learning predictions.

It is believed that by going deeper in the layers of the neural networks, increasing levels of abstraction are learned, and oftentimes they are aligned with human-interpretable concepts [132, 33]. However, there is no explicit mechanism in the training of the DNNs to enforce the learning of semantic parts. Typically these semantic parts are found using a rigorous exploration of the roles of individual units inside the DNNs [5]. In this thesis, unlike the aforementioned methods, we propose methods to limit the capacity of the DNNs to only a few neurons, hoping to encourage (partial) disentanglement in their latent space. We investigate our hypothesis using two different approaches.

In our first approach (chapter 3), we embed the high-dimensional activation space of the DNNs into a low-dimensional explanation space while retaining the *faithfulness* to the original network, meaning the prediction of the original (to-be-explained) network is faithfully approximated from the few concepts in the embedded space. Further, we encourage the low-dimensional space to have the following properties: *locality*, that the concepts are relatively spatially-localized in images so that humans can understand them, and *orthogonality*, that the concepts themselves are as independent of each other as possible. It should be noted, this is a *post-hoc* explanation method, meaning it would explain the predictions of the networks which are already trained.

In our second approach (chapter 4), we design a method to learn a factorized activation space in an *end-to-end* manner for the goal of feature disentanglement. Non-negative Matrix Factorization (NMF) has been able to generate sparse and interpretable representation due to the non-negative constraints over the factorization matrices [61]. Supporting the belief that DNNs would learn semantic part-of-object filters during training [33, 5], we develop a method to train the DNNs while having NMF over their activations in an *end-to-end manner*. Jointly training an NMF decomposition with deep learning is highly non-convex and cannot be addressed by the conventional backpropagation and SGD algorithms. To address this obstacle, we also introduce a novel training scheme for training DNNs using ADMM which allows for simultaneous leaning of a *non-differentiable* decomposition.

The rest of the thesis is organized as the following. In chapter 2, we will review the common training schemes for training the DNNs and will go over the ADMM algorithm for solving constrained optimization problems. In addition, we introduce favorable properties of the NMF and the previous related work in using NMF toward interpretability. Finally, we give a short introduction to interpretability in deep neural network and how it is related to our work. Chapter 3 and chapter 4 will cover our two approaches that have been introduced above. Lastly, chapter 5 concludes this thesis and provides a summary of the contributions along with the direction for future works.

Chapter 2: Background and Related Works

2.1 Training Deep Neural Networks

Consider a simple binary classification problem given input $\boldsymbol{x}$ and label $y \in \{0, 1\}$ using *logistic regression*, a special case of a neural network (NN) with single neuron, Sigmoid activation function, and *binary cross-entropy* (BCE) loss,

$$\begin{aligned} &\hat{y} = \sigma(\boldsymbol{x}^T \boldsymbol{W} + \boldsymbol{b}); \quad \sigma(x) = \frac{1}{1 + e^{-x}} \\ &\text{loss}(\hat{y}, y) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y}) \end{aligned} \tag{2.1}$$

In the above formulation of logistic regression using a simple NN rises the question *"How are the parameters of the NN, $\theta = \{W, b\}$, being learnt?"* The objective of the classification task is to minimize the loss with respect to the θ. However, no analytical solution can be derived to determine θ in the problem 2.1. This is almost always the case for optimizing the problems formulated by NNs.

Instead, the current approaches use Maximum Likelihood Estimation (MLE) to learn the parameters of the network. MLE estimate the parameters by maximizing the log-likelihood of the data under the proposed statistical model (NN). In other words, MLE selects the parameter values that make the observed data most probable. This is equivalent to finding the parameters that minimize a given loss function. The best set of parameters that minimize the loss is taken as the maximum likelihood estimate. For problem 2.1, we have the following MLE,

$$
\begin{aligned}
\theta_1 &= \arg\max_{\theta} \; p(\boldsymbol{y}|\boldsymbol{X}, \theta) = p_{model}(\boldsymbol{y}|\boldsymbol{X}) \\
&= \arg\max_{\theta} \; \prod_{i=1}^{N} p_{model}(y_i|\boldsymbol{x}_i) \\
&= \arg\max_{\theta} \; \sum_{i=1}^{N} \log p_{model}(y_i|\boldsymbol{x}_i) \\
&= \arg\max_{\theta} \mathbb{E}_{\boldsymbol{x},y\sim p_{data}} \log p_{model}(y|\boldsymbol{x}) \\
&= \arg\min_{\theta} -\mathbb{E}_{\boldsymbol{x},y\sim p_{data}} \log p_{model}(y|\boldsymbol{x}) \\
&= \arg\min_{\theta} \text{Loss}(\theta)
\end{aligned}
\tag{2.2}
$$

where N is the number of data points, $\mathbb{E}_{\boldsymbol{x},y\sim p_{data}}$ is the expectation over the input data, and Loss(θ) is the *empirical risk* or also called *loss function*. The loss function derived in 2.2 is commonly referred to as *Negative Log-Likelihood*. The objective of the *training* in NNs is to minimize the loss, i.e., maximize the log-likelihood.

2.1.1 Gradient Descent and BackPropagation

The minimization problem in 2.2 generally does not have a closed-form solution. To determine the parameters of the model (NN) that lead to the minimum loss value, the first-order iterative optimization method, *Gradient Descent* (GD), and its variants are typically used. Note, this is under the assumption that all the components of the model are (piece-wise) differentiable. The gradient of the loss function at a given point, guides to the direction in which the parameters should be updated to further decrease the loss value.

GD can lead to the global optimum if the optimization objective is convex. However, NNs generally have highly non-convex (and non-smooth) parameter space and there is no guarantee that GD finds the global optimum. This hurdle is more severe in DNNs as the complexity of the parameter space increases. By carefully taking small-enough steps, GD can theoretically lead the optimization into a local optimum which can be far away from the global minimum of the loss function.

In practice, *Stochastic* Gradient Descent (*SGD*) and its variants such as *Adam* [54] are used widely used. These methods enable efficient training on large data sets wherein each *iteration* of the gradient descent update, only a fraction of the data called a *mini-batch* is used. This only gives only an approximation of the gradient. The inherent gradient noise introduced by mini-batch sampling seems to help the optimization in avoiding narrow local optima [51].

To efficiently update the parameters of the NN, *Backpropagation* (BP) [83] algorithm is commonly used to train the feed-forward networks. Backpropagation, instead of calculating the gradient of the loss function (output layer) with respect to each parameter in the network, computes the gradients one layer at a time in a backward fashion. Using the *chain rule*, all the parameters in the network can be efficiently updated in one backward path. After one backpropagation of the error from the last layer to the input layer, one can do a forward-pass in the network to calculate the new value of the loss function and evaluate the performance. The backward and forward steps are repeated until the optimization reaches to the (sub-)optimal minimum.

2.1.2 Training DNNs as Constrained Optimization Problem

As stated in section 2.1.1, SGD and its adaptive learning rate variants e.g., *Adam* are commonly used to optimize a deep neural network (DNN) by the backpropagation algorithm. Although these approaches have been the most successful, there are drawbacks related to SGD training in DNNs [98], such as vanishing gradients in deep layers, a significant memory footprint for storing the gradients, difficulty to parallelize across layers because backpropagation has to be done sequentially, and inability to extend to problems where non-differentiable layers exist.

A recent line of research has focused on training DNNs using optimization techniques that decompose the training into smaller sub-problems, including Block Coordinate Descent (BCD) and ADMM, which we will elaborate more on in section 2.2. On the BCD algorithms, [10] was the earliest to propose training a DNN in a distributed setting by formulating it as a constrained optimization problem. Further, [118, 127, 3, 38] lifted the non-convex activations (e.g. ReLU) and formulating the DNN training as a multi-convex problem and solved it using BCD. [113] proposed simultaneous clustering of the latent space in an auto-encoder and used alternating SGD for training.

On the other hand, [98] proposed a batch gradient-free algorithm for training neural networks using a variant of ADMM. However, due to the closed-form update of all the parameters, the proposed method has limitations (*e.g.* only capable of using simple losses such as Hinge loss and MSE), and cannot be further extended into more complex problems and larger datasets. [128] proposed an ADMM scheme for training deep networks while the parameter updates were using SGD which allowed for more complex problems such as binary hashing.

More close to our work, [36] split DNNs into blocks and trained them separately by introducing gluing variables. This is very close to ADMM, but it did not use the dual variables common in ADMM and did not present a convergence proof for their method. Recently, [105, 119] has provided convergence analysis of ADMM in deep learning by linearly approximating the non-linear constraints in the DNN training problem. However, their work did not address stochastic gradients as in our work.

2.2 Alternating Direction Method of Multipliers

Alternating Direction Method of Multipliers (ADMM) is a class of algorithms for solving constrained optimization problems that gained considerable popularity after [7]. ADMM brings the superior convergence properties of the *method of multipliers* and the decomposability of the *dual ascent* method together. To solve a *separable* optimization problem of the form,

$$\begin{aligned} \underset{\boldsymbol{x},\boldsymbol{z}}{\text{minimize}} \quad & f(\boldsymbol{x}) + g(\boldsymbol{z}) \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{z} = \boldsymbol{c} \end{aligned} \tag{2.3}$$

ADMM, firstly, forms the augmented Lagrangian,

$$L(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{y}) = f(\boldsymbol{x}) + g(\boldsymbol{z}) + \boldsymbol{y}^T(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{z} - \boldsymbol{c}) + \frac{\rho}{2}\|\boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{z} - \boldsymbol{c}\|_2^2 \tag{2.4}$$

where $\boldsymbol{y}$ is the *dual variable* (*Lagrange multiplier*) and $\rho > 0$ is called the *penalty parameter*. Advantaging the *decomposability* introduced in the optimization problem 2.3,

ADMM breaks the augmented Lagrangian into smaller sub-problems by updating the variables in an alternating manner, i.e. at each iteration, the augmented Lagrangian is optimized with respect to only one variable while keeping the other variables fixed at their most recent updated value,

$$\begin{aligned} \boldsymbol{x}^{k+1} &:= \arg\min_{\boldsymbol{x}} L(\boldsymbol{x}, \boldsymbol{z}^k, \boldsymbol{y}^k) \\ \boldsymbol{z}^{k+1} &:= \arg\min_{\boldsymbol{z}} L(\boldsymbol{x}^{k+1}, \boldsymbol{z}, \boldsymbol{y}^k) \\ \boldsymbol{y}^{k+1} &:= \boldsymbol{y}^k + \rho(\boldsymbol{A}\boldsymbol{x}^{k+1} + \boldsymbol{B}\boldsymbol{z}^{k+1} - \boldsymbol{c}) \end{aligned} \tag{2.5}$$

The dual variable update uses a step size equal to the augmented Lagrangian parameter ρ. Throughout this thesis, for more convenience, we are using the *scaled-form* of the ADMM, which can be simply derived by combining the linear and quadratic terms in the augmented Lagrangian and scaling the dual variable $\boldsymbol{u} = (1/\rho)\boldsymbol{y}$,

$$L(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{y}) = f(\boldsymbol{x}) + g(\boldsymbol{z}) + \frac{\rho}{2}\|\boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{z} - \boldsymbol{c} + \boldsymbol{u}\|_2^2 - \frac{\rho}{2}\|\boldsymbol{u}\|_2^2 \tag{2.6}$$

where $\boldsymbol{u}$ is the *scaled dual variable*. This simplifies the ADMM updates to the following.

$$\begin{aligned} \boldsymbol{x}^{k+1} &:= \arg\min_{\boldsymbol{x}} \left(f(\boldsymbol{x}) + \frac{\rho}{2}\|\boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{z}^k - \boldsymbol{c} + \boldsymbol{u}^k\|_2^2 \right) \\ \boldsymbol{z}^{k+1} &:= \arg\min_{\boldsymbol{z}} \left(g(\boldsymbol{z}) + \frac{\rho}{2}\|\boldsymbol{A}\boldsymbol{x}^{k+1} + \boldsymbol{B}\boldsymbol{z} - \boldsymbol{c} + \boldsymbol{u}^k\|_2^2 \right) \\ \boldsymbol{u}^{k+1} &:= \boldsymbol{u}^k + \boldsymbol{A}\boldsymbol{x}^{k+1} + \boldsymbol{B}\boldsymbol{z}^{k+1} - \boldsymbol{c} \end{aligned} \tag{2.7}$$

ADMM is a powerful optimization method that has shown promise in solving complicated optimization problems, especially in large-scale and data-distributed machine learning applications [98]. As mentioned above, the power of ADMM comes from its decomposition of the augmented Lagrangian into simpler loosely-coupled sub-problems which enables it to solve each sub-problem in an efficient and potentially parallel manner.

In the recent years, *stochastic* and *online* versions of the ADMM have been proposed [72, 104, 131]. However, due to large variance of stochastic gradients, slower convergence

rate of $\mathcal{O}(\frac{1}{\sqrt{T}})$ compared with $\mathcal{O}(\frac{1}{T})$ for original ADMM on convex problems has been shown. Moreover, ADMM extensions for a certain family of non-convex problems have been recently proposed [106, 44]. However, DNNs generally do not fall into this family of problems.

That being said, ADMM has been proposed for solving different optimization problems related to deep learning literature. [52, 115] used ADMM for model compression and parameter pruning in deep networks. [71] developed an iterative algorithm using ADMM to enforce constraints on the latent space for inference. [94] proposed a deep network for MRI image reconstruction using ADMM fashion for optimization. Further, there have been attempts to train deep neural networks using ADMM which will be more elaborated on in section 2.1.2.

2.3 Non-negative Matrix Factorization

Non-negative Matrix Factorization (NMF) [61] is a powerful data analysis tool by dimension reduction. NMF linearly approximate a given matrix $\boldsymbol{X} \in \mathbb{R}^{D\times N}$ (containing N samples with input dimensionality D) using low rank matrices namely, the basis matrix $\boldsymbol{M} \in \mathbb{R}^{D\times r}$ and the score matrix $\boldsymbol{S} \in \mathbb{R}^{r\times N}$,

$$\begin{aligned} \text{NMF}(\boldsymbol{X}, r) \;= \underset{\boldsymbol{M},\boldsymbol{S}}{\text{minimize}} \quad & \frac{1}{2}\left\|\boldsymbol{X} - \boldsymbol{M}\boldsymbol{S}\right\|_F^2 \\ \text{subject to} \quad & \boldsymbol{M}_{ij} \geq 0,\ \boldsymbol{S}_{ij} \geq 0 \quad \forall i, j \end{aligned} \tag{2.8}$$

where r is the decomposition rank and usually $r \ll \min(D, N)$.

Unlike methods such as Principle Component Analysis (PCA) [49], by directly imposing non-negativity constraints over the factors in NMF, the latent representation inherits an interpretable decomposition in an unsupervised manner. In other words, NMF does not allow cancellation of the features by subtraction and data is decomposed only by *additive* components which implicitly results into a *meaningful* and *sparse* (part-based) representation [61, 66].

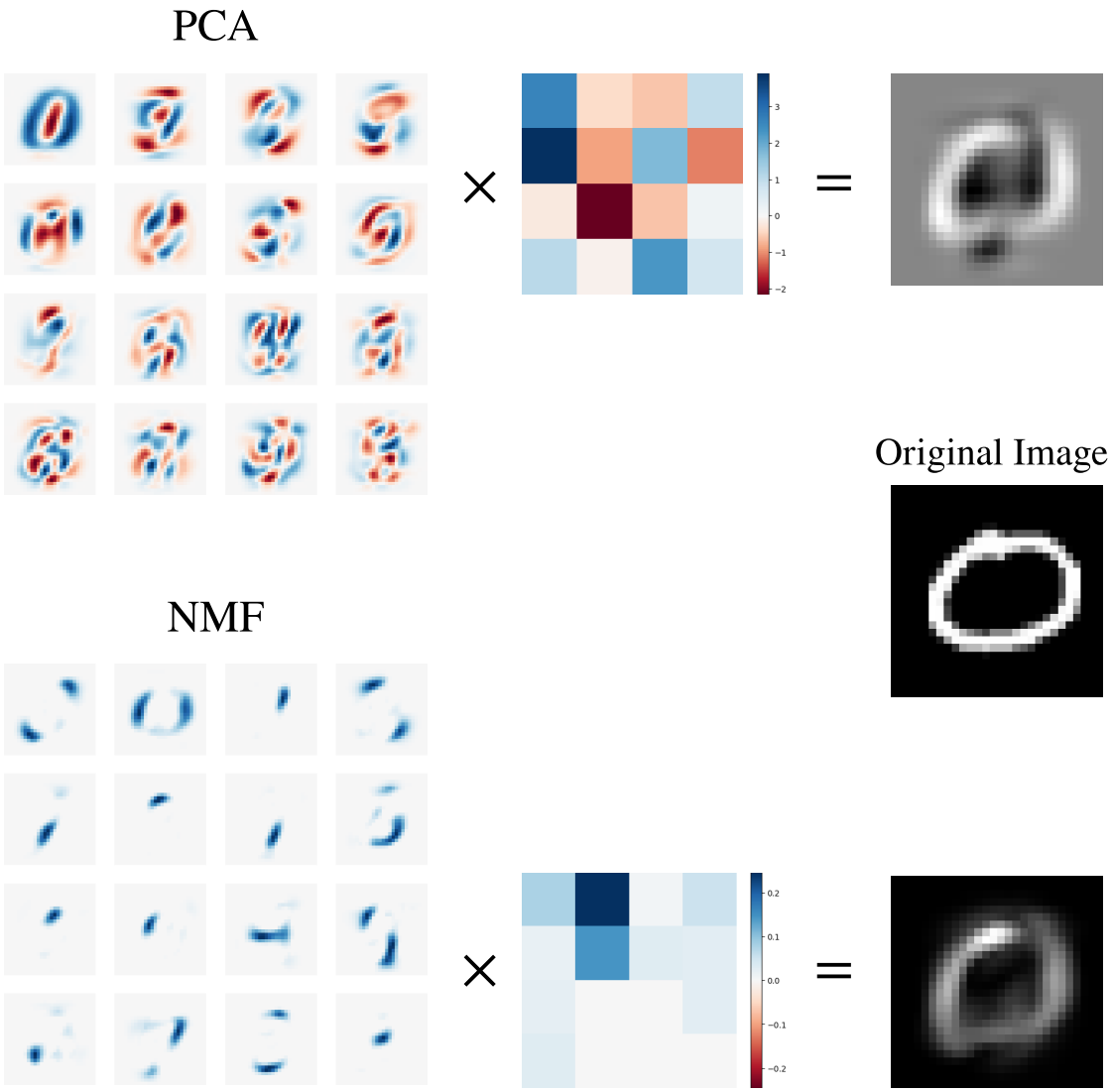


Figure 2.1: Comparison of the decomposition from PCA (top) and NMF (button) with rank $r = 16$. It can be observed that NMF basis vectors are more local and intuitive to humans while PCA basis vectors are similar to distorted whole images. In PCA, the positive (blue) and negative (red) values cancel out each other which is counter-intuitive while in NMF, the additive nature of the decomposition leads to an interpretable and sparse representation — only a few of the basis vectors are contributing (blue) and the rest have near zero contribution (white).

In Figure 2.1, a sample image from MNIST dataset is factorized using PCA and NMF with rank 16. The basis vectors of PCA and NMF are already trained over the (60,000) training samples of MNSIT. As mentioned above, the NMF has more localized and intuitive basis vectors (part-of-object features) and only a few of the basis vectors are positively contributing to the reconstruction: sparse representation. On the other hand, the PCA basis vectors more resemble distorted whole image representations and negative (red) and positive (blue) values cancel out each other in the decomposition which is counter intuitive for human understanding. For the same reason, PCA does not show sparsity properties.

Another favorable property of NMF that contributes to its abstract latent representation is the low(er) rank of factorized matrices — $r \ll min(D, N)$ limits the capacity of the feature representations. Evidently, the new factorized matrices usually also need much lower memory space to store.

NMF has shown promise in various domains including spectral data analysis, text mining, speech processing, data clustering, etc. [107, 86, 111, 37, 75, 27]. Further, it has been shown that variants of NMF are roughly equivalent to k-means clustering algorithm [20] and in a sense, by normalizing each column of the score matrix, it can be interpreted as a posterior cluster membership probabilities — *soft clustering*. Subsequently, build upon these properties, nested and deep models of NMF have been introduced in the literature where the score matrix form one NMF is the input to the next. The nested low-dimensional representations of these deep models have shown to capture hierarchical structures in the data [116, 99, 26, 60].

More recently, the desirable properties of NMF mentioned earlier motivated researches to incorporate it into the DNN architectures. [121] used a DNN to non-linearly transform the data prior to applying the NMF, [14, 48, 102] used extracted features from NMF as input to the DNNs, and [92, 22, 43] used neural networks to perform the (non-negative) matrix factorization.

One needs to keep in mind that a prerequisite for NMF is the input data needs to be also non-negative. This is commonly true for most of the natural data such as text, images, and speech. Respectfully, due to the extensive use of non-negative activation functions in the DNN structures such as *ReLU*, *Sigmoid*, *SoftPlus*, etc. NMF analysis can be readily extended to the activation space of the DNNs. [16] applied NMF over *convolutional* activations which have shown interpretable and coherent behavior over

image parts. However, in their work, NMF was applied post-hoc over pre-trained CNN activations. There is no guarantee that the disentanglement is faithful to the underlying mechanism of the DNN. To the best of our knowledge, NMF layers jointly trained with a deep neural network have not been studied in the past.

2.4 Interpretability in Deep Neural Networks

The development of novel algorithms along with advances in computational hardware has enabled designing deep and complex deep neural networks. On the whole, the notable advantage of the DNNs comes from cascading information through multiple non-linear mappings. This gives them the high capacity to capture complex structures in the data which has resulted in unparalleled performance in many domains such as computer vision [40, 45]. Yet, this comes at the cost of obstructed interpretability as understanding the decision making of the DNNs, involving millions of parameters, is very hard to interpret by humans.

Interpretability is a crucial part of advancing the artificial intelligence (AI) systems, and lack of interpretability hinders the efforts to trust, refine, or learn from them [21]. There typically exists a trade-off between the interpretability and the performance of these systems. Classical machine learning methods such as *rule-based* methods [100], despite having relatively weak performance, are highly interpretable. On the other end of the spectrum, deep learning methods suffer from lack of interpretability while even outperforming humans in certain domains [89].

The explanation for the black-box models has become a significant need in many real applications. In the medical domain, several approaches were proposed to utilize interpretable models to explain the predictions for individual patients in a concise way [11, 63, 101]. In Natural Language Processing, [59] proposes an interactive system which builds a cycle of explanations from the learning system to the user, and then back to the system.

In computer vision, which is most related to the designed experiments in this thesis, several approaches have been taken to make deep networks more interpretable to humans including associating the images with captions/descriptions [55, 56, 65, 50, 41], visualizing individual convolutional filters in the network [117, 5], and *heatmaps* (*saliency maps*) that indicate important regions in the original input space [91, 8, 133, 122, 85, 53, 12].

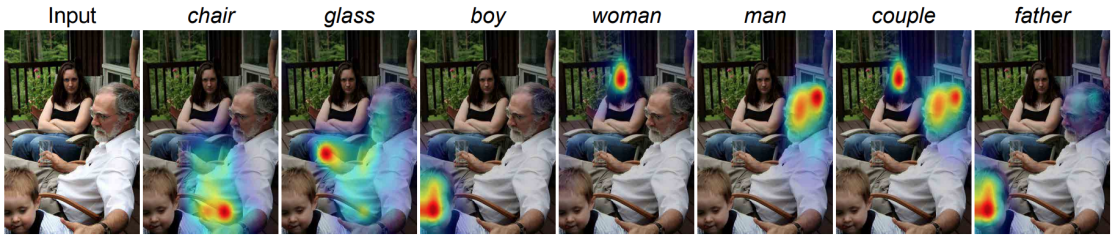


Figure 2.2: Saliency maps generated from Excitation BP [122] corresponding to individual predictions of the CNN classifier.

Saliency maps are roughly the most widely-used method for explaining deep networks, particularly over *convolutional neural networks* (CNNs) in that they preserve the spatial information as going forward in the network. This makes them more intuitive by backtracking the output of the network to the original input (image), unlike in (*fully-connected* networks. Figure 2.2 shows an example of using saliency maps to explain the prediction of the deep networks.

There are two main approaches to generate saliency maps: first, *One-step backpropagation* methods that use the gradient to visualize the explanation. This makes them fast to compute. However, these methods suffer from disadvantages such as being independent of the network weights and performing (partial) image recovery (Guided BackProp [93]), visualizing a diffuse heatmap that is not in-line with human interpretability (Integrated Gradient [95]), and the fact that they only reflect infinitesimal changes in the output which for deep networks (that are highly non-linear functions) this is not necessarily reflective of large enough changes to alter their final prediction. Second, *perturbation-based* (masking) methods that perturb parts of the input to observe which parts are most important in regard to preserving the network's output. This makes them more intuitive to humans, nevertheless, the optimization problem to find the mask is highly memory/time consuming (Mask [17]).

[73] and [85] propose to explain via visual question answering which utilized both natural language descriptions and heatmaps. [82] proposes an explanation technique that tries to explain a single prediction of general models and select several representative predictions to provide a global view of the model. [69] and [24] propose a unified

approach and a streaming algorithm, respectively, to interpreting model predictions. Image captioning approaches [55, 56, 65, 50, 41] need to be trained on human-generated sentences, hence they would not work in any domain where human is not an expert in. Our approach in chapter 3 does not require any natural language descriptions.

Visualizing individual neurons/filters were important for human intuition about CNNs [132, 117, 47]. [5] went to great lengths in visualizing thousands of neurons and asking humans to name each of them. However, it is difficult for such efforts to provide a concise yet complete representation. [1] analyzed the number of filters required to generate good performance on the PASCAL VOC dataset and the conclusion is that each class would need at least dozens of filters. [13] learns a decision tree on top of the deep networks as an attempt for an explainable model. And similar to us, [135] proposed a framework to decompose the activation feature vector of a network into several semantic concepts. These 2 approaches train on an existing vocabulary of attributes, which are not able to achieve very high faithfulness. We adopt the heatmap approach in [122], but visualize explanation features instead of directly visualizing classification results. With this approach, we can generate high-level concepts that are broadly applicable to multiple images in the same category.

There has been a focus on detecting parts using deep neural networks without part annotations, usually in fine-grained classification. [90] and [110] use combinations of convolutional filters to generate part proposals that improve prediction performance. [32] and [120, 126, 125] use various approaches to detect parts. Our focus is different in that we focus on explaining a trained deep model instead of trying to enhance it, and explanations may not necessarily be parts that are usually expressed in terms of bounding boxes as in those approaches. [33] conducted comprehensive experiments on whether semantic parts naturally emerge from convolutional filters. They explored combinations of filters using a genetic algorithm but only combine an average of 5 filters, hence they did not have the dramatic dimensionality reduction effect as in our work.

[129] train a hybrid CNN-LSTM model featuring diversified attention models jointly and generate diverse attention maps similar to ours in the middle of the network, but it cannot be utilized to explain an already-trained DNN because of the joint training that is needed, and there was no attempt in quantitatively evaluating the explanations.

Model compression for deep learning was proposed in [4], where a shallow model is used to mimic the output of a deep network. Most model compression works were used

for speeding up inference [15, 79] instead of explanation. [70, 108] are sparse feature selection methods. Both use L1 regularization (in [70], from different layers) to select nodes from the DNN. XNN (chapter 3) is instead feature extraction. None of [70, 108] can utilize very few (3-7) extracted features as XNN to be faithful to the original CNN both kept up to 10-15% of the original features (about 28 (MNIST) to 248 (ImageNet)). Its hard to visualize these features to humans for the user study we did. Hence, we believe XNN is significantly better than these 2 in conciseness. [124] requires user-defined templates as concepts and retraining of CNN. XNN doesnt need pre-defined templates nor retraining of the CNN.

The main difference between XNN and [64, 2] is that they need to train new networks, while XNN explains an already-trained network. Hence the results in [64] and [2] are mostly on simpler datasets. [64] proposes a prototype-based network that can explain its predictions. The limitation of this approach is that first, it needs to train a new prototype-based network to solve the classification problem; second, the prototype needs to be found from the dataset, thus, the dataset cannot be too complex. The authors only did experiments on simple datasets such as MNIST, 3D-car models, and Fashion MNIST. Similar limitations also exist in [2]. [2] proposes a self-explaining network that consists of a concept encoder, an input-dependent parametrizer, and an aggregation function. It also needs to train a new network to solve the classification and can only be applied to some simple datasets. The above two methods cannot achieve either good classification results or good explanation performance on the complex datasets. Our approach focuses on explaining the existing trained models without changing the models original classification results. Hence, our approach can be applied to more complex datasets and models. After the publication of an earlier version of XNN [78], [97] proposed an approach for explaining complex models but is more suitable for neural nets trained on tabular data. It cannot be applied to complex image datasets.

Chapter 3: Embedding Deep Networks into Visual Explanations

3.1 Introduction

Deep learning has made significant strides in recent years, surpassing human performance in many tasks, such as image classification [58, 40], go-playing [88], and classification of medical images [25]. However, the usage of deep learning in real applications still must overcome a trust barrier. Imagine scenarios with a doctor facing a deep learning prediction: this CT image indicates malignant cancer, or a pilot facing a prediction: make an emergency landing immediately. These predictions may be backed up with a claimed high accuracy on benchmarks, but it is human nature not to trust them unless we are *convinced* that they are reasonable for each individual case. The lack of trust is worsened because of known cases where adversarial examples can fool deep learning to output wrong answers [96, 35]. In order to establish trust, human needs to understand how deep learning makes decisions. Such understanding could also help the human to gain additional insights into new problems, potentially improve deep learning algorithms, and improve human-machine collaboration.

Dictionaries often contain explanations of a concept in the form "A is something because of B, C, and D", e.g. this is a bird because it has feathers, wings and a beak. This type of explanation has two properties. Firstly, it is concise – there are not a hundred reasons that add up to explain that A is something. Secondly, it relies on B, C, and D, which are also high-level concepts. Both are often at odds with deep learning predictions, which are combinations of outputs from thousands of neurons in dozens of layers. Approaches have been proposed to visualize each of the filters [117] and for humans to name them [5], but it is difficult for these approaches to obtain a concise representation. On the other hand, many other approaches generate attention maps that backtrack a decision to its relevant areas in the original image [91, 8, 133, 122, 85]. These are often nice and quite informative, but they work on individual images and do not provide high-level concepts that can be broadly applicable to many images simultaneously, nor can we believe they are complete explanations of deep learning predictions.

In this chapter, we make an attempt to reconcile these explanation approaches by extracting several high-level concepts from deep networks to aid human understanding (Figure 3.1). Our model attaches a separate explanation network to a layer in the deep network to reduce the network to a few human-understandable concepts, from where one can generate predictions similar to the original deep network (Figure 3.2(a)). We focus on making those concepts to have several properties: *faithfulness*, that the deep learning predictions can be faithfully approximated from those few concepts; *locality*, that the concepts are relatively spatially localized in images so that human can understand them; and *orthogonality*, that the concepts themselves are as independent from each other as possible.

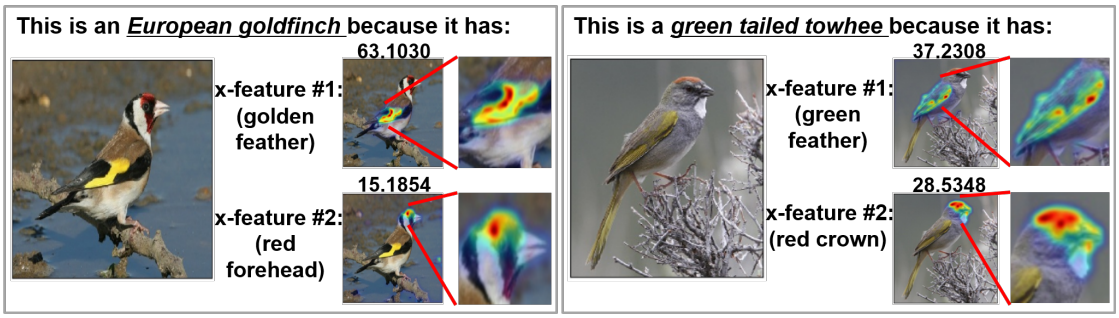


Figure 3.1: Examples of explanations that our approach can generate: the left figure shows that there are two key features (two with highest contribution value indicated on top of each heatmap) to predict *European goldfinch*: golden feather and red forehead; the right figure shows that there are two key features to predict *green tailed towhee*: green feather and red crown. Note that our approach generates the visualizations for human to deduct those feature maps, without requiring any textual annotation to train.

Our model does NOT train from ground truth concepts defined by human, either defined by labels, attributes, or text. It directly infers concepts from the learning network. The reason we deliberately choose not to use human concepts is to adapt to future situations where the deep network may perform a task in a domain in which human does not have expert knowledge. In such cases, explanation methods based on human knowledge would fail, while ours can still work. We evaluate our approach by 1) Human evaluation, where humans are presented different explanations to check which one improves their categorization capabilities. 2) Metric-based evaluation, where we define

quantitative metrics for the aforementioned desired properties of an explanation network and evaluate them on two different datasets — a fine-grained bird classification and a scene recognition datasets which both have rich ground truth annotations allowing us to compute the introduced metrics.

Although the experiments in the chapter focus on convolutional neural networks (CNN) applied to images, the explanation framework we develop is general and applicable to other types of deep networks as well. We believe this is one of the first steps towards general explainable deep learning that can advance human knowledge and enhance future collaboration between humans and machines.

Our contributions in this chapter are as follows:

- We propose a novel explanation network to form a low-dimensional explainable concept space from deep networks. A sparse reconstruction autoencoder with a pull-away term is proposed to make the explanation network faithful and orthogonal as defined previously.
- We present a visualization paradigm for human understanding of the concept space.
- We present a user study that shows our explanations can improve human performance on difficult tasks.
- We propose automatic quantitative metrics to evaluate the performance of an explanation algorithm for faithfulness, locality and orthogonality. Experimental results show that the proposed explanation methods provide insights to how the deep network models work.

3.2 XNN: eXplanation Neural Network

3.2.1 The Explanation Network

Given a deep learning network (DNN) as a prediction model, we propose to learn an extra Explanation Neural Network (XNN) (Figure 3.2(a)), which can be attached to any intermediate layer of the DNN. The XNN attempts to learn an embedding that lowers the dimensionality of the intermediate layer of the DNN, and then directly learns a mapping from the embedding space that mimics the output of the original DNN model. We denote

the input feature space of the XNN as $\mathbf{Z}(\mathbf{x}; \mathbf{W})$, where $\mathbf{x}$ are the input features and $\mathbf{W}$ are the parameters (from all layers) of the original DNN model, respectively, and $\mathbf{Z}$ represents the output of a particular intermediate layer of the DNN. The XNN is used to embed $\mathbf{Z}$ to an explanation space, denoted as $\mathbf{E}_{\boldsymbol{\theta}}(\mathbf{Z})$, where $\boldsymbol{\theta}$ represents parameters of the embedding that needs to be learned. As a shorthand, we will also refer to the explanation space as an *x-layer*, and each dimension in the x-layer an *x-feature*. Note that in the explanation, we do not attempt to change the parameters $\mathbf{W}$ of the original DNN model. The explanation network can in principle be attached to any intermediate layer of the DNN, although the closer to the prediction, the higher level the concepts are and it becomes easier to mimic the prediction of the DNN with a low-dimensional embedding.

We believe that for the explanation network to be understandable, it needs to generate a few concepts that preserve the original prediction results $\hat{\mathbf{y}}$, i.e., we would need a low-dimensional feature embedding to be faithful to the DNN. We propose to obtain faithfulness by explaining binary classification or classification with a few outputs. An explanation to classification with hundreds of classes can be decomposed into explaining smaller tasks such as those. For a prediction $\hat{\mathbf{y}}$ with lower dimensionality than the explanation space, we can definitely assume that the explanation network could remain faithful to the prediction, since a naive case would be to use $\hat{\mathbf{y}}$ as the explanation, which is perfectly faithful but not interpretable. Hence, the low-dimensional embedding $\mathbf{E}$ can also be thought of as expanding $\hat{\mathbf{y}}$ to a few more dimensions, therefore enriching the explanations.

In this chapter, we focus on attaching our proposed XNN to fully-connected layers, although some experiments attaching XNN to convolutional layers will also be shown. The concepts generated in these layers are rather high-level, and our conceptual goal is to visualize those concepts and to make humans learn them: human has an excellent deep neural network in the brain for learning and generalizing perceptual concepts. We would like to show humans examples from a small number of perceptual concepts from the explanation space, so that people can utilize their own perceptual neural network for learning and naming those. Our primary tool for this display is heatmaps (e.g. Figure 3.1) highlighting a specific region in the image, similar as those used in attention models in prior work. Our work will provide several different and largely orthogonal concepts, visualized by heatmaps, for improving the understanding of the predictions from a DNN. The two main topics in the explanation network are the embedding algorithm and the

visualization of the explanations, which will be discussed in the next three subsections.

3.2.2 Embedding to the Explanation Space

First of all, explanation space optimization attempts to be faithful to the prediction of the original DNN:

$$L_{FS} = \min_{\boldsymbol{\theta},\mathbf{v}} \frac{1}{M} \sum_{i=1}^{M} L\Big(f\big(\mathbf{E}_{\boldsymbol{\theta}}(\mathbf{Z}^{(i)}); \mathbf{v}\big), \hat{\mathbf{y}}^{(i)}\Big) \tag{3.1}$$

where $\mathbf{Z}^{(i)} = \mathbf{Z}(\mathbf{x}^{(i)}; \mathbf{W})$ is the output of an intermediate layer in DNN for instance $\mathbf{x}^{(i)}$; parameter $\boldsymbol{\theta}$ is used to form the explanation space $\mathbf{E}_{\boldsymbol{\theta}}(\mathbf{Z}^{(i)})$; parameter $\mathbf{v}$ is used to build a predictor $f(\mathbf{E}; \mathbf{v})$ from the x-features to mimic $\hat{\mathbf{y}}^{(i)}$, $\hat{\mathbf{y}}^{(i)}$ is the output of the original DNN model as well as the explanation target for instance $\mathbf{x}^{(i)}$, we usually use the DNN output in layers before the softmax layer to prevent interactions with predictions on other categories; M is the number of training examples; L is a loss function, usually a regression loss such as squared loss or log loss. However, as we argued in Sec. 3.2.1, this formulation might be almost degenerate if $\hat{\mathbf{y}}^{(i)}$ can be used as the explanation variable. Hence, additional terms need to be added to prevent degeneracy and improve interpretability.

We claim that low-dimensional embeddings are more effective when they reconstruct the original high-dimensional feature space better. The degenerate solution $\hat{\mathbf{y}}$ is usually not good in reconstructing the high-dimensional deep feature space. By jointly optimizing on the faithfulness and reconstruction loss, we hope to explain the prediction $\hat{\mathbf{y}}$ in several aspects, and these individual aspects may reconstruct $\mathbf{Z}$ better. Thus, adding reconstruction loss $L\Big(\mathbf{E}_{\widetilde{\boldsymbol{\theta}}}^{-1}\big(\mathbf{E}_{\boldsymbol{\theta}}(\mathbf{Z}^{(i)})\big), \mathbf{Z}^{(i)}\Big)$ to optimization (3.1) will prevent degeneracy and improve locality. Here $\mathbf{E}_{\widetilde{\boldsymbol{\theta}}}^{-1}$ is a mapping that maps from the explanation space $\mathbf{E}$ back to $\mathbf{Z}$, $\widetilde{\boldsymbol{\theta}}$ is the parameter for this mapping. However, when the weight of the reconstruction loss is large in the optimization, features irrelevant to the prediction target may also be reconstructed.

To avoid this, we propose to enhance the objective by adding a sparsity term which reconstructs *some dimensions* of the original features $\mathbf{Z}$, but not all of them. By attempting to reconstruct some dimensions of $\mathbf{Z}$ with only a few embeddings, and to mimic the original predictions $\hat{\mathbf{y}}$ with the same embeddings, the maximal amount of diverse information that is relevant to $\hat{\mathbf{y}}$ in $\mathbf{Z}$ needs to be packed in the low-dimensional

space. Packing redundant information in correlated dimensions would be harmful for reconstruction, and reconstructing irrelevant features would harm the ability to recover $\hat{\mathbf{y}}$. By introducing a sparse penalty, we define the sparse reconstruction loss as:

$$L_{SR} = \text{Sparsity}(\mathbf{Q}); \quad Q_k = \frac{1}{M}\sum_{i=1}^{M} L\Big(\mathbf{E}_{\widetilde{\boldsymbol{\theta}}}^{-1}\big(\mathbf{E}_{\boldsymbol{\theta}}(\mathbf{Z}^{(i)})\big)_k, Z_k^{(i)}\Big) \tag{3.2}$$

where Q_k, $Z_k^{(i)}$, and $\mathbf{E}_{\widetilde{\boldsymbol{\theta}}}^{-1}\big(\mathbf{E}_{\boldsymbol{\theta}}(\mathbf{Z}^{(i)})\big)_k$ are the k-th dimension of $\mathbf{Q}$, $\mathbf{Z}^{(i)}$, and $\mathbf{E}_{\widetilde{\boldsymbol{\theta}}}^{-1}\big(\mathbf{E}_{\boldsymbol{\theta}}(\mathbf{Z}^{(i)})\big)$, respectively. In the optimization, Q_k measures the capability of reconstructing the k-th dimension in the space of $\mathbf{Z}$. The sparsity term will be detailed in Sec. 3.2.3.

Finally, to make the x-features in the explanation space more orthogonal to each other, an orthogonality loss can also be added to the optimization. Here we utilize the pull-away term (PT) [130] that has been successfully applied in generative adversarial networks:

$$L_{PT} = \frac{1}{n(n-1)}\sum_{l=1}^{n}\sum_{l'\neq l}\Big(\frac{\mathbf{E}_l^T\mathbf{E}_{l'}}{\|\mathbf{E}_l\|\,\|\mathbf{E}_{l'}\|}\Big)^2 \tag{3.3}$$

where n is the number of x-features, $\mathbf{E}_l = \mathbf{E}_{\boldsymbol{\theta}}(\mathbf{Z})_l$ represents the vector for the l-th x-feature over the training set $\mathbf{Z}$. We define the final optimization problem as:

$$L_{SRAE} = L_{FS} + L_{SR} + L_{PT} \tag{3.4}$$

where SRAE (Sparse Reconstruction Autoencoder) is our proposed model which handles the faithfulness, locality, and orthogonality objectives. SRAE is a neural network hence can be seamlessly combined with the prediction DNN, making the following visualization process (introduced in Sec. 3.2.4) simple.

3.2.3 Implementation Details

Our aim is at reconstructing some specific features which focus on the prediction target instead of reconstructing the whole feature space. Various sparsity functions can be used here such as the L_1 penalty function, epsilon-L_1 penalty function [62], etc. We choose the log penalty $\log(1 + q \cdot r^2)$ [62] (Figure 3.2(c-d)). Here we use $r^2 = Q_k$, the average

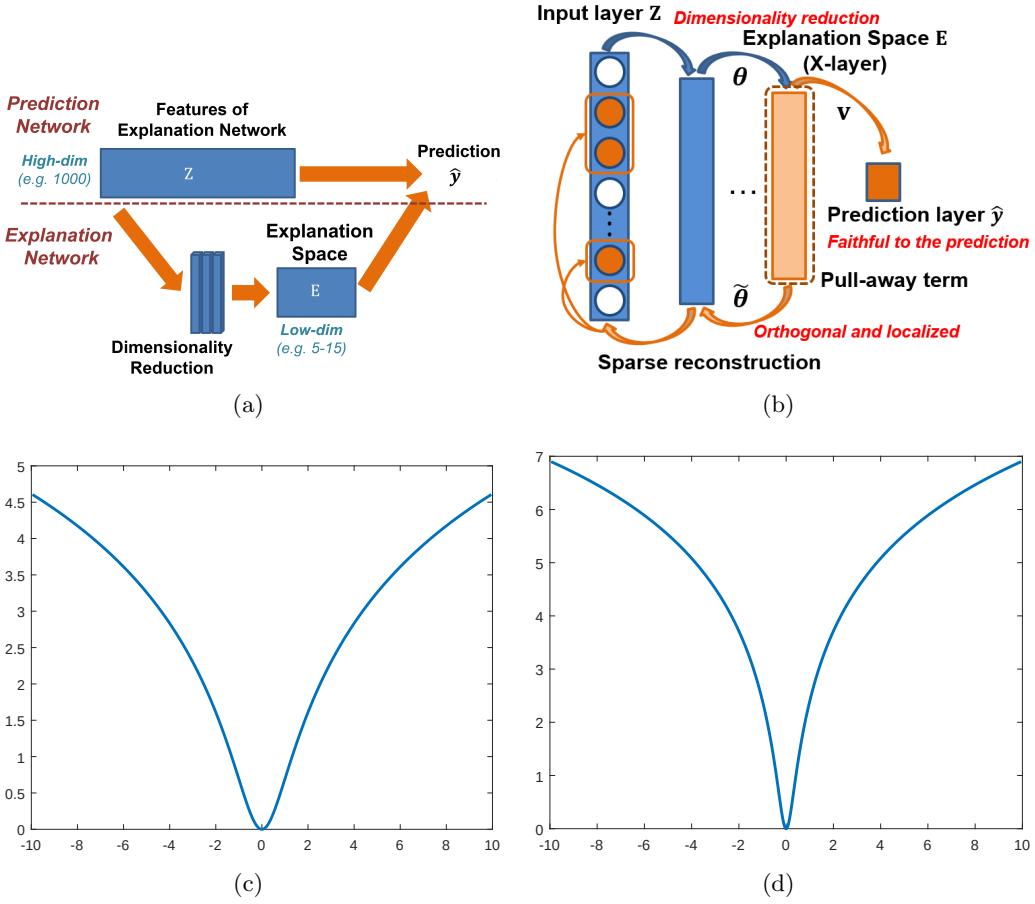


Figure 3.2: (a) Conceptually, the explanation network (XNN) is a mimic network with a small bottleneck layer so that the original deep learning prediction $\hat{\mathbf{y}}$ can be reproduced from this low-dimensional space. An explanation network can be attached to any layer in the prediction deep network (DNN). The output of the DNN can be faithfully recovered from this low-dimensional explanation space, which represents high-level features that are interpretable to humans. (b) Illustration of the SRAE used for the explanation network. Both the prediction and sparse reconstruction are generated from the explanation space. Also, the pull-away term helps the explanation space to be orthogonal; (c) The log penalty function $\log(1 + q \cdot r^2)$ when $q = 1$; (d) The log penalty function $\log(1 + q \cdot r^2)$ when $q = 10$.

squared reconstruction error on the k-th dimension over the whole training set:

$$\begin{aligned} Sparsity(\mathbf{Q}) &= \frac{1}{S_z}\sum_{k=1}^{S_z}\log(1+q\cdot Q_k), \\ Q_k &= \frac{1}{M}\sum_{i=1}^{M}\left\|\mathbf{E}_{\tilde{\boldsymbol{\theta}}}^{-1}\Big(\mathbf{E}_{\boldsymbol{\theta}}(\mathbf{Z}^{(i)})\Big)_k - Z_k^{(i)}\right\|^2 \end{aligned} \tag{3.5}$$

where $q > 0$ is a sparsity parameter (Figure 3.2(c-d)), S_z is the dimensionality of the feature space $\mathbf{Z}$. Note that SRAE is different from conventional sparse autoencoders in which the autoencoder activations in the hidden layers are constrained to be sparse. In SRAE, the sparsity constraint is on the amount of input dimensions to be reconstructed. The log penalty (Figure 3.2(c-d)) is a robust loss function, in the sense that large r increases the loss function sublinearly (less than an L_1 penalty $|r|$ where the increase is linear). Some dimensions of $\mathbf{Z}$ can afford to have no reconstruction at all (large r) without suffering too much loss. Hence this loss function achieves the goal that only some of the input dimensions are *selectively reconstructed*, instead of all of them. The exact dimensions that are reconstructed are chosen automatically by the learning procedure itself.

The illustration of the proposed SRAE used for the explanation network is shown in Figure 3.2(b). The encoding layer in SRAE forms the explanation space $\mathbf{E}$ (Figure 3.2(b)). The pull-away term can be applied directly to the encoding layer of SRAE. Using the least squares loss again for faithfulness, the optimization of the SRAE in equation (3.4) can be re-written as follows:

$$\begin{aligned} \min_{\boldsymbol{\theta},\tilde{\boldsymbol{\theta}},\mathbf{v}} \frac{1}{M}\sum_{i=1}^{M}\left\|\mathbf{v}^\top\mathbf{E}_{\boldsymbol{\theta}}(\mathbf{Z}^{(i)}) - \hat{\mathbf{y}}^{(i)}\right\|^2 + \beta\cdot\frac{1}{S_z}\sum_{k=1}^{S_z}\log(1+q\cdot Q_k) \\ + \eta\cdot\frac{1}{n(n-1)}\sum_{l=1}^{n}\sum_{l'\neq l}\Big(\frac{\mathbf{E}_l^T\mathbf{E}_{l'}}{\|\mathbf{E}_l\|\,\|\mathbf{E}_{l'}\|}\Big)^2 \end{aligned} \tag{3.6}$$

where the 3 terms are faithfulness, sparse reconstruction and orthogonality terms, respectively. β is the parameter for the sparse reconstruction; η is the parameter for the orthogonality term. The prediction result $\mathbf{v}^\top\mathbf{E}_{\boldsymbol{\theta}}(\mathbf{Z}^{(i)})$ of SRAE is denoted as $\bar{y}_j^{(i)}$.

With a trained XNN we would obtain an explanation embedding $\mathbf{E}_{\boldsymbol{\theta}}(\mathbf{Z})$ and a linear

predictor $\mathbf{v}^\top \mathbf{E}$ so that the output of the network can be explained with a weighted sum of x-features $\mathbf{E}_{\boldsymbol{\theta}}(\mathbf{Z})$ with the weight $\mathbf{v}^T$ indicating the contribution of each individual x-feature to the final prediction. In conjunction with the visualization paradigm in the next subsection, this facilitates a better understanding of the black-box DNN model.

3.2.4 Visualizing the Explanation Space

The goal in the visualization of low-dimensional explanation features is to bridge the communication gap between human and machine, and enable human to name concepts learned by the explanation network and be able to construct sentences with those named concepts in the future. Nevertheless, in this chapter, we only focus on visualizing the concepts. We utilize ExcitationBP [122] to compute the contrastive marginal winning probability (c-MWP) from each neuron in x-layer to the pixels in the original image, then generate the heatmaps using c-MWP normalized on each neuron for each image. The reason for choosing this algorithm over competitors such as Grad-CAM [85] or RISE [76] is that this algorithm gives more detailed predictions while those others are quite blurry and hard for humans to interpret. Figure 3.1 shows some examples of the visualization results of these concepts learned by the proposed explanation network. The number above each heatmap represents the contribution of this x-feature to the final prediction. For instance, The left example in Figure 3.1 shows that there are two key features to predict *European goldfinch*: x-feature #1 (contribution: 63.1030) and x-feature #2 (contribution: 15.1854) where x-feature #1 is much more important than x-feature #2 based on their contribution to the final prediction. Note that our approach generates the visualizations for human to deduct those features, without requiring any textual annotation to train. For the case of Figure 3.1, humans can study the visualizations and then name x-features #1 and #2 as *golden feather* and *red forehead*, respectively.

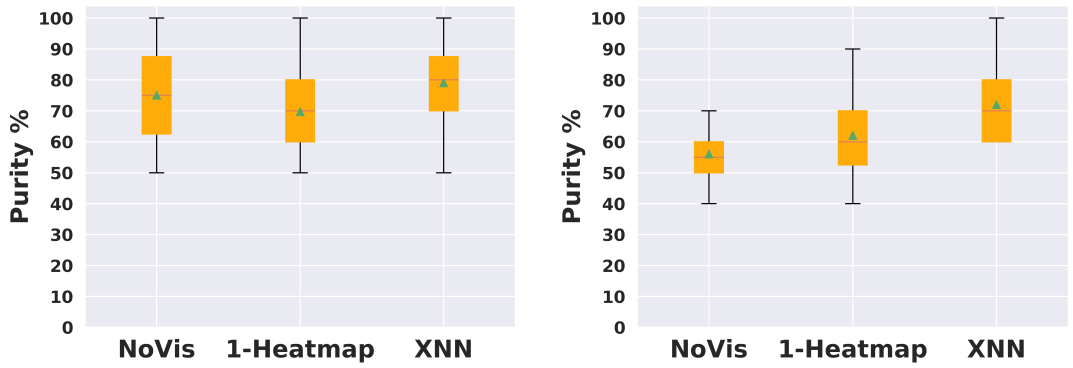
3.3 Experiments and Results

3.3.1 Human Evaluation

To investigate the effectiveness of the proposed XNN with SRAE, we designed a user study where participants were asked to cluster images that are normally difficult to

discern by untrained humans. The goal was to inspect whether the XNN can provide interpretable visualizations to non-experts so they can differentiate between unlabeled samples about which they do not have prior knowledge. We compare **XNN** against 2 baselines, one without any visualization (**NoVis**), and one with a single heatmap derived from the original output $\hat{\mathbf{y}}$ (**1-Heatmap**). For fairness, the same heatmap approach, ExcitationBP [122] is used throughout the study. This approach is chosen since it obtains detailed heatmaps, rather than some other performant approaches which are more blurry [85].

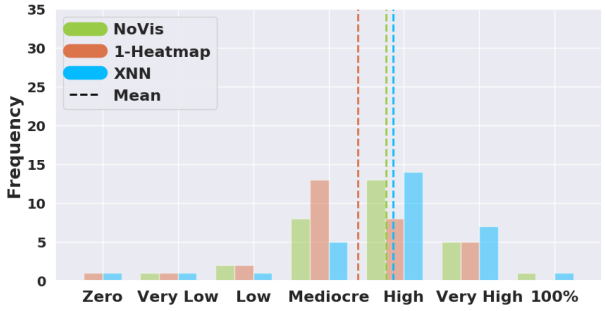
We selected 4 categories of image samples from 2 different datasets where the distinction among the categories is not obvious to non-experts: first, 4 categories of visually alike seagulls (California Gull, Herring Gull, Slaty-backed Gull, and Western Gull) were selected from the CUB-200-2011 dataset [103], referred to as "CUB" in the rest of the chapter. Second, we also selected 4 categories of breast cancer cells (Actinedge, Hemispherebleb, Lamellipodia, and Smalbleb) from microscopic images [23], referred to as "CEL". For each dataset, 200 images were selected where each category contained 50 images.



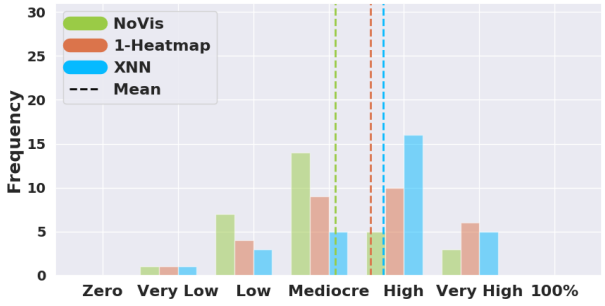
(a) (b)

Figure 3.3: Boxplot of clustering purity with respect to the original CNN prediction obtained from the human evaluation for NoVis, 1-Heatmap, and the XNN on the (a) CEL dataset and (b) CUB dataset.

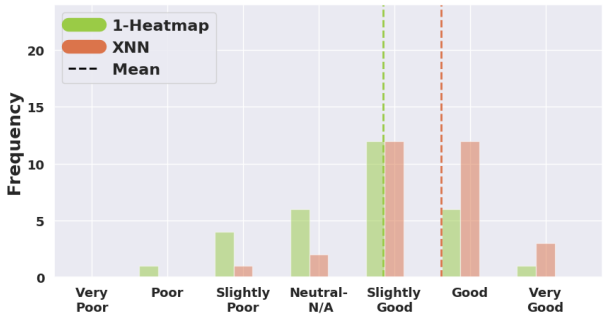
The evaluation interface comprises of 3 main stages: **NoVis** (No Visualization) where the participants were not provided with any visual explanation and they were asked



(a)



(b)



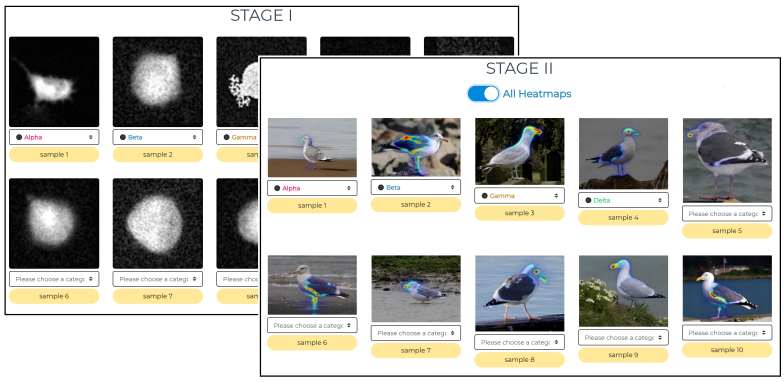
(c)

Figure 3.4: Participants' confidence level in different stages that their answers would correctly group all the sample images for the (a) CEL dataset (b) CUB dataset. (c) Participants' answers to the question that how do they evaluate the performance of the 2 different AI agents helping them on grouping images, 1-Heatmap as the baseline (in green color) and XNN as our approach (in red).

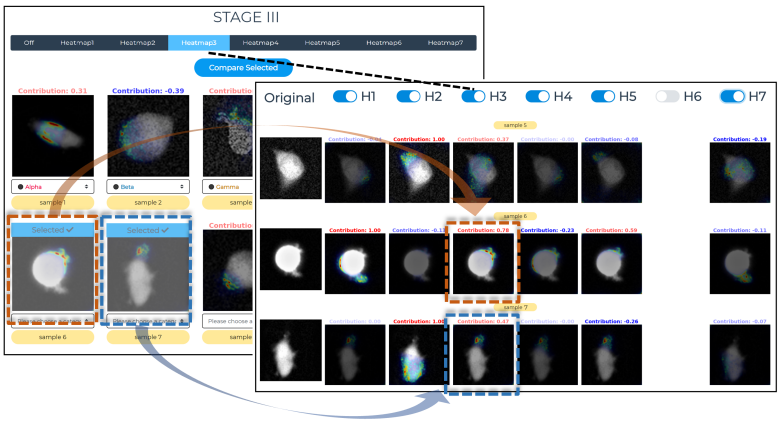
to cluster images only based on their own beliefs of the differences and/or similarities among the samples. **1-Heatmap** is the baseline where the participants were provided with heatmaps derived from the original output $\hat{\mathbf{y}}$ using ExcitationBP to assist them on clustering. They were told that the AI agent utilizes these heatmaps to distinguish one sample from another. **XNN** refers to our method where for each sample image users were provided with different heatmaps generated by ExcitationBP corresponding to each x-feature to help them on decision making. They were told that a contribution of all the different heatmaps was used by the AI agent when trying to identify sample images. Snaps of our user interface at the aforementioned stages can be found in Figure 3.5.

For each stage, users were shown 2 batches containing 10 sample images, one from the CUB dataset and the other one from the CEL dataset. To reduce randomicity, 5 sets of 10 images are drawn randomly at first, and the batch shown to the users were randomly selected from these 5 sets. In addition, the order whether XNN or 1-Heatmap was shown first is random, the order whether CEL or CUB is shown first is also random, but the dataset would always be alternated. For instance, one possible order for the stages and the samples shown can be as following: *(NoVis-CEL, NoVis-CUB, XNN-CEL, XNN-CUB, 1-Heatmap-CEL, 1-Heatmap-CUB).* Hence, the users had to alternate between the 2 datasets to reduce memorizations of the previous visualizations they have seen on a dataset. For both datasets, the visualizations shown in the XNN stage were generated from the 7 x-features that were obtained from embedding the high-dimensional feature space using our proposed model formulated in section 3.2.

Human study results from 30 participants are shown in Figure 3.3. We measured purity of user clusterings with respect to the original predictions of the network in each stage. For the CUB dataset, **XNN** achieved purity of 72% while **NoVis** and **1-Heatmap** achieved 56% and 62% , respectively. For the CEL dataset, mean purity of 75%, 69.6%, and 79% was obtained for **NoVis**, **1-Heatmap**, and **XNN**, respectively (Fig. 3.3(a)). The difference of the results between the **XNN** and **1-Heatmap** was statistically significant for both datasets, which means XNN helps human understand more on these difficult classification tasks than just seeing one heatmap. The results show that humans are better in classifying the CEL dataset even if no visualization is shown, but struggles significantly to distinguish the 4 types of seagulls in the CUB dataset. XNN significantly improved their performance in CUB by pointing out the exact features that are salient to separate the birds which are otherwise very similar. In CEL, 1-Heatmap



(a)



(b)

Figure 3.5: Snaps from different stages of our designed human interface. (a) NoVis stage with CEL images (back) and ExcitationBP stage with CUB images (front). In the ExcitaitonBP stage, users could either hover over the sample images to see the ExcitationBP generated heatmaps or toggle on/off heatmaps for all images at once using a button in the interface (b) the XNN stage with another batch of CEL (back) images where the users could toggle through all the visualizations from x-features 1-7. In this snap, Heatmap 3 is toggled on. The value on top of each heatmap indicates the normalized contribution of that x-feature to the final prediction (positive values in red and negative values in blue). Also, users could select images by clicking on them (the ones with blue banner on top) and compare their heatmaps from all x-features in one place (front). Here, Heatmaps (H) 1-5&7 are toggled on and Heatmap (H) 6 is toggled off.

seems to have even hurt human performance a bit.

During the human study, the participants were also asked the question " *What is your confidence level that the answers you picked will group all the samples correctly*?" with Likert scale answers: *('Zero', 'Very Low', 'Low', 'Mediocre', 'High', 'Very High', '100%')*. The obtained results are illustrated in Figure 3.4. It can be seen that the users' confidence level in the validity of their answers is in accordance with the actual clustering results that were obtained. For the CEL dataset, participants' confidence in the XNN stage was marginally higher than the 1-Heatmap stage (p-value: 0.083). However, for the CUB dataset, the difference of the visualization approaches is not noticeable yet both of them seem to elevate the confidence level of the participants compared to no visualization approach.

At the end of the study, we handed out printed questions to the participants and asked them to rate the performance of the 2 different AI agents (visualization methods) that were assisting them during the study, the EcxitaitonBP on the class (baseline) and x-features (our approach) in distinguishing images from each other. The users could rate the performance in a Likert scale with the options: *('Very Good', 'Good', 'Slightly Good', 'Neutral or N/A', 'Slightly Poor', 'Poor', 'Very Poor')*. The obtained results are depicted in Figure 3.4(c). This shows that humans tend to significantly prefer the performance of our approach over the baseline (p-value: 0.0049).

3.3.2 Quantitative Evaluation Metrics

We believe that evaluating explanations objectively without a human study is also important, because simple parameter variations can easily generate thousands of different explanations, vastly outpacing the speed of human studies. In this chapter, we make an attempt to define some quantitative metrics without human evaluation. We utilize the CUB-200-2011 dataset [103] and Places365 dataset [134] in our quantitative experiments. The former dataset is a fine-grained classification task with 200 categories of birds. From a network trained on all the 200 categories, we attempt to explain the logits (before the softmax layer) of each category separately with one XNN per category. This dataset is chosen because in addition to category labels and bounding boxes surrounding each object, it also has part labels denoted as one pixel per part for each object as additional ground truth. One can argue that the majority of bird classifications are based on

specific, discriminative parts of the bird, which can be confirmed from encyclopedias and expert annotations [81]. The latter dataset (Places) is a task for scene understanding and recognition, where we selected 10 room categories from the ADE20K dataset [136]. This dataset is chosen because besides the scene labels, there also exist segmentation labels as additional ground truth for some images in some categories.

For the CUB-200-2011 dataset, the fine-tuned VGG19 model [90] is used as the prediction DNN to be explained. For the Places365 dataset, the fine-tuned VGG16 model [134] is used as the prediction DNN to be explained. The explanation network is a 3-middle-layer SRAE with $800, 100, n$ hidden units in each layer, respectively, where n represents the number of x-features. For CUB, we trained an explanation network on each of the 200 bird categories. For each category, we utilized 50 positive examples and $8,000$ negative examples as the training data; the remaining positive examples $(8-10)$ and $2,000$ negative examples as the testing data. In the training process, we enhance the weights of the positive examples to avoid imbalance. n is set to 5, as our experiments showed that more x-features do not improve performance and create x-features which have 0 weight in $v_i E_i$, indicating that one one-against-all classifier of one bird may not depend on many high-level visual features. For Places, we trained an explanation network on each of the 10 scene categories which are different kinds of rooms and have enough images with object labels in ADE20K to evaluate, including *bathroom, bedroom, conference room, dining room, home office, hotel room, kitchen, living room, office, and waiting room.* For each category, we utilized $4,000$ positive examples and $20,000$ negative examples as the training data; about $1,000$ positive examples and $4,000$ negative examples as the testing data. n is also set to 5 for Places. We compared the proposed SRAE with a fully-connected neural network (NN), a conventional stacked autoencoder with faithfulness loss and traditional reconstruction loss (SAE), a classic autoencoder with only traditional reconstruction loss and without faithfulness loss (CAE), a feature selection model (Lasso) on $\mathbf{Z}$, as well as directly performing ExcitationBP on the classification output $\hat{\mathbf{y}}$ (1-Heatmap). The baseline neural network methods (NN and SAE) can also perform a faithful dimensionality reduction and are the most closely related to our approach. Lasso represents a feature selection approach which selects several most useful dimensions directly from $\mathbf{Z}$ and tries to mimic the network decision as a linear combination of these features. All the learning-based approaches (SRAE, NN, SAE, CAE, and Lasso) were tuned to the optimal parameters by cross-validation on the

training set.

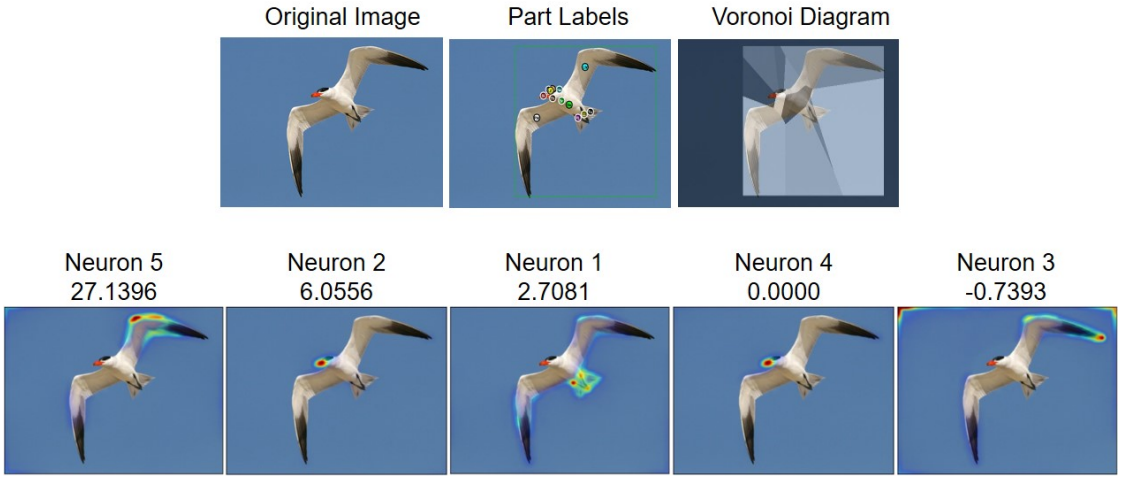


Figure 3.6: A simple example generated by the explanation module. The first line shows the original image, the part labels of the image in the ground truth, and the Voronoi diagram of the image; the second line shows the visualization results for the 5 neurons in the x-layer sorted by the weights $(v_i E_i, i = 1, 2, \ldots, 5)$ for the final prediction.

Given image I_m, for each neuron n in the x-layer and each pixel (i, j) in I_m, we denote $S_{i,j}^{n,m} \triangleq P(\text{Pixel}_{i,j}^m|\text{Neuron}_n) = \frac{C_{i,j}^{n,m}}{\sum_{(i,j)\in I} C_{i,j}^{n,m}}$, where $C_{i,j}^{n,m}$ is the c-MWP generated by ExcitationBP for pixel (i, j) in I_m with neuron n in x-layer, (i, j) is the coordinate of the pixel. For the CUB dataset, the given part label $(p = 1, ..., 15)$ of each image is just one pixel in the middle of the part, and there is no extra information about the shape and the size of the part regions. For the p-th part label of image I_m, we denote (i_p, j_p) as its pixel location. The **pixel level probability** is defined as $S_{i_p,j_p}^{n,m} = P(\text{Pixel}_{i_p,j_p}^m|\text{Neuron}_n)$. Figure 3.7(a) shows the probability $S_{i_p,j_p}^{n,m}$ for each neuron $(n = 1, \ldots, 5)$ at the pixel locations of the part labels $(p = 1, \ldots, 15)$ for the example image shown in Figure 3.6. From Figure 3.7(a) we observe that the probability $S_{i_p,j_p}^{n,m}$ is reasonable when capturing small parts like *beak*, but is not on larger parts like *wing*, for the part label is just one pixel in the middle of the *wing*, while the x-features mainly focus on the edges (Fig. 3.6 shows a simple example).

Thus, we utilize the Voronoi diagram to partition the bounding box into 15 regions in which the nearest neighbor part annotation in each region would be the same,

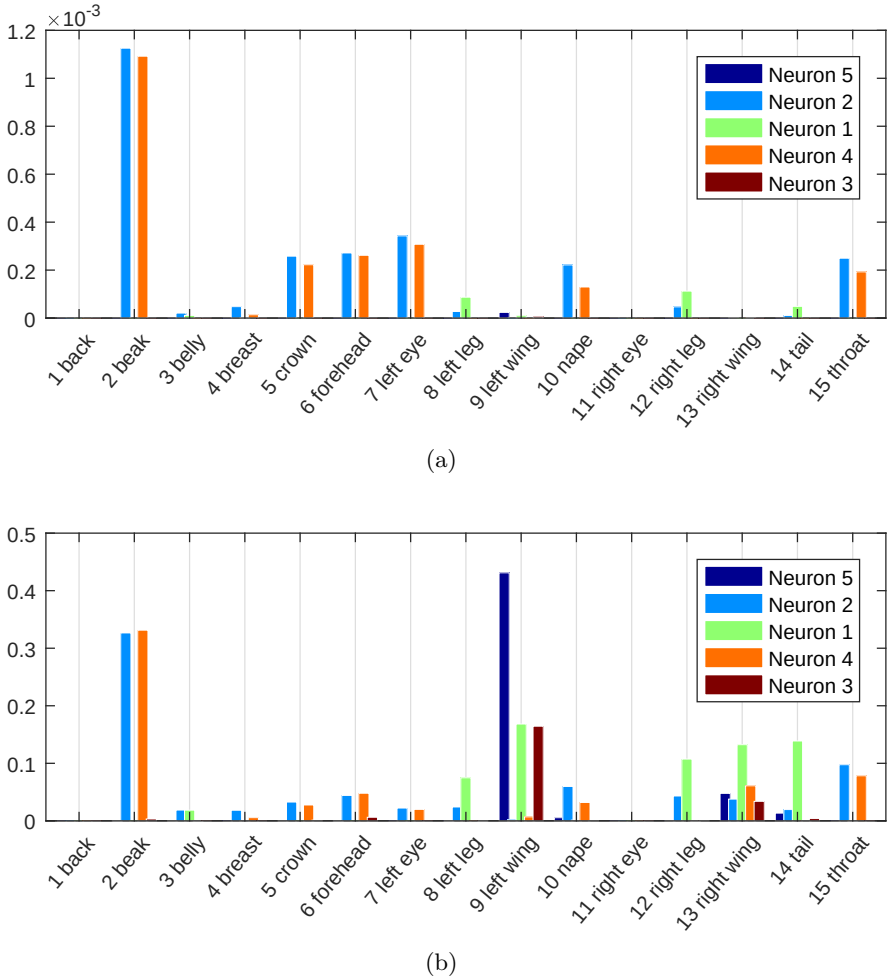


Figure 3.7: (a) Pixel-level probability $S^{n,m}_{i_p,j_p}$; (b) Voronoi-based probability $S^{n,m}_p$ for the example image in Figure 3.6.

and then compute the **Voronoi-based probability** $S^{n,m}_p \triangleq P(\text{Part}^m_p|\text{Neuron}_n) = \sum_{(i,j)\in I_m} P(\text{Part}^m_p|\text{Pixel}^m_{i,j})P(\text{Pixel}^m_{i,j}|\text{Neuron}_n)$. The Voronoi diagram is used instead of a segmentation because firstly we do not have segmentation ground truth and do not wish to include additional errors from an arbitrary segmentation algorithm, and secondly, because some of the heatmap activations fall slightly outside the object and we still want to capture those. However, the larger parts such as *wing* and *tail* always obtain much higher scores than the smaller parts such as *beak* and *eye* do; and there are also many

background pixels far from the center contained in the Voronoi diagram. To solve these issues, we introduce the inverse distance as a factor when computing the **Voronoi-based probability** $S_p^{n,m}$ in Algorithm 1 in the supplementary material, trying to keep the balance between the large part region and the small part region.

Figure 3.7(b) shows the probability $S_p^{n,m}$ for each neuron and each part label for the same example image in Figure 3.6. From Figure 3.7(b) one can also see evidence that the probabilities on *wing* and *tail* of some neurons are higher, indicating the metric based on the Voronoi diagram enhances the evaluation on these larger parts. For all the c-MWP outside of the ground truth bounding box, we introduce a 16-th part called *context*, which indicates that the x-feature is using the context to classify rather than the object features. For the Places dataset, since we have the exact object regions for different object labels of each image, we compute the probability $S_p^{n,m} \triangleq P(\text{Object}_p^m|\text{Neuron}_n) = \sum_{(i,j)\in\text{Object}_p^m} P(\text{Pixel}_{i,j}^m|\text{Neuron}_n)$. For each x-feature n we have a histogram $\mathbf{S}_n$ whose element is $\bar{S}_p^n = \frac{1}{M}\sum_m S_p^{n,m}$.

We propose several metrics to evaluate the performance of the explanation network without a human. This includes:

1. **Faithfulness**: We introduce a regression metric and a classification metric for faithfulness. (a) $F_{reg} = \frac{1}{M}\sum_m L(\bar{y}^{(m)} - \hat{y}^{(m)}) = \frac{1}{M}\sum_m |\bar{y}^{(m)} - \hat{y}^{(m)}|$, the mean absolute loss between $\hat{y}^{(m)}$ and its approximation $\bar{y}^{(m)}$; (b) We replace $\hat{y}^{(m)}$ with $\bar{y}^{(m)}$ in the original multi-class prediction vector $\hat{\mathbf{y}}^{(m)}$ before softmax and check whether the classification result changes. We denote c_r as the number of examples whose classification results remain the same, then $F_{cls} = \frac{c_r}{M}$.

2. **Orthogonality**: In order to measure whether different attention maps fall on the same region, we directly treat attention maps of different x-features as different vectors and compute their covariance matrix. We denote $\mathbf{C}$ as the covariance matrix among x-features aggregated over the dataset. Then $\mathbf{P} = \text{diag}(\mathbf{C})^{-1/2}\mathbf{C}\text{diag}(\mathbf{C})^{-1/2}$ is the matrix of correlation coefficients. The orthogonality between neurons in the x-layer is defined as: (a) $O_1 = ||\mathbf{P}||_F - \sqrt{n}$, where $||\cdot||_F$ is the Frobenius norm for matrix; (b) $O_2 = -\text{logdet}(\mathbf{P})$, where logdet is the logarithm of determinant of a matrix. Both O_1 and O_2 obtain the optimum at 0, when $\mathbf{P}$ is a unit matrix.

3. **Locality**: In order to measure locality, we propose a metric which associates x-features with various parts (CUB) or objects (Places), and measures how well they

associate with these parts or objects. The locality for each x-feature is defined as the entropy: $H_n = -\sum_p \left(\frac{\bar{S}_p^n}{\sum_p \bar{S}_p^n} \cdot \log(\frac{\bar{S}_p^n}{\sum_p \bar{S}_p^n}) \right)$. Locality is roughly measuring the log of the number of parts or objects captured by each x-feature.

Note that the current locality metric would not accurately represent features that do not represent a single part or object, it merely reflects our current best efforts in quantitatively measuring different explanations and especially only applicable on these limited datasets where the classification may likely to be explained by parts. Even on these datasets, we suffer from many annotated parts/objects being spatially close and semantically similar. For example, in CUB, there are 6 parts located on the small region of the head of each bird, saliency maps usually tend to capture all these parts together hence it is very difficult to obtain a smaller number of parts on each x-feature. In places, there are a total of 173 object categories with e.g. around 15 different categories for various objects on a bed. Hence the locality should be interpreted more in a relative sense to compare different approaches, the absolute number of parts captured is not exactly indicative of the performance of the approach.

3.3.3 Quantitative Results

Table 3.1: The average faithfulness, orthogonality, and locality of different approaches over all the 200 categories of the CUB dataset. The column **Z** represents the average locality computed over all the dimensions of **Z**, the $4,096$-dimensional first fully-connected layer of the deep network. This is obtained by separately running ExcitationBP on each dimension of **Z** and evaluating the resulting heatmaps. 1-Heatmap refers to the heatmap from $\hat{y}$.

Method		SRAE	NN	SAE	Lasso	CAE	**Z**	1-Heatmap
F_{reg}	Train	0.081	0.070	0.097	3.579	4.151	—	—
	Test	0.166	0.130	0.198	3.793	4.002	—	—
F_{cls}	Train	99.9%	100%	99.9%	73.1%	65.3%	—	—
	Test	99.9%	100%	99.9%	71.5%	69.3%	—	—
O1		**0.655**	0.976	0.879	1.2052	**0.630**	—	—
O2		**2.431**	4.911	3.506	3.985	**2.388**	—	—
Locality		**1.971**	2.436	2.200	2.108	2.123	**1.969**	2.566

The experiment setup for both the following sections is explained in the supplementary material. We compare among the proposed SRAE with plain NN (no reconstruction), SAE (a normal sparse autoencoder with no sparse reconstruction term), CAE (no faithfulness) and LASSO (only feature selection from Z), as well as locality of the baseline 1-Heatmap.

3.3.3.1 Results on CUB

In Table 3.1, we summarize the results for different explanation embedding approaches with different parameters for all the 200 categories of the CUB dataset. Results show that we can achieve excellent faithfulness to the predictions when using SRAE, NN, and SAE. The F_{reg} in both training and testing are less than 0.2. Since $\hat{\mathbf{y}}$ before softmax usually has a range in $[0, 50]$ and especially large in the positive examples, we consider the regression loss to be small. The classification faithfulness F_{cls} is even better, as only $1 - 2$ examples out of all the categories we tested have switched labels after replacing the original $\hat{\mathbf{y}}$ with the approximation from the x-features. This shows the major advantage of the XNN approach in that it indeed very faithfully explained the deep network. From Table 3.1 we observe that the faithfulness for Lasso are all very bad with different parameters, indicating that it is almost impossible for the feature selection method to select few X-features form $\mathbf{Z}$ directly to make the prediction faithful.

In terms of orthogonality and locality, our algorithm showed significant improvements over NN, SAE, and Lasso ($\alpha = 2.5$ in Table 3.1). The orthogonality of CAE is better than that of the proposed SRAE, which is reasonable because the features in $\mathbf{E}(\mathbf{Z})$ are definitely more orthogonal when there is only reconstruction loss in the optimization and no attempt to achieve faithfulness. The locality of CAE is slightly worse than SRAE, but the most important problem is that it is very difficult for CAE to achieve faithfulness to the original predictions because of the lack of the faithfulness loss in the optimization. Besides, the locality of SRAE improves significantly over the ones from 1-Heatmap, indicating that we are capable of separating information that come from different parts. The average locality of the x-features generated by SRAE is almost matching the average locality of features in $\mathbf{Z}$. This means we are close to the limit of part separation on this layer: many of the features on the $\mathbf{Z}$ layer already represent multiple parts.

We also show some qualitative examples between heatmaps on output $\hat{y}$ (1-Heatmap) and on x-features (XNN) from different categories in Figure 3.8. The weight above the

feature is $v_i E_i$, the product of the weight of the x-feature in the approximation of $\hat{\mathbf{y}}^{(k)}$ timed by the activation of the x-feature, which shows the contribution of the x-feature to the final prediction. One can see x-features nicely separate different discriminative aspects of the bird while 1-Heatmap sometimes focuses only on one part and miss others, and sometimes produces a heatmap that incorporates many parts simultaneously. Also, each x-feature seems distinct enough as a concept.

3.3.3.2 Results on Places

Table 3.2: The average faithfulness, orthogonality, and locality of different approaches for 10 categories of the Places dataset.

Method		SRAE	NN	SAE	Lasso	CAE	1-Heatmap
F_{reg}	Train	0.5527	0.3346	1.4768	4.0726	4.3579	—
	Test	1.0260	0.8736	1.5505	4.3366	4.6553	—
F_{cls}	Train	97.22%	97.17%	94.59%	90.19%	90.11%	—
	Test	94.79%	94.86%	93.29%	88.55%	88.42%	—
O1		**0.2252**	0.3472	0.4578	0.4729	0.2741	—
O2		**0.5617**	0.8852	1.0799	0.9194	0.5945	—
Locality		**2.7208**	2.7756	2.7819	2.7282	2.7627	2.7591

In Table 3.2, we summarize the results for different explanation embedding approaches with different parameters for 10 categories of the Places dataset, including all the categories that represent a room and have enough images with object labels in ADE20K. The results on the Places dataset in Table 3.2 show that our proposed method SRAE works well in explaining the scene recognition results for the Places dataset. The locality of the proposed method on Places is not as low as that on CUB, because in Places there are hundreds of different objects labels contained in ADE20K.

Figure 3.9 shows some qualitative examples between heatmaps on output $\hat{y}$ (1-Heatmap) and on x-features (XNN) for the Places dataset. The weight above the feature is $v_i E_i$, the product of the weight of the x-feature in the approximation of $\hat{\mathbf{y}}$ timed by the activation of the x-feature, which shows the contribution of the x-feature to the final

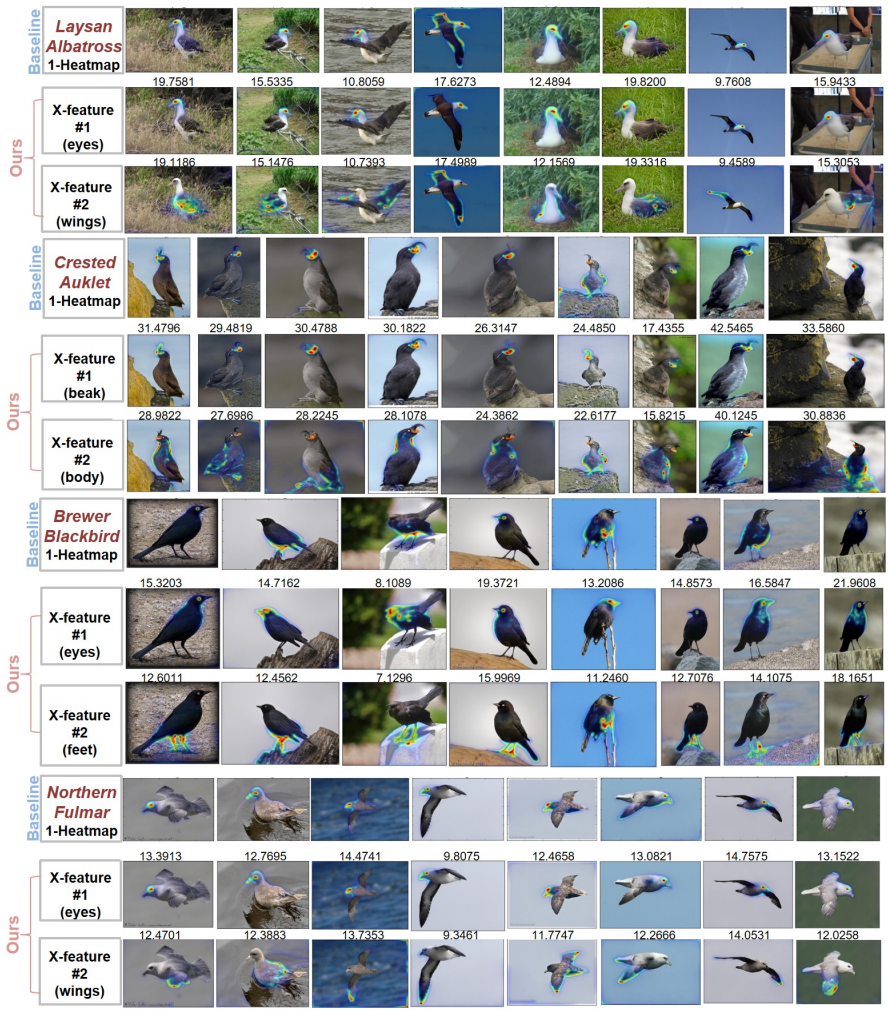


Figure 3.8: Comparison of heatmaps on the original output $\hat{\mathbf{y}}$ (baseline: 1-Heatmap) and on x-features (our approach XNN) for CUB. The weight above the feature is $v_i E_i$, the product of the weight of the x-feature in the approximation of $\hat{\mathbf{y}}^{(k)}$ timed by the activation of the x-feature, which shows the contribution of the x-feature to the final prediction.

prediction.

From Figure 3.9 we observe that each x-feature seems distinct enough as either a specific or a general concept. For example, when predicting the category of *kitchen*,

Figure 3.9: Comparison of heatmaps on the original output $\hat{y}$ (baseline: 1-Heatmap) and on x-features (our approach XNN) for *Kitchen* and *Bedroom* examples from Places dataset.

x-feature #1 captures cupboards and kitchenware, and x-feature #2 always focuses on lamps; while 1-Heatmap focuses on different objects in different images. When predicting the category of *bedroom*, x-feature #1 always captures bed, and x-feature #2 focuses on a general concept of rectangles; while 1-Heatmap still focuses on mixed concepts. The x-features surprised us as authors in both categories. In bedroom we expected bed, but did not expect that the *rectangular corners* in picture frames and window frames would help the classification. In the *kitchen*, we expected the stove being a feature, but did not expect that the CNN was actually using *lighting* as a feature. These surprises showed the additional insights human can gain of deep networks by running XNN instead of training on known concepts from language. More qualitive examples on the Places dataset can be found in the supplementary material.

From the results on both CUB and Places, we believe that the x-features learned by our proposed model indeed provide concise conceptual explanations on the decisions made by CNN algorithms.

3.3.4 XNN on convolutional layers

Table 3.3: The faithfulness for the fully convolutional XNN on different convolutional blocks for the CUB dataset.

Method		pool1	pool2	pool3	pool4	pool5
F_{reg}	Train	2.627	3.530	4.067	2.605	2.766
	Test	10.883	9.164	8.938	5.849	3.954

We also performed experiments on convolutional layers to evaluate the performance of a fully convolutional XNN on the CUB dataset and the MNIST dataset. For CUB, the fine-tuned VGG19 model is used as the original prediction model. The explanation XNN is attached on different convolutional blocks (from pooling layer 1 to pooling layer 5), which contains 3 encode convolutional layers, 3 decode convolutional layers, and a fully connected layer. For MNIST, the original prediction model is a simple 2 layers CNN whose accuracy is 98.70% on testing set of MNIST. The explanation XNN is attached on the last pooling layer of the prediction model, which contains an encode convolutional layer, a decode convolutional layer, a maxpooling layer, and a fully connected layer. For the convolutional layers in XNN, the filter size is 1×1. The number of the x-features is 5.

Table 3.3 shows the faithfulness F_{reg} for the convolutional XNNs from pooling layer 1 to pooling layer 5 on the CUB dataset. From Table 3.3 we observe that the faithfulness of the convolutional layers is always significantly worse than that of the fully connected layer. The closer it gets to the fully connected layers, the better the faithfulness is. And it is more likely to be overfitting for the bottom convolutional layers. Hence, we conclude that it is difficult to only utilize the features from the convolutional layers to build a fully convolutional XNN which is faithful to the prediction of the original DNN. Without the faithfulness, XNN is no longer an explanation of the original DNN.

On MNIST, for training set, $F_{reg} = 1.1952$; for testing set, $F_{reg} = 2.0422$ for the convolutional XNN on the last pooling layer. The faithfulness on MNIST is a little better than that on CUB, because the prediction model on MNIST is much simpler than that on CUB. Figure 3.10-3.11 shows some experimental results for the fully convolutional XNN on the MNIST dataset. Although the faithfulness on MNIST is still not as good as that of the fully connected XNN, we can find some interesting results from the x-features

of the fully convolutional XNN. Figure 3.10 shows the visualizations of the x-features for some positive examples when explaining the category of number 4; Figure 3.11 shows the visualizations of the x-features for some negative examples. From Figure 3.10 we can see that the most important feature for identifying number 4 is two vertical lines (X3); and the second important feature is one horizontal line in the middle (X5). From Figure 3.11 we can see that the most irrelevant features for category 4 are horizontal lines on the top and on the bottom.

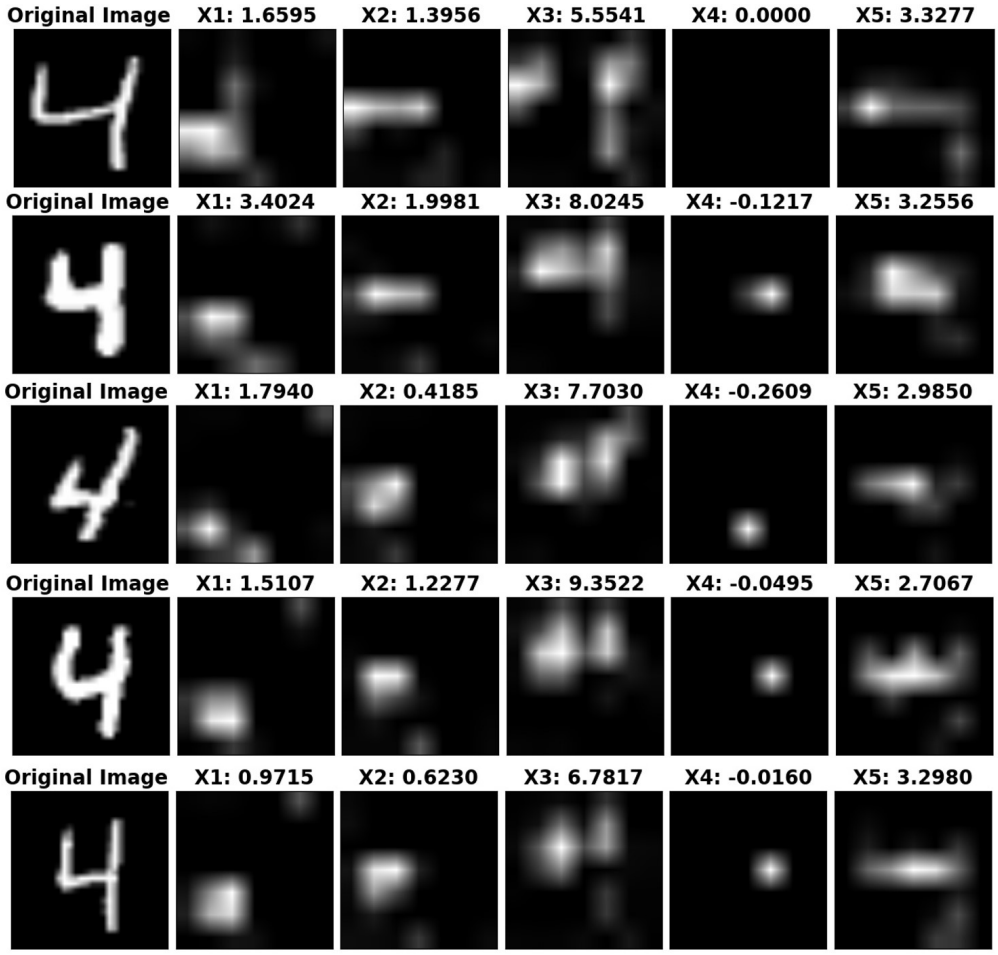


Figure 3.10: The visualizations of the x-features for some positive examples when explaining the category of 4.

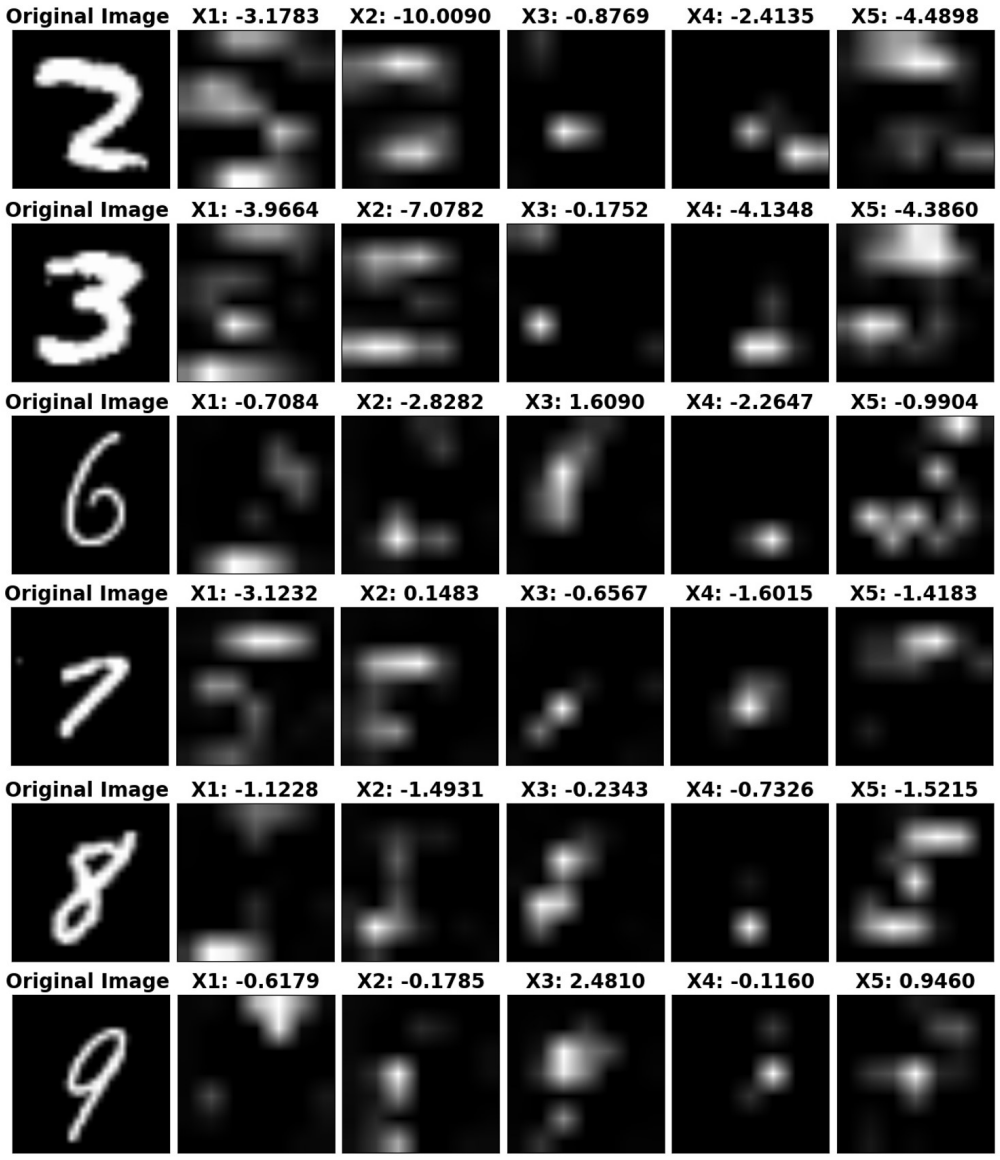


Figure 3.11: The visualizations of the x-features for some negative examples when explaining the category of 4.

3.4 Summary

In this chapter, we proposed an explanation network, that can be attached to any layer in a deep network to compress the layer into several concepts that can approximate an N-dimensional prediction output from the network. A sparse reconstruction autoencoder (SRAE) is proposed to avoid degeneracy and improve orthogonality. A human evaluation is conducted to investigate the performance of our approach against a baseline. We also proposed automatic evaluation metrics to evaluate the explanations on a fine-grained bird classification dataset and a scene recognition dataset. Quantitative and qualitative results show that the network can be extremely faithful to the original prediction network, and can indeed extract high-level concepts from a CNN that make sense to human, and different from existing concepts. We performed experiments training XNN on convolutional layers, which indicated some difficulty in faithfully explaining the DNN prediction from early convolutional layers.

Chapter 4: Stochastic Block ADMM for Training Deep Networks

4.1 Introduction

Deep Neural Networks (DNNs) are highly non-convex functions with ill-conditioned Hessians and are believed to have multiple local minima and saddle points. This makes training them very challenging [18]. Optimization methods for training DNNs have been an active line of research; commonly, Stochastic Gradient Descent (*SGD*) and its adaptive learning rate variants *e.g., Adam* [54] are used to optimize the DNNs by the back-propagation algorithm. Although these approaches have been the most successful, there are drawbacks related to SGD training in DNNs [98], such as vanishing gradients in deep layers, a significant memory footprint for storing the gradients, and difficulty to parallelize across layers because backpropagation has to be done sequentially.

Alternating Direction Method of Multipliers (ADMM) is an approach that decouples optimization variables and optimize the augmented Lagrangian in a primal-dual scheme. It is known to be a simple yet powerful algorithm to solve convex problems with linear convergence in a distributed and parallel manner [7]. It also has shown promise in solving certain families of non-convex problems [106, 44].

Recently, optimization of the neural networks with such alternating direction techniques has gained rising attention [118, 119, 127, 38, 3, 31] which would potentially avoid the disadvantages of the SGD and introduce beneficial properties such as *fast(er)* convergence, ease of *parallelization* and *distributed* training, and being able to enforce additional constraints on the DNN parameter tensors.

Despite their advantages, there are several reasons ADMM-like methods are not widely used in DNN training. The performance of these methods is usually not as good as SGD, the algorithms are usually batch mode which directly restricts the amount of trainable parameters and training data, updates are in closed-from which prohibits the use of complicated architectures while being memory intensive, *etc.* Work of [98] is of this kind which, despite the parallelization capabilities introduced by ADMM, the size of the training data is linearly limited by the number of cores. [36] proposed to split

DNN into blocks using gluing variables similar to ADMM, but they did not utilize dual variables as in ADMM.

In this chapter, we propose stochastic block ADMM which addresses many of the aforementioned issues. Stochastic block ADMM separates DNN parameters into many blocks, and uses stochastic gradients to update each block. We provide a convergence proof for the proposed approach, and verify its performance on several deep learning benchmarks.

An ADMM formulation of deep learning also allows us to add additional constraints to the learning problem in a heterogeneous setting where back propagation cannot be applied. In this chapter, we explore adding non-negative factorization of the intermediate layers for the goal of feature disentanglement. Non-negative Matrix Factorization (NMF) has been able to generate sparse and interpretable representation due to the non-negative constraints over the factorization matrices [61]. [16] proposed applying NMF over different activations of CNN for object localization. This would support the belief that the CNNs would learn semantic part-of-object filters during training [34, 5]. Jointly training an NMF decomposition with deep learning is highly non-convex and cannot be addressed by the conventional backpropagation and SGD algorithms. We show results training these networks via ADMM and their performance on a supervised feature disentanglement benchmark.

In summary, this chapter makes the following contributions:

- We propose stochastic block ADMM for training deep networks. Experiments show our algorithm outperforms previous attempts of using ADMM in deep network training.
- We prove the local convergence of the proposed stochastic block ADMM algorithm.
- We propose DeepFacto, which jointly trains a non-negative matrix factorization layer with a deep network using ADMM, and show its capability in supervised feature disentanglement.

4.2 Stochastic Block ADMM

4.2.1 Training DNN using ADMM

Alternating Direction Method of Multipliers (ADMM) [30, 7] is a class of optimization methods belonging to *operator splitting techniques* which borrows benefits from both dual decomposition and augmented Lagrangian methods for constrained optimization. To show the potentials of standard ADMM, we first revisit a general formulation of ADMM in DNN training, similar to those used in prior work, then we propose our stochastic block ADMM in the next subsection.

To formulate training an L-layer DNN in a general supervised setting, we would have the following non-convex constrained optimization problem [118]:

$$\begin{aligned} \underset{\mathcal{W},\mathcal{A},\mathcal{Z}}{\text{minimize}} \quad & \mathcal{J}\left(\boldsymbol{Y}, \boldsymbol{Z}_L\right) + \sum_{\ell=1}^{L} \lambda_\ell \mathbf{r}_\ell(\boldsymbol{W}_\ell) \\ \text{subject to} \quad & \boldsymbol{A}_\ell - \boldsymbol{\phi}_\ell\left(\boldsymbol{Z}_\ell\right) = \mathbf{0}, \quad \ell = 1, \ldots, L-1 \\ \text{subject to} \quad & \boldsymbol{Z}_\ell - \boldsymbol{W}_\ell \boldsymbol{A}_{\ell-1} = \mathbf{0}, \quad \ell = 1, \ldots, L \end{aligned} \tag{4.1}$$

where $\mathcal{J}$ is the main objective (*e.g.*, cross-entropy, mean-squared-error loss functions) that needs to be minimized. The subscript ℓ denotes the ℓ-th layer in the network. The optimization variables are $\mathcal{W} = \{\boldsymbol{W}_\ell\}_{\ell=1}^{L}$, $\mathcal{A} = \{\boldsymbol{A}_\ell\}_{\ell=1}^{L-1}$, and $\mathcal{Z} = \{\boldsymbol{Z}_\ell\}_{\ell=1}^{L}$ where $\boldsymbol{W}_\ell$, $\boldsymbol{Z}_\ell$, $\boldsymbol{A}_\ell$, and $\boldsymbol{\phi}_\ell(.)$ are the weight matrix, output matrix, activation matrix, and the activation function (*e.g.*, ReLU) at the ℓ-th layer, respectively. Note that $\boldsymbol{A}_0 = \boldsymbol{X}$ where $\boldsymbol{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\} \in \mathbb{R}^{M \times N}$ is the input data matrix containing N samples with input dimensionality M; $\boldsymbol{Y} = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N\} \in \mathbb{R}^{C \times N}$ is the target matrix pair comprised of N one-hot vector label of dimension C, representing number of prediction classes. Also, $\mathbf{r}(.)$ is the regularization term with (*e.g.*, Frobenius norm $\|.\|_F^2$) corresponding penalty weight λ_ℓ. Note that the regularization term can be simply ignored by setting λ_ℓ to zero. In this formulation, the intercept in each layer is ignored for simplicity as it can be simply be added by slightly modifying the $\boldsymbol{W}_\ell$ and the input to each layer. The formulation in Eq. (4.1) breaks the the conventional multi-layer backpropagation optimization of DNNs into simpler sub-problems that can be solved efficiently (e.g. reducing to least-squares

Algorithm 1 ADMM for DNN Training

Input: data $\boldsymbol{X}$, labels $\boldsymbol{Y}$
Params: $\beta_\ell > 0, \gamma_\ell > 0, \lambda_\ell > 0$
Initialize: $\{\boldsymbol{W}_\ell^0\}_{\ell=1}^L, \{\boldsymbol{U}_\ell^0\}_{\ell=1}^L, \{\boldsymbol{V}_\ell^0\}_{\ell=1}^{L-1}, \{\boldsymbol{Z}_\ell^0\}_{\ell=1}^L, \{\boldsymbol{A}_\ell^0\}_{\ell=1}^{L-1}$ $k \leftarrow 0$
repeat
  for $\ell = 1$ **to** L **do**
    $\boldsymbol{W}_\ell^{k+1} \leftarrow \arg\min \{\mathcal{P}_\ell(.) + \lambda_\ell \mathbf{r}_\ell(\boldsymbol{W}_\ell^k)\}$
  end for
  for $\ell = 1$ **to** $L-1$ **do**
    $\boldsymbol{Z}_\ell^{k+1} \leftarrow \arg\min \{\mathcal{P}_\ell(.) + \mathcal{Q}_\ell(.)\}$
    $\boldsymbol{A}_\ell^{k+1} \leftarrow \arg\min \{\mathcal{P}_{\ell+1}(.) + \mathcal{Q}_\ell(.)\}$
  end for
  $\boldsymbol{Z}_L^{k+1} \leftarrow \arg\min \{\mathcal{J}\left(\boldsymbol{Y}, \boldsymbol{Z}_L^k\right) + \mathcal{P}_L(.)\}$
  for $\ell = 1$ **to** $L-1$ **do**
    $\boldsymbol{U}_\ell^{k+1} \leftarrow \boldsymbol{U}_\ell^k + \boldsymbol{Z}_\ell^{k+1} - \boldsymbol{W}_\ell^{k+1}\boldsymbol{A}_{\ell-1}^{k+1}$
    $\boldsymbol{V}_\ell^{k+1} \leftarrow \boldsymbol{V}_\ell^k + \boldsymbol{A}_\ell^{k+1} - \boldsymbol{\phi}_\ell(\boldsymbol{Z}_\ell^{k+1})$
  end for
  $\boldsymbol{U}_L^{k+1} \leftarrow \boldsymbol{U}_L^k + \boldsymbol{Z}_L^{k+1} - \boldsymbol{W}_L^{k+1}\boldsymbol{A}_{L-1}^{k+1}$
until some stopping criterion is reached.

problem). This also facilitates training in a distributed manner – as the layers of the DNN are decoupled and the variables can be updated in parallel across layers ($\boldsymbol{W}_\ell$) and data points ($\boldsymbol{W}_\ell, \boldsymbol{Z}_\ell, \boldsymbol{A}_\ell$).

To enforce the constraints in problem (4.1) and solve the optimization using ADMM, we would have the following augmented Lagrangian problem:

$$\begin{aligned} \underset{\mathcal{W},\mathcal{A},\mathcal{Z}}{\text{minimize}} \quad & \mathcal{J}\left(\boldsymbol{Y}, \boldsymbol{Z}_L\right) + \sum_{\ell=1}^{L} \lambda_\ell \mathbf{r}_\ell(\boldsymbol{W}_\ell) \\ & + \sum_{\ell=1}^{L} \frac{\beta_\ell}{2} \|\boldsymbol{Z}_\ell - \boldsymbol{W}_\ell \boldsymbol{A}_{\ell-1} + \boldsymbol{U}_\ell\|_F^2 \\ & + \sum_{\ell=1}^{L-1} \frac{\gamma_\ell}{2} \|\boldsymbol{A}_\ell - \boldsymbol{\phi}_\ell(\boldsymbol{Z}_\ell) + \boldsymbol{V}_\ell\|_F^2 \end{aligned} \tag{4.2}$$

where $\beta_\ell, \gamma_\ell > 0$ are the step sizes, $\boldsymbol{U}_\ell$ and $\boldsymbol{V}_\ell$ are the *(scaled) dual variables* [7] for the

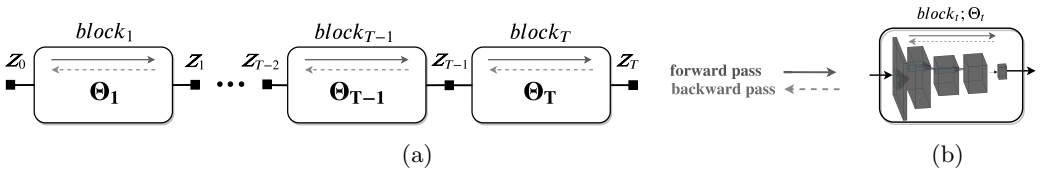


Figure 4.1: a) General Architecture for training DNNs proposed in Stochastic Block-ADMM. b) A few differential layers selected from a parent network are stacked inside a block. The parameters Θ_t are updated by SGD in a forward-backward pass.

equality constraint at the layer ℓ. Algorithm 1 shows a standard ADMM scheme for optimizing eq. (4.2). Note, the parameters are updated in a closed-form as analytical solution can be simply derived. For simplicity of the equations, we denote $\mathcal{P}_\ell(.) = \frac{\beta_\ell}{2}\|\boldsymbol{Z}_\ell - \boldsymbol{W}_\ell \boldsymbol{A}_{\ell-1} + \boldsymbol{U}_\ell\|_F^2$ and $\mathcal{Q}_\ell(.) = \frac{\gamma_\ell}{2}\|\boldsymbol{A}_\ell - \boldsymbol{\phi}_\ell(\boldsymbol{Z}_\ell) + \boldsymbol{V}_\ell\|_F^2$. This algorithm is similar to [98, 105] with the difference that all the equality constraints in problem (4.1) are enforced using multipliers, while previous work only enforced the constraints on the last layer L while other constraints were only loosely enforced using quadratic penalty.

While the standard ADMM Algorithm 1 has potentials in training (simple) DNNs [98], there exists hurdles that confines extending ADMM to more complex problems — the global convergence proof of the ADMM [19] assumes that $\mathcal{J}$ is deterministic and the global solution is calculated at each iteration of the cyclic parameter updates. This makes standard ADMM computationally expensive thus impractical for training of many large-scale optimization problems. Specifically, for deep learning, this would impose a severe restriction on training set size when limited computational resources are available. In addition, since the variable updates in standard ADMM are analytically driven, the extent of its applications is limit to trivial tasks [98], making it incompetent to perform on par with the recent complex architectures introduced in deep learning (e.g. [40]).

4.2.2 Stochastic Block-ADMM

In this section, we introduce a novel variant of ADMM for training DNNs, the stochastic Block-ADMM. We first split the conventional multi-layer network architectures into arbitrary number of *blocks*, each containing only a part of the network. To make each block's parameters independent from its neighbors, *decoupling variables* $\{\boldsymbol{Z}_t,\ t = 1, \ldots, T\}$

are introduced as shown in Fig. 4.1(a). These variables pass the information forward and backward in the architecture to train blocks in a cyclic manner until consensus is reached. Each $block_t$ consists of one or multiple differentiable layers (e.g., convolutional layers, activation layers, etc.) that are detached from the rest of the network via coupling variables. Denote the set of all learnable parameters of each $block_t$ as Θ_t. As an example, a $block_t$ wrapping multiple layers can be seen in Figure 4.1(b). Our formulation is:

$$\begin{aligned} &\underset{\boldsymbol{\Theta},\mathcal{Z}}{\text{minimize}}\ \mathcal{J}\left(\boldsymbol{Y},\boldsymbol{Z}_T\right) \\ &\text{subject to } \boldsymbol{Z}_t = \text{block}_{\boldsymbol{\Theta}_t}(\boldsymbol{Z}_{t-1}), \quad \boldsymbol{Z}_0 = \boldsymbol{X} \end{aligned} \tag{4.3}$$

where $\boldsymbol{\Theta} = \{\Theta_t\}_{t=1}^{T}$and $\mathcal{Z} = \{\boldsymbol{Z}_t\}_{t=1}^{T}$. $\mathcal{J}$ is the desired cost to be minimized, and T is the total number of blocks.

To train this new approach of training of DNNs, similar to problem (4.2), we would have the following augmented Lagrangian minimization problem to enforce the equality constraints needed for training,

$$\begin{aligned} &\min_{\boldsymbol{\Theta},\mathcal{Z}}\ \mathcal{J}\left(\boldsymbol{Y},\boldsymbol{Z}_T\right) + \sum_{t=1}^{T} \frac{\beta_t}{2} \|\boldsymbol{Z}_t - \text{block}_{\Theta_t}(\boldsymbol{Z}_{t-1}) + \boldsymbol{U}_t\|_F^2 \\ &\text{subject to} \quad \boldsymbol{Z}_0 = \boldsymbol{X} \end{aligned} \tag{4.4}$$

where β_t and $\boldsymbol{U}_t$ are the (scaled) step size and the Lagrange multiplier corresponding to the t-th Block, respectively. Our proposed Stochastic Block-ADMM method for training problem (4.4) is presented in Algorithm 2. In Stochastic Block-ADMM, parameters of the t-th block, Θ_t are updated using the *Stochastic Gradient Descent* optimizer or its adaptive learning rate variants (e.g. *Adam* [54]). We have found Adam to consistently outperform other counterparts, particularly in updating the decoupling variables $\boldsymbol{Z}_t$. ζ_t and η_t show the learning rate in each update step for $\boldsymbol{Z}_t$ and Θ_t, respectively. Similar to training conventional neural networks, each block is updated by first going in a forward pass through the block and update the parameters using back-propagation. Update of the block parameters Θ_t is done using mini-batches of data. The same goes for the decoupling variables $\boldsymbol{Z}_t$. Note, in each cycle of the parameter update in Algorithm 2, all

Algorithm 2 Stochastic Block-ADMM

Input: data $\boldsymbol{X}$, labels $\boldsymbol{Y}$
Params: $\beta_t > 0,\ \zeta_t > 0, \eta_t > 0$
Define: $\mathcal{T}(\boldsymbol{Z}_t, \boldsymbol{Z}_{t-1}, \boldsymbol{U}_t, \Theta_t) = \frac{\beta_t}{2}\|\boldsymbol{Z}_t - \text{block}_{\Theta_t}(\boldsymbol{Z}_{t-1}) + \boldsymbol{U}_t\|_F^2$
Initialize: $\{\Theta_t^0\}_{t=1}^T, \{\boldsymbol{U}_t^0\}_{t=1}^T,\ k \leftarrow 0$
Initialize: $\{\boldsymbol{Z}_t\}_{t=1}^T$ in a forward pass.
repeat
  $\boldsymbol{Z}_L^{k+1} \leftarrow \boldsymbol{Z}_L^k - \zeta_L \nabla_{\boldsymbol{Z}_L^k}(\mathcal{J}\left(\boldsymbol{Y}_i, \boldsymbol{Z}_L^k\right)$
  $+\mathcal{T}(\boldsymbol{Z}_L^k, \boldsymbol{Z}_{L-1}^k, \boldsymbol{U}_L^k, \Theta_L^k))$
  for $t = L-1$ **to** 1 **do**
    $\boldsymbol{Z}_t^{k+1} \leftarrow \boldsymbol{Z}_t^k - \zeta_t \nabla_{\boldsymbol{Z}_t^k}(\mathcal{T}(\boldsymbol{Z}_t^k, \boldsymbol{Z}_{t-1}^k, \boldsymbol{U}_t^k, \Theta_t^k)$
    $+\mathcal{T}(\boldsymbol{Z}_{t+1}^k, \boldsymbol{Z}_t^k, \boldsymbol{U}_{t+1}^k, \Theta_{t+1}^k))$
  end for
  for $t = 1$ **to** L **do**
    ${\Theta_t}^{k+1} \leftarrow {\Theta_t}^k - \eta_t \nabla_{\Theta_t} \mathcal{T}(\boldsymbol{Z}_{t,i}^{k+1}, \boldsymbol{Z}_{t-1,i}^{k+1}, \boldsymbol{U}_{t,i}^k, \Theta_t^k)$,
    draw $i \subset \{1, \ldots, N\}$
    $\boldsymbol{U}_t^{k+1} \leftarrow \boldsymbol{U}_t^k + \boldsymbol{Z}_t^k - \text{block}_{\Theta_t}^{k+1}(\boldsymbol{Z}_{t-1}^{k+1})$
  end for
until some stopping criterion is reached.

the samples of $\boldsymbol{Z}$ are updated, while this is not necessary for Θ_t. In addition, due to local update of the parameters at each iteration, one can perform each update in Algorithm 2 multiple times. However, we found one step update to be sufficient in our experiments.

In Algorithm 2, we take the reverse order for updating the decoupling variables $\boldsymbol{Z}_t$, which we have empirically found more efficient, as analogous to backpropagation where gradient flows backwards as well.

Similar to [118], our method can readily mitigate the long-known vanishing gradient problem by splitting a conventional DNN into arbitrary sized blocks. Note that during testing time, one could follow eq. (4.4) to solve an optimization problem. But in practice, it suffices to use a straight-through estimator by removing the decoupling variables and simply pass the output of each layer to the next, equivalent of doing a forward pass in a conventional DNN. We have taken this approach in our experiments as the error induced by ignoring the decoupling variables is negligible. It should be noted that update step for $\boldsymbol{Z}_\ell$ is dependent on the adjacent blocks, hence can only be parallelized across the data points. However, fixing $\boldsymbol{Z}$, all the block parameters Θ_t are independent of each other,

hence can be updated in parallel across blocks as well as data points.

4.2.3 Discussions on Convergence Properties

Let us consider the following shorthand representation of the deep network training problem:

$$\begin{aligned} &\underset{\mathcal{Z},\boldsymbol{\Theta}}{\text{minimize}}\ f(\mathcal{Z}) \\ &\text{subject to } h(\mathcal{Z},\boldsymbol{\Theta}) = \mathbf{0}. \end{aligned} \tag{4.5}$$

where $\mathcal{Z}$ and $\boldsymbol{\Theta}$ are as defined in Sec. 4.2.2, and $f(\cdot)$ represents the training objective, and $h(\cdot)$ represents the layer coupling equalities as in eq. (4.3). We also assume that both $f(\cdot)$ and $h(\cdot)$ are differentiable functions. Note that both f and h can be non-convex.

Let us consider the following augmented Lagrangian:

$$\mathcal{L}_{\rho_k}(\mathcal{Z},\boldsymbol{\Theta},\boldsymbol{\lambda}) = f(\mathcal{Z}) + \langle \boldsymbol{\lambda}, h(\mathcal{Z},\boldsymbol{\Theta})\rangle + \frac{1}{2\rho_k}\|h(\boldsymbol{Z},\boldsymbol{\Theta})\|_{\mathbf{2}}^{\mathbf{2}},$$

where $\boldsymbol{\lambda}$ collects all the dual variables $\boldsymbol{U}_1,\ldots,\boldsymbol{U}_T$ that correspond to different layers. The standard primal-dual updates can be summarized as follows:

$$(\boldsymbol{Z}^{k+1},\Theta^{k+1}) \leftarrow \arg\min_{\mathcal{Z},\boldsymbol{\Theta}} \mathcal{L}_{\rho_k}(\mathcal{Z},\boldsymbol{\Theta},\boldsymbol{\lambda}^{\mathbf{k}}), \tag{4.6a}$$

$$\boldsymbol{\lambda}^{k+1} \leftarrow \boldsymbol{\lambda}^{k} + \frac{1}{\rho_k} h(\boldsymbol{Z}^{k+1},\Theta^{k+1}), \tag{4.6b}$$

In our case, since the subproblem in eq. (4.6a) is nonconvex, exactly minimizing this function may not be possible. In the previous section, the primal update is carried out by stochastic optimization w.r.t. $\mathcal{Z}$ and $\boldsymbol{\Theta}$ in an alternating fashion—which is a computationally lightweight algorithm that converges to a stationary point of $\mathcal{L}_{\rho_k}(\mathcal{Z},\boldsymbol{\Theta},\boldsymbol{\lambda}^k)$ under certain conditions [6, 112]. The convergence of this type of primal-dual algorithm with inexact stochastic solution for the primal problem is unclear. In this work, we offer convergence support for our designed deep network training algorithm. Our idea follows

recent work in [87] that handles deterministic primal problems under nonconvex equality constraints.

To be specific, we employ the trick in [87] for adaptively adjusting the parameter ρ_k. We assume that ρ_k is adjusted by

$$\rho_{k+1} \leftarrow \begin{cases} \rho_k, & \|h(\boldsymbol{Z}^k, \boldsymbol{\Theta}^k)\| \leq \eta_k, \\ c\rho_k,\ 0 < c < 1, & \text{o.w.} \end{cases} \tag{4.7}$$

where η_k for $k = 1, 2, \ldots$ is a pre-specified sequence that bounds the equality-enforcing error.

Our analysis shows the following convergence result:

Proposition 1 *Assume $h(\mathcal{Z}, \boldsymbol{\Theta}) = \mathbf{0}$ satisfies the Robinson's condition. Also assume for each update in eq. (4.6a), the subproblem solution solved by stochastic alternating optimization satisfies*

$$\mathbb{E}\left[\left\|\mathcal{G}(\boldsymbol{x}^k)\right\|^2\right] \leq \varepsilon_k,\ \mathbb{V}\left[\mathcal{G}(\boldsymbol{x}^k)\right] \leq \sigma_k^2, \tag{4.8}$$

where $\boldsymbol{x} = (\mathcal{Z}, \boldsymbol{\Theta})$ is a vector that collects all the optimization variables and $\mathcal{G}(\boldsymbol{x}^k)$ collects the stochastic gradients that we used for updating $(\mathcal{Z}, \boldsymbol{\Theta})$. Assume that the stochastic gradient for the primal update is unbiased, i.e.,

$$\mathbb{E}[\mathcal{G}(\boldsymbol{x}^k)] = \nabla \mathcal{L}_{\rho_k}(\boldsymbol{x}_k),\ \forall k. \tag{4.9}$$

Then, every limit point of the solution sequence produced by the algorithm in eq. (4.6) converges to a KKT point of the problem in eq. (4.5), if $\eta_k \to 0$, $\sigma_k^2 \to 0$ and $\varepsilon_k \to 0$.

Proof: We follow the steps in the proof for similar problems in [28] and [87] with deterministic primal updates. Proper modifications are made to cover the stochastic primal update in our proof.

Note that we have

$$\nabla \mathcal{L}_{\rho_k}(\boldsymbol{X}^k) = \nabla f(\boldsymbol{X}^k) + \nabla h(\boldsymbol{X}^k)^T \boldsymbol{\mu}^k,$$

where

$$\boldsymbol{\mu}^k = (1/\rho_k)h(\boldsymbol{X}^k) + \boldsymbol{\lambda}^k.$$

Our first step is to show that $\{\boldsymbol{\mu}^k\}$ is a convergent sequence. To see this, we define

$$\bar{\boldsymbol{\mu}}^k = \frac{\boldsymbol{\mu}^k}{\|\boldsymbol{\mu}^k\|}.$$

Since $\bar{\boldsymbol{\mu}}^k$ is bounded, it converges to a limit point $\bar{\boldsymbol{\mu}}$. Also let $\boldsymbol{x}^\star$ be a limit point of $\boldsymbol{x}^k$. Because we have assumed that

$$\varepsilon_k \to 0, \quad \sigma_k^2 \to 0,$$

it means that the mean and variance of the stochastic gradient of our primal update goes to zero. Since our stochastic gradient is unbiased, we have

$$\mathcal{G}(\boldsymbol{X}^k) \to \nabla \mathcal{L}_{\rho_k}(\boldsymbol{X}^\star).$$

This also means that we must have $\mathcal{G}(\boldsymbol{x}^k) \to \mathbf{0}$ and

$$\nabla L_{\rho_k}(\boldsymbol{x}^k) \to \mathbf{0}.$$

Hence, the following holds when $k \to \infty$:

$$\nabla L_{\rho_k}(\boldsymbol{X}^\star) = \nabla f(\boldsymbol{X}^\star) + \nabla h(\boldsymbol{X}^\star)^T \boldsymbol{\mu}^\infty = 0, \tag{4.10}$$

Suppose $\boldsymbol{\mu}^k$ is unbounded. By dividing eq. (4.10) by the above $\|\boldsymbol{\mu}^k\|$ and considering $k \to \infty$, we must have

$$\nabla h(\boldsymbol{X}^\star)^T \bar{\boldsymbol{\mu}} = 0, \quad \forall \boldsymbol{X}. \tag{4.11}$$

The term $\nabla f(\boldsymbol{X}^\star)/\|\boldsymbol{\mu}\|$ is zero since we assumed $\bar{\boldsymbol{\mu}}$ is unbounded. Since $h(\boldsymbol{X}) = \mathbf{0}$ satisfies the Robinson's condition, then, for any $\boldsymbol{w}$, there exists $\beta > 0$ and $\boldsymbol{x}$ such that

$$\boldsymbol{w} = \beta \nabla h(\boldsymbol{X}^\star)(\boldsymbol{X} - \boldsymbol{X}^\star).$$

This together with eq. (4.11) says that $\bar{\boldsymbol{\mu}} = \mathbf{0}$. This contradicts to the fact $\|\bar{\boldsymbol{\mu}}\| = 1$.

Hence, $\{\boldsymbol{\mu}^k\}$ must be a bounded sequence and thus admits a limit point. Denote $\boldsymbol{\mu}^\star$ as this limit point, and take limit of both sides of eq. (4.10). We have:

$$\nabla f(\boldsymbol{X}^\star) + \nabla h(\boldsymbol{X}^\star)^T \boldsymbol{\mu}^\star = \mathbf{0}, \quad \forall \boldsymbol{X}. \tag{4.12}$$

In addition, since

$$\rho_k(\boldsymbol{\mu}^k - \boldsymbol{\lambda}^k) = h(\mathbf{X^k})$$

with $\rho_k \to 0$ or $\boldsymbol{\mu}_k - \boldsymbol{\lambda}_k \to 0$ (per our updating rule and $\eta_k \to 0$), the constraints will be enforced in the limit. $\square$

Proposition 1 asserts that the algorithm converges to a KKT point under some conditions. There are, however, a few caveats. First, the condition that ε_k and σ_k^2 converging to zero along the iterations is nontrivial to satisfy. The condition $\varepsilon_k \to 0$ means that the primal problem needs to be solved more and more accurately when k grows, in terms of approaching the stationary point of the subproblem using block stochastic gradient. This can be achieved via gradually increasing the number of iterations for the primal updates. Note that stochastic block gradient can provably attain $\mathbb{E}[\|\mathcal{G}(\boldsymbol{X}^k)\|^2] \leq \varepsilon_k$; see [112]. The condition $\sigma_k^2 \to 0$ means that the variance of the stochastic gradient needs to shrink when k increases. This can be achieved by increasing the batch size when k grows; see discussions in [112]. Hence, to satisfy both conditions in theory, the complexity for carrying out each iteration k may grow—which is not desired. Nonetheless, our empirical experience shows that using a fixed number of iterations for stochastic primal optimization and a fixed batch size in general does not hurt convergence. We hypothesize that the momentum may play an important role in increasing the effective batch size since with momentum gradients from previous batches are remembered and utilized in the subsequent steps. We leave further analysis with momentum to future research.

Second, the unbiasedness of the primal stochastic gradient [cf. eq. (4.9)] is not always easy to establish under block coordinate descent settings [112]. Nonetheless, this can be fixed via a simple randomization strategy among the blocks [29].

The third challenge is that the hyperparameter c and the sequence $\{\eta_k\}$ are not necessarily easy to select in some cases [87, 28]. These parameters control how quickly one should adjust the update settings to accommodate the current iteration, which varies from case to case. However, interestingly, we find that these hyperparameters are relatively

easy to tune in our case. In particular, our extensive experiments show that fixed ρ_k and η_k work reasonably well for our deep learning problems.

The slightly stringent conditions in Proposition 1 and the relatively 'benign' convergence behavior observed in practice pose a gap between theory and practice—and an interesting direction for future research.

4.3 Deepfacto: End-to-End Factorization of the DNN Activations

To show the power and flexibility of our proposed method in training heterogeneous networks, we will investigate the well established dimension reduction method, *Nonenegative Matrix Factorization (NMF)*, over the activations of the DNN in an end-to-end training scheme. NMF has shown to learn disentangled representation over the data [61, 16] and with that we aim to learn disentangled features in a deep network.

Figure 4.2 shows an *NMF module* with *rank* r incorporated between two arbitrary neighboring blocks. The output from the $block_t$ is factorized into $\boldsymbol{M}_t$ and $\boldsymbol{S}_t$, namely, the basis and score matrices. In this configuration, only the score matrix $\boldsymbol{S}_t$ is passed to the next blocks. The score matrix is low-rank, sparse and nonnegative hence can possibly represent features that are more disentangled than the original network. However, integrating such optimization problem nested in the deep neural network architecture is not trivial as conventional back-propagation would not be applicable in this heterogeneous problem with no gradient path from $\boldsymbol{S}_t$ to Z_t. We extend the ADMM framework (4.4) into having nonnegative factorization constraints over its activations and formulate the following optimization problem:

$$
\begin{aligned}
\min_{\boldsymbol{\Theta},\mathcal{Z},\boldsymbol{S},\boldsymbol{M}} \; & \mathcal{J}\left(\boldsymbol{Y}, \boldsymbol{Z}_T\right) \\
& + \sum_{k=1,k\neq t+1}^{T} \frac{\beta_k}{2} \|\boldsymbol{Z}_k - block_k(\boldsymbol{Z}_{k-1}) + \boldsymbol{U}_k\|_F^2 \\
& + \frac{\beta_{t+1}}{2} \|\boldsymbol{Z}_{t+1} - block_{t+1}(\boldsymbol{S}_t) + \boldsymbol{U}_{t+1}\|_F^2 \\
& + \frac{\gamma_t}{2} \|\boldsymbol{Z}_t - \boldsymbol{M}_t\boldsymbol{S}_t + \boldsymbol{V}_t\|_F^2 \\
& \forall i, j \; \boldsymbol{M}_{\ell,ij} \geq 0, \; \boldsymbol{S}_{\ell,ij} \geq 0
\end{aligned}
\tag{4.13}
$$

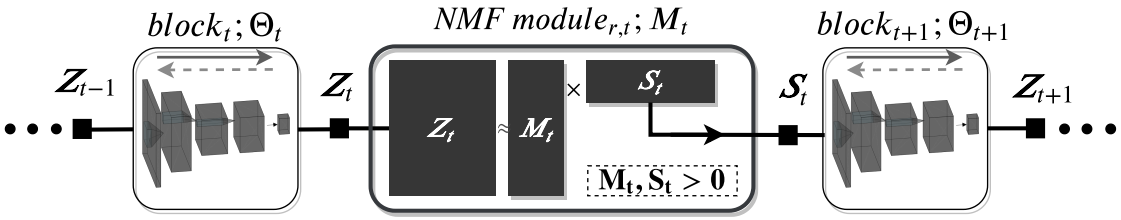


Figure 4.2: General architecture for Deepfacto: an NMF module with rank r is added in the middle of two arbitrary blocks. Note, only $\boldsymbol{S}_t$ is passed to the next blocks.

where γ_t is the step-size and $\boldsymbol{V}_t$ is the corresponding multipliers to enforce the matrix factorization equality $\boldsymbol{Z}_t = \boldsymbol{M}_t\boldsymbol{S}_t$. The NMF module adds a nonconvex term to the optimization. However, in the alternating optimization scheme, while keeping either $\boldsymbol{M}_t$ or $\boldsymbol{S}_t$ constant, solving for the other term would reduce to a normal convex least-squares problem. The rest of the updates are the same as in section 4.2.2. Note that, trivially to not change the input dimension of the next block after the NMF module, one can simply add an affine layer to increase the dimensions without changing the formulation.

At testing time, one only needs to perform a nonnegative projection since the basis matrix M will be given, which can be solved using a convex solver such as LBFGS. Note that for simplicity, we only formulated adding *one* NMF module in the middle of the blocks. This can be simply extended to as many NMF modules as needed in the architecture.

4.4 Experiments and Results

4.4.1 Setup

Our implementation uses the PyTorch framework [74]. All the experiments are run on a machine with a single NVIDIA GeForce RTX 2080 Ti GPU. The results presented for each of the following experiments are selected from their best performance after grid search over the hyper-parameters, both for our method and the baselines. Note, in the following Figures (4.3, 4.4, 4.5), each line shows average test set accuracy over five runs with different initialization. The shaded area corresponds to ± 1 standard deviation.

4.4.2 Supervised Deep Network Training

In this section, we present the experiment results from training conventional neural networks as a constrained optimization problem in a supervised setting. The results from the proposed methods in section 4.2.2 are compared with baselines including training a conventional neural network in an end-to-end setting using SGD, ADMM formulation in [98], and training the neural networks by BCD in [118]. Further, we evaluate the Stochastic Block-ADMM algorithm proposed in section 4.2.2 and have compared the result on ResNet-18 [40].

4.4.2.1 MNIST

For the first part of the experiments in the supervised setting, MNIST dataset of handwritten digits [114], is used. Throughout the experiments, 60,000 samples are used during training and the test time performance using the remaining 10,000 samples are reported in Figure 4.3. The architecture of the *shallow* network used for the experiments incorporates three fully-connected layers with 128-neuron hidden layers ($784 - 128 - 128 - 10$) and $ReLU$ nonlinearity. In order to make a fair comparison with [98] which can only work with Mean Squared Error (MSE), we utilize MSE as the training objective ($\mathcal{J}$) while the more common Cross-Entropy (CE) can effortlessly be adapted to our Block-ADMM architecture (e.g. Sec. 4.4.2.2). For training the standard ADMM and [98], all the parameters are initialized by sampling from the uniform distribution $x \sim U(0, 1)$ and down scaled by a factor of 0.0001. we set $\beta_l = \gamma_l = 10$ for all of the layers. Note that Algorithm 1 can be converted to the the method in [98] by setting dual variables $\forall \ell \neq L \; \boldsymbol{U}_\ell = \boldsymbol{0}$; and discarding their updates. To regularize the weights, $\boldsymbol{W}_l$ during the training, l_2 norm (weight decay) is used with $\lambda_\ell = 5 \times 10^{-5}$. We observed that the regularization term significantly improves the optimization behavior and without it, the training was not stable. In addition, for conventional neural network training using back-propagation in Fig. 4.3, SGD with learning rate of 0.005 is used.

For the training of Stochastic Block-ADMM presented in Algorithm 2, the shallow architecture is split into 3 one-layer blocks. For each blocks $\beta_t = 1$ is set, the weights are initialized using normal distribution, dual variables $\boldsymbol{U}_t$ are initialized using a uniform distribution, and auxiliary variables $\boldsymbol{Z}_t$ are initialized in a forward pass through the

network. During training, a mini-batch size of 128 is chosen, and both of sub-problem updates for block_{Θ_t} and $\boldsymbol{Z}_t$ are performed using Adam. Figure 4.3 shows that Block-ADMM outperforms the baselines by reaching 97.09% average test accuracy. Note accuracy for all methods is lower than normal because of the MSE loss function used that is not the best choice for classification but chosen for comparison with previous ADMM methods.

To further analyze robustness of stochastic Block-ADMM against vanishing gradients, we run the previous experiments on an unconventional architecture with 10 fully-connected

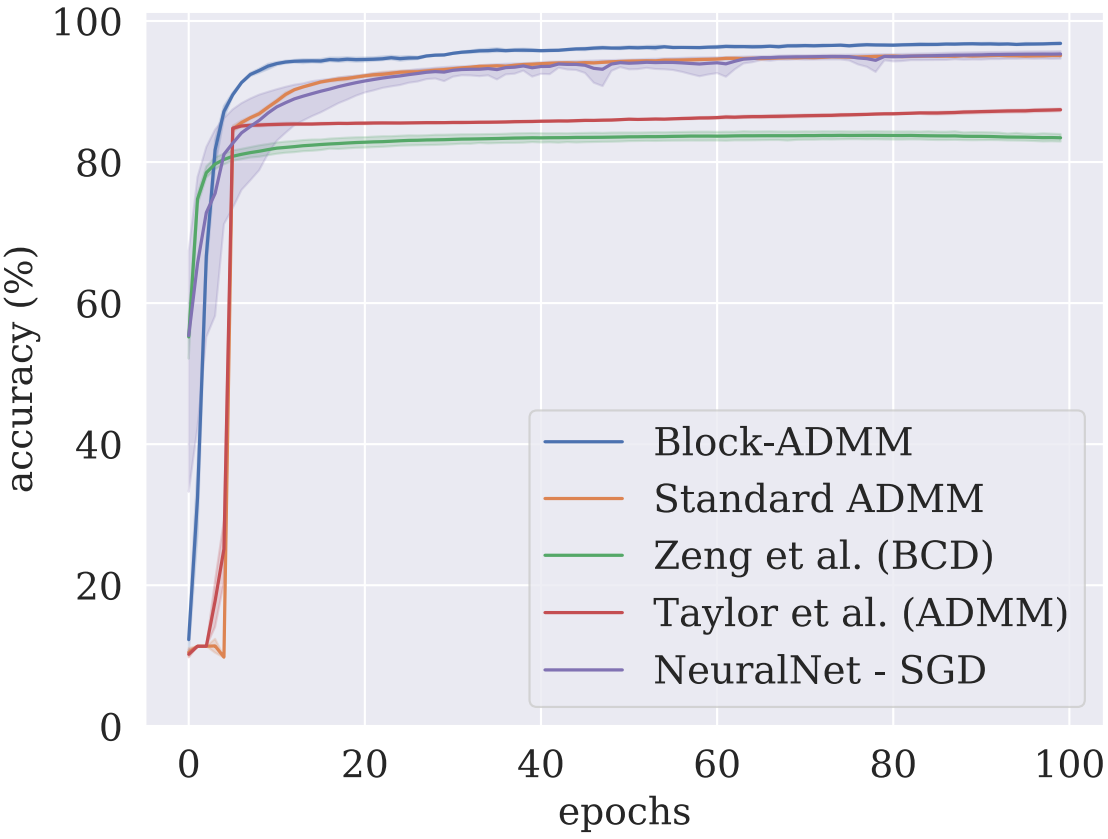


Figure 4.3: Test set accuracy on MNIST using network with 3 fully-connected layers: $784 - 128 - 128 - 10$. Final test accuracies: "Stochastic Block ADMM": **97.53**%, "Standard ADMM": 95.02%, "Zeng *et al.*": 83.28% , "Taylor *et al.*":87.52%, "SGD": 95.29%. (Best viewed in color)

layers. Note that normally this will not be adopted because of the severe overfitting and gradient vanishing problems, but here we utilized this setting to test our resistance to these problems. Figure 4.4 illustrates the experiment results. Block-ADMM reaches average final test accuracy of 94.43% while SGD and ADAM reache 10.28% and 58%, respectively. As it can be seen in Figure 4.4, we also compared our method with the recent work of [118]. We observed their BCD formulation[1] to be unstable, sensitive to network architectures, and eventually, not converging after 300 epochs. Although we still exhibited some overfitting, we can see our approach is significantly better in handling of the vanishing gradient problem. We even tested our performance with 20 fully-connected layers. Results show that although there is slightly more overfitting, our algorithm can still find a reasonable solution (Fig. 4.4), showing its potential in helping with training scenarios with vanishing gradients.

4.4.2.2 CIFAR-10

To test the performance of Block-ADMM in a more complex supervised setting than the previous baselines of ADMM, we trained Resnet-18 [40] on the CIFAR10 dataset [57]. We used 50,000 samples for training and the remaining 10,000 for evaluation. It is critical to validate Block-ADMM in settings where deep and modern architectures such as deep residual networks, convolutional layers, cross-entropy loss function, etc., are used. To have a fair comparison, we followed the configuration suggested in [36] by converting Resnet-18 network into two blocks ($T = 2$), with the splitting point located at the end of CONV3_X layer. We used the Adam optimizer to update both the blocks and the decoupling variables with the learning rates of $\eta_t = 0.005$ and $\zeta_t = 0.5$. We noted since the auxiliary variables $\boldsymbol{Z}_t$ are not "shared parameters" across data samples, they usually require a higher learning rate in Algorithm 2. We also found the step size $\beta_t = 1$ sufficient to enforce the block's coupling. Figure. 4.5 shows the results from our method compared with two baselines: [36], and conventional end-to-end neural network training using back-propagation. Our algorithm consistently outperformed [36] however cannot match the conventional SGD results. There are several factors that we hypothesize that might have contributed to the performance difference: 1) in a ResNet the residual structure already partially solved the vanishing gradient problem, hence SGD/Adam performs

[1] code taken from: `https://github.com/timlautk/BCD-for-DNNs-PyTorch`

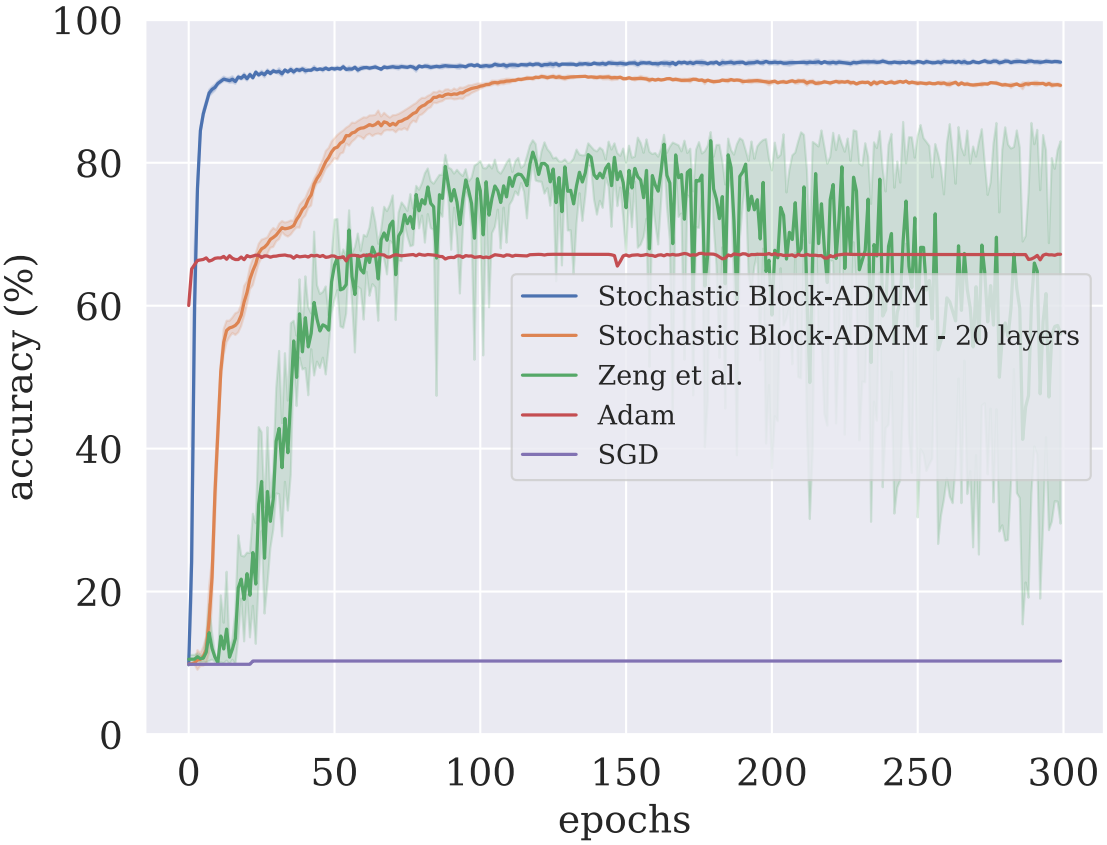


Figure 4.4: Test accuracies from deep architectures on MNIST. Block-ADMM demonstrates stable convergence and obtains final test accuracy of **94.43**% (10 layers), and 91.75% (20 layers) respectively, while SGD and Adam (10 layers) fail due to vanishing gradients (Best viewed in color)

significantly better than a fully-connected version; 2) The common data augmentation in CIFAR will end up sending a different training example to the optimization algorithm at each iteration, which does not seem to affect SGD but seem to affect ADMM convergence somewhat; 3) we noticed decreasing the learning rate for Θ_t updates does not impact the performance as it does for an end-to-end back-propagation using SGD. Still, we obtained the best performance of ADMM-type methods on both MNIST and CIFAR datasets, showing the promise of our approach.

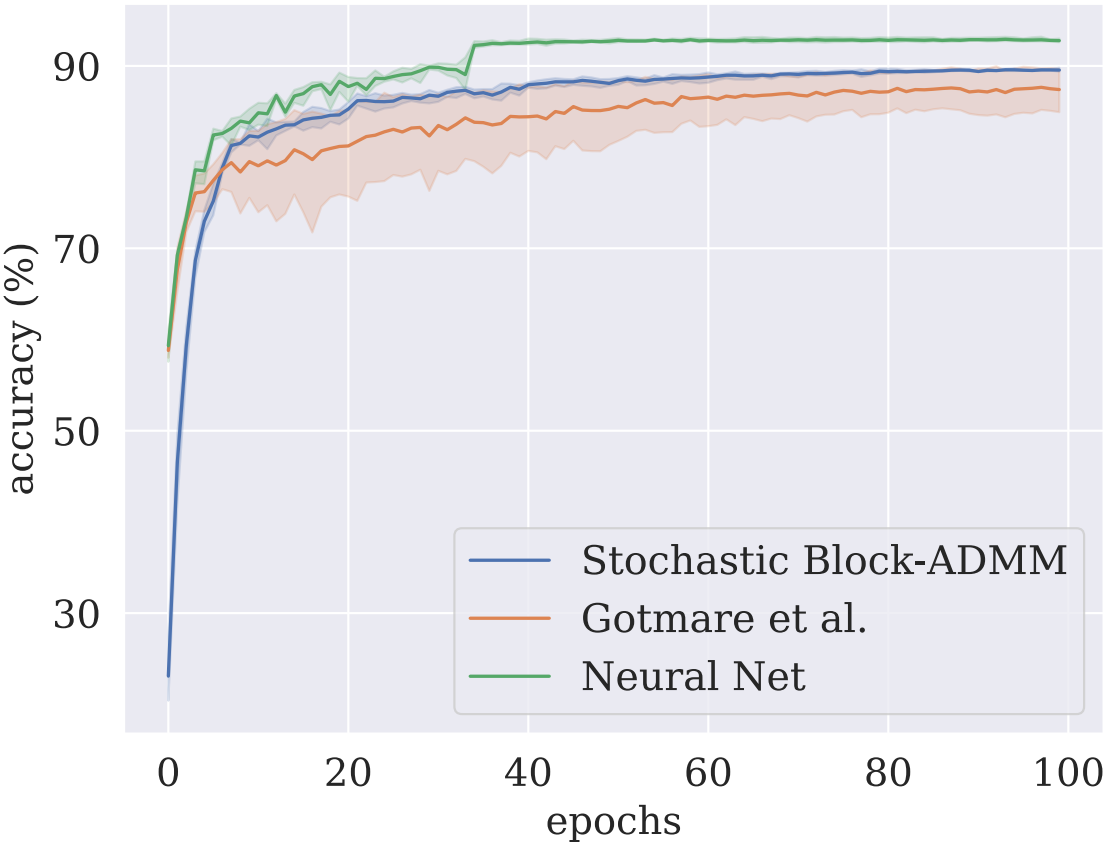


Figure 4.5: Test set accuracy on CIFAR-10 dataset. Final accuracy "Block ADMM": 89.66%, "Gotmare *et al.*":87.12%, "SGD": **92.70**%. (Best viewed in color.)

4.4.2.3 Wall Clock Time Comparison

In this section, we setup a experiment to further analyse the efficiency of Stochastic Block ADMM and compare its training wall clock time against other baselines: [36, 118] (BCD), and [105] (ADMM). For this purpose, we follow the similar settings as in section 4.1 for a supervised Deep Neural Network (DNN) training over MNIST dataset. Figure 4.6 shows the test set accuracy v.s. the training wall clock time from different methods. All the experiments are run on a machine with a single NVIDIA GeForce RTX 2080 Ti GPU.

The methods are implemented in PyTorch framework – except for dlADMM [105] that is implemented[2] in "cupy", a NumPy-compatible matrix library accelerated by CUDA. [36] and Stochastic Block ADMM are trained with a mini-batch size of 128 and [118, 105] are trained in a batch setting. Note that in Figure 4.6, the time recorded merely shows the *training time* and excludes the time taken for initialization, data loading, etc. It can be observed that [36] and dlADMM are showing much slower convergence behaviors than Stochastic Block ADMM. We speculate that enforcing all the constraints by dual variables along with the efficient and cheap mini-batch updates in our method highly contributes to the convergence speed as well as its performance superiority over the other methods, including [118].

4.4.3 Weakly Supervised Training

4.4.3.1 Factorizing the activations

With the assumption that the observations are formed by a linear combination of few basis vectors, one can approximate a given matrix $\boldsymbol{X} \in \mathbb{R}^{m\times n}$ into a *basis* matrix $\boldsymbol{M} \in \mathbb{R}^{m\times r}$ and an *score* matrix $\boldsymbol{S} \in \mathbb{R}^{r\times n}$ such that $\boldsymbol{X} \approx \boldsymbol{M}\boldsymbol{S}$ where r is the (reduced) *rank* of the factorized matrices – commonly $r \ll \min(m, n)$. Methods such as NMF would restrict the entries of $\boldsymbol{M}$ and $\boldsymbol{S}$ to be non-negative ($\forall i, j\ \boldsymbol{M}_{ij} \geq 0,\ \boldsymbol{S}_{ij} \geq 0$) which forces the decomposition to be only *additive*. This has been shown to result in a parts-based representation that is intuitively more close to human perception. It is also worth mentioning that obviously, the matrix $\boldsymbol{X}$ needs to be positive ($\forall i, j\ \boldsymbol{X}_{ij} \geq 0$). For non-negative factorization on the activations of the DNNS, due to the common use of activation functions such as $ReLU$, this would not impose any constraints in most of the problems.

Activations of the CNN networks are generally Tensors of the shape $\mathbf{Z}_\ell \in \mathbb{R}^{(N,C,H,W)}$ which namely represent the batch size of the input, the number of the channels, the height of each channel, and the corresponding width. To adapt such tensors for the NMF problem, we reshape the tensor into the matrix $\boldsymbol{Z}_\ell \in \mathbb{R}^{C\times(N*H*W)}$ by stacking it over its channels while flattening the other dimensions. This way, the channels would be embedded into a pre-defined small dimension r while keeping each sample and pixels

[2] code taken from `https://github.com/xianggebenben/dlADMM`

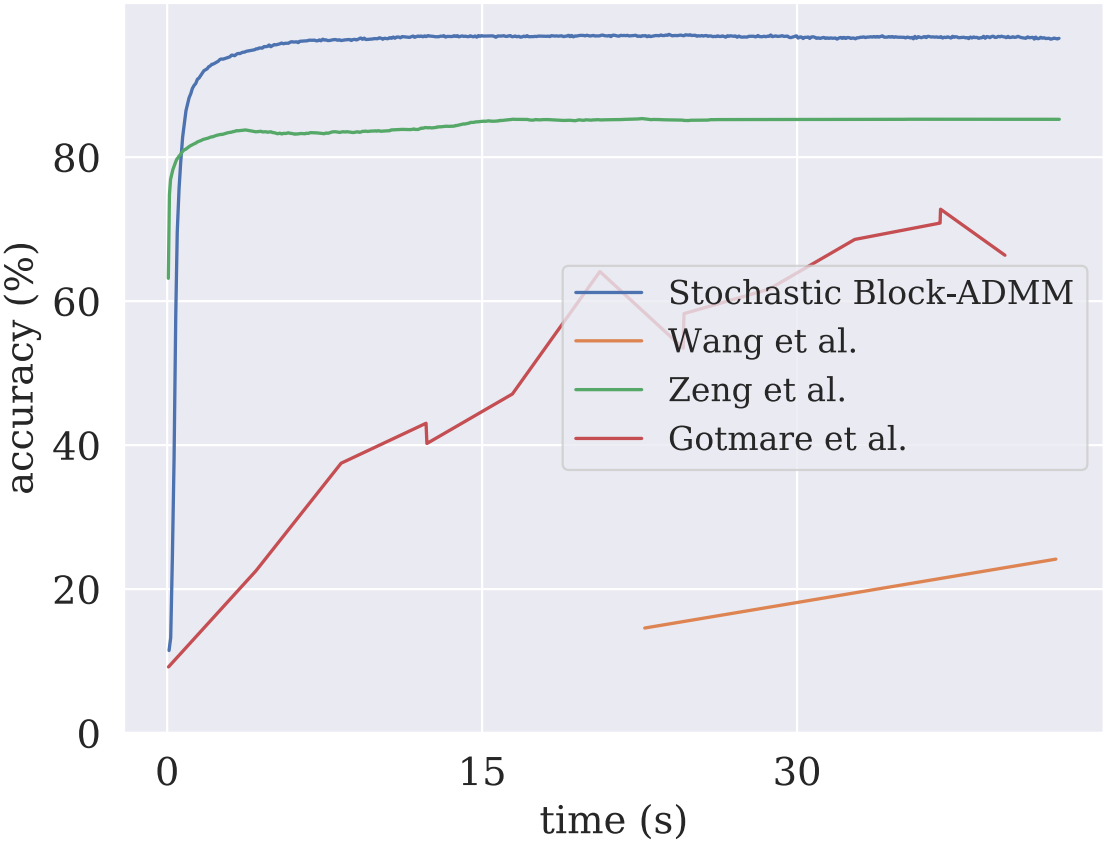


Figure 4.6: Test set accuracy v.s. training wall clock time comparison of different alternating optimization methods for training DNNs on MNIST dataset. Our method (blue) shows superior performance while presenting comparable convergence speed against [118] (green).

information.

4.4.3.2 LFWA

To further show that Block-ADMM can be adapted to complex settings, we evaluate our proposed method in a weakly-supervised problem where we used DeepFacto 4.3 to learn a non-negative factorized representation of the DNN activations while training

end-to-end on the LFWA dataset [46]. Next, similar to [67], linear SVMs are used over the factorized space to predict face attributes. The reason for this setup is to examine whether the network can extract a disentangled representation that linearly corresponds to human-marked attributes that the network does not have prior knowledge of. We used the Inception-Resnet architecture from [84], pre-trained on the VGGFace-2 [9] dataset as the back-bone. For the weakly-supervised problem of attribute classification using DeepFacto, we attached the DeepFacto module to the last convolutional layer of the Inception-Resnet-V1 architecture followed by a *ReLU*. This layer has 1792 channels and, for a given input of the size 160×160 pixels (the original input size from the LFWA dataset), the height and the width are both equal to 3. To incorporate an NMF, we follow the same approach as in Figure 4.2 where the pretrained DNN is the first block, and we add a simple fully-connected layer over the score matrix $\boldsymbol{S}_t$ to train a face-verification network with a triplet loss [42]. The choice of a simple fully-connected layer is two-folded. First, to lift the dimensions of the embedding needed for training, particularly when $\boldsymbol{S}_t$ is low rank. Second, the embedding would be only a linear combination of the score matrix $\boldsymbol{S}_t$, directly guiding it using the supervised signal coming from the Triplet Loss. We conjecture the score matrix $\boldsymbol{S}_t$ will be guided to learn an interpretable factorization due to the nonnegativity constraint [16]. Note that the latest activation in the network that is followed by a ReLU is selected from the Resnet-Inception network. This is due to the nonnegative constraint in the NMF, i.e. the input to the NMF should be also positive. To have a warm start for an end-to-end training of DeepFacto, we first pre-train the NMF module having the Inception-Resnet block freezed. Then, we fine-tune the block parameters as well as the NMF module in an alternating fashion, similar to Algorithm 2. Note, the rank of the NMF in DeepFacto is a hyperparameter and we selected three different values ($r = 4, 32, 256$) in the experiments. The final $r = 256$ is also the latent space dimensionality in [67]. Table. 4.1 illustrates average prediction accuracy over LFWA attributes [3] from DeepFacto and other supervised and weakly supervised baselines. This validates that DeepFacto has learned a meaningful representation of the attributes by unsupervised factorization of the activations. More details about the experiments are presented in the supplementary material.

Table 4.2 shows the prediction accuracy of each attribute in LFWA dataset and compares DeepFacto with different ranks ($r = 4, 32, 256$) against other supervised and

[3]The common 40 attributes with Celeb-A dataset [68]

Table 4.1: Average prediction accuracy on 40 attributes from LFWA dataset. Weakly-supervised methods train the network without access to attribute labels. Final classification then comes from a linear SVM on their latent representations.

LFWA	Accuracy
[123] (supervised)	81.00%
[68] (supervised)	84.00%
[67] (weakly-supervised)	83.16%
DeepFacto - rank 4 (weakly-supervised)	74.80%
DeepFacto - rank 32 (weakly-supervised)	81.39%
DeepFacto - rank 256 (weakly-supervised)	**87.03**%

weakly-supervised baselines. It can be noted that our method can generate highly informative representation of the LFWA attributes without accessing their labels. This supports our conjecture that DeepFacto, by non-negatively factorizing the activations of the DNNs in and end-to-end training, can lead to an interpretable decomposition of the DNN activations.

4.5 Summary

In this chapter, we proposed stochastic Block-ADMM as an approach to train deep networks. Through updates with stochastic gradients, we improve over the performance of previous attempts using ADMM to train neural networks. Although the performance of ADMM is still not up to par with SGD/Adam in residual networks, we have shown improvements over them in training deep networks without residual connections. We also propose DeepFacto which jointly trains an NMF layer within a deep network and show encouraging results on a supervised disentanglement benchmark.

Table 4.2: Prediction accuracy (%) of individual attributes in LFWA dataset. DeepFacto with other weakly-supervised and supervised baselines.

Attributes	DeepFacto (Weakly-Supervised)			[68] (Weakly-Supervised)	[67] (Supervised)	[123] (Supervised)
	r =256	32	4			
5 o Clock Shadow	83.3	80.0	68.7	78.8	**84**	**84**
Arched Eyebrows	**86.6**	83.9	79.2	78.1	82	79
Attractive	**84.3**	79.8	73.3	79.2	83	81
Bags Under Eyes	**83.9**	72.5	64.5	83.1	83	80
Bald	**94.3**	93.3	89.3	84.8	88	84
Bangs	**93.2**	88.4	84.4	86.5	88	84
Big Lips	**83.2**	77.0	71.9	75.2	75	73
Big Nose'	80.1	68.7	61.4	**81.3**	81	79
Black Hair	**92.7**	91.4	87.4	87.4	90	87
Blond Hair	**97.9**	97.3	93.2	94.2	97	94
Blurry	**90.4**	90.5	86.5	78.4	74	74
Brown Hair	**78.4**	74.4	70.2	72.9	77	74
Bushy Eyebrows	**84.0**	78.6	63.4	83.0	82	79
Chubby	**80.5**	75.2	71.1	74.6	73	69
Double Chin	**86.0**	77.9	72.3	80.2	78	75
Eyeglasses	94.3	89.6	84.8	89.5	**95**	89
Goatee	**89.1**	85.4	80.0	78.6	78	75
Gray Hair	**91.9**	90	85.6	86.9	84	81
Heavy Makeup	**96.3**	91.5	87.4	94.5	95	93
High Cheekbones	**90.4**	79.0	72.1	88.8	88	86
Male	81.3	76.6	70.5	**94.3**	94	92
Mouth Slightly Open	**85.4**	78.0	73.3	81.7	82	78
Mustache	**96.6**	93.2	91.3	83.3	92	87
Narrow Eyes	**78.3**	69.3	58.4	77.5	81	73
No Beard	**79.5**	73.0	65.5	77.7	79	75
Oval Face	**80.6**	73.2	66.1	78.7	74	72
Pale Skin	75.1	66.7	60.6	**89.8**	84	84
Pointy Nose'	**81.6**	73.7	62.2	79.8	80	76
Receding Hairline	84.0	80.9	73.8	**88.0**	85	84
Rosy Cheeks	**87.3**	87.4	83.4	79.9	78	73
Sideburns	**85.4**	81.5	75.8	80.5	77	76
Smiling	**92.6**	78.7	69.8	92.2	91	89
Straight Hair	**82.8**	77.0	72.1	73.6	76	73
Wavy Hair	80.4	77.0	68.3	**81.7**	76	75
Wearing Earrings	**95.4**	91.6	87.1	89.7	94	92
Wearing Hat	**93.0**	90.2	87.0	80.5	88	82
Wearing Lipstick	**95.8**	92.8	89.0	91.4	95	93
Wearing Necklace	**93.0**	89.8	85.1	84.0	88	86
Wearing Necktie	**79.8**	75.2	70.6	78.7	79	79
Young	**91.0**	88.4	84.4	79.2	86	82
Average	**87.0**	81.4	74.8	83.1	84	81

Chapter 5: Conclusion

In this thesis, we have taken attempts to (partially) disentangle the high-dimensional activation space of the deep networks into low-dimensional representations that favorable constraints are enforced on. In chapter 3 we proposed a novel explanation network, that can be attached to any layer in a deep network to embed the layer into several concepts that can approximate a N-dimensional prediction output from the network. A sparse reconstruction autoencoder (SRAE) is proposed to avoid degeneracy and improve orthogonality. Through the XNN framework, faithfulness, that the deep learning predictions can be faithfully approximated from those few concepts; locality, that the concepts are relatively spatially localized in images so that human can understand them; and orthogonality, that the concepts themselves are as independent of each other as possible were imposed over the embedded space. We evaluated our method through extensive qualitative and quantitative experiments. Human evaluation was conducted to investigate the performance of our approach against a baseline. We also proposed automatic evaluation metrics to evaluate the explanations on a fine-grained bird classification dataset (CUB-200-2011) and a scene recognition dataset (Places365). Our quantitative and qualitative results show that the network can be extremely faithful to the original prediction network, and can indeed extract high-level concepts from a CNN that is meaningful to human, and different from existing concepts. We also performed experiments training XNN on convolutional layers, which indicated some difficulty in faithfully explaining the DNN prediction from early convolutional layers. The future steps to broaden the scope of chapter 3 would to extend XNN to explain other types of neural networks, such as recurrent networks and convolutional-recurrent ones.

Furthermore, in chapter 4, we proposed a novel method to learn a factorized activation space of the deep networks while training in an end-to-end fashion. This enabled us to train the deep networks while explicitly encouraging (partial) disentanglement over their activations space. To that end, we presented DeepFacto which jointly trains an NMF layer within a deep network. We evaluated DeepFacto in a weakly-supervised setting on a face verification network. While there was no supervision over the attribute labels of

the training samples, using linear SVMs on the factorized space learned from our method has shown superior performance against supervised and weakly-supervised baselines in attribute prediction. Since jointly training an NMF decomposition with deep learning is highly non-convex and cannot be addressed by the conventional backpropagation and SGD algorithms, we proposed Stochastic Block ADMM as a novel approach to training deep networks with non-differentiable layers. We evaluated Stochastic Block ADMM through experiments in a supervised setting on MNIST and CIFAR-10 datasets. Also, we have shown that the Stochastic Block ADMM method can mitigate the long-known problem of gradient vanishing and get to a reasonably good performance while methods such as SGD and *Adam* fail. Through updates with stochastic gradients, we improve over the performance of previous attempts using ADMM to train neural networks. We also have shown that Stochastic Block ADMM can show comparable results in training more sophisticated networks such as ResNet-18. Although the performance of ADMM is still not up to par with SGD/Adam in residual networks, we have shown improvements over them in training deep networks without residual connections. We believe the preliminary results presented in this work set up future work that further explores different aspects of utilizing ADMM in deep network training, including parallelization, stability and data augmentation.

Bibliography

[1] Pulkit Agrawal, Ross Girshick, and Jitendra Malik. Analyzing the performance of multilayer neural networks for object recognition. In *European Conference on Computer Vision*, pages 329–344. Springer, 2014.

[2] David Alvarez-Melis and Tommi S. Jaakkola. Towards robust interpretability with self-explaining neural networks. In *NIPS*, pages 7786–7795, 2018.

[3] Armin Askari, Geoffrey Negiar, Rajiv Sambharya, and Laurent El Ghaoui. Lifted neural networks. *arXiv preprint arXiv:1805.01532*, 2018.

[4] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Advances in Neural Information Processing Systems*, 2014.

[5] David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection: Quantifying interpretability of deep visual representations. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[6] Léon Bottou. Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*, pages 421–436. Springer, 2012.

[7] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011.

[8] Chunshui Cao, Xianming Liu, Yi Yang, Yinan Yu, Jiang Wang, Zilei Wang, Yongzhen Huang, Liang Wang, Chang Huang, Wei Xu, et al. Look and think twice: Capturing top-down visual attention with feedback convolutional neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2956–2964, 2015.

[9] Q. Cao, L. Shen, W. Xie, O. M. Parkhi, and A. Zisserman. Vggface2: A dataset for recognising faces across pose and age. In *International Conference on Automatic Face and Gesture Recognition*, 2018.

[10] Miguel Carreira-Perpinan and Weiran Wang. Distributed optimization of deeply nested systems. In *Artificial Intelligence and Statistics*, pages 10–19, 2014.

[11] Rich Caruana, Yin Lou, Johannes Gehrke, Paul Koch, Marc Sturm, and Noemie Elhadad. Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pages 1721–1730, 2015.

[12] Aditya Chattopadhyay, Anirban Sarkar, Prantik Howlader, and Vineeth N. Balasubramanian. Grad-cam++: Generalized gradient-based visual explanations for deep convolutional networks. *CoRR*, abs/1710.11063, 2017.

[13] Zhengping Che, Sanjay Purushotham, Robinder Khemani, and Yan Liu. Interpretable deep models for icu outcome prediction. In *American Medical Informatics Association Annual Symposium*, 2016.

[14] Hongling Chen, Mingyan Gao, Ying Zhang, Wenbin Liang, and Xianchun Zou. Attention-based multi-nmf deep neural network with multimodality data for breast cancer prognosis model. *BioMed Research International*, 2019, 2019.

[15] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *International Conference on Machine Learning*, pages 2285–2294, 2015.

[16] Edo Collins, Radhakrishna Achanta, and Sabine Susstrunk. Deep feature factorization for concept discovery. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 336–352, 2018.

[17] Piotr Dabkowski and Yarin Gal. Real time image saliency for black box classifiers. In *Advances in Neural Information Processing Systems*, pages 6967–6976, 2017.

[18] Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in neural information processing systems*, pages 2933–2941, 2014.

[19] Wei Deng and Wotao Yin. On the global and linear convergence of the generalized alternating direction method of multipliers. *Journal of Scientific Computing*, 66(3):889–916, 2016.

[20] Chris Ding, Xiaofeng He, and Horst D Simon. On the equivalence of nonnegative matrix factorization and spectral clustering. In *Proceedings of the 2005 SIAM international conference on data mining*, pages 606–610. SIAM, 2005.

[21] Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017.

[22] Gintare Karolina Dziugaite and Daniel M Roy. Neural network matrix factorization. *arXiv preprint arXiv:1511.06443*, 2015.

[23] Christopher Z Eddy, Xinyao Wang, Fuxin Li, and Bo Sun. The morphodynamics of 3d migrating cancer cells. *arXiv preprint arXiv:1807.10822*, 2018.

[24] Ethan Elenberg, Alexandros G Dimakis, Moran Feldman, and Amin Karbasi. Streaming weak submodularity: Interpreting neural networks on the fly. In *Advances in Neural Information Processing Systems*, pages 4044–4054, 2017.

[25] Andre Esteva, Brett Kuprel, Roberto A. Novoa, Justin Ko, Susan M. Swetter, Helen M. Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542:115–118, 2017.

[26] Jennifer Flenner and Blake Hunter. A deep non-negative matrix factorization neural network.

[27] Xiao Fu, Kejun Huang, Nicholas D Sidiropoulos, and Wing-Kin Ma. Nonnegative matrix factorization for signal and data analytics: Identifiability, algorithms, and applications. *arXiv preprint arXiv:1803.01257*, 2018.

[28] Xiao Fu, Kejun Huang, Nicholas D Sidiropoulos, Qingjiang Shi, and Mingyi Hong. Anchor-free correlated topic modeling. *IEEE transactions on pattern analysis and machine intelligence*, 41(5):1056–1071, 2018.

[29] Xiao Fu, Shahana Ibrahim, Hoi-To Wai, Cheng Gao, and Kejun Huang. Block-randomized stochastic proximal gradient for low-rank tensor factorization. *arXiv preprint arXiv:1901.05529*, 2019.

[30] Daniel Gabay and Bertrand Mercier. *A dual algorithm for the solution of non linear variational problems via finite element approximation.*

[31] Leila Ghorbanzadeh, Ahmad Esmaili Torshabi, Jamshid Soltani Nabipour, and Moslem Ahmadi Arbatan. Development of a synthetic adaptive neuro-fuzzy prediction model for tumor motion tracking in external radiotherapy by evaluating various data clustering algorithms. *Technology in cancer research & treatment*, 15(2):334–347, 2016.

[32] Georgia Gkioxari, Ross Girshick, and Jitendra Malik. Actions and attributes from wholes and parts. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2470–2478, 2015.

[33] Abel Gonzalez-Garcia, Davide Modolo, and Vittorio Ferrari. Do semantic parts emerge in convolutional neural networks? *arXiv preprint arXiv:1607.03738*, 2016.

[34] Abel Gonzalez-Garcia, Davide Modolo, and Vittorio Ferrari. Do semantic parts emerge in convolutional neural networks? *International Journal of Computer Vision*, 126(5):476–494, 2018.

[35] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[36] Akhilesh Gotmare, Valentin Thomas, Johanni Brea, and Martin Jaggi. Decoupling backpropagation using constrained optimization methods. 2018.

[37] Emad M Grais and Hakan Erdogan. Single channel speech music separation using nonnegative matrix factorization and spectral masks. In *2011 17th International Conference on Digital Signal Processing (DSP)*, pages 1–6. IEEE, 2011.

[38] Fangda Gu, Armin Askari, and Laurent El Ghaoui. Fenchel lifted networks: A lagrange relaxation of neural network training. *arXiv preprint arXiv:1811.08039*, 2018.

[39] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[40] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[41] Lisa Anne Hendricks, Zeynep Akata, Marcus Rohrbach, Jeff Donahue, Bernt Schiele, and Trevor Darrell. Generating visual explanations. In *European Conference on Computer Vision*, 2016.

[42] Elad Hoffer and Nir Ailon. Deep metric learning using triplet network. In *International Workshop on Similarity-Based Pattern Recognition*, pages 84–92.

[43] Ehsan Hosseini-Asl, Jacek M Zurada, and Olfa Nasraoui. Deep learning of part-based representation of data using sparse autoencoders with nonnegativity constraints. *IEEE transactions on neural networks and learning systems*, 27(12):2486–2498, 2015.

[44] Feihu Huang and Songcan Chen. Mini-batch stochastic admms for nonconvex nonsmooth optimization. *arXiv preprint arXiv:1802.03284*, 2018.

[45] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[46] Gary B. Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical Report 07-49, University of Massachusetts, Amherst, October 2007.

[47] Ashesh Jain, Amir R Zamir, Silvio Savarese, and Ashutosh Saxena. Structural-rnn: Deep learning on spatio-temporal graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5308–5317, 2016.

[48] Hangyan Jiang and Zexi Jin. *Deep Convolutional NMF Net*. PhD thesis, 2018.

[49] Ian T Jolliffe. Principal components in regression analysis. In *Principal component analysis*, pages 129–155. Springer, 1986.

[50] Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[51] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

[52] Farkhondeh Kiaee, Christian Gagné, and Mahdieh Abbasi. Alternating direction method of multipliers for sparse convolutional neural networks. *arXiv preprint arXiv:1611.01590*, 2016.

[53] Pieter-Jan Kindermans, Sara Hooker, Julius Adebayo, Maximilian Alber, Kristof T Schutt, Sven Dahne, Dumitru Erhan, and Been Kim. The (un)reliability of saliency methods. *arXiv preprint arXiv:1711.00867v1*, 2017.

[54] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[55] Ryan Kiros, Ruslan Salakhutdinov, and Richard S Zemel. Multimodal neural language models. In *Icml*, volume 14, pages 595–603, 2014.

[56] Chen Kong, Dahua Lin, Mohit Bansal, Raquel Urtasun, and Sanja Fidler. What are you talking about? text-to-image coreference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3558–3565, 2014.

[57] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research).

[58] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[59] Todd Kulesza, Margaret Burnett, Weng-Keen Wong, and Simone Stumpf. Principles of explanatory debugging to personalize interactive machine learning. In *Proceedings of the 20th International Conference on Intelligent User Interfaces*, pages 126–137. ACM, 2015.

[60] Jonathan Le Roux, John R Hershey, and Felix Weninger. Deep nmf for speech separation. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 66–70. IEEE, 2015.

[61] Daniel D Lee and H Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788, 1999.

[62] Honglak Lee, Alexis Battle, Rajat Raina, and Andrew Y Ng. Efficient sparse coding algorithms. *Advances in neural information processing systems*, 19:801, 2007.

[63] Benjamin Letham, Cynthia Rudin, Tyler H. McCormick, and David Madigan. Interpretable classifiers using rules and bayesian analysis: Building a better stroke prediction model. *The Annals of Applied Statistics*, 9:1350–1371, 2015.

[64] Oscar Li, Hao Liu, Chaofan Chen, and Cynthia Rudin. Deep learning for case-based reasoning through prototypes: A neural network that explains its predictions. In *AAAI*, pages 3530–3537, 2018.

[65] Dahua Lin, Sanja Fidler, Chen Kong, and Raquel Urtasun. Visual semantic search: Retrieving videos via complex textual queries. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2657–2664, 2014.

[66] Haifeng Liu, Zhaohui Wu, Xuelong Li, Deng Cai, and Thomas S Huang. Constrained nonnegative matrix factorization for image representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(7):1299–1311, 2011.

[67] Yu Liu, Fangyin Wei, Jing Shao, Lu Sheng, Junjie Yan, and Xiaogang Wang. Exploring disentangled feature representation beyond face identification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2080–2089, 2018.

[68] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of the IEEE international conference on computer vision*, pages 3730–3738, 2015.

[69] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, pages 4765–4774, 2017.

[70] José Oramas M., Kaili Wang, and Tinne Tuytelaars. Visual explanation by interpretation: Improving visual feedback capabilities of deep neural networks. *CoRR*, abs/1712.06302, 2017.

[71] Calvin Murdock, MingFang Chang, and Simon Lucey. Deep component analysis via alternating direction neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 820–836, 2018.

[72] Hua Ouyang, Niao He, Long Tran, and Alexander Gray. Stochastic alternating direction method of multipliers. In *International Conference on Machine Learning*, pages 80–88, 2013.

[73] Dong Huk Park, Lisa Anne Hendricks, Zeynep Akata, Bernt Schiele, Trevor Darrell, and Marcus Rohrbach. Attentive explanations: Justifying decisions and pointing to the evidence. *arXiv Preprint:1612.04757*, 2016.

[74] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[75] V Paul Pauca, Jon Piper, and Robert J Plemmons. Nonnegative matrix factorization for spectral data analysis. *Linear algebra and its applications*, 416(1):29–47, 2006.

[76] Vitali Petsiuk, Abir Das, and Kate Saenko. Rise: Randomized input sampling for explanation of black-box models. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2018.

[77] Zhongang Qi, Saeed Khorram, and Fuxin Li. Visualizing deep networks by optimizing with integrated gradients.

[78] Zhongang Qi and Fuxin Li. Learning explainable embeddings for deep networks. In *NIPS 2017 workshop: Interpreting, Explaining and Visualizing Deep Learning - now what?*, 2017.

[79] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[80] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[81] Scott Reed, Zeynep Akata, Bernt Schiele, and Honglak Lee. Learning deep representations of fine-grained visual descriptions. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[82] 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.

[83] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[84] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[85] Ramprasaath R Selvaraju, Abhishek Das, Ramakrishna Vedantam, Michael Cogswell, Devi Parikh, and Dhruv Batra. Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization. *arXiv preprint arXiv:1610.02391*, 2016.

[86] Farial Shahnaz, Michael W Berry, V Paul Pauca, and Robert J Plemmons. Document clustering using nonnegative matrix factorization. *Information Processing & Management*, 42(2):373–386, 2006.

[87] Qingjiang Shi, Mingyi Hong, Xiao Fu, and Tsung-Hui Chang. Penalty dual decomposition method for nonsmooth nonconvex optimization. *arXiv preprint arXiv:1712.04767*, 2017.

[88] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach adn Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 2016.

[89] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[90] Marcel Simon and Erik Rodner. Neural activation constellations: Unsupervised part model discovery with convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1143–1151, 2015.

[91] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *ICLR Workshop*, 2014.

[92] Paris Smaragdis and Shrikant Venkataramani. A neural network alternative to non-negative audio models. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 86–90. IEEE, 2017.

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

[94] Jian Sun, Huibin Li, Zongben Xu, et al. Deep admm-net for compressive sensing mri. In *Advances in neural information processing systems*, pages 10–18, 2016.

[95] 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.

[96] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[97] S. Tan, R. Caruana, G. Hooker, P. Koch, and A. Gordo. Learning Global Additive Explanations for Neural Nets Using Model Distillation. *arXiv e-prints*, January 2018.

[98] Gavin Taylor, Ryan Burmeister, Zheng Xu, Bharat Singh, Ankit Patel, and Tom Goldstein. Training neural networks without gradients: A scalable admm approach. In *International conference on machine learning*, pages 2722–2731, 2016.

[99] George Trigeorgis, Konstantinos Bousmalis, Stefanos Zafeiriou, and Bjoern Schuller. A deep semi-nmf model for learning hidden representations. In *International Conference on Machine Learning*, pages 1692–1700, 2014.

[100] Ryan J Urbanowicz and Jason H Moore. Learning classifier systems: a complete introduction, review, and roadmap. *Journal of Artificial Evolution and Applications*, 2009, 2009.

[101] Berk Ustun and Cynthia Rudin. Supersparse linear integer models for optimized medical scoring systems. *Machine Learning*, 102(3):349–391, 2016.

[102] Thanh T Vu, Benjamin Bigot, and Eng Siong Chng. Combining non-negative matrix factorization and deep neural networks for speech enhancement and automatic

speech recognition. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 499–503. IEEE, 2016.

[103] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The caltech-ucsd birds-200-2011 dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011.

[104] Huahua Wang and Arindam Banerjee. Online alternating direction method (longer version). *arXiv preprint arXiv:1306.3721*, 2013.

[105] Junxiang Wang, Fuxun Yu, Xiang Chen, and Liang Zhao. Admm for efficient deep learning with global convergence. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 111–119, 2019.

[106] Yu Wang, Wotao Yin, and Jinshan Zeng. Global convergence of admm in nonconvex nonsmooth optimization. *Journal of Scientific Computing*, 78(1):29–63, 2019.

[107] Yu-Xiong Wang and Yu-Jin Zhang. Nonnegative matrix factorization: A comprehensive review. *IEEE Transactions on Knowledge and Data Engineering*, 25(6):1336–1353, 2012.

[108] Yulong Wang, Hang Su, Bo Zhang, and Xiaolin Hu. Interpret neural networks by identifying critical data routing paths. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[109] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[110] Tianjun Xiao, Yichong Xu, Kuiyuan Yang, Jiaxing Zhang, Yuxin Peng, and Zheng Zhang. The application of two-level attention models in deep convolutional neural network for fine-grained image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 842–850, 2015.

[111] Wei Xu, Xin Liu, and Yihong Gong. Document clustering based on non-negative matrix factorization. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 267–273. ACM, 2003.

[112] Yangyang Xu and Wotao Yin. Block stochastic gradient iteration for convex and nonconvex optimization. *SIAM Journal on Optimization*, 25(3):1686–1716, 2015.

[113] Bo Yang, Xiao Fu, Nicholas D Sidiropoulos, and Mingyi Hong. Towards k-means-friendly spaces: Simultaneous deep learning and clustering. In *Proceedings of the*

34th International Conference on Machine Learning-Volume 70, pages 3861–3870. JMLR. org, 2017.

[114] Christopher J.C. Burges Yann LeCun, Corinna Cortes. THE MNIST DATABASE of handwritten digits.

[115] Shaokai Ye, Tianyun Zhang, Kaiqi Zhang, Jiayu Li, Kaidi Xu, Yunfei Yang, Fuxun Yu, Jian Tang, Makan Fardad, Sijia Liu, et al. Progressive weight pruning of deep neural networks using admm. *arXiv preprint arXiv:1810.07378*, 2018.

[116] Jinshi Yu, Guoxu Zhou, Andrzej Cichocki, and Shengli Xie. Learning the hierarchical parts of objects by deep non-smooth nonnegative matrix factorization. *IEEE Access*, 6:58096–58105, 2018.

[117] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*, pages 818–833. Springer, 2014.

[118] Jinshan Zeng, Tim Tsz-Kit Lau, Shaobo Lin, and Yuan Yao. Global convergence of block coordinate descent in deep learning. In *International Conference on Machine Learning*, pages 7313–7323, 2019.

[119] Jinshan Zeng, Shao-Bo Lin, and Yuan Yao. A convergence analysis of nonlinearly constrained admm in deep learning. *arXiv preprint arXiv:1902.02060*, 2019.

[120] Han Zhang, Tao Xu, Mohamed Elhoseiny, Xiaolei Huang, Shaoting Zhang, Ahmed Elgammal, and Dimitris Metaxas. Spda-cnn: Unifying semantic part detection and abstraction for fine-grained recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1143–1152, 2016.

[121] Hui Zhang, Huaping Liu, Rui Song, and Fuchun Sun. Nonlinear non-negative matrix factorization using deep learning. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 477–482. IEEE, 2016.

[122] Jianming Zhang, Zhe Lin, Jonathan Brandt, Xiaohui Shen, and Stan Sclaroff. Top-down neural attention by excitation backprop. In *European Conference on Computer Vision*, pages 543–559. Springer, 2016.

[123] Ning Zhang, Manohar Paluri, Marc'Aurelio Ranzato, Trevor Darrell, and Lubomir Bourdev. Panda: Pose aligned networks for deep attribute modeling. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1637–1644, 2014.

[124] Quanshi Zhang, Ying Nian Wu, and Song-Chun Zhu. Interpretable convolutional neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8827–8836, 2018.

[125] Xiaopeng Zhang, Hongkai Xiong, Wengang Zhou, Weiyao Lin, and Qi Tian. Picking deep filter responses for fine-grained image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1134–1142, 2016.

[126] Yu Zhang, Xiu-Shen Wei, Jianxin Wu, Jianfei Cai, Jiangbo Lu, Viet-Anh Nguyen, and Minh N Do. Weakly supervised fine-grained categorization with part-based image representation. *IEEE Transactions on Image Processing*, 25(4):1713–1725, 2016.

[127] Ziming Zhang and Matthew Brand. Convergent block coordinate descent for training tikhonov regularized deep neural networks. In *Advances in Neural Information Processing Systems*, pages 1721–1730, 2017.

[128] Ziming Zhang, Yuting Chen, and Venkatesh Saligrama. Efficient training of very deep neural networks for supervised hashing. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1487–1495, 2016.

[129] Bo Zhao, Xiao Wu, Jiashi Feng, Qiang Peng, and Shuicheng Yan. Diversified visual attention networks for fine-grained object classification. *IEEE Transactions on Multimedia, to appear*, 2017.

[130] Junbo Zhao, Michael Mathieu, and Yann LeCun. Energy-based generative adversarial network. In *Proceedings of the International Conference on Learning Representations*, 2017.

[131] Wenliang Zhong and James Kwok. Fast stochastic alternating direction method of multipliers. In *International Conference on Machine Learning*, pages 46–54, 2014.

[132] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Object detectors emerge in deep scene cnns. In *ICLR*, 2015.

[133] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2921–2929, 2016.

[134] Bolei Zhou, Agata Lapedriza, Aditya Khosla, Aude Oliva, and Antonio Torralba. Places: A 10 million image database for scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[135] Bolei Zhou, Yiyou Sun, David Bau, and Antonio Torralba. Interpretable basis decomposition for visual explanation.

[136] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

APPENDICES

Appendix A: XNN

I. Human Evaluation

1. Comments from participants

In the printed questions that the participants took, they were also asked to optionally elaborate on their thoughts regarding the 2 AI agents (visualization approaches) and the overall course of the study. A few interesting comments would be referred in the following. One of the points that we noticed is that the participants found the overall task of grouping the given images difficult, especially in case of the seagulls in the CUB dataset. They were asked the question of how difficult they are finding the task of grouping images. From the answers scaled from 0 to 7, the mean value of 3.83 was obtained for the difficulty of grouping given images. As we indented to have a challenging task for non experts, the users elaborated this on their answers: {*Not many differences between the categories!*}, {*Many of the images looked very similar, so it would be hard to parse which picture went to which group*}, {*There was a lot of similarities throughout many of the images.*}, etc. Further, the overall performance of the participants was weaker in grouping the seagull images and this was noticed in the comments of the participants as well: {*I found it very difficult to distinguish the seagulls but I found it less difficult to group the cells.*}, {*Bird images were harder [to group] than cell images.*}, {*Some cells had dendritic outgrowths that made them obvious but seagulls looked very similar.*}, etc. This gives the intuition that since it was more plausible for the participants to find distinct features among the cell samples, they performed fairly as good as our approach in grouping the images even with no visualization. However, in the task of grouping the seagull images, as the samples were quite similar with no easy-to-find distinctive features, both the visualization methods helped the performance.

Algorithm 3 The metric based on Voronoi diagram

for each Neuron n of X layer in image I_m **do**
  for each Part p with its Voronoi graph G_p **do**
    for each Pixel $(i, j) \in G_p$ **do**
      Compute the distance between (i, j) and part label (i_p, j_p): $d_{ijp} = \left((i - i_p)^2 + (j - j_p)^2\right)^{\frac{1}{2}}$
    end for
    Normalize the distance d_{ijp} into $[0, 1]$, obtain the normalized distance $\bar{d}_{ijp}$
    for each Pixel $(i, j) \notin G_p$ **do**
      $\bar{d}_{ijp} = 1$
    end for
    $P(\text{Part}_p^m|\text{Pixel}_{i,j}^m) = 1 - \bar{d}_{ijp}$
    Compute the probability $S_p^{n,m} \triangleq P(\text{Part}_p^m|\text{Neuron}_n) = \sum_{(i,j)\in I_m} P(\text{Part}_p^m|\text{Pixel}_{i,j}^m)P(\text{Pixel}_{i,j}^m|\text{Neuron}_n) = \sum_{(i,j)\in I_m}(1 - \bar{d}_{ijp})S_{i,j}^{n,m}$.
  end for
end for

2. Participants' statistics

In the beginning of the study, the participants were asked to select their answers for general questions about their gender, age, their background in Computer Science, etc. and We will represent the recorded statistics from our 30 participants in the following. Age of the participants (in years): {'18-30': 76.7%, '31-40': 16.7%, '41-50': 3.3%, and '51-60': 3.3%}, Gender of the participants: {'Female': 43.3%, 'Male': 53.3%, and 'Trans Male': 3.3%}, Background (taken courses) of the participants in Computer Science: {'No course': 36.7%, '1 course': 36.7%, and '4 courses or more': 26.7%}, Background of the participants in Artificial Intelligence: {'No course': 76.7%, '1 course': 3.3%, '2 courses': 6.7%, '3 courses': 3.3%, and '4 courses or more': 10%}.

II. Results on the CUB dataset

1. Voronoi-based probability

In Algorithm 3 we introduce the inverse distance as a factor when computing the **Voronoi-based probability** $S_p^{n,m}$, trying to keep the balance between the large part region and the small part region.

Table A.1: The average faithfulness for Lasso with different α for 30 randomly selected categories of the CUB dataset.

Lasso	α	2.5	1.5	0.5	0.1
Num_x		8	21	68	232
F_{reg}	Training	3.80	3.06	1.86	1.00
	Testing	3.70	2.99	1.84	1.03

2. The faithfulness for Lasso

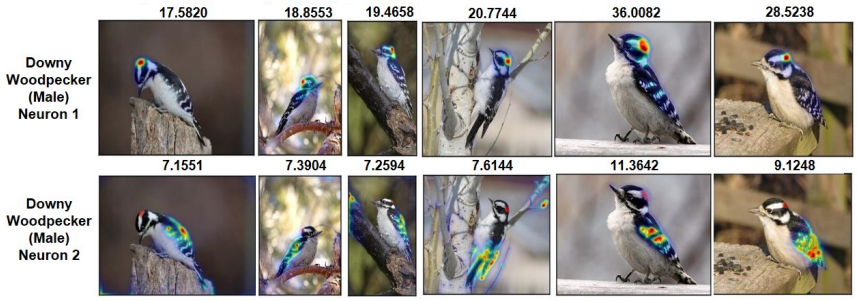
We summarize the faithfulness for Lasso using different parameter settings for 30 randomly selected categories of the CUB dataset in Table A.1, where α is the parameter that multiplies the L_1 term in Lasso, Num_x is the average number of the selected features for the 30 categories. From Table A.1 we observe that the faithfulness values for Lasso with different parameters are poor, indicating that it is almost impossible for the feature selection method to select few X-features directly from $\mathbf{Z}$ to make a faithful prediction.

3. Masking the image with generated heatmaps

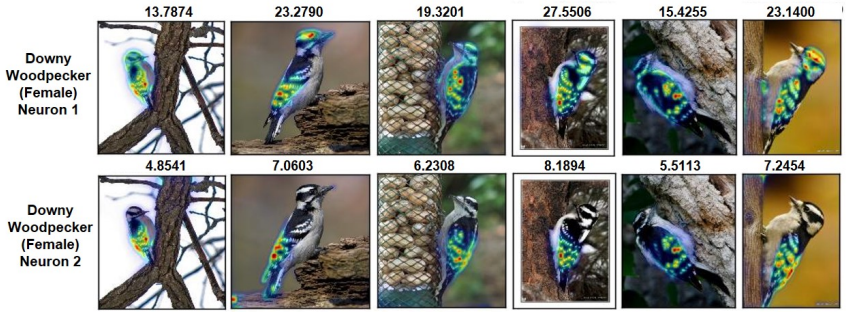
To further examine whether our proposed approach offers a complete explanation of the decision made by the CNN, we attempted to do classification just using the regions that are presented in the heatmaps, similar to [33]. First, we only keep top n pixels that have the highest response in the heatmap, and the rest of the pixels are painted as black (we keep the highlighted regions while masking the background, which is different from [33] where the highlighted regions are removed). Then, an inpainting algorithm is applied to recover the masked images, and finally, we utilize the prediction CNN to classify the recovered images and test the classification accuracy. Table A.2 shows results for the masking, inpainting, and classification task on 30 randomly selected categories of the CUB dataset. The results show that our method is slightly better than the baseline 1-Heatmap. But we would like to note that the capability of our method to generate separate concepts is non-existent in 1-Heatmap, which merges all concepts into a single heatmap.

Table A.2: The average classification accuracy for images masked by our method (XNN) and the baseline (1-Heatmap) with the same number of kept pixels on 30 randomly selected categories of the CUB dataset.

	Original image	Pixels kept	Mask by x-features	Mask by 1-Heatmap
Classification Accuracy	0.8798	300	**0.6767**	0.6713
		500	**0.7456**	0.7381
		1000	**0.8297**	0.8264



(a) Male downy woodpeckers

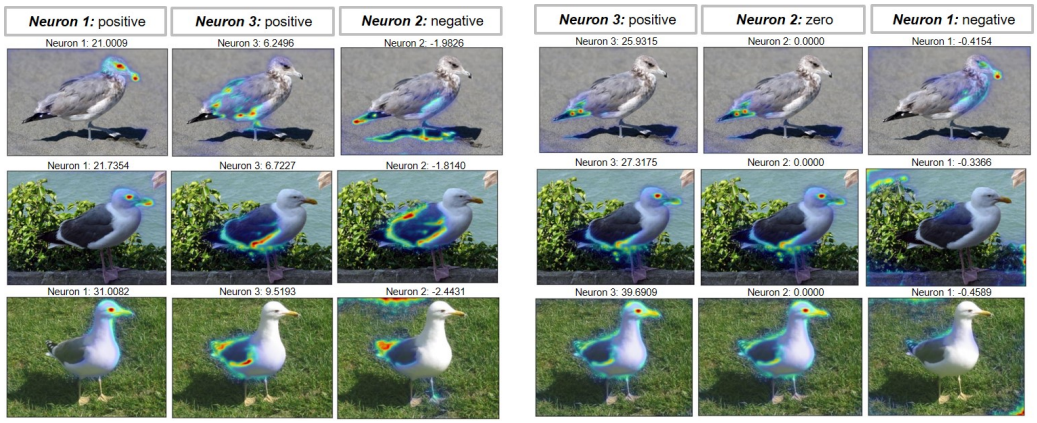


(b) Female downy woodpeckers

Figure A.1: The x-features for male and female downy woodpeckers.

4. Examples of degeneration

Figure A.2 shows some examples to illustrate the degeneration issue. Our propose method SRAE can avoid degeneration, and make the prediction model explainable.



(a) (b)

Figure A.2: (a) Good examples learned by SRAE, the number of the x-feature is 3, where the 3 neurons are orthogonal to each other; (b) Degenerated examples learned by NN, the number of the x-feature is 3, where the first two neurons are very similar, and there is only one positive neuron.

5. An interesting failure case

Figure A.1 shows the x-features for male and female birds of downy woodpecker, respectively. The difference between the male and female birds of downy woodpecker is that the male birds have a red spot on the head while the female birds do not. Hence, for male birds Neuron 1 in the explanation space captures the red spot; while for female birds Neuron 1 captures the stripes on the head and the body. Neuron 2 in the explanation space captures the strips on the body for both male and female birds of downy woodpecker. The results indicate that the x-features in the explanation space truly justify the classification decisions by capturing the key features of the birds, and the proposed model generates visualizations which are explainable to human. However, the orthogonality and locality on the female birds suffer, probably because the most indicative feature (Neuron 1) was only available in the males, hence the algorithm went on to pick some other features into Neuron 1 as well. Neuron 2 was, however, consistent in both the male and the female birds.

Table A.3: The average faithfulness for Lasso with different α on 10 categories of the Places dataset.

Lasso	α	15	10	5	2.5	1	0.1
Num_x		5	11	31	69	283	1782
F_{reg}	Train	4.0726	3.8078	3.3673	3.0566	2.2385	1.1677
	Test	4.3366	4.0214	3.5281	3.1655	2.3382	1.3131

III. Results on the Places dataset

1. The faithfulness for Lasso

For the Places dataset, we summarize the faithfulness for Lasso using different parameter settings for 10 categories in Table A.3, where α is the parameter that multiplies the L_1 term in Lasso, Num_x is the average number of the selected features for the 10 categories. From Table A.3, we observe that the faithfulness values for Lasso with small Num_x are all very poor for different parameters, indicating that it is almost impossible for the feature selection method to select few x-features from $\mathbf{Z}$ directly to make the prediction faithful on the Places dataset.

2. More qualitative examples between heatmaps on the original output $\hat{\mathbf{y}}$ (baseline: 1-Heatmap) and on x-features (our approach XNN).

Despite the hundreds of object labels, the x-features generated by the explanation module truly capture meaningful and consistent visual concepts on Places. Figure A.3 shows more qualitative examples between heatmaps on $\hat{\mathbf{y}}$ (baseline: 1-Heatmap) and on x-features (our approach XNN) for the Places dataset. One can see that the x-features capture several specific or general concepts, like *faucet & toilet*, *PC monitor*, etc. The visualization of the baseline 1-Heatmap falls on different objects in different images, while the visualizations of x-features are more consistently focusing on the same concepts in the images. From the results on both CUB and Places, we believe that the x-features learned by our proposed model indeed provide concise conceptual explanations of the

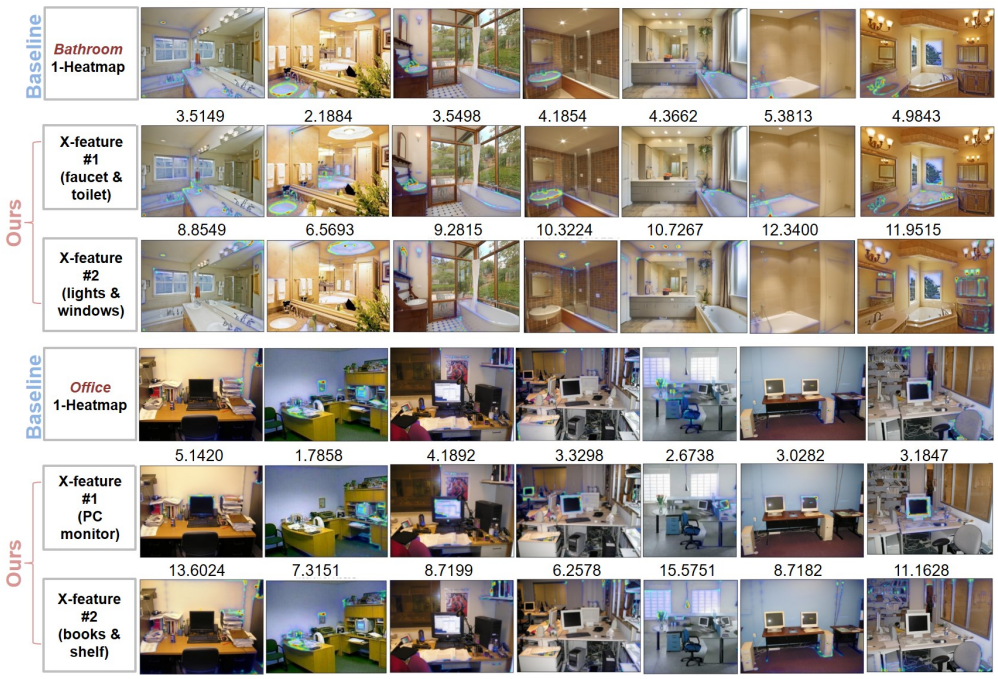


Figure A.3: Comparison of heatmaps on the original output $\hat{\mathbf{y}}$ (baseline: 1-Heatmap) and on x-features (our approach XNN) for Places.

decisions made by CNN algorithms.

Appendix B: DeepFacto

B. Classification on Fashion-Mnist

To compare our method with dlADMM [105], we evaluated the performance of our method on the Fashion-MNIST dataset [109] with 60,000 training samples and 10,000 testing samples. We followed the settings in [105] by having 2 hidden layers with 1000 neurons each, and Cross-Entropy loss at the final layer. Also, the batch size is set to 128, $\beta_t = 1$, and the updates for $\boldsymbol{Z}_t$ and Θ_t (eq. 6a) are performed 3 times at each epoch. Figure B.1 shows the test set accuracy results over 200 epochs of training. It can be noticed that Stochastic Block ADMM is converging at lower epochs and reaching a higher test accuracy while performing efficient mini-batch updates. Further, in section C., it will be demonstrated that Stochastic Block ADMM converges drastically faster than dlADMM on wall clock time.

Heat maps

To further investigate the interpretability of the factorized representations learned from DeepFacto, similar to [16], one can visualize the score matrix $\boldsymbol{S}$. Each dimension of the score matrix $\boldsymbol{S}$ can be reshaped back to the original activation size and be up-sampled to the size of the input using bi-linear interpolation. In Figure B.2, the score matrix learned form the DeepFacto with $r = 32$ (average attribute prediction of 81.4%) is used where three different heat maps (out of 32) are depicted over different samples from LFWA dataset. We have found $r = 4$ to be very low to represent interpretable heat maps for the attributes and $r = 256$ to contain redundant heat maps. It can be seen, that the heat maps can show local and persistent attention over different face identities: *eyes*, *forehead*, *nose*, etc.

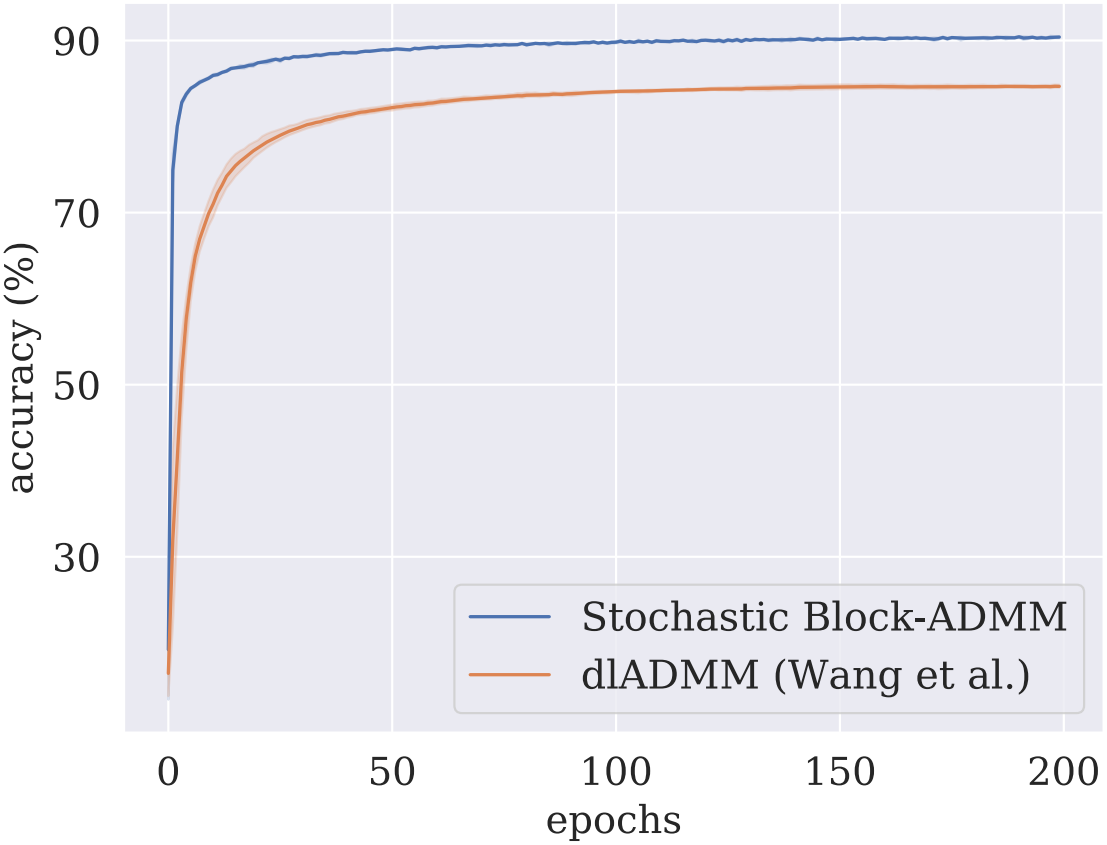


Figure B.1: Test accuracy comparison of Stochastic Block ADMM and dlADMM [105] on Fashion-MNIST dataset using a network with 3 fully-connected layers: $784-1000-1000-10$. Final test accuracy: "Stochastic Block ADMM": **90.39**%, "Wang *et al.*":84.67% (averaged over 5 runs).

Figure B.2: Heat map visualizations from three different dimensions of the score matrix $\boldsymbol{S}$ (rows) trained by DeepFacto-32 over different samples (columns) in LFWA dataset. These dimensions can capture interpretable representations over different faces identities: *eyes* (top), *forehead* (middle), and *nose* (bottom).