Efficient Deep Learning Models with Autoencoder Regularization and Information Bottleneck Compression

Jerome Oskar Williams

Supervisors: Prof. Gustavo Carneiro, Prof. David Suter, Dr. Michele Sasdelli

May 2019

Thesis submitted for the degree of MPHIL in Computer Science

SCHOOL OF COMPUTER SCIENCE

THE UNIVERSITY OF ADELAIDE AUSTRALIA

Declaration

I certify that this work contains no material which has been accepted for the award of any other degree or diploma in my name in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text. In addition, I certify that no part of this work will, in the future, be used in a submission in my name for any other degree or diploma in any university or other tertiary institution without the prior approval of the University of Adelaide and where applicable, any partner institution responsible for the joint award of this degree. The author acknowledges that copyright of published works contained within this thesis resides with the copyright holder(s) of those works. I give permission for the digital version of my thesis to be made available on the web, via the University's digital research repository, the Library Search and also through web search engines, unless permission has been granted by the University to restrict access for a period of time. I acknowledge the support I have received for my research through the provision of an Australian Government Research Training Program Scholarship.

Signed by student:

Date: 13 May 2019

Signed by supervisor:

Date: 10 May 2019

Abstract

Improving efficiency in deep learning models implies achieving a more accurate model for a given computational budget, or conversely a faster, leaner model without losing accuracy. In order to improve efficiency, we can use regularization to to improve generalization to the real world, and compression to improve speed. Due to the information-restricting nature of regularization, these two methods are related. Firstly we present a novel autoencoder architecture as a method of regularization for Pedestrian Detection. Secondly, we present a hyperparameter-free, iterative compression method based on measuring the information content of the model with the Information Bottleneck principle.

Acknowledgements

First I would like to thank my supervisors, Prof. Gustavo Carneiro and Dr. Michele Sasdelli of the Australian Institute of Machine Learning, and Prof. David Suter, currently at Edith Cowan University, for their patience, encouragement and support through the course of this project. My colleagues in the Australian Institute of Machine Learning, Dr. Saroj Weerasekera and Ergnoor Shehu, were unstintingly generous with advice and discussion relating to CUDA programming and information theory respectively. Thanh T. Nguyen of the Ulsan National Institute of Science and Technology was very helpful in his correspondence, answering questions about his Parametric Information Bottleneck method. This project was supported by Industry Linkage Project LP130100521, "Intelligent Collision Avoidance System for Mobile Industrial Platforms", obtained by Prof. David Suter (then at the University of Adelaide), and Prof. Ali Bab-Hadiashar of the Royal Institute of Technology, Melbourne.

My family and friends have been incredibly supportive of me during this period. Christopher, Emelia, June, Naomi and Therese Williams, Benjamin Lancer, Aaron Leung and Andrew Vawser have all helped to keep me going. I would also like to thank Dr. Zygmunt Ladyslaw, Prof. Wojciech Chojnacki, Prof. Brad Alexander, Dr. Adrian Johnston and Dr. John Bastian for many interesting hours of discussion. Special thanks goes out to Profs. Tat-Jun Chin from the School of Computer Science and Gerard O'Brien and Jon Opie of the Department of Philosophy for stimulating my interest in artificial intelligence.

Contents

Citation Listing of Included Publications

1. Williams J, Carneiro G, Suter D. *Region of Interest Autoencoders with an Application to Pedestrian Detection.* In 2017 International Conference on Digital Image Computing: Techniques and Applications (DICTA) 2017 Nov 29 (pp. 1-8). IEEE.

Chapter 1

Introduction

Deep Learning models are a new and powerful element in the machine learning toolkit. Great advances have been made in recent years applying deep learning to computer vision problems such as object classification, detection and semantic segmentation. Prior methods for classification operated as two-stage process: a feature transform such as the Scale Invariant Feature Transform (SIFT)[67] or the Histogram of Oriented Gradients (HOG)[68] would extract useful information from the data, and these features would be used as inputs for a classifier such as a Perceptron, Support Vector Machine or Decision Tree. Features are parts of the image that are distinctive and convey information about it. The classic examples are edges and corners. To a human eye, a sketch of an object is often sufficient for a viewer to classify it. By extracting features rather than using raw image data, the problem is made much easier by throwing away irrelevant information. Classifiers use a training set of labelled examples to find a function that divides the input space. This is done by solving an optimization problem to minimize the probability of incorrectly classifying samples from the training set. This function can then be applied to predict the class of new examples without labels. 'Deep Learning' refers to the use of neural network models with many layers. The final layer corresponds to the classifier, and the preceding layers correspond to feature extractors. The difference with respect to the prior methods mentioned above (e.g., SIFT, HOG) is that the entire model is optimized as a whole, so feature transforms can be 'learned' rather than 'hand-crafted'. Furthermore, each layer operates as a feature transform on the previous layer, providing features of other features. The Convolutional Neural Network (CNN) and the Rectified Linear (ReLU) activation function were the key algorithmic advances that allowed these deep networks to be trained. However, the deep layers of features that give deep learning models good performance are also computationally expensive for large models. The current widespread use of CNNs is due to the use of graphics processing units (GPUs) to compute features in parallel, where CUDA, developed for nVidia's GPUs, is the most common API.

With the ability to train large models and achieve good performance on 'real world' computer vision tasks, there is now a demand for models that can be deployed on cheaper hardware or process a video feed in real time (for example, Pedestrian Detection for self-driving cars). In this thesis we explore two methods for utilizing computing resources more efficiently when developing deep learning models: *regularization* and *model compression*. Regularization improves model accuracy by reducing overfitting, the tendency of a model to perform well on its training set but worse on a withheld testing set. A better-regularized model will perform better on the testing set for the same computational cost. Model compression reduces the size of the model in memory, and depending on the implementation this can reduce the time taken to compute the features. A good compressed model will have less computational cost but the same accuracy. Using either of these strategies, better performance can be obtained for a given computational budget. A large model can be trained and later compressed to a smaller size, or a smaller model can be trained with a regularizer so that it achieves better performance with the resources available.

Not only are regularization and model compression both methods for achieving better models for a given computational budget, they are also ultimately based on the same mechanism: selec-

tively constraining the information stored in the model about the data. Given enough flexibility, any machine learning model can simply memorize the training data rather than modelling the data-generating distribution. Rather than storing properties of individual examples we find the strongest factors of variation within the dataset and create a feature transform to convert inputs into feature vectors. Information not expressed in these feature vectors is thrown away. While the feature vectors are simpler than the input, information must also be carried by the model parameters. A feature with a simple output and many parameters can still overfit, thus there exist regularization methods that address information carried by feature vectors and methods that address information carried by the model parameters. Model compression is obviously assisted if there are fewer parameters that need to be stored, and to reduce memory usage at runtime it is also helpful to restrict the size of feature vectors. Just like the regularizers, there are model compression algorithms that operate on the model parameters, and ones that operate on the feature vectors. The key difference is that in model compression algorithms the format of the information is critical. Simply reducing the number of channels in a convolutional layer's output, for instance, carries no overhead when inference is done on the compressed model. Eliminating model weights rather than channels requires a sparse matrix storage to take advantage of, and it is possible to find methods for compression, such as Deep Compression [53], in which the model must be decompressed for inference in the absence of hardware support. Unfortunately the practical success in the field of machine learning has (for now) outstripped our theoretical analysis. Terms like 'flexibility', 'capacity', 'dimensionality' and 'information content' are used in a non-rigorous, qualitative fashion, simply because while we see their effects all the time, we lack a way to quantify the information expressed in a high dimensional, nonlinear model like a deep neural network. However, theoretical frameworks such as the Minimum Description Length [73][49] and Information Bottleneck Method [35] both point to methods to quantify, if only approximately, just how flexible our models are, and where the information stored in them comes from.

In this thesis, we propose two methods to improve the efficiency of deep learning models, one by regularization and one by compression. In Chapter 2 we provide an introduction to regularization and compression. In Chapter 3, we propose a special type of autoencoder, the Region of Interest Autoencoder, to regularize a pedestrian detection problem and improve performance without increasing the model size or evaluation time. In Chapter 4, we demonstrate a new model compression algorithm, PIBprune, that uses a variant on the Information Bottleneck method, the *Parametric* Information Bottleneck, to estimate the information content of deep learning models. Information Bottleneck methods were originally developed as regularizers to improve generalization, but we show how they can also be used for compression. Our PIBprune algorithm conservatively removes nodes with low information content, while preserving nodes with high information content to achieve smaller, more efficient models while retaining performance.

Chapter 2

Background

2.1 Regularization

A regularizer is an addition to the training program to prevent overfitting and improve the generalization of a model so that it performs well on new examples. Regularizers are usually associated with some hyperparameter that controls the strength of the regularization. If the regularization is too weak, the model will overfit and 'remember' the training data rather than modelling it in a way that generalizes well. If the regularization is too strong, the model *underfits* instead, and is not flexible enough to model the training data accurately. This is a feature that regularization shares with architecture selection; a model that is too large may overfit, while a model that is too small may underfit. Rather than a hard limit on dimensionality, regularizers use penalties, soft constraints or auxiliary tasks during training. Instead of guessing at an ideal architecture, a large model is usually chosen and the regularizer is left to constrain the flexibility of the model during training.

2.1.1 Weight Decay

The classic example in deep learning and other neural network methods is weight decay [78], which adds a penalty to the cost function in the training program based on the magnitude of the parameters in the model. The most frequently used version is L2 weight decay or Tikhonov regularization,

$$\mathcal{L}_{regularized} = \mathcal{L}_{cls} + \lambda||\mathbf{W}||_2^2, \tag{2.1.1}$$

where $\mathbf{W}$ is the set of weights in the model, $\mathcal{L}_{cls}$ is a classification loss function dependent on $\mathbf{W}$ and λ is a hyperparameter controlling the strength of the regularization. Because it depends on the squared norm, L2 weight decay tends to heavily penalize large magnitude parameters and encourage small but non-zero parameter magnitudes. Also used is the L1 or Lasso (Least Absolute Shrinkage and Selection Operator) regularizer,

$$\mathcal{L}_{regularized} = \mathcal{L}_{cls} + \lambda||\mathbf{W}||_1, \tag{2.1.2}$$

which penalizes all weights according to their magnitude rather than their squared magnitude, and tends towards a sparse model with many zeroed parameters. During training, weight decay introduces a trade-off between the accuracy of the model on the training set, and the complexity of the function. Out of several models that fit the data equally well, a regularized training will produce the simplest model. Finding the simplest model as per Occam's Razor is a key principle of regularization, but there are many ways of forcing a model to be 'simpler'.

2.1.2 Dropout and Dropconnect

Instead of penalizing the weights, Dropout [64][65] tries to prevent co-adaptation of nodes. To be robust, a node should not be dependant on any single input. Nor should it be dependant on

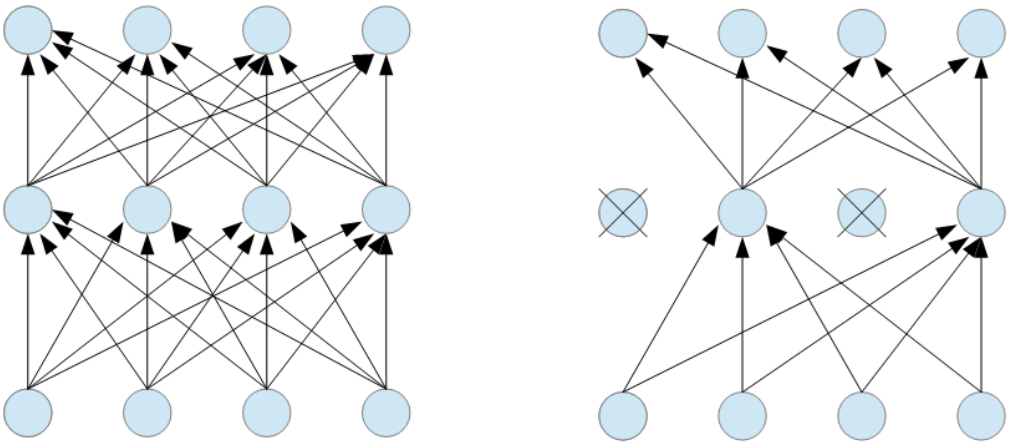


Figure 2.1.1: Dropout: nodes are randomly dropped from the affected layer at training time, creating a new sparse architecture for each training step.

any two inputs being available at the same time. Rather, all inputs should be useful in some way. The authors of Dropout use an analogy to sexual reproduction in nature. The child organism has a mix of genes from both parents, and genes that are useful in many different combinations are favoured. To encourage this behaviour, dropout randomly zeroes out the outputs of nodes in the affected layer. During training, a neural network layer with m inputs and n nodes, weights $\mathbf{W}$ and bias $\mathbf{b}$ will have a hyperparameter p, that controls the likelihood of nodes being retained. Given p we can sample a binary mask for the nodes $\mathbf{r} \sim \text{Bern}\,(p)$, where Bern(.) denotes the Bernoulli distribution.

For any training step, an expected fraction p of the nodes with dropout will be active, the remaining being inactive. An example of dropped out nodes can be seen in in Figure 2.1.1. This means that during each training step, the network will have different connectivity and effectively a different sparse architecture. During training, a neural network layer with weights $\mathbf{W}$, bias $\mathbf{b}$ and dropout rate p will have activations

$$\mathbf{z} = \mathbf{r} \odot \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}). \tag{2.1.3}$$

During testing we use the expected value of this process. All the nodes are active but weighted by a factor of $\frac{1}{p}$. This gives us

$$\mathbf{z} = \frac{1}{p}\sigma(\mathbf{W}\mathbf{x} + \mathbf{b}). \tag{2.1.4}$$

Dropconnect [66] is a similar method that zeroes out *input connections* to nodes rather than outputs as shown in Figure 2.1.2, resulting in more possible architectures than Dropout. With Dropout and Dropconnect, the activations of nodes in the network are stochastically modified to train the model to cope with missing data and different activation patterns for the same input data. For Dropconnect we also use a hyperparameter p, but the equations are altered so that the masking happens on the inputs to the node. The mask becomes a matrix $R \sim \text{Bern}(p)$ of size $m \times n$. During training the activation function is

$$\mathbf{z} = \sigma\left(\Big(\mathbf{R} \odot \mathbf{W}\Big)\mathbf{x} + \mathbf{b}\right), \tag{2.1.5}$$

with

$$\mathbf{z} = \sigma\left(\Big(\frac{1}{p}\mathbf{W}\Big)\mathbf{x} + \mathbf{b}\right) \tag{2.1.6}$$

during testing.

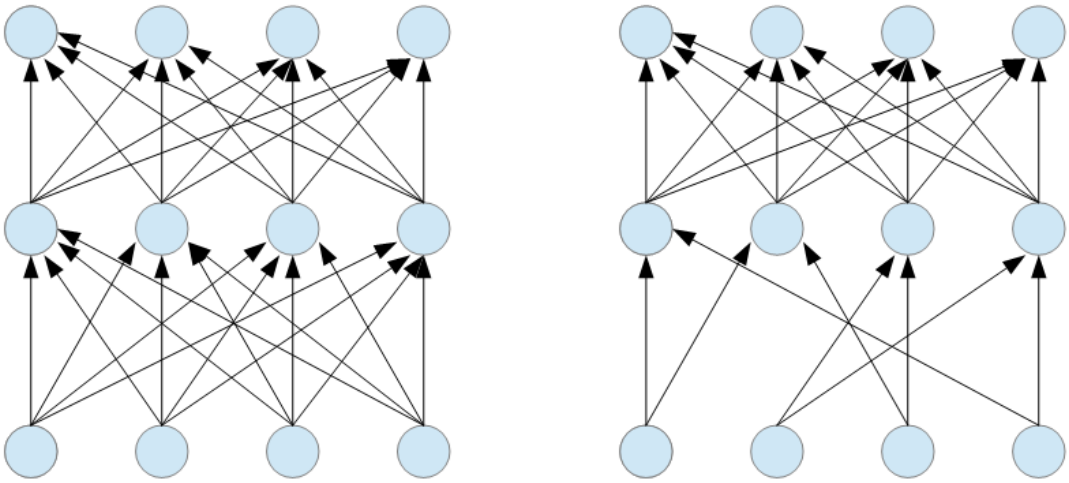


Figure 2.1.2: Dropconnect: input connections are randomly dropped from the affected layer at training time, creating a new sparse architecture for each training step.

2.1.3 Semi-Supervised Learning

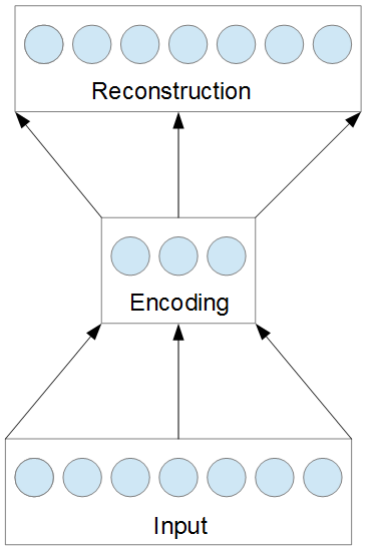


Figure 2.1.3: A simple autoencoder: a high dimensional input is mapped to a low dimensional input and reconstructed from its encoding.

Semi-supervised learning attempts to regularize a supervised learning problem such as classification or detection by combining it with an unsupervised task. This can be performed with or without extra unlabelled data. Restricted Boltzmann Machines (RBMs) [81][82] and Autoencoders [86][89][91] are trained to learn an encoding function to encode data into a (constrained) latent space and a decoding function to reconstruct the original from the encoding as in Figure 2.1.3.

For our purposes we focus on autoencoders, which are simpler than RBMs and can be jointly trained with a supervised neural network model. A simple autoencoder (see Figure 2.1.3) has an input space $\mathbf{x} \in \mathbb{R}^m$, and an encoding space $\mathbf{z} \in \mathbb{R}^n$. A neural network layer is used to define a

deterministic mapping from the input to the encoding space,

$$\mathbf{z} = f_\theta(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}), \tag{2.1.7}$$

where σ is the activation function and the mapping function f is parameterized by $\theta = \{\mathbf{W}, \mathbf{b}\}$. $\mathbf{W}$ is the encoding layer's weight matrix with size $m \times n$ and $\mathbf{b}$ the $n \times 1$ bias vector. We then use a decoding layer to attempt to recover an approximate $\mathbf{x}$ from the encoding $\mathbf{z}$. We designate the reconstruction $\mathbf{x}'$. The mapping from the encoding space back to the input space is given by

$$\mathbf{x}' = g'_\theta(\mathbf{z}) = \sigma(\mathbf{W}'\mathbf{z} + \mathbf{b}'), \tag{2.1.8}$$

where g is parameterized by $\theta' = \{\mathbf{W}', \mathbf{b}'\}$. $\mathbf{W}'$ is the decoding layer's weight matrix with size $n \times m$ and $\mathbf{b}'$ is its $m \times 1$ bias vector. The loss function is a simple quadratic loss

$$\mathcal{L}(\mathbf{x}, \mathbf{x}') = ||\mathbf{x} - \mathbf{x}'||_2^2. \tag{2.1.9}$$

The loss function depends on the difference between the reconstruction and the original. If this is combined with a supervised task, the learned features have to satisfy being useful for the supervised task, while being able to efficiently model the data distribution well enough to reconstruct examples. Because the problem is unsupervised, factors of variation within classes, not just between classes, are represented in the feature space. Information that is *not* required to represent these variations is discarded. In order to learn interesting features about the data, we need to constrain the latent space, where in a basic autoencoder this constraint takes the form of dimensionality reduction. Without such a constraint, the model could learn the identity function or a similar trivial mapping. To learn good features, only information necessary for a good reconstruction should be preserved.

To use these auxiliary tasks as a regularizer, either pre-training or joint training can be used. Pre-training was used to construct Deep Belief Networks [83] (DBNs) and Deep Boltzmann Machines [84] (DBMs) out of RBMs, and later for autoencoder-based semi-supervised learning such as the Stacked Denoising Autoencoder [89] and Stacked What-Where Autoencoder [91]. As training techniques have improved, joint training has superseded pre-training, especially on convolutional networks [91][92]. Pre-training trains a series of layers, one at a time, with each layer using the previous layers as feature transforms. A classification layer is then added to the end of the network and the whole network trained with the supervised loss function. The assumption in pre-training is that the unsupervised objectives will find a useful 'basin of attraction' in the parameter space in which the supervised task will be confined. If the training for the supervised task is too good, it could escape the basin. On the other hand, joint training uses a combined loss function that adds the loss functions for the unsupervised tasks in all layers to the supervised task, optimizing for all objectives at the same time. This allows the unsupervised tasks to constrain the supervised task throughout the entire training process.

Whereas a basic autoencoder simply encodes the input into a low-dimensional latent space, a denoising autoencoder [88] can work even if the latent space has more dimensions than the input space. Denoising autoencoders corrupt the input $\mathbf{x}$ to get a noisy input $\tilde{\mathbf{x}}$ and then attempt to reconstruct the uncorrupted original. The encoding mapping becomes

$$\mathbf{z} = f_\theta(\tilde{\mathbf{x}}) = \sigma(\mathbf{W}\tilde{\mathbf{x}} + \mathbf{b}), \tag{2.1.10}$$

with $\tilde{\mathbf{x}} \sim (\mathbf{x} + \mathcal{N}(0, k))$, where k is a hyperparameter that controls the amount of corruption. An analysis by Alain and Bengio [90] shows that denoising autoencoders implicitly model the probability density function of the data and its gradient (see Fig. 2.1.4). Inputs that are in low density areas in the input space are shifted towards higher density areas in the reconstruction, and in the latent space. This means that inputs that have not been seen before will be transformed so they are closer in the latent space to images that have been seen before. When data points are close in the latent space, the supervised task will treat them similarly, all other things being equal. The

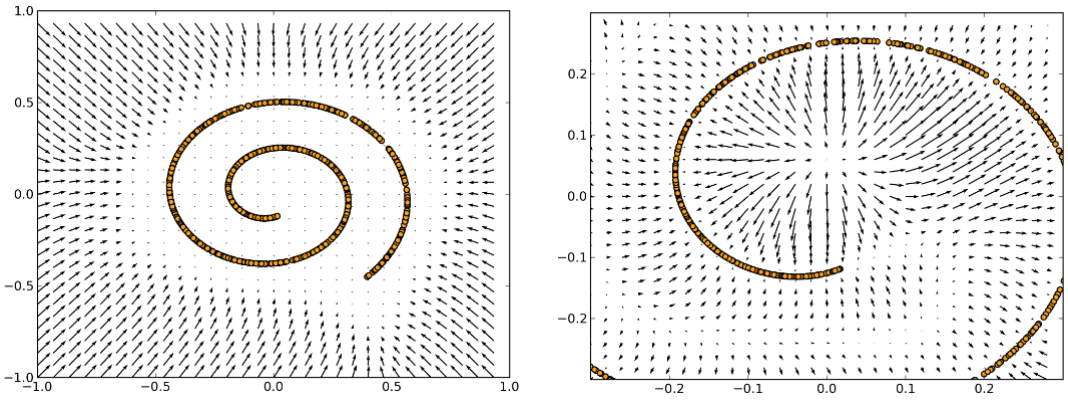


Figure 2.1.4: Figure 5 from Alain and Bengio's analysis [90] demonstrates the effect of a denoising criterion as a field. Reconstructed points move closer to the areas with high data density.

failure modes for semi-supervised learning are when the reconstruction task is so constrained that it is unable to model the data effectively, or so unconstrained that it becomes equivalent to the identity transform. In joint training the influence of the unsupervised and supervised tasks needs to be balanced so that the supervised task is regularized but not to the point of underfitting.

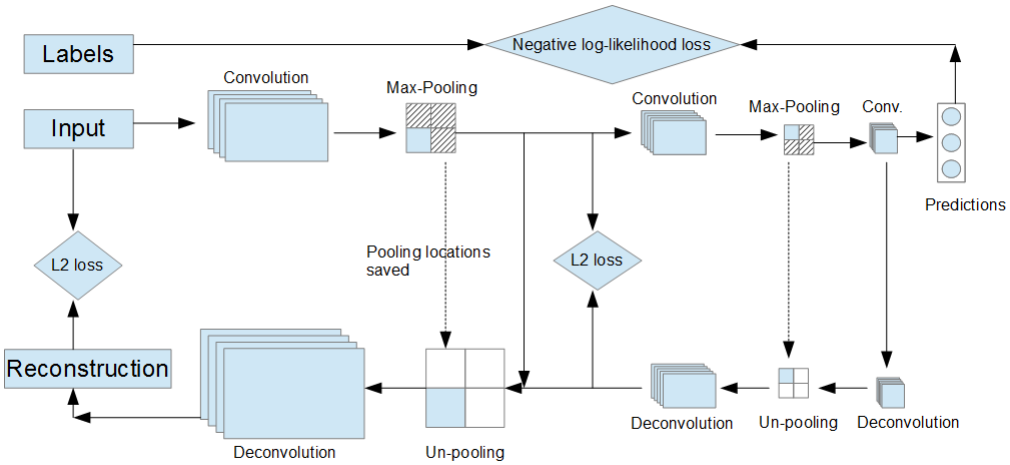


Figure 2.1.5: Stacked What-Where Autoencoder. A convolutional network is trained jointly on a supervised task and multiple reconstruction objectives to assist in learning features that generalize. The pooling operator is partially inverted by saving the spatial locations or 'pooling switches' chosen for max-pooling.

While weight decay and Dropout have become standard practice in engineering deep learning models, most autoencoder work has been with fully-connected layers and on small input images. Scaling it up had two main issues: firstly, it has to be made compatible with convolutional networks, and secondly, reconstructions become harder with larger inputs. In 2015 a method for adapting autoencoders for use with CNNs was devised – this method is called the Stacked What-Where Autoencoder[91] (see Figure 2.1.5). This method uses a stack of convolutional autoencoders that are jointly trained with a supervised loss, with a special method called 'unpooling' to recover information from the max-pooling operation. Recently it was demonstrated [92] that joint training with the Stacked What-Where Autoencoder could improve the performance of a state-of-the-art CNN on Imagenet, with 224x224 colour images. In Chapter 3 we adapt this method for our own purposes for a detection problem and further demonstrate the power of semi-supervised methods to

improve convolutional networks. Like weight decay and Dropout, semi-supervised learning imposes no overhead on the finished model; it is as fast as an un-regularized model but more accurate on unseen examples. We have presented an overview of regularization, methods to improve model accuracy on new examples without using more resources. To see how we can make good models smaller without losing accuracy we turn to model compression methods.

2.2 Model Compression

While regularization involves taking a large model and limiting its capacity it so that it can't overfit to the training data, model compression limits a model's capacity so that it is faster and takes up less storage space, while retaining its accuracy. The key difference between a regularization and compression algorithm is that to the compression algorithm it is the number of weights and nodes, and the dimensionality of the activations that matter rather than the flexibility of the parameters. A critical element in most model compression algorithms is allowing the model to adapt to the changes caused by the compression process. Because of the complexity of deep learning models it is usually easier to fine tune a compressed model using gradient descent than it is for the compression algorithm to adjust the compressed parameters. Because the fine-tuned model may have very different parameters, it might even be possible to compress the model even more. This points to an iterative process of compression and fine tuning that we will explore in Chapter 4.

2.2.1 Weight Pruning

The weight decay method for regularization has a counterpart in the compression method of weight pruning. Weight pruning exploits the fact that many weights in neural network models are close to zero, so that removing the weights will only result in a small change to the model's predictions. A simple weight pruning method simply removes weights with a magnitude below a certain threshold. If L1 weight decay was used as a regularizer in training the model, weight pruning is already built in, because many weights will have been forced to 0. This is one example of regularization and compression overlapping when it comes to restricting the capacity of the model. L1 weight decay thus provides model compression without any drop in model accuracy - and indeed a possible increase in accuracy over an unregularized, uncompressed model. However, with weight pruning *and* fine tuning, L2 weight decay provides better results - more compression for the same accuracy [41]. A better regularized model will tend to have more robust features that can cope with the changes induced by compression, just as they can generalize to unseen samples.

Weight pruning was further developed by more principled algorithms such as Optimal Brain Surgeon (OBS)[57]. These measure the impact that changes in each weight have on the final prediction of the model. They do this by computing the Hessian matrix of the weights with respect to the loss function. However these methods do not scale well. Weight pruning can be used to speed up model inference, but this requires the exploitation of sparse matrix-vector operations which are not available in most machine learning toolkits.

2.2.2 Quantization and the Deep Compression Method

While weight pruning relies on sparsifying the weights in the model, quantization reduces the number of bits necessary to represent each weight. Rather than setting weights to zero, a small codebook of values is generated and used to approximate the values of all the weights. Quantization has been combined with weight pruning in Deep Compression [53] to great effect. Rather than set weight values to zero they are set to their closest approximation in the codebook. They then use Huffman Coding, a type of lossless run-length encoding, to take advantage of the reduced number of unique values in the quantized model. This method relies on iteratively compressing each layer and then fine tuning the model to adapt to the changes. Using this, it is possible to compress a model to use very little storage, but the storage necessary to store intermediate results is unaffected. The authors do report an increase in speed, but this is primarily due to the model being small enough to store in the cache. The model cannot be used for inference when

fully compressed, and has to be expanded by reversing the Huffman Coding and quantization phases to yield a sparse network. The developers of Deep Compression treat this as a hardware issue and proposed the Efficient Inference Engine (EIE) to perform inference on the quantized model.

2.2.3 Tensor Decomposition

To compress models in order to obtain an increase in speed we need to be able to perform inference efficiently on the compressed model. For this purpose, we can use tensor decomposition or node pruning. Tensor decomposition involves converting one (linear) layer in the model to two layers which are smaller than the original when combined. For this purpose we can take advantage of existing matrix and higher order tensor decompositions, especially the truncated Singular Value Decomposition (SVD). For fully-connected layers, the SVD is simple and effective. The $m \times n$ weight matrix $\mathbf{W}$ is factorized to

$$\mathbf{W} = \mathbf{U}\Sigma\mathbf{V}^\top, \tag{2.2.1}$$

where Σ is a diagonal matrix containing the singular values of $\mathbf{W}$, $\mathbf{U}$ and $\mathbf{V}$ are unitary square matrices containing the left- and right- singular vectors respectively, with $\mathbf{U}$ having shape $m \times m$ and $\mathbf{V}$ having shape $n \times n$. The truncated SVD uses the k largest singular values and the corresponding singular vectors in $\mathbf{U}$ and $\mathbf{V}$ to obtain an approximate transform

$$\mathbf{W}' = \mathbf{U}'\Sigma'\mathbf{V}'^\top, \tag{2.2.2}$$

where Σ' is now a $k \times k$ diagonal matrix, $\mathbf{U}'$ is an $m \times k$ matrix containing the k most significant left-singular vectors of $\mathbf{W}$, and $\mathbf{V}$ is an $n \times k$ matrix containing the k most significant right-singular vectors. For an input vector $\mathbf{x}$, instead of the activation $\mathbf{W}\mathbf{x} + \mathbf{b}$, we have $\left(\mathbf{U}'\Sigma'(\mathbf{V}'^\top\mathbf{x})\right) + \mathbf{b}$. The efficiency in terms of saved weights is dependant on the size of k. For an $m \times n$ weight matrix, there are mn weights. Using truncated SVD there are $mk + kn$ weights. For a square matrix $m = n$, k must be less than $\frac{m}{2}$ for the decomposed transform to use fewer weights. The values of Σ can simply be multiplied into $\mathbf{U}'$ and do not need to be saved. Because convolutional neural networks use 4-dimensional convolutional kernels rather than 2-dimensional weight matrices higher order tensor decompositions need to be applied. Examples include the Higher Order SVD, Tucker [70] [93] and CP [94] decompositions.

2.2.4 Node and Channel Pruning

To mitigate compression-related overhead at test time, we can use node pruning and simply shrink the architecture by removing nodes from the network. The compressed model has the same number of layers, but fewer nodes in each. The issue here is preserving the accuracy in the compressed model. Nodes are much harder to remove than weights because their output is linked to each node in the next layer. The success of Dropout seems to indicate that neural network models *can* compensate the loss of nodes to some extent, but inference on Dropout at test time still relies on all nodes being active. It is harder to prune channels than fully-connected nodes, but most of the memory usage in deep learning models comes from convolutional layers, despite the fully-connected layers having more parameters, so it is still potentially useful. One approach for node pruning, similar to L1 weight decay and weight pruning, is to introduce a penalty on the number of nodes into the training process and let the optimizer adapt the model as needed. L1 weight decay can be extended to groups of weights in the Group Sparse Lasso regularizer. Let $\mathbf{W}$ be an $m \times n$ weight vector,

$$\mathcal{L}_{regularized} = \mathcal{L}_{cls} + \lambda_1 \sum_{i=1}^{m} ||\mathbf{W}_i||_2 + \lambda_2 ||\mathbf{W}||_1, \tag{2.2.3}$$

where $\mathbf{W}_i$ is the row vector in $\mathbf{W}$ corresponding to the weights attached to the i-th node. By grouping the weights by node, weight pruning by L1 can be converted into node pruning, where all the weights in a node tend to be driven to 0 at once. When node-pruning is applied to convolutional layers it becomes channel pruning [63]. Louizos et al. [51] use L0 regularization for this

purpose, which penalizes the *number* of non-zero weights rather than their magnitudes. Dropout derived methods [62] can also be used. These include trainable gates in the model that can reduce or expand the effective number of nodes at runtime.

2.2.5 Choosing the Compression Rate

Like regularization, compression usually requires selecting a hyperparameter to specify how constrained the model will be. In the case of compression, the ideal compression rate may vary significantly between layers depending on the architecture. Hyperparameter selection can be a tedious process, so much so that efficient search methods for good hyperparameters has become a research field in its own right [80]. An ideal compression rate is one that shrinks the model without removing information necessary to classify the model. We know that deep neural network models are massively overparameterized, which makes training easier, but which damages the models. Regularizers allow large numbers of parameters while restricting the information they can store about the training data to prevent the model from overfitting. Empirically, we can find compression rates for existing compression algorithms that shrink the model without reducing its accuracy. If we can quantify the proportion of the model that contains useful information, we can specify a compression rate that allows useful information to remain. The dimensionality and the VC dimension are inadequate tools for deep neural networks, but in recent years new and powerful theoretical tools using the Minimum Description Length (MDL) [73] and Information Bottleneck (IB) [35] have been developed. These can be applied to quantify the information present in the model. Comparing the information present to the maximum amount of information that can be stored in the model should yield a good compression rate without treating the model as a black box and using a trial-and-error approach.

In the last few years, advanced node pruning methods have been proposed that are motivated motivated by the Minimum Description Length and Information Bottleneck. Recently the VIBnet [45] was proposed that also uses the IB method, and Bayesian Compression [50], which uses Group Sparsity combined with the MDL. Interestingly, both the Information Bottleneck and Minimum Description Length were first applied to neural network as regularizers, to find principled ways to improve generalization. We are surprised to find that despite using theoretical approaches that can quantify the information used by the model, there have been very few attempts to find a hyperparameter-free model compression algorithm. In Chapter 4 we present our node pruning method, PIBprune, which is derived from the Information Bottleneck method.

Statement of Authorship

Title of Paper	Region of Interest Autoencoders with an Application to Pedestrian Detection
Publication Status	■ Published ☐ Accepted for Publication ☐ Submitted for Publication ☐ Unpublished and Unsubmitted w ork w ritten in manuscript style
Publication Details	Jerome Williams, Gustavo Carneiro and David Suter, *Region of Interest Autoencoders with an Application to Pedestrian Detection*, in proceedings of the 2017 International Conference in Digital Image Computing:Techniques and Applications (DICTA)

Principal Author

Name of Principal Author (Candidate)	Jerome Williams		
Contribution to the Paper	Formulating the algorithm, implementating the algorithm, performing experiments and drafting the paper		
Overall percentage (%)	80%		
Certification:	This paper reports on original research I conducted during the period of my Higher Degree by Research candidature and is not subject to any obligations or contractual agreements with a third party that would constrain its inclusion in this thesis. I am the primary author of this paper.		
Signature		Date	09/05/2019

Co-Author Contributions

By signing the Statement of Authorship, each author certifies that:

i. the candidate's stated contribution to the publication is accurate (as detailed above);

ii. permission is granted for the candidate in include the publication in the thesis; and

iii. the sum of all co-author contributions is equal to 100% less the candidate's stated contribution.

Name of Co-Author	Gustavo Carneiro		
Contribution to the Paper	Supervised the work, provided feedback and helped edited the manuscript		
Signature		Date	18-04-2019

Name of Co-Author	David Suter		
Contribution to the Paper	Supervised the work, provided feedback and helped edited the manuscript		
Signature		Date	08/05/19

Please cut and paste additional co-author panels here as required.

Chapter 3

Pedestrian Detection Augmented by Autoencoders

An earlier version of this chapter was published as a conference paper in the proceedings of DICTA 2017.

Region of Interest Autoencoders with an Application to Pedestrian Detection

We present the Region of Interest Autoencoder (ROIAE), a combined supervised and reconstruction model for the automatic visual detection of objects. More specifically, we augment the detection loss function with a reconstruction loss that targets only foreground examples. This allows us to exploit more effectively the information available in the sparsely populated foreground training data used in common detection problems. Using this training strategy we improve the accuracy of deep learning detection models. We carry out experiments on the Caltech-USA pedestrian detection data set and demonstrate improvements over two supervised baselines. Our first experiment extends Fast R-CNN and achieves a 4% relative improvement in test accuracy over its purely supervised baseline. Our second experiment extends Region Proposal Networks, achieving a 14% relative improvement in test accuracy.

3.1 Introduction

The detection of visual objects is one of the most studied problems in computer vision [1]. A particularly relevant example of this problem is the detection of pedestrians, which is an important task in the self-driving car industry [7]. With rapidly improving hardware and increasingly large annotated data sets, we are able to apply powerful machine learning techniques to real-world detection problems. The main methodology being explored for the task of pedestrian detection is based on deep learning models [17][10][4][11], where the main challenge lies in the adaptation of such models to the unique setup of the data sets available for training.

Deep learning models use machine learning to simultaneously learn features that represent useful characteristics of the data, and a classifier to distinguish between classes. data sets include a *training set* that the model learns from, and a *testing set* that is used to test the model's accuracy on new data. A model's accuracy on the training set measures how successful the training process was at minimizing its cost function. Accuracy on the testing set measures how well the model generalizes to new examples, and indicates how well the model will do when deployed. Improving deep learning models can be done by addressing training or generalization. Better training will improve the training accuracy, but this is only useful if the testing accuracy improves with it. Better generalization closes the gap between training and testing accuracy; this is useful as long as such a gap exists. In both cases the key measurement of success is accuracy on the testing set.

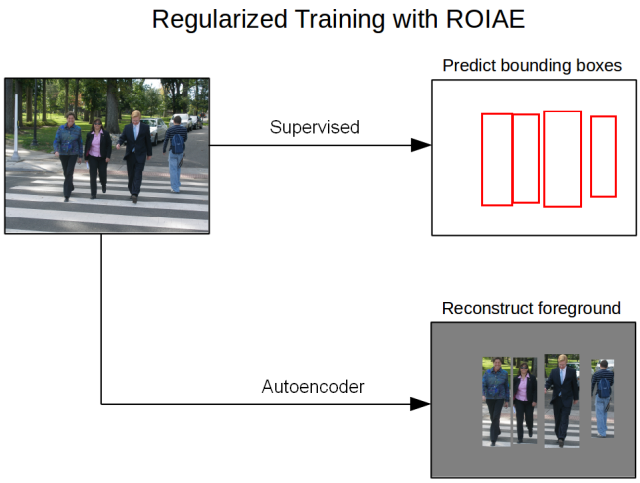


Figure 3.1.1: The Region of Interest Autoencoder combines supervised and reconstruction tasks.

Pedestrian detection data sets are typically based on images showing a small number of pedestrians (identified with a bounding box). Positive examples come from the region inside the bounding box, and negative examples come from bounding boxes sampled from elsewhere in the image. The most common data sets used as benchmarks in pedestrian detection include Caltech-USA [5][6], ETH [24], Daimler [25] and KITTI's pedestrian data set [23].

In these data sets, the number of positive bounding boxes corresponding to pedestrians are overwhelmingly smaller than the number of candidates corresponding to background, resulting in a severe class imbalance problem. Training deep learning models usually relies on a relatively balanced set of examples in order to stop the training from collapsing to a trivial solution, such as predicting everything as background. The small number of pedestrians also hurts generalization because the few examples the model has for the positive class (i.e., pedestrians) only allows it to represent a limited amount of variations. Conversely, the presence of a large amount of background data increases the variation in the distribution of negative samples.

The *training* difficulties of class imbalance are addressed by existing methods such as R-CNN [1], which uses sampling to present a more balanced set of samples to the detector while training. To improve *generalization* with a small amount of pedestrian samples is harder. More training samples can be gathered, the model parameters can be regularized or the problem can be re-parameterized with a more adequate model architecture. Gathering and annotating more data is effective, but costly and time consuming. This is particularly true for pedestrian detection, where data collected usually has so few positive examples. Regularizing model parameters is a widely applied solution but existing regularizers, such as weight decay, are too general to represent well the characteristics of the data set. Re-parameterization aims at improving training without increasing the number of parameters, by changing the training method and the model structure. Examples include the changes to the R-CNN model parameterization by Fast R-CNN [2] and Faster R-CNN [3], which are designed specifically for detection applications. Specializing further, for pedestrian detection instead of general detection, we have hybrid algorithms that incorporate boosted decision forests [17][10][9], and specialized part-based models that integrate prior knowledge of human structure [30]. However, increasing specialization for the pedestrian detection problem relies on increasing amounts of human effort to find good priors and heuristics, to some extent replacing the machine learning it is meant to improve. We argue that it is more interesting and useful for the computer vision community to develop a regularization approach that can be applied more generally in other detection problems. Our proposed regularization is based on a reconstruction objective function that has characteristics of all of the above methods. More specifically, we extend the training of supervised pedestrian detection models with autoencoders for image reconstruction. Weight sharing between the supervised detector and autoencoders improves the accuracy of the

pedestrian detection model on the test set. The training of supervised deep learning models with autoencoders has been successful on classification tasks such as CIFAR [91] and ILSVRC [92], but to the best of our knowledge this approach has not been used to assist detection problems.

Our proposed methodology combines modern supervised deep learning detection models with an autoencoder model to form a novel deep learning approach for pedestrian detection (see Fig. 3.1.1). We call this combined model the Region of Interest Autoencoder (ROIAE). Unlike previous supervised and autoencoder combinations, we restrict our autoencoder task to only reconstruct the image's foreground regions. Because of the scarcity and smaller variation among pedestrians, this method makes the reconstruction task easier to train by efficiently using the model capacity. We demonstrate our model using the Caltech-USA data set [5][6] for training and testing. Using the supervised detector of [4] as a baseline, our proposed method achieves a 14% log-average miss rate using a VGG-16-based Region Proposal Network (RPN) [3] (the published detection result from [4] was 14.9%). Training this baseline model with our proposed ROIAE method instead yields 12% log-average miss rate (a 2% absolute and 14% relative improvement over the purely supervised version). We also test another baseline using supervised Fast R-CNN and AlexNet that achieves a 23% log-average miss rate, while the ROIAE-trained version yields 22%. An important observation is that the autoencoder component is only used to regularize the training process. After training, the autoencoders can be discarded, leaving a detector with better accuracy on the test set and without any extra overhead in evaluation, allowing us to continue running the detector in real time (5 frames per second) on commodity hardware.

3.2 Literature Review

Our contribution is to take an existing pedestrian detector and to regularize its training process with autoencoders to improve generalization. In this section we summarize the literature of each area that the ROIAE draws upon: detection with deep learning and regularized training with autoencoders.

3.2.1 Detection with Deep Learning

Detection can be viewed as classification over regions in an image. A sliding window detector moves a 'window' over the input and classifies image patches into foreground or background using this window. To ensure every possible object is covered, the collection of windows needs to cover the whole image and overlap with a limited stride and with multiple scales. While this is a natural approach to solve this problem, in practice converting one input image into several thousand and classifying each one separately can be too slow, if not algorithmically optimized.

The first successful deep learning model for general detection was the Region-Based Convolutional Neural Network (R-CNN) [1], which re-parameterizes the convolutional neural network (CNN) classifiers such as AlexNet [14] and VGG-16 [16] for the problem of pedestrian detection. In R-CNN, an external method is used to propose potential regions of interest (ROIs) based on 'objectness' [1]. These ROIs are cropped out and warped to fit into the input of a neural network classifier. This is an improvement over pure sliding window because of the reduced number of background proposals, but the external proposal method itself is computationally complex. This method was extended by Fast R-CNN [2] which reuses convolutional feature maps for all ROIs in an image. The Faster R-CNN method [3] replaces the external region proposal mechanism with a deep learning based approach, called *Region Proposal Networks* (RPN). This means that the features for ROI proposal and classification can now be trained and tested in an end-to-end manner. This end-to-end training can potentially find better candidate ROIs, because the final detection loss is used in the optimization of the RPN, so the RPN is trained to produce optimal ROI candidates. R-CNN was successfully adapted for the Caltech-USA pedestrian detection data set by Hosang et al. in 2015 [9], achieving competitive results. In particular, Hosang et al. [9] used a boosted decision forest for region proposal, making this a hybrid model. This idea was extended by the "Scale Aware Fast R-CNN" model [10] which achieved state of the art results by creating distinct sub-networks to detect small-sized pedestrians, again using a boosted decision forest to

generate region proposals. The CompACT-Deep [17] model took an alternative approach and used pretrained neural network features in a boosted decision forest detector.

Zhang et al. [4] demonstrated that it is possible to train an accurate pedestrian or other single class detector, *using region proposal networks alone*. In adapting the model to pedestrian detection they needed to address problems caused by small-sized pedestrians. While they are able to integrate R-CNN to improve the baseline RPN detector, this is only possible with *a trous* convolution to generate higher resolution outputs [4], while adding Fast R-CNN to the RPN actually degrades performance even with *a trous*. They then extended their work by creating an RPN/decision forest hybrid model, but the novelty was the ability of the RPN to get accurate pedestrian detection results on its own. Because of its speed, competitiveness and the fact that it is a pure deep learning method, we use the RPN as our main baseline model.

The RPN approach was extended by the current state of the art for pedestrian detection, the Fused Deep Neural Network (F-DNN) [11]. F-DNN uses model fusion to create a fast and accurate detector based on the Single Shot Multibox Detector (SSD) [12]. While they also provide a fast neural network only detector, F-DNN's contribution is a different supervised architecture while ours regularizes an existing supervised network with autoencoders at training time. These, along with other approaches such as decision forest hybrids, hard negative mining, ensembles and use of visual flow are not mutually exclusive and can be combined in principle.

3.2.2 Regularized Training with Autoencoders

In order to improve detection accuracy we look at previous works that explored the extension of supervised training with autoencoder learning tasks. This approach was explored for the training of the Deep Belief Network or DBN [83]. The DBN model is based on a series of smaller models called Restricted Boltzmann Machines (RBM), a type of energy-based bipartite graphical model. Training an RBM involves the minimization of a layer-wise reconstruction loss. After training one RBM, the parameters can be fixed and a second RBM is placed on top of the first and trained using the output of the hidden layer from the first RBM in a process called 'generative pre-training'. At first, generative pre-training was just used to overcome the difficulties of training deep neural networks via gradient descent, but Erhan et al. [40] demonstrated that generative pre-training can also improve generalization.

This property was exploited by the Stacked Denoising Autoencoder (SDAE) [89], which added noise to its training data and learned to reconstruct the denoised input. The SDAE learned to preserve information along vectors of variation within the training data set, but to throw away irrelevant information. This results in a contraction of any inputs towards a manifold in feature space that contains the training data. By learning these invariants, the model generalized better than prior stacked autoencoders or the RBM-based networks.

Autoencoders were initially proposed to augment fully-connected networks and performed well at this task, but to augment most 'real world' models they have to be converted to convolutional networks. The main problem here is that the pooling operation in CNNs is not invertible, leading to difficulty in reconstruction [20][91][27]. Another difficulty in applying autoencoders to help real world supervised problems is the spatial size of the input to be reconstructed; the larger the spatial size of the input, the more difficult the reconstruction task. We base our ROIAE's autoencoder component on the Stacked What-Where Autoencoder (SWWAE) [91]. This model uses unpooling to partially invert the max-pooling process by saving the location of the pooled pixels in the original image. SWWAEs do not use generative pre-training; instead all the partial autoencoders and an additional end-to-end autoencoder are trained jointly with the supervised task. The cost function for training is a weighted sum of the cost of the supervised task and all the autoencoders. The SWWAEs showed improvement on the CIFAR [31] and STL [32] data sets when unlabelled data was used in addition to the standard labelled data. While they do perform reconstruction, SWWAEs do not use denoising or other contractive objectives to ensure contraction to a manifold.

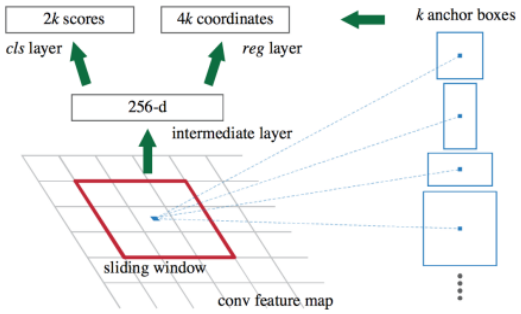


Figure 3.3.1: Region Proposal Network, from [3]. The RPN implements a sliding window over a CNN's convolutional feature map and, for each location in the output feature map, predicts bounding boxes relative to k fixed proposals called anchors.

By contrast, Ladder Networks [27] use noisy skip connections in place of unpooling to recover information lost in downsampling.

Recently, SWWAE was applied to the ILSVRC classification problem, demonstrating the scalability of the model [92]. Joint training performed better than generative pre-training, and both end-to-end and intermediate autoencoders were required for best results. This model did not use additional unlabelled data; the autoencoders were able to exploit information in the labelled data that the supervised training could not. Interestingly, this model improved both training and testing accuracy, suggesting that autoencoders can improve convergence and generalization at the same time.

Our ROIAE method jointly trains an RPN for pedestrian detection with autoencoders inspired by the SWWAE described above. Unlike the methods mentioned above, we force our model to focus on the foreground (i.e., pedestrian) examples, using its limited representation capacity as efficiently as possible for our purposes. Using our proposed method we can improve detection accuracy by making up for the shortage of foreground data in detection problems.

3.3 Methodology

In this section, we first describe the existing supervised RPN detector in isolation. We then describe our ROIAE method, which extends the RPN during training by joint training with autoencoders that reconstruct foreground examples.

3.3.1 Detection with Region Proposal Networks

An RPN is a type of fully-convolutional network (FCN) [28], which is fine tuned from a widely available classification model such as VGG-16 [16] that has been pre-trained on the Imagenet [29] classification task. Assume that our data set is represented by $\mathcal{D} = \{(I, \mathcal{B})_i\}_{i=1}^{|\mathcal{D}|}$, where $I : \Omega \mapsto \mathbb{R}^3$ defines an image, $\Omega \in \mathbb{R}^{H \times W}$, and $\mathcal{B} = \{b_i\}_{i=1}^{|\mathcal{B}|}, b = [x_1, y_1, x_2, y_2] \in \mathbb{R}^4$ defines a set of manually annotated bounding boxes. The RPN implements a deep learning model represented by a sequence of L pairs of linear and nonlinear transforms: $f(I, \theta) = f^L \circ f^{L-1} \circ f^{L-2} ... \circ f^1(I, \theta)$, where θ denotes the model parameters.

During testing, the model takes a test image $\tilde{I}$ as input and returns $k \times n$ bounding boxes: $\{(\tilde{b}, \tilde{c})_i\}_{i=1}^{k \times n} = f(\tilde{I}, \theta)$, where $\tilde{c} \in [0, 1]$ denotes a confidence value for each bounding box $\tilde{b}$. The output of the final layer L consists of $k \times n$ bounding boxes from the output of f^{L-1}, where n is the size of the input feature maps to layer L and k is the number of channels in L. These k channels correspond to 'anchor boxes', which are prototype bounding boxes with a preset aspect ratio and scale (see Figure 3.3.1). The predicted bounding boxes $\tilde{b}$ are defined relative to their corresponding

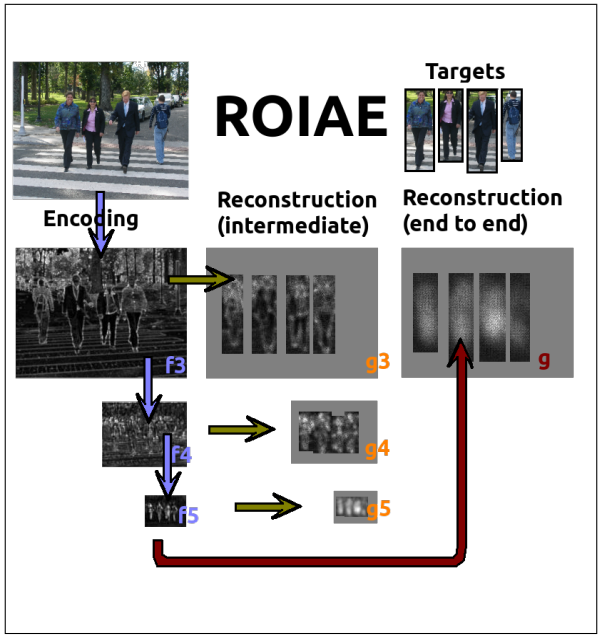


Figure 3.3.2: Average responses of convolutional layers in an ROIAE to an annotated image. Three encoding layers are part of both a supervised RPN and the autoencoders. In this specific model we do not use g^1 or g^2. Intermediate reconstruction is generated by the autoencoders $g^3 \circ f^3, g^4 \circ f^4, g^5 \circ f^5$. The end-to-end autoencoder $g \circ f$ reconstructs f^2 rather than I.

anchor. The t-highest predicted bounding boxes are chosen as candidates, where $t < k$ is a hyper-parameter. Greedy Non-maximum suppression (NMS) is applied to prevent multiple detections for the same object, then a second, tighter confidence threshold is applied to get the final $s < t$ predictions, where s is another hyperparameter. For further details on the parameterization of the RPN see [3], [2] and [1].

To train the model and find θ, we use a training set extracted from data set $\mathcal{D}$ defined earlier, containing images and ground-truth bounding boxes. There are two loss functions, a classification loss function L_{cls} and a regression loss function L_{reg}. The regression loss compares the s bounding boxes in $\tilde{\mathcal{B}}$ to the ground truth bounding boxes in $\mathcal{B}$. Each $\tilde{b} \in \tilde{\mathcal{B}}$ is assigned its nearest counterpart in $b \in \mathcal{B}$ as a regression target (see [1]).

$$L_{reg}(B, \tilde{B}) = \sum_{i=1}^{|B|} \begin{cases} ||\tilde{b}_i - b_i||_2, & \text{if } ||\tilde{b}_i - b_i||_1 \leq 1 \\ ||\tilde{b}_i - b_i||_1, & \text{otherwise} \end{cases}. \tag{3.3.1}$$

The classification loss L_{cls} addresses each $\tilde{b}$'s confidence level $\tilde{c}$. The confidence target c depends on the accuracy of $\tilde{b}$.

$$c = \begin{cases} 1, & \text{if } \frac{\tilde{b} \cap b}{\tilde{b} \cup b} > 0.5 \\ 0, & \text{otherwise} \end{cases}. \tag{3.3.2}$$

L_{cls} uses the softmax loss:

$$L_{cls}(c, \tilde{c}) = c - \frac{e^{\tilde{c}}}{\sum_{i=1}^{n} e^{\tilde{c}_i}}. \tag{3.3.3}$$

The final supervised RPN loss is a weighted sum $L_{RPN} = L_{cls} + \lambda L_{reg}$. The parameters are updated using this weighted loss with Stochastic Gradient Descent (SGD). The RPN is used both as our baseline and as the supervised component in our ROIAE.

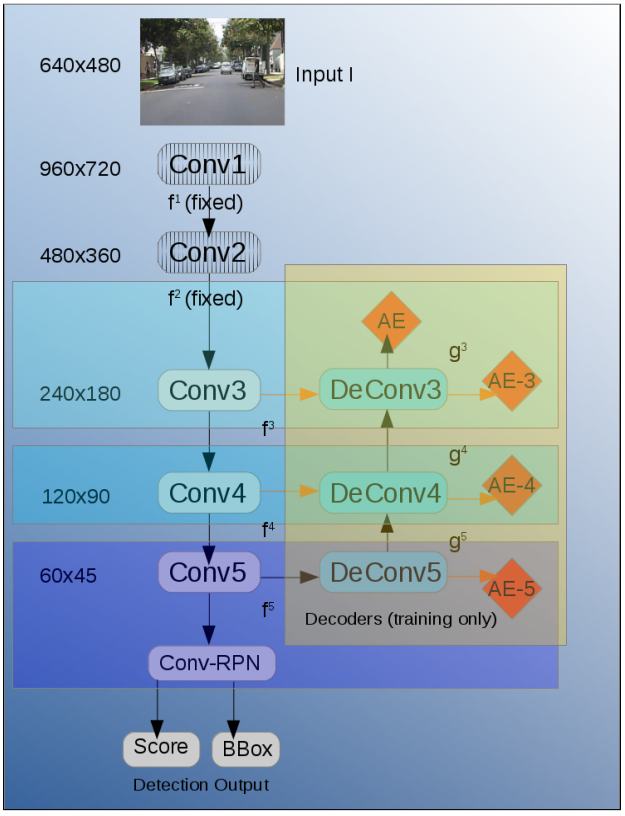


Figure 3.3.3: VGG-16 RPN extended with Region of Interest Autoencoder. Orange arrows indicate intermediate reconstructions.

3.3.2 Region of Interest Autoencoder (ROIAE)

Our ROIAE model extends the RPN by adding an autoencoder component that minimizes an image reconstruction loss. Unlike previous autoencoders we only want to reconstruct areas of an input image I within the ground truth bounding boxes $b \in B$ from our detection problem. To achieve this we construct a binary mask $M^{(I)} : \Omega \mapsto \{0, 1\}$, where 0 denotes background and 1 denotes foreground (i.e., regions containing pedestrians). The masked image is defined by $I^{(M)} = I \odot M$. In the autoencoder we use the RPN's set of transforms $f(I, \theta)$ as encoders, and we introduce up to L new transforms $g(I, \theta) = g^1 \circ g^2 ... \circ g^L \circ f(I, \theta)$ as decoders, $l \in \{\mathbb{N} \mid 1 \leq l \leq L\}$. By abuse of notation we use f^l and g^l to refer to both the function itself and the output of the function. Because reconstruction on the earlier feature maps is harder due to their large spatial size, reconstruction for *all* layers may be too difficult, or the model may do better with the early layers fixed. To address this we can remove the first k decoders and use layers $l \in \{\mathbb{N} \mid k \leq l \leq L\}$.

Once the model is trained we expect $I^{(M)} \approx I^{*(M)} = M \odot (g(I, \theta))$, where I^* is a reconstruction of I (see Figure 3.3.2 for examples of reconstructions). $g(I, \theta)$ is called the *end-to-end autoencoder*. We also define autoencoders that reconstruct encoded images f^{l-1} from a deeper encoding f^l, via decoder g^l. The encoder f^l and a decoder g^l can be turned into the *intermediate autoencoder* $g^l \circ f^l$ (if $l = 1$ the autoencoder $g^1 \circ f^1$ reconstructs I, the input to f^1). The supervised detector and autoencoder reconstruction models share f and both contribute to training its parameters (Fig. 3.3.3 depicts the proposed training structure using the VGG-16 based RPN). The loss function is

$$
\begin{aligned}
L_{ROIAE} = L_{cls} &+ \lambda_{Bbox} L_{Bbox} \\
&+ \sum_{l=1}^{L} \lambda_l ||M^{(I)} \odot (f^{l-1} - g^l)||_H \\
&+ \lambda_{end-to-end} ||M^{(I)} \odot (I - (g^1 \circ g^2 ... \circ g^L \circ f^L))||_H,
\end{aligned}
\tag{3.3.4}
$$

where the H is the Huber norm

$$
||x||_H = \begin{cases} ||x||_2, & \text{if } ||x||_1 \leq 1 \\ ||x||_1, & \text{otherwise} \end{cases}. \tag{3.3.5}
$$

The ROIAE is trained with the same scheme as the RPN, with SGD. Our autoencoder loss function uses the l_1 norm to keep the magnitude of the loss (and thus the gradient descent steps) relatively small, to ensure numerical stability. The masking operation is critical in enabling the autoencoders to discover variations that characterize the sparse foreground. During training, the decoding layers in g learn to reconstruct foreground examples and forces the encoding layers in f to update themselves to preserve information necessary for reconstruction. Preserving the variations within the foreground aims to regularize the training process. At test time the decoder layers are discarded, leaving a model of the same size as the RPN baseline but with better-regularized features.

3.4 Experiments

In this section we describe the Caltech-USA pedestrian detection data set used to evaluate our method, describe our experiments in detail and present our results.

3.4.1 Caltech-USA data set

Caltech-USA [5][6] is one of the main benchmarks for pedestrian detection. Caltech-USA not only provides a data set, but a detailed evaluation protocol for comparing performance. Approximately 10 hours of video have been annotated from a car driving on-road. In total there are 250,000 annotated frames, where every 3rd frame is sampled for training. Pedestrians are divided into three scales based on the height of their bounding box: near (80 or more pixels), medium (30 to 80 pixels) and far (30 pixels or smaller). The creators of the Caltech-USA data set propose a "Reasonable" subset of the data. This set only includes pedestrians that are labeled 'person' (thus no one in crowds), at least 65% of the pedestrian visible and heights of 50 pixels or larger. This is the data set most used as a benchmark, and we use it for all our evaluations. We treat the video frames as still images and do not make use of any temporal information.

The detector is evaluated based on the bounding boxes it returns (after non-max suppression). Bounding boxes generated by the detector must be assigned one-to-one to the ground truth bounding boxes. Two bounding boxes can only be matched if their intersection-over-union (IOU) ratio is 50% or higher, the standard precision for bounding box matching in most object detection tasks. Ground truth bounding boxes with no match are counted as false negatives. Predicted bounding boxes with no match are counted as false positives. Ground truth bounding boxes marked 'People' (meaning a dense crowd) are set to 'ignore'. Ignored ground truth boxes can match multiple proposals at any IOU if they have not already been matched to a positive; neither the ignored ground truth or proposals matched to them are counted in the evaluation. Evaluation is presented as an f-ROC curve plotting false positives per image (FPPI) against the miss rate. Accuracy can be summarized with a scalar by taking the log-average miss rate between 10^{-2} and 10^{0}. In practice this gives similar results to the miss rate at 10^{-1} FPPI.

3.4.2 Experimental Setup

We demonstrate the ROIAE by extending two pedestrian detection models, a small scale model based on AlexNet and a larger one based on VGG-16. These models were created for the ILSVRC competition and are thus well known and pretrained versions are widely available.

AlexNet

Our initial work was inspired by the work of Hosang et al. [9], which used a boosted forest to provide region proposals for an R-CNN network based on AlexNet. As in [9] we use an ACF-based [21] decision forest to prepare region proposals, and a Fast R-CNN detector based on AlexNet. We use the implementation of Fast R-CNN from [2] and modify it to support the Caltech-USA data set. Our AlexNet model is first pretrained on Imagenet [29], then on Pascal-VOC [8]. We scale up the input images to 1500x1000 with bilinear upsampling to prevent the CNN from down-sampling excessively. Our AlexNet variant contains Batch Normalization (BatchNorm) modules that are not present in the original model to help with the autoencoder training. We load the features from the AlexNet variant pre-trained on PASCAL-VOC from [2] and let it adapt to the normalization during training. AlexNet has 5 convolutional layers and 2 fully connected layers. To create the autoencoder component of our ROIAE we construct intermediate autoencoders for convolution layers 3,4 and 5. Each intermediate autoencoder reconstructs the input of one of the convolutional layers from its output. There is also an end-to-end autoencoder that reconstructs the output of convolution layer 2 from convolution layer 5. The end-to-end and intermediate autoencoders share parameters. We keep λ_{Bbox} set at 1. The autoencoder loss weights are far smaller, to shrink the large magnitude of the regularization loss to the same order of magnitude as the other losses. The end-to-end autoencoder loss has a weighting of 5×10^{-5} and the intermediate losses have a weighting of 1×10^{-5}. As in [92] the weights need to be adjusted so that they regularize the training without over-regularizing and hurting convergence on the supervised detector. We train for 40,000 iterations (here, one iteration corresponds to the training of a mini-batch) using Nesterov momentum at 0.99%. Training beyond 40,000 iterations does not improve performance. We find Nesterov momentum with a high value necessary for the relatively small AlexNet to find good search directions. We use a batch size of 4 for our training, but batch sizes from 2 to 16 did not produce any noticeable change in the results.

VGG-16

We replicate the setup of Zhang et al.[4] for our Region Proposal Network baseline and the supervised component in our ROIAE. Like our Fast R-CNN model, the images are scaled up, this time to 960x720, the scale used by [4]. This model is a modified version of VGG-16, containing 5 different convolutional scales, sometimes called 'macro layers'. The first 2 macro layers contain 2 sequential convolutional layers, and the third, fourth and fifth macro layers contain 3 convolutional layers each. Each macro layer is followed by max pooling. We initialize the weights from the VGG-16 model and fix the first two macro layers in place. We insert Batch Normalization after every macro-layer. While large scale autoencoder regularization is possible without batch normalization (see [92]), it makes the model more tolerant of a range of hyper-parameters.

The autoencoder component of the ROIAE uses an end-to-end reconstruction that reconstructs the input of macro-layer 3 from the output of macro-layer 5, and has intermediate reconstruction objectives for each learnable macro-layer (i.e., macro-layers 3,4 and 5, see Figures 3.3.2 and 3.3.3). We use Parametric ReLU [34] in the decoder activation functions to help train the decoder layers and apply Dropout [64][65] with a value of 0.5 to all data entering each decoder. We train using SGD with Nesterov momentum 0.9 using a batch size of 1 (required by the RPN implementation) for 80,000 iterations, as in [4]. We start with a learning rate of 10^{-3} and reduce it to 10^{-4} after 60,000 iterations (again following from [4]). We keep λ_{Bbox} set at 5, meaning the bounding box regression is five times stronger than the classification loss, the same as in the baseline model. We use a loss weight of 5×10^{-7} for the end-to-end autoencoder, 1×10^{-7} for the macro-layer

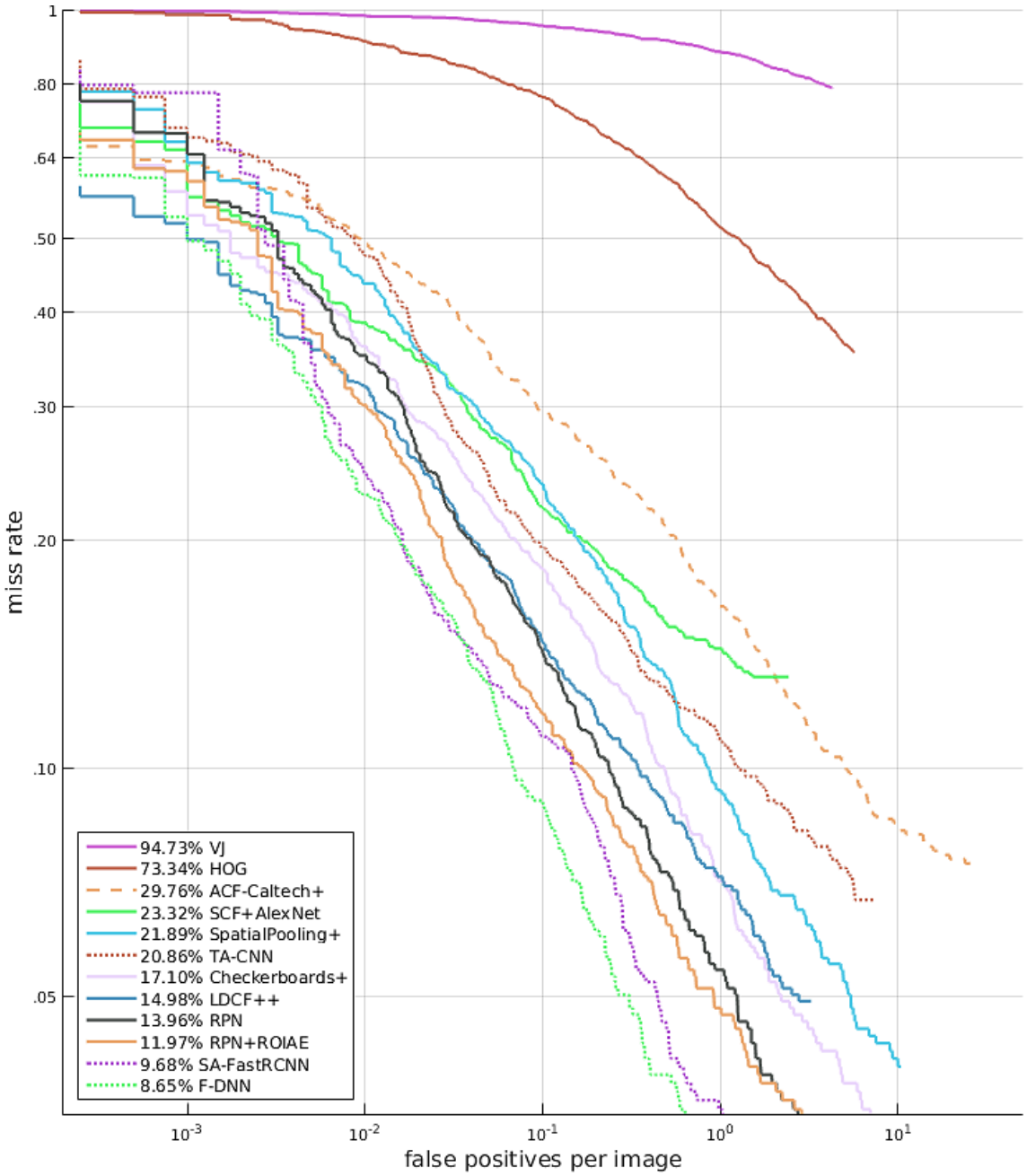


Figure 3.4.1: f-ROC curves of our baseline (RPN) and augmented model (RPN+ROIAE) with other published models on Caltech-USA. We include the two best models, Scale Aware Fast R-CNN and Fused DNN, for completeness but these are not directly comparable to ours. The f-ROC (Free-response Reciever Operating Characteristic) curve measures the trade off between false positives *per image* and the total number of false negatives in the test set. The curve is summarized by the log-average miss rate, calculated by averaging evenly sampled points along the curve in log space, between 10^{-2} and 10^{0}.

3 autoencoder, 1×10^{-9} for the macro-layer 4 autoencoder and 1×10^{-7} for the macro-layer 5 autoencoder. We found the loss weights by manual search.

We use the standard settings for Caltech, but we expand the ground truth available for training the autoencoder component by including boxes labelled 'ignore', which are usually excluded from training entirely, because these pedestrians are too close together to distinguish, too small, near

Table 3.1: Log-average Miss Rate on Caltech-USA

Model	Test	Train
(AlexNet)		
SCF + R-CNN (published by [9])	23.33%	N/A
ACF + Fast R-CNN	23.08%	13.38%
ACF + Fast R-CNN + ROIAE	**22.14** %	**10.28**%
(VGG-16)		
RPN (published by [4])	14.9%	N/A
RPN + R-CNN (published by [4])	13.1%	N/A
RPN (our implementation)	13.96%	12.38%
RPN (without BatchNorm)	15.42%	**7.03**%
RPN + SWWAE	13.87%	10.88%
RPN + ROIAE	**11.97**%	7.80%

the image border or too heavily occluded.

We build our RPN+ROIAE model in the Caffe framework [13] using the Matcaffe wrapped to integrate with Matlab. We use the Matlab implementation of RPN and Faster R-CNN provided by [4].

3.4.3 Results and Analysis

We compare the training and testing results of our baseline and autoencoder-augmented neural network models using Dollar's Caltech toolkit in Matlab to evaluate our model [5].

The results in Table 3.1 indicate that our proposed ROIAE improves the testing and training accuracy over the purely supervised baseline. The ROIAE achieves a 1 percentage point improvement in log-average miss rate for AlexNet and a 2 percentage point improvement for VGG-16, with relative improvements of 4% and 14%, respectively. This may seem small, but improvements of this scale are commonplace in the literature. For a comparison with state of the art see the f-ROC curves in Fig. 3.4.1. Although both training and testing improve in our models, we can confidently attribute our performance to an improvement in generalization rather than an improvement in training. It is relatively easy to overfit on this dataset because of the large number of pedestrians, and further training on both our baseline and autoencoder-augmented models results in overfitting, increasing the test error while decreasing the training error. In other words, our autoencoder-augmented model can reach a higher training accuracy before it starts to overfit and reduce the test accuracy. In Fig. 3.4.1 we also show the state of the art methods Scale Aware Fast R-CNN and Fused DNN, which achieve higher accuracy but use more computational resources. Scale Aware Fast R-CNN uses two sub-networks, one trained on large pedestrians and one on small pedestrians. They use a decision forest to generate candidate bounding boxes. F-DNN uses SSD, a more advanced baseline than RPN, which connects every feature map to the last layer, incorporates boosting, and finally trains on data from a *combined* dataset rather than just Caltech-USA's training set. Our method on the other hand concentrates on improving an existing architecture without additional use of resources for evaluation.

The motivation behind the ROIAE was to apply reconstruction objectives to *small but important* spatial regions of the model's feature maps. In our case this means that we only attempt to reconstruct regions with pedestrians rather than background. To compare the effect of our ROIAE to other autoencoders, we tested a non-masked autoencoder, identical to the ROIAE except that it did not mask out the background. This non-masked autoencoder training produced nearly the same result as the baseline: 13.87% vs the baseline's 13.96% (see SWWAE in Table 3.1). This validates the motivation behind the ROIAE: due to the small variation in the pedestrians compared to

the large variation in background, the autoencoder can be trained more effectively to reconstruct the set of pedestrians only. This addresses two issues; firstly that we need more robust models of pedestrians, not background, which is abundant. Secondly, large scale reconstruction is difficult to optimize and would tend to absorb a lot of the model's capacity, crowding out information useful for the supervised problem.

We performed our evaluations on an Nvidia 980 Ti GPU. The Fast R-CNN models using AlexNet take 110 milliseconds to evaluate each image (9 frames per second), while the Region Proposal Networks using VGG-16 take 175 milliseconds (5.7 frames per second) for each image. Because we can eliminate the decoders after training, our ROIAE models have the same speed as their respective baselines.

3.5 Conclusion

Pedestrian Detection is one of the fastest growing applications in computer vision and machine learning. With this in mind we have not chosen to aim for record breaking results, but to demonstrate that our ROIAE yields clear improvements over an existing baseline. The ROIAE could potentially be combined with other advances in detection to yield a model of higher accuracy without any additional test-time overhead.

Regularizing training with autoencoders was an important step forward in training fully connected deep neural networks. Advances made in the training of large convolutional networks have produced networks of immense depth. However there is a limit to how much supervised training alone can achieve with limited training data while avoiding overfitting. In classification, autoencoder-assisted learning can extract more useful features out of the same data than purely supervised learning by explicitly modeling variations in the data set.

The recent success of autoencoder-augmented convolutional networks on CIFAR [91] and Imagenet [92], and the results we present in this paper with our ROIAE, imply that even more can be accomplished on detection, where foreground training examples are so sparse. Because in our method the decoder elements are thrown away after training, there is no added computational cost to during the testing procedure. The model can be used alone or combined with other advances such as sensor fusion [7], multi-scale networks [10], boosting [17] or hard negative mining [33].

In the future, we plan to expand our model to more data sets, explore the potential of the ROIAE under neural network compression schemes and to explore the nature of the features learned and whether they are similar to those in semi-supervised classification. Our experiments so far have not used a denoising or other contractive criterion; using a ladder network might result in further improvements to generalization.

3.6 Acknowledgments

The authors are members of the Australian Centre for Visual Technologies (ACVT). This work was supported by Industry Linkage Project LP130100521, "Intelligent Collision Avoidance System for Mobile Industrial Platforms". G.C. acknowledges the support by the Australian Research Council Centre of Excellence for Robotic Vision (project number CE140100016).

Chapter 4

Iterative Node Pruning with the Parametric Information Bottleneck

It is difficult to compress neural network models while preserving accuracy. We use the Parametric Information Bottleneck framework to find a compression rate that reliably preserves the accuracy of compressed models under certain conditions. We take a trained neural network model and find an upper bound on each layer's ideal width, that is, the smallest width that will result in no reduction in accuracy from the original model. We demonstrate that iteratively compressing and fine tuning converges to a model with the same or greater test accuracy than the original. We examine the impact of regularization and starting model size on the final compression rate. However, this methodology is shown to only be applicable to binary images (i.e., black and white), such as the ones in MNIST, and for sigmoid activation.

4.1 Model Compression

Deep neural network models have achieved state of the art results on many machine learning tasks in the last few years despite the lack of theoretical guarantees on their performance. Remarkably, while large model sizes are needed to optimize a network for good performance, most of the features learned are highly redundant [60] and the model can be compressed while retaining most of its accuracy [57] [60] [53] [50]. The most likely reason for this is that as the number of dimensions in the weight space grows, the ratio of saddle points to local minima in the optimization program grows exponentially [42], which means that being trapped in a poor minimum is much less likely. Increasing model size beyond what is necessary to model the data enables an easier optimization problem, but this is no longer needed for the evaluation of a trained model because the model weights are frozen and some of the large number of weights that made the optimization easier become redundant. Model compression is useful for reducing resource use in terms of memory, electricity and computations, especially for mobile devices, but it can also be used to improve inference speed in real time applications such as pedestrian detection.

Model compression methods for neural networks fall into a few distinct categories:

- Weight Pruning
- Channel/Node Pruning
- Lossless Codes
- Tensor Decomposition
- Quantization

- Or some combination of the above.

They usually require setting hyperparameters that directly or indirectly control the compression rate. Achieving a good balance between model compression and classification accuracy requires a time-consuming hyperparameter search. A few exceptions are Net Trim [61] and the related Layerwise Optimal Brain Surgeon [58], but they require time consuming CPU calculations to solve quadratic optimization programs or the inversion of Hessian matrices. One of the first neural network compression methods, Optimal Brain Damage [56], consisted of a weight pruning approach that formed a regularization method and a compression algorithm. Compression methods can improve generalization because they reduce model capacity, forcing it to find a simpler model to fit the data. The fact that compression methods can act as a regulariser is important because it links over-fitting with model compressibility. Optimal Brain Surgeon [57], is able to find the least 'salient' weights, which would least affect the model output if removed, but to do this requires computing the model's Hessian, which is computationally intractable for large networks. The recently proposed Layer-Wise Optimal Brain Surgeon [58] uses a clever approximation method for the inverse Hessian using block matrices but they still require 48 CPUs and almost as much time as the training itself to compress a large network.

Because of the difficulties in computing the optimal compression parameters, most modern compression methods are focused on practical performance in the form of compression ratios vs. accuracy of the compressed model. Deep Compression [53] delivers very high compression ratios by combining simple pruning by zeroing out low-magnitude weights, quantization and Huffman Coding. This method requires decompression for inference, so the authors have also proposed a hardware system capable of efficient inference on these compressed models [55]. Compression with the objective of faster inference can be accomplished by tensor decomposition, such as truncated SVD or the Tucker Decomposition [93], which converts one layer into multiple smaller layers with fewer computations overall. However no consistent system for ranking these methods' performance has been adopted. In this chapter we present a new model compression algorithm using the channel pruning (for convolutional layers) and node pruning (for fully connected layers) compression paradigms. While this is more constrained than other methods such as weight pruning, models compressed by channel pruning have no inference overhead, and can easily exploit the smaller model size for faster inference. Channel pruning algorithms (e.g., [43]) generally solve some constrained optimization problem to minimize the damage caused by the removal of large blocks of the model. Instead, in our model, we propose a greedy algorithm that finds the smallest safe size for each layer. We measure the information contents present in the distribution of the activations of each layer – this tells us the proportion of the layer's capacity that is in use. We can then prune nodes or channels until the information capacity of the layer is approximately equal to the amount of information it contained in the original model. Lost information useful for the objective is recovered with fine tuning. This process can be repeated until convergence, when the algorithm cannot find any way to safely reduce the capacity of the model. Because we offload the recovery of information to the fine tuning process, our method makes efficient use of existing neural network frameworks using the GPU, with no external solvers or CPU-intensive calculations.

4.2 Measuring Model Capacity

To compress a neural network model without losing accuracy, we propose measuring the 'model capacity' of each layer and using this to determine each layer's compression rate. The term 'Model Capacity' is related to the amount of information a model can store about the data it is modelling[72]. The implication is that models with a larger capacity than is necessary for a particular problem can be compressed without losing accuracy [60].

Unfortunately, actually formalizing this notion, and computing the capacity is difficult. The ultimate lower bound is the Kolmogorov Complexity [71] which is informally the length of the smallest computer program that can output the model, but this is non-computable. One can compare the number of parameters in a model, but this is only useful for characterizing the differences

between *model architectures* and does not distinguish between models of different data sets using the same architecture. VC Dimension [77] does capture differences between data sets, but is a poor predictor of neural network performance. In machine learning, it is a common assumption that the data from a particular domain lie on a (relatively) low-dimensional subspace embedded in a high dimensional space. For instance in the set of 32x32 colour images, the subset of images of cars will exclude the large swathes of high entropy images of white noise. It is not necessary nor possible [76] to represent the entire input space, only a subspace that contains our problem domain [75].

Exploiting the structure of the data and the embedding space within a deep neural network requires analysing highly nonlinear relationships. The Information Bottleneck method [35], Minimum Message Length (MML) [74] and Minimum Description Length (MDL) [73] are methods that treat the model as a stochastic process and apply information theory to characterize the model capacity. Minimum Description Length has been used to find compression hyperparameters for quantization and node pruning and can be viewed as an alternative to our approach. MDL characterizes the information stored in the *model parameters* assuming a prior distribution, while the Information Bottleneck looks at the information contained by the *activations* of the neural network layers.

4.3 Rate Distortion and the Information Bottleneck Method

The Information Bottleneck (IB) method was proposed by Tishby et al. [35] as a generalization of the Rate-Distortion theory for the lossy compression of data. In Rate-Distortion theory, we encode elements of the data distribution $x \in X$ to compressed representations z in a code space Z. The mapping $X \to Z$ is described by the conditional probability density function $p(z \mid x)$. A good encoding function will minimize the *rate*, i.e., the *average* number of bits necessary to represent z. A lower bound on the rate is given by the mutual information

$$I(X;Z) = \sum_x \sum_z p(x,z) \log \left(\frac{p(z \mid x)}{p(z)} \right) \leq H(X). \tag{4.3.1}$$

where $H(X)$ is the entropy of X and $p(x,z)$ is the joint probability density function of x and z. $I(X;Z)$ is equivalent to the rate only if the encoded values are represented with no overhead. In addition to minimizing the rate, for a good encoding we need to limit the distortion or loss of information in the encoding. This is characterized by a *distortion function* $d = f(X,Z)$ which has to be defined a priori. This distortion function should monotonically output low values for a high fidelity compression and high values for a low fidelity compression, but specifying which distortions matter and which do not is left to the choice of d. Selecting d is equivalent to selecting features to encode the data set. In Rate-Distortion theory, the optimal encoding for a particular d is given by minimizing the functional

$$\mathcal{L}[p(z \mid x)] = I(Z;X) - \beta \langle d(x,z) \rangle_{p(x,z)} \tag{4.3.2}$$

over all normalized distributions $p(z \mid x)$, where β is a Lagrange multiplier controlling the trade-off between the rate and the expected distortion. The Information Bottleneck method introduces a new variable Y that represents the relevant information in X. This is task-dependent; for instance if X is a set of images, $y \in Y$ could be defined as class labels for X to define a supervised classification problem. In place of the old distortion function $d(x,z)$ in terms of X and Z we substitute $I(Z;Y)$, and define the Information Bottleneck Lagrangian:

$$\mathcal{L}[p(z \mid x)] = I(Z;X) - \beta I(Z;Y). \tag{4.3.3}$$

In classic Rate-Distortion, lossy compression algorithms such as the Discrete Cosine Transform (DCT) used in the JPEG standard rely on hand-crafted features to decide which information to throw away. By contrast, the IB method captures the notion of *relevant information* in terms of a dependent variable. When compressing data with the IB method, the objective is to preserve information that is relevant to predicting some interesting variable Y, while eliminating unrelated

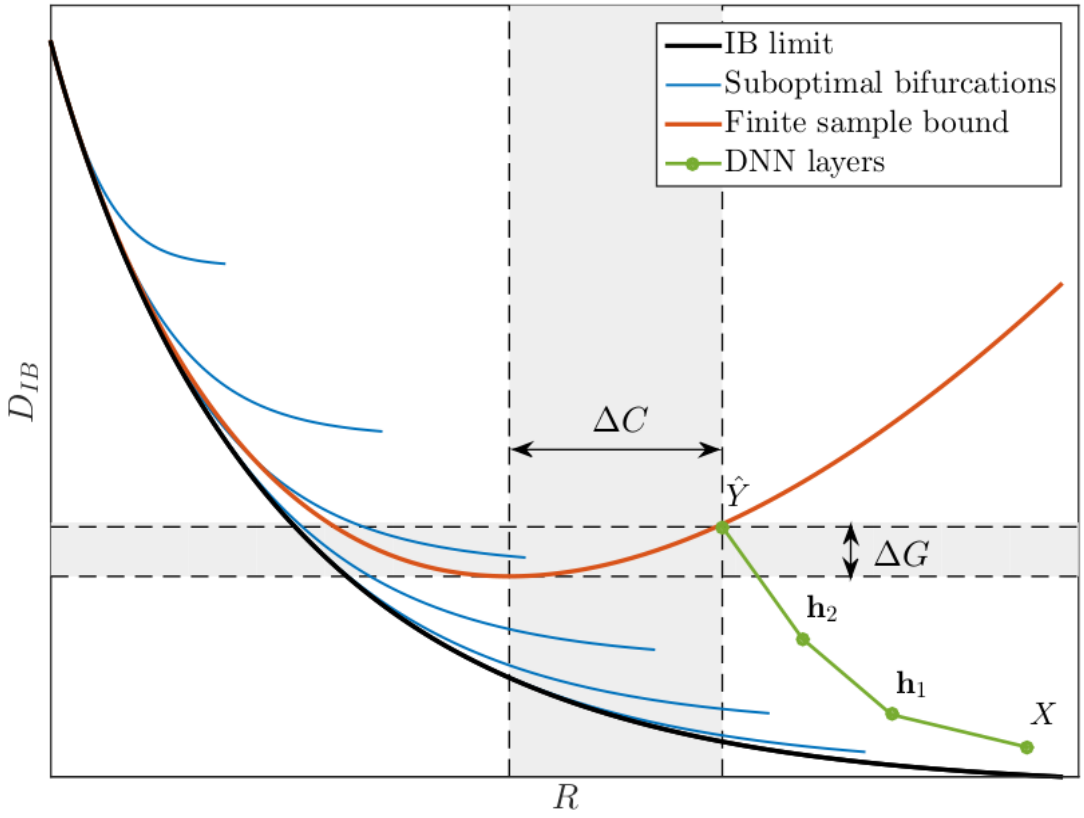


Figure 4.3.1: Figure 2 from [37] showing the relationship between the Information Bottleneck distortion function D_{IB} which measures the relevant information lost during encoding, and the encoding rate R which specifies the average bits used by an encoded datum. The red curve is the distortion upper bound for out of sample inputs. When only a finite sample is available there is a single optimal point with the least worst-case distortion, rather than the trade-off given by the IB curve. ΔC is the compression gap and ΔG is the generalization gap. The DNN layers shown give an example behaviour of a deep neural network used to model X. Each layer contains less information than the preceding one, culminating in $\hat{Y}$, the model's prediction of Y. In the example given, the network is overfitting, because it could achieve lower worst-case distortion D_{IB} with a lower rate R.

information.

The IB method defines the relevant information in a variable X as information it shares with a variable of interest Y, which is the mutual information $I(X;Y) \leq H(X)$. A good encoding $X \to Z$ will preserve information about Y and discard as much other information as possible. As well as finding an effective codebook for compression, such an encoding is equivalent to finding a feature space Z for an input X, for a task involving Y. The information preserved in the embedding from the data is given by Equation 4.3.1 which is a lower bound on and can be viewed as analogous to the rate. The relevant information in the embedding is given by

$$I(Z;Y) = \sum_y \sum_z p(y,z) \log \left(\frac{p(y,z)}{p(y)p(z)} \right) \leq I(X;Y), \tag{4.3.4}$$

which is analogous to the distortion. Ultimately we want to find Z such that $I(Z;Y)$ is maximized and $I(X;Z)$ is minimized. To do this we minimize the IB Lagrangian given in Equation 4.3.3.

We can now treat Z as the feature space in a model trying to predict ground truth class labels

Y from input images X, with a mapping $X \rightarrow Z \rightarrow Y$. $I(X;Z)$ becomes the information in the feature space about the input images X, and $I(Z;Y)$ becomes the information in the feature space about the ground truth labels Y. In Equation 4.3.3, minimizing the $I(X;Z)$ term reduces the complexity of the model by reducing the number of bits needed to represent each $x \in X$, and minimizing the $-I(Z;Y)$ term increases model accuracy. By changing the Lagrange multiplier β we can generate a set of solutions that satisfies this optimization problem; these solutions form a monotonic curve called the Information Bottleneck Limit. If we move along this curve by increasing β, the lower the distortion from the original, but the less it is compressed. However, in the context of machine learning this is an oversimplification. In machine learning we only have a finite number of samples to find a model, and accuracy must be measured on generalization to out-of-sample data. Adding complexity in this case can actually decrease generalization of accuracy due to overfitting (see Figure 4.3.1).

In [36], the authors give worst case bounds for the distortion in a withheld testing set (out-of-sample inputs):

$$I(Z;Y) \leq \hat{I}(Z;Y) + O\left(\frac{2^{I(Z;X)}|Y|}{\sqrt{n}}\right), \tag{4.3.5}$$

$$I(Z;X) \leq \hat{I}(Z;X) + O\left(\frac{2^{I(Z;X)}}{\sqrt{n}}\right), \tag{4.3.6}$$

where $\hat{I}(Z;Y)$ and $\hat{I}(Z;X)$ are the empirical information between the feature space and the class labels and input data respectively, n is the number of samples from X and $|Y|$ is the cardinality (or number of classes) of $|Y|$. In a one-hot encoding this becomes equivalent to the dimensionality of Y. At the limit of $n \rightarrow \infty$ the empirical (training) information is equal to the ground truth (testing) information. For finite n, the gap is proportional to the complexity of the feature space Z and inversely proportional to the square root of the sample size $\sqrt{n}$. To summarize, with a finite set of samples, machine learning models tend to have an ideal capacity in terms of the lower bound on the encoding rate $I(Z;X)$, that gives rise to the minimum distortion. Increasing the capacity of the model beyond this does not improve the accuracy; in fact it can reduce the model's ability to generalize. Because the rate $I(Z;X)$ is upper-bounded by $\log|Z|$, where $|Z|$ is the cardinality of the feature space, This points to a model compression strategy of shrinking $|Z|$ to a conservative upper bound on $I(Z;X)$.

4.4 Using the Information Bottleneck to Estimate Model Capacity for Pruning

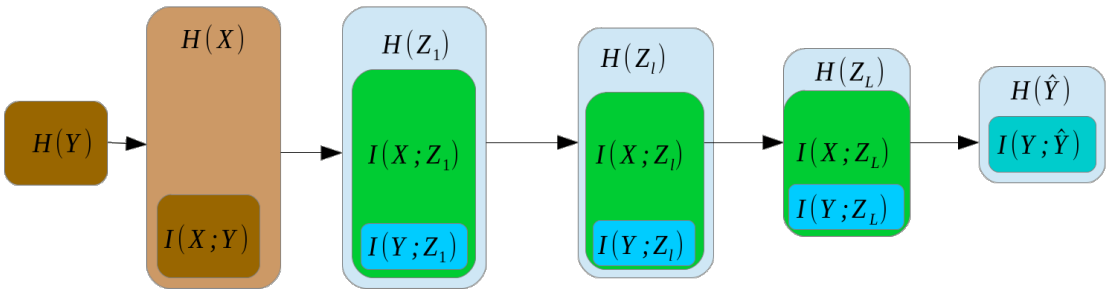


Figure 4.4.1: A neural network model viewed as a Markov chain. Each layer's output is conditioned on the previous layer's output. The information from the input image X is reduced, until only information relevant to classification remains. If $I(X;Z)$ is minimized and $I(Y;Z)$ is maximized a minimum sufficient statistic is obtained.

In neural network models, the amount of memory used in practice is not just dependent on the information needed to store the model parameters, but on the memory needed to store the output of each layer. This total memory usage is dependent on the model architecture. In the case of convolutional networks, the total memory used for inference will grow proportionally to the spatial size of the intermediate feature maps, while the memory needed to store the model itself is dependent on the spatial size of the kernel. Both parameter storage and intermediate memory for a convolutional layer are proportional to the number of channels.

For practical purposes, the most straightforward way to compress models to increase speed and decrease total memory usage, with no specialized hardware (as in EIE[55]) or sparsity exploiting algorithms, is to prune nodes (in the case of fully connected layers) or channels (in the case of convolutional layers). The number of nodes is linked both to the number of parameters and to the size of the intermediate features. This means that we can apply compression not to the parameters, but to the feature maps, and get parameter compression automatically. For instance, pruning a convolutional channel will remove one kernel, and one feature map for that layer. Instead of measuring the information content of the weights, which are not random variables and thus have no obvious way to measure the entropy without a prior distribution, which has to be chosen beforehand (see [49]), we can base our pruning decisions on the information content of the activations. According to the IB method the activations have information content based on their connection to the data-generating distribution.

The key observation is that each layer in the neural network *can preserve or discard information about its input, but it can never add new information.* The output of each layer l has less information about the input than the output of the previous layer $l-1$. This serves to reduce the vast, but mostly irrelevant, information in the input to a few highly informative features that preserve information useful for the final layer to perform classification or regression, as illustrated by Figure 4.4.1. This interpretation has been used with toy models to analyse the learning dynamics of neural network layers [59].

Other approaches such as the Variational Information Bottleneck (VIB)[39], which assumes Gaussian noise, have been applied to compression in the VIBnet [45], but to measure the IB statistics they need to train a stochastic model and incorporate the modelling assumptions as trainable parameters. Using the Parametric Information Bottleneck (PIB) [38], we compute information as if our model was a stochastic network, but actually train it using ordinary SGD.

There are two key problems that arise in the application of the IB method to neural networks: firstly, mutual information terms need to be accurately estimated, and secondly we need to model neural network layers as outputting probability distributions, while they are deterministic functions of their input. We address both of these issues with the PIB approach [38]. The PIB shows that a binary stochastic network can be converted into an equivalent deterministic network using sigmoid activations. The output of the deterministic network corresponds to the mode of the output of the stochastic model.

We denote our labelled data set $D = \{X, Y\}$, where X is the input data and Y the corresponding labels. The data set contains N samples , indexed by k, $D_k = \{X_k, Y_k\}$. To model this we describe a neural network as a series of layers where the l-th layer is represented by stochastic variable $Z_l \in Z$, with $Z_0 = X$. Connections between two adjacent layers l and $l-1$ are described by a stochastic map $z_{l-1} \in Z_{l-1}$ to $z_l \in Z_l$ by the conditional probability distribution $p(z_l \mid z_{l-1})$. This conditional distribution has the same expected value as the layer's deterministic activation function:

$$E[p(z_l \mid z_{l-1})] = \sigma(W_l z_{l-1} + b_l), \tag{4.4.1}$$

where σ is the sigmoid activation function, W_l is the weight matrix for layer l, and b_l the bias.

We can describe this network as a markov chain $X \to Z_1 \to Z_2 \to ... \to Z_L \to \hat{Y}$, where $\hat{Y}$ is the model's prediction of Y. This system obeys the Data Processing Inequality (DPI) which states

that

$$H(X) \geq I(Z_1; X) \geq I(Z_2; X) \geq ... \geq I(Z_N; X) \geq I(\hat{Y}; X), \tag{4.4.2}$$

and

$$H(Y) \geq I(Z_1; Y) \geq I(Z_2; Y) \geq ... \geq I(Z_N; Y) \geq I(\hat{Y}; Y). \tag{4.4.3}$$

Measuring $I(X; Z_l)$ when many layers seperate Z_l from the input is not trivial. Mutual Information is in general very hard to measure for high dimensions when we only have samples and not the exact distribution. Although several advances have been made in this area [48], [47] direct estimates of mutual information in high dimensions have large biases and do not accurately represent differences between highly correlated distributions. The PIB approximates $I(Z_l; X)$, the mutual information between a layer's output distribution Z_l and the data X with mutual information between two adjacent layers, $I(Z_l, Z_{l-1})$. This produces IB statistics that lie within the expected range, which changes smoothly with the parameters and do not have a large bias. Thus we can use it to characterize the information stored in each layer about the data.

Following the layerwise approximation $I(Z_l, X) \approx I(Z_l, Z_{l-1})$ in [38], we approximate the DPI with

$$H(X) \geq I(Z_1; X) \geq I(Z_2; Z_1) \geq ... \geq I(Z_L; Z_{L-1}) \geq I(\hat{Y}; Z_L), \tag{4.4.4}$$

which can be computed using information local to each layer. The PIB interprets the deterministic output of a sigmoid node as parameterizing a Bernoulli distribution. Instead of a sigmoid node outputting $q \in [0, 1]$ we have a binary stochastic node $Z_{l,i} \in Z_l$,

$$q = p(Z_{l,i} = 1 \mid Z_{l-1}), \tag{4.4.5}$$

$$1 - q = p(Z_{l,i} = 0 \mid Z_{l-1}). \tag{4.4.6}$$

Given enough samples the expected value of this stochastic layer converges to

$$E[p(Z_{l,i} \mid Z_{i-1})] = q. \tag{4.4.7}$$

Because of the Markov property, the output distributions of all the nodes $Z_{l,i}$ in Z_l are conditionally independent given Z_{l-1}. This means that the nodes in Z_l form a product distribution

$$p(z_l \mid z_{l-1}) = \Pi_i p(z_{l,i} \mid z_{l-1}). \tag{4.4.8}$$

From this we can derive the measurements for the mutual information $I(Z_l; Z_{l-1})$ by factorizing it into

$$I(Z_l; Z_{l-1}) = H(Z_l) - H(Z_l \mid Z_{l-1}), \tag{4.4.9}$$

which can be estimated using local information only. As per [38],

$$H(Z_{l,i} \mid Z_{l-1} = z_{l-1}) = -q \log(q) - (1 - q) \log(1 - q). \tag{4.4.10}$$

To implement this, we have a number of samples from the previous layer's output Z_{l-1}, in a minibatch $M \subseteq X$. We denote the elements of the minibatch M for the output of layer l as $\{ z_l^{(k)} \mid k \in M\}$, where $z_l^{(k)}$ is the activation at layer l for data X_k, each of which can be assumed to be equally likely, $p(z_l^{(k)}) = \frac{1}{|M|}$. The empirical conditional entropy is given by

$$H(Z_l \mid Z_{l-1}) = E_{p(z_{l-1})}\left[\sum_i H(Z_{l,i} \mid Z_{l-1} = z_{l-1})\right] = \sum_i \sum_{k \in M} \frac{1}{|M|} H(Z_{l,i} \mid Z_{l-1} = z_{l-1}^{(k)}). \tag{4.4.11}$$

To find the marginal entropy $H(Z_l)$ we again use samples of z_l from a mini-batch M as above. We use the maximum likelihood or 'plug in' estimator:

$$H(Z_l)_{MLE} = -E_{p(z_l)}[\log p(z_l)] \approx -\frac{1}{|M|} \sum_{k \in M} \log p(z_l^{(k)}), \tag{4.4.12}$$

but this is not stable when the size of Z_l is large. We need to factorize the output distribution for the layer into the output distributions of each node. We use the PIB's upper bound on the entropy based on Jensen's Inequality:

$$-\log p(z_l) = -\log \frac{1}{|M|} \sum_{k \in M} \left[p(z_l^{(k)} \mid z_{l-1}) \right] \geq -\frac{1}{|M|} \sum_{k \in M} \left[\log p(z_l^{(k)} \mid z_{l-1}) \right]. \tag{4.4.13}$$

We can then work entirely in log-space by converting our product distribution into a sum of log probabilities as follows:

$$\log p(z_l \mid z_{l-1}) = \log \Pi_i p(z_{l,i}^{(k)} \mid z_{l-1}) = \sum_i \log p(z_{l,i}^{(k)} \mid z_{l-1}). \tag{4.4.14}$$

Jensen's inequality could lead to this estimate going above maximum entropy for the Bernoulli distribution, so we threshold each node's estimated marginal entropy at 1 bit per node to yield

$$H(Z_l)_{MLE} = -\sum_i \min \left(1, \sum_{k \in M} \frac{1}{|M|} \log p(z_{l,i}^{(k)} = 1 \mid z_{l-1}) - \sum_{k \in M} \frac{1}{|M|} \log p(z_{l,i}^{(k)} = 0 \mid z_{l-1}) \right) \text{ bits.} \tag{4.4.15}$$

Using this interpretation, every node has a maximum entropy

$$H_{max}(Z_{l,i}) = 1 \text{ bit}, \tag{4.4.16}$$

which occurs when the sigmoid activation is $q = 0.5$. Each layer has

$$H_{max}(Z_l) = |Z_l| \text{ bits}, \tag{4.4.17}$$

where $|Z_l|$ is the number of nodes in the layer (for convolutional layers, substitute pixels in the output feature maps for nodes). We can now directly relate the information carried by the layer to the number of nodes needed to represent it (see Figure 4.5.1). Under the PIB modelling assumption, a layer's output Z_l with n nodes has a capacity of n bits, but only stores $\lceil I(Z_l; Z_{l-1}) \rceil \leq H_{max}(Z_l) = |Z_l|$ bits of information about the data. Ideally, we want to lower the number of nodes by pruning so that $\lceil I(Z_l; Z_{l-1}) \rceil \approx H_{max}(Z_l) = |Z_l|$. Which nodes to prune can be decided by finding the nodes that hold useful information.

4.5 PIBprune Algorithm

In the standard IB objective on a data set X with labels Y, and an embedding Z, the procedure is to maximize $I(Z, Y)$ and minimize $I(Z, X)$. To get effective compression by weight pruning, we want to instead reduce the number of nodes, or for convolutional networks the number of feature maps. We can introduce the number of nodes into the IB framework by noting that each node's output distribution $Z_{l,i}$ has a maximum entropy $H_{max}(Z_{l,i})$ that is a constant depending only on the family of distributions used. The entropy of the entire layer, $H_{max}(Z_l)$, is given by:

$$H_{max}(Z_l) = \sum_i H_{max}(Z_{l,i}) = |Z_l| \text{ bits}. \tag{4.5.1}$$

Instead of minimizing $I(Z_l; X)$, we want to minimize the number of nodes by minimizing $H_{max}(Z_{l,i})$. Rather than directly reducing information about the data, we reduce the maximum capacity of the layer, subject to the constraint that the model does not become smaller than the capacity required to model the problem. The general structure of the algorithm is shown in Algorithm 1.

Algorithm 1: PIBprune

Input: Pretrained Layers = $\{Z_{1..L}\} \in Z$
Data = $\{X, Y\}$
Output: Pruned Layers = $\{Z_{final_{1..L}}\} \in Z_{final}$

repeat
  $Z' := PIBCompression(X, Y, Z)$
  $Z := PIBRecovery(X, Y, Z')$
until *Convergence*
$Z_{final} = Z$
return Z_{final}

In pruning a neural network model, most of the information contained in the features remains intact, but fine tuning the pruned model is usually necessary to recover the accuracy, particularly with node pruning. This allows the network to compensate for the changed structure. Training the neural network model by minimizing the negative log-likelihood of the predictions,

$$NLL(Y, \hat{Y}) = -\log(p(Y = \hat{Y})), \tag{4.5.2}$$

where $\hat{Y} = softmax(Z_L)$, requires maximizing $I(Y; Z_L)$. Thus, the relevance maximization part of the compression-relevance trade-off is already present in the training and fine tuning.

Algorithm 2: PIBRecovery

Input: Layers = $\{Z'_{1..L}\} \in Z'$ with weights $\{W_{1..L}\}$ and biases $\{b_{1..L}\}$;
Data = $\{X, Y\}$
Output: Layers = $\{Z_{1..L}\} \in Z$

$Z := Z'$
$t := 0$
repeat
  $\hat{Y}_t := Z_L \circ Z_{L-1} \circ ...Z_1 \circ X$
  $C_t := NLL(Y, \hat{Y}_t)$
  for $l = 1$ *to* L **do**
    $W := W - \eta \frac{\partial W}{\partial C_t}$
    $b := b - \eta \frac{\partial b}{\partial C_t}$
  end
  $t := t + 1$
until *Convergence*
return Z

We propose to alternate between maximizing $I(Y; Z_L)$ in Algorithm 2 and minimizing $H_{max}(Z_l)$ in Algorithm 1. Maximizing $I(Y; Z_L)$ is a straightforward process of training by stochastic gradient descent, while keeping the size of the network constant. Minimizing $H_{max}(Z_l)$ for each layer requires deciding to prune the maximum number of nodes such that the next training step can recover the accuracy. We use the Information Bottleneck statistics to address the question of choosing which nodes to prune, and when to stop. Because of the granularity of node pruning, we have to be careful with restricting capacity. With each pruned node, we will inevitably throw away some information about the data as well as some unused capacity. We use a conservative estimate on the amount of unused capacity that can be eliminated. We can break down the information contained in the output distribution of each node $Z_{l,i}$ into the following components:

- $H_{max}(Z_{l,i})$, the maximum entropy a node can have,
- $H(Z_{l,i})$, the marginal entropy of the node,
- $H_{max}(Z_{l,i}) - H(Z_{l,i})$, the gap between the node's marginal and maximum entropy
- $H(Z_{l,i}) - H(Z_{l,i} \mid Z_{l-1}) > I(Z_{l,i}; X)$, an upper bound on the node's information about the data

- $H(Z_{l,i} \mid Z_{l-1}) > H(Z_{l,i}) - I(Z_{l,i}; X)$, an upper bound on the the 'noise entropy', or information in the layer *not* about the data and
- $I(Z_{l,i}; Y)$, the information about the label.

We want to constrain the new size of the layer Z_l to be at least:

$$MinWidth(Z_l) = \lceil H_{max}(Z_l) - H(Z_{l,i} \mid Z_{l-1}) \rceil, \tag{4.5.3}$$

which should leave enough capacity to represent the information about the data, plus the gap between the marginal entropy and the maximum entropy, which should leave flexibility in optimization. In choosing which nodes to prune, the best candidates are nodes with a high "noise" entropy $H(Z_{l,i} \mid Z_{l-1})$, which indicates a node containing useless information.

For an efficient representation we would like to delete $\lceil I(Z_l; Z_{l-1}) \rceil$ nodes from the layer Z_i but we need to do this by deleting whole nodes, which contain a mixture of conditional entropy and mutual information. To lose as little useful information as possible, we rank the nodes in order of informativeness with

$$InformScore(Z_{l,i}) = \hat{I}(Z_{l,i}; Z_{l-1}), \tag{4.5.4}$$

where

$$\hat{I}(Z_{l,i}; Z_{l-1}) = H_{max}(Z_{l,i}) - H(Z_{l,i} \mid Z_{l-1}) \geq I(Z_{l,i}; X). \tag{4.5.5}$$

If $InformScore(Z_{l,i}) < \frac{Hmax(Z_{l,i})}{2}$, we have more noise entropy in the node than all other factors combined, and we should be able to prune the node. This gives us a conservative constraint on both the number of nodes pruned per layer, and a requirement for any pruned node to have more noise entropy than useful information.

Algorithm 3: PIBCompression

Input: Layers = $\{Z_{0..L}\} \in Z$
Output: Layers = $\{Z'_{0..L}\} \in Z'$

for $l = 1$ *to* L **do**
  $Z'_l := Z_l$
  $NewWidth(Z_l) := |Z_l|$
  sort$(Z_{l,i} \in Z_l)$ by $InformScore(Z_{l,i})$
  for $i = 1$ *to* $|Z_l|$ **do**
    if $InformScore(Zl, i) \geq \frac{Hmax(Z_l,i)}{2}$ **then**
      continue
    else
      $Z'_l := \{Z'_l\} \setminus \{Z'_{l,i}\}$
      $NewWidth(Z_l) := NewWidth(Z_l) - 1$
      if $NewWidth(Z_l) = MinWidth(Z_l)$ **then**
        break
  end
end
return Z'

The effect of removing nodes with high noise entropy ensures that the ratio of $I(Z_l; Z_{l-1})$ to $H(Z_l \mid Z_{l-1})$ will not decrease (see Algorithm 3). We can then fine tune the network to recover the information lost in the pruning process (see Algorithm 2).

Once fine tuning is complete, we find that some of the information lost is not recovered, despite the accuracy being recovered and there being enough capacity in the model to hold it. Assuming the fine tuning optimization was successful, this is due to two factors: fine tuning may need to recover less information from the layer than what was pruned, ie $I(Z_l; X)$ not shared with $I(Z_l; Y)$. Also, simultaneous compression in other layers may reduce the input information. We let the fine tuning process restore lost information $I(Z_l; X)$ and $I(Z_l; Y)$ needed to recover accuracy, then find a new estimate of the ideal model size in the next step to further reduce the model size, until the

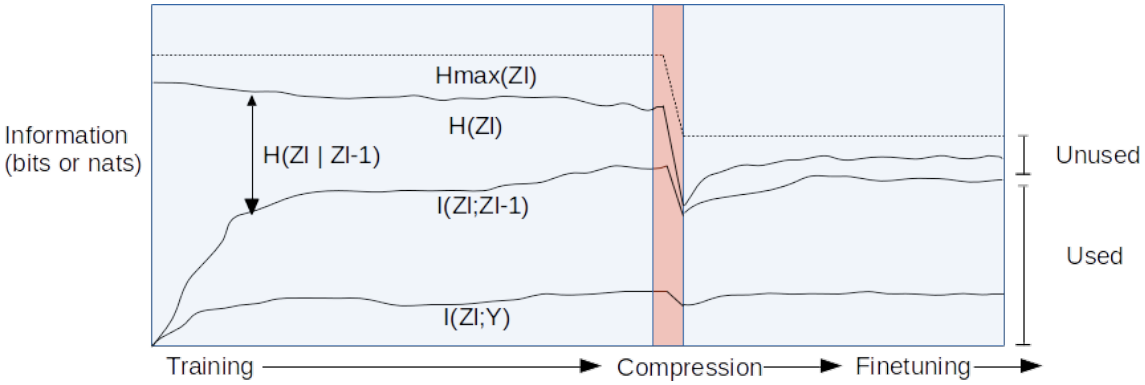


Figure 4.5.1: The PIBprune algorithm compresses up to a conservative bound on the capacity (illustration only).

process converges.

This gives us an iterative process (see Figure 4.5.1) which estimates the ideal layer size, prunes nodes and then fine tunes to recover any lost information useful for the objective function. At each stage a maximum of $\lceil H_{max}(Z_l) - H(Z_l \mid Z_{l-1}) \rceil$ nodes are pruned in each layer. Pruning in each layer is done independently of other layers and fine tuning takes place for all layers at once. We terminate the algorithm when there are no remaining nodes that can be safely pruned in any layer.

4.6 Algorithm Variants

It is possible to relax the above conditions to obtain tighter compression ratios. Firstly, the upper bound on the mutual information

$$InformScore(Z_{l,i}) = \hat{I}(Z_{l,i}; Z_{l-1}) = H_{max}(Z_l) - H(Z_l \mid Z_{l-1}) \tag{4.6.1}$$

can be replaced by the empirical mutual information $I(Z_l; Z_{l-1})$. Similarly, we can change the maximum number of nodes pruned in a single compression stage from $MinWidth(Z_l) = \lceil H_{max}(Z_l) - H(Z_{l,i} \mid Z_{l-1}) \rceil$ to

$$MinWidth(Z_l) = I(Z_{l,i}; Z_{l-1}) = \lceil H(Z_l) - H(Z_{l,i} \mid Z_{l-1}) \rceil. \tag{4.6.2}$$

In our experiments we denote our methods as follows:

- PIBprune-A: $MinWidth(Z_l) = \lceil H_{max}(Z_l) - H(Z_{l,i} \mid Z_{l-1}) \rceil$ and mutual information is replaced by the upper bound $InformScore(Z_{l,i}) = \hat{I}(Z_{l,i}; Z_{l-1}) = H_{max}(Z_l) - H(Z_l \mid Z_{l-1})$ when deciding which nodes to prune
- PIBprune-B: $MinWidth(Z_l) = \lceil H_{max}(Z_l) - H(Z_{l,i} \mid Z_{l-1}) \rceil$ but empirical mutual information is used to decide which nodes to prune, $InformScore(Z_{l,i}) = I(Z_{l,i}; Z_{l-1})$
- PIBprune-C: $MinWidth(Z_l) = \lceil I(Z_{l,i}; Z_{l-1}) \rceil$ and empirical mutual information is used to decide which nodes to prune, $InformScore(Z_{l,i}) = I(Z_{l,i}; Z_{l-1})$

4.7 Implementation Details

We use the Caffe framework's python wrapper to do the neural network training and Numpy to calculate the IB statistics. Node pruning is done by setting weights to zero and fixing the bias to give an output of zero. During fine tuning we reset to zero the weights in the pruned nodes

following every iteration. Convolutional layers are compressed by channel pruning, which is done by summing the IB stats of all the outputs in a feature map and either pruning the channel as a whole or leaving it alone. To measure mutual information we use a validation set of 100 images as our samples to determine the compression rate. Testing is done on separate test data. These are probabilistic formulae but they can be calculated from one forward pass over a deterministic network because the probabilities that we would obtain for stochastic sampling can be read off the network outputs. We can track the IB statistics in each layer at fixed intervals throughout training for visualization purposes, but this is unnecessary for the algorithm, which only uses this information during the pruning phase. This Bernoulli modelling assumption assumes the data are approximately bimodal, with two major clusters and few intermediate values. This allows the pixels to be represented as an event in a binary probability distribution.

4.8 Experiments

We test our algorithm on the MNIST data set. MNIST contains approximately binary (i.e., black and white) images of handwritten digits from 0 to 9. It is widely used to demonstrate new algorithms and is the data set for which we have the most reported results for competing compression methods. The data set has balanced classes, with 60,000 training examples and 10,000 testing examples. We let our algorithm choose compression rates automatically in all our experiments as described in Algorithm 3. Given a good enough training and fine tuning schedule we expect convergence to a smaller model with roughly the same or greater accuracy. We leave the output layer uncompressed. There is no generally agreed upon standard for comparing compression algorithms. Problems arise due to different methodologies; some algorithms use a joint training and compression algorithms, others prune after training and rely on a separate fine tuning process. Depending on the type of compression algorithm, weight pruning, node pruning, channel pruning and quantization may have different advantages on different metrics. The trade-off between accuracy and compression also has to be considered. In our algorithm, fully connected nodes and channels are either pruned or not pruned. There is no compression on the input to the nodes.

To evaluate our model, we look at the model accuracy, the compression ratio of feature maps R_n and the the compression ratio of weights R_w. The compression ratio of feature maps is computed as follows:

$$R_n = \frac{\sum_l |\tilde{Z}_l|}{\sum_l |Z_l|} \tag{4.8.1}$$

where $|Z_l|$ denotes the size of the output of the uncompressed layer l, and $|\tilde{Z}_l|$ the size of the output of the compressed layer. For a fully-connected layer, $|Z_l|$ is simply equal to the number of nodes in the layer. For a convolutional layer $|Z_l| = C_l \times W_l \times H_l$, where C_l is the number of convolutional channels in layer l, with shape $W_l \times H_l$. The compression ratio of weights is computed with

$$R_w = \frac{\sum_l |\tilde{W}_l|}{\sum_l |W_l|}, \tag{4.8.2}$$

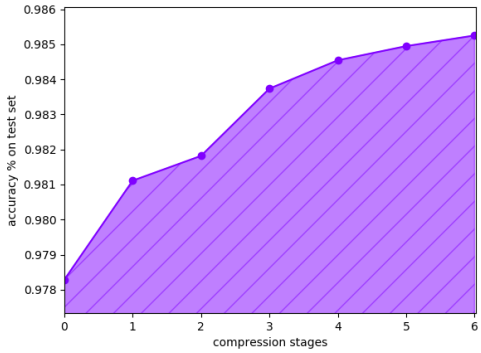
where for fully-connected layers with a weight matrix W_l, with shape $h_l \times w_l$, $|W_l| = h_l \times w_l$ is the number of elements in the weight matrix of layer l. For convolutional layers, the weights form a 4-tensor $|W_l| = C_l \times h_l \times w_l \times K_l$, where C_l is the number of input channels, K_l the number of output channels and $h_l \times w_l$ the size of each 2D convolution kernel. Fully-connected layers with more parameters tend to dominate the R_w measurement, while convolutional layers with large feature maps will dominate the R_n measurement.

In Table 4.1, we show experiments for a 3-layer multilayer perceptron (MLP) with sigmoid activations. We observe that varying the width of the network structure leads to similarly sized compressed models, indicating that the model capacity is being pruned to the necessary size. Adding more redundant capacity simply leads to more compression. This implies a useful method of automatic width selection: start with an overparameterized model and apply the PIBprune

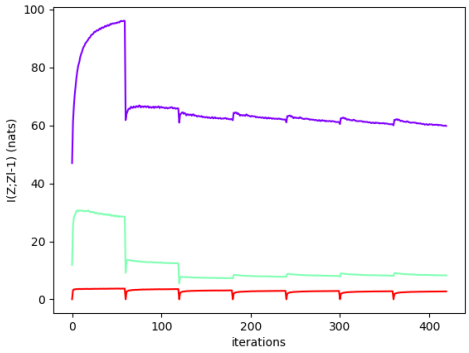
Table 4.1: PIBprune-A using simple 3-layer MLP on MNIST. We show the error of the model before compression (Original) and after compression and finetuning until convergence (Final). Original/Compressed nodes are the number of nodes in each layer of the network. The output layer is shown but not compressed.

Original Nodes	Compressed Nodes	R_w	R_n	Error: (Original)	(Final)
120→100→10	49→8→10	29%	36%	3%	3%
1000→200→10	66→5→10	5.3%	6.7%	3%	3%
1000→500→10	69→6→10	4.2%	5.6%	3%	3%

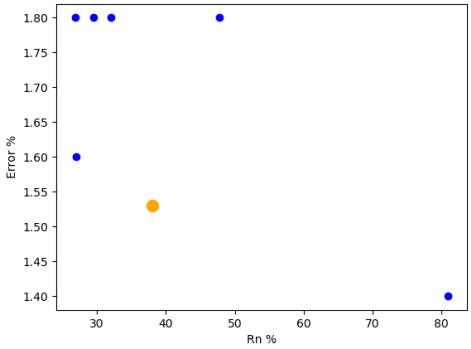
algorithm to shrink it down to the smallest safe size it can find.



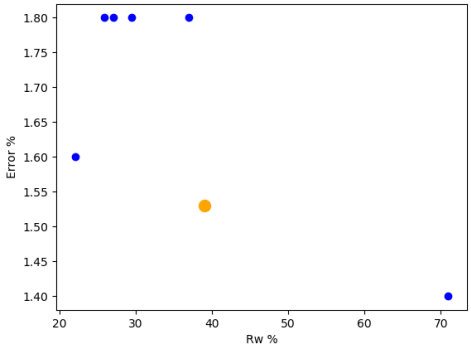
(a) Test Accuracy over time



(b) Mutual Information over time

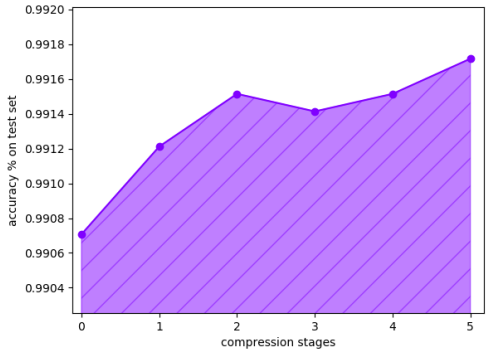


(c) R_n vs test error

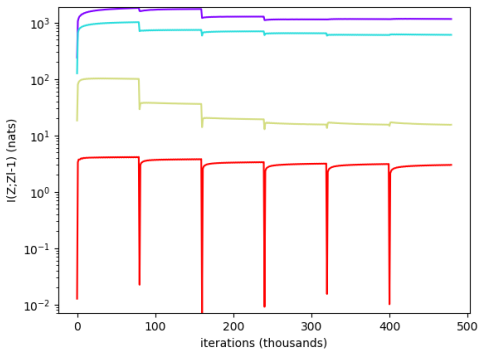


(d) Rw vs test error

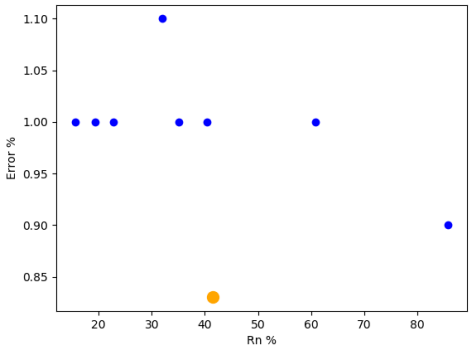
Figure 4.8.1: Compression and accuracy for PIBprune applied to lenet 300-100. $I(Z_l; Z_{l-1})$ is plotted over time for each layer. Layers are colour coded. The purple curve is the first layer, the green curve is the second layer and the red curve the output layer. Figures c) and d) compare our method (orange) to competing methods (blue).



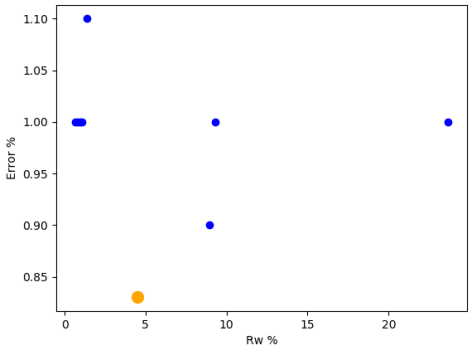
(a) Test Accuracy over time



(b) Mutual Information over time



(c) Rn vs test error



(d) Rw vs test error

Figure 4.8.2: R_n and R_w vs Accuracy for lenet-5. Figure b) shows $I(Z_l; Z_{l-1})$ is plotted over time for each layer. Layers are colour coded. The purple curve is the first layer, the cyan curve is the second layer, the yellow curve is the third layer and the red curve the output layer. Figures c) and d) compare our method (orange) to competing methods (blue).

Table 4.2: Lenet-300-100 on MNIST

Method	Original Nodes	Compressed Nodes	R_w	R_n	Error: (Original)	(Final)
VD[52]	300→100→10	114→72→10	36.9%	47.8%	N/A	1.8%
L0[51]	300→100→10	214→100→10	71%	81%	N/A	**1.4%**
L0-sep[51]	300→100→10	88→33→10	27.1%	31.9%	N/A	1.8%
VIBnet[45]	300→100→10	**71**→33→10	**22%**	27%	N/A	1.6%
BC-GNJ[50]	300→100→10	98→**13**→10	29.4%	29.5%	1.6%	1.8%
BC-GHS[50]	300→100→10	86→14→10	25.8%	**26.8%**	1.6%	1.8%
PIBprune-C-DSN	300→100→10	130→15→10	39%	38%	2.3%	1.53%

Table 4.3: Lenet-5 on MNIST

Method	Original Nodes	Compressed Nodes	R_w	R_n	Error: (Original)	(Final)
GD[62]	20→50→500→10	7→13→16→10	1.37%	32.00%	0.9%	1.1%
GL[63]	20→50→500→10	**3**→12→500→10	23.69%	19.35%	0.9%	1.0%
VD[52]	20→50→500→10	14→19→131→10	9.29%	60.78%	0.9%	1.0%
L0[51]	20→50→500→10	20→25→25→10	8.92%	85.82%	N/A	0.9%
L0-sep[51]	20→50→500→10	9→18→25→10	1.08%	40.36%	N/A	1.0%
BC-GNJ[50]	20→50→500→10	8→13→**13**→10	0.95%	35.03%	0.9%	1.0%
BC-GHS[50]	20→50→500→10	5→**10**→16→10	**0.64**%	22.80%	0.9%	1.0%
VIBnet[45]	20→50→500→10	N/A	0.83	**15.55**%	N/A	1.0%
PIBprune-C-DSN	20→50→500→10	7→26→35→10	4.5%	41.45%	0.93%	**0.83**%

For comparison, in Table 4.2 and Figure 4.8.1 we demonstrate the effect of compression using more powerful training methods. We apply dropout(0.2) to the data, use Orthonormal-EP[46] initialization and Deep Supervision [44] to train a sigmoid version of LeNet-300-100. Because our method has to operate on sigmoid activations, we do not take into account compression on the input when calculating the compression ratios. Competing methods' compression rates are recomputed on the assumption of uncompressed inputs.

In Figures 4.8.1 (b) and 4.8.2 (b) we can see the effects of repeated compression and fine tuning on the information content of the network. Each layer's $I(Z_l; Z_{l-1})$ increases and saturates as training converges. The compression phase induces a sharp drop and another climb and saturation as the training program recovers accuracy. The changes become smaller in magnitude with each iteration until the algorithm terminates.

Convolutional networks are harder because node pruning is replaced by channel pruning, and it is even harder to separate informative from uninformative channels than it is for nodes. However, in terms of memory usage, each channel deleted removes an entire feature map, yielding a great reduction saving in memory usage, if this can be accomplished. In Table 4.3 and Figure 4.8.2 we present comparisons on Lenet-5 models on MNIST. Again, in contrast to the other examples, our model uses sigmoid activations at all layers, rather than linear and ReLU activations, to model the IB statistics. To train the model we apply Deep Supervision. We note that VIBnet and GD also specify non-standard training objectives.

In both MNIST models for which we have data for competing compression methods, our PIBprune method is conservative enough that pruned models consistently remain more accurate than most of our competitors, while still providing competitive compression ratios. In particular we do better than another conservative method, L0. We do this *without* having to set compression hyperparameters. The compression rate is instead governed by the properties of the trained model. All other things being equal, increasing the model size initially will result in a relatively small change to the compressed size. We attempted to extend our method to colour input images but it was not compatible with our modelling assumptions. We rely on an equivalence between deterministic nodes and the expected value of *binary* stochastic nodes. Our attempts to experiment with this produced unreliable entropy and mutual information estimates. We suspect this to also be a weakness of the Parametric Information Bottleneck that our method is based upon. Note that the problem is not multiple *colour channels*, but non-bimodally distributed inputs.

4.9 Discussion and Future Work

Principled model compression algorithms are still in their infancy, but information theory is rapidly shedding light on exactly how much capacity is really needed to model a problem. We have presented a novel iterative model compression algorithm based on the Information Bottleneck Principle. While our model requires binary data and sigmoid activations to work, we nonetheless show competitive results on MNIST in terms of accuracy vs compression on weights and feature maps.

Unlike previous compression algorithms, our method finds compression rates automatically rather than relying on hyperparameters chosen by the user. In a large network, setting compression rates is a non-trivial task as it is not known a priori how much capacity is needed for each layer to model the data at that stage of processing. We see three main directions to extend this work. Firstly, our method only looks at the information content of activations and assumes that the information in the weights is similar. There exist other methods such as Bayesian Compression that instead finds the information content of the model weights subject to some prior. A combination of the two methods would likely perform better. Secondly, because of the Bernoulli distribution used to model the information content of the activations in our method, our model cannot be used for non-binary data, ReLU activations or compression on the input to the model. A modelling strategy that allows more application would be useful here, but most of these require introducing parameters to the modelling that are not dependant on the original model, but must be learned during training. Finally, our model does not fully exploit the regularization effect of the Information Bottleneck method during training, where it could further improve accuracy. Using the Information Bottleneck in the optimization stage instead of just for analysis and pruning would be very effective.

Chapter 5

Conclusion

We have explored regularization and compression in neural networks and made two contributions to the knowledge in this field. Our work on Region of Interest Autoencoders applied to Pedestrian detection has shown that joint training with autoencoders can be applied to large scale detection problems. By better exploiting all the information in the images in the data set rather than just the image/label correlations we can improve the generalization of the model without requiring more speed. By focusing on foreground reconstruction we encouraged better modelling of a single class, rather than the entire domain of photographic data present in the background. This enabled us to efficiently regularize our model of the pedestrians. In our work on compressing neural network models using the Parametric Information Bottleneck we demonstrate that neural networks can be compressed by modelling the information content that each layer preserves about the data, and reach a good compression rate for node pruning automatically.

The state of the art has now advanced to the point where the Information Bottleneck Method is being used both to better regularize neural network models and to compress them. Neural network layers do not create information, but they select which information from the input to keep, and which to eliminate. Keeping more information implies a larger model downstream, and it can also lead to overfitting. Because most models are overparameterized, reducing the information that the layers pass to the next layer can lead to a regularizing and a compression effect. We conjecture that a unified framework, where regularization and compression algorithms will be developed in the near future. The key difference in regularization and compression methods is that regularization methods, like weight decay and dropout, leave the feature dimensions intact but convey less information per dimension, while compression methods need to be coarser if inference can be done on the compressed model. Compromise methods can be reached, such as creating a sparse methods to efficiently exploit weight-pruned models. The Information Bottleneck does not discuss the information contained in the weights themselves. This is addressed by the Minimum Description Length (MDL), and a combination method that analyses both the activations and the weights of the model may do better.

Bibliography

[1] Girshick, Ross B., Jeff Donahue, Trevor Darrell and Jitendra Malik.
Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation, 2014 IEEE Conference on Computer Vision and Pattern Recognition (2014): 580-587.

[2] Girshick, Ross B..
Fast R-CNN, 2015 IEEE International Conference on Computer Vision (ICCV) (2015): 1440-1448.

[3] Ren, Shaoqing, Kaiming He, Ross B. Girshick and Jian Sun.
Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, IEEE Transactions on Pattern Analysis and Machine Intelligence 39 (2015): 1137-1149.

[4] Liliang Zhang, Liang Lin, Xiaodan Liang, Kaiming He
Is Faster R-CNN Doing Well for Pedestrian Detection?, ECCV 2016.

[5] A. Geiger, C. Wojek, B. Schiele, P. Perona.
Pedestrian Detection: A Benchmark, CVPR 2009.

[6] Dollr, Piotr, Christian Wojek, Bernt Schiele and Pietro Perona.
Pedestrian Detection: An Evaluation of the State of the Art, IEEE Transactions on Pattern Analysis and Machine Intelligence 34 (2012): 743-761.

[7] Cho H, Seo YW, Kumar BV, Rajkumar RR.
A multi-sensor fusion system for moving object detection and tracking in urban driving environments, In Robotics and Automation (ICRA), 2014 pp. 1836-1843).

[8] J. Ponce, T. Berg, M. Everingham, D. Forsyth, M. Hebert, S. Lazebnik, M. Marszaek, C. Schmid, C. Russell, A. Torralba, C. Williams, J. Zhang, and A. Zisserman.
Data set issues in object recognition, In Towards Category-Level Object Recognition, pages 2948. Springer, 2006.

[9] Hosang, Jan Hendrik, Mohsen Samet Omran, Rodrigo Benenson and Bernt Schiele.
Taking a deeper look at pedestrians 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2015): 4073-4082.

[10] Liang, Xiaodan, Shengmei Shen, Tingfa Xu, Jiashi Feng and Shuicheng Yan.
Scale-Aware Fast R-CNN for Pedestrian Detection, IEEE Transactions on Multimedia 20 (2018): 985-996.

[11] Du, Xianzhi, Mostafa El-Khamy, Jungwon Lee and Lloyd S. Davis.
Fused DNN: A Deep Neural Network Fusion Approach to Fast and Robust Pedestrian Detection. 2017 IEEE Winter Conference on Applications of Computer Vision (WACV) (2017): 953-961.

[12] Liu, Weiwei, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu and Alexander C. Berg.
SSD: Single Shot MultiBox Detector, ECCV (2016).

[13] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, Trevor Darrell
Caffe: Convolutional Architecture for Fast Feature Embedding, Proceedings of the 22nd ACM international conference on Multimedia, 675-678, 2014.

[14] Krizhevsky, Alex, Ilya Sutskever and Geoffrey E. Hinton.
ImageNet Classification with Deep Convolutional Neural Networks, Commun. ACM 60 (2012): 84-90.

[15] Ioffe, Sergey and Christian Szegedy.
Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, ArXiv abs/1502.03167 (2015): n. pag.

[16] Karen Simonyan and Andrew Zisserman
Very Deep Convolutional Networks for Large-Scale Image Recognition, CoRR abs/1409.1556 (2014)

[17] Zhaowei Cai, Mohammad Saberian, Nuno Vasconcelos
Learning Complexity-Aware Cascades for Deep Pedestrian Detection, 2015 IEEE International Conference on Computer Vision (ICCV) (2015): 3361-3369.

[18] Pierre Sermanet, Koray Kavukcuoglu, Soumith Chintala, Yann LeCun
Pedestrian Detection with Unsupervised Multi-Stage Feature Learning, 2013 IEEE Conference on Computer Vision and Pattern Recognition (2012): 3626-3633.

[19] Yann leCun, Leon Bottou, Yoshua Bengio, Patrick Haffner
Gradient-Based Learning Applied to Document Recognition, Proceedings of the IEEE, 86(11):2278-2324, November 1998.

[20] Hyeonwoo Noh Seunghoon Hong Bohyung Han
Learning Deconvolution Network for Semantic Segmentation, 2015 IEEE International Conference on Computer Vision (ICCV) (2015): 1520-1528.

[21] Piotr Dollar, Ron Appel, Serge Belongie, and Pietro Peron
Fast Feature Pyramids for Object Detection,IEEE Transactions on Pattern Analysis and Machine Intelligence 36 (2014): 1532-1545.

[22] Kunihiko Fukushima
Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position, Biological Cybernetics 36 (1980): 193-202.

[23] Geiger, Andreas, Philip Lenz and Raquel Urtasun
Are we ready for autonomous driving? The KITTI vision benchmark suite, 2012 IEEE Conference on Computer Vision and Pattern Recognition (2012): 3354-3361.

[24] A. Ess and B. Leibe and K. Schindler and and L. van Gool
A Mobile Vision System for Robust Multi-Person Tracking, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2008.

[25] M. Enzweiler and D. M. Gavrila
Monocular Pedestrian Detection: Survey and Experiments, IEEE Trans. on Pattern Analysis and Machine Intelligence, vol.31, no.12, pp.2179-2195, 2009.

[26] Rodrigo Benenson, Mohamed Omran, Jan Hosang, Bernt Schiele
Ten Years of Pedestrian Detection, What Have We Learned?, ECCV Workshops 2014.

[27] Antti Rasmus, Harri Valpola, Mikko Honkala, Mathias Berglund, Tapani Raiko
Semi-Supervised Learning with Ladder Networks, NIPS 2015.

[28] Jonathan Long, Evan Shelhamer, Trevor Darrell
Fully Convolutional Networks for Semantic Segmentation, CVPR 2015.

[29] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, Li Fei-Fei
ImageNet Large Scale Visual Recognition Challenge, IJCV 2015.

[30] Y. Tian, P. Luo, X. Wang, and X. Tang
Deep Learning Strong Parts for Pedestrian Detection, 2015 IEEE International Conference on Computer Vision (ICCV) (2015): 1904-1912.

[31] Alex Krizhevsky
Learning Multiple Layers of Features from Tiny Images, masters thesis, University of Toronto 2009.

[32] Adam Coates, Honglak Lee, Andrew Y. Ng
An Analysis of Single Layer Networks in Unsupervised Feature Learning, AISTATS 2011.

[33] Abhinav Shrivastava, Abhinav Gupta, Ross Girshick
Training Region-based Object Detectors with Online Hard Example Mining, 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016): 761-769.

[34] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun
Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification, 2015 IEEE International Conference on Computer Vision (ICCV) (2015): 1026-1034.

[35] Naftali Tishby and Fernando C. Pereira and William Bialek,
The Information Bottleneck Method 1999, In Proceedings of the 37th Annual Allerton Conference on Communication, Control and Computing, pp. 368-377

[36] Ohad Shamir and Sivan Sabato and Naftali Tishby 2010,
Learning and Generalization with the Information Bottleneck, Journal of Theoretical Computer Science, volume 411, pages 2696-2711

[37] Naftali Tishby and Noga Zaslavsky,
Deep Learning and the Information Bottleneck Principle 2015 IEEE Information Theory Workshop (ITW) (2015): 1-5.

[38] Nguyen, T. T. and Choi, J,
Layer-wise Learning of Stochastic Neural Networks with Information Bottleneck, ArXiv preprint abs/1712.01272 (2017)

[39] Alexander A. Alemi and Ian Fischer and Joshua V. Dillon and Kevin Murphy 2016
Deep Variational Information Bottleneck ArXiv abs/1612.00410 (2017)

[40] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent,
Why Does Unsupervised Pre-training Help Deep Learning? JMLR 2010.

[41] Han, Song and Pool, Jeff and Tran, John and Dally, William 2015,
Learning both Weights and Connections for Efficient Neural Networks Advances in Neural Information Processing Systems, Volume 28, pages 1135-1143

[42] Yann Dauphin, Razvan Pascanu, Caglar Gülehre, Kyunghyun Cho, Surya Ganguli and Yoshua Bengio 2014
Identifying and attacking the saddle point problem in high-dimensional non-convex optimization, Advances in Neural Information Processing Systems (NIPS)

[43] Yihui He, Xiangyu Zhang and Jian Sun
Channel Pruning for Accelerating Very Deep Neural Networks, Proceedings of the 2017 IEEE International Conference on Computer Vision (ICCV), pages 1398-1406

[44] Lee, Chen-Yu and Xie, Saining and Gallagher, Patrick W. and Zhang, Zhengyou and Tu, Zhuowen 2015
Deeply-Supervised Nets, Journal of Machine Learning Research, Volume 38

[45] Bin Dai, Yiyang Zhuang and David P. Wipf
Compressing Neural Networks using the Variational Information Bottleneck, ICML 2018

[46] Sansone, Emanuele and Natale, F.G.B. 2017
Training Feedforward Neural Networks with Standard Logistic Activations is Feasible, ArXiv abs/1710.01013 (2017)

[47] Kolchinsky, Artemy and Tracey, Brendan D.
Estimating Mixture Entropy with Pairwise Distances, Entropy 19 (2017): 361.

[48] Shuyang Gao, Greg Ver Steeg and Aram Galstyan, 2015
Efficient Estimation of Mutual Information for Strongly Dependent Variables, International Conference on Artificial Intelligence and Statistics (AISTATS)

[49] Hinton, Geoffrey E. and van Camp, Drew 1993
Keeping the Neural Networks Simple by Minimizing the Description Length of the Weights, Proceedings of the Sixth Annual Conference on Computational Learning Theory (COLT) pages 5-13

[50] Christos Louizos and Karen Ullrich and Max Welling 2017
Bayesian Compression for Deep Learning, Advances in Neural Information Processing Systems (NIPS)

[51] Louizos, Christos, Welling, Max, and Kingma, Diederik P. 2017
Learning sparse neural networks through L0 regularization CoRR, 2017b.

[52] Kingma, Diederik P, Salimans, Tim, and Welling, Max.
Variational dropout and the local reparameterization trick, In NIPS, pp. 25752583, 2015

[53] Song Han, Huizi Mao and William J. Dally
Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding, CoRR abs/1510.00149 (2015)

[54] Yihui He and Song Han 2018
ADC: Automated Deep Compression and Acceleration with Reinforcement Learning, ECCV 2018

[55] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz and William J Dally 2016
EIE: Efficient Inference Engine on Compressed Deep Neural Networks, Proceedings of the 43rd International Symposium on Computer Architecture (ISCA), pages 243-254

[56] LeCun, Yann and John S. Denker and Sara A. Solla 1990
Optimal Brain Damage, Advances in Neural Information Processing Systems volume 2, pages 598-605

[57] Babak Hassibi, David G. Stork, Gregory Wolff and Takahiro Watanabe 1993
Optimal Brain Surgeon: Extensions and Performance Comparisons, Neural Information Processing Systems (NIPS) pages 263-270

[58] Xin Dong, Shangyu Chen and Sinno Jialin Pan 2017
Learning to Prune Deep Neural Networks via Layer-wise Optimal Brain Surgeon, ArXiv abs/1705.07565 (2017)

[59] Ravid Shwartz-Ziv and Naftali Tishby 2017
Opening the Black Box of Deep Neural Networks via Information ArXiv abs/1703.00810 (2017)

[60] Denil, Misha and Shakibi, Babak and Dinh, Laurent and Ranzato, Marc Aurelio and de Freitas, Nando 2013
Predicting Parameters in Deep Learning, Advances in Neural Information Processing Systems, volume 26, pages 2148-2156

[61] Alireza Aghasi and Nam Nguyen and Justin Romberg 2017
Net-Trim: A Layer-wise Convex Pruning of Deep Neural Networks, Advances in Neural Information Processing Systems (NIPS) pages 3180-3189

[62] Suraj Srinivas and R Venkatesh Babu
Generalized Dropout, ArXiv abs/1611.06791 (2016)

[63] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li
Learning Structured Sparsity in Deep Neural Networks. In NIPS, pp. 20742082, 2016

[64] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever and R. R. Salakhutdinov
Improving neural networks by preventing co-adaptation of feature detectors, ArXiv abs/1207.0580 (2012)

[65] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever and R. R. Salakhutdinov
Dropout: A Simple Way to Prevent Neural Networks from Overfitting, Journal of Machine Learning Research 15 (2014)

[66] Wan, Li and Zeiler, Matthew and Zhang, Sixin and LeCun, Yann and Fergus, Rob
Regularization of neural networks using Dropconnect, Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28 (2013)

[67] Lowe, DG.
Distinctive Image Features from Scale-invariant Keypoints, International Journal of Computer Vision 60.2 (2004): 91. Web.

[68] Navneet Dalal and Bill Triggs
Histograms of Oriented Gradients for Human Detection, In CVPR 2005, pp. 886-893

[69] David J. C. Mackay
A Practical Bayesian Framework for Backprop Networks, Neural Computation (1991)

[70] L. R. Tucker
Implications of factor analysis of three-way matrices for measurement of change, in Problems in Measuring Change, University of Wisconsin Press (1963), pp. 122-137.

[71] Gregory J. Chaitin
On the Simplicity and Speed of Programs for Computing Infinite Sets of Natural Numbers, J. ASSOC. COMPUT. MACH (1969), vol. 16, pp 407-422.

[72] Ian Goodfellow and Yoshua Bengio and Aaron Courville
Deep Learning, book published by MIT Press (2016), Chapter 5 "Machine Learning Basics"

[73] Grunwald, Peter
The Minimum Description Length Principle
book published by MIT Press (2007) 10.7551/mitpress/4643.001.0001.

[74] C. S. Wallace and D. M. Boulton
An information measure for classification, Computer Journal, Vol 11, No 2, August 1968, pp 185-194, http://www.allisons.org/ll/MML/Structured/1968-WB-CJ/

[75] Yoshua Bengio and Yann Lecun
Scaling learning algorithms towards AI, book chapter in Large-Scale Kernel Machines (2007)

[76] David H. Wolpert and William G. Macready
No Free Lunch Theorems for Optimization IEEE Transactions on Evolutionary Computation, vol.1 (April 1997)

[77] V. N. Vapnik and A. Ya. Chervonenkis
On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities, Theory of Probability and its Applications, vol. 16(2), pp. 264280, 1971. https://epubs.siam.org/doi/10.1137/1116025

[78] Anders Krogh and John A. Hertz
A Simple Weight Decay can Improve Generalization, Advances in Neural Information Processing Systems 4 (NIPS 1991)

[79] C. M. Bishop
Training with Noise is Equivalent to Tikhonov Regularization, in Neural Computation, vol. 7, no. 1, pp. 108-116, Jan. 1995.

[80] Snoek J, Larochelle H, Adams RP
Practical bayesian optimization of machine learning algorithms. In Advances in neural information processing systems 2012 (pp. 2951-2959).

[81] Smolensky, Paul (1986)
Chapter 6: Information Processing in Dynamical Systems: Foundations of Harmony Theory, In Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations. MIT Press. pp. 194281.

[82] Hinton, G. E.; Salakhutdinov, R. R. (2006)
Reducing the Dimensionality of Data with Neural Networks. Science. 313 (5786): 504507.

[83] Geoffrey E. Hinton, Simon Osindero, Yee-Whye Teh
A Fast Learning Algorithm for Deep Belief Nets, Neural Computing 2006.

[84] Salakhutdinov R, Larochelle H
Efficient learning of deep Boltzmann machines. In Proceedings of the thirteenth international conference on artificial intelligence and statistics 2010 Mar 31 (pp. 693-700).

[85] Glorot X, Bordes A, Bengio Y
Deep sparse rectifier neural networks. In Proceedings of the fourteenth international conference on artificial intelligence and statistics 2011 Jun 14 (pp. 315-323).

[86] Bengio Y, Lamblin P, Popovici D, Larochelle H
Greedy layer-wise training of deep networks. In Advances in neural information processing systems 2007 (pp. 153-160).

[87] Rifai S, Mesnil G, Vincent P, Muller X, Bengio Y, Dauphin Y, Glorot X
Higher order contractive auto-encoder. InJoint European Conference on Machine Learning and Knowledge Discovery in Databases 2011 Sep 4 (pp. 645-660). Springer, Berlin, Heidelberg.

[88] Vincent P, Larochelle H, Bengio Y, Manzagol PA
Extracting and composing robust features with denoising autoencoders. In Proceedings of the 25th international conference on Machine learning 2008 Jul 5 (pp. 1096-1103). ACM.

[89] Vincent P, Larochelle H, Lajoie I, Bengio Y, Manzagol PA
Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. Journal of machine learning research. 2010;11(Dec):3371-408.

[90] Guilliame Alain and Yoshua Bengio
What Regularized Auto-Encoders Learn from the Data-Generating Distribution, Journal of Machine Learning Research 15 (2014) 3743-3773

[91] Junbo Zhao, Michael Mathieu, Ross Goroshin, Yann LeCun
Stacked What-Where Auto-encoders ArXiv abs/1506.02351 (2015)

[92] Yuting Zhang, Kibok Lee, Honglak Lee
Augmenting Supervised Neural Networks with Unsupervised Objectives for Large-scale Image Classification, ICML 2016.

[93] Kim YD, Park E, Yoo S, Choi T, Yang L, Shin D
Compression of deep convolutional neural networks for fast and low power mobile applications. arXiv preprint arXiv:1511.06530. 2015 Nov 20.

[94] Lebedev V, Ganin Y, Rakhuba M, Oseledets I, Lempitsky V
Speeding-up convolutional neural networks using fine-tuned cp-decomposition. arXiv preprint arXiv:1412.6553. 2014 Dec 19.