10-19-2020

Exploring the Potential of Sparse Coding for Machine Learning

Sheng Yang Lundquist
Portland State University

Recommended Citation
Lundquist, Sheng Yang, "Exploring the Potential of Sparse Coding for Machine Learning" (2020). *Dissertations and Theses.* Paper 5612.
https://doi.org/10.15760/etd.7484

Exploring the Potential of Sparse Coding for Machine Learning

by

Sheng Y. Lundquist

A dissertation submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy
in
Computer Science

Dissertation Committee:
Melanie Mitchell, Chair
Feng Liu
Bart Massey
Garrett Kenyon
Bruno Jedynak

Portland State University
2020

Abstract

While deep learning has proven to be successful for various tasks in the field of computer vision, there are several limitations of deep-learning models when compared to human performance. Specifically, human vision is largely robust to noise and distortions, whereas deep learning performance tends to be brittle to modifications of test images, including being susceptible to adversarial examples. Additionally, deep-learning methods typically require very large collections of training examples for good performance on a task, whereas humans can learn to perform the same task with a much smaller number of training examples.

In this dissertation, I investigate whether the use of a biologically informed, unsupervised sparse coding algorithm can help to alleviate these shortcomings within classification networks. I find that (1) the non-linear encoding scheme of convolutional sparse coding, as opposed to the dictionary learned, contributes to classification performance when used within a model. In addition, (2) sparse coding helps classification models trained on clean images to be more robust to adversarial examples and images corrupted with high frequency noise. Finally, (3) sparse coding helps alleviate the number of human-annotated training labels needed for classification on stereo-video data. Overall, using unsupervised sparse coding within supervised models can help alleviate various shortcomings of traditional deep neural networks.

Dedication

To my mother Jie Lundquist, my best friend Mackenzie Gray, and the best boy Bowser. Without your love and support, this dissertation would have forever been unfinished.

Artist: Laura Connelly

Acknowledgements

First and foremost, I'd like to thank my friend and colleague Dylan Paiton. Our collaborations and endless discussions has pushed me to be a better scientist, and I value your feedback in all of my work.

I would also like to thank John Lipor, Anthony Rhodes, Max Quinn, Jordan Witte, Erik Cosner, Will Landecker, Callie Bee, and Mackenzie Gray for valuable feedback regarding work done for this dissertation. Additionally, I'd like to thank my dissertation committee Melanie Mitchell, Feng Liu, Bart Massey, Garrett Kenyon, and Bruno Jedynak for your feedback. The supervised image classification section was work done in conjunction with Melanie Mitchell. The adversarial examples section was work done in conjunction with Dylan Paiton and Melanie Mitchell. The limited data section was work done in conjunction with Melanie Mitchell and Garrett Kenyon.

I would like to thank Garrett Kenyon in particular for introducing me to the field of computational neuroscience and machine learning, as well as the continued support and advisement. Thanks also to all my colleagues in the Kenyon lab.

I'd like to thank the people of the ultimate community for the countless friendships I've made there, while also keeping me healthy both physically and mentally. I can't wait to see all of you again soon.

Finally, I'd like to thank my PhD advisor Melanie Mitchell for overseeing and providing endless feedback and advice through my time at Portland State University. I couldn't have done it without you.

Table of Contents

Abstract i

Dedication ii

Acknowledgements iii

List of Tables vii

List of Figures viii

1 Introduction 1

2 Background for Deep Neural Networks 4

2.1 Supervised vs. Unsupervised Learning 4

2.2 Deep Convolutional Neural Networks 6

2.2.1 Fully Connected Layer 7

2.2.2 Convolutional Layer 8

2.2.3 Pooling Layer 10

2.3 Gradient Descent 10

2.4 Cross-Entropy Supervised Loss Function 13

3 Background for Sparse Coding 14

3.1 Encoding 16

3.2 Dictionary Learning 17

3.3 Convolutional Sparse Coding 18

3.4 Implementation 19

4 Exploring Convolutional Sparse Coding for Supervised Image Classification 22

4.1 Related Work 24
4.2 Experiments 26
4.2.1 Dataset 27
4.2.2 Dictionary Learning Methods 28
4.2.3 Encoding Methods 31
4.2.4 Classifier 32
4.2.5 Fully Supervised Model 33
4.3 Results 34
4.4 Summary 37
5 Sparse Coding for Model Robustness 39
5.1 Background on Adversarial Examples 41
5.2 Related Work 43
5.3 Robustness of Sparse Coding on Adversarial Examples 46
5.3.1 Effect of Sparsity on Adversarial Robustness 50
5.3.2 Calibration of Classification Models 52
5.4 Robustness of Sparse Coding on Corrupted Images 56
5.5 Summary 60
6 Sparse Coding with Limited Labels 61
6.1 Related Work 61
6.2 Sparse Coding of Stereo-Video for Vehicle Detection 63
6.2.1 Experiments 65
6.2.2 Results 68
7 Final Conclusions 74

References 77

Appendix A Sparse Coding Results 82

Appendix B Effect of Sparsity on Classification 88

Appendix C Adversarial Attacks on Sparse Coding 90

Appendix D Classification on Corrupted Images 94

List of Tables

1 Encoding versus dictionary learning accuracy 34

2 ECE of classification models 54

3 Average precision scores on stereo-video classification 69

List of Figures

1 Autoencoder illustration 5

2 Deep convolutional neural network illustration 6

3 Fully connected layer illustration 7

4 Convolutional layer illustration 9

5 Example dictionary on natural images 15

6 Convolutional sparse coding illustration 20

7 CIFAR-10 example images 27

8 Encoding versus dictionary learning accuracy range 36

9 Adversarial image illustration 42

10 Adversarial images attacking fully supervised and convolutional sparse coding models 48

11 Accuracy versus mean adversarial distance 49

12 Effect of sparisty on adversarial robustness 51

13 Range of ECE between fully supervised and convolutional sparse coding 53

14 Range of ECE for calibrated models 55

15 Accuracy versus mAD for calibrated models 55

16 CIFAR-10-C corrupted images examples 57

17 Mean corruption error on types of corruption 59

18 Number of training labels versus area under curve 71

19 Activation map comparison on stereo-video 72

1 Introduction

The field of machine learning has made immense progress in automating vision tasks such as image classification and object detection (e.g., [23,56,63]) within the past decade, with some studies claiming performance near or surpassing humans (e.g., [23,68]). Most if not all of these studies employ a deep convolutional neural network (DCNN) trained through supervised learning (i.e., training from hand-labeled data). Despite the success of DCNNs, there still exist limitations to what these types of models can do.

A clear indicator of the limitations of deep learning is to compare it to biological vision. In terms of image classification, humans are generally able to successfully classify images given a few examples of a class [34]. In contrast, supervised deep learning pipelines require millions of examples with corresponding ground truth labels that are typically labeled by humans (from datasets such as CIFAR-10 [32] or ImageNet [8]). Although humans have years of infancy to "train" our visual systems, we do so largely without explicit supervision [74]. Here, one potential solution for deep learning's need for huge labeled training sets is to use unsupervised training (i.e., training from data without explicit labels) within a supervised DCNN to help alleviate the amount of labeled training data required for a task.

Another limitation of deep learning is the lack of robustness in such models. Previous studies have shown that human performance on vision tasks dramatically surpass that of DCNNs when test images are blurred or distorted [9,15,16,28], which suggests that deep learning is brittle and dependent on superficial image statistics rather than human-like image understanding (e.g., the presence of an animal within a picture can be detected by looking at the blurriness of the background rather than "understanding" the animal [35]). Furthermore, there have been recent studies showing that DCNNs tend to be sensitive to adversarial examples [67], i.e., targeted perturbations that is imperceptible to humans but that completely changes the output of DCNNs. In all, these results show that deep-learning based classifiers and object detectors can be brittle to modifications to which humans are robust.

Sparse coding [50] is an unsupervised learning algorithm that aims to create efficient, non-redundant (i.e., sparse) encodings of an input (e.g., photographs and videos). The idea of sparse coding was originally inspired by theories of neural computation from Barlow and colleagues [3]. In particular, sparse coding has been shown to exhibit similar properties to biological neurons in the early stages of mammalian visual processing [50,77]. It follows that investigating biologically inspired algorithms could provide novel insights into alleviating the aforementioned limitations of deep learning.

In this dissertation, I explore the use of sparse coding within deep learning, and to test the effect on learning with limited data and for performance generalization. Specifically, I address the following questions:

1. What is the relative contribution of the network weights trained by unsupervised dictionary learning versus the activations from the nonlinear encoding computation of sparse coding when used within a supervised network for image classification (Section 4)?

2. Does using unsupervised sparse coding within a supervised network help the model be more robust to adversarial and corrupted images (Section 5)?
3. Does unsupervised sparse coding help reduce the number of labeled training examples needed by typical deep learning pipelines (Section 6)?

2 Background for Deep Neural Networks

2.1 Supervised vs. Unsupervised Learning

An artificial neural network is a type of machine learning model that aims to learn a nonlinear mapping from an input signal to a desired output. Here, the input can be from various domains, such as images, videos, time series, or text. The output can vary as well, such as a single label for supervised whole image classification, coordinates of a bounding box around an object of interest within an image for supervised object detection, or a reconstruction of the input image itself for unsupervised autoencoders.

In supervised learning, the objective of the network is to learn a mapping from an input (e.g., an image) to a target (e.g., an image label). In this domain of learning, a model requires corresponding inputs and targets for training, with human annotators typically providing the ground-truth targets. To train a supervised neural network for an image labeling task, the network takes an image as input and returns a "prediction": an object class label or a probability distribution over possible object class labels. An error is calculated based on some metric of similarity (e.g., cross-entropy distance, see Section 2.4) between the true class provided by ground-truth annotations and the prediction from the neural network. This error drives learning, which adjusts the weights of the neural network such that when the same image is presented again, the new prediction will be closer to the true class than before. The goal is to generalize to unseen examples after training on a sufficient number of training examples.

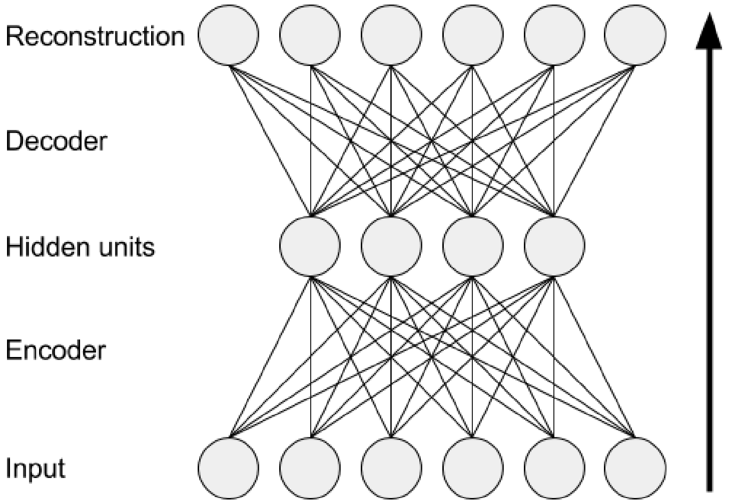


Figure 1: Illustration of an autoencoder. The autoencoder aims to encode input data to activations over hidden units. The goal is to learn the encoder and decoder weights such that the input is reconstructed with minimal degradation.

In contrast with supervised learning, unsupervised learning aims to learn structure from the data without the use of ground truth labels. For example, clustering algorithms such as K-means [41], which aim to find clusters from a set of data points, are considered to be a (non-neural) unsupervised learning algorithm. In the domain of neural networks, one form of unsupervised learning is an autoencoder, whose objective is to map input data to activations over hidden units such that the original data is recoverable with minimal degradation. Figure 1 illustrates this concept. During training, the network aims to find the best weights to minimize the difference between the original data and the reconstruction. Constraints are usually added to the objective to avoid the degenerate solution of an identity mapping. For example, a bottleneck autoencoder (shown in Figure 1) constrains the dimensions of the hidden units to be less than the dimensions of the image, which can be useful for image compression [2]. Such a representation can also decompose images in terms of recurring structures (e.g., edges), which can be more useful for image classification than using raw pixels as input to a supervised classifier [55].

2.2 Deep Convolutional Neural Networks

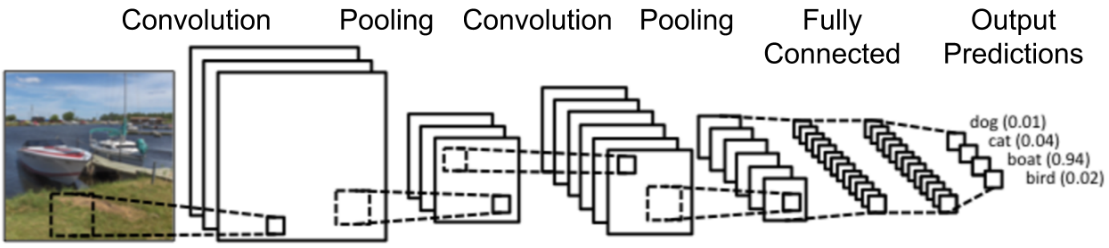


Figure 2: Illustration of a Deep Convolutional Neural Network (DCNN). DCNNs are comprised of different layers. Three common types of layers are convolutional, pooling, and fully connected layers. These layers are stacked in a hierarchy such that the output activations from one layer are fed into the next layer as input. Figure from [5].

When applied to image recognition tasks, Deep Convolutional Neural Networks (DCNNs) are trained to represent a nonlinear mapping from an input image to a class label (e.g., image category). Here, a neural network is composed of layers, where a single layer encapsulates an operation on the input to the layer. These layers are stacked in a hierarchy to form a deep network (the depth of the network corresponds to the number of layers that are stacked together) such that the output activations of a layer are fed as the input to the next layer. Figure 2 illustrates this concept. Three common types of layers are *fully connected* (Section 2.2.1), *convolutional* (Section 2.2.2), and *pooling* (Section 2.2.3) layers.

2.2.1 Fully Connected Layer

A fully connected layer (illustrated in Figure 3) applies an operation to an input vector, which can either be an input signal or the output of another layer within a DCNN. One common use of a fully connected layer is within a network that produces a single label for an input image. In this case, the model uses a fully connected layer at the end of the network to reduce the dimension of the output to the number of classes so that the output of the network can be interpreted as a probability distribution over all possible classes.

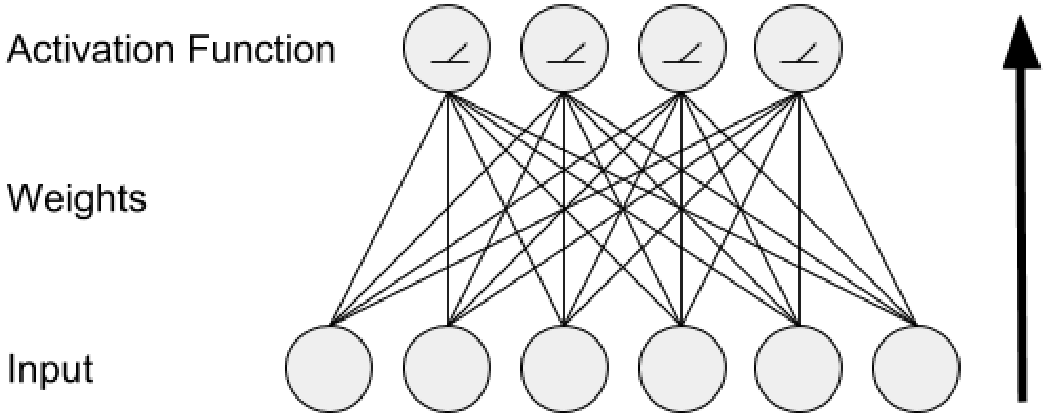


Figure 3: Illustration of a fully connected layer. The output activations are calculated by computing the matrix-vector product between a weight matrix and an input vector (which can either be of the input or the output of other layers in the DCNN). The output is then fed through a nonlinear activation function, such as the ReLU activation function.

Formally, given an m dimensional input vector $\boldsymbol{x} \in \mathbb{R}^m$, a trainable weight matrix $\boldsymbol{W} \in \mathbb{R}^{m \times f}$, and a trainable bias vector $\boldsymbol{b} \in \mathbb{R}^f$, a fully connected layer with f units computes $\boldsymbol{a} = \sigma(\boldsymbol{x}\boldsymbol{W} + \boldsymbol{b})$ with $\boldsymbol{a} \in \mathbb{R}^f$ as the output activations of the fully connected layer. Here, $\sigma(\cdot)$ denotes a nonlinear activation function, which allows nonlinear mappings from the input to the output. Typically, the activation takes the form of $\sigma(a) = \max(0, a)$, i.e., a Rectified Linear Unit (ReLU) activation function [17], but can also take the form of a sigmoid or hyperbolic tangent function.

2.2.2 Convolutional Layer

Similar to a fully connected layer, a convolutional layer computes a linear transform of a given input, followed by some nonlinear activation function. Specifically, given an input image $\boldsymbol{x}$ and a collection of trainable filters $\boldsymbol{W}$ (i.e., a filter is a set of weights), a convolution is defined as the dot product of an input patch and a filter, calculated for all filters over every input patch given a stride (i.e., the distance between two neighboring locations of input patches). Figure 4 illustrates this concept for one location. The output of the convolutional layer is downsampled by a factor of the stride. A convolutional layer typically takes the output activations of the previous layer in the hierarchy (the input to a convolution is not restricted to be an image).

In contrast with fully connected layers, a convolutional layer assumes translational invariance. Specifically, a single filter is used repeatedly across the image, and hence is invariant to the absolute position within the input. This in particular has the advantage of using fewer weights overall in the layer, which reduces the memory requirements of the model, as well as making the layer easier to train.

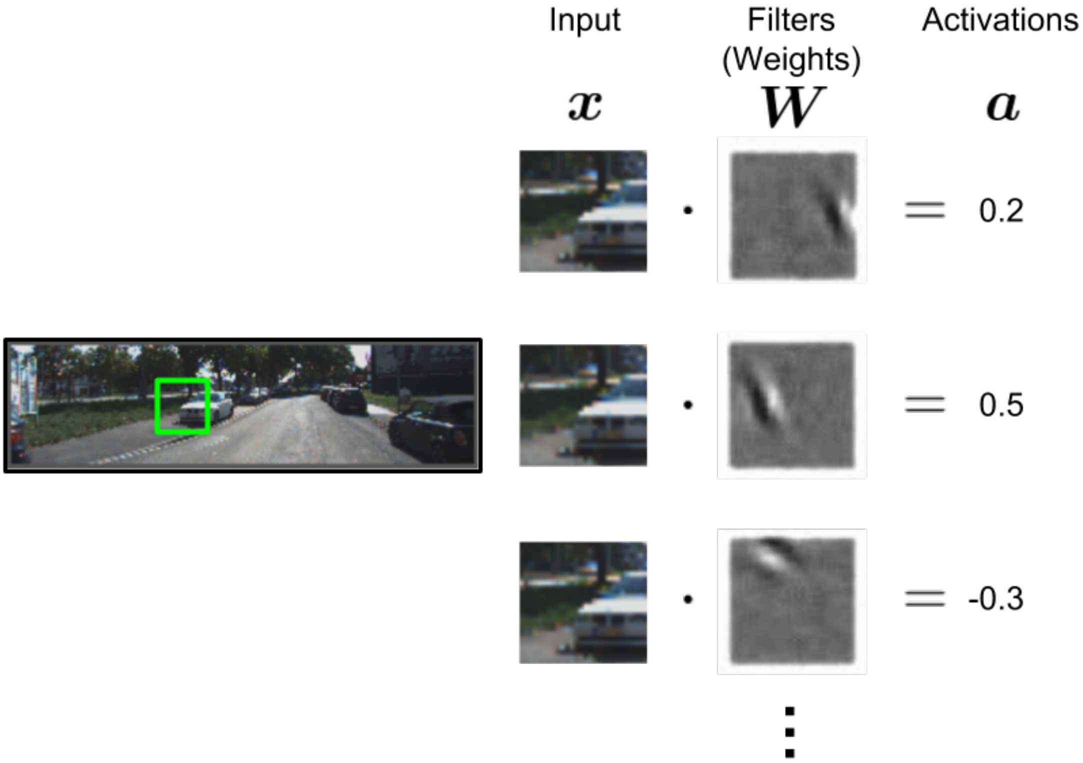


Figure 4: Illustration of a convolutional layer. The output activations for one location (in green) are calculated by taking the dot product between each of a set of filters and an image patch, followed by a nonlinear activation function, such as the ReLU activation function. This process is repeated for other image patches at different locations with the same set of filters (not shown here). Here, each filter is a set of weights, shown as a gray-scale image to indicate weight values. Activations values here are for illustration purposes only.

2.2.3 Pooling Layer

In a DCNN, a pooling layer is typically used to reduce the dimensions of the previous layer. Similar to a convolution, pooling is done on input patches (typically, on the output of convolutional layers). In contrast with a convolutional layer, a pooling layer's operation is to produce the maximum (for max pooling layers) or the average (for average pooling layers) of the input to the layer, over the spatial dimensions of the input. Additionally, pooling layers typically have no weights to optimize during training.

2.3 Gradient Descent

The objective of training a DCNN is to update the model's weights to minimize a loss function. These loss functions typically express the overall objective of the network. For example, minimizing the cross-entropy distance (see Section 2.4) between the provided ground truth class expressed as a one-hot vector (i.e., a vector where only one element has value 1 and the rest 0) and the output of a DCNN over an entire dataset optimizes the weights of a model for image labeling.

Formally, given a loss function $J(\boldsymbol{\theta}, \boldsymbol{X}, \boldsymbol{Y})$ that takes a set of trainable parameters (e.g., weights of a DCNN) $\boldsymbol{\theta}$, input data $\boldsymbol{X}$, and the target output (i.e., ground truth) $\boldsymbol{Y}$, the objective is to find

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}, \boldsymbol{X}, \boldsymbol{Y}) \ . \tag{1}$$

A common way to solve this optimization problem is via gradient descent (or backpropogation). Specifically, the procedure iteratively updates the parameters $\boldsymbol{\theta}$ relative to the partial derivative of the loss function with respect to the model weights, i.e.,

$$\mathit{\Delta}\boldsymbol{\theta} = -\eta \nabla_{\theta} J = -\eta \frac{\delta J(\boldsymbol{\theta}, \boldsymbol{X}, \boldsymbol{Y})}{\delta \boldsymbol{\theta}} \tag{2}$$

where η is a user-set parameter that controls the learning rate of the model. Intuitively, the algorithm iteratively takes a descending step in the direction of the steepest gradient repeatedly until convergence.

As written, Equation 1 optimizes over the entire training set $\boldsymbol{X}, \boldsymbol{Y}$. In practice, it is common to use mini-batch stochastic gradient descent:

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x},\boldsymbol{y} \in \boldsymbol{X},\boldsymbol{Y}} \left[J(\boldsymbol{\theta}, \boldsymbol{x}, \boldsymbol{y}) \right] \ . \tag{3}$$

Here, the algorithm randomly samples a collection of input data and target outputs $\boldsymbol{x}, \boldsymbol{y}$ from the dataset $\boldsymbol{X}, \boldsymbol{Y}$ (the number of data points sampled is defined by a user-set parameter, or the mini-batch size), and optimizes the expected value of the loss function with the sampled input.

Gradient descent has been shown to be inefficient at solving Equation 1 and Equation 3 [60]. One solution is to use momentum. In its simplest form, momentum updates Equation 2 to

$$\mathit{\Delta}\boldsymbol{\theta} = -\eta \nabla_{\theta} J + \beta \mathit{\Delta}\boldsymbol{\theta} \tag{4}$$

with β as a user-set parameter that controls the momentum term. Intuitively, the model adds a fraction of the previous update to the current update, which speeds up learning.

A common implementation of momentum in gradient descent is the Adam optimizer [30]. Here, Adam uses an adaptive learning rate for individual parameters within $\boldsymbol{\theta}$ based off of estimates of the first and second moments of the gradient of the loss function J. Formally, the update rule is defined below.

$$\boldsymbol{m}_t = (1 - \beta_1)\nabla_\theta J + \beta_1 \boldsymbol{m}_{t-1} \tag{5}$$

$$\boldsymbol{v}_t = (1 - \beta_2)(\nabla_\theta J)^2 + \beta_2 \boldsymbol{v}_{t-1} \tag{6}$$

$$\hat{\eta} = \eta \frac{\sqrt{1 - \beta_2^t}}{1 - \beta_1^t} \tag{7}$$

$$\Delta\boldsymbol{\theta}_t = -\hat{\eta} \frac{\boldsymbol{m}_t}{\sqrt{\boldsymbol{v}_t} + \varepsilon} \tag{8}$$

Here, η is a user defined learning rate, $\beta_1 = 0.9$ and $\beta_2 = 0.999$ are the momentum terms for the first and second moments respectively, $\varepsilon = 1 \times 10^{-8}$ as a small constant for numeric stability[1], t is the iteration time-step, and β_1^t and β_2^t denoting β_1 and β_2 to the power of t. The momentum variables $\boldsymbol{m}$ and $\boldsymbol{v}$ are initialized to 0 at the start of optimization.

Here, Equation 5 and 6 iteratively updates an estimate of the first and second moments of the gradient respectively. Equation 7 adjusts the learning rate to account for the biases in the moment estimations. See the work by Kingman et a. [30] for additional information on the Adam optimizer.

[1] These values of β_1, β_2, and ε are used for the Adam optimizer throughout this dissertation.

2.4 Cross-Entropy Supervised Loss Function

A common supervised loss function for multi-class classification is the cross-entropy loss function. In particular, the cross-entropy function defines a distance metric between an estimated probability distribution and a true probability distribution.

In terms of multi-class classification, the true probability distribution is the ground truth one-hot vector $\boldsymbol{y}$, which denotes probability of 1 for the true class and 0 otherwise. The estimated probability $\hat{\boldsymbol{y}}$ is the output of the estimator, which encompasses rescaling the output of a DCNN $\boldsymbol{a}$ to be a probability distribution via the softmax function $\sigma(\cdot)$:

$$\hat{\boldsymbol{y}}_i = \sigma(\boldsymbol{a})_i = \frac{e^{\boldsymbol{a}_i}}{\sum_j e^{\boldsymbol{a}_j}} \ . \tag{9}$$

The cross-entropy loss function is defined as

$$J(\boldsymbol{\theta}, \boldsymbol{x}, \boldsymbol{y}) = -\sum_i \boldsymbol{y}_i \log(f(\boldsymbol{\theta}, \boldsymbol{x})_i) = -\sum_i \boldsymbol{y}_i \log(\hat{\boldsymbol{y}}_i) \ , \tag{10}$$

which measures the distance between the output of the model $\hat{\boldsymbol{y}} = f(\cdot)$ interpreted as the estimated probability distribution over possible classes and the true distribution $\boldsymbol{y}$.

3 Background for Sparse Coding

Sparse coding [50] is an unsupervised learning algorithm based on Barlow and colleagues' theory of neural computation [3]. Specifically, sparse coding aims to encode an input as a set of sparse hidden unit activations such that the original signal is recoverable with minimal degradation. Such an encoding applied to the domain of images has achieved state-of-the-art results in image denoising (e.g., [10,43,44]) and classification (e.g., [42,45,55]).

Sparse coding shares the same goal as an autoencoder, in that both networks are unsupervised and calculate an encoding (i.e., activations over a set of hidden units) to represent a given input. Autoencoders however typically calculate activations for an input in a single forward pass (i.e., a feed-forward autoencoder), whereas sparse coding solves an optimization problem to find the sparse set of hidden unit activations that encode the input. In contrast with bottleneck autoencoders, sparse coding uses a sparsity constraint on the hidden units (i.e., most of the hidden unit activations should be zero). Both autoencoders and sparse coding learn a set of weights (called a dictionary for sparse coding) over a training set that optimizes the model for reconstruction.

Sparse coding is split into two parts: (1) learning a **dictionary** (i.e., the set of weights) from a training set, such that the dictionary is optimized for sparse representation, and (2) **encoding** a given input into its sparse representation (i.e., calculating the hidden unit activations) in terms of a dictionary. While encoding may be done with any dictionary, the aim is to learn a dictionary that enables better sparse representations. Here, encoding aims to represent an input using the sum of a sparse set of dictionary elements, where each dictionary element is a subset of weights called an atom.

Figure 5 shows a dictionary trained on natural scenes. Each square is a dictionary atom (or a set of weights), with white being the highest value and black being the lowest value. Note that weights in most patches enhances specific oriented lines, similar to the receptive fields of biological neurons found in the early stages of the visual cortex pipeline [50,77].

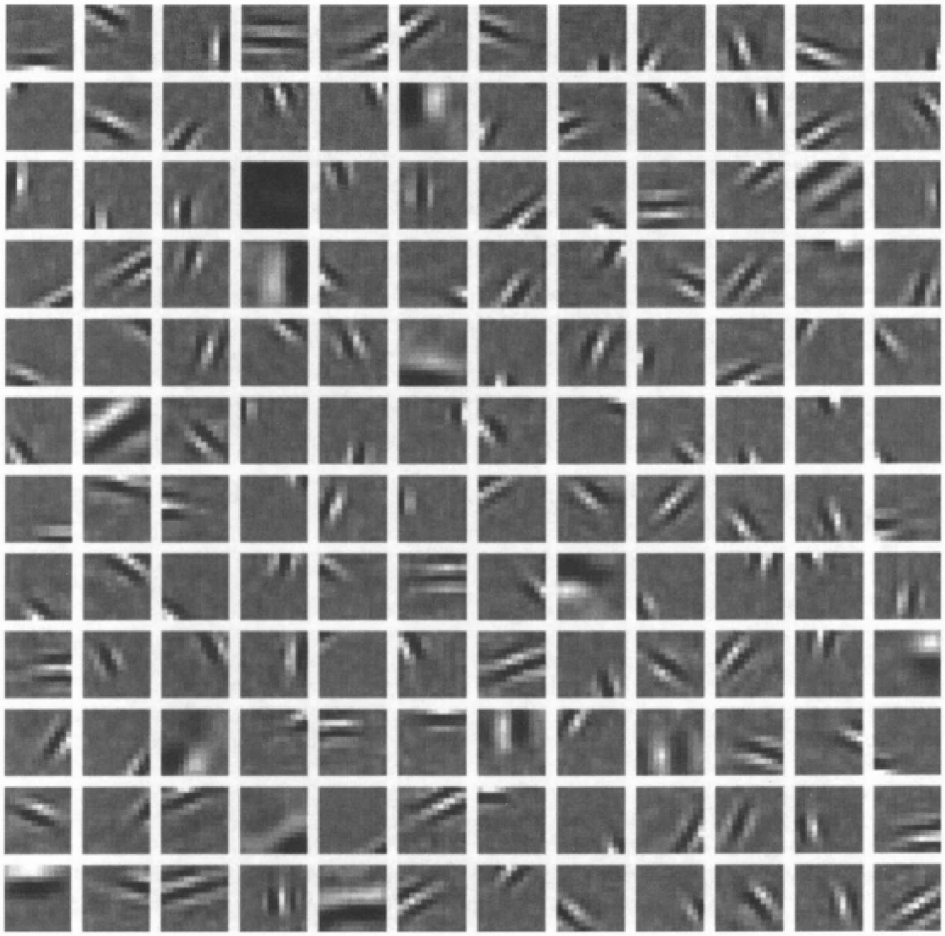


Figure 5: Example of a dictionary trained on natural images. Dictionary atoms are sensitive to oriented edges at certain frequencies. Figure from [50].

3.1 Encoding

In sparse coding, the mathematical formulation of encoding an input is that of a constrained least squares problem. The well-known least squares problem aims to decompose an m dimensional signal $\boldsymbol{x} \in \mathbb{R}^m$ (e.g., a vectorized image) such that $\boldsymbol{x} \approx \boldsymbol{D}\boldsymbol{a}$. Here, $\boldsymbol{D} \in \mathbb{R}^{m \times p}$ corresponds to a dictionary composed of p atoms, weighted by hidden unit activations $\boldsymbol{a} \in \mathbb{R}^p$. Sparse coding aims to solve an under-determined least squares problem (i.e., the dictionary D is over-complete, or $p > m$) under the constraint that the activations should be sparse (i.e., $\boldsymbol{a}$ should have few non-zero elements).

Formally, sparse coding aims to solve the following optimization problem:

$$S(\boldsymbol{x}, \boldsymbol{D}) = \hat{\boldsymbol{a}} = \arg\min_{\boldsymbol{a}} \frac{1}{2} \overbrace{\|\boldsymbol{x} - \boldsymbol{D}\boldsymbol{a}\|_2^2}^{\text{Reconstruction}} + \lambda \overbrace{\|\boldsymbol{a}\|_1}^{\text{Sparsity}} . \tag{11}$$

In other words, the problem of **encoding** consists of finding a sparse set of dictionary atoms (the columns of $\boldsymbol{D}$) multiplied by activations $\boldsymbol{a}$ that best represent the data (i.e., $\boldsymbol{x} \approx \sum_{i=0}^{p} \boldsymbol{d}_i a_i$ for $\boldsymbol{d}_i$ being the ith column of $\boldsymbol{D}$), as defined by the Euclidean distance (i.e., $\| \cdot \|_2$, or the ℓ_2 norm). Here, $\boldsymbol{a}$ is constrained to be sparse via $\| \cdot \|_1$ or the ℓ_1 norm[2], with λ as a user-set parameter that controls the trade-off between reconstruction error and sparsity. The activations $\hat{\boldsymbol{a}}$ are taken to be the encoding of the signal $\boldsymbol{x}$.

In this dissertation, I use the Locally Competitive Algorithm (LCA) [59], an iterative, hardware friendly, and biologically inspired optimization algorithm to

[2] The ℓ_1 norm is used as a surrogate to the ℓ_0 norm (i.e., the number of nonzero elements), as Equation 11 is nonconvex with respect to $\boldsymbol{a}$ if the ℓ_1 norm is replaced with an ℓ_0 norm.

solve for $\hat{\boldsymbol{a}}$ in Equation 11. The LCA algorithm defines an activation potential variable $\boldsymbol{u}$ that takes the form of a leaky integrator, with the same dimensions as $\boldsymbol{a}$. Specifically, the update rule is defined as

$$\mathit{\Delta}\boldsymbol{u} = \tau \left[\overbrace{\boldsymbol{x}\boldsymbol{D}}^{\text{Input drive}} - \overbrace{\boldsymbol{u}}^{\text{Leak}} - \overbrace{(\boldsymbol{D}^T\boldsymbol{D} - \boldsymbol{I})\boldsymbol{a}}^{\text{Competition}} \right] \tag{12}$$

$$\boldsymbol{a} = T(\boldsymbol{u}, \lambda) = \max(0, \boldsymbol{u} - \lambda) \tag{13}$$

where $T(\cdot)$ is the soft threshold operator, with λ set as the sparsity parameter in Equation 11. The activation potential vector $\boldsymbol{u}$ is driven by the similarity of the signal with each dictionary atom, and is inhibited by other activations $\boldsymbol{a}$ proportional to the similarity of the dictionary atoms. In other words, each atom competes for representation of the input with other atoms. τ denotes a user-set parameter that controls the learning rate, and $\boldsymbol{I}$ denotes the identity matrix to remove self competition. Rozell et al. [59] show that these dynamics solve Equation 11 to find $\hat{\boldsymbol{a}}$.

3.2 Dictionary Learning

Learning the dictionary $\boldsymbol{D}$ within sparse coding is analogous to learning filters in a convolutional layer and is done via gradient descent. However, the resulting dictionary is trained in an unsupervised manner; the training objective (i.e., reconstruction of the input) only requires a set of unlabeled examples, as opposed to a classification task which requires explicit labels.

Given a dataset of n training inputs $\boldsymbol{X} \in \mathbb{R}^{n \times m}$, dictionary learning aims to optimize a dictionary that allows for the best sparse representation of the dataset. Specifically, dictionary learning is defined as

$$\hat{\boldsymbol{D}} = \arg\min_{\boldsymbol{D}} \mathbb{E}_{\boldsymbol{x} \in \boldsymbol{X}}[S(\boldsymbol{x}, \boldsymbol{D})], \tag{14}$$

where $\boldsymbol{x}$ is sampled from $\boldsymbol{X}$ via mini-batches. When a dictionary is trained on images, the resulting dictionary atoms typically correspond to oriented edges at certain frequencies. Figure 5 shows an example of a dictionary trained on natural images.

There exists a degenerate solution to Equation 14 as it's written, in that the sparsity term in Equation 11 drives the magnitude of activations to be small, which in turn drives the magnitude of the dictionary atoms to be large. To avoid this, each dictionary atom is constrained to have unit ℓ_2 norm.

3.3 Convolutional Sparse Coding

Sparse coding defined in Equation 11 is analogous to a fully connected layer in a DCNN (Section 2.2.1), in that each dictionary atom spans the size of the input. In particular, previous works use image patches as input for fully connected sparse coding, e.g., the learned dictionary in Figure 5 [50]. One extension of sparse coding is using the convolution operation for reconstruction. Here, each dictionary element is used for all image patches within an image given a stride, analogous to a convolutional layer in a DCNN (Section 2.2.2).

Convolutional sparse coding is formally defined by changing Equation 11 to

$$S_c(\boldsymbol{x}, \boldsymbol{D}) = \hat{\boldsymbol{a}} = \arg\min_{\boldsymbol{a}} \frac{1}{2} \overbrace{\|\boldsymbol{x} - \boldsymbol{a} \circledast \boldsymbol{D}\|_2^2}^{\text{Reconstruction}} + \lambda \overbrace{\|\boldsymbol{a}\|_1}^{\text{Sparsity}} \ . \tag{15}$$

Here, $\boldsymbol{x}$ and $\boldsymbol{D}$ are multidimensional tensors corresponding to images and dictionaries respectively, and the $\circledast$ operation is the transposed convolution operation [75]. A transposed convolution is similar to a convolution, with the difference that the output is upsampled based on the stride as opposed to downsampled. Similar to a convolutional layer, convolutional sparse coding assumes translational invariance and replicates a dictionary across the entire image. See Figure 6 for an illustration of convolutional sparse coding.

In terms of encoding, each dictionary atom competes with other atoms for encoding (as seen in Equation 12) in a fully connected sparse coding model (i.e., Equation 11). In convolutional sparse coding, each atom additionally competes against other atoms (including itself) spatially translated based on the stride. This in particular results in a unique sparse coding model that is able to find an encoding for an entire image as a whole. See the work by Schultz et al. [62] for additional information on convolutional sparse coding and LCA.

3.4 Implementation

I use Tensorflow [1] for all models and experiments in this dissertation. In particular, I developed a novel implementation of the LCA algorithm (Equation 12) for this dissertation, such that the algorithm is GPU accelerated and fully differentiable for adversarial attacks done in Section 5. The code used for Section 4

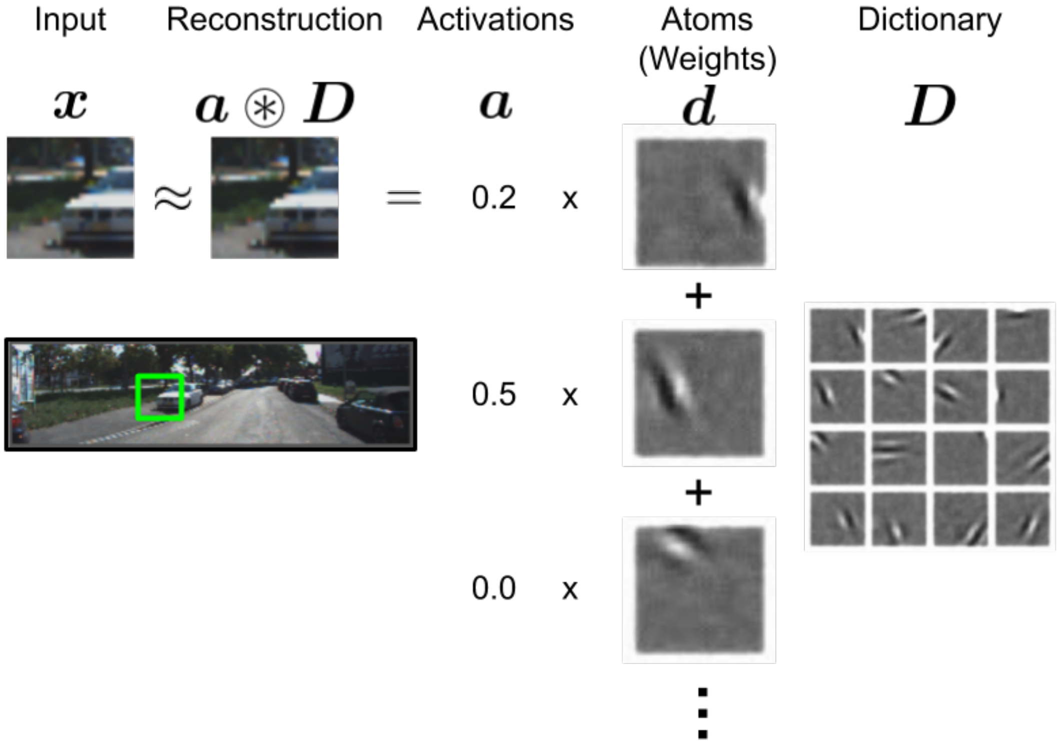


Figure 6: Illustration of convolutional sparse coding. Sparse coding aims to reconstruct the input $\boldsymbol{x}$ through a linear combination of dictionary atoms $\boldsymbol{d}$ drawn from a dictionary $\boldsymbol{D}$. Activations $\boldsymbol{a}$ are constrained to be sparse (i.e., to contain few nonzero activations). The reconstruction is calculated via a transposed convolution operation [75]. The input shown here is a patch (in green) from the whole image. Activations values here are for illustration purposes only.

and Section 5 is publicly available at [51]. The code used for Section 6 is publicly available at [38].

4 Exploring Convolutional Sparse Coding for Supervised Image Classification

Sparse coding is an unsupervised learning algorithm that aims to reconstruct data in terms of a linear combination of a sparse set of dictionary atoms. Despite the algorithm being optimized for image reconstruction, previous works have shown success in using sparse coding for discriminative tasks [4,42,45,72]. Most of these works combine both supervised and unsupervised learning (i.e., semi-supervised learning) for the ultimate task of classification. In this section, I focus on exploring the cause of the success of sparse coding in discriminative tasks.

Most learning algorithms can be split into two stages: a **feature learning** (or dictionary learning for sparse coding) stage that trains the parameters (e.g., weights) of the model, and an **encoding** stage that encodes a given input into a set of activations using the aforementioned features. In supervised DCNN layers, feature learning consists of training weights to optimize them for the supervised task, and encoding consists of computing the activations by convolving the input with the weights. Sparse coding follows this through dictionary learning and encoding. Here, dictionary learning optimizes weights for reconstruction and sparsity, and encoding consists of finding the sparse set of activations that represents the input. In this part of the dissertation, I explore the following question: to what extent does the success of sparse coding in discriminative tasks come from the learned **dictionary** (i.e., features) versus the **encoding** procedure (i.e., the sparse activations)?

Coates et al. [6] previously explored the question of why sparse coding is successful in classification. In particular, the authors claimed that the performance of sparse coding in supervised discriminative tasks comes from the encoding procedure. Specifically, they compare different dictionaries used by sparse coding and its effect on classification performance. One dictionary they use is optimized for a 1-sparse model (i.e., an encoding that only uses one active element per input), which is computationally cheaper than finding a dictionary optimized for sparse coding. They show that the difference of classification performance on the CIFAR-10 dataset [32] between the suboptimal 1-sparse dictionary versus the dictionary optimized for sparse coding is minimal, as long as the full encoding procedure of sparse coding is used.

The authors used a fully connected sparse coding model on patches in isolation, followed by stitching together the resulting sparse codes. Convolutional sparse coding defined in Section 3.3 differs from the patch-based sparse coding in that convolutional sparse coding solves for the encoding of the entire image. In patch-based sparse coding, all activations only compete with each other for representation on each patch in isolation. In convolutional sparse coding, activations compete spatially as well, as any activations with overlapping receptive fields contribute to the same pixels. This section aims to expand on the results shown by Coates et al., particularly with the use of convolutional sparse coding on the whole image instead of using fully connected sparse coding on patches as is done by Coates et al.

I compare the classification performance of sparse coding using different combinations of dictionary (i.e., feature) learning models with encoding to test the effect of dictionary versus encoding methods. I show that the **encoding** process contributes more to classification than the **dictionary** (i.e., feature) learning process. In particular, I find that the spatial competition utilized by the encoding procedure is crucial for performance in image classification with sparse coding. Finally, I show that although patch-based sparse coding achieves similar results to a feed-forward baseline, convolutional sparse coding achieves the best results, matching the performance of a fully supervised model optimized for image classification.

4.1 Related Work

The basis of this work stems from the work done by Coates et al. [6]. In particular, the authors show that the key to performance of sparse coding for classification comes from the nonlinear encoding scheme as opposed to the dictionary learned by sparse coding. Additionally, the authors claim that a simple nonlinear feed-forward soft threshold function achieves competitive results to sparse coding, even when the soft threshold encoding model uses random patches as a dictionary. In this section, I test if these results hold with the use of convolutional sparse coding.

One issue that arises is the computational complexity of the models trained in Coates et al. Specifically, one requirement of the models I test is the necessity of being able to adversarially attack these models in Section 5, especially through the sparse encoding optimization as described in Section 3.1. To this end, I reduced the number of dictionary atoms from 1600 to 512. To offset the loss of output activations per image, I maintain more spatial information by pooling over less spatial area. Overall, Coates et al. uses 12800 activations per image, versus 8192 activations used in the experiments I describe in Section 4.2.

Another issue related to adversarially attacking these models is that the model must be differentiable from end to end for the attacks defined in Section 5.1. Specifically, Coates et al. use an L2-SVM as their classifier of the output activations from their encoder, whereas I use a supervised fully connected layer trained via gradient descent for classification.

Finally, Coates et al. use whitening (i.e., removing linear dependencies within the image) as a preprocessing step. I find that whitening is unnecessary, as the model is able to learn a similar dictionary to dictionary learning algorithms that train on whitened images. In addition, removing the preprocessing step simplifies the model overall.

Another work by Coates et al. [7] compares various unsupervised first layers with a supervised classifier on an image classification task. In particular, the authors test the effect of receptive fields, number of dictionary atoms, and the stride of the unsupervised encoding on classification. Additionally, work done by Zhang et al. [76] tests the effect of patch sizes in a convolutional sparse coding model on classification results while holding the degree of over-completeness fixed. In this work, I aim to test the relative effects of dictionary learning versus encoding using sparse coding for classification.

Rigamonti et al. [57] test the effect of sparsity in sparse coding for dictionary learning and encoding on classification. The authors find that having a convolutional sparse encoding does not improve performance on image classification over a simple feed-forward encoder, but is key for learning useful features for classification. Specifically, the authors suggest that the performance from using a dictionary trained from convolutional sparse coding with a feed-forward encoder (i.e., an encoder that uses a single convolution to calculate its activities) matches that of using the encoding of convolutional sparse coding. In this section, I find contradicting results, in that a feed-forward encoder using unsupervised features performs much worse on image classification than an encoder that uses convolutional sparse coding.

4.2 Experiments

I aim to distinguish the effect of dictionary learning versus encoding on image classification tasks. I build a two layer model for classifying thumbnail images into one of 10 categories. I vary the first layer by choosing one of 3 encoding models (Section 4.2.3), while also choosing the dictionary the encoder uses from 4 different unsupervised dictionary learning models (Section 4.2.2). I also compare the unsupervised methods to a supervised feed-forward encoder with features optimized for image classification. All dictionary learning and encoding models for the first layer use 512 elements with a patch size of 8×8, with a stride of 2 in both spatial directions.

The second layer of the model is a fully connected layer trained for supervised image classification (Section 4.2.4). All models train with a mini-batch size of 8.

4.2.1 Dataset

I use the CIFAR-10 [32] dataset for training and evaluating. In particular, the CIFAR-10 dataset is a collection of 32 × 32 pixel color thumbnail images, with each image annotated by a human to belong to one of ten classes. Figure 7 shows example images from the dataset. This dataset in particular is useful for exploring models such as sparse coding due to the small image sizes, which allows for computational savings.

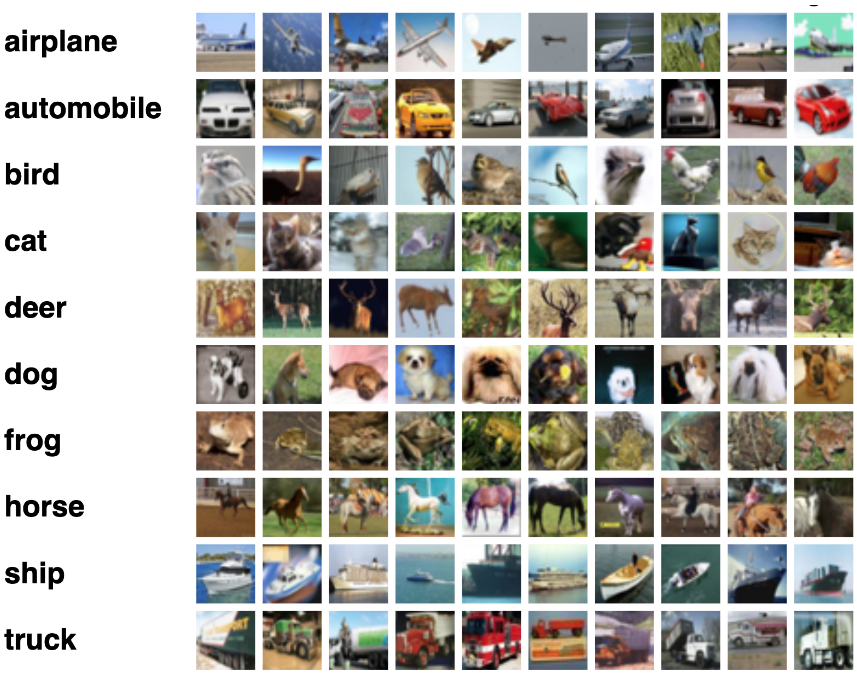


Figure 7: Examples of images from the CIFAR-10 dataset. Each image is human-annotated to be one of ten classes. Figure from [31].

CIFAR-10 contains a total of 60000 images, which is split into a training set of 50000 images and a test set of 10000 images. From the training set, I hold out an additional 10000 images as a validation set for parameter tuning, which leaves 40000 images and labels to train the models on.

As a preprocessing step, each image is normalized to have zero mean and unit standard deviation. During training, I augment the training dataset of by randomly cropping CIFAR-10 images from 32 × 32 to 28 × 28 pixels, followed by randomly horizontal flipping of the image. These types of augmentations during training allows us to artificially expand the dataset to contain more image samples without changing the semantic content of the image. During evaluation, each test image is center cropped to 28 × 28 pixels.

4.2.2 Dictionary Learning Methods

I use two different types of sparse coding algorithms to learn dictionary features: a patch-based sparse coding method as done by Coates et al. [6] and a convolutional sparse coding method. Here, I aim to explore the addition of spatial competition provided by convolutional sparse coding on the effects of dictionary learning for classification. As baselines, I additionally test random features and features extracted from random image patches in the dataset (denoted as an imprinted dictionary). These learning algorithms are described below.

Patch Sparse Coding: The first sparse coding model I use is a patch-based sparse coding model (**Patch SC**). Here, 8 × 8 pixel image patches are extracted from the dataset from valid image locations. These patches are vectorized (i.e., reshaped from the image dimensions into a one-dimensional vector), then encoded using the fully connected sparse coding model defined in Equation 11. Using these encodings, a dictionary is trained using Equation 14.

The sparse coding model has a hyper-parameter λ that controls the trade-off between sparsity and reconstruction error. I train the model with several values of λ to see the effect of sparsity for dictionary learning and classification. Specifically, I use $\lambda = \{0.5, 1.0, 1.5, 2.0, 2.5\}$.

To solve for Equation 11, I use LCA (Equation 12) with an encoding learning rate $\tau = 0.033$ and iterate for 50 steps.

The **Patch SC** dictionary learning method required pre-training with a lower value of λ (i.e., a less sparse encoding) before training the dictionary at the desired value. I found that this was required to train models with higher values of λ, as starting from random dictionaries resulted in elements that were never activated. Specifically, I train all patch-based sparse coding models using stochastic gradient descent (Equation 3) for 1×10^5 steps first with $\lambda = 0.5$ with a dictionary learning rate of $\eta = 5 \times 10^{-3}$, followed by training the dictionary for an additional 9×10^5 steps with the desired λ value with a dictionary learning rate of $\eta = 2 \times 10^{-3}$.

I train the **Patch SC** model with a batch size of 8. Here, the algorithm trains on all patches extracted from these 8 images for a single time-step.

Convolutional Sparse Coding: The second sparse coding model I use is convolutional sparse coding (**Conv SC**), as defined in Section 3.3. In particular, convolutional sparse coding solves for the whole image, which results in competition between elements shifted based on some stride. When learning a dictionary with convolutional sparse coding, the dictionary is more over-complete than that of a patch-based sparse coding method with similar patch sizes and strides [62].

Similar to patch-based sparse coding, I use several values of λ to compare effects of sparsity and reconstruction error on dictionary learning and classification. Specifically, I use $\lambda = \{0.1, 0.2, 0.3, 0.4, 0.5\}$.

To encode images using convolutional SC for dictionary learning, I used LCA (Equation 12) for convolutional sparse coding, using $\tau = 0.005$ for 75 steps. For dictionary learning, I train the dictionary using stochastic gradient descent for 1×10^6 steps using a dictionary learning rate of $\eta = 1 \times 10^{-3}$.

Random Features: As a control, I generate a set of random features as a dictionary. In particular, these random features help determine the impact that dictionary learning has on encoding and classification.

The random features are generated from a normal distribution with zero mean and a standard deviation of 0.5, and truncated such that any values more than two standard deviations from the mean are resampled. These features are then normalized such that each feature has an ℓ_2 norm of one.

Imprinted Features: As an intermediate step between random and learned features, I use an set of imprinted features as a control. Here, the idea is to generate a set of features that contains more structure than random features. Imprinted features, along with random features, help determine the impact of learning a dictionary from sparse coding.

To generate the dictionary, features are set to be patches randomly sampled from the dataset. Each patch selected from the dataset are preprocessed to have zero mean and unit standard deviation, followed by normalization such that each patch has an ℓ_2 norm of one.

4.2.3 Encoding Methods

In the previous section, I describe methods for dictionary learning. In this section, I describe the following methods for encoding (i.e., representing the input as a set of activations: a patch-based sparse coding model as done by Coates et al. [6] and a convolutional sparse coding model to see the effect of spatial competition on encoding. I additionally test a simple feed-forward soft threshold model as a control for the contribution of sparse encoding for classification.

Patch Sparse Coding: The patch-based sparse coding method (**Patch SC**) uses the encoding defined in Equation 11. Here, patches are extracted from a zero padded input image such that the output size is a factor of the input size defined by the stride. After encoding each patch in isolation, the sparse codes are then stitched back together as the output.

I additionally test the effect of the trade-off parameter λ on encoding and classification. Here, I mirror the values used for dictionary learning, i.e., $\lambda = \{0.5, 1.0, 1.5, 2.0, 2.5\}$. I also use the encoding parameters described in Section 4.2.2 for Patch Sparse Coding.

Convolutional Sparse Coding: The convolutional sparse coding method (**Conv SC**) solves for the entire image as a whole. Here, the encoding is unique in that a single activation is nonlinearly dependent on all other activations within its receptive field.

Similar to dictionary learning, I use several values of λ for testing its effect on classification. I mirror the values used for dictionary learning, i.e., $\lambda = \{0.1, 0.2, 0.3, 0.4, 0.5\}$. I also use the encoding parameters described in Section 4.2.2 for Convolutional Sparse Coding.

Soft Threshold: For the soft threshold encoding, the layer simply encodes the image by computing the convolution of the input with the dictionary. In particular, this is a control for testing the effectiveness of the computationally expensive encoding methods used by sparse coding.

The resulting activations are then fed through a soft threshold $T(u, \alpha) = \max(0, u - \alpha)$, where u is a single activation and α is a parameter controlling the threshold. Given a large enough α, the resulting set of activations is sparse.

I use several values of α for testing its effects on classification: $\alpha = \{0.5, 1.0, 2.0, 3.0, 4.0\}$.

4.2.4 Classifier

After encoding the input image using one of the encoding methods defined in Section 4.2.3, which uses one of the dictionaries defined in Section 4.2.2, the output is then fed into a classifier trained via supervised learning for image classification. First, the output encoding is max pooled with a patch size of 5×5 with a stride of 4 in both spatial directions. This pooled encoding is then fed into a supervised fully connected layer with ten output activations corresponding to the ten output classes of CIFAR-10. The output is fed through a softmax function and uses the supervised cross-entropy loss function (see Section 2.4) to train the classifier.

I use the Adam optimizer (Equation 8) to train the weights of the supervised classifier, with a learning rate $\eta = 1 \times 10^{-3}$ for 3×10^5 steps. Additionally, I anneal (i.e., lower over time) the learning rate by a factor of 0.9 every 1×10^5 steps.

4.2.5 Fully Supervised Model

The models I've described in the sections above consist of a sparse-coding layer that feeds into a fully connected (classification) layer; the weights and activations in the sparse-coding layer are computed in an unsupervised way, whereas the weights in the fully connected layer are trained in a supervised way (i.e., using labeled training data).

In this section, I describe a fully supervised model of the same size as the other models. The first layer of the fully supervised model is a convolutional layer with a leaky ReLU activation function [40] defined below:

$$\sigma(a_i) = \begin{cases} a_i & a_i > 0 \\ \beta a_i & \text{otherwise} \end{cases} . \tag{16}$$

Here, the leaky ReLU is similar to the ReLU, but adds a positive gradient $\beta = 0.2$ when the activation a is below 0 (i.e., the derivative of $\sigma(\cdot)$ is β when $a \leq 0$). This allows units to not be stuck when $a \leq 0$ during gradient descent. I find better performance for the fully supervised layer using the leaky ReLU as opposed to the normal ReLU nonlinear activation function.

The weights in all layers of the fully supervised model are trained via backpropagation of the supervised loss function. That is, the weights for the first layer of this model, in contrast with the models incorporating sparse coding, are optimized for image classification.

4.3 Results

Recall that the purpose of these experiments is to explore the relative contributions of dictionary (i.e., feature) learning and encoding to classification performance. Further results on the sparse coding models can be found in Appendix A, and further classification results can be found in Appendix B.

Table 1 shows the best accuracy achieved while varying either λ or α for each dictionary learning and encoding combination. In general, I find that **Conv SC** encodings with **Conv SC** dictionaries achieves the best result with an accuracy of 73.55%.

		Encoding			
		Fully Supervised	Soft Threshold	Patch SC	Conv SC
Dictionary Learning	Fully Supervised	0.7298	-	-	-
	Random	-	0.6614	0.5852	0.6383
	Imprinted	-	0.5674	0.6284	0.6911
	Patch SC	-	0.6391	0.6639	0.7184
	Conv SC	-	0.6441	0.6717	**0.7355**

Table 1: Table of the highest accuracies with varying parameters for each combination of dictionary learning and encoding model. This table is a subset of Table 6 in the appendix.

One question that arises is the significance of the differences in accuracies between all of the models tested. For example, is the difference between the values in the right most column of Table 1 simply due to the stochastic nature of training a classifier (random initial weights, random ordering of training examples, and stochastic data augmentation)? To this end, I take a subset of the experiments tested and train the classifier 10 times with different initial conditions and data augmentations. The results are shown in Figure 8. Ideally, each dictionary would also be retrained with random initial conditions, but I held the dictionary fixed through each independent run due to the computational expense.

I found that the effect of random initial conditions with soft threshold and **Patch SC** vary approximately 3%. **Conv SC** and fully supervised networks vary approximately 1%.

I first examine the effect of using different dictionaries. Overall, most models perform similarity across different trained dictionaries. For example, Table 1 shows that the **Conv SC** encoding model achieves within 2% accuracy from using a suboptimal **Patch SC** dictionary versus using a **Conv SC** dictionary. This difference in accuracies can be explained by random initial conditions (from Figure 8). I find similar results for the **Patch SC** encoding model. Surprisingly, I find that the **Patch SC** encoding model achieves the best performance using a **Conv SC** dictionary. However, this could also explained through random initial conditions, as the difference in accuracies fall within the range shown in Figure 8. Finally, using an imprinted dictionary for both sparse coding models gets within 5% accuracy of the sparse coding dictionaries. This is surprising, since the imprinted dictionary did not require any training, compared to the computationally expensive dictionary training procedures of sparse coding.

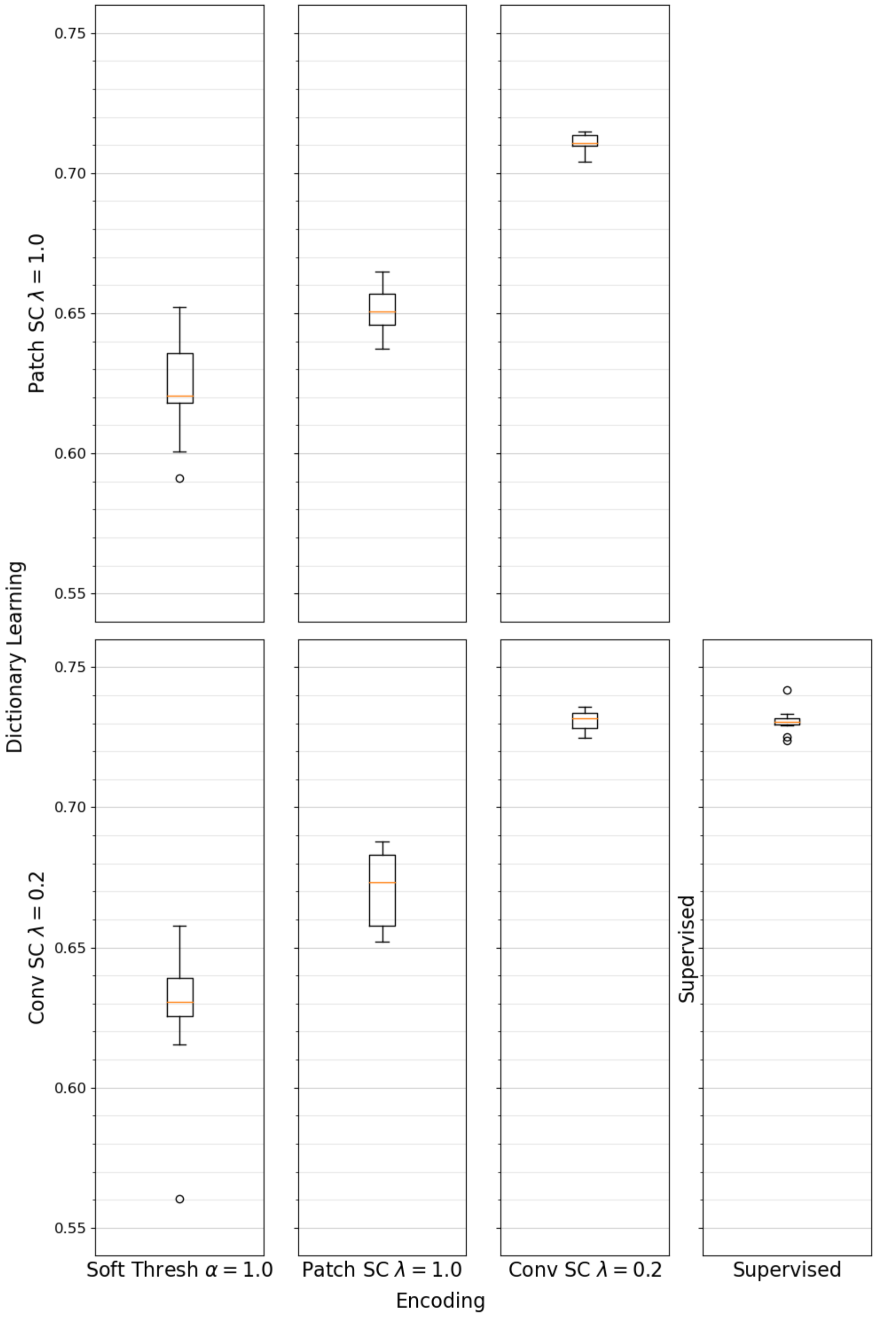


Figure 8: Range of accuracies with different initial conditions for the classifier from a subset of dictionary learning (rows) and encoder (columns) combinations. Each model was ran 10 times. The resulting plot shows accuracies across the 10 runs. Here, the box-and-whiskers plot's circles show outliers (i.e., points above or below 1.5 of the interquartile range), and the notches shows (from top to bottom) the maximum (excluding outliers), 75% quartile, median, 25% quartile, and minimum (excluding outliers) of the data.

Unlike the effects of using different dictionaries, there are substantial differences in performance when using a convolutional sparse coding method to encode images versus the other tested encoding methods. In particular, similar to the findings from Coates et al. [6], I find that using a random dictionary with a simple soft threshold achieves competitive results against **Patch SC** encoding models. However, neither model reaches the performance of convolutional sparse coding, which suggests that the more complex encoding method is crucial for good classification performance.

Overall, I find that the encoding model is more important than the dictionary used for each model, with convolutional encodings being key to achieve good classification performance. In addition, I find that a simple feed forward encoding model, while competitive against the patch-based sparse coding models, is outperformed by the convolutional sparse coding models.

4.4 Summary

In this section, I explored the relative contribution of dictionary learning versus encoding of sparse coding for supervised image classification. I find that, similar to Coates et al., the nonlinear encoding method of sparse coding contributes more to the classification performance than the dictionary learned from sparse coding. However, I find that a feed-forward soft threshold encoder does not outperform a convolutional sparse coding model as an encoder, but does perform similarity to the patch-based sparse coding model. Overall, this suggests that there may be computational savings in using a sub-optimal dictionary for classification.

Future work entails testing different types of dictionaries, such as handcrafted dictionaries or trained from different, more efficient dictionary learning models. In particular, I found that performance using a dictionary of imprinted features achieved relatively high accuracy independent of encoding models. Here, the hope is to find a middle ground in finding a good set of features between the computationally cheap imprinted features and computationally expensive sparse coding dictionaries. For example, Coates et al. [6] suggests using K-means for learning features.

Overall, I find that the model with an unsupervised first layer does not significantly outperform a fully supervised model of the same size. However, I explore various situations in which an unsupervised sparse coding model could benefit classification. Specifically, I test cases when inputs are corrupted by either adversarial or random noise in Section 5, and when the number of training labels are limited in Section 6.

5 Sparse Coding for Model Robustness

DCNNs have been shown to generalize well to a test set, i.e., images that have not been used for training the model, and are therefore unseen by the model. However, recent literature has questioned the true generalization ability of DCNNs, as such models perform poorly on test images that contain visually irrelevant differences in statistics or corruptions from the training images, some of which are imperceptible to humans. The classical example for showcasing the brittleness of DCNNs is that of adversarial examples. In particular, Szegedy et al. [67] showed that, in the domain of image classification, it is easy to fool a DCNN into classifying an image incorrectly with high confidence by adding an imperceptibly small change to the image (Figure 9 shows an example of an adversarial example). Furthermore, the authors showed that such adversarial examples are generalizable across multiple architectures.

Other studies show the brittleness of DCNNs with examples of modifications that are not explicitly constructed to fool such models. For example, DCNNs have been shown to perform poorly on blurred or distorted images when trained on clean images [9,16]. Worse yet, it has been found that statistics of images (e.g., the frequency components of a given image) tend to be a strong cue for image classification [69]. DCNNs tend to overfit to such statistics, resulting in brittle performance when a test image does not contain these statistical cues. Specifically, one particular study by Jo et al. [28] shows that the performance of DCNNs degrades drastically when applied to test images that are simply filtered to contain different frequency statistics than training images. In all these cases, humans correctly classify the modified images. Overall, these studies bring into question the ability of DCNNs to learn high level semantics instead of simply learning statistical regularities.

Unsupervised learning provides a method for learning high level abstractions that are not class specific, which may help such models to generalize better to unseen examples. In particular, recent studies showed that sparse coding is resistant to adversarial examples that affect a wide set of supervised DCNNs in a face detection task [64]. Furthermore, sparse coding has been shown to de-noise image inputs [10,27], which can help denoise small perturbations in test examples while not losing key information from the image. Finally, previous work done by my colleagues and I suggest that the population nonlinearity (i.e., an activation value is dependent on other activations within the layer) of sparse coding helps classification models be more robust to adversarial examples [52].

In this section, I aim to explore how effective sparse coding is in alleviating the poor performance of traditional DCNNs on modified test examples. Specifically, Section 5.3 explores the use of sparse coding against adversarial examples, and Section 5.4 tests the use of sparse coding to generalize against common corruptions of images.

5.1 Background on Adversarial Examples

Given a trained neural network $\hat{\boldsymbol{y}} = f(\boldsymbol{x}, \boldsymbol{\theta})$ which classifies input $\boldsymbol{x}$ using model parameters $\boldsymbol{\theta}$, adversarial examples are formed by finding

$$\arg\min_{\boldsymbol{x}^*} \|\boldsymbol{x} - \boldsymbol{x}^*\|_2 \text{ s.t. } f(\boldsymbol{x}^*, \boldsymbol{\theta}) \neq \hat{\boldsymbol{y}} \ . \tag{17}$$

Here, the image $\boldsymbol{x}^*$ an adversarial example for the network $f(\cdot)$: the goal is for $\boldsymbol{x}^*$ to be as similar as possible to $\boldsymbol{x}$ (e.g., using ℓ_2 distance as a metric of similarity), but for $f(\cdot)$ to misclassify $\boldsymbol{x}^*$ with high confidence (assuming the network correctly classified the input, i.e., $\hat{\boldsymbol{y}} = \boldsymbol{y}$).

In particular, one method for finding adversarial examples is the fast gradient sign method [18]. Here, the original loss function $J(\boldsymbol{\theta}, \boldsymbol{x}, \boldsymbol{y})$ of a classifier aims to minimize the difference of the calculated output $\hat{\boldsymbol{y}} = f(\boldsymbol{x}, \boldsymbol{\theta})$ and the ground truth $\boldsymbol{y}$. The fast gradient sign method computes an adversarial image using a single step in the direction of the gradient (i.e., the opposite direction of Equation 1 used during training) using the true class $\boldsymbol{y}$:

$$\boldsymbol{x}^* = \boldsymbol{x} + \epsilon \, \mathrm{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, \boldsymbol{y})) \tag{18}$$

where ϵ is a hyper-parameter that is typically small in magnitude. This method has been shown to consistently find adversarial examples for networks. Figure 9 shows an example of an adversarial example derived using the fast gradient sign method on a popular image classification task.

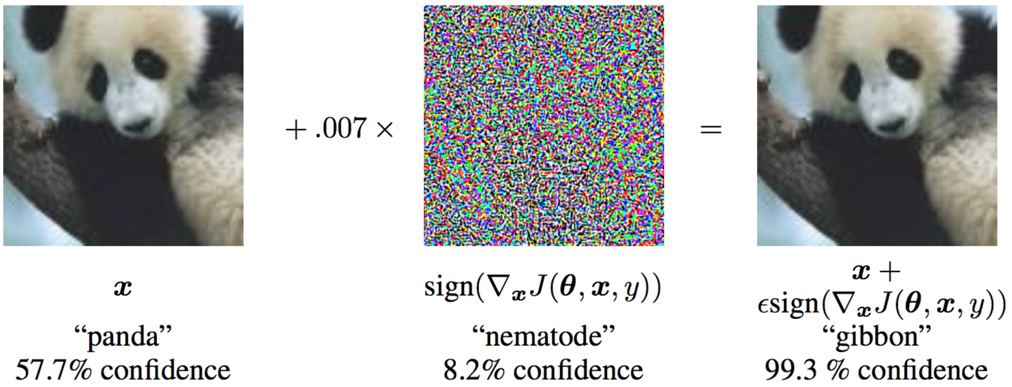

+ .007 ×
=
$\boldsymbol{x}$
"panda"
57.7% confidence
$\mathrm{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"nematode"
8.2% confidence
$\boldsymbol{x} + \epsilon\mathrm{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"gibbon"
99.3 % confidence

Figure 9: Example of an adversarial image on a trained image classification DCNN. An image classified as a "panda" is then classified as a "gibbon" with 99% confidence when structured noise is added to the image. Figure from [18].

As written, Equation 18 aims to find an adversarial image $\boldsymbol{x}^*$ to simply misclassify: this is defined as an untargeted attack. In contrast, one can define a target class $\bar{\boldsymbol{y}} \neq \boldsymbol{y}$, in which the goal is to create an adversarial image to be classified to the target class. Formally, Equation 18 can be rewritten as follows:

$$\boldsymbol{x}^* = \boldsymbol{x} - \epsilon \, \mathrm{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, \bar{\boldsymbol{y}})) \ . \tag{19}$$

Note that the sign is changed from Equation 18, as the objective of Equation 19 is to step towards the adversarial target class $\bar{\boldsymbol{y}}$.

An extension of the fast gradient sign method is to apply it multiple times with a step size γ. For example, for the targeted method, the update rule for finding the adversarial image is as follows:

$$\mathit{\Delta}\boldsymbol{x}^* = -\gamma \, \mathrm{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, \bar{\boldsymbol{y}})) \ . \tag{20}$$

I use this attack throughout this section.

5.2 Related Work

Brittleness of DCNNs: There are various papers that showcase the lack of robustness in DCNNs on classification tasks. In particular, previous papers [9,16,28] show that neural networks perform poorly when statistics of test images differ from those in the training set. Rosenfeld et al. [58] show that placing a cropped object into an image results in non-local changes on an object detection model. In addition, Szegedy et al. [67] have shown the existence of adversarial examples. Overall, these studies show that the performance of DCNNs can be brittle to small changes to test images.

Towards Robustness to Adversarial Examples: One technique to improve overall robustness is to augment the training set with adversarial examples. In particular, Goodfellow et al. [18] has suggested training on adversarial example to improve robustness to such examples. Other studies [20,48,71] use an adversary within the model to implicitly augment the dataset, or to regularize against small perturbations. Here, I propose unsupervised sparse coding as an alternative model to help improve robustness of DCNNs, and note that data augmentation techniques are readily applicable to the techniques proposed here.

Gu et al. [20] explores several other types of preprocessing techniques to help alleviate adversarial examples. In particular, the authors show that adding Gaussian noise and Gaussian blurring tend to help alleviate poor performance on adversarial examples, but show that such preprocessing techniques tend to hurt overall performance. The authors also show that using a denoising autoencoder to map from noisy to clean example as a preprocessing step tends to help performance on adversarial examples. However, the authors find that new adversarial examples can be found from the stacked network (i.e., autoencoder plus classifier), and that the stacked network overall is more sensitive than the original network to adversarial examples. I extend this work by exploring using sparse coding to test the network's robustness to adversarial examples generated for the whole classification model.

Efficient Representations and Adversarial Examples: Previous works suggest that efficient representations within models provide robustness to adversarial examples. In particular, it has been shown that compressing the model to have fewer overall free parameters [73] or reducing the dimensions of the input data as a preprocessing step [61] helps classification models to be more robust to adversarial examples. Other works have shown that constraining a model to have sparse weights or activations [19,47,22] allows for more robust models to adversarial examples. In this dissertation, I explore the use of an over-complete (i.e., dimensionality expansion) sparse representation for adversarial robustness.

Sparse Coding and Adversarial Examples: Springer et al. [64] showed that a sparse coding model trained for face detection is robust to adversarial examples that are generated to fool fully supervised DCNNs via the fast gradient sign method [18]. Additionally, Kim et al. [29] showed that sparse coding provides robustness against adversarial examples by minimally distorting the input image while achieving minimal degradation of accuracy on clean data. Finally, recent work by Sun et al. [66] showed that using a convolutional sparse coding algorithm as the first layer of a DCNN provides a robust defense to adversarial examples. While these adversarial examples have been shown to generalize to multiple architectures [67], all the above works have explored adversarial examples in a black-box setting (i.e., attacks that are model agnostic), and do not test adversarial examples generated explicitly to attack sparse coding. In contrast, I propose to find adversarial examples that attack the model as a whole, including the sparse coding layer.

Previous work done my colleagues and I showed that the population non-linearity (i.e., activations are non-linearity dependent on other activations) exhibited by sparse coding result in activations that are less sensitive to adversarial examples [52]. In particular, we present a theoretical explanation to the selectivity of sparse coding activations versus point-wise non-linearity (i.e., nonlinear activation functions that are independent of other activations in a layer, e.g., RELU or sigmoid non-linearity activation functions). We show that sparse coding activations exhibit more selectivity towards preferred image features (e.g., edges), and less selectivity towards perceptually irrelevant features, such as adversarial perturbations. Consequently, adversarial attacks against a sparse coding model within a classification model result in the image looking more like the targeted class. Overall, we show that a supervised model incorporating unsupervised sparse coding is more robust to adversarial attacks than a model incorporating only point-wise non-linearities, such as the typical supervised convolutional layer.

One limitation of the results on adversarial robustness of our previous work [52] is the dataset; we found convincing results on adversarial robustness on the MNIST dataset [36], but less so for the more complicated gray-scale CIFAR dataset. Additionally, the sparse coding model tested was a fully connected sparse coding model on the entirety of the image, as opposed to a convolutional model that exhibits better scalability to large image sizes.

In this dissertation, I aim to further explore the adversarial robustness of sparse coding. Specifically, I aim to test the robustness of convolutional sparse coding on the color CIFAR dataset, and to explore the effect of sparsity on adversarial robustness.

5.3 Robustness of Sparse Coding on Adversarial Examples

I aim to test the susceptibility of models that incorporate unsupervised sparse coding to adversarial examples. Specifically, I aim to compare the performance between a classification model that uses unsupervised convolutional sparse coding as the first layer versus a fully supervised model.

One goal of these experiments is to use a “white-box” attack on the classification model, in that the attack is done on the model with known parameters (as opposed to a “black-box attack”, which assumes the model parameters are unknown to the attacker). Consequently, the sparse coding and classifier model must be differentiable from end to end, so that the gradient with respect to the images can be calculated in Equation 20. To achieve this, I unroll the iterative encoding method in Equation 12 and propagate the adversarial loss to the input through the sparse encoder.

I adversarially attack the convolutional sparse coding model defined in Sections 4.2.2 and 4.2.3 with $\lambda = 0.2$ (for both encoding and dictionary learning), as well as the fully supervised model defined in Section 4.2.5. Both the sparse coding model and the fully supervised model have 10 runs with random initial weights, random ordering of training examples, and random data augmentations, as well as random target labels during the adversarial attack.

To compare the robustness of various models to adversarial examples, I attack the models using Equation 20. Here, I perturb the adversarial image $\boldsymbol{x}^*$ until the model reaches 90% confidence on a given target class $\bar{\boldsymbol{y}}$. Figure 10 shows several examples of adversarial images for the two models.

The metric I use to measure robustness to adversarial examples is the absolute distance (AD) between the original and adversarial image, i.e., $\sum |\boldsymbol{x} - \boldsymbol{x}^*|$, averaged across test images for which the attack was successful (mean AD, or mAD). The pixel values of the image is ranged from 0 to 255 when calculating this metric. Here, the mAD can be interpreted as the average change in pixel intensity needed to fool the model to 90% confidence. A high mAD means that the adversary has to make a larger perturbation to an image in order to fool the model to 90% confidence, which means the model is more robust to attacks – it forces an adversary to “work harder” to fool it.

Note that the attack is unbounded (i.e., there is no bound on the magnitude of the perturbation). I find that this results in a successful attack (i.e., the model

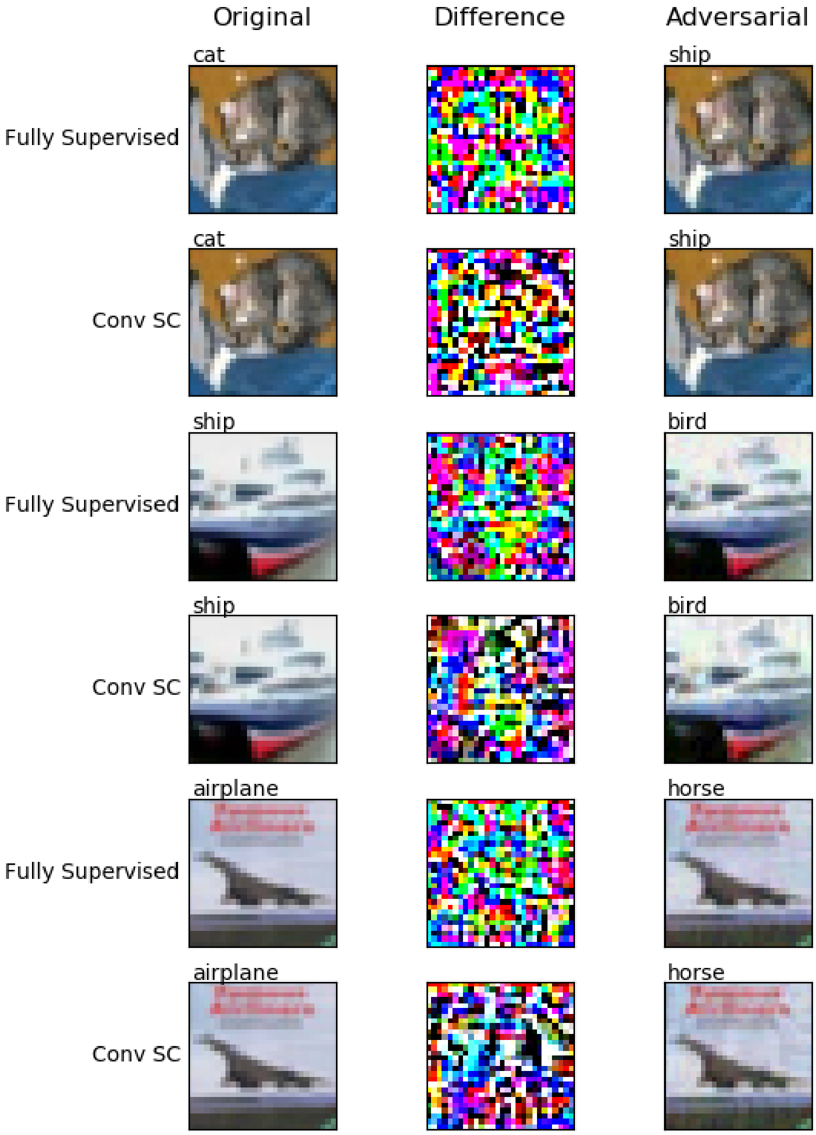


Figure 10: Example adversarial images for the fully supervised model and the convolutional sparse coding model. The output class of the model given the image is shown above the image. The difference denotes the adversarial perturbation, scaled to the range of the image.

misclassified a perturbed image to be the target class) for 99% of the tested images. Table 8 in the appendix shows the success rates for all models tested.

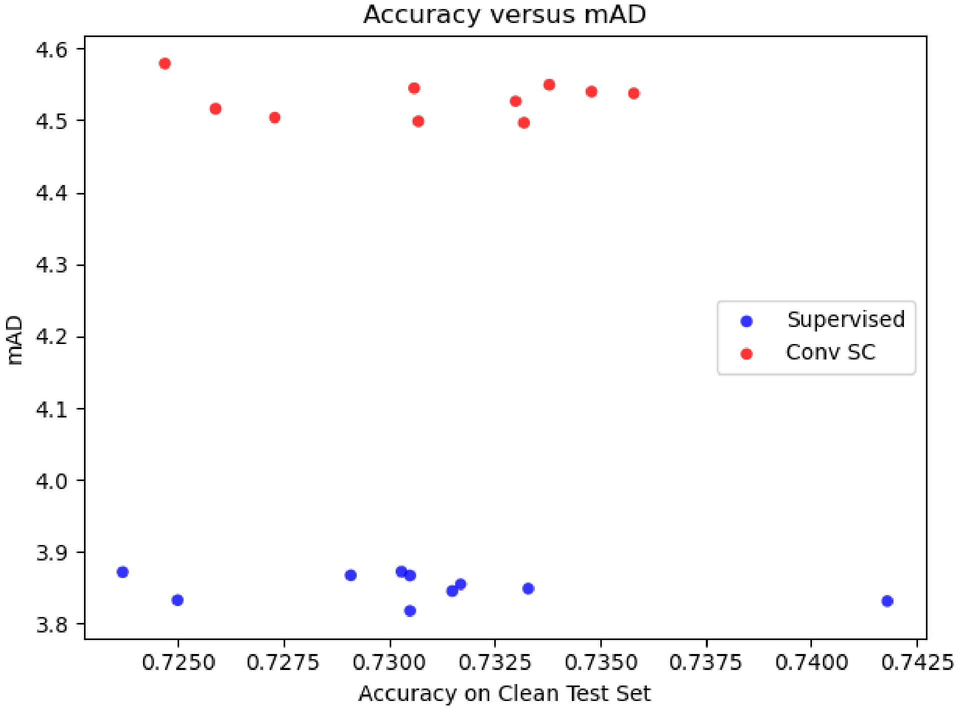


Figure 11: Accuracy on the clean test set versus mean adversarial distance (mAD) between the original and adversarial image, averaged across each image of the test set. Adversarial images are created by perturbing the image until the model reaches 90% confidence on a randomly selected target class. Here, the fully supervised and sparse coding models is run 10 times with random initial weights, random data augmentation, random presentation of training images, and random adversarial target classes.

Tsipras et al. [70] has shown that there exists a trade-off between model robustness and accuracy. In other words, a model that achieves better accuracy (on a clean test set) typically is less robust to adversarial examples. To account for this trade-off with different models, I compare each model's accuracy on the clean test set versus the mAD. Figure 11 shows the results for the 10 runs per model for the two models tested. I find that the sparse coding model takes approximately

0.7 mAD more to achieve 90% confidence on a target class than the fully supervised model. Overall, this suggests that sparse coding is slightly more robust to adversarial examples than the fully supervised model, in that it takes a bigger perturbation of the initial image to fool the network to 90% confidence.

5.3.1 Effect of Sparsity on Adversarial Robustness

One question that arises is how the sparsity level (controlled by the λ hyperparameter) of the sparse coding models affects adversarial robustness. While I only focus on convolutional sparse coding in this section, I adversarially attack all the models defined in Sections 4.2.2 and 4.2.3 with the same set of target labels for each test image across all models. The mAD for these results are shown in Table 7 in the appendix.

I take the convolutional sparse coding models using the five different λ values tested. Each λ value is used for both dictionary learning and encoding. Figure 12 compares the mAD versus the clean accuracy for these five models.

Here, the figure shows there exists a correlation between the sparsity value and the mAD. Specifically, I find that the sparser the code, the larger the perturbation needed to achieve 90% confidence to a target class at the expense of accuracy. However, the loss in accuracy is minimal, in that the difference in accuracy between the five models is within the range of accuracies due to random training conditions shown in Figure 8. Overall, this finding supports other works that have shown that sparse representations help protect against adversarial examples.

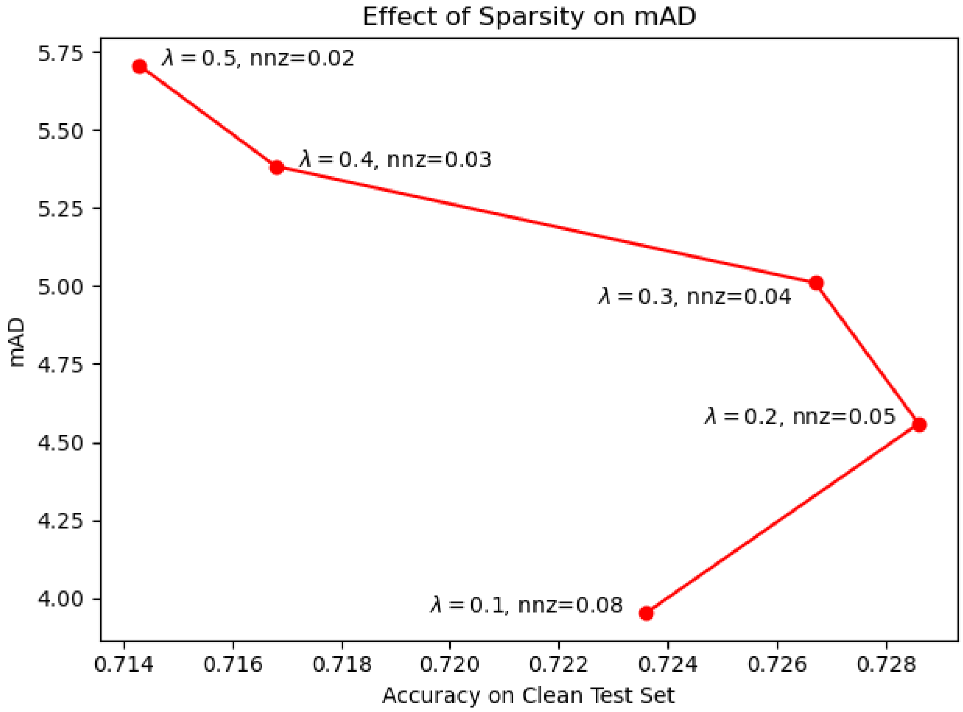


Figure 12: Accuracy on the clean test set versus mAD on sparse coding models with different λ values, averaged across each image of the test set. The fraction of nonzero elements (nnz) is shown next to each data point. The target adversarial class is set to be identical between different models.

5.3.2 Calibration of Classification Models

One explanation of the difference in adversarial robustness between the sparse coding and fully supervised models can be due to the confidence calibration of the model. Here, confidence calibration is defined as tuning the output of the softmax function (Equation 9) to represent the likelihood of an estimate to be the correct class [21]. While the calibration of a model's output to represent a likelihood is outside the scope of this work, the metric of the calibration is important when comparing different models on adversarial robustness. In particular, the stopping criteria for adversarial perturbation is set to be when the output reaches 90% confidence on a target class. Hence, different calibrations of confidence can result in different stopping points between models.

I use the expected calibration error (ECE) as a measure of calibration [49] in order to equate the different models to have similar confidence outputs. Here, the ECE is computed by binning a set of confidence outputs of size n into M equally spaced intervals. Let B_m be all samples that fall into a given interval m. The ECE is defined as

$$\text{ECE} = 100 \sum_{m=1}^{M} \frac{|B_m|}{n} \Big| \text{acc}(B_m) - \text{conf}(B_m) \Big| \tag{21}$$

where $\text{acc}(B_m)$ and $\text{conf}(B_m)$ measures the accuracy and average confidence of the given samples B_m respectively. Note that the ECE is multiplied by 100 to interpret the result as a percentage. Intuitively, a low ECE implies that the output confidence values accurately represents the likelihood of being the correct class. I use 50 intervals for all experiments.

Figure 13 shows the ECE values for the 10 independent runs for the fully supervised model and the convolutional sparse coding model corresponding to Figure 11. Table 2 shows the ECE values for the sparse coding models while varying the sparsity level corresponding to Figure 12. Table 9 in the appendix shows the ECE for all tested models. Here, I find a substantial difference in ECE between the fully supervised networks and the convolutional sparse coding, but less so when comparing the same model while varying λ.

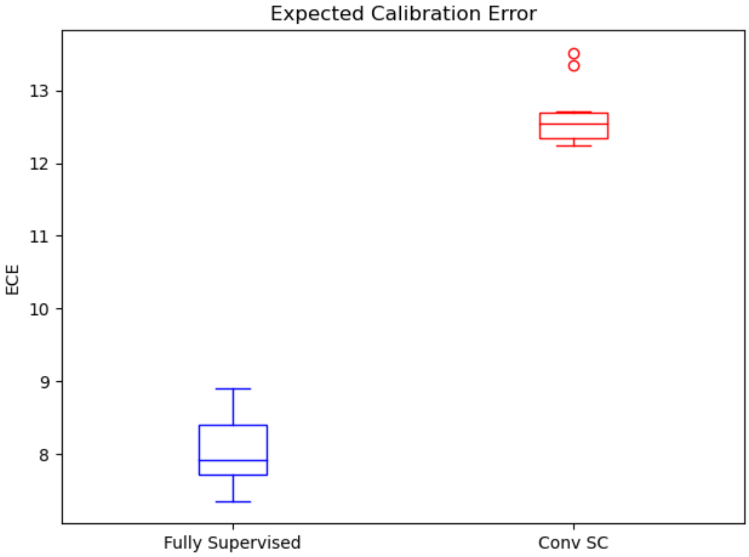


Figure 13: The Expected Calibration Error (ECE) values of the 10 independent runs for fully supervised versus convolutional sparse coding. The ECE reflects the errors of the confidence values of a given model.

In order to calibrate the models' probability, I use temperature scaling [21]. Here, let T be a scalar parameter that scales the activation values before computing the softmax. Formally, Equation 9 is updated to

$$\hat{\boldsymbol{y}}_i = \sigma(\boldsymbol{a}/T)_i \tag{22}$$

Model	ECE
$\lambda = 0.1$	13.7190
$\lambda = 0.2$	12.3581
$\lambda = 0.3$	11.3411
$\lambda = 0.4$	13.0975
$\lambda = 0.5$	12.1165

Table 2: Expected Calibration Error of classification models using convolutional sparse coding with different λ values for encoding and dictionary learning.

where I calculate the output of the model $\hat{\boldsymbol{y}}$ by dividing the activations $\boldsymbol{a}$ by a scalar T before feeding into the softmax function σ. If $T > 1$, the output confidences are "softened", i.e., make the model less confident overall. Conversely, if $T < 1$, the output overall is more confident in its predictions. Note that the temperature scaling does not change the output class prediction, and therefore does not change the accuracy of the model.

I calibrate the 10 runs for both the fully supervised model and the sparse coding model. Here, I use $T = 0.2$ for the fully supervised network and $T = 0.16$ for the sparse coding model to lower the ECE for both models. Figure 14 shows the resulting ECE values for both models.

Figure 15 shows the accuracy versus mAD on the calibrated models. Overall, both calibrated models exhibit slightly less robustness to adversarial examples than the uncalibrated models. The convolutional sparse coding model is still more robust to adversarial examples than the fully supervised model. However, the calibration has lessened the difference in adversarial distance between the two models to approximately 0.25 mAD.

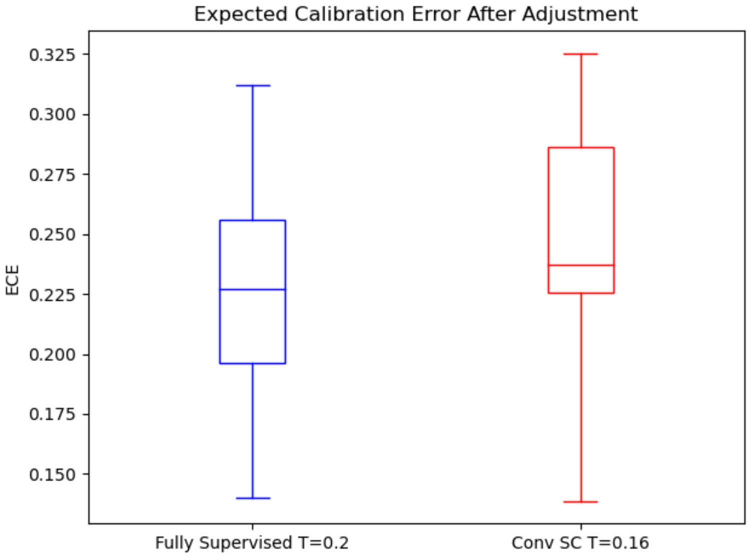


Figure 14: Box-and-whisker plot of the Expected Calibration Error (ECE) values of the 10 independent runs for fully supervised versus convolutional sparse coding after calibration. T is the scalar temperature value used.

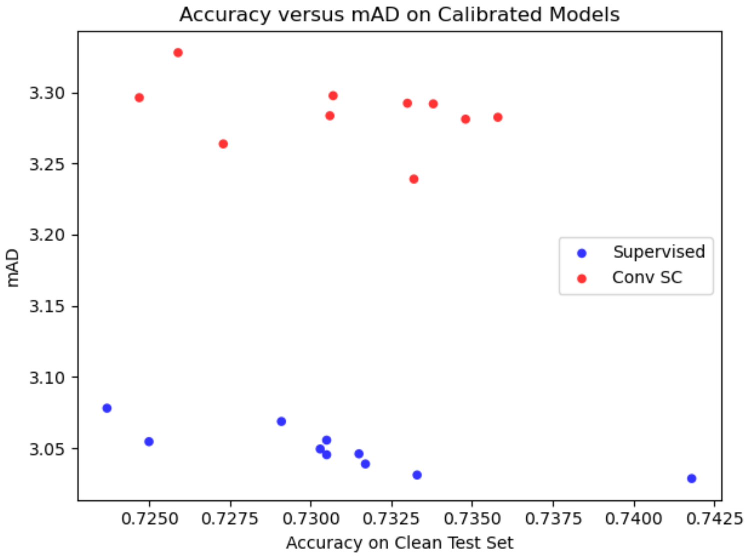


Figure 15: Accuracy versus mean adversarial distance (mAD) on temperature calibrated model.

5.4 Robustness of Sparse Coding on Corrupted Images

Section 5.3 explored sparse coding and adversarial examples. In this section, I explore the ability of sparse coding within a classification model versus a fully supervised model to generalize to (non-adversarial) common corruptions on images when trained on clean images.

The dataset I use for corrupted images is the CIFAR-10-C dataset [25]. Here, the dataset contains 19 different types of corruptions, classified into 4 categories defined below:

- **Noise**: Gaussian noise, impulse noise, shot noise, spatter, speckle
- **Blur**: Defocus blur, Gaussian blur, glass blur, motion blur, zoom blur
- **Weather**: Brightness, fog, frost, snow
- **Digital**: Contrast, elastic transform, JPEG compression, pixelate, saturate

Each corruption has 5 different levels of severity. Figure 16 shows examples of all corruptions at the highest corruption severity. Figure 21 in the appendix shows examples of all 5 severity levels.

I use the convolutional sparse coding model defined in Section 4.2.2 and 4.2.3 with $\lambda = 0.2$ (for both encoding and dictionary learning), along with the fully supervised model defined in Section 4.2.5. Here, both models are trained 10 independent times on the clean training set with random initial weights, random ordering of training examples, and random data augmentations. After training, I test each model with the test set for each of the 19 corruption at the 5 severity levels. The full results are shown in Figure 22 in the appendix.

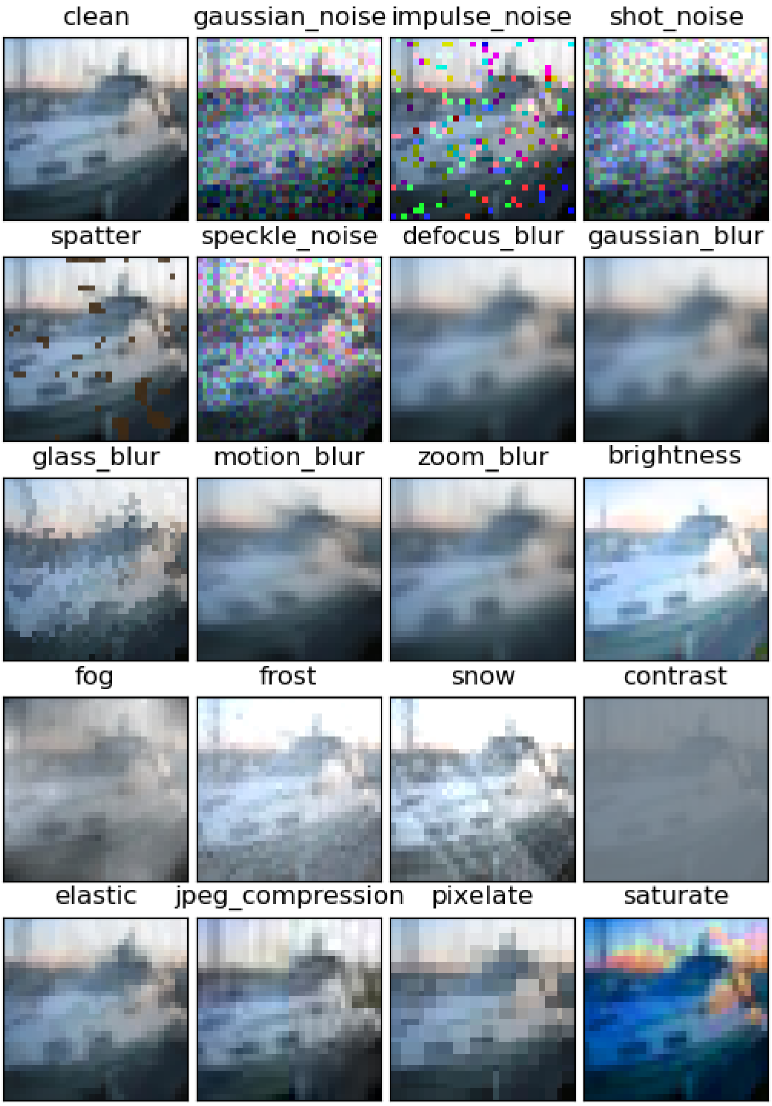


Figure 16: Examples of corrupted images from the CIFAR-10-C dataset.

In order to aggregate the performance across corruptions of varying difficulty, I use the Corruption Error (CE) as defined by Hendrycks et al. [25]. Here, the CE aims to normalize each accuracy based on the performance of a baseline model per corruption. Formally, the CE is defined as

$$\mathrm{CE}_c^f = \left(\sum_{s=1}^{5} E_{s,c}^f\right) \Bigg/ \left(\sum_{s=1}^{5} E_{s,c}^{f^*}\right) \quad . \tag{23}$$

Here, each classification error E for model f, corruption type c, and severity level s is normalized by the performance per corruption and severity on a baseline model f^*. I use the fully supervised model with the median accuracy out of the 10 runs as the baseline model. The normalized CE scores can then be averaged to find the mean CE (mCE) across collections of different corruptions. Here, lower values of mCE correspond to more robustness for the corruption type. Figure 17 shows the mCE averaged across all corruptions, as well as the four different corruption types.

Here, I find that sparse coding is able to generalize to noise corruptions better than a fully supervised model. This is expected, as sparse coding has been shown to remove high frequency noise [10,27]. However, sparse coding tends to achieve worse performance than the fully supervised model on weather and digital corruptions, which tended to keep low frequency features. While the fully supervised model tended to be slightly more robust to low frequency blur corruptions, the range for fully supervised mCE is wider than that of the sparse coding model. One possible

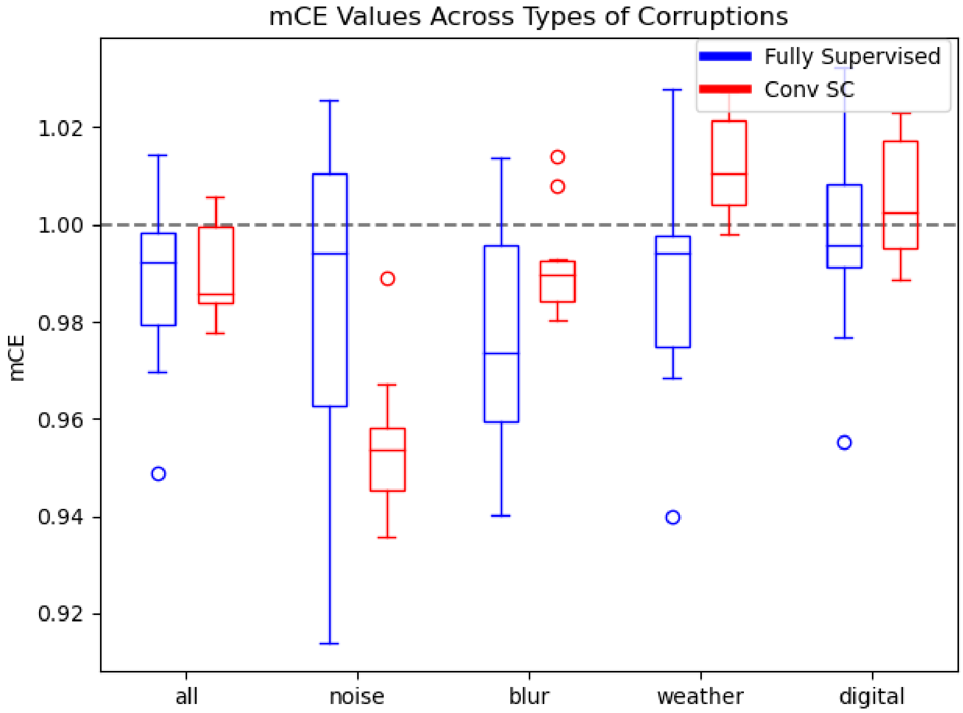


Figure 17: Mean corruption error (mCE) for a fully supervised model versus convolutional sparse coding on different types of corruption. Lower values of mCE corresponds to more robustness to the corruption type. Each model is trained 10 times with random conditions. The mCE is normalized per corruption and severity on a baseline model, such that the baseline model has an mCE of 1.

explanation of the difference in range is that the unsupervised sparse coding layer was not retrained for each of the 10 runs.

5.5 Summary

In summary, I find that using sparse coding within a classification framework results in robustness against adversarial examples. This is likely due to the population nonlinearity exhibited by sparse coding models, which makes activations more selective to image features instead of adversarial noise [52]. Another explanation could be the denoising nature of sparse coding. However, the adversarial attacks were done in a white-box setting, which takes into account the sparse coding model itself.

For non-adversarial corruptions, sparse coding is able to generalize better than fully supervised networks to unseen high frequency noise corruptions. However, the fully supervised network tended to be more robust to corruptions that kept low frequency image features. One possible explanation is that the fully supervised model can be using low frequency features (e.g., color) for classification more than the sparse coding model.

Future work entails exploring the effect of the denoising nature of sparse coding for model robustness. In particular, testing other types of less computationally expensive denoising algorithms would determine the relative contribution of denoising versus the nonlinear encoding of sparse coding.

6 Sparse Coding with Limited Labels

One limitation of DCNNs is that they rely on large collections of human-labeled training data, e.g., ImageNet [8]. As such, DCNNs are restricted to domains for which large datasets of labeled examples are available.

In this section, I will explore the use of unsupervised sparse coding within a supervised DCNN to help alleviate the need for an abundance of training labels. Here, I explore the use of unsupervised sparse coding on stereo-video data on the task of a binary object classification task. I show that replacing a traditional convolutional layer trained via supervised learning with an unsupervised sparse coding layer allows for better and more consistent performance on the task when there are limited amounts of training labels. I additionally show that by applying unsupervised sparse coding to stereo-video data, the coding exhibits depth selectivity (i.e., activations are selective to elements based on the distance from the camera) as an emergent property of the model. In contrast, a comparable convolutional layer trained with supervised learning for the classification task does not exhibit depth selectivity. This may explain the difference in performance between the two models.

6.1 Related Work

Data Augmentation: One classical method of machine learning to help overcome limited training examples is to artificially enlarge a dataset via data augmentation. For example, Krizhevsky et al. [33] augmented the ImageNet dataset by randomly cropping and mirroring dataset images. Additionally, the authors systematically adjusted the intensity values of images. These techniques, while useful in practice, still suffer from limited data, as the resulting augmented examples are highly correlated. In this dissertation, I focus on exploring modifications to neural network architectures and training losses for alleviating limited data problems, and note that augmentation techniques are readily applicable to the models presented in this dissertation.

Transfer Learning: One solution to learning with limited data (and the solution I explore in this section) is that of transfer learning. For example, in the supervised setting, Marmanis et al. [46] takes a model trained on ImageNet and fine-tunes the model to classify overhead satellite imagery. In terms of using unsupervised learning, Raina et al. [55] poses this problem as self-taught learning and trains features using sparse coding for transfer to supervised tasks. Additionally, Erhan et al. [11] suggests theoretical reasons as to why unsupervised pre-training for supervised learning works. While these studies of unsupervised transfer learning explore the use of unsupervised learning techniques within a supervised network, they do not explore performance on natural scenes (instead, focusing on datasets such as MNIST [36] for handwritten digit recognition). Here, I extend these studies to domains of data captured "in the wild". Additionally, I explicitly compare performance between the use of unsupervised sparse-coding layer within a DCNN versus a fully supervised model.

Few-Shot Learning: Fei-Fei et al. [12] aimed to achieve one-shot or few-shot learning (i.e., learning a new class using one or few labeled training examples) using a Bayesian model that incorporates previously learned object categories into learning a target novel object. In contrast, this dissertation proposes to utilize unsupervised sparse coding to help alleviate the need for many labeled data points.

Unsupervised Learning for Classification: Lotter et al. [37] utilizes unsupervised learning to alleviate the need for labeled training data. Specifically, the authors use a recurrent neural network (RNN) [26] to predict future frames of a video. They additionally show that their network achieves better performance than standard DCNNs when each is trained on only a limited amount of training data. In contrast to future frame prediction, this dissertation proposes to achieve image representation through sparse coding.

Sparse Coding of Stereo Images: My colleagues and I have demonstrated that representations of stereo images obtained through sparse coding allow for an encoding that achieves better performance than a convolutional layer in the task of pixel-wise depth estimation [39]. We showed that the sparse encoding is inherently depth selective, whereas the convolutional encoding is not. In this dissertation, I extend this study to encode stereo-video clips and compare encodings on a vehicle-detection task.

6.2 Sparse Coding of Stereo-Video for Vehicle Detection

One domain in which sparse coding should be useful is in multi-view sensing, i.e., sensing an environment given multiple views of the scene. A sparse encoding, which must efficiently represent a given scene, should learn correlated visual features (that is generated from some physical object) between different viewpoints of the same scene, with some offset based on the viewpoints. These correlated offsets represent disparity in stereo images and optic flow in consecutive frames from a video. It follows that an encoding that accounts for such offsets should have some notion of depth [13,54].

I compare two types of convolutional network models that differ only in the first layer: (1) a *sparse-coding* network, in which the weights and activations in the first layer are computed via unsupervised convolutional sparse coding, and (2) a *fully-supervised* network, in which the first-layer weights are learned via supervised training and activations are computed using these layer weights. In both network models, the weights and activations in all subsequent layers are computed via supervised convolutional layers.

The data I use here is stereo-video data from KITTI [14]. The KITTI dataset consists of videos captured from two horizontally offset cameras (i.e., stereo cameras) mounted on a car. Multiple views of a scene are obtained from the stereo cameras, as well as multiple views in time due to the moving car. The final task is to detect vehicles in the scene.

I show that sparse-coding networks are able to achieve better performance than fully supervised networks on a vehicle detection task on stereo-video data with a minimal amount of training labels. Additionally, I show that the performance of sparse-coding networks is more consistent—i.e., more robust to randomized order of training data and random initializations—than comparable fully supervised networks. Finally, I show that activations in the first (sparse-coding) layer in the sparse-coding networks are depth selective, which may provide an explanation for the differences in performance I observe in this study.

6.2.1 Experiments

I compare fully supervised networks with networks incorporating an unsupervised sparse-coding layer by testing performance on a vehicle detection task using stereo-video. I test the effect of training set size by varying the number of labeled training examples available to the network. The KITTI dataset contains of approximately 7000 examples, which I split into 6000 for training and 1000 for testing. Each example consists of three stereo frames ordered in time, with bounding box annotations for various objects in the left camera's last frame as ground truth. I normalize the stereo-video inputs to have zero mean and unit standard deviation and I downsampled them to be 256×64 pixels. I concatenated stereo inputs such that the input contains six features, i.e., RGB inputs from both left and right cameras. I kept time in a separate dimension for three-dimensional convolutions for convolutional layers or transposed convolution for sparse-coding layers (see Section 3.3) across the time, height, and width axes of the input.

I generated the ground truth for this task by sliding a 32×16-pixel non-overlapping window across the left camera's last frame. I considered a window to be a positive instance if the window overlaps with any part of a car, van, or truck bounding box provided by the original ground truth. The final output of each network is the probability of a window containing a vehicle, for all windows in the frame. I use the cross-entropy (see Section 2.4) between the ground truth and estimated probabilities as the supervised cost function to train all supervised layers within the network.

I tested various encoding schemes along with various weight initializations for the first layer of n-layer networks, as follows:

- **ConvSup**: Convolutional encoding. Weights are initialized randomly and trained via supervised training for vehicle detection.
- **SparseUnsup**: Unsupervised sparse encoding to learn activations and weights.
- **ConvRand**: Convolutional encoding. Weights are initialized randomly and are not updated. This gives a random-weight baseline for the networks.
- **ConvUnsup**: Convolutional encoding. Weights are initialized from weights trained via unsupervised sparse coding and are not updated. Here, the activations are calculated from a single convolution, as opposed to **SparseUnsup** which finds a sparse encoding. This control tests the effect of weights versus encoding scheme on performance.
- **ConvFinetune**: Convolutional encoding. Weights are initialized from weights trained via unsupervised sparse coding. The weights were additionally trained via supervised training for vehicle detection. This control is similar to **ConvUnsup** but tests the effect of additional training on the first-layer weights.

Once the first layer is set to one of the five possible options, the remaining $n-1$ layers contain convolutional layers learned via backpropagation of the supervised loss. Each model was trained six times on the training data with different random initial conditions and random presentation order of the training data to get a range in performance for each network. Here, each model was limited to running six times due to computational complexity constraints.

Detection Metrics: The metric commonly used in classification tasks is accuracy: the percent of correct estimates versus the total number of samples. This metric works well when the number of samples per class is balanced, such as the CIFAR-10 dataset used in previous sections. When there is an imbalance of samples per class, such as the number of tiles that contains cars within the dataset, precision and recall are better metrics to use to assess a model's performance. Intuitively, precision measures the correct number of predictions out of all positive cases predicted, and recall measures the correct number of predictions out of all actual positive cases. Formally, precision is defined as $\frac{\mathrm{TP}}{\mathrm{TP+FP}}$ and recall as $\frac{\mathrm{TP}}{\mathrm{TP+FN}}$, where TP are true positives, FP are false positives, and FN are false negatives.

To aggregate both precision and recall into a single metric, I use the area under the precision versus recall plot (AUC under the PvR), or the Average Precision (AP). Here, the metric calculates the precision of a detection system at different recall values. Specifically, a precision versus recall curve can be made by varying a threshold on detection confidence to measure the trade-off between precision and recall. Finally, finding the area under this curve gives the AP score. The AP score ranges from 0 to 1, and a higher AP score corresponds to a better detection system overall.

6.2.2 Results

Table 3 shows the Average Precision (AP) of all models trained on all available training data and evaluated on the test data, each tested with two, three, and four layers. Each network was trained on the vehicle-detection task six times, with random weight initialization in higher layers and random ordering of training examples, in order to obtain the range of AP scores. The range columns in Table 1 gives the difference between the maximum and minimum scores over the six runs. A lower range in AP scores represent a model that is less susceptible to random training conditions.

Here, I find that **SparseUnsup** performs worse than **ConvSup** and **ConvFinetune** with two layers, which agrees with the findings of Coates et al. [7]. However, **SparseUnsup** outperforms **ConvSup** in networks with three or more layers. This difference in performance due to the number of layers is likely because of the lack in capacity of the two layer model to solve the detection task; more layers are needed.

One key finding is that **SparseUnsup** is much more consistent (i.e., much less susceptible to random initial conditions and ordering of training examples) compared to all other models, as shown by the low range of AP scores. This consistency is favorable, for example, when a model is too computationally expensive to run multiple times. Interestingly, **SparseUnsup** has a smaller range in performance than **ConvUnsup**, where both models use unsupervised weights learned via sparse coding and only differ in encoding scheme. This suggests that inferring

Model	2 layers	Range	3 layers	Range	4 layers	Range
ConvSup	**0.672**	0.045	0.672	0.021	0.681	0.086
SparseUnsup	0.619	**0.004**	**0.681**	**0.009**	**0.693**	**0.014**
ConvRand	0.467	0.009	0.574	0.052	0.592	0.033
ConvUnsup	0.561	0.044	0.609	0.028	0.609	0.033
ConvFinetune	0.660	0.020	0.641	0.081	0.691	0.117

Table 3: The Average Precision (AP) scores for all models tested with varying depths. Each model was run six times with different random training conditions. Each value represents the median AP score over the six runs, and the range represents the difference between the highest score and the lowest score. Bold face determines the highest median AP and lowest range of models tested.

activations in sparse coding is likely the reason for the additional consistency in performance.

Figure 18 gives the performance of **SparseUnsup** and **ConvSup** while varying the number of labeled training examples that each network was trained on. Here, the unsupervised weights were learned from all available training data, without using training labels. I find that **SparseUnsup** achieves better performance than **ConvSup** across all numbers of labeled training examples tested for three and four layer networks. Additionally, **SparseUnsup** achieves better performance with two layer networks when the number of training examples is limited to only 100 training labels. Finally, **SparseUnsup** is much more consistent (as shown by the low range in AP scores) in performance than that of **ConvSup**, for all models tested. Overall, these results show that a sparse coding network achieves better and more consistent results than fully supervised networks when training data is limited.

I compare activation maps for the first layer of **SparseUnsup**, **ConvUnsup**, and **ConvSup** in Figure 19a, 19b, and 19c respectively. I find that the sparse-coding activations are selective to certain depths. For example, in Figure 19a, the top row shows a fast moving edge detector with a large binocular shift that corresponds to image features close to the camera, whereas the bottom row shows a static edge detector with no binocular shift that corresponds to image features far from the camera. In contrast, no convolutional layers show depth selectivity.

One explanation of the depth selectivity seen in **SparseUnsup** is the level of sparsity exhibited in the model. In particular, I aim to see if sparsity alone, achieved with simply thresholding activation values, can account for depth selectivity instead of the sparse coding process. Shown in Figure 19d, I applied a threshold to convolutional activations in **ConvSup** (Figure 19c) to match the number of nonzero activations in sparse coding across the test set. Sparse controls for **ConvUnsup** produced all zero activations for the shown example. Here, I show that simply applying a threshold to activations from **ConvSup** is insufficient to explain depth selective activations. Overall, the depth selectivity of **SparseUnsup** (and the lack of depth selectivity in the other models) may explain the difference in performance in vehicle detection.

In this section, I have shown that a neural network that incorporates unsupervised learning is able to outperform a fully supervised network when there exists limited labeled training data. Additionally, I show that performance of fully super-

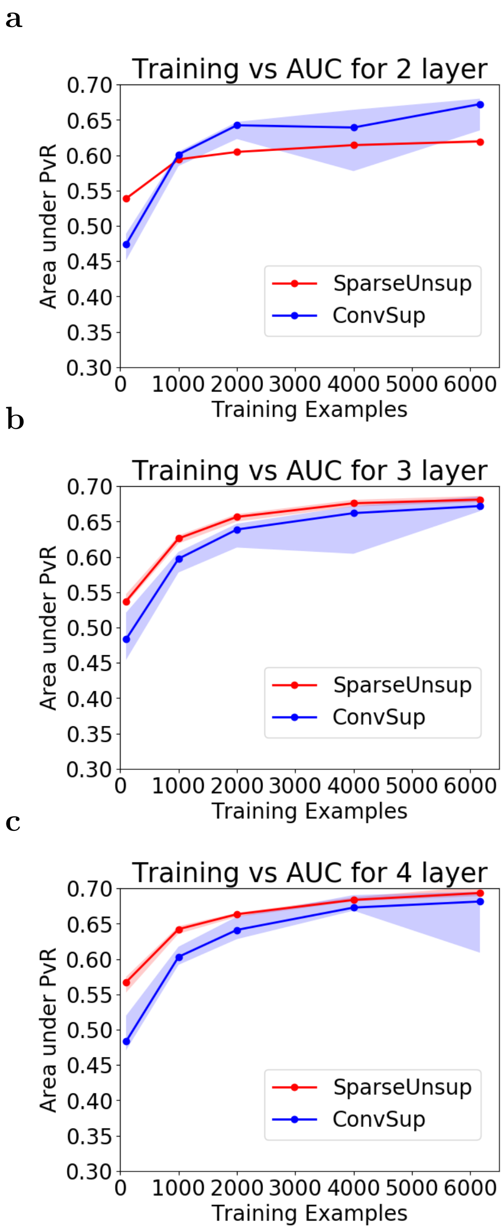


Figure 18: Number of training examples available versus Area Under Curve (AUC) of Precision versus Recall (PvR) scores for **SparseUnsup** (red) and **ConvSup** (blue) for two (**a**), three (**b**), and four (**c**) layer networks. Each point is the median score over six independent runs, with the area between the maximum and minimum score filled in. The leftmost point in each plot uses 100 training examples. Note that **SparseUnsup** (red) range is minimal, and therefore not visible in the plot.

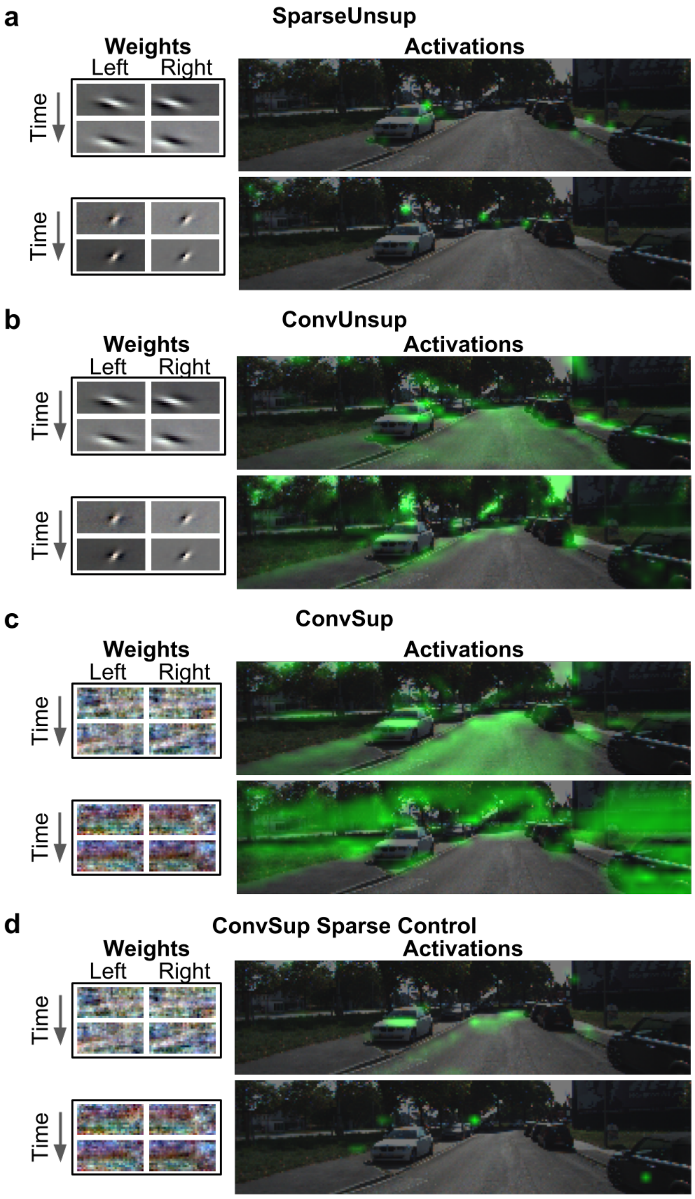


Figure 19: Nonzero activations of example weights overlaid on the input image. Magnitude of pixel values in green correspond to magnitude of activations. **a**: Activations of **SparseUnsup** for near tuned (top) and far tuned (bottom) weights for the sparse-coding layer. **d**: Activations from **ConvSup** with a threshold applied such that the number of activations matched that of sparse coding across the dataset. Sparse encodings show depth selectivity, whereas convolutional encodings do not.

vised networks can vary substantially based on initial conditions when compared to networks with a sparse-coding layer. Finally, I compare activations and show that depth selective activations emerge from applying sparse coding to stereo-video data. In all, these results show that unsupervised sparse coding can be useful in domains where there exists a limited amount of available labeled training data.

Further work entails determining if the result is constrained to highly correlated stereo-video data. In particular, further work must be done to see if these results generalize to multi-class classification on static images such as those from CIFAR-10. In addition, work must be done to see if these results generalize to multi-class object detection, where the objective is to draw tight bounding boxes around objects of interest.

7 Final Conclusions

In this dissertation, I explored the potential of using sparse coding within supervised DCNNs to help alleviate some shortcomings of such models. In particular, I explore the lack of robustness shown in DCNNs to adversarial and corrupted images not seen during training. Additionally, I explore the necessity of an abundance of human-annotated training examples used by DCNNs.

In Section 4, I test the relative contribution of dictionary learning versus encoding of unsupervised sparse coding when used within a supervised classification model. I find that the encoding method of unsupervised sparse coding, as opposed to the unsupervised dictionaries learned, contributes more to better classification results. In particular, the convolutional sparse coding method is key for getting good results on classification. Overall, however, the performance of a model that uses sparse coding is similar to that of a fully supervised model.

In Section 5, I test if using unsupervised sparse coding helps build a more robust classification model. Specifically, I test sparse coding's ability to protect against adversarial examples (Section 5.3) and find that using sparse coding within supervised networks helps alleviate the model's susceptibility to adversarial examples, even when attacking the sparse coding model in a white-box setting. I also test the robustness of sparse coding within classification models on common corruptions (Section 5.4), and find that sparse coding is robust to high frequency noise corruptions, but less so to corruptions that remove high frequency image features, such as blurring.

In Section 6, I test if unsupervised sparse coding helps alleviate the number of labeled training examples typically needed by DCNNs on stereo-video data. I find that using unsupervised sparse coding within a classification model helps achieve better performance than a fully supervised model when limiting the number of training labels.

Overall, this dissertation shows that the use of unsupervised sparse coding within DCNNs for classification tasks can help augment existing fully supervised solutions to classification. In addition, the advantages of sparse coding shown in the domain of model robustness and limited data can help confirm the ideas of efficient representations in biology.

While these results shows potential for using sparse coding within classification networks, there are some potential issues with the experiments done in this dissertation. In particular, most state-of-the-art DCNNs contain a large number of layers, whereas the models tested in this dissertation contain relatively few layers. Additionally, other DCNN methods use methods such as residual learning [24] to achieve state-of-the-art results. Overall, these cases show that classification performance for the models tested in this dissertation is far from saturated. To this end, future work entails testing larger and deeper networks for classification, such as incorporating recent work on deep sparse coding models (e.g., [65]).

Another aspect for future work entails comparing the performance of sparse coding for model robustness and with limited labels to other methods. For example, one common solution to alleviate these shortcomings is to use data augmentation. Data augmentation in particular can be used to train DCNNs to account for common types of corruptions, including adversarial examples [18,20,48,71]. In addition, data augmentation can artificially make a dataset larger to alleviate the number of training labels required [33]. While these techniques are easy to implement and use during evaluation, the goal of this dissertation is to explore substantial architectural changes to DCNNs as an alternative. Here, I note that these data augmentation techniques are readily applicable to the architectural changes proposed in this dissertation, and future work must be done in combining these techniques to explore overall performance of these models in cases of corrupted images and limited labels.

Finally, future work entails comparing sparse coding to other types of unsupervised feature and encoding techniques. For example, variational autoencoders [53] are a commonly used unsupervised learning technique, where the latent representation can even be trained to represent image labels for classification.

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015), `https://www.tensorflow.org/`, software available from tensorflow.org
2. Baldi, P., Hornik, K.: Neural networks and principal component analysis: Learning from examples without local minima. Neural Networks 2(1), 53–58 (1989)
3. Barlow, H.B.: Unsupervised learning. Neural Computation 1(3), 295–311 (1989)
4. Boureau, Y.L., Bach, F., LeCun, Y., Ponce, J.: Learning mid-level features for recognition. In: 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. pp. 2559–2566. Citeseer (2010)
5. Britz, D.: Understanding convolutional neural networks for NLP (2015), `"http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/"`, Online; accessed Feburary-25-2020
6. Coates, A., Ng, A.Y.: The importance of encoding versus training with sparse coding and vector quantization. In: Proceedings of the 28th International Conference on Machine Learning (ICML-11). pp. 921–928 (2011)
7. Coates, A., Ng, A.Y., Lee, H.: An analysis of single-layer networks in unsupervised feature learning. In: International Conference on Artificial Intelligence and Statistics (AISTATS). pp. 215–223 (2011)
8. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: ImageNet: A large-scale hierarchical image database. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 248–255. IEEE (2009)
9. Dodge, S., Karam, L.: A study and comparison of human and deep learning recognition performance under visual distortions. In: Computer Communication and Networks (ICCCN), 2017 26th International Conference on. pp. 1–7. IEEE (2017)
10. Elad, M., Aharon, M.: Image denoising via sparse and redundant representations over learned dictionaries. IEEE Transactions on Image Processing 15(12), 3736–3745 (2006)
11. Erhan, D., Bengio, Y., Courville, A., Manzagol, P.A., Vincent, P., Bengio, S.: Why does unsupervised pre-training help deep learning? Journal of Machine Learning Research 11(Feb), 625–660 (2010)
12. Fei-Fei, L., Fergus, R., Perona, P.: One-shot learning of object categories. IEEE transactions on pattern analysis and machine intelligence 28(4), 594–611 (2006)

13. Ferris, S.H.: Motion parallax and absolute distance. Journal of Experimental Psychology 95(2), 258 (1972)
14. Geiger, A., Lenz, P., Urtasun, R.: Are we ready for autonomous driving? The KITTI vision benchmark suite. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2012)
15. Geirhos, R., Janssen, D.H., Schütt, H.H., Rauber, J., Bethge, M., Wichmann, F.A.: Comparing deep neural networks against humans: Object recognition when the signal gets weaker. arXiv preprint arXiv:1706.06969 (2017)
16. Geirhos, R., Temme, C.R.M., Rauber, J., Schuett, H.H., Bethge, M., Wichmann, F.A.: Generalisation in humans and deep neural networks. arXiv preprint arXiv:1808.08750 (2018)
17. Glorot, X., Bordes, A., Bengio, Y.: Deep sparse rectifier neural networks. In: Aistats. p. 275 (2011)
18. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572 (2014)
19. Gopalakrishnan, S., Marzi, Z., Madhow, U., Pedarsani, R.: Combating adversarial attacks using sparse representations. arXiv preprint arXiv:1803.03880 (2018)
20. Gu, S., Rigazio, L.: Towards deep neural network architectures robust to adversarial examples. arXiv preprint arXiv:1412.5068 (2014)
21. Guo, C., Pleiss, G., Sun, Y., Weinberger, K.Q.: On calibration of modern neural networks. arXiv preprint arXiv:1706.04599 (2017)
22. Guo, Y., Zhang, C., Zhang, C., Chen, Y.: Sparse DNNs with improved adversarial robustness. In: Advances in Neural Information Processing Systems. pp. 242–251 (2018)
23. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 1026–1034 (2015)
24. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
25. Hendrycks, D., Dietterich, T.: Benchmarking neural network robustness to common corruptions and perturbations. arXiv preprint arXiv:1903.12261 (2019)
26. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Computation 9(8), 1735–1780 (1997)
27. Hyvärinen, A., Hoyer, P.O., Oja, E.: Sparse code shrinkage: Denoising by nonlinear maximum likelihood estimation. In: Advances in Neural Information Processing Systems. pp. 473–479 (1999)
28. Jo, J., Bengio, Y.: Measuring the tendency of CNNs to learn surface statistical regularities. arXiv preprint arXiv:1711.11561 (2017)
29. Kim, E., Yarnall, J., Shah, P., Kenyon, G.T.: A neuromorphic sparse coding defense to adversarial images. In: Proceedings of the International Conference on Neuromorphic Systems. pp. 1–8 (2019)
30. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)

31. Krizhevsky, A.: The CIFAR-10 dataset. "https://www.cs.toronto.edu/~kriz/cifar.html" (2009), Online; accessed Feburary-25-2020
32. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images. Tech. rep., University of Toronto (2009)
33. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems. pp. 1097–1105 (2012)
34. Lake, B.M., Ullman, T.D., Tenenbaum, J.B., Gershman, S.J.: Building machines that learn and think like people. Behavioral and Brain Sciences 40 (2017)
35. Landecker, W.: Interpretable machine learning and sparse coding for computer vision. Ph.D. thesis, Portland State University (2014)
36. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE 86(11), 2278–2324 (1998)
37. Lotter, W., Kreiman, G., Cox, D.: Deep predictive coding networks for video prediction and unsupervised learning. In: International Conference on Learning Representations (ICLR) (2017)
38. Lundquist, S.Y.: TFSparseCode. https://github.com/slundqui/TFSparseCode (2020)
39. Lundquist, S.Y., Paiton, D.M., Schultz, P.F., Kenyon, G.T.: Sparse encoding of binocular images for depth inference. In: IEEE Southwest Symposium on Image Analysis and Interpretation (SSIAI). pp. 121–124. IEEE (2016)
40. Maas, A.L., Hannun, A.Y., Ng, A.Y.: Rectifier nonlinearities improve neural network acoustic models. In: International Conference on Machine Learning Workshop on Deep Learning for Audio, Speech and Language Processing (2013)
41. MacQueen, J., et al.: Some methods for classification and analysis of multivariate observations. In: Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability. University of California (1967)
42. Mairal, J., Bach, F., Ponce, J., Sapiro, G., Zisserman, A.: Discriminative learned dictionaries for local image analysis. In: Computer Vision and Pattern Recognition (CVPR), 2008. IEEE Conference on. pp. 1–8. IEEE (2008)
43. Mairal, J., Bach, F., Ponce, J., Sapiro, G., Zisserman, A.: Non-local sparse models for image restoration. In: Computer Vision, 2009 IEEE 12th International Conference on. pp. 2272–2279. IEEE (2009)
44. Mairal, J., Elad, M., Sapiro, G.: Sparse representation for color image restoration. IEEE Transactions on Image Processing 17(1), 53–69 (2008)
45. Mairal, J., Ponce, J., Sapiro, G., Zisserman, A., Bach, F.R.: Supervised dictionary learning. In: Advances in Neural Information Processing Systems. pp. 1033–1040 (2009)
46. Marmanis, D., Datcu, M., Esch, T., Stilla, U.: Deep learning earth observation classification using ImageNet pretrained networks. IEEE Geoscience and Remote Sensing Letters 13(1), 105–109 (2016)
47. Marzi, Z., Gopalakrishnan, S., Madhow, U., Pedarsani, R.: Sparsity-based defense against adversarial attacks on linear classifiers. In: 2018 IEEE International Symposium on Information Theory (ISIT). pp. 31–35. IEEE (2018)

48. Miyato, T., Maeda, S.i., Koyama, M., Nakae, K., Ishii, S.: Distributional smoothing with virtual adversarial training. arXiv preprint arXiv:1507.00677 (2015)
49. Naeini, M.P., Cooper, G.F., Hauskrecht, M.: Obtaining well calibrated probabilities using Bayesian binning. In: Proceedings of the Association for the Advancement of Artifical Intelligence (AAAI) Conference on Artificial Intelligence. vol. 2015, p. 2901. NIH Public Access (2015)
50. Olshausen, B.A., Field, D.J.: Sparse coding with an overcomplete basis set: A strategy employed by V1? Vision Research 37(23), 3311–3325 (1997)
51. Paiton, D.M.: DeepSparseCoding. `https://github.com/dpaiton/DeepSparseCoding` (2020)
52. Paiton, D.M., Frye, C.G., Lundquist, S.Y., Bowen, J.D., Zarcone, R., Olshausen, B.A.: Selectivity and robustness of sparse coding networks. Journal of Vision (2020), In press.
53. Pu, Y., Gan, Z., Henao, R., Yuan, X., Li, C., Stevens, A., Carin, L.: Variational autoencoder for deep learning of images, labels and captions. In: Advances in Neural Information Processing Systems. pp. 2352–2360 (2016)
54. Qian, N.: Binocular disparity and the perception of depth. Neuron 18(3), 359–368 (1997)
55. Raina, R., Battle, A., Lee, H., Packer, B., Ng, A.Y.: Self-taught learning: transfer learning from unlabeled data. In: Proceedings of the 24th International Conference on Machine Learning. pp. 759–766. ACM (2007)
56. Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: Towards real-time object detection with region proposal networks. In: Advances in Neural Information Processing Systems (NIPS). pp. 91–99 (2015)
57. Rigamonti, R., Brown, M.A., Lepetit, V.: Are sparse representations really relevant for image classification? In: Computer Vision and Pattern Recognition (CVPR), 2011. IEEE Conference on. pp. 1545–1552. IEEE (2011)
58. Rosenfeld, A., Zemel, R., Tsotsos, J.K.: The elephant in the room. arXiv preprint arXiv:1808.03305 (2018)
59. Rozell, C., Johnson, D., Baraniuk, R., Olshausen, B.: Locally competitive algorithms for sparse approximation. In: IEEE International Conference on Image Processing (ICIP). vol. 4, pp. IV–169. IEEE (2007)
60. Ruder, S.: An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747 (2016)
61. Sahay, R., Mahfuz, R., El Gamal, A.: Combatting adversarial attacks through denoising and dimensionality reduction: A cascaded autoencoder approach. In: 2019 53rd Annual Conference on Information Sciences and Systems (CISS). pp. 1–6. IEEE (2019)
62. Schultz, P.F., Paiton, D.M., Lu, W., Kenyon, G.T.: Replicating kernels with a short stride allows sparse reconstructions with fewer independent kernels. arXiv preprint arXiv:1406.4205 (2014)
63. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: International Conference on Learning Representations (ICLR) (2015)

64. Springer, J.M., Strauss, C.S., Thresher, A.M., Kim, E., Kenyon, G.T.: Classifiers based on deep sparse coding architectures are robust to deep learning transferable examples. arXiv preprint arXiv:1811.07211 (2018)
65. Sulam, J., Papyan, V., Romano, Y., Elad, M.: Multilayer convolutional sparse modeling: Pursuit and dictionary learning. IEEE Transactions on Signal Processing 66(15), 4090–4104 (2018)
66. Sun, B., Tsai, N.h., Liu, F., Yu, R., Su, H.: Adversarial defense by stratified convolutional sparse coding. arXiv preprint arXiv:1812.00037 (2018)
67. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199 (2013)
68. Taigman, Y., Yang, M., Ranzato, M., Wolf, L.: Deepface: Closing the gap to human-level performance in face verification. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 1701–1708 (2014)
69. Torralba, A., Oliva, A.: Statistics of natural image categories. Network: Computation in Neural Systems 14(3), 391–412 (2003)
70. Tsipras, D., Santurkar, S., Engstrom, L., Turner, A., Madry, A.: Robustness may be at odds with accuracy. arXiv preprint arXiv:1805.12152 (2018)
71. Wang, X., Shrivastava, A., Gupta, A.: A-Fast-RCNN: Hard positive generation via adversary for object detection. In: IEEE Conference on Computer Vision and Pattern Recognition (2017)
72. Yang, J., Yu, K., Gong, Y., Huang, T.: Linear spatial pyramid matching using sparse coding for image classification. In: IEEE Conference on Computer Vision and Pattern Recognition. pp. 1794–1801. IEEE (2009)
73. Ye, S., Xu, K., Liu, S., Cheng, H., Lambrechts, J.H., Zhang, H., Zhou, A., Ma, K., Wang, Y., Lin, X.: Adversarial robustness vs. model compression, or both? In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 111–120 (2019)
74. Zaadnoordijk, L., Besold, T.R., Cusack, R.: The next big thing(s) in unsupervised machine learning: Five lessons from infant learning. arXiv preprint arXiv:2009.08497 (2020)
75. Zeiler, M.D., Krishnan, D., Taylor, G.W., Fergus, R.: Deconvolutional networks. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 2528–2535. IEEE (2010)
76. Zhang, X., Kenyon, G.: A deconvolutional strategy for implementing large patch sizes supports improved image classification. In: Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS). pp. 529–534 (2016)
77. Zhu, M., Rozell, C.J.: Visual nonclassical receptive field effects emerge from sparse coding in a dynamical system. PLoS Computational Biology 9(8), e1003191 (2013)

Appendix A

Sparse Coding Results

In this section, I report various results of the sparse coding algorithm on both dictionary learning (Section 4.2.2) and encoding (Section 4.2.3).

Table 4 shows the reconstruction error for each tested model. Specifically, the error is calculated as $\|\boldsymbol{x} - \boldsymbol{D}\boldsymbol{a}_2^2\|$, or the ℓ_2 norm of the difference between the input and reconstruction, averaged across the entire test set. Table 5) shows the percent of active elements for all models tested.

I find that using the same λ value for both encoding and dictionary learning doesn't necessarily result in lower reconstruction error. For example, shown in Table 4, the best reconstructions using **Patch SC** with $\lambda = 1.5$ for encoding uses a dictionary trained with $\lambda = 0.5$. **Conv SC** encoding models tend to have the lowest reconstruction errors using the dictionary with the same value of λ as the encoding model. However, using a dictionary trained with a patch-based model in a convolutional sparse coding model encoder results in much worse reconstruction errors, and vice versa.

The percent of active elements (shown in Table 5) is highly dependent on the λ value used for encoding, but isn't dependent on the dictionary used, with the exception of imprinted and random dictionaries.

Figure 20 shows a subset of elements trained from the dictionary learning models defined in Section 4.2.2. Here, I find that the **Conv SC** dictionary are small edge detectors in a localized area. In contrast, the **Patch SC** dictionary contains

			Encoding Model									
			Patch SC					Conv SC				
			$\lambda = 0.5$	$\lambda = 1.0$	$\lambda = 1.5$	$\lambda = 2.0$	$\lambda = 2.5$	$\lambda = 0.1$	$\lambda = 0.2$	$\lambda = 0.3$	$\lambda = 0.4$	$\lambda = 0.5$
		Imprinted	8.6098	12.1507	15.4016	18.4985	21.5836	39.1536	41.0901	54.8490	68.5837	81.8337
		Random	38.9825	59.0116	72.3550	79.4238	82.6790	21.5020	77.7090	174.2944	311.5984	475.7586
Dictionary Learning Model	Patch SC	$\lambda = 0.5$	**3.6396**	**6.8210**	**10.2304**	13.7782	17.4653	11.2586	20.5298	31.0603	42.5035	54.6425
		$\lambda = 1.0$	4.4969	7.3402	10.4028	**13.6154**	16.9810	16.3972	25.9504	36.1265	46.9060	58.2100
		$\lambda = 1.5$	5.4602	8.0773	10.8660	13.8047	**16.9108**	21.5620	31.8765	42.1807	52.6703	63.4265
		$\lambda = 2.0$	6.4653	8.9265	11.4972	14.2166	17.1191	27.7955	38.1144	48.6017	58.8919	69.2251
		$\lambda = 2.5$	7.3224	9.7176	12.1581	14.7243	17.4670	33.9988	43.5082	54.4212	64.8445	75.1153
	Conv SC	$\lambda = 0.1$	12.7344	16.8004	20.3497	23.7387	27.1250	**4.4059**	11.7025	21.1712	31.9539	43.5809
		$\lambda = 0.2$	10.8614	14.9977	18.6075	22.0185	25.3981	5.1816	**11.3914**	19.6396	29.5353	40.6781
		$\lambda = 0.3$	10.1375	14.0457	17.5899	21.0338	24.4975	6.5013	12.3048	**19.6059**	**28.3632**	38.4194
		$\lambda = 0.4$	9.4505	13.3924	16.9703	20.4389	23.9331	8.0470	13.8080	20.5595	28.4280	**37.4210**
		$\lambda = 0.5$	9.3991	13.1821	16.6367	20.0327	23.4998	9.6918	15.6497	22.1949	29.5360	37.7679

Table 4: Table of reconstruction errors for each combination of dictionary learning and encoding. The lowest reconstruction error for each encoding model (i.e., each column) is bold.

		Encoding Model														
		Soft Threshold					Patch SC					Conv SC				
		$\alpha = 0.5$	$\alpha = 1.0$	$\alpha = 2.0$	$\alpha = 3.0$	$\alpha = 4.0$	$\lambda = 0.5$	$\lambda = 1.0$	$\lambda = 1.5$	$\lambda = 2.0$	$\lambda = 2.5$	$\lambda = 0.1$	$\lambda = 0.2$	$\lambda = 0.3$	$\lambda = 0.4$	$\lambda = 0.5$
	Imprinted	0.4747	0.4291	0.3439	0.2699	0.2081	0.1127	0.0712	0.0534	0.0431	0.0360	0.0880	0.0547	0.0411	0.0335	0.0284
	Random	0.2744	0.1279	0.0221	0.0036	0.0006	0.1819	0.0699	0.0248	0.0084	0.0029	0.1729	0.1035	0.0671	0.0434	0.0274
Dictionary Learning Model Patch SC	$\lambda = 0.5$	0.4439	0.3772	0.2692	0.1901	0.1323	0.1066	0.0651	0.0476	0.0376	0.0309	0.0848	0.0516	0.0379	0.0301	0.0250
	$\lambda = 1.0$	0.4537	0.3939	0.2915	0.2120	0.1518	0.1055	0.0643	0.0471	0.0372	0.0307	0.0847	0.0518	0.0382	0.0305	0.0255
	$\lambda = 1.5$	0.4585	0.4026	0.3050	0.2271	0.1666	0.1041	0.0640	0.0469	0.0371	0.0307	0.0846	0.0518	0.0382	0.0306	0.0256
	$\lambda = 2.0$	0.4644	0.4137	0.3216	0.2442	0.1816	0.1019	0.0634	0.0469	0.0373	0.0310	0.0840	0.0516	0.0383	0.0307	0.0258
	$\lambda = 2.5$	0.4673	0.4186	0.3297	0.2538	0.1913	0.1005	0.0631	0.0469	0.0375	0.0311	0.0839	0.0517	0.0384	0.0309	0.0260
Conv SC	$\lambda = 0.1$	0.4439	0.3760	0.2644	0.1832	0.1258	0.0999	0.0606	0.0446	0.0355	0.0294	0.0821	0.0499	0.0368	0.0294	0.0246
	$\lambda = 0.2$	0.4487	0.3856	0.2769	0.1938	0.1336	0.1025	0.0602	0.0430	0.0334	0.0272	0.0833	0.0498	0.0361	0.0285	0.0236
	$\lambda = 0.3$	0.4534	0.3939	0.2880	0.2040	0.1414	0.1040	0.0616	0.0439	0.0339	0.0273	0.0839	0.0502	0.0364	0.0287	0.0237
	$\lambda = 0.4$	0.4568	0.3999	0.2972	0.2130	0.1486	0.1062	0.0634	0.0452	0.0348	0.0278	0.0843	0.0507	0.0369	0.0291	0.0240
	$\lambda = 0.5$	0.4572	0.4015	0.3018	0.2197	0.1555	0.1075	0.0651	0.0469	0.0363	0.0291	0.0843	0.0510	0.0372	0.0294	0.0243

Table 5: Table of the percent of non-zero activations for each combination of dictionary learning and encoding.

edge detectors that span the size of the patch. Intuitively, this makes sense: encoding image patches in isolation requires nonzero weights that span the size of the patch in order to reconstruct the patch, whereas a convolutional encoding is able to rely on other positionally translated elements for reconstruction. In particular, this is why convolutional sparse coding allows for a more over-complete dictionary than patch-based sparse coding when the number of elements is held fixed. It follows that the elements in the patch-based sparse coding method can likely be more localized given more elements.

I also find that the sparsity trade-off results in different dictionaries. In particular, for **Patch SC** dictionaries with a low value of λ (i.e., the encodings are less sparse with less reconstruction error), the trained dictionary contains highly localized elements. In contrast, when λ is high (i.e., the activations are constrained to be more sparse at the expense of more reconstruction error), the elements span the size of the patch. These results likely tie directly to the number of non-zero activations. In a non-sparse model, an active element can afford smaller contribution to the reconstruction since there are more active elements overall. In a sparser model, one active element must reconstruct more of the image, resulting in less localized elements. While this effect is more pronounced for **Patch SC**, **Conv SC** also exhibits this property to a lesser degree.

The weights learned from supervised training exhibit an interesting characteristic, in that a single element contains a mixture of edge detectors and low frequency image features. In contrast, the sparse coding models tend to separate

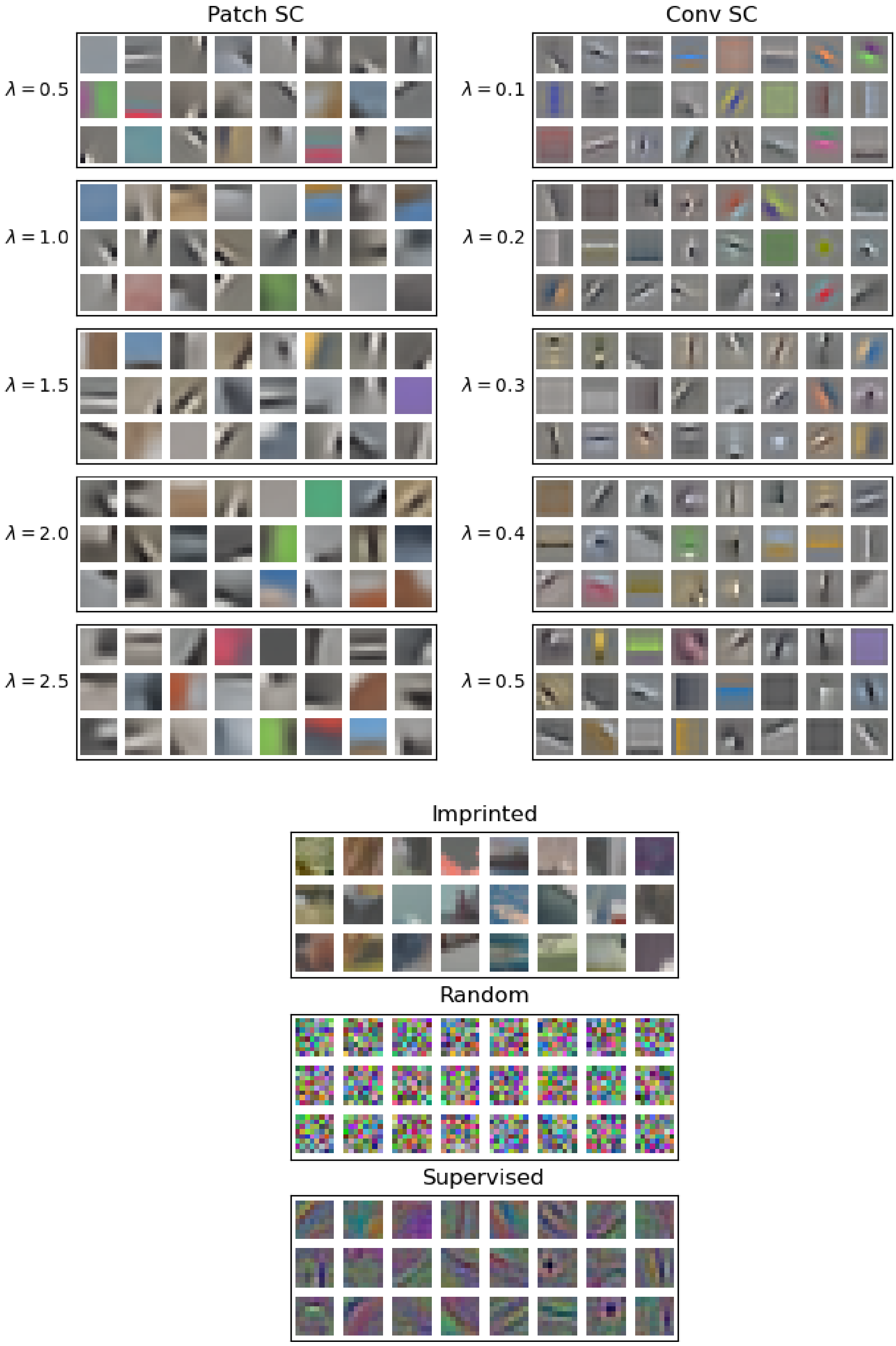


Figure 20: A subset of elements from dictionaries trained using **Patch SC**, **Conv SC**, imprinted (extracted from input images), and random dictionaries for image reconstruction. Supervised weights trained for image classification are also shown.

out high frequency edge detectors and low frequency color detectors into different elements.

Appendix B

Effect of Sparsity on Classification

This section details additional classification results for Section 4. Table 6 shows classification results on models with varying dictionary learning and encoding methods.

Throughout the models that incorporate a sparse coding first layer as an encoder, I find that the classifier benefits from a lower value of λ, i.e., when the model is less sparse (as shown in Figure 6). This effect has also been shown by Rigamonti et al. [57]. Here, one can argue that lower sparsity values can help due to lower reconstruction error, i.e., a more accurate representation of the input image. However, I find that a **Conv SC** encoding model with $\lambda = 0.1$ gets the best classification result with a dictionary trained with **Conv SC** with $\lambda = 0.3$, which does not achieve the lowest reconstruction errors for that encoding model.

Dictionary Learning Model		Encoding Model														
		Soft Threshold					Patch SC					Conv SC				
		$\alpha = 0.5$	$\alpha = 1.0$	$\alpha = 2.0$	$\alpha = 3.0$	$\alpha = 4.0$	$\lambda = 0.5$	$\lambda = 1.0$	$\lambda = 1.5$	$\lambda = 2.0$	$\lambda = 2.5$	$\lambda = 0.1$	$\lambda = 0.2$	$\lambda = 0.3$	$\lambda = 0.4$	$\lambda = 0.5$
	Imprinted	0.5386	0.5666	0.5618	**0.5674**	0.5382	**0.6284**	0.6024	0.6168	0.6186	0.6142	0.6864	**0.6911**	0.6819	0.6747	0.6697
	Random	0.6566	**0.6614**	0.5979	0.4929	0.3312	**0.5852**	0.5829	0.5718	0.5355	0.4664	0.6348	**0.6383**	0.6345	0.6309	0.6139
Patch SC	$\lambda = 0.5$	0.6177	0.6368	**0.6391**	0.6323	0.6275	0.6573	0.6537	0.6354	0.6229	0.6200	**0.7184**	0.7171	0.7106	0.7090	0.7032
	$\lambda = 1.0$	0.6102	0.5987	0.5890	0.6210	0.6259	**0.6639**	0.6389	0.6356	0.6306	0.6169	0.7150	0.7119	0.7043	0.7029	0.6897
	$\lambda = 1.5$	0.5982	0.5857	0.6128	0.6153	0.6073	0.6418	0.6471	0.6391	0.6352	0.6246	0.7112	0.7100	0.7073	0.6986	0.6977
	$\lambda = 2.0$	0.5698	0.5995	0.5729	0.5983	0.5922	0.6443	0.6277	0.6189	0.6275	0.6188	0.7062	0.7037	0.6983	0.6906	0.6839
	$\lambda = 2.5$	0.5624	0.5525	0.5973	0.5709	0.6063	0.6317	0.6281	0.6243	0.6300	0.6265	0.6981	0.7025	0.6968	0.6943	0.6853
Conv SC	$\lambda = 0.1$	0.6058	0.6196	0.6379	0.6332	0.6300	0.6359	0.6595	0.6369	0.6310	0.6256	0.7236	0.7243	0.7102	0.7032	0.6918
	$\lambda = 0.2$	0.6345	0.6174	**0.6441**	0.6140	0.6331	0.6254	0.6452	0.6632	0.6489	0.6183	0.7280	0.7286	0.7190	0.7232	0.7043
	$\lambda = 0.3$	0.6318	0.6099	0.6165	0.6190	0.6252	0.6622	0.6714	0.6487	0.6403	0.6355	**0.7355**	0.7321	0.7267	0.7168	0.7086
	$\lambda = 0.4$	0.6021	0.5916	0.5990	0.6199	0.6145	**0.6717**	0.6594	0.6639	0.6478	0.6343	0.7291	0.7261	0.7249	0.7168	0.7072
	$\lambda = 0.5$	0.6372	0.6155	0.6148	0.6260	0.6290	0.6656	0.6690	0.6494	0.6458	0.6576	0.7263	0.7297	0.7237	0.7179	0.7143

Table 6: Table of accuracies for each combination of dictionary learning and encoding model. Highest accuracies in each dictionary learning and encoding block is bold. The fully supervised model achieved 0.7298.

Appendix C

Adversarial Attacks on Sparse Coding

This section shows additional tables for Section 5.3. Table 7 shows the mean Adversarial Distance (mAD) between the original image and the adversarial image attacking each model, for all models tested with varying dictionary learning and encoding methods. Table 8 shows the success rate of the adversarial attack on each model. Table 9 shows the Expected Calibration Error (ECE) values for each model.

		Encoding Model														
		Soft Thresh					Patch SC					Conv SC				
		$\alpha = 0.5$	$\alpha = 1.0$	$\alpha = 2.0$	$\alpha = 3.0$	$\alpha = 4.0$	$\lambda = 0.5$	$\lambda = 1.0$	$\lambda = 1.5$	$\lambda = 2.0$	$\lambda = 2.5$	$\lambda = 0.1$	$\lambda = 0.2$	$\lambda = 0.3$	$\lambda = 0.4$	$\lambda = 0.5$
	Imprinted	7.3086	7.0375	7.0887	7.0705	**7.4486**	3.7323	4.1940	4.3104	4.5493	**4.7488**	4.5821	4.9882	5.2872	5.5594	**5.7808**
	Random	2.9223	2.9383	3.4707	5.2587	**10.2163**	2.2523	2.4095	2.7886	3.5338	**5.1120**	**3.6733**	3.3406	3.3068	3.4258	3.6206
Dictionary Learning Model Patch SC	$\lambda = 0.5$	5.6712	5.7696	6.0360	6.3974	6.8100	3.5691	4.1010	4.5685	4.8480	5.0845	4.4602	4.9066	5.2866	5.6079	5.9170
	$\lambda = 1.0$	6.1547	6.2828	6.5488	6.6244	6.8841	3.8523	4.3108	4.7269	5.0346	5.3149	4.6679	5.0719	5.4592	5.7847	6.0627
	$\lambda = 1.5$	6.3718	6.6003	6.6149	6.8646	7.1056	4.0154	4.4960	4.8024	5.0892	5.3843	4.8027	5.2550	5.6257	5.8967	6.1736
	$\lambda = 2.0$	6.9868	6.7924	7.0255	7.0032	7.2305	4.1817	4.6209	4.8981	5.2521	5.4119	4.9502	5.3991	5.7212	6.0103	6.2713
	$\lambda = 2.5$	7.1043	**7.3316**	7.0831	7.2576	7.2870	4.3214	4.7262	5.0077	5.3252	**5.5325**	5.0548	5.5047	5.8452	6.1396	**6.3614**
Convolutional SC	$\lambda = 0.1$	5.1701	5.1570	5.4119	5.9706	6.4348	3.2903	3.9431	4.4698	4.9577	**5.2894**	3.9523	4.5344	5.0666	5.5072	**5.8630**
	$\lambda = 0.2$	5.4822	5.4912	5.5565	5.9962	6.3536	3.3419	3.9024	4.3509	4.8291	5.2218	4.1263	4.5586	5.0003	5.3991	5.7617
	$\lambda = 0.3$	5.7966	5.8281	5.8683	6.0168	6.3764	3.3206	3.8212	4.2516	4.7196	5.0151	4.2978	4.6516	5.0101	5.3713	5.7222
	$\lambda = 0.4$	5.9410	6.1402	6.0750	6.1374	6.4350	3.3896	3.8118	4.1971	4.5665	4.9220	4.4354	4.7485	5.0626	5.3841	5.6941
	$\lambda = 0.5$	5.8906	6.1382	6.1558	6.3276	**6.4898**	3.4189	3.8291	4.2322	4.5368	4.8151	4.5706	4.8685	5.1392	5.4287	5.7070

Table 7: mAD between the clean and adversarial image for all models tested. The most robust model (i.e., highest mAD) within each block is in bold.

		Encoding Model														
		Soft Thresh					Patch SC					Convolutional SC				
		$\alpha = 0.5$	$\alpha = 1.0$	$\alpha = 2.0$	$\alpha = 3.0$	$\alpha = 4.0$	$\lambda = 0.5$	$\lambda = 1.0$	$\lambda = 1.5$	$\lambda = 2.0$	$\lambda = 2.5$	$\lambda = 0.1$	$\lambda = 0.2$	$\lambda = 0.3$	$\lambda = 0.4$	$\lambda = 0.5$
	Imprinted	0.9846	0.9891	0.9876	0.9923	0.9909	0.9848	0.9816	0.9845	0.9845	0.9855	0.9975	0.9991	0.9988	0.9985	0.9987
	Random	0.9973	0.9980	0.9980	0.9965	0.8513	0.9828	0.9891	0.9941	0.9966	0.9959	0.9987	0.9984	0.9986	0.9985	0.9987
Dictionary Learning Model Patch SC	$\lambda = 0.5$	0.9903	0.9910	0.9935	0.9944	0.9942	0.9840	0.9841	0.9815	0.9825	0.9849	0.9988	0.9983	0.9982	0.9985	0.9985
	$\lambda = 1.0$	0.9885	0.9879	0.9903	0.9914	0.9930	0.9849	0.9835	0.9822	0.9831	0.9835	0.9988	0.9990	0.9981	0.9977	0.9978
	$\lambda = 1.5$	0.9883	0.9877	0.9913	0.9913	0.9931	0.9840	0.9840	0.9824	0.9816	0.9854	0.9989	0.9986	0.9984	0.9982	0.9984
	$\lambda = 2.0$	0.9856	0.9889	0.9892	0.9919	0.9909	0.9851	0.9807	0.9828	0.9811	0.9835	0.9988	0.9990	0.9987	0.9988	0.9981
	$\lambda = 2.5$	0.9867	0.9849	0.9897	0.9889	0.9933	0.9825	0.9821	0.9815	0.9810	0.9829	0.9989	0.9986	0.9986	0.9983	0.9982
Convolutional SC	$\lambda = 0.1$	0.9898	0.9923	0.9931	0.9940	0.9965	0.9852	0.9892	0.9892	0.9892	0.9904	0.9991	0.9986	0.9982	0.9987	0.9982
	$\lambda = 0.2$	0.9904	0.9905	0.9925	0.9933	0.9944	0.9813	0.9845	0.9870	0.9879	0.9885	0.9987	0.9989	0.9983	0.9981	0.9984
	$\lambda = 0.3$	0.9895	0.9909	0.9924	0.9934	0.9948	0.9865	0.9872	0.9847	0.9869	0.9870	0.9990	0.9988	0.9986	0.9986	0.9989
	$\lambda = 0.4$	0.9886	0.9892	0.9905	0.9920	0.9936	0.9852	0.9854	0.9869	0.9851	0.9869	0.9991	0.9990	0.9984	0.9987	0.9984
	$\lambda = 0.5$	0.9917	0.9903	0.9924	0.9942	0.9946	0.9854	0.9867	0.9833	0.9848	0.9842	0.9991	0.9988	0.9985	0.9982	0.9972

Table 8: Success rate of the unbounded targeted adversarial attack.

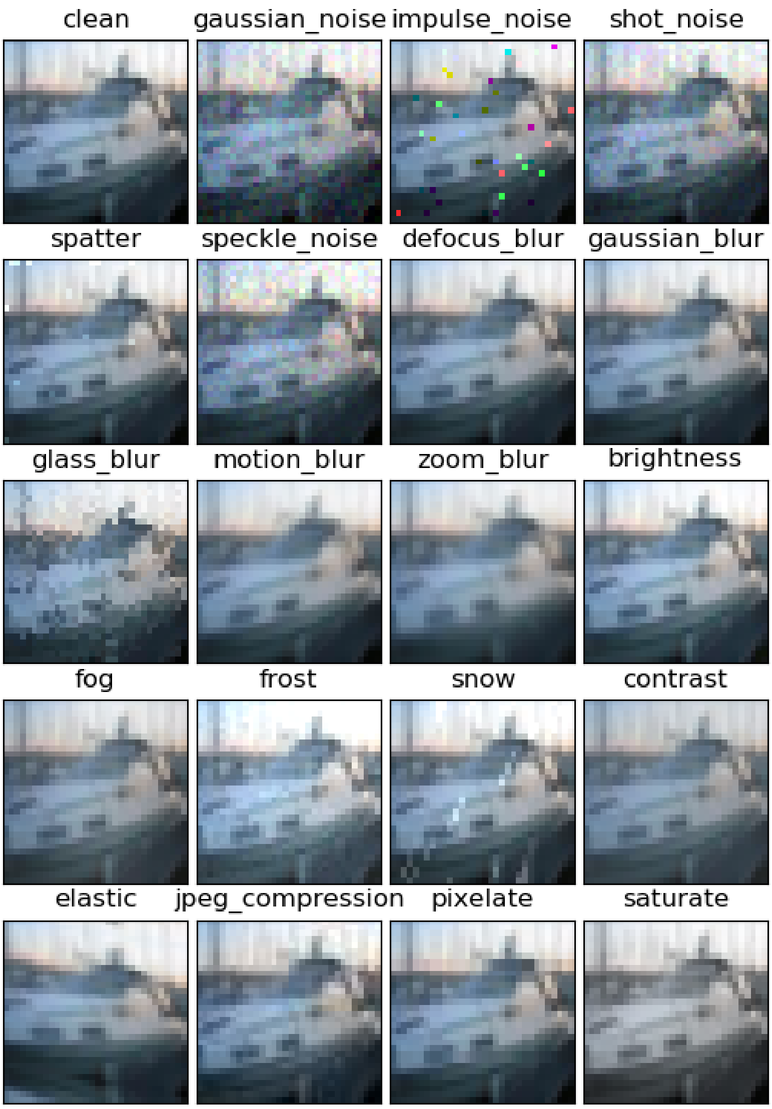
		Encoding Model														
		Soft Thresh					Patch SC					Conv SC				
		$\alpha = 0.5$	$\alpha = 1.0$	$\alpha = 2.0$	$\alpha = 3.0$	$\alpha = 4.0$	$\lambda = 0.5$	$\lambda = 1.0$	$\lambda = 1.5$	$\lambda = 2.0$	$\lambda = 2.5$	$\lambda = 0.1$	$\lambda = 0.2$	$\lambda = 0.3$	$\lambda = 0.4$	$\lambda = 0.5$
	Imprinted	4.7547	4.3208	5.2131	6.4561	7.1893	1.8600	2.2795	2.9646	3.0237	2.6000	16.0253	14.8100	15.4687	14.4317	14.3126
	Random	12.6667	13.1081	19.2271	30.5122	46.2933	3.3333	5.8849	11.2628	20.7856	30.5761	14.3035	14.4894	14.6367	15.8438	18.5163
Dictionary Learning Model Patch SC	$\lambda = 0.5$	4.5220	3.4485	5.4125	6.3231	7.3765	1.7792	1.7823	1.7336	1.7972	2.9653	15.2610	14.1037	12.9164	13.1675	14.0881
	$\lambda = 1.0$	3.7345	3.5822	4.7168	6.2674	6.7618	1.4721	1.3911	2.1655	2.6939	2.7678	14.4951	13.5306	13.0901	13.0447	12.2027
	$\lambda = 1.5$	3.9090	3.6354	5.4960	5.5558	6.4029	1.5758	1.7065	1.4866	1.8830	2.1135	15.9293	12.8469	12.3489	13.3012	13.0981
	$\lambda = 2.0$	3.4851	5.1618	4.2991	5.1067	5.8417	2.0613	1.7722	1.5878	1.3969	2.1255	15.3304	13.2480	12.8056	12.5277	13.3893
	$\lambda = 2.5$	3.5918	4.2268	5.1548	5.7962	6.5156	2.2753	2.0257	1.9365	1.6049	2.1084	15.5148	14.4902	14.1162	13.7821	13.7148
Convolutional SC	$\lambda = 0.1$	4.0621	6.1275	5.8228	7.1789	9.6671	2.1688	3.0488	3.4808	3.6342	4.8919	13.7190	12.9923	12.9833	13.1683	13.7721
	$\lambda = 0.2$	4.0229	4.8672	5.8251	5.5889	9.0718	1.5376	1.9266	3.0610	3.3468	4.2774	12.6532	12.3581	12.4045	12.4835	12.7097
	$\lambda = 0.3$	4.2711	4.3442	4.8287	5.4678	7.4904	1.8321	2.0741	2.4554	2.3968	3.2222	13.7177	12.7143	11.3411	11.2907	11.4001
	$\lambda = 0.4$	4.5674	4.6006	4.4746	6.1790	6.6376	1.8262	2.2363	2.6156	2.2600	2.9880	13.8662	12.4691	12.1148	13.0975	12.2681
	$\lambda = 0.5$	3.8583	4.0536	5.5701	5.8401	7.0445	1.5093	1.8999	2.0249	2.4451	2.1319	13.9212	11.6405	12.4232	12.2136	12.1165

Table 9: Expected Calibration Error (ECE) of tested models.

Appendix D

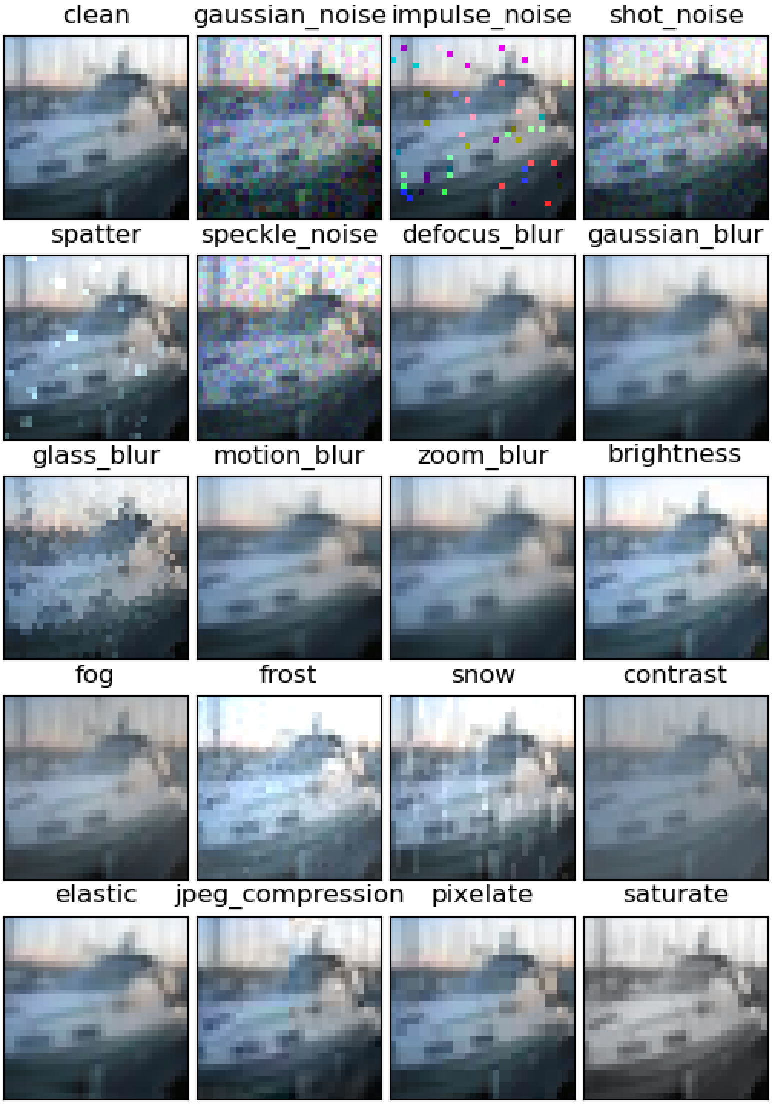
Classification on Corrupted Images

This section shows additional figures for Section 5.4. Figure 21 show examples of each corruption type at each level of severity. Figure 22 shows the accuracy of fully supervised versus convolutional sparse coding for each corruption type at each severity level.



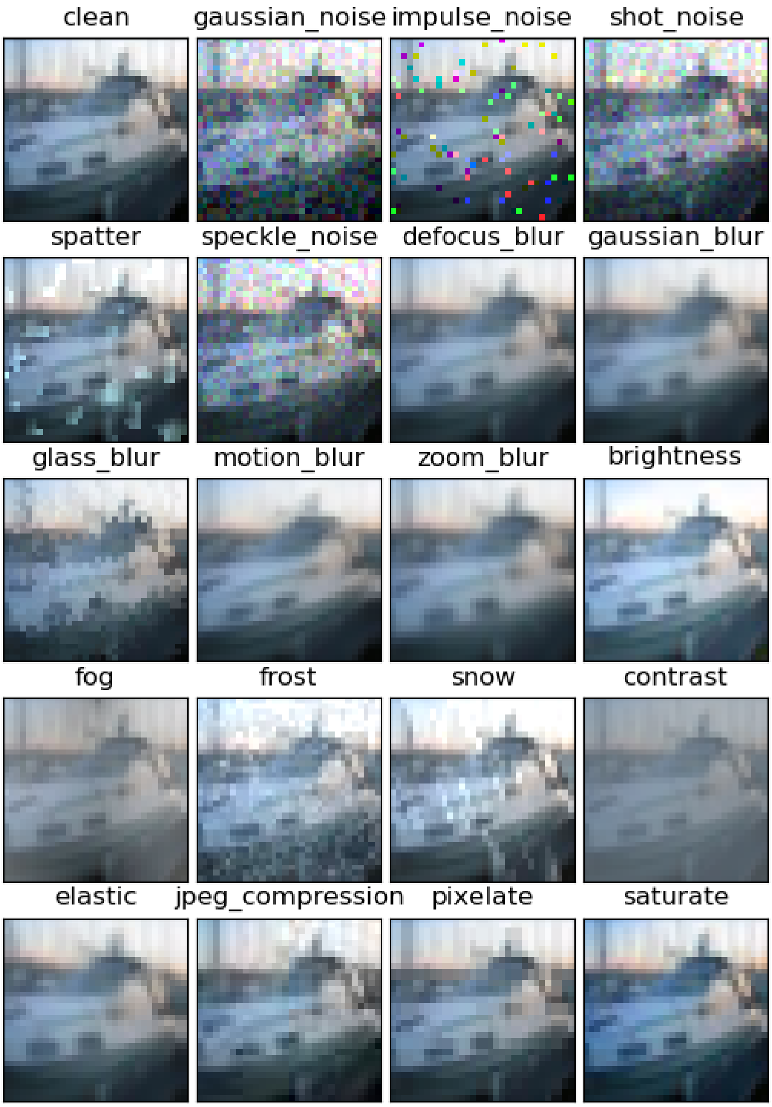
(a) Corruption severity 1

Figure 21: Examples of corrupted images from CIFAR-10-C [25].



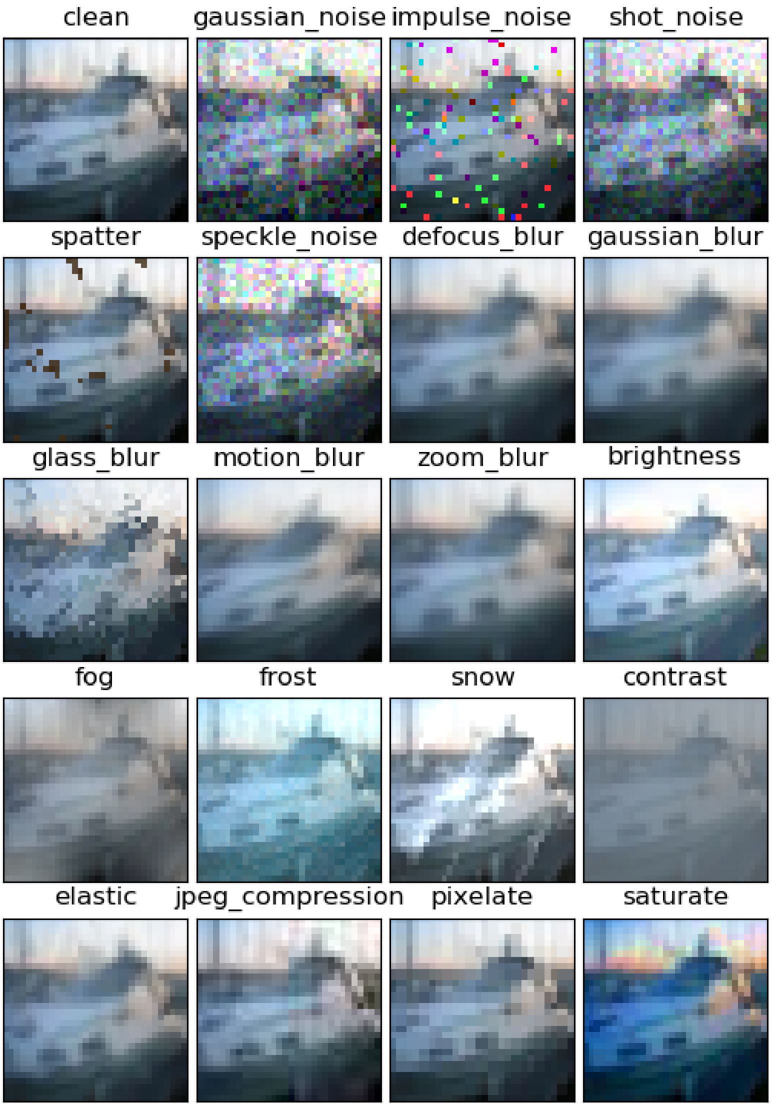
(b) Corruption severity 2

Figure 21: Examples of corrupted images from CIFAR-10-C [25]. (cont.)



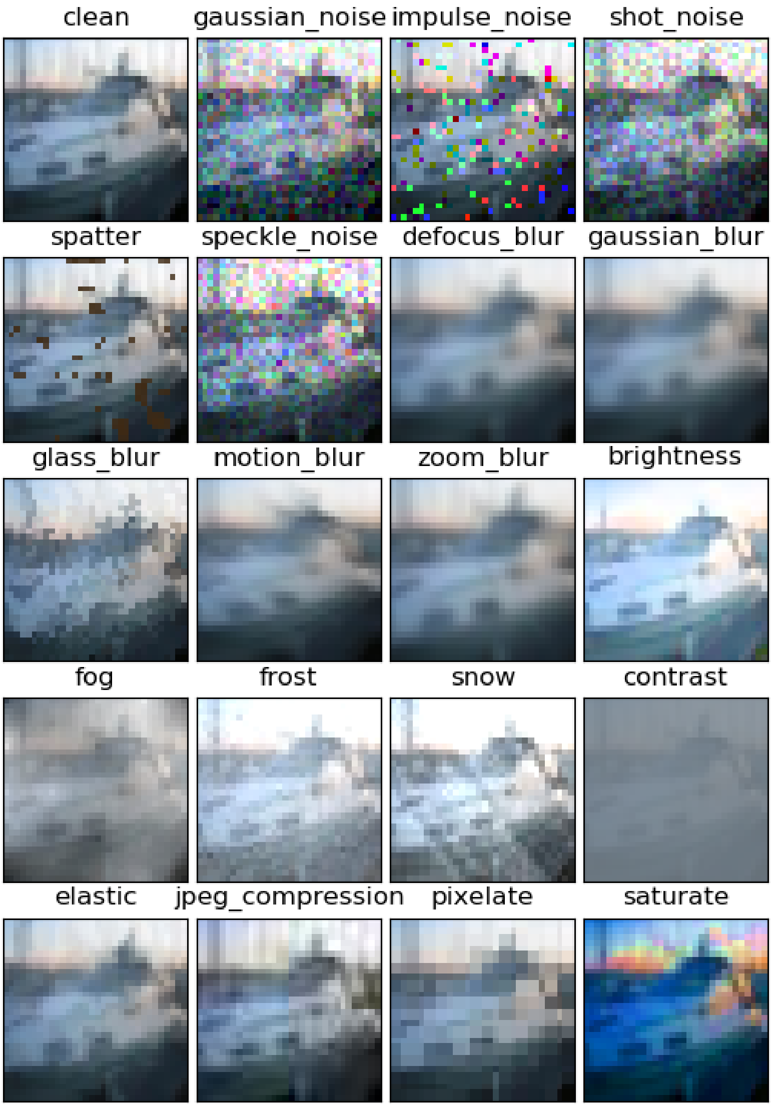
(c) Corruption severity 3

Figure 21: Examples of corrupted images from CIFAR-10-C [25]. (cont.)



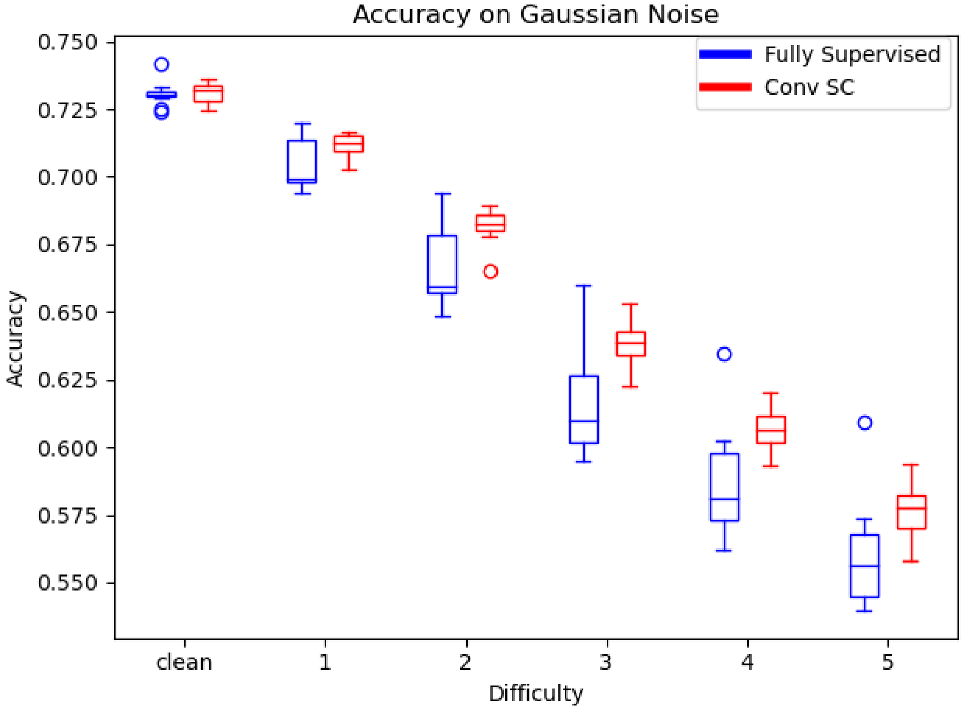
(d) Corruption severity 4

Figure 21: Examples of corrupted images from CIFAR-10-C [25]. (cont.)

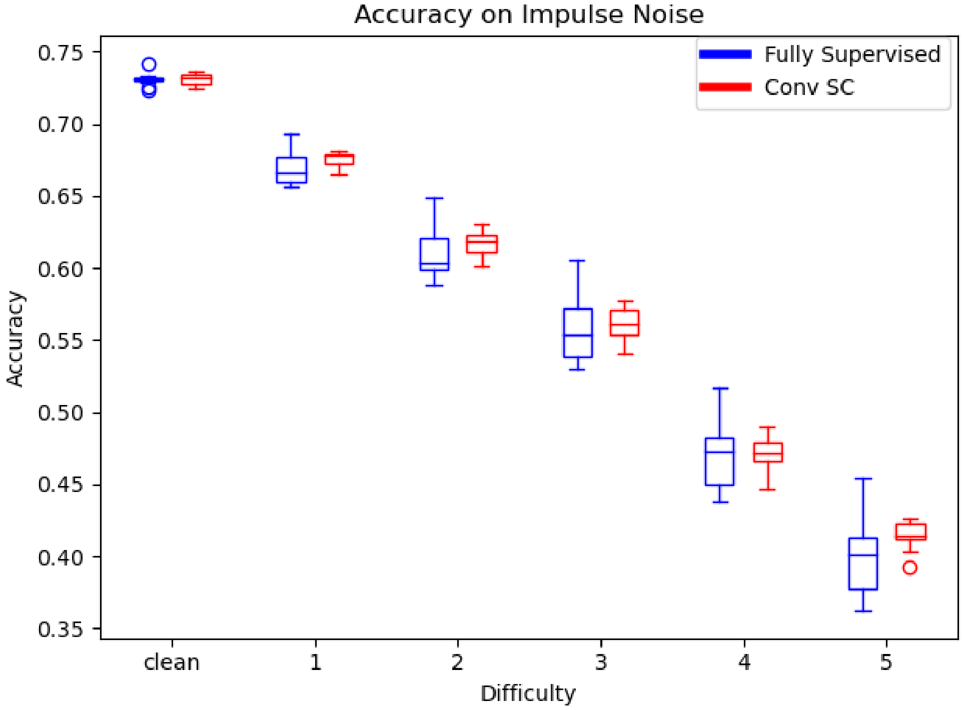


(e) Corruption severity 5

Figure 21: Examples of corrupted images from CIFAR-10-C [25]. (cont.)

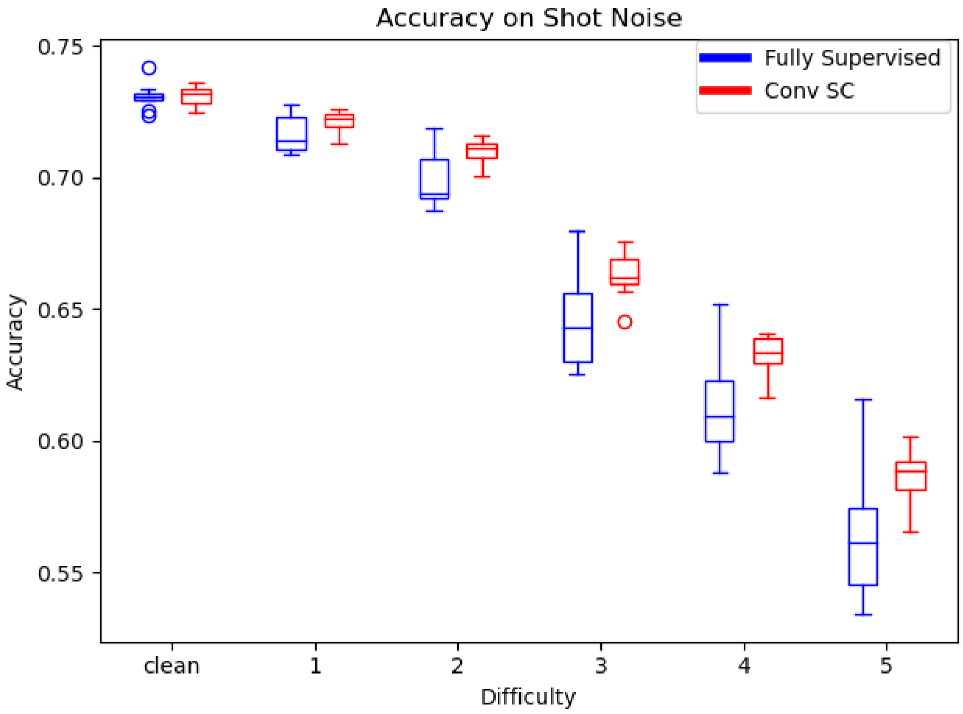


(a) Gaussian noise

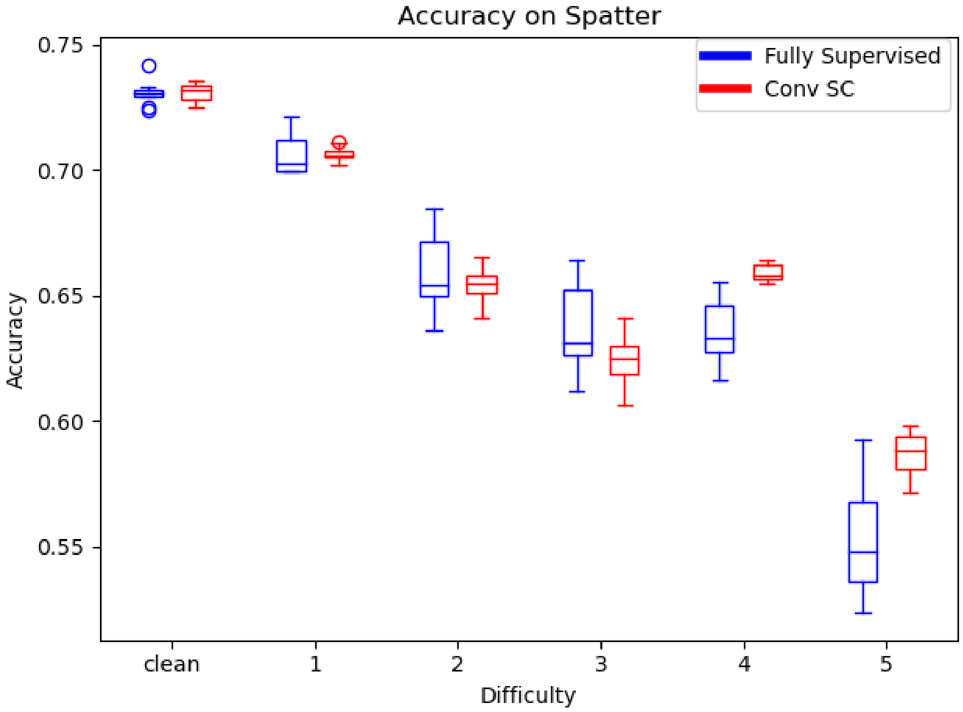


(b) Impulse noise

Figure 22: Accuracy versus corruption severity on various types of corruptions.

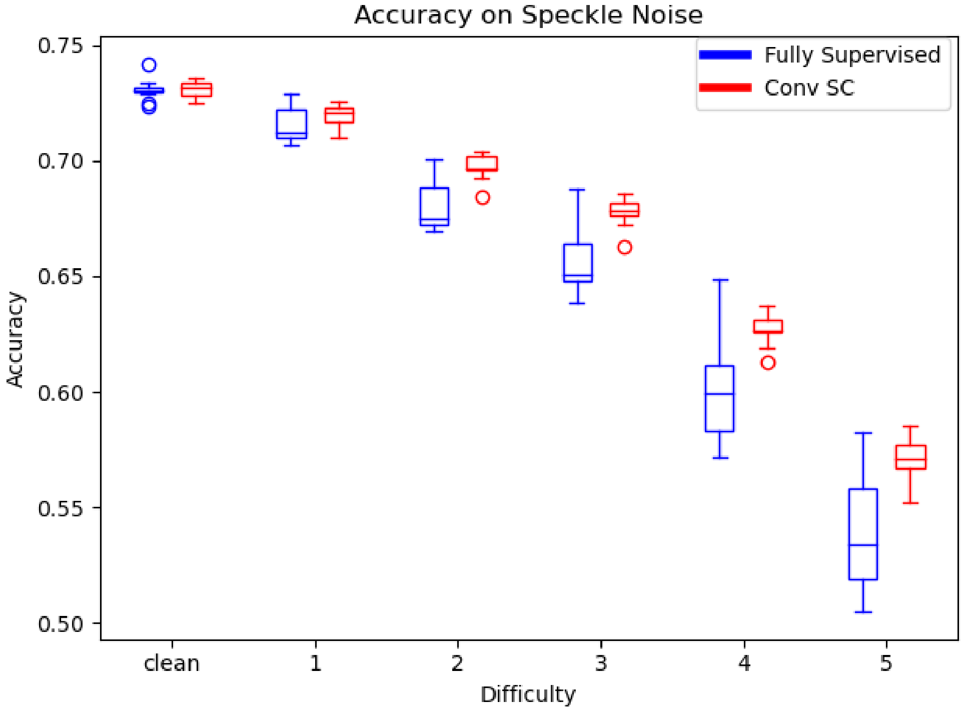


(c) Shot noise

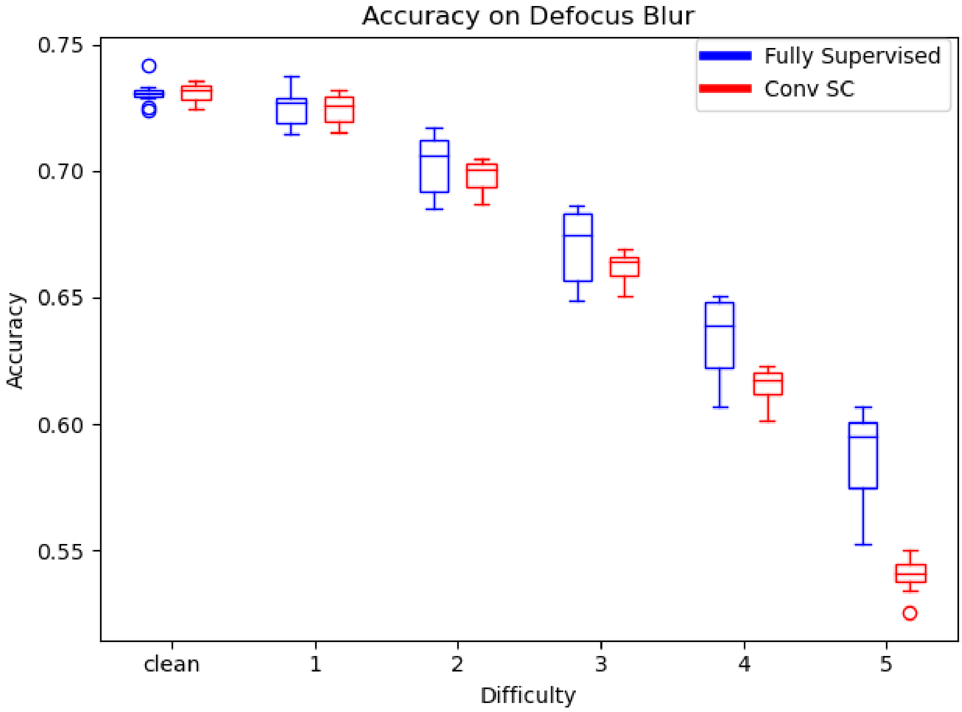


(d) Spatter

Figure 22: Accuracy versus corruption severity on various types of corruptions. (cont.)

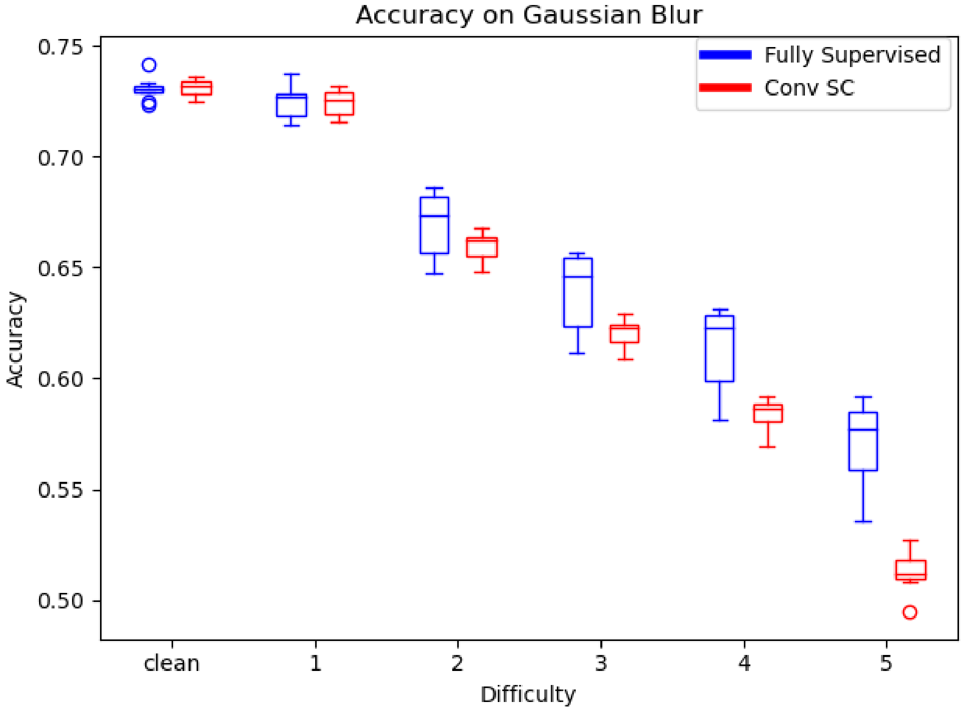


(e) Speckle noise

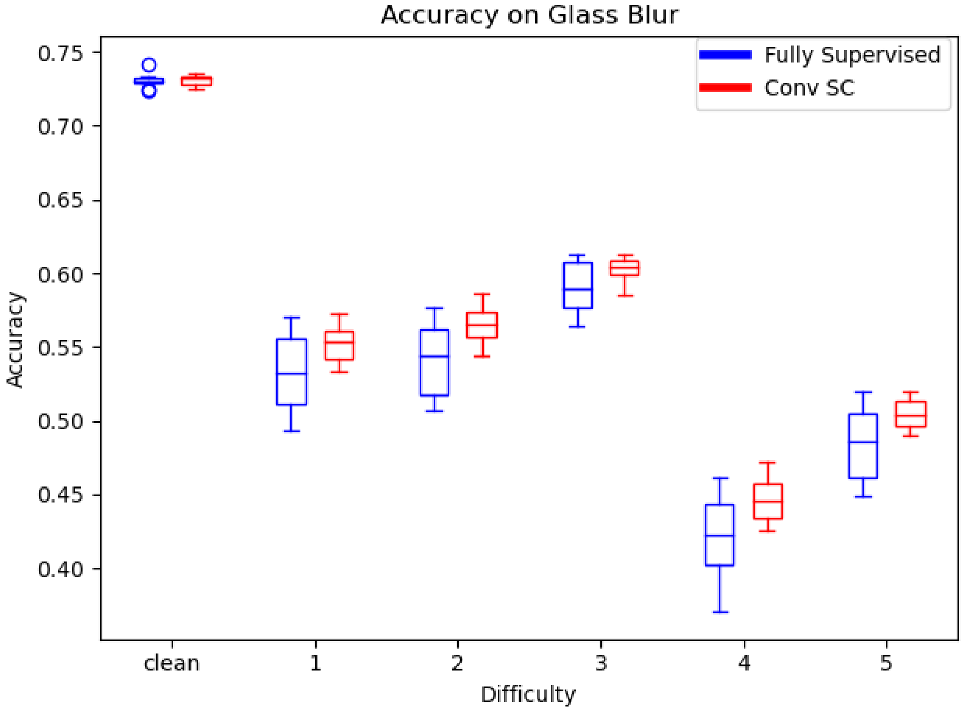


(f) Defocus blur

Figure 22: Accuracy versus corruption severity on various types of corruptions. (cont.)

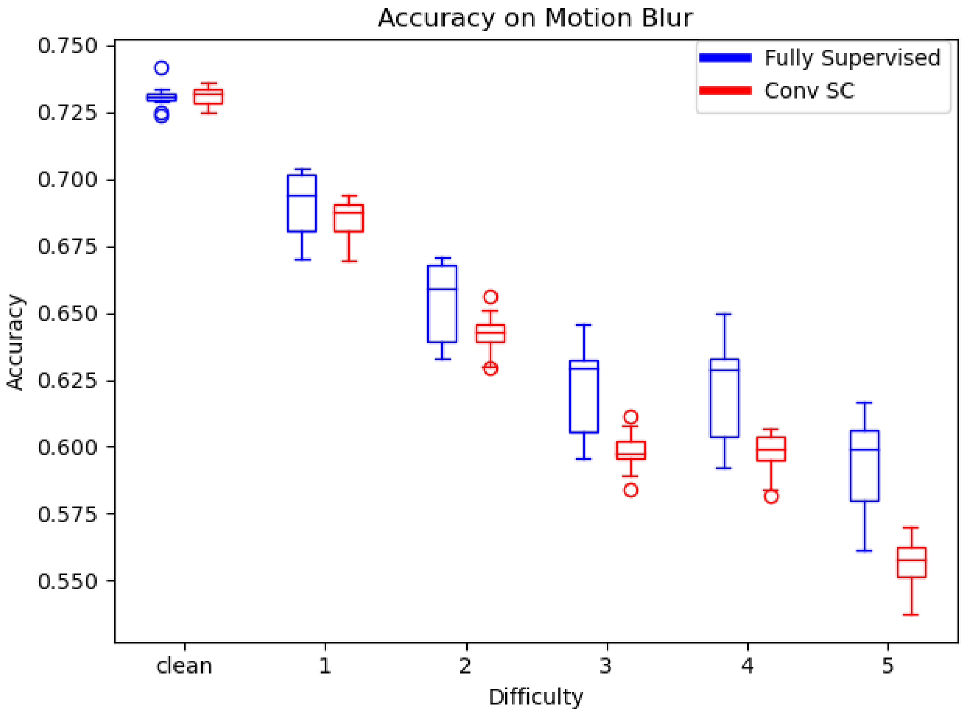


(g) Gaussian blur

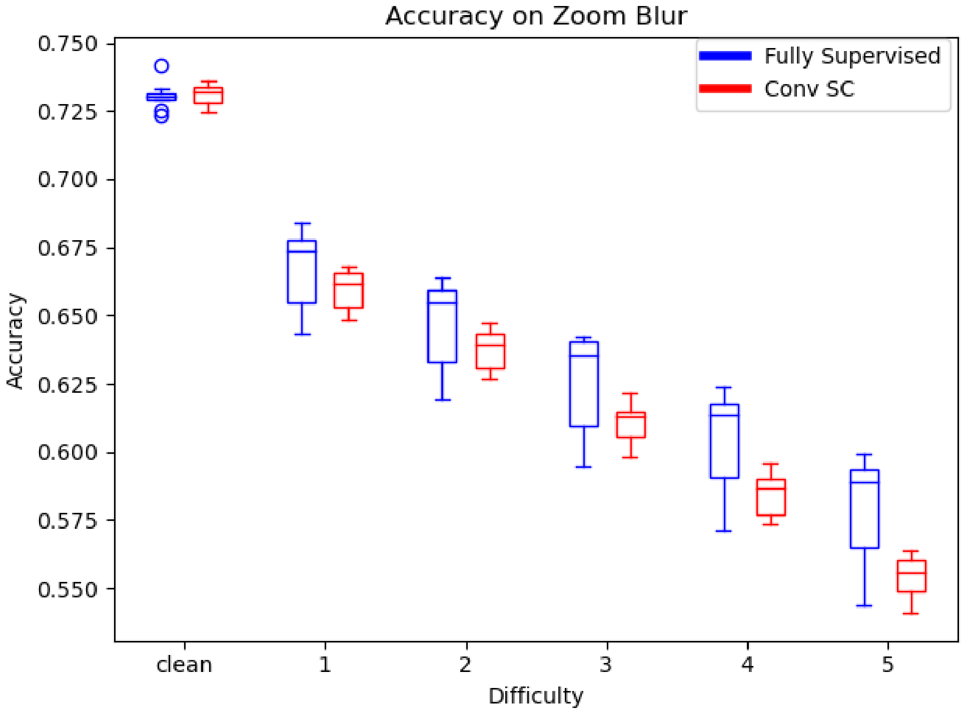


(h) Glass blur

Figure 22: Accuracy versus corruption severity on various types of corruptions. (cont.)

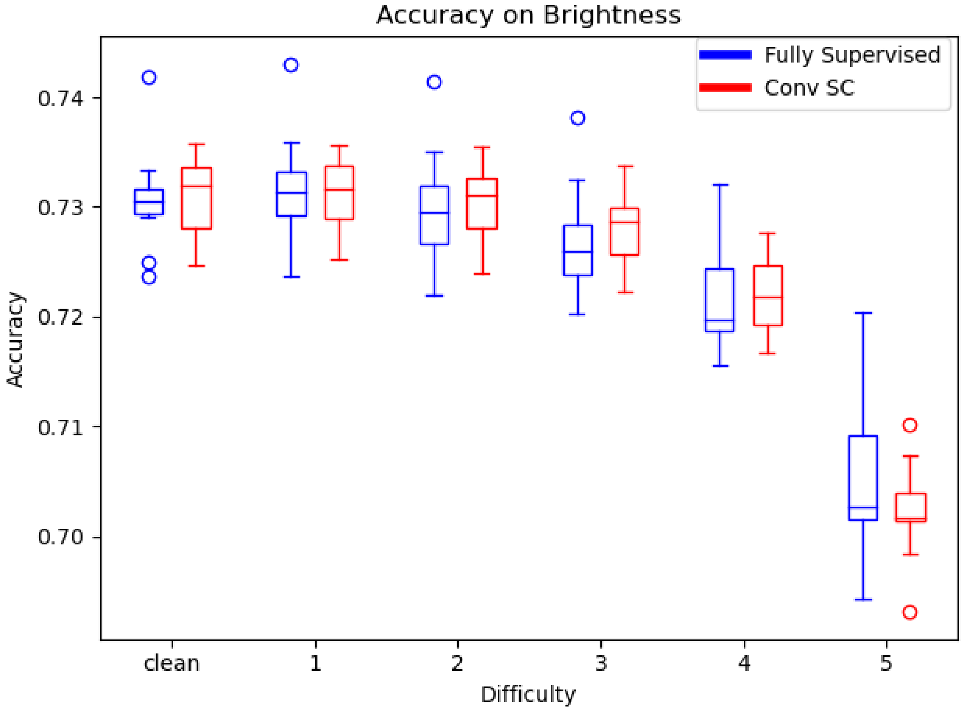


(i) Motion blur

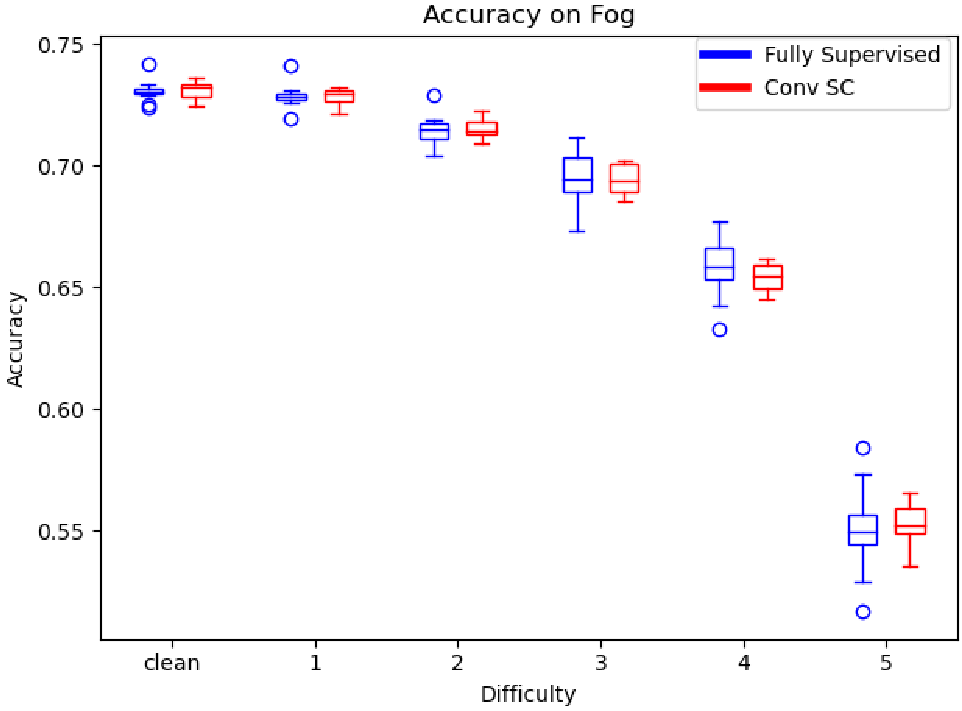


(j) Zoom blur

Figure 22: Accuracy versus corruption severity on various types of corruptions. (cont.)

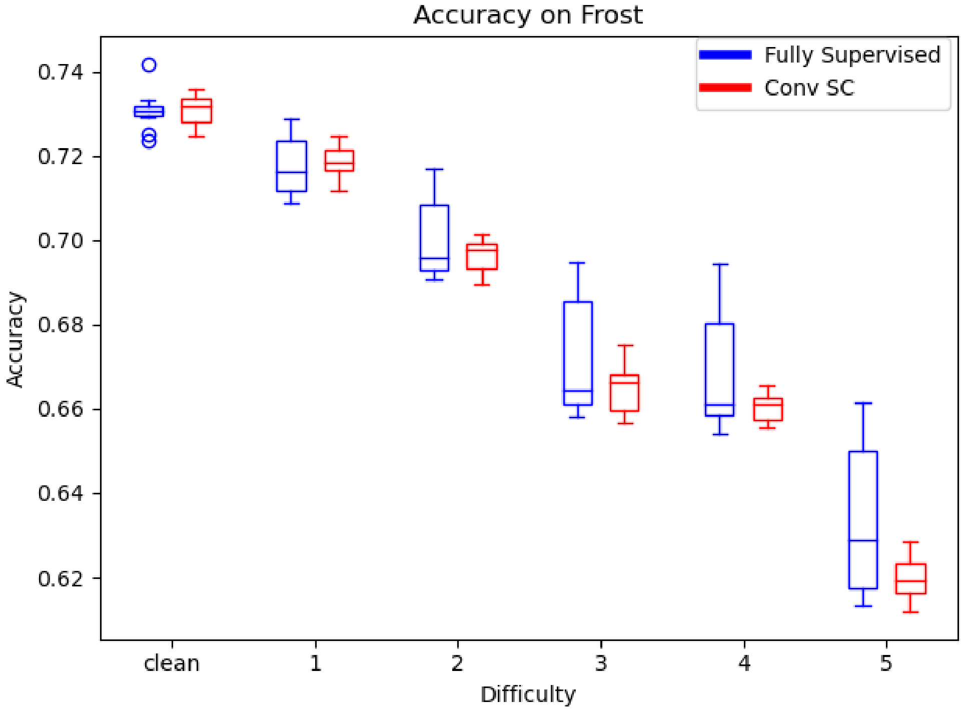


(k) Brightness

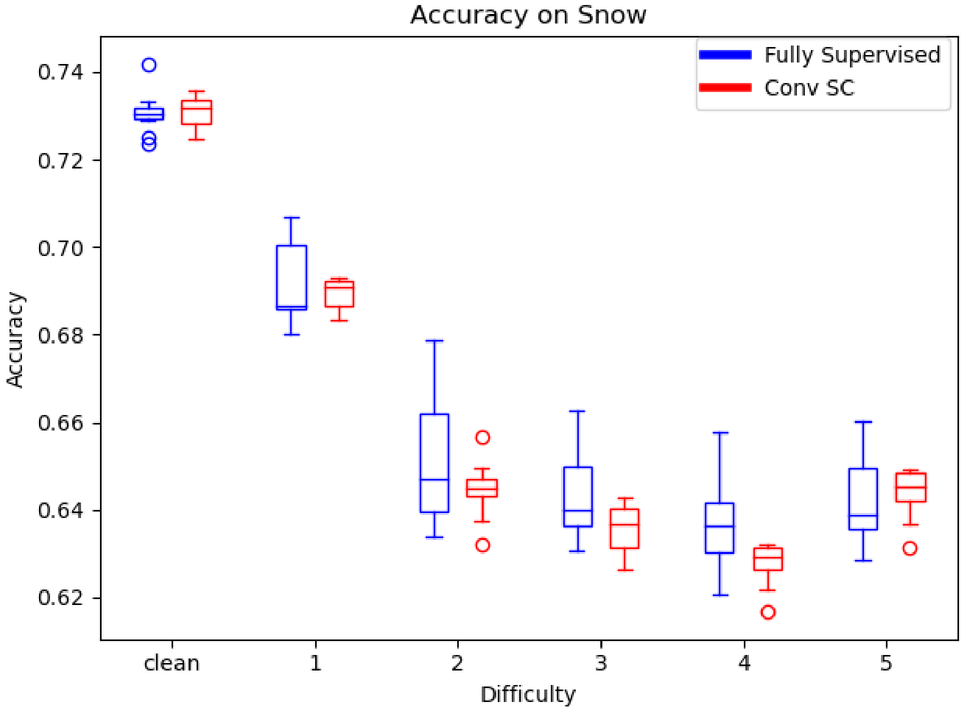


(l) Fog

Figure 22: Accuracy versus corruption severity on various types of corruptions. (cont.)

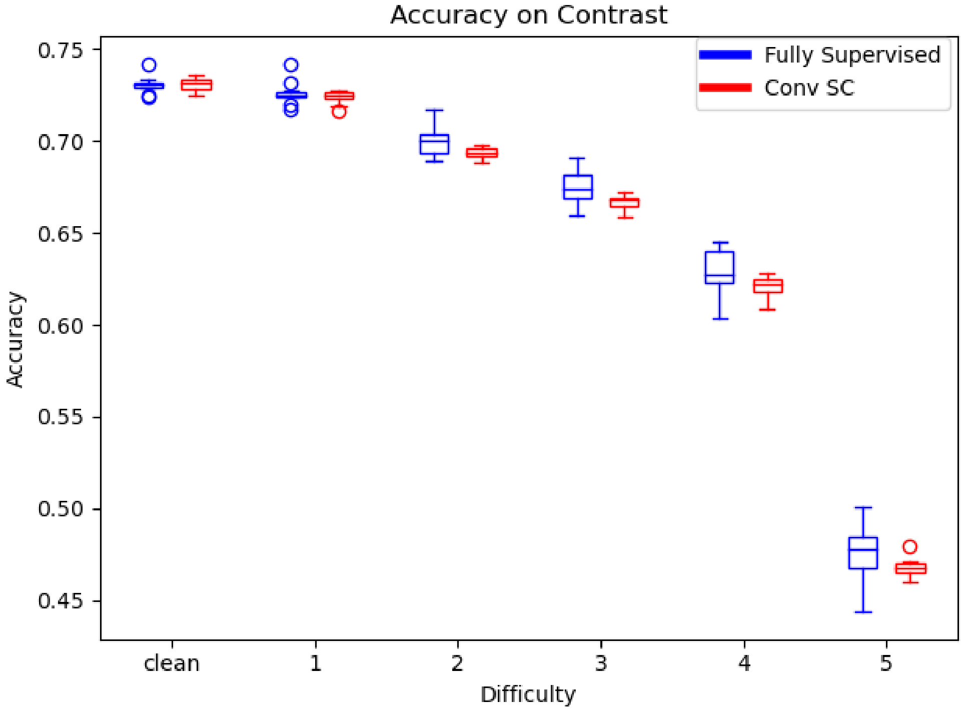


(m) Frost

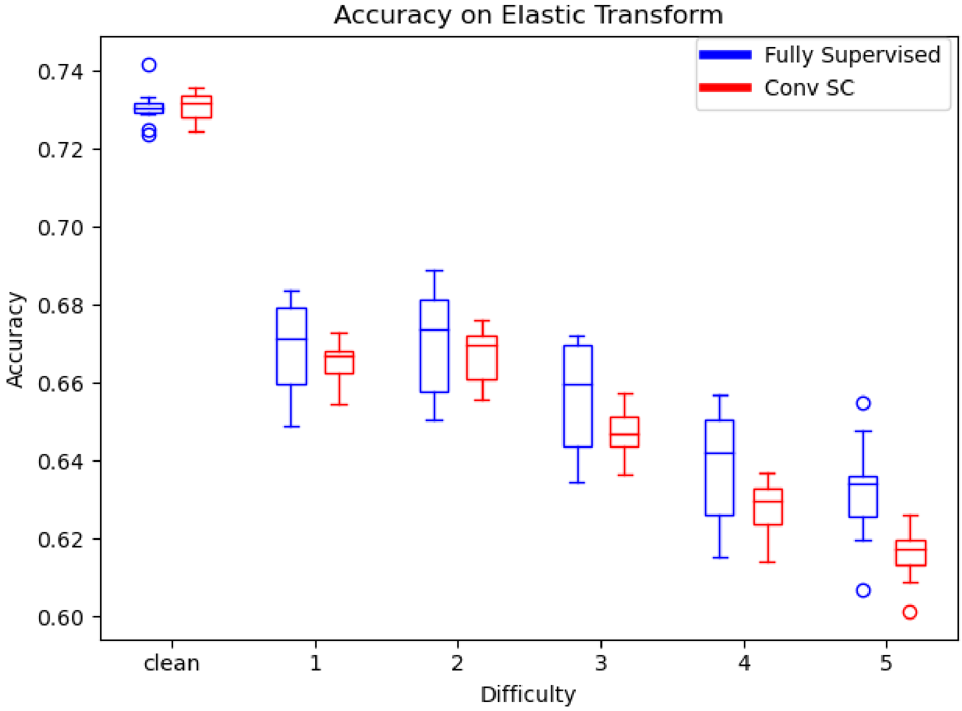


(n) Snow

Figure 22: Accuracy versus corruption severity on various types of corruptions. (cont.)

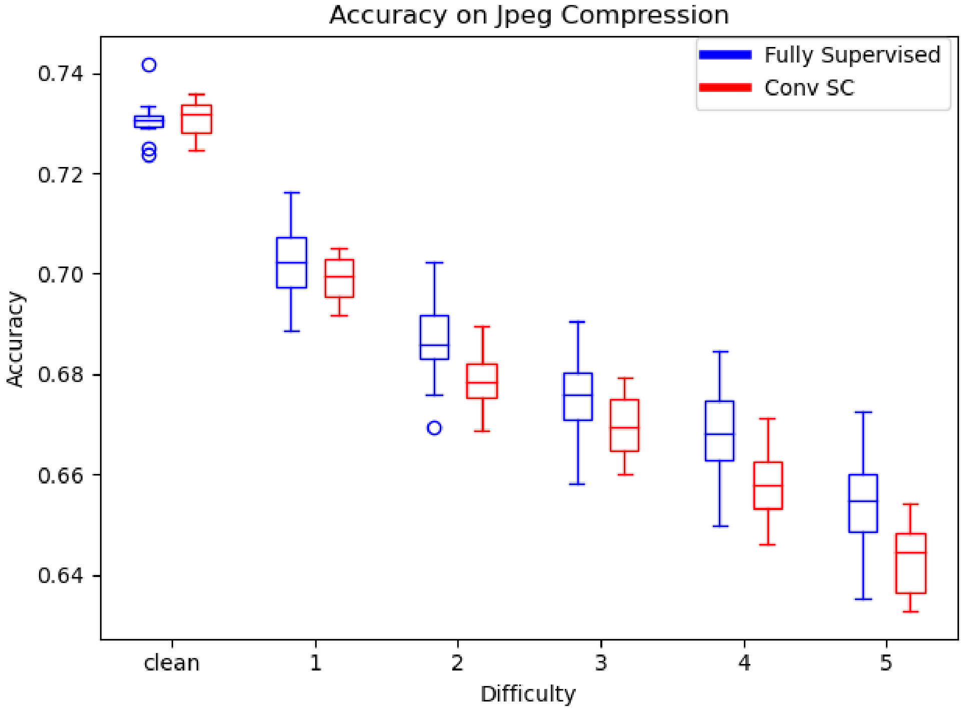


(o) Contrast

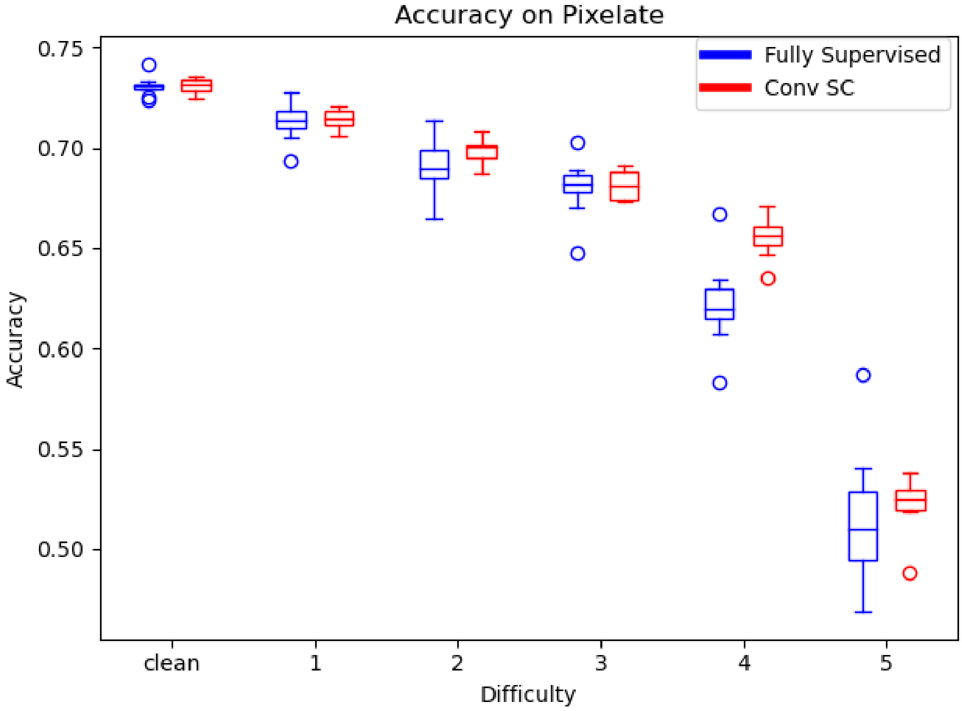


(p) Elastic transform

Figure 22: Accuracy versus corruption severity on various types of corruptions. (cont.)

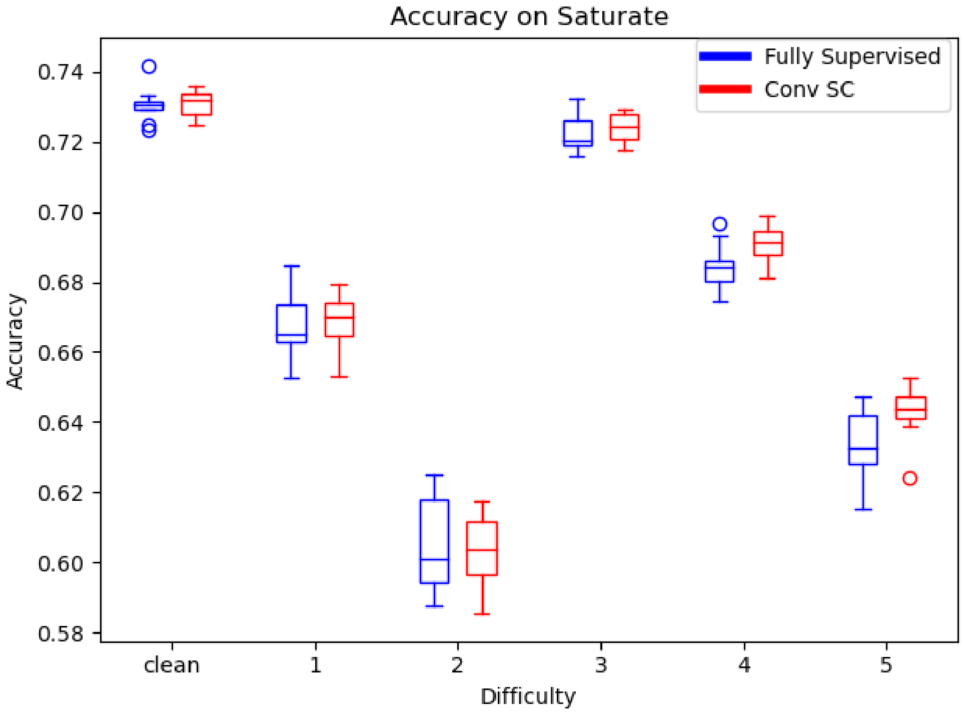


(q) JPEG compression



(r) Pixelate

Figure 22: Accuracy versus corruption severity on various types of corruptions. (cont.)



(s) Saturate

Figure 22: Accuracy versus corruption severity on various types of corruptions. (cont.)