Understanding Representations and Reducing their Redundancy in Deep Networks

Micheal Cogswell

Thesis submitted to the Faculty of

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Science and Applications

Chair Bert Huang

Co-chair Dhruv Batra

B. Aditya Prakash

February 18, 2016
Blacksburg, Virginia

Keywords: Computer Vision, Machine Learning, Object Recognition, Overfitting

Understanding Representations and Reducing their Redundancy in Deep Networks

Micheal Cogswell

Abstract

Neural networks in their modern deep learning incarnation have achieved state of the art performance on a wide variety of tasks and domains. A core intuition behind these methods is that they learn layers of features which interpolate between two domains in a series of related parts.

The first part of this thesis introduces the building blocks of neural networks for computer vision. It starts with linear models then proceeds to deep multilayer perceptrons and convolutional neural networks, presenting the core details of each. However, the introduction also focuses on intuition by visualizing concrete examples of the parts of a modern network.

The second part of this thesis investigates regularization of neural networks. Methods like dropout and others have been proposed to favor certain (empirically better) solutions over others. However, big deep neural networks still overfit very easily. This section proposes a new regularizer called `DeCov`, which leads to significantly reduced overfitting (difference between train and val performance) and greater generalization, sometimes better than dropout and other times not. The regularizer is based on the cross-covariance of hidden representations and takes advantage of the intuition that different features should try to represent different things, an intuition others have explored with similar losses. Experiments across a range of datasets and network architectures demonstrate reduced overfitting due to `DeCov` while almost always maintaining or increasing generalization performance and often improving performance over dropout.

Acknowledgement

First and foremost I'd like to thank Dr. Dhruv Batra and Dr. Devi Parikh for setting up an environment where high quality research can thrive. Dhruv allows me to explore ideas I think are most interesting without getting lost. Suspended skepticism seems like a great way to approach research.

I'd also like to thank Faruk Ahmed, Dr. Larry Zitnick, and Dr. Ross Girshick for their help and advice on the some of the work described in this thesis.

Contents

1 Introduction

Artificial intelligence has recently made great progress by applying some old methods to a wide range of tasks. Neural networks in particular have outperformed precise engineering by experts and much of that success has been related to perception tasks, especially in computer vision. These models have learned to classify objects [Kri+12a], recognize scenes [Zho+14], identify people [Zha+15; Sch+15], detect objects [Gir15], caption images [Vin+14; CZ15], answer questions about images [Ant+15], and play video games [Mni+13] among other things. This diversity, along with the fact that learned features transfer to new tasks [Don+14], suggests a general ability of these models to understand high level concepts. Further support is lent by visualizations and human studies which show some pieces of such models respond to human identifiable concepts which were not *explicitly* taught [ZF14; Zho+15].

This success is based on applying an old idea, Convolutional Neural Networks (CNNs) [LeC+98], to big data (e.g., [Rus+15]) with faster computers (Graphical Processing Units applied to general computing), though some new algorithmic improvements (e.g., dropout [Sri+14]) played an important role.

This thesis has two goals. First, it aims introduce the fundamentals of deep learning for computer vision in a way that maintains mathematical rigor while focusing on intuition. In particular, the pervasive idea of modeling something by its parts is investigated as an interpretation of how CNNs work. Second, a new loss function is proposed based on the idea that the parts a network learn to identify should not be redundant. Experiments show that this encourages networks to generalize better and overfit less. Thus, the main part of this work identifies a principle about learned representations and establishes a previously unknown connection between that principle and generalization.

2 An Introduction to Deep Learning for Vision

> *. . . It's the problem of designing a language that can communicate ideas to machines, but unfortunately we don't know what [the] ideas are, so we don't know how to do it.*
>
> — **Richard Hamming**
> Lectures from "The Art of Doing Science and Engineering" [Ham]

This chapter introduces deep learning as applied in computer vision. Often the mathematical mechanics of these models are the main focus and intuitions are secondary. In contrast, this work aims to explain intuitions while keeping enough mathematics that the intuitions can be related to precise ideas. As a result, details about representation are kept while while other details about learning are briefly skimmed. A reader without much prior information will not gain the necessary knowledge to implement a neural network; for that, see Efficient Backprop [LeC+12] (chapter), a Hacker's guide to Neural Networks [Kar14] (blog), or Neural Networks for Machine Learning [Hin] (MOOC).

2.1 Image Classification

A fundamental task in computer vision is image classification, where the goal is to label the content of an image, which is necessarily described in terms of its pixels. Pictures of cats, content that might be labeled, come in different colors, viewpoints, poses, and lighting while still being cats; they have large intra-class variance. Pictures of dogs can have the same variations as cat pictures, yet some dogs can look quite similar to cats; the latter have small inter-class variance. These problems mean that two things can be described quite differently in terms of pixels, but exactly the same in terms of high level semantics (object class). (Fig. 2.1)

The problem is hard and humans have difficulty understanding this. Though we know when we see a cat, we can't explain why we think it's a cat at the level of pixels. That makes it very hard to write a program that identifies cats from pixels. Hence it has taken decades before now to get a decent solution—and today's systems still have constraints and mistakes.

Fig. 2.1: The goal is to find a function that classifies all of these images correctly. The rightmost images are of different classes, but have a lot in common, while the leftmost images are both cats, but have less in common. (These images are from ImageNet [Rus+15]
.)

However, research has found that Convolutional Neural Networks (CNNs) are quite effective for this type of problem.

To be more precise, the data of interest is $\mathcal{D} = \{(\boldsymbol{x}^1, \boldsymbol{y}^1), \ldots, (\boldsymbol{x}^n, \boldsymbol{y}^n), \ldots, (\boldsymbol{x}^N, \boldsymbol{y}^N)\}$ where $\boldsymbol{x}^n \in \mathbb{R}^d$ and $\boldsymbol{y}^n \in [0,1]^C$. Images are thought of as vectors in $\mathbb{R}^d$ by concatenating all of their pixels. Sometimes $\boldsymbol{x}$ is used when the particular example is not important. Each input $\boldsymbol{x}^n$ has d dimensions and there are C possible labels. Each example contains the pixels of an image $\boldsymbol{x}^n$ and a label $\boldsymbol{y}^n$ such that $y_c^n = 1$ if $\boldsymbol{x}^n$ contains class $c \in \{1, \ldots, C\}$ (*e.g.*, the image contains a cat), or $y_c^n = 0$ if not. The goal is to find a function $f : \mathbb{R}^d \mapsto [0,1]^C$ that assigns high probability to the category (or categories) contained within an image and low probability to the other categories. We want a function f such that $f(\boldsymbol{x}^n) = \boldsymbol{y}^n$.

2.2 Linear Models

2.2.1 Functional Form

The first step is to consider a small subset of all possible classifiers where each of the possible fs will be completely specified by a matrix of real numbers $W \in \mathbb{R}^{C \times d}$ using [1]

$$f(\boldsymbol{x}, W) = \phi(W\boldsymbol{x}) \tag{2.1}$$

where the softmax $\phi : \mathbb{R}^C \mapsto \mathbb{R}^C$ maps scores $\boldsymbol{z} \in \mathbb{R}^C$ to probability vectors such that

$$\phi(\boldsymbol{z})_c = \frac{e^{z_c}}{\sum_s e^{z_s}}. \tag{2.2}$$

[1] This chapter ignores biases ($W\boldsymbol{x} + \boldsymbol{b}$) for simplicity as they can be integrated fairly easily. (To see this, suppose a 1 is appended at the end of every input vector, increasing its dimension by 1.)

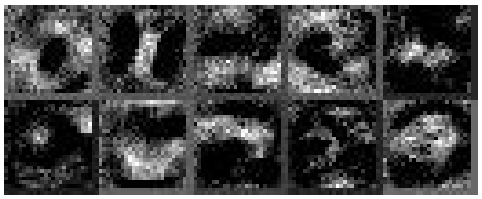
The cth component of $W\boldsymbol{x}$ is this model's score for the cth class. This simple scoring function—a linear transformation of inputs by a weight matrix—is the hallmark of a linear model.

For example, take a relatively simple problem: MNIST classification [LeC+98]. Given a handwritten number in the form of a small, centered, black and white image (Fig. 2.2a), the goal is to predict the digit it represents (0 through 9).

Though we haven't yet talked about how to find a good W, let's look at one such weight matrix $\hat{W} \in \mathbb{R}^{10\times 784}$ ($784 = 28 \cdot 28$) to get a sense of how it works. Each row of $\hat{W}$ corresponds to one digit. Taking the dot product of the 0th row of $\hat{W}$ and the pixels of a digit gives the score that digit gets for the 0 class. By shaping these rows into images and normalizing [2] we can visualize the weights. This has been done in figure 2.2b. The top left digit is the 0 weight vector, to its right is the 1 weight vector, .etc, with the 5 weight vector visualized in the bottom left.

In this visualization, pixels that should be white in the input are white in the filter visualizations because that would help produce the highest score. Similarly, those that should be black are black, and those that aren't relevant for telling a particular class from the others are gray. The top left image is mostly white where 0 pixels would be and black/gray elsewhere. The digit 1 is also pretty easy to make out. Other digits, like 4 and 5, have parts missing, but can still be made out. In this case, a linear combination of pixels is a reasonable scoring function because intra-class variation is small (7s all look the same) and inter-class variation is relatively large (7s are different enough from 5s). Looking at the digit 5 weights shows 3 main pieces that help discriminate 5s from other digits and some missing bits that have too much in common with other digits to be discriminative. Fortunately, 10 classes are few enough that some parts can be discriminative.

(a)Example MNIST Digits



(b)$\hat{W}$ that performs relatively well (about 80% accuracy on held out data). Weights are directly visualized by re-shaping a row of the weight matrix into an image and normalizing pixel intensities to a fixed range.

Fig. 2.2: Visualization of a linear classifier for MNIST.

Now consider harder consider the harder cat/dog problem from (Fig. 2.1). A dog template as in figure 2.2b would either be too specific and match only a few dogs or it would be too general and also match cats. Because cats and dogs can have high intra-class variance and

[2] Black pixels are the lowest value (negative) weights and white values are the highest value (positive) weights.

low inter-class variance no single template will work. However, linear templates are still quite useful. Instead of describing cats and dogs with one template each, we will describe them with in terms of some parts where the parts have low intra-class variance and high iter-class variance, allowing them to be easily detected by linear templates. To build these parts we will end up using a hierarchy of linear templates that first find low level parts then use those parts to build higher level parts. However, first we need to figure out how to choose W in the simple linear case explored so far.

2.2.2 What about learning W?

There are two nagging questions which should be addressed briefly:

1. Is a given W good or not?

2. How is a good W found?

Is a given W good or not?

The idea that good values of W should encourage large scores for the correct class is key. We'll formally express this intuition as a loss function that assigns high values to bad weights and low values to good ones. This is the key point of communication between human and machine. A principle insight from machine learning and statistics is that this lack of understanding can be partially made up for using data. This section just describes a common loss, but next chapter will modify this loss to increase performance.

Given a candidate W, predict a probability vector $\hat{\boldsymbol{y}}^n = f(\boldsymbol{x}^n)$ for every example in $\mathcal{D}$ then compute how good the prediction is on average. The idea is to penalize W if it's confident and wrong, reward it if it's confident and right, and penalize it only a bit if it's not so confident and wrong. This is the intuition captured by cross entropy:

$$\mathcal{L}(\mathcal{D}, W) = \sum_n \sum_c -y_c^n \log(\hat{y}_c^n) \tag{2.3}$$

Remember that $\boldsymbol{y}^n$ is all 0s except for one 1 at the index of the correct class. For example n, call that class $\bar{c}^n = \text{argmax}_c\, y_c^n$. Then the expected cross entropy between the true predictor and the predictor given by W can be estimated as

$$\begin{aligned}
\mathcal{L}(\mathcal{D}, W) &= \sum_n -y_{\bar{c}^n}^n \log(\hat{y}_{\bar{c}^n}^n) && (2.4)\\
&= \sum_n -y_{\bar{c}^n}^n \log(\phi(W\boldsymbol{x})_{\bar{c}^n}) && (2.5)\\
&= \sum_n -\log(\phi(W\boldsymbol{x})_{\bar{c}^n}). && (2.6)
\end{aligned}$$

This last equation (2.6) shows that the loss (in the setup so far) only considers the class of the correct example, which is simply penalized heavily for a lack of confidence (predicting probabilities farther from 1.0). The loss corresponds to maximizing the data's likelihood and completes the idea of a multinomial logistic regressor, which is the name of this particular linear classifier.

How is a good W found?

Typically first order convex optimization algorithms—often Stochastic Gradient Descent (SGD)—are used to search for a W that minimizes the loss. This chapter will not do anything different. By first picking some random initial weight matrix W_0 and repeatedly perturbing it in the direction that most decreases the loss in a local ϵ neighborhood ($\frac{\partial \mathcal{L}(\mathcal{D},W)}{\partial W}$) we can arrive at a final weight matrix that scores examples from $\mathcal{D}$ best in some local neighborhood around W_0. Note that this requires computing $\frac{\partial \mathcal{L}(\mathcal{D},W)}{\partial W}$, which can be a gradient or a subgradient. Of course, this optimization is called learning.

This chapter will not mention more, except to say that convex optimization algorithms like SGD are typically guaranteed to find the globally optimal weight matrix for linear models (because their loss surfaces are usually convex), but not for more interesting models like the deep ones in the next section.

One final distinction is crucial. Good performance on the examples in $\mathcal{D}$ is not enough. Rather, performance on examples that come from the same distribution as the ones in $\mathcal{D}$ is much more interesting. [3] A practical way to do this is to split $\mathcal{D}$ into disjoint training and testing sets $\mathcal{D}_{train}$ and $\mathcal{D}_{test}$, then train algorithms only with $\mathcal{D}_{train}$ and test only with $\mathcal{D}_{test}$.

2.3 Deeper Models

2.3.1 Multilayer Perceptrons

Most deep models are neural networks formed by composing linear models separated by non-linear activation functions. For example, consider a 2 layer model. The first layer predicts a vector of d_1 hidden activations using weight matrix $W_1 \in \mathbb{R}^{d_1 \times d_0}$ ($d_0 = d$ is the input dimensionality):

$$\boldsymbol{h} = \sigma(W_1 \boldsymbol{x}) \tag{2.7}$$

[3] Assume each example is sampled from a random variable which is independent from and distributed identically to the others.

where σ is the sigmoid function applied element-wise to map $z \in \mathbb{R}$ into the interval $[0, 1]$ by

$$\sigma(z) = \frac{1}{1 + e^{-z}}. \tag{2.8}$$

Note that $\boldsymbol{h} \in \mathbb{R}^{d_1}$. Instead of supervising these activations directly, they are used as input to the next layer

$$\hat{\boldsymbol{y}} = \phi(W_2 \boldsymbol{h}) \tag{2.9}$$

$$= \phi(W_2 \sigma(W_1 \boldsymbol{x})). \tag{2.10}$$

This is a good point to introduce some nomenclature. When talking about neural networks like this one there are a number of names for the entries hidden vectors such as $\boldsymbol{h}$. An entry h_j might be called an "activation", "hidden activation", "unit", "hidden unit", "feature", or "neuron"; when talking about Convolutional Neural Networks (CNNs) the analog is called a "feature map". In any case, a vector of hidden activations just represents some scores as introduced in linear models. The "hidden" part only indicates there is no expected intuition for the scores because they don't correspond to an input or an output. Furthermore, an Multilayer Perceptron (MLP) that involves an unconstrained linear transformation (*e.g.*, the two layers here) is said to be "fully connected".

A 2 layer MLP like this one is typically supervised by the same cross entropy loss defined before, except defined over the set of all weights using the notational convenience $W = \{W_1, W_2\}$:

$$\mathcal{L}(\mathcal{D}, W) = \sum_n -\log(\hat{y}^n_{\tilde{c}^n}). \tag{2.11}$$

In general, L layer models can be defined. Let the input $\boldsymbol{x}$ have $d_0 = d$ dimensions for notational convenience. A particular layer $l \in \{0, 1, \ldots, L\}$ computes d_l hidden activations using a learned linear transformation $W_l \in \mathbb{R}^{d_l \times d_{l-1}}$ followed by a non-linearity. Rather than be "hidden", layer 0 is fixed to the input ($\boldsymbol{h}_0 = \boldsymbol{x}$). Activations are computed during forward propagation, which starts at layer 1 and computes each vector of hidden activations from the previous one:

$$\boldsymbol{h}_l = \sigma(W_l \boldsymbol{h}_{l-1}) \tag{2.12}$$

$$= \sigma(W_l \sigma(W_{l-1} \sigma(...\sigma(W_1 \boldsymbol{x})))). \tag{2.13}$$

for each $l \in \{1, \ldots, L-1\}$. Note that each layer *is* a linear model (2.12): inputs are scored using by multiplying them with a weight matrix then the outputs are re-scaled by some non-linearity. Finally, a logistic regressor is placed on top of this apparatus

$$f(\boldsymbol{x}) = \hat{\boldsymbol{y}} = \boldsymbol{h}_L = \phi(W_L \boldsymbol{h}_{L-1}). \tag{2.14}$$

At first it may look hard to compute gradients of the the loss $\mathcal{L}(\mathcal{D}, W)$ w.r.t. the weights W, but there's an efficient algorithm for doing so called backpropagation discovered in various forms [Sch15], but most prominently by Rumelhart *et al.* [Rum+88]. The key insight is that gradients can be computed using chain rule (convince yourself by staring at equation (2.10) then equation (2.13)), and that intermediate results can be cached as in dynamic programming. Since gradients can be computed, the model can be optimized with Stochastic Gradient Descent, as was the case for the linear model. As evidenced by the successes mentioned in the introduction, local optimization works well in practice despite the non-convexity presented by more than 1 layer [Dau+14; Cho+15].

2.3.2 Modern Networks for Images

There are two important changes to the neural networks of last century which bring us to today's networks: learned convolutional layers and rectified linear activation functions.

Convolutions

In computer vision a very restricted version of an MLP is used for efficiency and regularization. Suppose a 2 layer MLP as in the last section where the input is a relatively small 256x256 image (65536 pixels). The first layer, which has d_1 units, would require $65536 \cdot d_1$ entries in the weight matrix. With even 20 units this quickly involves millions of weights and a costly matrix multiplication.

An efficient alternative to this naïve approach is to compute new images called feature maps instead of hidden activations or neurons. Similar to an input images, which has one grid of pixels for each input channel (*e.g.*, red, green, blue), a feature map is a grid of pixels where high intensity pixels indicate the presence of a certain template that will be learned. An example is shown at the top of figure 2.3. As the next section shows, feature maps can learn to activate highly for more complex concepts than color.

For this to be efficient requires a convolutional layer, an idea that alters this linear transformation of all pixels in an image by making two reasonable assumptions. First, each pixel in an output feature map can be described by a small corresponding patch centered around it in the input feature maps (locality). Second, patches at different locations can be described

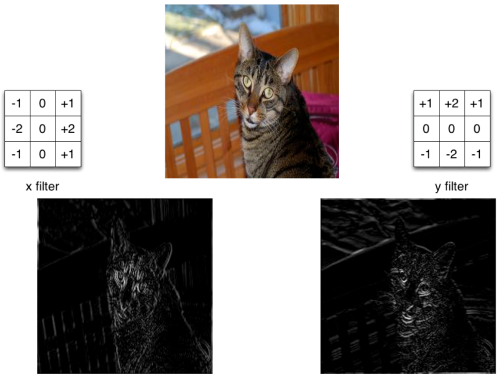


Fig. 2.3: Two feature maps computed from one of the example cat images. The feature maps are on the bottom left and right and are computed by convolving the top cat image with the filters on the top left and right. (Again, the image is from ImageNet [Rus+15].)

in the same way (stationarity of statistics). A convolutional layer computes *each pixel* of a feature map using a small patch below it.

To describe convolutions mathematically, it's convenient to use a different notation for images. Instead of indexing all pixels as one big vector ($\boldsymbol{x}$), an image $I \in \mathbb{R}^{C \times W \times H}$ is a matrix with height H, width W, and C channels. Thinking of the first layer, instead of a weight matrix $W \in \mathbb{R}^{d_1 \times d_0}$, the learned weights will be small kxk (*e.g.*, 3x3) kernel images $K \in \mathbb{R}^{d_1 \times d_0 \times k \times k}$. That is, K is a tensor containing $d_1 \cdot d_0$ such images, each the analog of one entry in the first weight matrix W_1. Denote the image patch/submatrix for channel c between columns i and j and rows p and q by $I_{c,i:j,p:q}$. Furthermore, let v be a vectorizor function that flattens matrices and tensors into pixels (*e.g.*, $v(I) = \boldsymbol{x}$).

Each pixel of the tth feature map $M_t \in \mathbb{R}^{h \times w}$ computed from input I is produced by a linear combination of the pixels in the patch below:

$$M_{t,i,j} = \sigma\left(\sum_{c=1}^{C} v(I_{c,i:j,p:q})^T v(K_{t,c}^T)\right) \tag{2.15}$$

$$= \sigma\left(\sum_{c=1}^{C}\sum_{h=1}^{k}\sum_{w=1}^{k} I_{c,i-h,j-w} K_{t,c,h,w}\right). \tag{2.16}$$

In a more standard form, this can be expressed with the convolution operator $*$:

$$M_t = \sigma\left(\sum_{c=1}^{C} I_c * K_{t,c}\right). \tag{2.17}$$

Note that this is still a linear operation (this can also be seen using Toeplitz matrices), so it can be thought of using the same intuition as for linear models. Per equation 2.15, each pixel in a feature map is scored by a linear transformation. In fact, some efficient imple-

mentations [Che+14a] of convolutional layers are implemented as a call to the Generalized Matrix Multiply routine.

ReLUs

A popular choice of non-linearity in modern networks, and the one use here, is the Rectified Linear Unit (ReLU) [NH10]

$$\sigma(z) = \max(0, z) \tag{2.18}$$

This activation function is much faster to compute than sigmoids and helps avoid the problem of saturated activations, where gradients corresponding to very large activations become small due the non-linearity. ReLUs make training deeper networks a faster, less finicky endeavor.

Other Differences

The rest of the chapter talks specifically about AlexNet [Kri+12a] as implemented and released in the Caffe framework [Jia+14]. That particular network has pooling and local response normalization layers, and there are many other architectures which deviate from the basic formula laid out here. Such extensions can clearly improve the framework, but experiments as in [Spr+14] show that these basic building blocks are the core of the method and give competitive performance on their own.

2.3.3 Visualizations

Next section describes an intuition behind deep models with concrete examples taken from a real network, as for linear models with MNIST. To prepare for that, this section describes how these examples are visualized.

Visualizations like the one in figure 2.4 correspond to a particular activation in a neural network (*e.g.*, the jth component of $\boldsymbol{h}_l$, or the tth feature map of M). The left shows some patches of images while something akin to the corresponding gradient patches are on the right. Across the ImageNet classification challenge validation dataset [Rus+14] (50,000 images), these patches produce highest values for that activation. The unit detects concepts which are represented best by these patches. Correlating these patches visually shows that this neuron detects vertical edges with blue on the left and orange/yellow/red on the right. By looking at the "gradients" on the right allows one to see that adding more blue to the left of any patch or orange to the right will increase the activation's response to the patch because those gradient patches are blue on the left and orange on the right.

Actually, the only difference between the "gradient" patches on the right and the actual gradient of the outputs with repect of the input ($\frac{\partial \mathcal{L}(\mathcal{D},W)}{\partial I}$) is that some gradients are set to 0 where they otherwise might not have been. Normally the ReLU subgradient is

$$\frac{\partial \mathcal{L}}{\partial z} = \frac{\partial \mathcal{L}}{\partial h}\frac{\partial h}{\partial z} = \frac{\partial \mathcal{L}}{\partial h}[\![z \geq 0]\!] \tag{2.19}$$

where $[\![\cdot]\!]$ is the indicator function. However, these functions replace $\frac{\partial \mathcal{L}}{\partial z}$ with

$$\frac{\partial \mathcal{L}}{\partial h}[\![z \geq 0]\!][\![\frac{\partial \mathcal{L}}{\partial h} \geq 0]\!], \tag{2.20}$$

which is no longer a subgradient because $[\![z \geq 0]\!][\![\frac{\partial \mathcal{L}}{\partial h} \geq 0]\!]$ is not in the subdifferential set $\frac{\partial h}{\partial z}$.

Other than that, these visualizations compute the Jacobian w.r.t. the image. $\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}}$ and they are thought of as gradients. The only reason to use these images instead of the actual gradient images is that the actual gradients are less interpretable. See [Spr+14], derived from [ZF14], for more detail. Visualization in this manner provides interpretability to intermediate layers of a model by showing what a neuron is interested in (image patches) with pixel precision (gradient patches).

2.3.4 Parts of Parts

Natural images like the cat in the first example (Fig. 2.1; reproduced in 2.5) have variations which can't be so easily captured. Even though they're in different classes, the cat and the dog have many more pixels in common than the two cats, so a simple filter (linear combination of pixels) can't hope to classify these examples. Imagine a cat filter like the "1" filter in (Fig. 2.2b).

It would be useful if elements of $\boldsymbol{x}$ represented cat parts instead of pixels, which are cat parts in the same sense that the atoms are car parts. It's much more useful to describe the thing you drive to work every day in terms of its wheels, frame, and engine than it is to precisely describe all the atoms that make up the car. Similarly, it's more useful to describe a cat

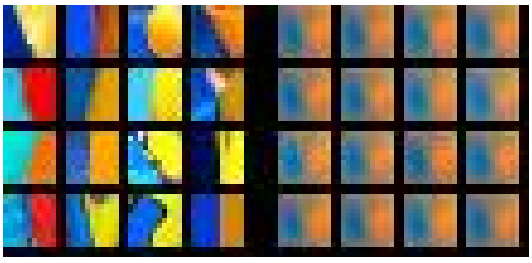


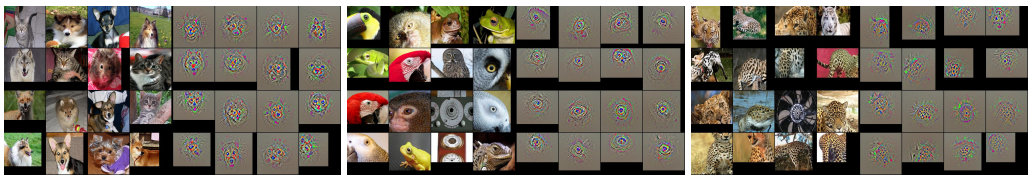
Fig. 2.4: An example of the visualizations used in section 2.3.4. Representative images patches are on the left and corresponding "gradient" patches that indicate important parts of the patches are on the right.

picture in terms of eyes, nose, and fur rather than pixels. The following visualizations inspect a deep CNN and show that the layers detect some of the mentioned parts, so they capture increasing levels of abstraction. These layers of abstraction are automatically *learned* by a deep CNN.

Fig. 2.5: Intra-class variance can be smaller than inter-class variance. (replication of 2.1)

The fact remains: computers that recognize cats must do so by relating them to pixels.

Convolutional Neural Networks (CNNs) automatically represent these parts. By focusing on of activations from the leftmost cat in Fig. 2.5 and tracing highly activated neurons through the network this can be seen. At first, abstract concepts that humans can understand appear—*e.g.*, the 3 neurons from the 5th convolutional layer in figure 2.6—but eventually these concepts translate into pixels one layer at a time.

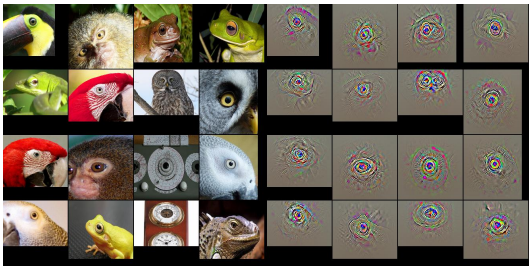


(a) Cat Faces **(b)**Eyes **(c)**Spotted Fur

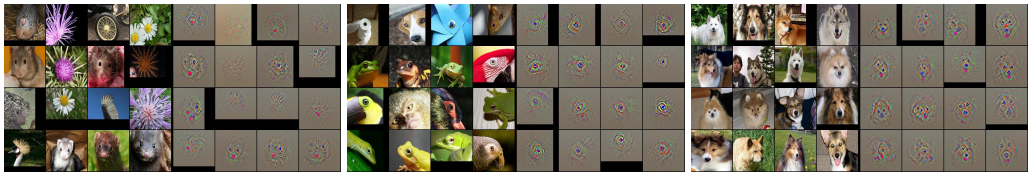
Fig. 2.6: These neurons are highly activated for the left cat in figure 2.5.

AlexNet appears to detect pet faces, eyes, and spotted fur, which are parts of cats. These high level interpretable features have been observed elsewhere in the literature [ZF14; Zho+15].

Now let's connected the layers. Fig. 2.7a shows the eye detector from Fig. 2.6b on top and units in the 4th convolutional layer which highly activate it on the bottom. It shows how the conv5 eye detector is made of simpler parts which are closer to being understood in terms of pixels. Of course, this makes sense because a convolutional layer is just a linear model, so it just weights parts (feature maps from the previous layer). Think of this just like the MNIST example (Fig. 2.2), except detecting eyes instead of 5s, and using faces and motifs (Fig. 2.7b-Fig. 2.7d) instead of pixels. That one of the components seems to be an eye detector itself could be a problem with the network or a failure of interpretation; it is left for future work. Next, Fig. 2.8 investigates the new eye detector of Fig. 2.7c.

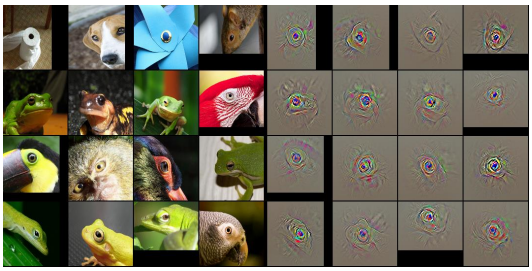


(a)(conv5) eye detector from (Fig. 2.6b)

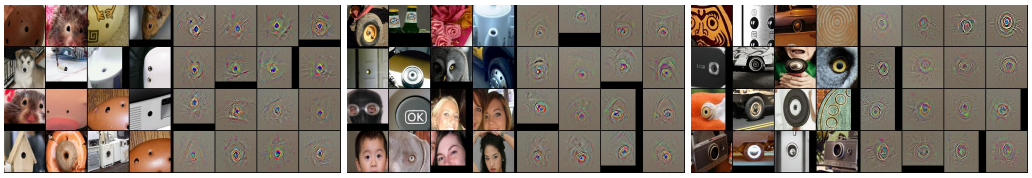


(b)(conv4) radial purple lines **(c)**(conv4) eyes less context **(d)**(conv4) another face detector

Fig. 2.7: Parts of an animal eye detector. (conv5)



(a)(conv4) eye detector from (Fig. 2.7c)



(b)(conv3) black dots **(c)**(conv3) eyeish things **(d)**(conv3) concentric circles

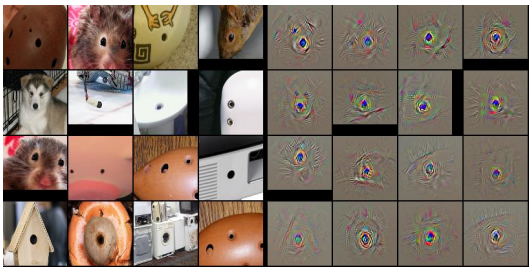
Fig. 2.8: Parts of an eye detector. (conv4)

In this case, the eye detector in conv4 picked up on some simpler components of eyes: black shiny dots (Fig. 2.8b), eyeish things (Fig. 2.8c), and concentric circles (Fig. 2.8d). It's starting to become more clear how these can be extracted from pixels, but it will become even more clear by going one more layer down and visualizing the black shiny dot detector (Fig. 2.8b), which is done in Fig. 2.9.

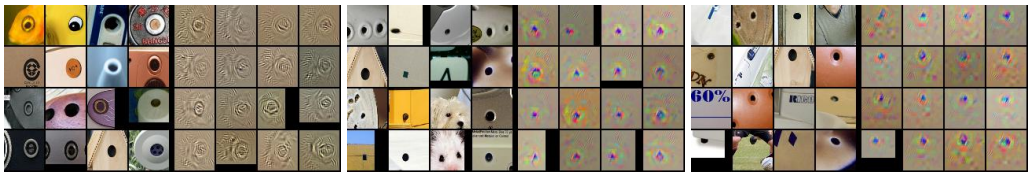
These neurons are definitely not looking for faces, but just dark dots (Fig. 2.9c-Fig. 2.9d) and textured circles (Fig. 2.9b). Notice that now there are very few examplars which have animal eyes. The number of face-activated dot detectors has gone down from layer to layer, supporting the idea that features are getting more generic.

The 3rd layer is more closely related to pixel level features, but not quite all the way there, so the final figure (Fig. 2.10) visualizes layer 2 in terms of layer 1 by focusing on the middle dot detector (Fig. 2.9c). This last visualization (Fig. 2.10) shows how the dot detector is composed of very simple edges. It's no coincidence that the vision community has been interested in edges for a long time [MH80].

This gives one example of how CNNs recognize abstract concepts in images. It clearly shows one part of the hierarchy that has been used to describe deep networks using using specific examples, allowing more concrete intuitions. The method is fairly easy and computationally efficient enough to trace other paths through the network.



(a)(conv3) dot detector from (Fig. 2.8b)

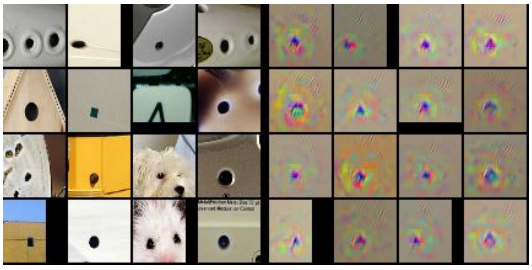


(b)(conv2) circles surrounded by texture

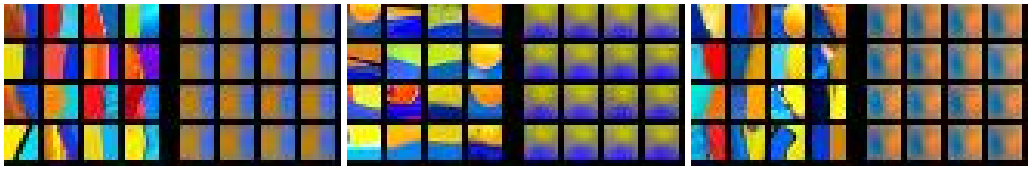
(c)(conv2) dot detector

(d)(conv2) dot detector

Fig. 2.9: An eye detector that looks for dots and circles. (conv3)



(a)(conv2) dot detector from (Fig. 2.9c)



(b)(conv1) orange to blue vertical edge

(c)(conv1) yellow to blue horizontal edge

(d)(conv1) blue to orange vertical edge

Fig. 2.10: Low level features which contribute to a dot detector and can very clearly be understood in terms of pixels. (conv1)

3 DeCov: Decorrelating Hidden Representations

This chapter changes gears to focus on training neural networks and the problem of overfitting, though the approach can be motivated by intuitions about parts in the last chapter.

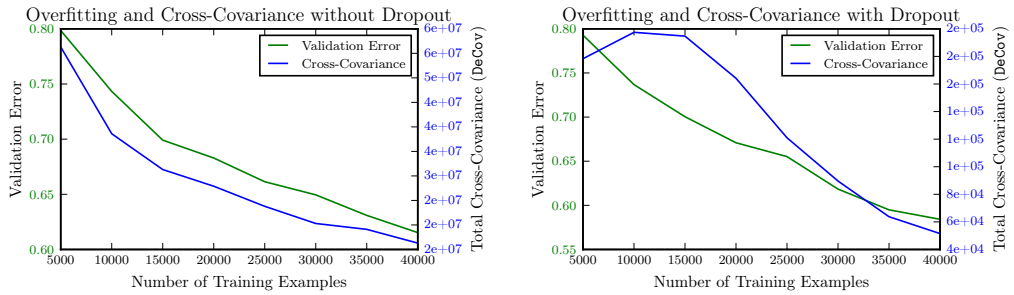
Consider AlexNet [Kri+12a], which is trained on a 1000-way image classification tasks. Without preventative measures, it can achieve 99.49% accuracy on training data while reaching 45.98% accuracy on test data. Overfitting is clearly a major issue for this network. The model learns lots of concepts that are specific to the training data, but it does not generalize to unseen data. This emphasizes why performance needs to be measured using data that can't be memorized beforehand.

Part of deep learning's success is driven by training networks with more layers, more neurons per layer, and enormous datasets. Big data clearly helps reduce overfitting, but even with lots of examples networks can only generalize so well [Yos+14]. This is particularly problematic when training a network for one task, say image classification [Kri+12b], and then fine-tuning or adapting the network for another task, say object detection [Gir+14] or attribute classification [Zha+13]. If the amount of training data available on the fine-tuning task is limited, it is common to overfit in that stage, even after discriminative pre-training for initialization.

Creating ever larger datasets for all such tasks of interest is a possible solution, but dataset creation and annotation is expensive and time-consuming. A promising alternative is to apply different forms of regularization that avoid overfitting by changing the optimization landscape so it defines different local minima. These methods include regularizing the norm of the weights [Tik43], lasso [Tib96], dropout [Sri+14], etc.

One particular regularizer of interest to deep nets is dropout [Sri+14], which attempts to prevent co-adaptation of neuron activations. This phenomenon occurs when two or more hidden units rely on one-another to perform some function which helps fit training data, thus becoming highly correlated. Co-adaptation is reduced by dropout using an approximate model averaging technique that sets randomly selected activations to zero at training time. The results in [Sri+14] show that this performs regularization, leading to increased generalization and also sparser, less correlated features without *explicitly* encouraging sparsity or decorrelation.

In section 7 of [Sri+14], the correlation effect is observed through feature visualizations (similar to Fig. 3.3a and Fig. 3.3c). Features from a network trained without dropout appear self-similar, but adding dropout leads to more distinct features. Figure 3.1 measures cross-covariance [1] and validation error on two CNNs trained for image classification (CIFAR100 [KH09]). The figure shows that two ways of preventing overfitting (training with more data and applying dropout) are correlated with cross-covariance of a vector of hidden activations. Thus, placing an explicit penalty on cross-covariance might reduce overfitting.



(a) Without dropout **(b)** With dropout

Fig. 3.1: Two principal ways to prevent overfitting in deep models are to train with more data (x axis) and to train with dropout (right plot). As expected, both of these decrease validation error (left axis), but they also happen to decrease hidden activation cross-covariance (right axis). Perhaps decreasing cross-covariance will also prevent overfitting.

This leads to the main question – Is it possible to bias networks toward decorrelated representations by directly reducing correlation between hidden units? And do such decorrelated representations generalize better?

3.1 Overview

The goal is to learn deep networks with decorrelated activations and study the effect of this decorrelation on their generalization performance. To this end, a novel layer called `DeCov` is developed to explicitly encourage decorrelation between activations in a deep neural network. Crucially, this loss requires no additional supervision, so it can be added to any existing network.

In addition to the success of dropout [Sri+14], another motivation comes from the classical literature on bagging and ensemble averaging [HS90; PC93; Bre96], which suggests that decorrelated ensembles perform better than correlated ones.

Experiments on a range of datasets (MNIST [LeC+95], CIFAR10/100 [KH09], ImageNet [Rus+14]), and network architectures (Caffe example architectures based MNIST,

[1] More precisely, this is the `DeCov` loss proposed in the next section, which is the L2 norm squared of cross-covariances less that of variances.

CIFAR10, and CIFAR100 along with AlexNet and the Network in Network [Lin+14]) suggest that networks trained in the presence of a DeCov layer result in parameters that overfit less and generalize better *in every single experiment*. DeCov not only outperforms existing regularizers such as weight decay and dropout in all experiments, but in a number of cases also provides improvements *even in the presence of dropout*. This suggests that DeCov acts as a novel and useful regularizer and that it might be able to replace the role of dropout in some existing networks.

3.2 Approach: DeCov Loss

This section expresses the notion of redundant or co-adapated features by imposing a loss on the activations of a chosen hidden layer. Let $\mathbf{h}^n \in \mathbb{R}^d$ denote the activations at the chosen hidden layer, where $n \in \{1, \ldots, N\}$ is one example from a batch of N. The covariances between all pairs of activations i and j form a matrix C:

$$C_{i,j} = \frac{1}{N} \sum_n (h_i^n - \mu_i)(h_j^n - \mu_j) \tag{3.1}$$

where $\mu_i = \frac{1}{N} \sum_n h_i^n$ is the mean of activation i over the batch.

The norm of C is not penalized directly because the diagonal contains the variance of each hidden activation. There is no reason to require the dynamic range of activations to be small, so this term is subtracted from the matrix norm to get the final DeCov loss

$$\mathcal{L}_{\texttt{DeCov}} = \frac{1}{2} \left(\|C\|_F^2 - \|diag(C)\|_2^2 \right) \tag{3.2}$$

where $\| \cdot \|_F$ is the frobenius norm, and the $diag(\cdot)$ operator extracts the main diagonal of a matrix into a vector. In experiments, subtracting the diagonal made little difference for small networks, but led to increased stability for larger networks.

Perhaps the best quality of this loss is that it requires no supervision, so it can be added to any set of activations. Here it's applied to the activations immediately before a network's softmax layer (*e.g.*, the 6th and 7th layers of AlexNet). Imposing the loss deeper in the network means it affects almost all weights as opposed to just the few in lower layers. This strategy is inspired by dropout, which is most often applied just to deeper layers.

At first glance, one seeming peculiarity about this loss is that its global minimum can be found by setting all weights for $\mathbf{h}$ to 0. In that fashion, DeCov is similar to an L_2 regularizer. The difference is that DeCov prefers penalizing some weights more harshly than others in a fashion that's dependent on the data while L_2 is not a function of the data.

To understand this further, let's look at some intuition behind cross-covariance. Consider the gradient of DeCov with respect to a particular activation a for a particular example m

$$\frac{\partial \mathcal{L}_{\texttt{DeCov}}}{\partial h_a^m} = \sum_{j \neq a} \left[\frac{1}{N} \sum_n (h_a^n - \mu_a)(h_j^n - \mu_j) \right] \frac{1}{N} (h_j^m - \mu_j). \tag{3.3}$$

Denote the term on the right of this expression by $I(j, m) = \frac{1}{N}(h_j^m - \mu_j)$.

This is large (in absolute value) when feature j differentiates example m from the rest of the batch. If j were not "unique" for m then h_j^m would be close to μ_j. Hence I is an "importance" term, corresponding to a notion of how significant feature j is for example m.

Also notice that the term on the left in the gradient expression is simply the covariance between feature a and feature j. Thus, the gradient can be re-written as

$$\frac{\partial \mathcal{L}_{\texttt{DeCov}}}{\partial h_a^m} = \sum_{j \neq a} C_{a,j} \cdot I(j, m). \tag{3.4}$$

Interpretation

Intuitively, the covariance term can be thought of as measuring (linear) redundancy: features a and j are redundant if they vary together. Thus, DeCov loss tries to prevent features from being redundant, but redundancy is weighted by importance (I). Specifically, a feature j contributes towards a large gradient of feature a on example m if j is itself important for m *and* correlated with a. A feature which fires only in specialized situations (*e.g.*, a cat's fur) will receive small gradients when it is not relevant because it will be nearly identical across examples or noisy.

3.3 Related Work

Redundancy Based Representations.

The idea of using low redundancy to learn representations has been around for decades. In an early attempt to model human perception, [Bar61] lists 3 possible learning principles, the 3rd being the notion that representations should not be redundant.

Later work continued to investigate this intuition in the context of unsupervised feature learning. Three objectives emerged, each of which formalize the notion differently. (1) An information theoretic view is expressed by [Lin88]. The main idea is to maximize

information gained by predicting the next representation/layer between input and output. (2) The closest objective to `DeCov` is cross-correlation (not cross-covariance), which appears in [BB09] and complements a temporal coherence objective. It also appears in [PH86] where it complements an objective which encourages units to capture higher order input statistics. (3) Finally, redundancy minimization is realized through predictability minimization in [Sch92] for the purpose of learning factorial codes (representations whose units are independent). This objective says that one unit should not be predictable given *all* of the others in its layer as input.

All of these works focus on unsupervised feature learning and do not experiment with supervised models. Furthermore, these early pioneering works were limited by data and evaluated small networks without many of the modern design choices and features (e.g. ReLus, Dropout, SGD instead of Hebb's update rule, batch-normalization, *etc*.). In contrast, this work proposes redundancy minimization for a new purpose (regularization), evaluates it using modern techniques such as end-to-end learning using SGD with respect to a supervised objective, and does this in the context of harder challenges presented by modern datasets. To the author's best knowledge, such a setting has not been considered before.

Correlation/Covariance Based Losses in Other Settings.

Still other works have used similar penalties, but in very different ways and to different effects. Deep Canonical Correlation Analysis (Deep CCA) [And+13] and Correlational Neural Networks (CorrNets) [Cha+15] apply a similar loss which *maximizes* correlation, unlike our *minimization* of cross-covariance. Both methods are used to learn better features in the presence of multiple views or modalities. They embed inputs to a common space and maximize correlation between aligned pairs.

In particular, CorrNets are motivated by the following scenario – consider processing recordings of TV shows with subtitles to infer which actors are saying what. Both the subtitles and the audio from the video can be used – subtitles let us infer what was said and audio helps us infer who said it. To do this, [Cha+15] projects both the video signal with the subtitles and the audio signal into a common embedding space. To force the embeddings of the different modalities (video, audio) of the same dialog to be similar, CorrNets *maximize* the correlation between the two embeddings. The correlation term in the loss they impose is almost identical to `DeCov`, excepting two differences: (1) it is correlation (covariance normalized by standard deviation) and (2) their loss takes two *different* representations instead of just one representation, which is not the case in this work.

Perhaps most similar to the goal here is the work of Cheung *et al*. [Che+14b], who aim to discover and disentangle hidden factors by separating supervised factors of variation (*e.g*., class of MNIST digits) from unsupervised factors of variation (*e.g*., handwriting

style). In order to achieve this goal, they impose a covariance (not correlation) loss between (1) the softmax outputs of a neural network trained to recognize digits and (2) a hidden representation which is used in conjunction with (1) to reconstruct the input (via an auto-encoder).

These two approaches suggest that this type of decorrelation loss does have a significant impact on the parts that are learned by the hidden units. One key difference between these two approaches and `DeCov` is that while the other formulations decorrelate [Che+14b] and disregard [Cha+15] parts of *different* representations, `DeCov` tries to decorrelate parts of the *same* representation. Moreover, the ultimate goals are different. Unlike these approaches, the goal of `DeCov` is simply to improve supervised classification performance by reducing overfitting, and not to reconstruct the original data.

Other Regularizers

Two recent approaches to regularization in deep neural networks are dropout [Sri+14] and to some extent Batch Normalization [IS15]. Dropout aligns with the `DeCov` intuition and goals more closely as it aims to improve classification performance by reducing co-adaptation of activations. On the other hand, Batch Normalization focuses on faster optimization by reducing *internal co-variate shift*, which is the constant variation of a layer's input as it learns. Some Batch Normalization results indicate it could act as a regularizer, but this has not been exhaustively verified yet. This approach is most similar to Batch Normalization due to its use of mini-batch statistics.

3.4 Experiments

To begin, consider a synthetic experiment which uses two explicit modalities. This synthesized experiment allows us to control the amount of correlation or *inject* artificial bias into the training data. Such a setup serves as a sanity check for `DeCov`, since there is high confidence it is a good loss to apply. Following this experiment, results are presented on image classification tasks using data from CIFAR10/100 and ImageNet alongside different popular architectures.

3.4.1 Dual modality experiments with MNIST: Predicting Side-by-Side Digits

Consider the following synthetic dual "modality" task on MNIST – simultaneously predict the class labels for two digits placed adjacent in an image. Each example in the dataset will consist of two MNIST digit images horizontally concatenated, but separated by 16 black

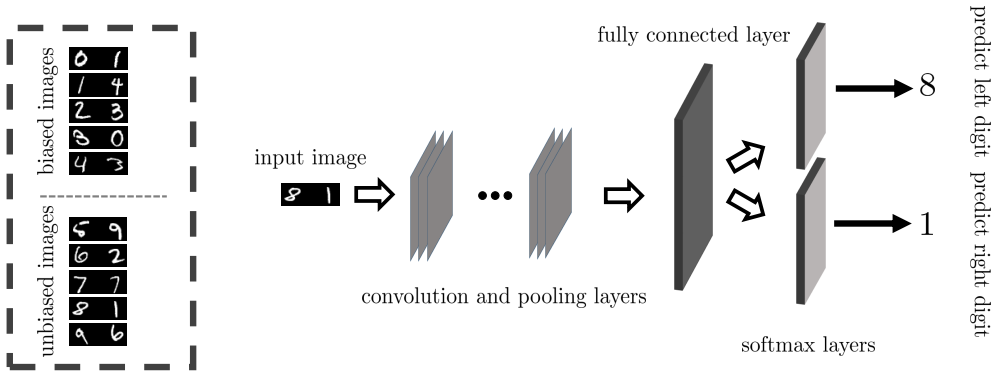


Fig. 3.2: Consider the task of simultaneously predicting two MNIST digits placed side by side. By training on a biased dataset `DeCov` can be tested in a situation where it's highly likely to improve performance.

pixels to prevent interference between feature maps in the first layers. Fig. 3.2 shows a few examples.

The important detail of this experiment is the particular bias injected into the distribution of left and right digits. Let

$$P(l) = 0.1 \quad \text{and} \quad P(r|l) = \begin{cases} 0 & \text{if } l \in \{0,\dots,4\} \text{ and } r \in \{0,\dots,4\} \\ 0.2 & \text{if } l \in \{0,\dots,4\} \text{ and } r \in \{5,\dots,9\} \\ 0.1 & \text{if } l \in \{5,\dots,9\} \end{cases}. \tag{3.5}$$

To generate one example, first sample the left digit using $P(l)$ then the right using $P(r|l)$. Conditional entropies of one digit given the other can be computed to get $H(l|r) = 2.0868$ and $H(r|l) = 1.9360$. Since $H(l|r) > H(r|l)$, the left digit is more informative of the right than the right is of the left. There is no cross-digit signal at test time, so features for the right and left digits should be completely decorrelated to generalize, but learned features will have some correlation between left and right. Intuitively, `DeCov` should help generalization in this scenario. This can be seen in Tab. 3.1, where the right digit classifier always has worse generalization (test) error than the left digit classifier.

		Left Digit			Right Digit		
`DeCov`	dropout	train	test	train - test	train	test	train - test
no	no	99.98 ± 0.01	97.94 ± 0.18	2.05 ± 0.19	100.00 ± 0.00	96.75 ± 0.24	3.25 ± 0.24
no	yes	99.99 ± 0.00	98.45 ± 0.04	1.54 ± 0.04	99.99 ± 0.00	97.39 ± 0.20	2.61 ± 0.20
yes	yes	99.97 ± 0.01	98.59 ± 0.12	1.38 ± 0.12	99.99 ± 0.00	97.81 ± 0.07	2.18 ± 0.06
yes	no	99.99 ± 0.00	$\mathbf{98.74 \pm 0.03}$	$\mathbf{1.25 \pm 0.04}$	99.99 ± 0.00	$\mathbf{97.99 \pm 0.12}$	$\mathbf{2.00 \pm 0.12}$
weight decay		99.97	97.86	2.11	99.97	96.21	3.76

Tab. 3.1: MNIST side by side results.

The model uses Caffe's [Jia13] example architecture, which is based on LeNet [LeC+95]. It has two convolution layers, each followed by pooling, then a fully connected layer with 500 hidden units which are shared between the two softmax layers. DeCov and/or dropout is applied on the 500 hidden units of the fully connected layer.

Results

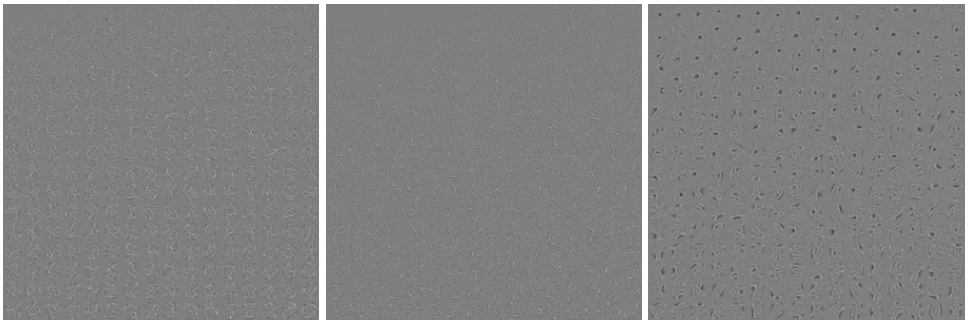
(Tab. 3.1) reports the accuracy of left and right digit classifiers. Injected dataset bias can be clearly seen in the lower test accuracy and higher train-test gap of the right classifier, indicating that all of our networks incorporate the train time bias into their predictions. Mean accuracies across 4 trials are reported, along with the standard deviation.

The main result is that the gaps between the performance of DeCov and the baselines are larger for the biased right digit (*e.g.*, right digit test accuracy shows a $\sim$0.6% improvement when switching from Dropout-alone to DeCov-alone while the improvement for left digits is just $\sim$0.3%). This suggests that the baselines pick up on the false bias and that DeCov does the best job of correcting for it. DeCov also improves generalization for both classifiers since test accuracy is higher in the bottom two rows and the train - test gap is lower in those rows. Combining Dropout with our DeCov loss hurts slightly, but note that the error bars overlap in some cases, so this is not a statistically significant difference.

One hypothesis is that the DeCov loss is simply enforcing something akin to an L2 penalty on the weights. If this was the case then putting a heavier weight on the L2 penalty would give results that are similar to just adding the DeCov loss. To test this hypothesis, consider a range of weight decay penalties $\{0.0005, 0.001, 0.01, 0.1, 1.0\}$ without using DeCov or dropout. The best accuracies are reported in the last row of Tab. 3.1 and they show that the best L2 penalty does not provide as significant a boost as DeCov.

3.4.2 MNIST Autoencoder

Figure 3.3 visualizes learned features using the 2 layer autoencoder experiment from section 7 of dropout [Sri+14]. In that experiment an autoencoder was trained on raw pixels of single MNIST digits. It had an encoder with 1 layer of 256 ReLU units and a decoder (untied weights) that produced a 784 (28×28) ReLU output. This is just a 2 layer MLP which regresses to digit images. In Fig. 3.3 these weights from the first layer of the autoencoder are reshaped in the same manner as section 2.2.1. Inputs similar to these weight images will cause high activations for the corresponding feature. The figure also reports the mean squared reconstruction error which measures how accurately the autoencoder was able to compress then reconstruct MNIST digits.



(a)Baseline with MSE = 1.47 (b)DeCov with MSE = 0.98 (c)dropout with MSE = 3.03

Fig. 3.3: Weights learned by the first layer of a 2 layer autoencoder are reshaped into images and visualized for a model with no DeCov or dropout (Fig. 3.3a), a model with DeCov (Fig. 3.3b), and a model with dropout (Fig. 3.3c).

Weight initialization turned out to be an important factor for the visualizations. Initializing all weights by sampling from $U[-\sqrt{\frac{3}{n}}, \sqrt{\frac{3}{n}}]$ (based on [GB10]) led to visualizations as seen in [Sri+14] (the baseline looks like noise), but sampling weights from a Gaussian with mean 0 and standard deviation 0.001 led to baseline visualizations with feint digit outlines. The latter initialization was used in (Fig. 3.3).

The baseline (Fig. 3.3a) shows feint digit outlines and is able to reconstruct inputs fairly well (MSE = 1.47). DeCov shows similar outlines, though there it has lower intensities and perhaps fewer complete digit outlines than the baseline. DeCov also has the best reconstruction error (MSE = 0.98). On the other hand, dropout features are much more localized and don't encode the image as well (MSE = 3.03).

What matters here is not so much the reconstruction error of the autoencoder, since the goal is supervised learning, but a comparison of DeCov and dropout. Remember that DeCov was motivated by an observation that dropout decreases Cross-Covariance (Fig. 3.1). DeCov explicitly applies that same intuition, so a plausible result would be for DeCov to behave similarly to dropout. This experiment suggests that there is a significant difference between the two because of the difference in visualizations and the different reconstruction errors.

3.4.3 Image Classification

CIFAR10

The CIFAR10 dataset contains 60,000 32x32 images sorted into 10 distinct categories [KH09]; this is a simple task because small images have limited visual complexity and because the 10 classes have fairly large inter-class variance. Results are reported by training on the 50,000 given training examples and testing on the 10,000 specified test samples. Hyper-parameters are chosen by the standard train/val split.

These experiments use Caffe's quick CIFAR10 architecture, which has 3 convolutional layers followed by a fully connected (linear) layer with 64 hidden units and a softmax layer. The hidden fully connected layer is not followed by a non-linearity. DeCov loss is added only to the 64 hidden units of the hidden fully connected layer.

DeCov	dropout	train	test	train - test
no	no	100.0 ± 0.00	75.24 ± 0.27	24.77 ± 0.27
no	yes	99.10 ± 0.17	77.45 ± 0.21	21.65 ± 0.22
yes	yes	87.78 ± 0.08	$\mathbf{79.75 \pm 0.17}$	$\mathbf{8.04 \pm 0.16}$
yes	no	88.78 ± 0.23	79.72 ± 0.14	9.06 ± 0.22
weight decay		100.0	75.29	24.71

Tab. 3.2: CIFAR10 Classification

Results

In Table 3.2 there is a significant improvement when using the DeCov loss – a ∼4.5% increase in test accuracy. Moreover, the DeCov loss reduces the gap between train and val accuracies by ∼15% (without dropout) and ∼16% (with dropout)!

Comparing the four combinations, using DeCov alone provides a larger improvement than using dropout. Using both DeCov and dropout further improves the generalization (as measured by the gap in train and test accuracies), but the improvement in absolute test performance does not seem statistically significant. All reported results are average performance over 4 trials with the standard deviation indicated alongside.

Though it didn't for MNIST, perhaps L2 weight decay can provide similar improvements in this larger scale experiment, so the table shows experiments with different amounts of decay. Once again, the best setting gives little improvement over the baseline. Even though DeCov can be minimized by favoring small weight vectors (L2 regularization), it seems clear that DeCov is doing something more selective than simply reducing the norms of the weights.

One more experiment with a larger network for image classification on CIFAR10 illustrates the benefit of an effective regularizer. To get this larger network, add another fully connected layer to the network used in the previous experiment, double the number of feature maps in each convolutional layer, and double the number of units in the fully connected layers. As expected, performance increases, but so does overfitting.

This larger network performs better than the smaller version – all accuracies in Tab. 3.3 are higher than corresponding entries in Tab. 3.2. However, there are the stronger indications of overfitting in this network – specifically, the train accuracies are much higher than test

accuracies (when compared to the previous network). Trends are similar to the previous experiments – there are significant gains from using DeCov alone compared to dropout alone, and there is a further slight improvement from combining both. Using dropout alone gives a $\sim$1.5% boost in test accuracy, while using DeCov alone provides a $\sim$4% increase in test accuracy. Using both yields roughly the same test performance, but the trainval and test gap is further reduced.

DeCov	dropout	(train+val)	test	(train+val) - test
no	no	100.00	77.38	22.62
no	yes	100.00	79.93	20.07
yes	yes	96.76	**81.68**	**15.08**
yes	no	98.15	81.63	16.52

Tab. 3.3: CIFAR10 Classification with a bigger version of the base network

CIFAR100

This section scales up the experiments on CIFAR10 by moving to CIFAR100 [KH09]. The architecture used here is identical to the base architecture for CIFAR10, except the softmax has 100 outputs. Table 3.4 shows that dropout alone has higher test performance than DeCov alone, but DeCov leads to a smaller train-test gap. Using both regularizers not only achieves the highest test accuracy, but also the smallest train-test gap, which is *severely* reduced. This suggests that the two regularizers have complementary effects, however previous (MNIST) and future experiments sometimes show decreased accuracy from simultaneous dropout and DeCov.

DeCov	dropout	train	test	train - test
no	no	99.77	38.52	61.25
no	yes	87.35	43.55	43.80
yes	yes	72.53	**45.10**	**27.43**
yes	no	77.92	40.34	37.58

Tab. 3.4: CIFAR100 Classification Accuracies

In all cases the weight on the DeCov loss has been chosen using grid search. Unfortunately, it's hard to find a pattern which shows how to balance main cross entropy loss and DeCov. However, looking at performance across small variations in architecture for the same task and dataset shows some consistency. Performing a grid search over DeCov weights ($\{10, 1, 0.1, 0.01, 0.001, 0.0001, 0.00001\}$) and the number of activations in the one hidden full connected layer ($\{32, 64, 96, 128, 256, 384, 512\}$) shows that the best DeCov weight is always 0.1.

ImageNet - AlexNet

This section demonstrates large scale results using networks trained for ImageNet classification, starting by applying DeCov to fc6 and fc7 in AlexNet [Kri+12b]. The last 50,000 of the ILSVRC 2012 train images are held out for validation and the implementation comes from Caffe. In particular, it uses a fixed schedule that multiplies the learning rate by 1/10 every 100,000 iterations (see jumps in (Fig. 3.4)). We do not use early stopping and do not perform color augmentation.

In (Fig. 3.4) notice that when neither of the two regularizers – Dropout or DeCov– are applied (blue line), the network overfits (it even gets 100% train accuracy), and the DeCov loss (hidden activation redundancy) is higher than with any other combination of the regularizers. Applying either of the regularizers also causes a synchronous drop in both losses. Explicitly minimizing the DeCov loss naturally leads to much lower DeCov losses and significantly reduced overfitting. Interestingly, Dropout results in relatively lower DeCov loss too, even when DeCov is not optimized for. This is further indication of the link between redundant activations and overfitting.

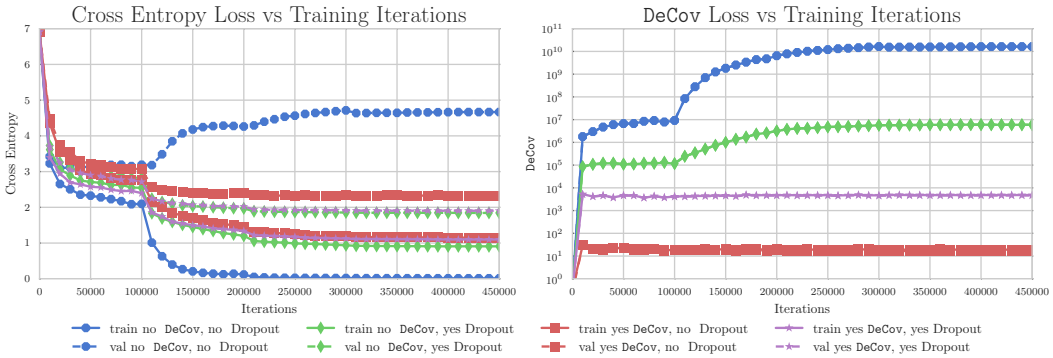


Fig. 3.4: Cross Entropy and DeCov losses over the course of training AlexNet with 256x256 images. Note that the DeCov val curves are hidden by the train curves. Interestingly, DeCov is reduced even by Dropout, though not nearly as much as when it is explicitly minimized.

(Fig. 3.5) shows accuracies across different image resolutions we used to train AlexNet. AlexNet is typically trained with 256x256 images, but training with smaller images is faster [2] *and* reduces the number of parameters in the network. Smaller images (we use 128x128, 160x160, 192x192, and 224x224) lead to smaller feature maps output by pool5, so the dense connection between pool5 and fc6 has fewer parameters, the model has less capacity, and it's less likely to overfit. For example, images scaled to 256x256 (taking 227x227 crops [3]) lead to a weight matrix with 38 million parameters while 128x128 images (with 99x99 crops) result in a 4 million parameter matrix. Generally, accuracies (left plots) and the train-val gap (right plots) have a slight positve slope, confirming that performance and overfitting

[2] Using CuDNNv3, AlexNet with 128x128 inputs takes 103ms averaged over 50 runs to compute a forward and backward pass. For 256x256 images this time is 449ms.

[3] At train time crops are sampled and mirrored randomly. At test time only the center 227x227 crop is used.

increase with resolution and model capacity. Note that the DeCov loss weight was tuned using grid search at each resolution both with and without Dropout.

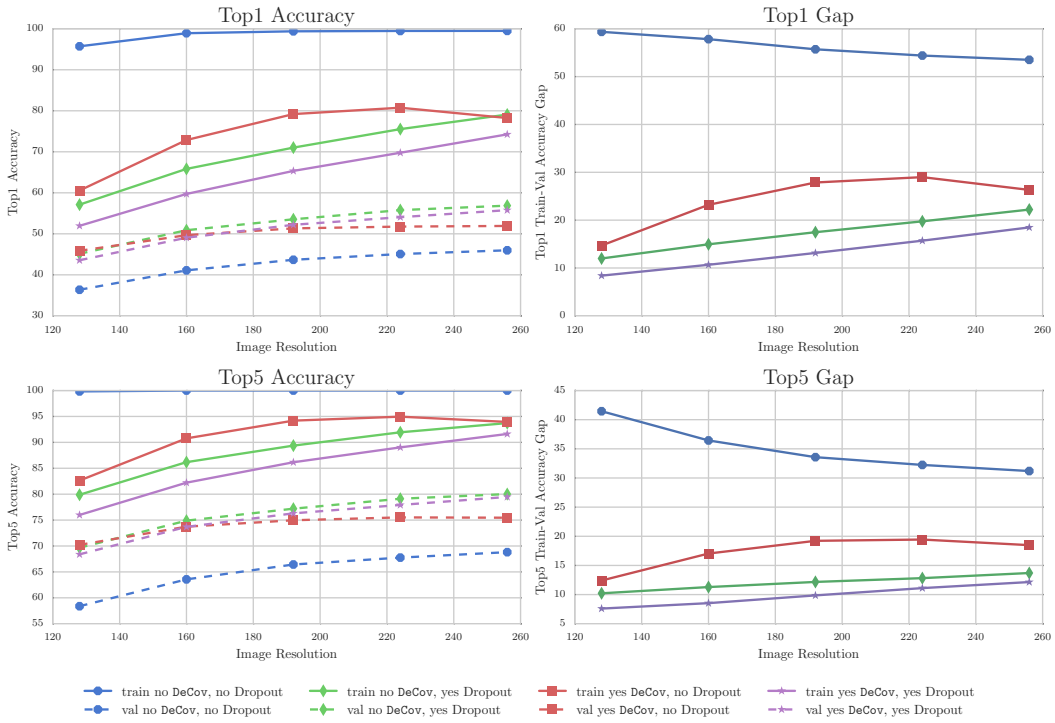


Fig. 3.5: ImageNet classification performance using AlexNet. Plots on the left show training and validation (ILSVRC 2012 validation set) accuracy at different resolutions. Note how all curves have a much lower train-val gap than the (blue) baseline.

We see that Dropout alone (green) usually has the best val accuracy, which is slightly higher than the two losses combined (purple) and a couple points higher than DeCov alone (red) at higher resolutions. At the lowest resolution Dropout alone is tied with DeCov alone. Dropout also reduces overfitting more than DeCov, though both independently reduce overfitting by a large margin – from 59.35% to 14.7% in the case of DeCov @ 128x128.

ImageNet - Network in Network

Finally, the new regularizer is tested on ILSVRC 2012 with one more architecture – that of Network in Network [Lin+14].[4] The architecture is fully convolutional: it contains 4 convolutional layers, with 96, 256, 384, and 1024 feature maps, respectively. Between each of these layers and after the last are two convolutional layers which have 1x1 kernels, which further process each feature map output by the main convolutional layers before being fed into the next layer. To produce 1000 softmax activations, 1000 feature maps are averaged over spatial locations to produce one feature vector. DeCov is applied to these average pooled feature vectors. Surprisingly, this architecture had very little overfitting

[4]This is the model provided in the Caffe Model Zoo: https://gist.github.com/mavenlin/d802a5849de39225bcc6

to begin with. However, adding a DeCov loss still decreases overfitting substantially, and improves validation accuracies (Tab. 3.5) - there is a small boost in performance on validation accuracies, but there is significant decrease of ∼3% (for top 1) and ∼2% (for top 5) in the difference between the train and val accuracies.

DeCov	dropout	ILSVRC 2012 train top 1	ILSVRC 2012 val top 1	train - val
no	no	71.68	58.67	13.01
no	yes	71.32	58.95	12.37
yes	yes	68.28	**59.08**	**9.20**
yes	no	68.33	58.85	9.48
DeCov	dropout	ILSVRC 2012 train top 5	ILSVRC 2012 val top 5	train - val
no	no	89.91	81.18	8.73
no	yes	89.63	81.53	8.10
yes	yes	87.99	**81.94**	**6.05**
yes	no	87.88	81.57	**6.05**

Tab. 3.5: ImageNet Classification Accuracies with Network in Network

3.5 Discussion

The main takeaway is that, across a variety of scales and architectures, all experiments indicate that DeCov reduces overfitting as measured by the gap between train and test performance.

In the presented experiments networks were always trained from scratch, but it's also possible to fine-tuning networks in different scenarios – *i.e.*, train a network for one task then use those weights to initialize a similar network for another task (or loss). Both ImageNet networks (Network in Network and AlexNet) were fine-tuned from parameters that weren't trained with a DeCov loss, but were trained with dropout. In both cases performance either stayed where it was at fine-tuning initialization or it decreased slightly (within statistical significance). Similar results were found when fine-tuning for other tasks like attribute classification (fine-tuning AlexNet) and object detection (Fast R-CNN [Gir15]).

This, along with some cases where combining dropout and DeCov decrease performance, suggests that the DeCov loss may possibly be acting adversarially to activations learned by dropout. Fine-tuning with DeCov is an interesting direction for future work.

4 Conclusion

This thesis introduced deep representations, then developed an intuition about those representations into a loss which led to better representations.

First, deep neural network representations were explained mathematically and intuitively. These models are a composition of layers of linear models, except each stage of the composition uses a non-linear function to throw out or de-emphasize some information. Each level of composition combines simpler parts to form more complex parts, thereby bridging the gap between indirectly (non-linearly) related concepts.

Second, a new regularizer for deep networks was investigated. It enforces the idea that parts/features learned by different hidden activations should be unique. This was realized by the `DeCov` loss, which explicitly penalized the covariance between activations in the same layer of a neural network in an unsupervised fashion, acting as a strong regularizer for deep neural networks and offering a new tool to combat overfitting. Extensive experiments show that `DeCov` can compete with dropout over a range of datasets and architectures. Though it sometimes doesn't outperform dropout, it does always reduce overfitting.

Both of these chapters used an intution about *parts* which is pervasive in Machine Learning. Deep models learn which parts are useful, requiring less supervision from humans who might not understand what they're supervising, and it showed how an intuition about parts – that they should not be redundant – can improve image classification models.

Bibliography

[And+13] Galen Andrew, Raman Arora, Jeff Bilmes, and Karen Livescu. „Deep canonical correlation analysis“. In: *Proceedings of the 30th International Conference on Machine Learning*. 2013, pp. 1247–1255 (cit. on p. 19).

[Ant+15] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, et al. „VQA: Visual Question Answering“. In: *International Conference on Computer Vision (ICCV)*. 2015 (cit. on p. 1).

[Bar61] Horace B Barlow. „Possible principles underlying the transformations of sensory messages“. In: (1961) (cit. on p. 18).

[BB09] Yoshua Bengio and James S Bergstra. „Slow, decorrelated features for pretraining complex cell-like networks“. In: *Advances in neural information processing systems*. 2009, pp. 99–107 (cit. on p. 19).

[Bre96] Leo Breiman. „Bagging predictors“. In: *Machine learning* 24.2 (1996), pp. 123–140 (cit. on p. 16).

[Cha+15] Sarath Chandar, Mitesh M Khapra, Hugo Larochelle, and Balaraman Ravindran. „Correlational Neural Networks“. In: *arXiv preprint arXiv:1504.07225* (2015) (cit. on pp. 19, 20).

[Che+14a] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, et al. „cudnn: Efficient primitives for deep learning“. In: *arXiv preprint arXiv:1410.0759* (2014) (cit. on p. 10).

[Che+14b] Brian Cheung, Jesse A. Livezey, Arjun K. Bansal, and Bruno A. Olshausen. „Discovering Hidden Factors of Variation in Deep Networks“. In: *Proceedings of the International Conference on Learning Representations (ICLR)* abs/1412.6583 (2014) (cit. on pp. 19, 20).

[Cho+15] Anna Choromanska, Yann LeCun, and Gérard Ben Arous. „Open Problem: The landscape of the loss surfaces of multilayer networks“. In: *Proceedings of The 28th Conference on Learning Theory*. 2015, pp. 1756–1760 (cit. on p. 8).

[CZ15] Xinlei Chen and C Lawrence Zitnick. „Mind’s eye: A recurrent visual representation for image caption generation“. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 2422–2431 (cit. on p. 1).

[Dau+14] Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, et al. „Identifying and attacking the saddle point problem in high-dimensional non-convex optimization“. In: *Advances in neural information processing systems*. 2014, pp. 2933–2941 (cit. on p. 8).

[Don+14] Jeff Donahue, Yangqing Jia, Oriol Vinyals, et al. „DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition.“ In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2014 (cit. on p. 1).

[GB10] Xavier Glorot and Yoshua Bengio. „Understanding the difficulty of training deep feedforward neural networks“. In: *International conference on artificial intelligence and statistics*. 2010, pp. 249–256 (cit. on p. 23).

[Gir+14] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jagannath Malik. „Rich feature hierarchies for accurate object detection and semantic segmentation“. In: *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*. IEEE. 2014, pp. 580–587 (cit. on p. 15).

[Gir15] Ross Girshick. „Fast R-CNN“. In: *International Conference on Computer Vision (ICCV)*. 2015 (cit. on pp. 1, 28).

[Ham] *History of Computers - Software*. https://www.youtube.com/watch?v=2e5_Z6oZ0rM. Accessed: 2015-07-21; quote at 30 mintues 0 seconds. 1995 (cit. on p. 2).

[Hin] *Neural Networks for Machine Learning*. https://www.coursera.org/course/neuralnets. 2012 (cit. on p. 2).

[HS90] Lars Kai Hansen and Peter Salamon. „Neural network ensembles“. In: *IEEE transactions on pattern analysis and machine intelligence* 12.10 (1990), pp. 993–1001 (cit. on p. 16).

[IS15] Sergey Ioffe and Christian Szegedy. „Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift“. In: *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*. 2015, pp. 448–456 (cit. on p. 20).

[Jia+14] Yangqing Jia, Evan Shelhamer, Jeff Donahue, et al. „Caffe: Convolutional architecture for fast feature embedding“. In: *Proceedings of the ACM International Conference on Multimedia*. ACM. 2014, pp. 675–678 (cit. on p. 10).

[Jia13] Yangqing Jia. *Caffe: An Open Source Convolutional Architecture for Fast Feature Embedding*. http://caffe.berkeleyvision.org/. 2013 (cit. on p. 22).

[Kar14] Andrej Karpathy. *Hacker's guide to Neural Networks*. 2014 (cit. on p. 2).

[KH09] Alex Krizhevsky and Geoffrey Hinton. „Learning multiple layers of features from tiny images“. In: *Computer Science Department, University of Toronto, Tech. Rep* 1.4 (2009), p. 7 (cit. on pp. 16, 23, 25).

[Kri+12a] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. „Imagenet classification with deep convolutional neural networks“. In: *Advances in neural information processing systems*. 2012, pp. 1097–1105 (cit. on pp. 1, 10, 15).

[Kri+12b] Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton. „ImageNet Classification with Deep Convolutional Neural Networks“. In: *NIPS*. 2012 (cit. on pp. 15, 26).

[LeC+12] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. „Efficient backprop“. In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 9–48 (cit. on p. 2).

[LeC+95] Yann LeCun, LD Jackel, L Bottou, et al. „Comparison of learning algorithms for handwritten digit recognition“. In: *International conference on artificial neural networks*. Vol. 60. 1995, pp. 53–60 (cit. on pp. 16, 22).

[LeC+98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. „Gradient-based learning applied to document recognition“. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324 (cit. on pp. 1, 4).

[Lin+14] Min Lin, Qiang Chen, and Shuicheng Yan. „Network in network“. In: *Proceedings of the International Conference on Learning Representations (ICLR)* (2014) (cit. on pp. 17, 27).

[Lin88] Ralph Linsker. „Self-organization in a perceptual network“. In: *Computer* 21.3 (1988), pp. 105–117 (cit. on p. 18).

[MH80] David Marr and Ellen Hildreth. „Theory of edge detection“. In: *Proceedings of the Royal Society of London B: Biological Sciences* 207.1167 (1980), pp. 187–217 (cit. on p. 14).

[Mni+13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, et al. „Playing atari with deep reinforcement learning“. In: *NIPS Deep Learning Workshop* (2013) (cit. on p. 1).

[NH10] Vinod Nair and Geoffrey E Hinton. „Rectified linear units improve restricted boltzmann machines“. In: *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*. 2010, pp. 807–814 (cit. on p. 10).

[PC93] Michael P. Perrone and Leaon N. Cooper. „When Networks Disagree: Ensemble Methods for Hybrid Neural Networks“. In: *Tech Report*. Chapman and Hall, 1993, pp. 126–142 (cit. on p. 16).

[PH86] Barak A Pearlmutter and Geoffrey Hinton. „G-maximization: An unsupervised learning procedure for discovering regularities“. In: *AIP conference proceedings*. Vol. 151. American Institute of Physics. 1986, pp. 333–338 (cit. on p. 19).

[Rum+88] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. „Learning representations by back-propagating errors“. In: *Cognitive modeling* 5 (1988), p. 3 (cit. on p. 8).

[Rus+14] Olga Russakovsky, Jia Deng, Hao Su, et al. „Imagenet large scale visual recognition challenge“. In: *International Journal of Computer Vision* (2014), pp. 1–42 (cit. on pp. 10, 16).

[Rus+15] Olga Russakovsky, Jia Deng, Hao Su, et al. „ImageNet Large Scale Visual Recognition Challenge“. In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252 (cit. on pp. 1, 3, 9).

[Sch+15] Florian Schroff, Dmitry Kalenichenko, and James Philbin. „Facenet: A unified embedding for face recognition and clustering“. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 815–823 (cit. on p. 1).

[Sch15] Jürgen Schmidhuber. „Deep learning in neural networks: An overview“. In: *Neural Networks* 61 (2015), pp. 85–117 (cit. on p. 8).

[Sch92] Jürgen Schmidhuber. „Learning factorial codes by predictability minimization“. In: *Neural Computation* 4.6 (1992), pp. 863–879 (cit. on p. 19).

[Spr+14] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. „Striving for Simplicity: The All Convolutional Net“. In: *International Conference on Learning Representations (ICLR) Workshop Track* (2014) (cit. on pp. 10, 11).

[Sri+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. „Dropout: A simple way to prevent neural networks from overfitting“. In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958 (cit. on pp. 1, 15, 16, 20, 22, 23).

[Tib96] Robert Tibshirani. „Regression Shrinkage and Selection via the Lasso“. In: *Journal of the Royal Statistical Society, Series B* 58 (1 1996), pp. 267 –288 (cit. on p. 15).

[Tik43] Andrey Nikolayevich Tikhonov. „On the stability of inverse problems“. In: *Dokl. Akad. Nauk SSSR* (1943), pp. 195–198 (cit. on p. 15).

[Vin+14] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. „Show and Tell: A Neural Image Caption Generator“. In: (2014). arXiv: 1411.4555 (cit. on p. 1).

[Yos+14] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. „How transferable are features in deep neural networks?“ In: *Advances in Neural Information Processing Systems*. 2014, pp. 3320–3328 (cit. on p. 15).

[ZF14] Matthew D. Zeiler and Rob Fergus. „Visualizing and Understanding Convolutional Networks“. In: *Proceedings of European Conference on Computer Vision (ECCV)*. 2014 (cit. on pp. 1, 11, 12).

[Zha+13] Ning Zhang, Manohar Paluri, Marc'Aurelio Ranzato, Trevor Darrell, and Lubomir Bourdev. „PANDA: Pose Aligned Networks for Deep Attribute Modeling“. In: *CVPR 2014 and arXiv preprint arXiv:1311.5591* (2013) (cit. on p. 15).

[Zha+15] Ning Zhang, Manohar Paluri, Yaniv Taigman, Rob Fergus, and Lubomir Bourdev. „Beyond frontal faces: Improving Person Recognition using multiple cues“. In: *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*. IEEE. 2015, pp. 4804–4813 (cit. on p. 1).

[Zho+14] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva. „Learning Deep Features for Scene Recognition using Places Database“. In: *NIPS*. 2014 (cit. on p. 1).

[Zho+15] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. „Object detectors emerge in deep scene cnns“. In: *Proceedings of the International Conference on Learning Representations (ICLR)* (2015) (cit. on pp. 1, 12).

[Zit14] Larry Zitnick. „Computer Vision - StAR Lecture Series: Object Recognition“. In: Presented as part of a Microsoft Research Talk series, 2014.

List of Figures

List of Tables

Colophon

This thesis was typeset with LaTeX 2_ε. It uses the *Clean Thesis* style developed by Ricardo Langner. The design of the *Clean Thesis* style is inspired by user guide documents from Apple Inc.

Download the *Clean Thesis* style at http://cleanthesis.der-ric.de/.