Correlated Noise in Deep Convolutional Neural Networks

by

Shamak Dutta

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Systems Design Engineering

Waterloo, Ontario, Canada, 2019

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

This thesis explores one of the differences between the visual cortex and deep convolutional neural networks, namely, correlated fluctuations of neuron response strength. First, we describe the similarities and differences between biological and artificial neural networks and provide motivation for bridging the gap between the two, with a focus on correlated variability.

Next, we present a regularisation method for convolutional neural networks using correlated noise. The structure of the correlations are inspired from biological observations using two factors: spatial extent and tuning similarity of neurons. We provide empirical results on improved robust image classification in the setting of occluded and corrupted images.

We then move on to studying the connection between popular dataset augmentation techniques and correlated variability. We compute a smooth estimate of the correlations of neural network activity as a function of distance and kernel similarity, and show the similarity to biological correlations.

Finally, we introduce a structured form of Dropout for convolutional neural networks, taking into account spatial and kernel correlations. We show improvement in image classification for the VGG architecture on the CIFAR-10 dataset.

Acknowledgements

The work presented in this document would not have been possible without the help of my advisors Bryan Tripp and Graham Taylor. I appreciate the freedom they gave me to pursue various ideas over the past two years. Their support and guidance has shaped my ability to conduct scientific research.

I thank Mom, Dad, Riteja, and Satya for their unwavering love and support. They have helped me whenever I needed it the most.

Despite being continents apart, Sneha has been extremely supportive and understanding. My time at Waterloo would not have been the same without her.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The past decade has witnessed deep neural networks dominate multiple domains of machine learning. While neural networks were originally inspired from biology, there is much left to be said about its reconciliation with biological brains. In this thesis we develop an understanding of one difference between the two: correlated neuronal variability and how it inspires training strategies for artificial neural networks.

We begin with the first chapter on regularization in convolutional networks using correlated noise models inspired from the brain. Then, we move on to analysing current dataset augmentation techniques for deep learning, connecting it to the structure of biological correlations. We end with the last chapter on a new regularization technique for convolutional networks called CorrDrop.

In the following sections, we describe the connections between biological and artificial neural networks.

1.1 The Intersection of Deep Learning and Computational Neuroscience

Neural networks are a powerful class of algorithms essential to solving many difficult problems in machine learning. Over the past few years, there has been tremendous development in model architectures, hardware, and training procedures that have constantly improved task-related performance. The resurgence of deep learning accelerated with the pioneering work of Alex Krizhevsky, Ilya Sustkever, and Geoffrey Hinton in building AlexNet [22], a

deep convolutional network that outperformed other image recognition algorithms on the challenging ImageNet dataset [4]. Since then, neural networks have greatly improved in object recognition tasks such as localization [11, 38] and classification [12, 33]. However, compared to humans, neural networks are fragile. For example, they misclassify adversarial examples: inputs with imperceptible perturbations [10] that are designed to change a network's response.

Modern neural networks are loosely inspired from neuroscience, capturing some features of biological networks. The basic unit of computation in a neural network is a linear combination of the responses of multiple other units which pass through non-linear functions such as a rectified linear unit (sets negative values to zero). Convolutional neural networks process images in a fashion similar to the visual stream in the brain. The inputs pass through a hierarchical sequence of retinotopic representations where each unit only processes local information from the feature map produced from the previous layer. Along this sequence, the inputs pass through a series of non-linear functions and produce receptive fields of increasing size and invariance. At the same time, convolutional neural networks are also different from biological neural networks. They do not include lateral connections and operate with continuous outputs versus spikes, among other differences. Despite these differences, deep convolutional networks trained on image recognition predict neuron responses in the visual cortex with high accuracy. For example, early layers show Gabor-like receptive fields similar to V1 [46], late layers that are highly predictive of V4 [44], and inferotemporal cortex (IT) responses [45]. These results indicate that convolutional neural networks produce intermediate representations similar to the primate brain to solve the task of object recognition.

While the connections between biological and deep learning are informal, brain-inspired neural networks are revolutionising artificial intelligence. There is rich potential in research of bridging the gap between the two systems.

1.2 Correlated Variability in the Brain

Neural coding is concerned with characterising the relationship between input stimuli and individual or ensemble neuronal responses. In population coding, the joint activities of multiple neurons encode the value of a quantity. For example, as with the control of eye [24] and arm [7] movements, visual discrimination in the primary visual cortex (V1) is much more accurate than would be predicted from the responses from individual neurons [28]. However, neurons are noisy and trial-to-trial fluctuations in response strength make it difficult to discern the relationship between stimuli and neural activity. If the fluctuations

of neuron responses are not correlated with other neuron responses, this problem may be reduced by averaging. The effectiveness of the averaging depends on the pattern of correlations, if any. Unfortunately, noise in the brain is correlated [48, 20, 34] on a range of time scales. The correlations arise from shared input but the involved circuitry relaying this information is unclear [34, 20]. While the information increases linearly with the number of neurons in the case of uncorrelated response variability [26], the impact of correlated variability on stimulus information can be detrimental or beneficial depending on its properties [2] and the decoding method [40]. The majority of previous work on the effect of correlations has studied impact on stimulus information, but stimulus information doesn't fully describe brain function, because information is probably thrown away as behaviorally relevant distinctions are made. The authors in [27] have developed a theory which predicts that perceptual uncertainty is directly encoded by the variability. Our goal is to understand if correlated variability has benefits that can be realized in artificial neural networks by studying its influence on certain tasks.

1.3 Summary of Contributions

The theme of this thesis is understanding the role of correlations (artificially induced or naturally occurring) in convolutional neural networks.

- In chapter 2, we show how convolutional networks can be trained with high-dimensional correlated noise.
  - We introduce convolutional networks, current regularisation methods, back-propagation with stochastic units, and the motivation for robust image classification in section 2.1.
  - In section 2.2, we present four different noise distributions and how sampling can be accomplished while letting the back-propagation algorithm proceed.
  - Inspired from the cortex, we induce a correlation structure that depends on spatial distance between units and their tuning similarity. This is described in section 2.3.
  - The empirical results from image classification experiments on occluded and corrupted images is presented in section 2.4.
- In chapter 3, we explore the connections between the correlations induced by current dataset augmentation techniques and biological correlations.

  - We discuss two forms of dataset augmentation, both in input and latent space for neural networks in section 3.1. The motivation for connecting it to biological correlations is also discussed.
  - In section 3.2, we explain how to compute the activations for determining the correlations. We also show how to make a smooth estimator of correlations as a function of distance and tuning similarity. Finally, we compare the induced correlations across different training paradigms and biological correlations.

- In chapter 4, we introduce a modified version of Dropout [36] for convolutional networks.
  - We explain the motivation for structural forms of Dropout for convolutional neural networks in section 4.1. Two successful methods, SpatialDropout [39] and DropBlock [8], that use spatially structured Dropout are described.
  - Two modifications to SpatialDropout and DropBlock using both spatial and kernel similarity are described and evaluated on image classification using convolutional networks in section 4.2.

Chapter 2

Regularisation using Correlated Noise

We focus on the problem of regularization of artificial neural networks: the process of modifying the learning algorithm to reduce the generalisation error but not the training error. A common approach to regularise deep learning models is to inject noise during training. For example, this can be accomplished by adding or multiplying noise to the hidden units of the model. Most solutions include additive or multiplicative noise because of its simplicity and effectiveness. In this chapter, we show the impact of noise with a correlation structure inspired from biological observations.

One major concern with stochasticity in neural networks is the tendency to break differentiability, which prevents gradient computation via back-propagation. Depending on the noise distribution, one may ignore the stochasticity in back-propagation (e.g. the straight-through estimator in the case of binary stochastic neurons [3]), or one may apply computationally convenient estimators for non-differentiable functions. We show how to sample from correlated multivariate Gaussian and Poisson distributions while keeping the procedure differentiable, so that back-propagation can proceed as usual.

2.1 Introduction

While the field of deep learning is broad and extensive [9], we have described in the following sections the topics which we believe are relevant to the understanding of our contributions in regularisation using correlated noise.

2.1.1 Convolutional Neural Networks

The goal of a neural network is to approximate some function f^*. For example, an image classifier, $y = f(\mathbf{x})$, maps an input $\mathbf{x}$ to a category y. Neural networks process inputs by composing together many different functions. Each function is typically a linear transformation followed by a non-linearity. By composing several functions, the 'depth' of the model is increased, giving rise to the name 'deep' learning.

Convolutional networks, or CNNs, are a type of neural network which uses the convolution operation. The convolution is a special kind of linear operation. In this thesis, we will focus on convolutions applied to 2D images. In this context, the convolution operation applies a two-dimensional kernel at different image locations. We can define the convolution for a two-dimensional image I using a two-dimensional kernel K as:

$$C[i,j] = (I * K)[i,j] = \sum_m \sum_n I[m,n]K[i-m, j-n], \tag{2.1}$$

where m and n are the dimensions of the image.

The sparse connectivity and weight sharing properties of a convolutional neural network make them more efficient than the fully connected neural network counterparts. By enforcing local computations using the same weights, meaningful features about the input can be calculated while reducing memory requirements. This makes CNNs especially suitable for image processing. The learning algorithm that guides parameter updates is the same as in any deep neural network: gradient descent. It employs the backpropagation algorithm to efficiently compute gradients with respect to the parameters [32].

2.1.2 Regularisation using Uncorrelated Noise

The analysis of noise in deep networks has focused on models that use uncorrelated noise to perturb the activations of neurons. For example, Dropout [36] is an effective method to regularize neural networks where each unit is independently dropped with a fixed probability. Uncorrelated Gaussian noise has also been extensively explored in [31], where noise is added to the input, or before or after the non-linearity in other layers. The authors connect different forms of independent noise injection to certain forms of optimization penalties in a special form of a denoising autoencoder. One connection shown is zero-mean additive Gaussian noise with variance equal to the unit activation (Fano factor of 1) relates to a penalty that encourages sparsity on hidden unit activations. This is interesting because the time intervals between spikes in a cortical neuron are irregular and can be modeled

using a Poisson process, such that the variance of the spike count in a small time interval is equal to the mean spike count. This motivates us to investigate whether we can model artificial neurons as samples from a Poisson distribution. The work done in [31] focused on unsupervised learning in autoencoders and used the learned hidden representations as inputs to a classifier. Our approach differs in the fact that we are directly using a supervised classifier (CNN) to analyze the injection of noise.

2.1.3 Back-Propagation Through Stochastic Nodes

Gradient-based learning which leverages back-propagation in neural networks requires that all operations that depend on the trainable parameters be differentiable. Stochastic neural networks usually involve samples from a distribution on the forward pass. However, the difficulty is that we cannot back-propagate through the sampling operation. It is shown in [3] that in the case of injection of additive or multiplicative noise in a computational graph, gradients can be computed as usual. The concept of a straight-through estimator is also introduced in [3], where a copy of the gradient with respect to the stochastic output is used directly as an estimator of the gradient with respect to the non-linear operator. The work in [25, 18] allows for back-propagation through samples from discrete categorical distributions. While we do not use this technique in our work, it is possible to choose an upper bound K to convert a Poisson distribution to a categorical distribution of size K. The sample from the distribution would be the expected value of this estimated categorical distribution and back-propagation can proceed since the entire process is made differentiable. This is explained in further detail in subsection 2.2.3.

2.1.4 Robust Image Classification

Robust image classifiers possess the desirable ability to remain resilient even in the face of corruption, adversarial examples, or abstract changes in structure and style. This requires the ability to robustly model invariance to certain transformations of the input. The primary method of gaining invariance to transformations is data driven, either by attempting to collect instances of the object under a wide variety of conditions, or augmenting the existing dataset via synthetic transformations. The authors in [43] suggest that the right way to learn invariance is not by adding data, as corruptions, perturbations, and occlusions follow a long-tail distribution which cannot be covered, even by large-scale efforts in collecting data. This motivates the need to modify the structure of the network to learn invariance. This serves as motivation to see if correlated variability in convolutional neural networks has benefits in robust image classification.

2.1.5 Connections to Data Augmentation

Dataset augmentation is a cheap and effective way to generate more training data with variability that is expected at inference time. Recently, some works have considered augmentation techniques, such as the addition of noise, as well as interpolation and extrapolation from pairs of examples, not in input space, but in a learned feature space [5]. However, they show that simple noise models (e.g. Gaussian) do not work effectively when compared to extrapolation. We are motivated by the fact that more sophisticated noise models (e.g. correlated) could be useful for feature space-based augmentation. The connections between input space data augmentation [47] and latent space data augmentation [41] with correlated activations in CNNs is explained in more detail in chapter 3.

2.2 High Dimensional Noise in Deep Neural Networks

We consider noise sampled from two distributions: Gaussian and Poisson. In this section, we demonstrate sampling from these distributions (uncorrelated and correlated) while allowing gradient computation to proceed as usual to enable back-propagation.

2.2.1 Uncorrelated Gaussian Noise

We consider $\mathrm{h_i}$ to be the output of a stochastic neuron. The output is a deterministic function of a differentiable transformation a_i and a noise source z_i, as considered in [3]. a_i is typically a transformation of its inputs (vector output of other neurons) and trainable parameters (weights and biases of a network). The output of the stochastic neuron is:

$$\mathrm{h_i} = f(a_i, \mathrm{z}_i). \tag{2.2}$$

As long as $\mathrm{h_i}$ is differentiable with respect to a_i and has a non-zero gradient, we can train the network with back-propagation. In this section, one form of noise we consider is additive independent Gaussian noise with zero mean and σ^2 variance, where:

$$f(a_i, \mathrm{z}_i) = a_i + \mathrm{z}_i, \tag{2.3}$$

$$\mathrm{z}_i \sim \mathcal{N}(0, \sigma^2). \tag{2.4}$$

At test time, we can compute the expectation of the noisy activations by sampling from the distribution $\mathcal{N}(a_i, \sigma^2)$; however, this can be computationally expensive for large

datasets. Instead, we can approximate the expectation by scaling the units by their expectation, as in Dropout [36]. Since we are using zero mean additive noise, no scaling is required, as the expectation does not change.

A special case of Equation 2.4 is when $\sigma^2 = a_i$. The distribution of activations for a specific stimulus will follow $\mathcal{N}(a_i, a_i)$, which has a Fano factor of 1. This is similar to biological neurons, which exhibit Poisson-like statistics with a Fano factor of approximately 1. It also means that z_i is now a function of a_i and the gradient of z_i with respect to a_i exists through re-parameterization. A sample from a normal distribution $\mathcal{N}(\mu, \sigma^2)$ can be constructed as:

$$\begin{aligned} \mathrm{x}_i &\sim \mathcal{N}(0, 1), \\ \mathrm{z}_i &= \mu + \sigma \mathrm{x}_i. \end{aligned} \tag{2.5}$$

In the case when $\sigma^2 = a_i$ and $\mu = 0$, $\mathrm{z}_i = \sqrt{a_i}\mathrm{x}_i$. In practice, the gradients can be unstable for the square-root function for small values. To solve this problem, we add a small value of $\epsilon = 1e - 6$ to the argument of the square-root function.

2.2.2 Correlated Gaussian Noise

We consider $\mathbf{h} \in \mathbb{R}^{\mathrm{n}}$ to be a vector of outputs of n stochastic neurons. $\mathbf{h}$ is a sum of a noise vector $\boldsymbol{z} \in \mathbb{R}^{\mathrm{n}}$ and a vector output $\boldsymbol{a} \in \mathbb{R}^{\mathrm{n}}$, which is a differentiable transformation of its inputs and trainable parameters. The vector output is:

$$\mathbf{h} = \mathbf{z} + \boldsymbol{a}, \tag{2.6}$$

$$\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}). \tag{2.7}$$

Given a desired mean $\boldsymbol{\mu}$ and correlation matrix $\boldsymbol{\Sigma}$, $\mathbf{z} \in \mathbb{R}^{\mathrm{n}}$ is sampled as follows:

- Sample $\mathbf{X} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}_n)$
- Compute the Cholesky decomposition: $\boldsymbol{\Sigma} = LL^*$, where L^* is the conjugate transpose of L.
- $\mathbf{z} = \boldsymbol{\mu} + L\mathbf{X}$

- If $\boldsymbol{\sigma} \in \mathbb{R}^{\mathrm{n}}$ is the desired standard deviation, then

$$\mathbf{z} = \mathrm{diag}(\boldsymbol{\sigma})(\boldsymbol{\mu} + L\mathbf{X}), \tag{2.8}$$

  where $\mathrm{diag}(\boldsymbol{\sigma}) \in \mathbb{R}^{\mathrm{n}\times\mathrm{n}}$ is a matrix with the standard deviations on the diagonal and zeros elsewhere.

2.2.3 Uncorrelated Poisson Noise

We consider $\mathrm{h_i}$ to be the output of a stochastic neuron. The definition of a_i is the same as in Section 2.2.1. In the case of independent Poisson noise, the output of the neuron is:

$$\mathrm{h}_i \sim \mathrm{Poisson}(\lambda = a_i), \tag{2.9}$$

where the mean is given by a_i. This poses a problem in back-propagation, as there is no gradient of h_i with respect to the parameter $\lambda = a_i$. The re-parameterization trick in [19] cannot be applied in this case because the Poisson distribution is discrete. To avoid this problem, we set the output of the unit to its mean value during the backward pass, which is a_i. We still propagate the sample from the distribution on the forward pass. This is similar to the straight-through estimator for back-propagation through stochastic binary neurons [3].

Continuous Relaxation for Sampling from Poisson Distributions

The Gumbel-softmax method [25, 18] can be used here if the Poisson distribution is converted to a categorical distribution using an upper threshold K. While we do not use this technique in our work, it might be of interest to the reader. The procedure is:

1. Given a rate $\lambda \in \mathbb{R}$ of a Poisson distribution, the K-categorical distribution is constructed as:

$$z = [\pi_1 = P(X = 0; \lambda), \pi_2 = P(X = 1; \lambda), ..., \pi_k = P(X = K; \lambda)] \in \mathbb{R}^{\mathrm{k}}. \tag{2.10}$$

   This is an approximation as we are using the probability mass function of the Poisson distribution of rate λ to determine the individual probabilities of a K-categorical

distribution. The ideal method would be to determine the pmf of the truncated Poisson distribution $[P(X = 0|X \leq K; \lambda), P(X = 1|X \leq K; \lambda), ..., P(X = K|X \leq K; \lambda)]$. If we choose K to be much larger than λ, the subsequent probabilities $P(X = K + 1; \lambda)$, $P(X = K + 2; \lambda)$, ..., $P(X = K + N; \lambda)$ will be small.

2. Using the Gumbel-Softmax distribution and a temperature parameter τ, a one-hot sample $y \in \mathbb{R}^K$ from the categorical distribution z can be generated using the softmax function:

$$y_i = \frac{\exp\left(\left(\log\left(\pi_i\right) + g_i\right)/\tau\right)}{\sum_{j=1}^{k} \exp\left(\left(\log\left(\pi_j\right) + g_j\right)/\tau\right)} \quad \text{for } i = 1, \ldots, K, \tag{2.11}$$

where $g_1, \ldots, g_k$ are samples drawn from Gumbel(0,1). The Gumbel(0,1) distribution can be sampled by drawing $u \sim$ Uniform(0,1), and computing $g = -\log(-\log(u))$.

3. We discretize y on the forward pass using arg max, but use the continuous approximation (using the softmax) on the backward pass. The approximate sample $p \in \mathbb{R}$ from the Poisson distribution is:

$$p = \arg\max_i\{y\} \tag{2.12}$$

Depending on the rate λ of the target Poisson distribution, the threshold K for the categorical distribution has to be chosen carefully. If it is too small, it will not capture the distribution well. The dependence on K is shown in Figure 2.1.

2.2.4 Correlated Poisson Noise

Similar to subsection 2.2.2, $\mathbf{h} \in \mathbb{R}^n$ is a vector of outputs of n stochastic neurons. $\mathbf{h}$ is a sample from a correlated Poisson distribution of mean $\boldsymbol{\lambda}$ and correlation matrix $\boldsymbol{\Sigma}$,

$$\mathbf{h} \sim \text{Poisson}(\boldsymbol{\lambda}, \boldsymbol{\Sigma}). \tag{2.13}$$

We draw approximate samples from this distribution in the following way. Given a desired mean $\boldsymbol{\lambda} \in \mathbb{R}^n$ and correlation matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{n \times n}$:

- Sample $\mathbf{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $\boldsymbol{\mu}$ is chosen arbitrarily.

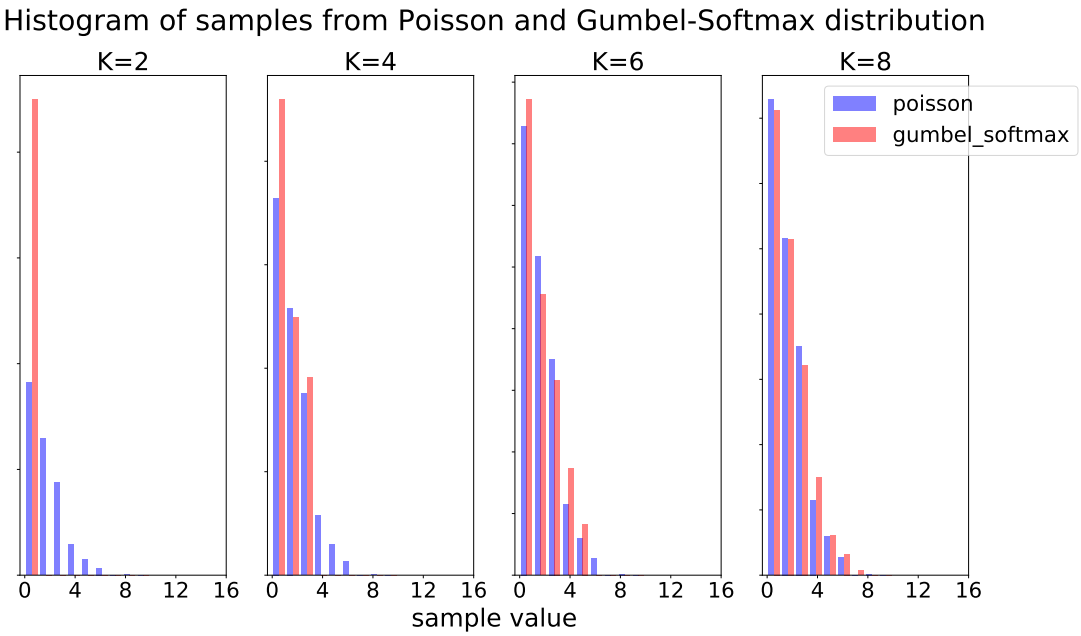


Figure 2.1: Histogram of 10000 samples from the Poisson with $\lambda = 2$ and the approximate Poisson samples of Gumbel-Softmax distribution for different values of K. When K is small, the approximation is poor. As the value of K increases, the approximation quality increases.

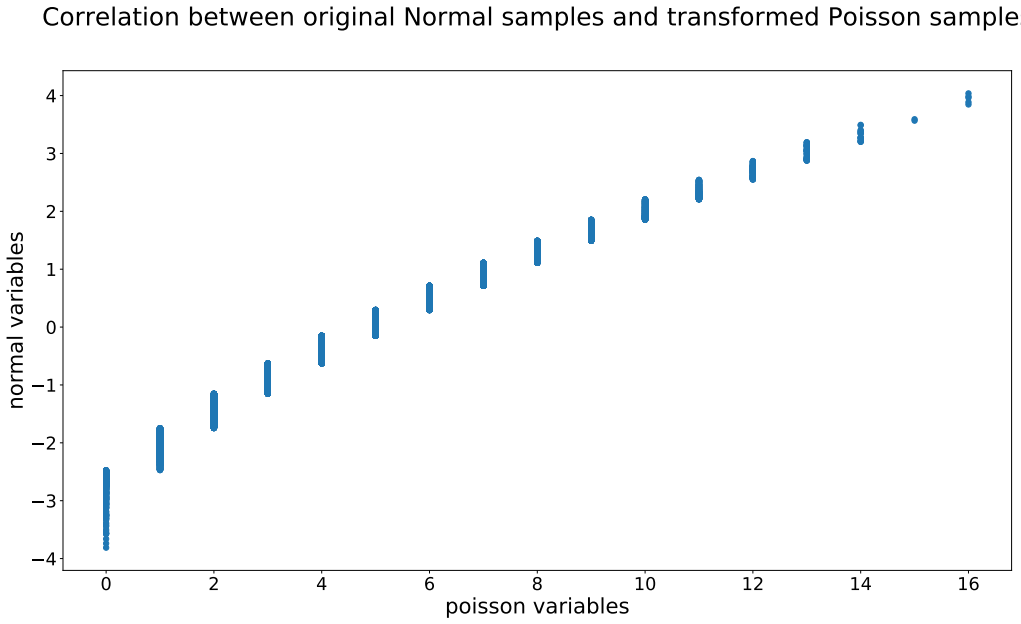


Figure 2.2: The correlations between samples from the normal distribution and transformed Poisson variables. The correlations are quite similar after the transformation. However, the correlations will not be exact because any transformation from the original will reduce the correlations between the variables.

- Apply the univariate normal cumulative distribution function (CDF): $\boldsymbol{Y} = F_x(\mathbf{X}; \mu = 0, \sigma = 1)$.
- Apply the quantile (inverse CDF) of the Poisson distribution: $\mathbf{z} \sim F_z^{-1}(\boldsymbol{Y}; \boldsymbol{\lambda})$.

This sampling procedure is non-differentiable for a correlated Poisson. We estimate the gradient using the mean $\boldsymbol{\lambda}$ on the backward pass, similar to the straight through estimator in [3]. The actual correlations between Poisson variables depend on the rates $\boldsymbol{\lambda}$ and are smaller than the desired correlation matrix $\boldsymbol{\Sigma}$ as shown in Figure 2.2.

2.3 Correlations Inspired from the Visual Cortex

We discuss how the correlation matrix $\boldsymbol{\Sigma}$ is constructed, which is used in subsection 2.2.2 and subsection 2.2.4. Note that the neuron equivalent in a CNN is a unit in the output

feature map of a given layer.

Neurons in the cortex are correlated in their stochasticity. This correlation is a function of the spatial spread and tuning similarity [34]. In the visual cortex, correlations have been studied between neurons in the same visual area. Analogously, we consider correlations between units in the same layer of a CNN. The details of spatial similarity and tuning similarity are described in subsection 2.3.1 and subsection 2.3.2. For a given layer in a convolutional network, if the width and height of the feature maps, as well as the number of feature maps are defined as w, h, and k, respectively, then the dimension d of the correlated distribution we draw samples from is $d = whk$ and the correlation matrix is $\mathbf{\Sigma} \in \mathbb{R}^{\text{whk}\times\text{whk}}$.

Similar to a relationship suggested in [34], the correlation between two neurons, x_1 and x_2, is determined as:

$$f(x_1, x_2) = [a - b(d(x_1, x_2))]^+ \cdot e^{\frac{k(x1,x2)-1}{\tau}} + c, \tag{2.14}$$

where $[\cdot]^+$ is $\max(\cdot, 0)$, $d(\cdot, \cdot)$ is a function that returns the scaled Euclidean distance between two neurons, $k(\cdot, \cdot)$ is a function that returns the tuning similarity between two neurons (bounded in $[-1, 1]$), and a, b, c, and τ are hyper- parameters. The correlation is summarized in Figure 2.3 for specific values of a, b, c, and τ. To summarize, within a layer of a CNN, every neuron is correlated to every other neuron in the same layer by a value determined by how far apart they are and their tuning similarity.

2.3.1 Spatial Similarity

The spatial similarity between two units in any output feature map within a layer is determined by the Euclidean distance between the coordinates of the neurons within their respective feature maps. The spatial distance between two neurons that are a part of the i^{th} and j^{th} feature maps, respectively, with coordinates $\boldsymbol{p} = (x_i, y_i)$ and $\boldsymbol{q} = (x_j, y_j)$ is:

$$d(\boldsymbol{p}, \boldsymbol{q}) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}, \tag{2.15}$$

Since the dimensions of the feature map do not change as training progresses, we can pre-compute the spatial correlations for all pairs of neurons before training begins.

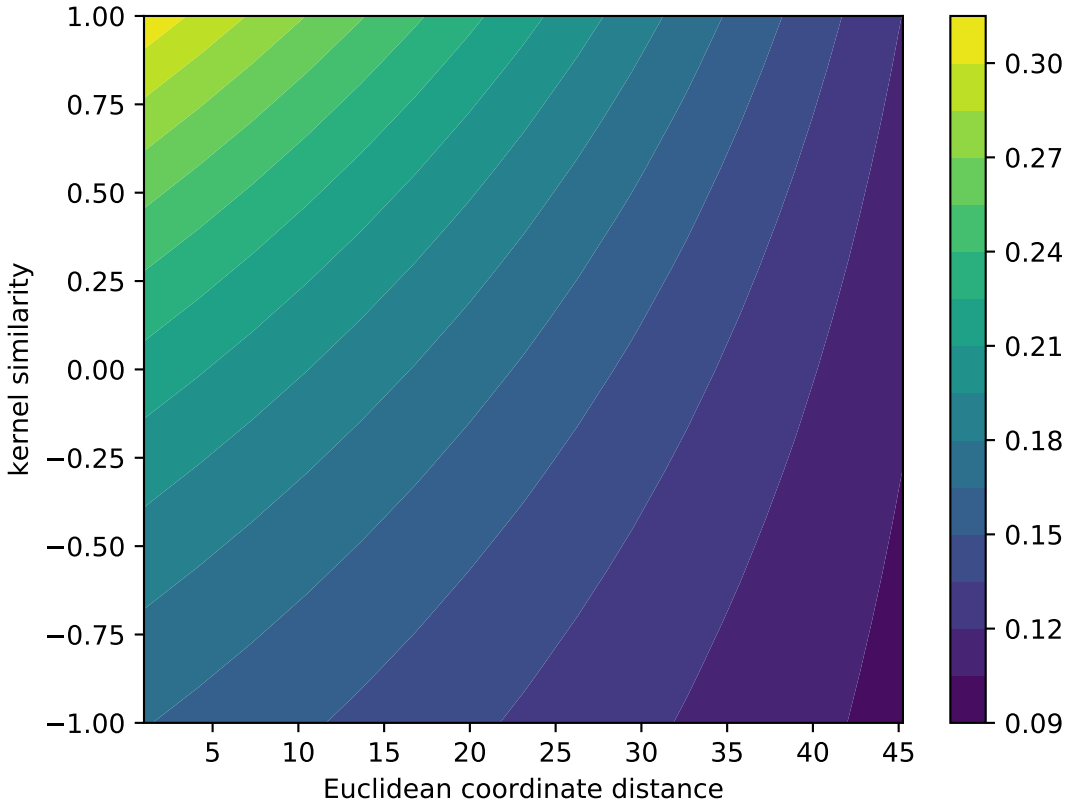


Figure 2.3: Correlation dependence on distance and tuning similarity for $a = 0.225$, $b = 0.0043$, $c = 0.09$ and $\tau = 1.87$. The correlation value of the color plot is indicated by the bar on the right.

2.3.2 Tuning Similarity

The tuning similarity between two units in any output feature maps within a layer is determined by the cosine similarity of the normalized weight matrices that produced them. Consider a weight tensor (kernel in a convolutional network) of dimension $d = k \times k \times m \times n$, where k is the kernel size, m is the number of input channels from the previous layer, and n is the number of kernels. The i^{th} kernel $\boldsymbol{w_i}$ is of dimension $d = kkm \times 1$. The tuning similarity between two neurons that are a part of the i^{th} and j^{th} feature maps, respectively, with coordinates $\boldsymbol{p} = (x_i, y_i)$ and $\boldsymbol{q} = (x_j, y_j)$ is:

$$k(\boldsymbol{p}, \boldsymbol{q}) = \left(\frac{\boldsymbol{w_i}}{\|\boldsymbol{w_i}\|} \right)^T \cdot \frac{\boldsymbol{w_j}}{\|\boldsymbol{w_j}\|}, \tag{2.16}$$

since we are calculating the cosine similarity, $k(\cdot, \cdot) \in [-1, 1]$. Note that the tuning similarity solely depends on the feature maps that the output units are a part of.

2.3.3 Nearest Positive Definite Correlation Matrix

A valid correlation matrix is a symmetric matrix with a unit diagonal and non-negative eigenvalues. Following the procedure in Equation 2.14, the computed correlation matrix may not be valid due to the possibility of negative eigenvalues. In this scenario, we are interested in finding the nearest correlation matrix in Frobenius norm to the computed correlation matrix.

Let $\boldsymbol{\Sigma}$ be the invalid target matrix computed as per section 2.3. If B is the valid correlation matrix we need to find, we require the eigenvalues of B to be positive. We can formulate this as a convex optimisation problem to solve for the closest possible B to $\boldsymbol{\Sigma}$ shown in Equation 2.17.

$$\begin{array}{ll} \text{minimize} & \|\boldsymbol{\Sigma} - B\|_F \\ \text{subject to} & B \succ 0 \end{array} \tag{2.17}$$

We use the CVXPY [6] software package to solve this optimisation problem. It should be noted that CVXPY relaxes the condition for B to be positive semi-definite. In practice, the method has converged to solutions which are positive-definite but in some cases the optimum may end up being positive semi-definite.

2.4 Image Classification Experiments

We first describe the experimental setup including the dataset, model architecture, and robustness evaluation criteria. Then, the quantitative results are discussed in the following sections.

Model name	abbreviation
AllConvNet baseline	Baseline
AllConvNet uncorrelated Gaussian $\sigma = a_i$	UG
AllConvNet correlated Gaussian	CG
AllConvNet uncorrelated Poisson	UP
AllConvNet correlated Poisson	CP

Table 2.1: Different AllConvNet models evaluated as part of layer 1 tests.

2.4.1 Model Architecture

We performed preliminary experiments with the noise models described in section 2.2 using an architecture equivalent to the AllConvnet network [35] (specifically the All-CNN-C architecture) with one exception: the first layer contains 10 feature maps instead of 96, as shown in Table 2.2. This was done for computational tractability, as the correlation matrix grows as $(whk)^2$, where w and h are the width and height of a feature map, and k is the number of feature maps in a layer. As a result, the sampling procedure from a correlated distribution can slow training by a large amount. For example, typical deep convolutional networks have feature maps whose number varies in the range [256,1024]. In the early layers where feature maps are larger in size, the correlation matrix would be of size $(32 \times 32 \times 256)^2 = 262144^2$.

2.4.2 Dataset

The CIFAR-10 dataset [21] is used for the experiments in this section. It consists of 60000 32×32 colour images in 10 classes, with over 6000 images per class. There are 50000 training images and 10000 test images. Neither the training set nor the test set contains any occlusions or corruptions. While we do not aim for state-of-the-art results, our goal is to obtain a model that achieves respectable performance on CIFAR-10 and analyze the effect of adding various kinds of noise. The model with the best validation set (10000 images randomly picked from the training set) accuracy is saved for evaluation on the occlusions and common corruptions dataset.

Input: 32×32 RGB image
Layer 1: 3×3 conv. 10 filters, ReLU
Layer 2: 3×3 conv. 96 filters, ReLU
Layer 3: 3×3 conv. 96 filters, stride $= 2$, ReLU
Layer 4: 3×3 conv. 192 filters, ReLU
Layer 5: 3×3 conv. 192 filters, ReLU
Layer 6: 3×3 conv. 192 filters, stride $= 2$, ReLU
Layer 7: 3×3 conv. 192 filters, ReLU
Layer 8: 1×1 conv. 192 filters, ReLU
Layer 9: 1×1 conv. 10 filters, ReLU
Layer 10: global averaging over 6×6 spatial dimensions
10-way softmax

Table 2.2: Modified version of AllConvnet (ALL-CNN-C) architecture with 10 filters in layer 1 used in experiments with CIFAR-10.

2.4.3 Occlusions

The first set of robustness experiments involves evaluating the model on occluded images from the test set. These occlusions may not fully depict real world scenarios, but serve as a good preliminary tests for classifier robustness. The different types of occlusions are listed in Figure 2.4. Each of the six types of occlusions is applied to every test image, giving us a new test set of 60000 images.

We first experiment by incremental addition of the stochastic behaviour to understand its effect. The baseline model is described in Table 2.2. First, we add noise in layer 1 of the AllConvNet architecture. The different noise architectures for layer 1 are shown in Table 2.1.

We then analyze whether the benefits of noise can be realized in the presence of another regularizer by evaluating the effect of noise in the presence of Dropout. In this case, the baseline model shown in Table 2.2 is trained with Dropout in specific layers. As before, we add different types of noise to layer 1 in order to understand its impact.

The experiments in this section were implemented in TensorFlow [1]. The optimization algorithm used for parameter updates is stochastic gradient descent with Nesterov momentum [30, 37]. The models are trained for 350 epochs with a learning rate of 0.01, decaying at steps 200, 250, and 300 by a factor of 0.1 each time. An L2 weight decay factor of $1e-3$ is also used as part of the objective function.

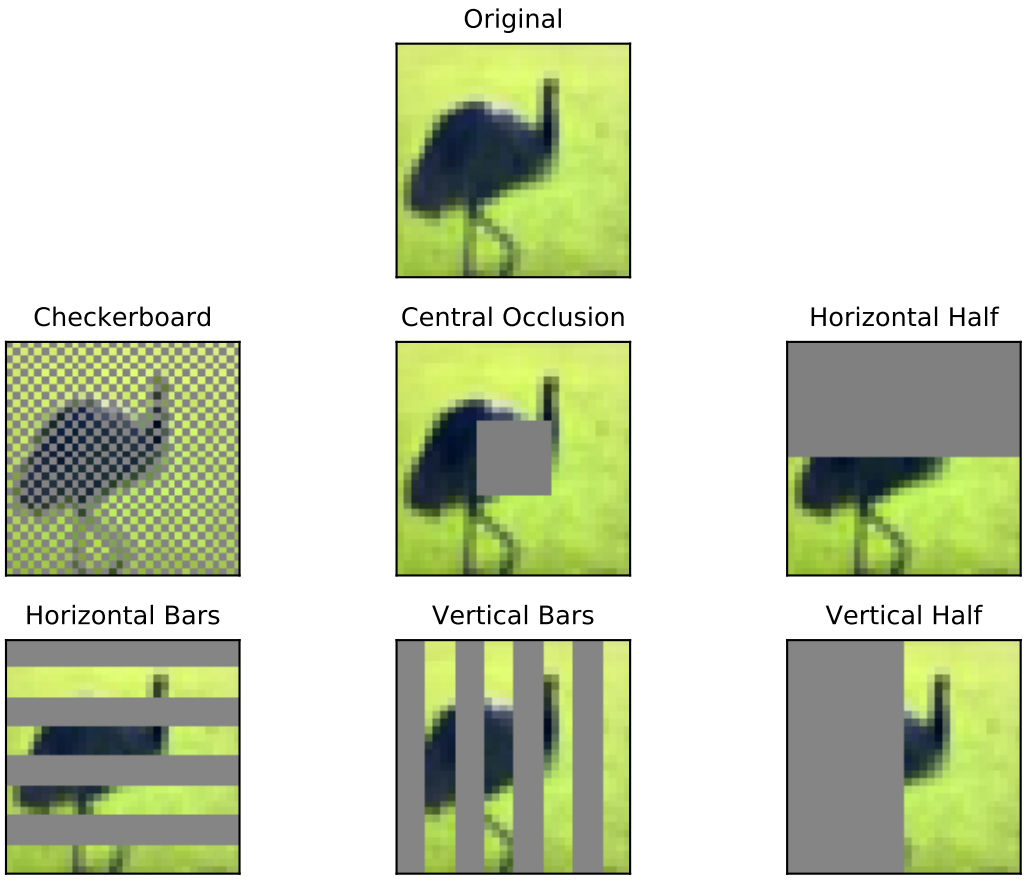


Figure 2.4: Different types of occlusions used to evaluate recognition performance. Images shown are taken from CIFAR-10.

Absence of Dropout

The classification performance of the models that do not incorporate Dropout are summarized in Table 2.3. The models are abbreviated, as shown in Table 2.1. All models are evaluated against the baseline.

Uncorrelated Gaussian noise with zero mean and standard deviation equal to the activation values and uncorrelated Poisson noise, individually, improve classification performance on two out of the six occlusion tests, while having worse test set performance.

Correlated Gaussian noise improves classification performance in three out of the six

occlusion tests but also has comparable performance in the remaining occlusion tests and the test set. This is encouraging because the model achieves better performance on a held out transformed dataset without sacrificing the original test set accuracy.

Correlated Poisson noise also improves classification performance in three out of the six occlusion tests but has worse performance in the remaining tests. The weak performance of the Poisson models (both correlated and uncorrelated) may stem from the fact that the mean value is used during back-propagation instead of the actual sample value which might affect learning.

In the case of models trained without additional regularisation (Dropout), in five out of the six occlusion tests, the best performing models involved correlated noise.

	Baseline		UG		UP		CG		CP	
	mean (%)	std (%)	mean (%)	std (%)	mean (%)	std (%)	mean (%)	std (%)	mean (%)	std (%)
Test Set	**83.6**	0.7	80.7	0.5	81.1	1.5	82.5	1.7	75.8	0.0
Central Occlusion	64.0	0.7	63.1	0.2	63.6	0.4	**65.0**	1.2	61.0	0.0
Checker Board	54.1	1.7	69.7	1.8	72.5	1.7	**75.7**	2.4	73.4	0.0
Horizontal Bars	23.2	1.3	26.4	1.5	27.4	1.5	28.5	1.7	**30.6**	0.0
Vertical Bars	20.9	1.7	20.8	1.2	20.6	1.0	21.8	0.8	**23.1**	0.0
Horizontal Half	**49.2**	1.8	42.5	1.3	44.9	1.7	48.9	1.7	43.5	0.0
Vertical Half	40.3	1.2	35.7	1.8	38.9	1.1	**45.0**	0.7	35.0	0.0

Table 2.3: Image classification performance of AllConvNet without any additional regularization on occlusions averaged over 5 runs.

Presence of Dropout

Dropout was applied to the input image, as well as after each of the layers that has a stride of 2 (simply a convolutional layer that replaces pooling [35], specifically layer 3 and 6). The Dropout probability at the input was 20% and was 50% otherwise.

The classification performance of the models that incorporate Dropout are summarized in Table 2.4, with the model abbreviations shown in Table 2.1.

Adding Dropout to the baseline model improves the test set accuracy by 0.7% from the model trained without any Dropout. It improves the accuracy on three occlusion tests but also reduces accuracy on the other three occlusions. This suggests that Dropout may not be the best regularisation method for robust image classification.

Uncorrelated Gaussian noise paired with Dropout improves accuracy on four out of the six occlusion tests, but with lower test set accuracy. This suggests that uncorrelated

Gaussian noise by itself may not be a good technique for robust image classification, but when paired with Dropout, it has desirable results. Uncorrelated Poisson noise also has the same behaviour as uncorrelated Gaussian noise.

Correlated Gaussian noise paired with Dropout improves classification performance for five out of six occlusion tests while having comparable accuracy on the test set. Correlated Poisson noise has significantly weaker test set accuracy, but still improves accuracy on two out of six occlusion tests.

In the case of models trained with additional regularisation (Dropout), in five out of the six occlusion tests, the best performing models involved correlated noise.

	Baseline		UG		UP		CG		CP	
	mean (%)	std (%)	mean (%)	std (%)	mean (%)	std (%)	mean (%)	std (%)	mean (%)	std (%)
Test Set	**84.3**	0.6	83.1	0.2	83.1	0.1	84.1	0.1	76.3	0.0
Central Occlusion	65.1	1.4	67.0	0.5	67.1	0.3	65.9	0.1	**70.9**	0.0
Checker Board	73.2	1.5	68.6	1.2	69.8	0.7	**78.0**	1.3	70.9	0.0
Horizontal Bars	21.7	0.5	25.7	1.7	25.5	1.4	25.7	1.5	**32.9**	0.0
Vertical Bars	22.9	1.3	**24.1**	1.1	23.6	0.9	**24.1**	1.3	22.1	0.0
Horizontal Half	**41.2**	1.0	38.6	1.4	39.8	0.7	38.5	1.5	40.3	0.0
Vertical Half	34.9	1.2	37.6	1.0	38.6	1.8	**40.4**	0.0	25.2	0.0

Table 2.4: Image classification performance of AllConvNet with additional regularization (Dropout) on occlusions averaged over 5 runs.

2.4.4 Common Corruptions

The authors in [13] have created a dataset called CIFAR-10-C, a set of 95 common visual corruptions applied to the CIFAR-10 test set. This new dataset serves as a strong benchmark to evaluate performance on image corruptions. The 15 main corruptions are shown in Figure 2.5 with 4 additional types shown in Figure 2.6. Each corruption has 5 levels of varying severity creating a new test set of 950000 images.

The experiments in this section are run using PyTorch [29]. The models are trained for 350 epochs with a learning rate of 0.01, decaying at steps 200, 250, and 300 by a factor of 0.1 each time. An L2 weight decay factor of $1e-3$ is also used as part of the objective function.

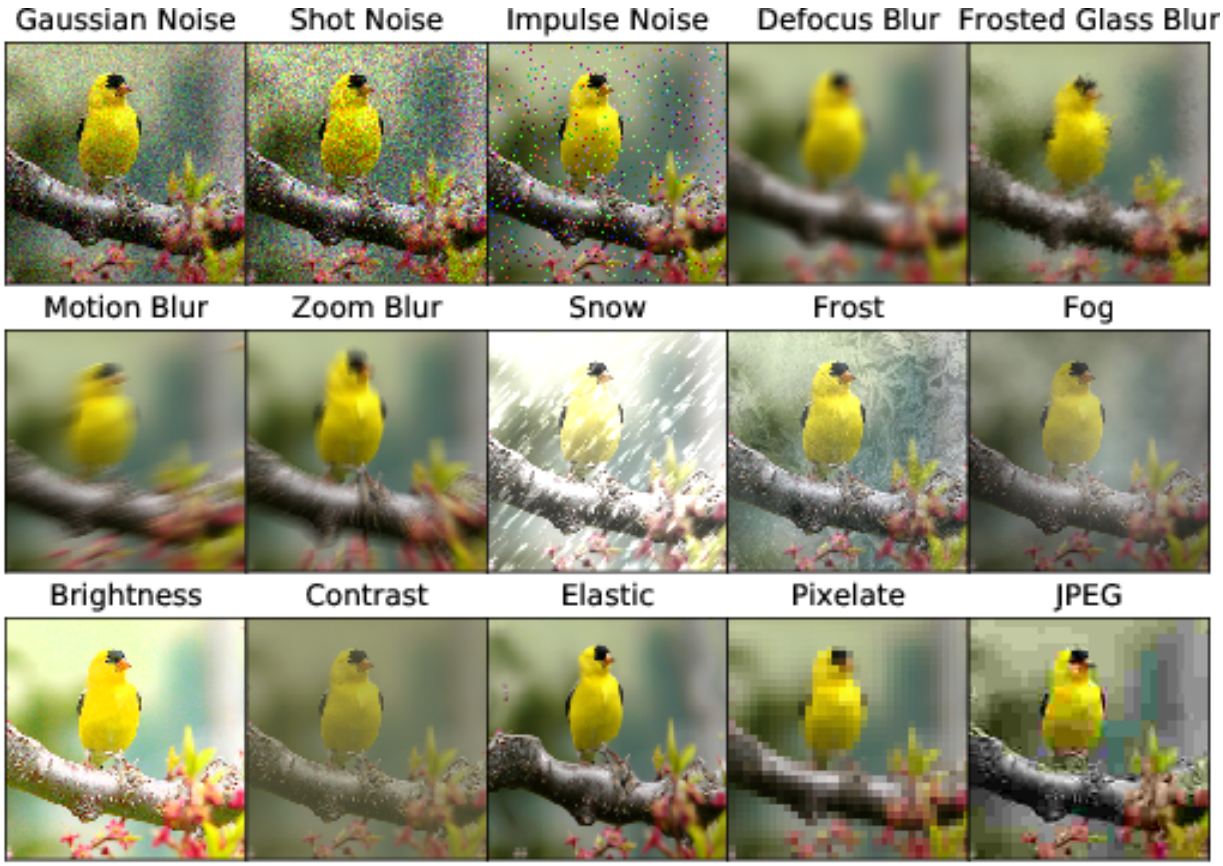


Figure 2.5: CIFAR-10-C dataset consists of 15 main types of corruptions spanning various categories. Each category has 5 levels of varying severity resulting in 75 distinct corruptions. This image is taken from ImageNet-C as a representative example [13].

Absence of Dropout

The classification performance of the models that do not incorporate Dropout are summarized in Table 2.5. The models are abbreviated, as shown in Table 2.1.

The baseline model has the best image classification accuracy for two out of the nineteen corruptions. Uncorrelated Gaussian models do increase performance on certain corruptions, but has worse test set accuracy. Uncorrelated Poisson models have the best accuracy across all models on five corruptions, with much lower test set accuracy.

Correlated Gaussian noise has the best accuracy on fourteen out of nineteen corruptions while having comparable test set accuracy to the baseline. In fact, the improvement from

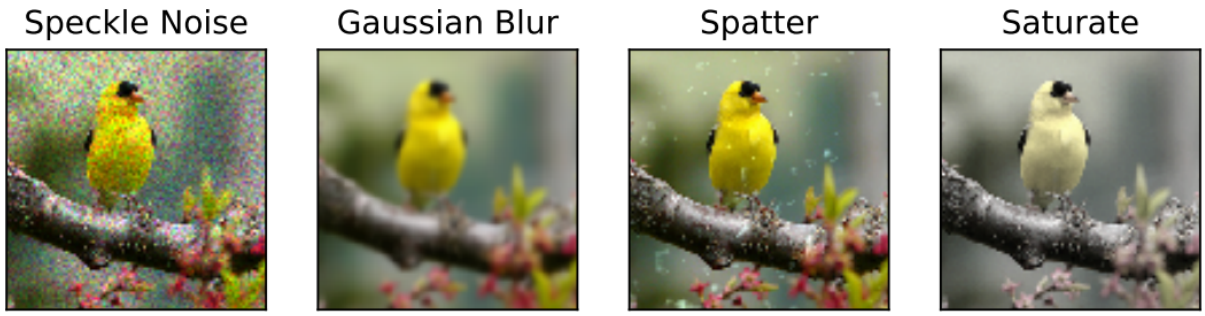


Figure 2.6: The extra 4 corruptions types for CIFAR-10-C. With 5 varying levels of severity, this gives 20 additional distinct corruptions. This image is taken from ImageNet-C as a representative example [13].

the baseline is on seventeen out of nineteen corruptions. This suggests that test set classification and robust image classification are not necessarily competing objectives. While correlated Poisson noise has significantly worse test set accuracy compared to the baseline, it still improves accuracy on seven corruption tests.

In the case of models trained without additional regularisation (Dropout), in fifteen out of the nineteen corruptions tests, the best performing models involved correlated noise.

Presence of Dropout

Dropout was applied to the input image, as well as after each of the layers that has a stride of 2 (simply a convolutional layer that replaces pooling [35], specifically layer 3 and 6). The Dropout probability at the input was 20% and was 50% otherwise.

The classification performance of the models that incorporate Dropout are summarized in Table 2.6, with the model abbreviations shown in Table 2.1.

Introducing Dropout into the baseline model provides mixed benefits for classification of the corrupted images: in some cases it performs better and in others, it does not. However, Dropout does seem to stabilise the training of these networks and reduce the variance in the performance across runs, which is important.

Correlated Gaussian noise has the best accuracy across fourteen corruption tests while having comparable test set accuracy and on the remaining corruption tests. The other models do not seem to provide any significant benefits in test set or the corruptions set.

	Baseline		UG		UP		CG		CP	
	mean (%)	std (%)	mean (%)	std (%)	mean (%)	std (%)	mean (%)	std (%)	mean (%)	std (%)
Test Set	**83.6**	0.7	80.7	0.5	81.1	1.5	82.5	1.7	75.7	1.2
Brightness	**81.9**	0.7	78.1	1.1	78.5	1.0	81.4	1.2	70.8	1.2
Contrast	58.6	2.0	55.0	0.2	56.0	0.5	**63.4**	0.7	52.2	0.6
Defocus Blur	73.4	1.3	74.2	0.2	75.5	0.9	**76.6**	0.7	71.1	1.1
Elastic	72.7	0.8	72.0	0.1	73.0	0.6	**73.9**	1.0	67.8	0.8
Fog	72.4	1.7	67.7	0.6	68.3	0.2	**73.5**	1.9	59.5	2.1
Frost	73.0	1.6	72.6	0.9	73.0	1.2	**75.5**	1.3	64.6	1.7
Gaussian Blur	68.1	1.6	71.1	0.5	72.8	1.4	**73.8**	0.6	69.3	1.8
Gaussian Noise	67.8	2.5	75.6	0.6	**76.4**	0.9	76.1	0.4	74.4	0.4
Glass Blur	58.5	2.4	68.5	1.1	**70.0**	1.5	68.1	1.7	69.6	1.9
Impulse Noise	63.4	2.2	70.1	1.3	71.9	1.6	71.0	1.5	**72.0**	2.3
JPEG Compression	78.6	1.0	78.9	0.1	79.4	0.6	**80.5**	1.2	74.2	1.2
Motion Blur	64.6	1.3	66.6	0.9	68.0	1.8	**69.2**	0.8	66.0	1.4
Pixelate	75.0	1.0	78.0	0.2	78.7	0.5	**79.4**	1.0	73.8	0.4
Saturate	**80.0**	0.5	76.5	0.5	77.2	0.6	79.2	2.0	69.9	0.8
Shot Noise	71.5	2.1	77.0	0.2	77.7	0.6	**77.8**	0.3	74.8	1.1
Snow	71.2	1.6	71.4	0.3	72.2	0.7	**73.4**	1.3	69.2	1.3
Spatter	75.3	1.1	74.2	0.3	74.7	0.4	**75.6**	1.0	72.0	1.3
Speckle Noise	71.1	2.0	76.8	0.4	77.4	0.4	**77.5**	0.2	74.6	0.6
Zoom Blur	68.8	1.1	71.5	0.2	73.0	1.2	**74.0**	0.5	68.3	1.2

Table 2.5: Image classification performance of AllConvNet with no additional regularization on the test set and common corruptions averaged over 5 runs.

In the case of models trained with additional regularisation (Dropout), in fourteen out of the nineteen corruptions tests, the best performing models involved correlated noise.

2.5 Summary

- We propose a model of regularisation where correlations are induced between neural network activations based on the spatial distance and selectivity of the neurons. This structure is inspired from biological correlations observed in the brain.
- We use the above computed correlations to sample noise from different multivariate noise distributions (Gaussian, Poisson). This noise is used additively as a regulariser in convolutional neural networks.
- We show how samples can be constructed in a differentiable manner for both distributions so that back-propagation may proceed as normal. In the case of the Poisson distribution, we relied on an estimate of the gradient. We also show how a continuous-relaxation to the Poisson distribution can be constructed so that gradient computation can be carried out.

	Baseline		UG		UP		CG		CP	
	mean (%)	std (%)	mean (%)	std (%)	mean (%)	std (%)	mean (%)	std (%)	mean (%)	std (%)
Test Set	**84.3**	0.6	83.1	0.1	83.1	0.1	84.1	0.1	76.2	0.2
Brightness	**80.0**	0.3	78.0	0.2	77.9	0.4	79.1	0.5	70.0	0.6
Contrast	47.9	2.0	46.9	0.7	47.1	0.6	**53.0**	1.2	41.4	1.4
Defocus Blur	74.8	1.6	75.6	0.1	75.8	0.2	**77.8**	0.1	70.4	0.9
Elastic	73.9	1.5	74.1	0.1	74.2	0.1	**76.0**	0.1	68.4	0.4
Fog	62.9	1.6	61.6	0.4	62.4	0.5	**68.0**	0.4	57.8	0.8
Frost	**72.7**	0.2	71.2	0.1	70.7	0.6	72.5	1.1	61.1	1.2
Gaussian Blur	70.7	1.9	72.2	0.2	72.6	0.3	**75.1**	0.2	68.4	1.3
Gaussian Noise	82.7	0.5	82.3	0.1	82.3	0.3	**83.1**	0.2	76.4	1.4
Glass Blur	75.3	1.0	75.5	0.1	75.6	0.1	**76.8**	0.2	70.7	1.6
Impulse Noise	80.6	1.2	81.1	0.1	81.0	0.3	**81.5**	0.4	76.2	0.8
JPEG Compression	82.1	0.6	81.2	0.1	81.7	0.1	**82.4**	0.2	74.7	1.2
Motion Blur	66.4	1.5	68.5	0.2	68.6	0.4	**71.2**	0.4	64.6	1.3
Pixelate	81.3	0.8	80.6	0.1	80.4	0.1	**81.6**	0.2	74.0	0.9
Saturate	**78.6**	0.4	76.4	0.3	76.4	0.2	77.8	0.4	69.5	0.2
Shot Noise	83.1	0.6	82.5	0.1	82.4	0.2	**83.3**	0.2	76.2	0.6
Snow	**77.4**	0.5	76.4	0.1	76.4	0.4	77.0	0.5	70.5	0.5
Spatter	**80.2**	0.6	79.8	0.1	79.8	0.2	80.1	0.1	74.4	0.1
Speckle Noise	83.1	0.5	82.7	0.1	82.5	0.3	**83.4**	0.2	76.3	0.1
Zoom Blur	71.1	1.2	72.6	0.2	72.8	0.2	**75.7**	0.1	68.0	0.8

Table 2.6: Image classification performance of AllConvNet with additional regularization (Dropout) on the test set and common corruptions averaged over 5 runs.

- We evaluate trained networks on occluded images as well as images subject to common corruptions. Preliminary results show that correlated variability added to a single layer performs better than other noise models. In thirty-nine out of fifty tests we studied (ten out of twelve for occlusions, twenty-nine out of thirty-eight for corruptions), with and without additional regularisation, our best performing models had correlated noise.

Chapter 3

Dataset Augmentation and Correlated Activations

Interesting methods of dataset augmentation have been developed in recent years. These have helped deep neural networks to achieve better performance across a wide variety of domains. Dataset augmentation can be done in numerous ways, including transformations in input space and the latent space of the network. In this chapter, we explore the connections between popular dataset augmentation techniques and correlations observed in the brain. Specifically, we show that the pattern of correlations induced by networks trained with these augmentation techniques is quite similar to the structure of biological correlations.

3.1 Introduction

One of the reasons for the success of deep learning is the availability of large datasets. In the case of supervised learning, a large scale effort is required to label the data. This can be done in many ways. An inefficient and cumbersome method is to manually annotate each data point, requiring investment of human effort. Another variant is to create synthetic data from computer simulations. However, if the simulator is not depictive of the real world, the generated data is not useful. Another option is to use dataset augmentation, where existing data is transformed to create new points that (hopefully) reflect the same training data distribution. Depending on the task at hand, this requires specific knowledge of realistic types of transformations. For example, in image classification, it is common to create image reflections, rotations, and mild scaling to augment existing data.

Recent work [47, 41] considers domain-agnostic data augmentation techniques used in training neural networks. We describe these methods briefly in the following subsections.

3.1.1 Mixup: Input Space Data Augmentation

The work in [47] introduces a simple and data-agnostic data augmentation routine, termed *mixup*. The goal is to create virtual training examples using the method shown in Equation 3.1.

$$\begin{aligned} \tilde{x} &= \lambda x_i + (1-\lambda)x_j, \quad \text{where } x_i, x_j \text{ are raw input vectors} \\ \tilde{y} &= \lambda y_i + (1-\lambda)y_j, \quad \text{where } y_i, y_j \text{ are one-hot label encodings} \end{aligned} \tag{3.1}$$

(x_i, y_i) and (x_j, y_j) are two examples drawn from the training set and $\lambda \in [0, 1]$. This method incorporates the prior knowledge that convex combinations of inputs should be associated with convex combinations of the corresponding targets.

This technique brings better image classification performance than previous methods on CIFAR-10 [21], CIFAR-100 [21], and ImageNet [4]. It also improves robustness of neural networks to adversarial examples and can be used to stabilise training of generative adversarial networks.

3.1.2 Manifold Mixup: Latent Space Data Augmentation

Manifold mixup is a method to train neural networks on linear combinations of hidden representations of training examples shown in [41]. This method applies the mixup technique to hidden layers in a neural network. To be specific, consider a deep neural network $f(x) = f_k(g_k(x))$, where g_k denotes the part of the neural network mapping the input data to the hidden representation at layer k, and f_k denotes the part mapping such hidden representation to the output $f(x)$. The procedure is shown below.

- Select a random layer k in the network. Compute the forward pass of two minibatches of data (x, y) and (x', y') until layer k. Let the activations of the two minibatches be $g_k(x)$ and $g_k(x')$.
- The two minibatch intermediate activations are linearly combined as per Equation 3.1, using the layer k activations as x. This produces the mixed minibatch $(\tilde{g}_k, \tilde{y}) := (\text{Mix}_\lambda(g_k(x), g_k(x')), \text{Mix}_\lambda(y, y'))$.

- The forward pass is continued using the mixed batch activations.
- This output is used to compute the loss and gradients that update the parameters of the neural network.

This method of augmentation in latent space demonstrates benefits in generalization, log-likelihood on test samples, and classification of images on CIFAR-100 subject to minor deformations.

3.1.3 Connecting Correlations to Augmentation

In chapter 2, we implemented a regulariser based on correlated noise. The noise was added to intermediate layers of a neural network and improved classification performance under certain types of image transformations. By adding noise to the units in a layer, the network was trained on activation values that were correlated by design. To understand the role of this regularisation, an interesting experiment would be to find the inputs that produce the correlated activations on the forward pass of the network. However, this is difficult to implement because the traditional intermediate functions used in neural networks are non-invertible. There is recent work in designing invertible convolutional neural networks, but we did not work with these models [17].

We wish to compare noise correlations in the brain to correlations in neural networks. To make a suitable comparison, when computing correlations, we interpret one image to be the input and mixing a secondary batch of data using the *mixup* technique as the source of variability. This setup will be most similar to the setting of noise correlations in the brain. Sorting out the similarities and differences between the structures will give us insight into the functional roles of different types of regularisers.

3.2 Correlations of Neural Network Activity

We describe the method of computing and plotting the correlations as a function of distance between units and their tuning similarity. The resulting structures are compared against the correlations implemented in chapter 2.

3.2.1 Gathering Neural Network Activations

To compute the correlations between units of a layer in a neural network, we need the activation values over a series of observations. Let $N = H \times W \times C$ represent the number of units in a layer K of a convolutional neural network, where H is the height of the feature map, W is the width of the feature map, and C is the number of feature maps. Thus, the correlation matrix $\mathbf{\Sigma} \in \mathbb{R}^{\mathrm{N} \times \mathrm{N}}$. Let S be the number of fixed input images under consideration. In our experiments, we set $S = 100$. The entire procedure is described below.

- Train a convolutional neural network on image classification. After training is complete, pick S random images from the test set.
- For each image, pick a random mini-batch of size B from the test set. Apply the input *mixup* method using the chosen image and the random mini-batch for B λ values interpolated between $[0, 1]$.
- Using this mixed data, compute the forward pass activations until layer K. Calculate the Pearson product-moment correlation on the N random variables (unit activations of the layer K) using the B observations. Let this be Σ_i, the computed correlation on example i.
- After processing the S inputs, calculate the average correlation element-wise,

$$\Sigma' = \frac{1}{S} \sum_{i=1}^{S} \Sigma_i \tag{3.2}$$

3.2.2 Estimating the Correlation as a Function of Distance and Tuning using Kernel Smoothing

The computed correlation Σ' represents the linear relationship between the activity values of units within a specific layer of the convolutional network. This is shown in Figure 3.1. We are interested in understanding the relationship between Σ' and two variables: the distance between two units in a feature map and the tuning similarity of the two units. In Equation 2.14, we pre-computed this function to be the product of a linear function of the distance and an exponential function of the tuning similarity and induced the corresponding correlations as regularisation.

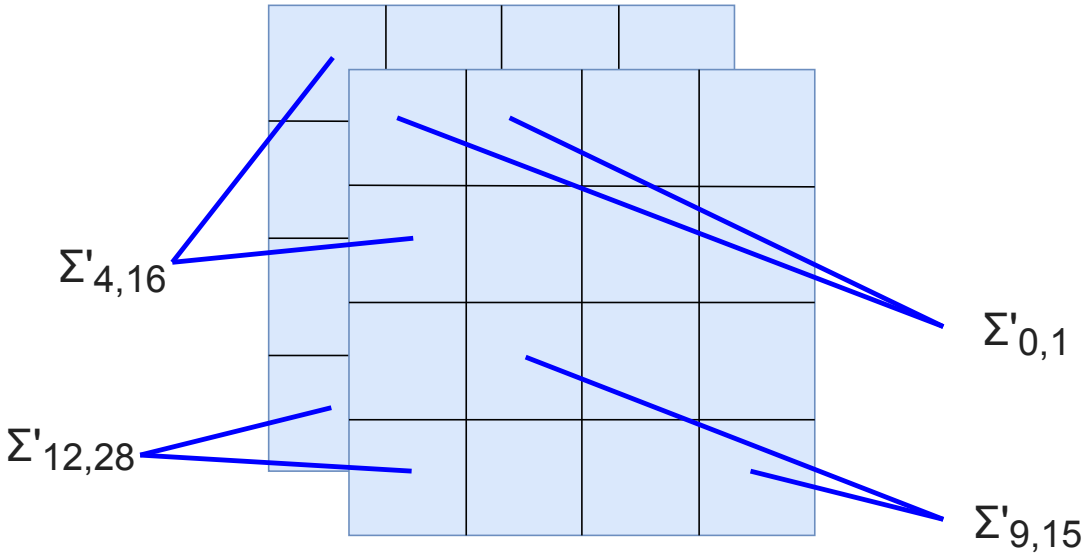


Figure 3.1: For a sample $\mathbf{\Sigma'} \in \mathbb{R}^{32\times32}$ in a layer with 2 feature maps of size 4×4, this figure shows which units each entry in $\mathbf{\Sigma'}$ is related to.

Given $\Sigma'_{i,j} \in \mathbb{R}^{\mathrm{N}\times\mathrm{N}}$, we can compute which two units in the layer this refers to. Specifically, let $N = H \times W \times C$ represent the number of units in a layer K of a convolutional neural network, where H is the height of the feature map, W is the width of the feature map, and C is the number of feature maps. Given a linear index $i \in [0, N)$, it refers to the unit at coordinates (x, y) at the z^{th} feature map. x, y, and z are calculated as per Equation 3.3. Let this function be $f(i)$, where it is given a scalar input i and it returns a 3-tuple (x, y, z). Note // refers to floor division and % refers to the modulo operator.

$$\begin{aligned} z &= i \;//\; (H \times W) \\ x &= \Big[i \;\%\; (H \times W) \Big] \;//\; W \\ y &= \Big[i \;\%\; (H \times W) \Big] \;\%\; W \end{aligned} \tag{3.3}$$

Since $\Sigma' \in \mathbb{R}^{\mathrm{N}\times\mathrm{N}}$ is a symmetric matrix, we have $\frac{N(N-1)}{2}$ possible data points from which to get a smooth estimate of the correlation function. We describe the procedure to create the dataset from which we will compute a smooth estimator.

- Let $\mathrm{A} = (i,j)_{i=0,j=i}^{N,N}$ be the sequence of indices of the entries sampled from Σ'.
- Let $(X_{A_k}, Y_{A_k})_{k=0}^{|A|}$ be the dataset we are creating:

$$X_{A_k} = X_{(i,j)} = \left[d\left(f(i), f(j)\right), k\left(f(i), f(j)\right) \right] \in \mathbb{R}^2 \tag{3.4}$$

$$Y_{A_k} = Y_{i,j} = \Sigma'_{i,j}, \tag{3.5}$$

  where $d(\cdot,\cdot)$ and $k(\cdot,\cdot)$ are given by Equation 2.15 and Equation 2.16.

We use kernel smoothing to estimate the real valued function $g : \mathbb{R}^2 \to \mathbb{R}$ as a weighted average of the neighbouring observed data. Formally, let $K\left(X_0, X\right) = \exp\left(-\frac{\|X - X_0\|}{2b^2}\right)$ be a Gaussian kernel, where $X, X_0 \in \mathbb{R}^2$, $\|\cdot\|$ is the Euclidean norm, and b is the length scale for the input space. For each $X_0 \in \mathbb{R}^2$, the kernel-weighted average (smooth $g(X)$ estimation) is defined by

$$g\left(X_0\right) = \frac{\sum_{i=1}^{N} K\left(X_0, X_i\right) Y\left(X_i\right)}{\sum_{i=1}^{N} K\left(X_0, X_i\right)}, \tag{3.6}$$

where N is the number of observed points and in our case $N = |A|$.

3.2.3 Model Architecture and Dataset

We use the VGG-11 architecture [33] trained on the CIFAR-10 [21] dataset. The architecture is shown in Table 3.1. The network is trained using stochastic gradient descent with momentum = 0.9 and a weight decay of 0.0001. The networks are trained until the test set accuracy of image classification is $\sim 80\%$. We are looking for a network that has decent test set accuracy for our correlation estimation. The experiments in this section are implemented in PyTorch [29].

3.2.4 Analysis

In this section, we compare the estimated correlation function with Figure 2.3, both visually and quantitatively. We train three networks: VGG-11 with input mixup, VGG-11 with manifold mixup, and vanilla VGG-11 without any augmentation for comparison. For each network, we compute the estimate of the correlation function for layers 3, 5, 6, 8, and 9.

Input: 32×32 RGB image
Layer 1: 3×3 conv. 64 filters, BatchNorm, ReLU
Layer 2: 2×2 Max-Pooling, stride $= 2$
Layer 3: 3×3 conv. 128 filters, BatchNorm, ReLU
Layer 4: 2×2 Max-Pooling, stride $= 2$
Layer 5: 3×3 conv. 256 filters, BatchNorm, ReLU
Layer 6: 3×3 conv. 256 filters, BatchNorm, ReLU
Layer 7: 2×2 Max-Pooling, stride $= 2$
Layer 8: 3×3 conv. 512 filters, BatchNorm, ReLU
Layer 9: 3×3 conv. 512 filters, BatchNorm, ReLU
Layer 10: 2×2 Max-Pooling, stride $= 2$
Layer 11: 3×3 conv. 512 filters, BatchNorm, ReLU
Layer 12: 3×3 conv. 512 filters, BatchNorm, ReLU
Layer 13: 2×2 Max-Pooling, stride $= 2$
Layer 14: 512×10 Fully-connected Layer
10-way softmax

Table 3.1: VGG-11 Architecture used for training on CIFAR-10.

The contour plots for each layer can be seen in Figure 3.2, Figure 3.3, Figure 3.4, Figure 3.5, and Figure 3.6. We also record the maximum, minimum, average, and standard deviation of the sampled correlation values from Σ' for quantitative comparison shown in Table 3.2.

The first observation that stands out when viewing the estimated correlation function for all layers is the dependence on distance and kernel similarity. The correlation tends to decrease as distance between units increase and the correlation also decreases when the cosine similarity between two kernels decreases. This is in agreement with the biological structure discussed earlier. However, this may not be a surprising finding. By nature of the convolution operation, we expect nearby units to be correlated because of the same kernel being convolved over the feature map. This would be more pronounced in earlier layers (layers close to the inputs) because the inputs are natural images which tend to be smooth with nearby pixels being correlated. This is reinforced by the peak and average correlations in layer 3 of VGG-11 shown in Table 3.2 in comparison to the other layers. This accounts for the spatial dependence. As for the kernel dependence, since all kernels in a convolutional layer are computing the same operation, it is natural for similar kernels to produce similar outputs, hence increasing the correlation.

Another observation is the dependence on distance decays faster as the inputs pass

		Vanilla	Input Mixup	Manifold Mixup	Linear Exponential
Layer 3	Max.	0.43	0.49	0.58	0.27
	Min.	-0.05	-0.07	-0.07	0.09
	Mean	0.10	0.12	0.10	0.14
	Std.	0.06	0.11	0.09	0.04
Layer 5	Max.	0.23	0.41	0.33	0.27
	Min.	0.02	0.03	0.04	0.14
	Mean	0.10	0.13	0.11	0.19
	Std.	0.05	0.06	0.04	0.03
Layer 6	Max.	0.22	0.22	0.22	0.27
	Min.	0.03	0.04	0.03	0.14
	Mean	0.06	0.11	0.07	0.19
	Std.	0.02	0.02	0.01	0.03
Layer 8	Max.	0.24	0.30	0.38	0.27
	Min.	0.00	0.00	-0.03	0.14
	Mean	0.03	0.07	0.07	0.19
	Std.	0.02	0.06	0.10	0.03
Layer 9	Max.	0.23	0.46	0.46	0.27
	Min.	0.00	-0.02	-0.01	0.14
	Mean	0.03	0.13	0.13	0.19
	Std.	0.02	0.13	0.06	0.03

Table 3.2: Minimum, maximum, average, and standard deviation of sampled correlations from Σ' for a VGG-11 without data augmentation, with mixup, and manifold mixup. The linear exponential refers to the pre-computed induced correlations discussed in chapter 2.

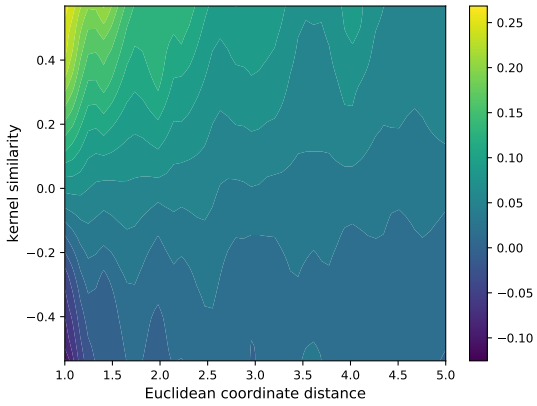
through the network. In the early layers, the correlations die out after a distance of 5 units while in the intermediate layers they die out after 1.5 units. This can be seen in Figure 3.2, Figure 3.3, Figure 3.4, Figure 3.5, and Figure 3.6.

We also see mixup and manifold mixup increases the peak and average correlations in the early and later layers. For visual comparison, this is shown in 3.7 for layer 9 units in VGG-11. Since mixup and manifold mixup provide better test performance, it suggests inducing correlations might be a good idea. This is also in line with results from chapter 2 where correlations induced in early layers as regularisers improved robust image

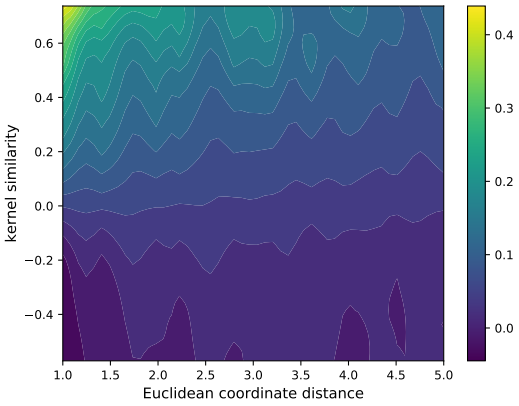
classification.

3.3 Summary

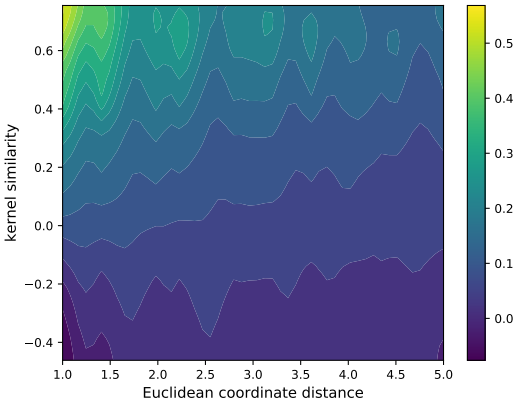
- We demonstrated how we can compute an estimate of the correlations between units in a layer of a convolutional neural network using kernel smoothing.
- We compared this estimate across different training paradigms: vanilla convolutional networks, networks trained with input mixup, and networks trained with manifold mixup. We see input and manifold mixup tend to induce higher correlations in early and late layers in VGG-11.
- We also show the unsurprising similarity to biological correlations. This makes sense because of the nature of the convolution operation and the smoothness of natural images.



(a) Vanilla VGG-11

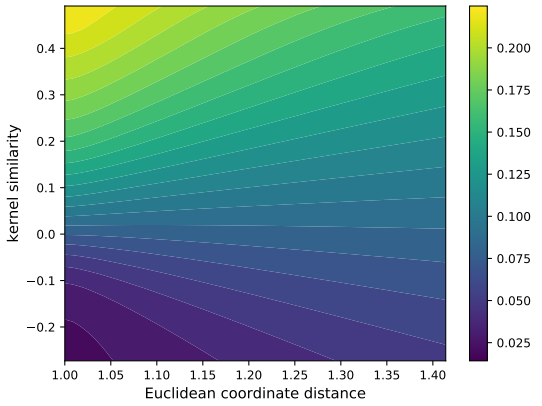


(b) Input Mixup

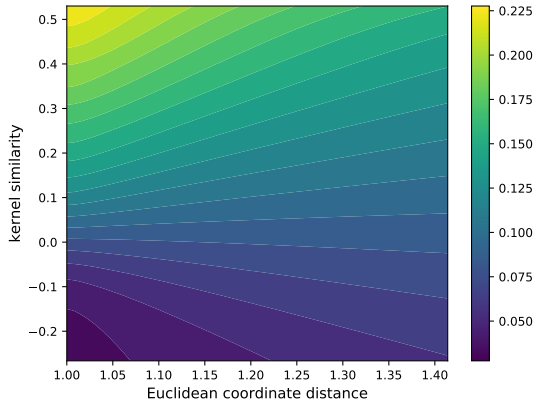


(c) Manifold Mixup

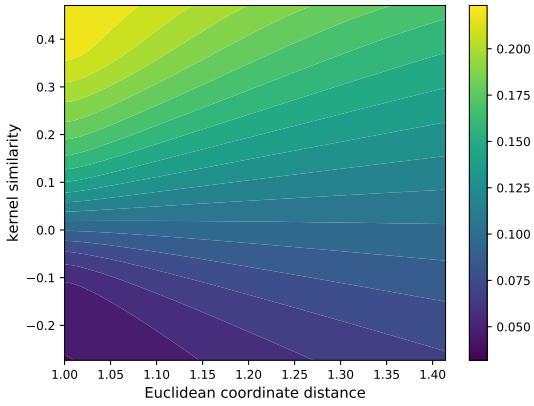
Figure 3.2: Estimation of smooth correlation function on layer 3 for three VGG-11 networks trained under different paradigms.



(a) Vanilla VGG-11

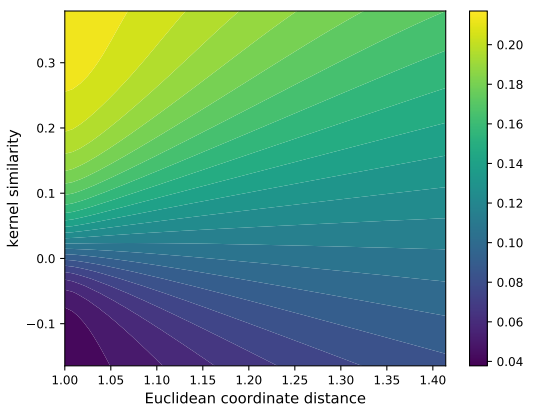


(b) Input Mixup

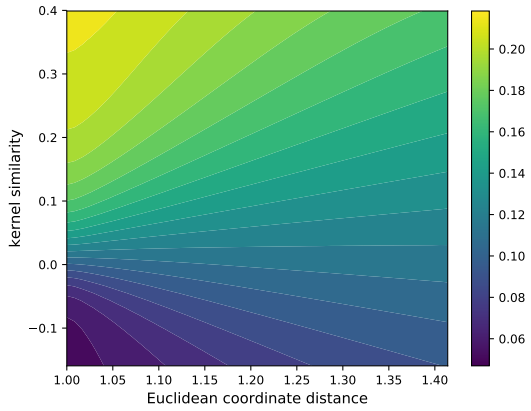


(c) Manifold Mixup

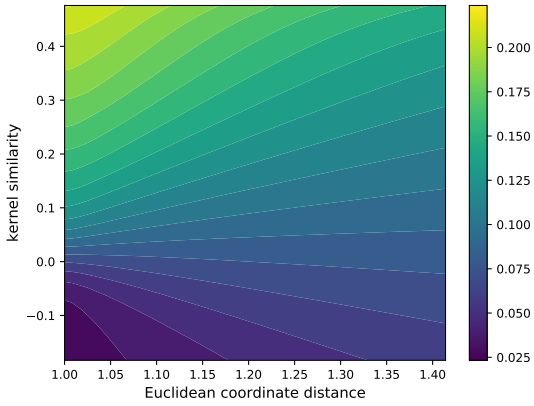
Figure 3.3: Estimation of smooth correlation function on layer 5 for three VGG-11 networks trained under different paradigms.



(a) Vanilla VGG-11

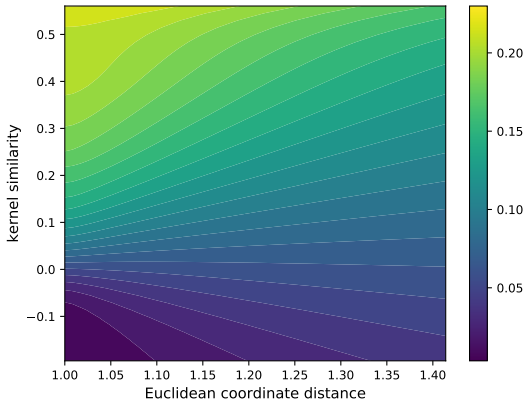


(b) Input Mixup

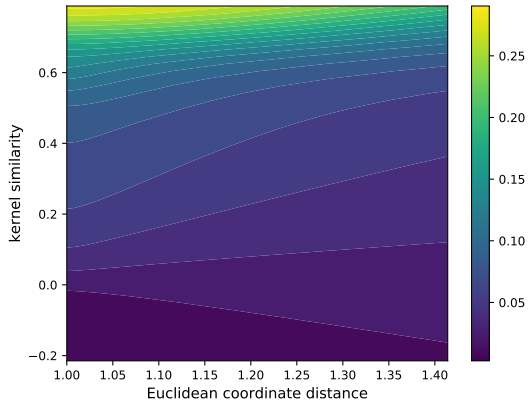


(c) Manifold Mixup

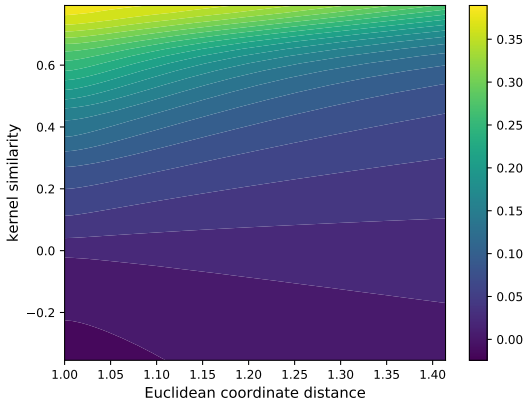
Figure 3.4: Estimation of smooth correlation function on layer 6 for three VGG-11 networks trained under different paradigms.



(a) Vanilla VGG-11

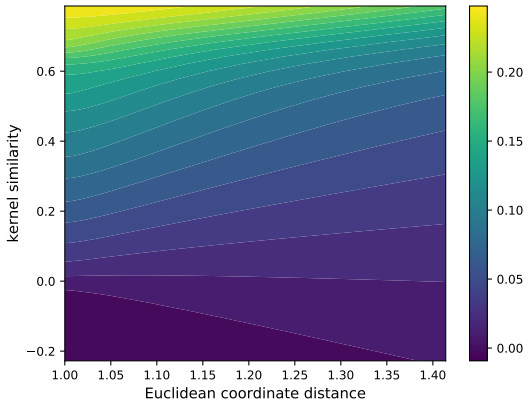


(b) Input Mixup

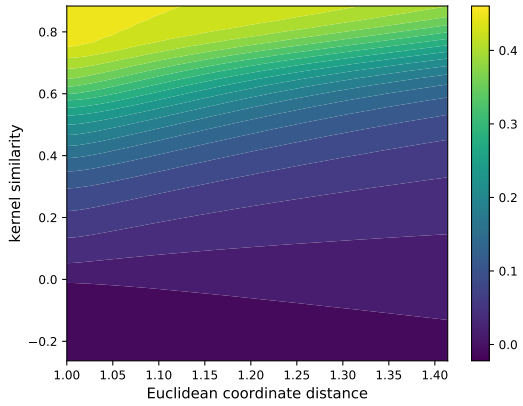


(c) Manifold Mixup

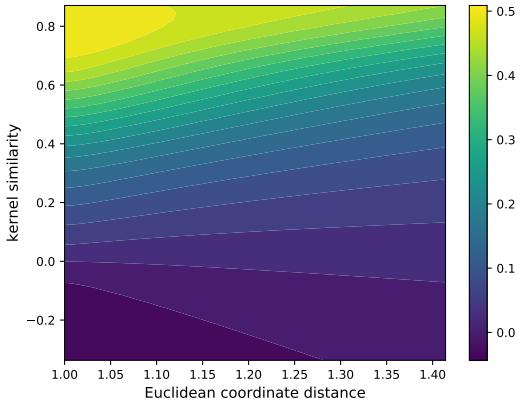
Figure 3.5: Estimation of smooth correlation function on layer 8 for three VGG-11 networks trained under different paradigms.



(a) Vanilla VGG-11

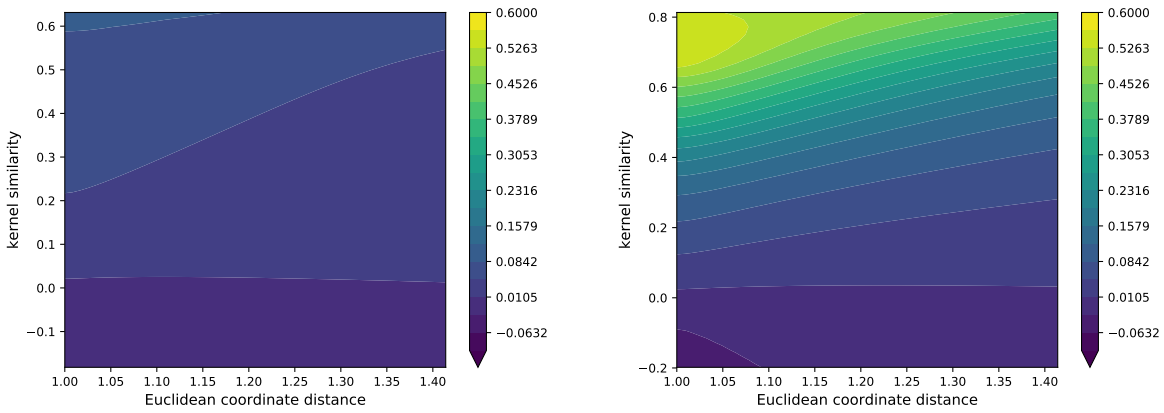


(b) Input Mixup



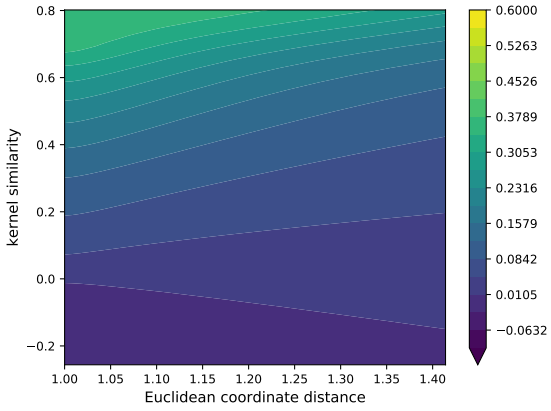
(c) Manifold Mixup

Figure 3.6: Estimation of smooth correlation function on layer 9 for three VGG-11 networks trained under different paradigms.



(a) Vanilla VGG-11

(b) Input Mixup



(c) Manifold Mixup

Figure 3.7: Estimation of smooth correlation function on layer 9 for three VGG-11 networks trained under different paradigms. In this figure, since the same colour bars are used across the sub-plots, it is clear that input mixup and manifold mixup increase the correlations across units in layer 9 of VGG-11.

Chapter 4

Dropping Correlated Units in Convolutional Layers

Deep neural networks are over-parameterised models trained with extensive amounts of regularisation, such as weight decay and dropout [36], to prevent overfitting to the training data. With the introduction of batch normalisation [16], weight decay is not as popular anymore, but is still used in some neural networks. In the seminal work of training AlexNet [22], Dropout was used at the fully-connected layers and not at the convolutional layers. Even in current deep learning systems, it is rare to see Dropout used in convolutional layers of a neural network. One hypothesis is Dropout is less suitable for convolutional layers because it drops features randomly. In convolutional layers, units tend to be correlated spatially by nature of the convolution operation and the input data, which are mostly natural images. Recent work [8, 39, 42, 23] has experimented with spatially structured forms of Dropout. Keeping up with the theme of this thesis, we extend spatially structured Dropout to involve correlations across feature maps. We show improvements in image classification on CIFAR-10 using the VGG-11 architecture.

4.1 Introduction

Overfitting in neural networks can be reduced using Dropout to prevent complex co-adaptations on the training data [14]. In the forward pass of the network, each unit has a probability p of being zeroed out, preventing units on depending on the presence of other hidden units. Dropout is an efficient way to perform an average consensus of all possible

sub-networks, weighting each sub-network by its likelihood. The standard way to achieve this ensemble is to train separate models and then average their results at inference time. However, this is computationally expensive and inefficient. Dropout provides a method to train an exponential number of models which share the same parameters. At test time, the units' activations are averaged, but scaled down by the probability p to compensate for the fact that $\frac{1}{p}$ neurons are active.

In convolutional neural networks, units tend to be correlated spatially. As shown in chapter 3, units in convolutional layers are correlated spatially as well as across feature maps. Since the initial goal of Dropout is to prevent co-adaptation between units, it makes intuitive sense to drop these correlated units. This has motivated spatially structured Dropout in convolutional networks [15, 39, 8, 49].

We focus on dropping correlated units based on tuning/kernel similarity in addition to spatial correlation. We use the definition of tuning similarity as in Equation 2.16 shown in chapter 2. The following sections will describe closely related work to our contributions.

4.1.1 Structured Forms of Dropout for Convolutional Networks

The contributions in this chapter are most similar to SpatialDropout [39] and DropBlock [8]. We describe these two methods first.

Spatial Dropout

The authors in [39] introduce SpatialDropout, a specific method of Dropout where entire feature maps are dropped during training instead of randomly picking units to zero out. This method is more suitable for convolutional networks due to spatial correlation between units in feature maps.

DropBlock

DropBlock is a method similar to Dropout and SpatialDropout. It is similar to Dropout because it drops units during training but different because it drops contiguous regions in feature maps instead of random units. It resembles SpatialDropout but does not drop out the entire feature map. It creates contiguous regions controlled by two hyperparameters: *block_size* and γ. *block_size* is the size of the block to be dropped and γ controls how many

activation units to drop. This is elucidated in Algorithm 1 and Figure 4.1. This work has shown best results with a shared mask across all feature maps.

Input output activations of a layer (A), block_size, γ, *mode*;
if *mode* == Inference **then**;
  return A;
end if;
Randomly sample mask M: $M_{i,j} \sim \text{Bernoulli}(\gamma)$;
For each zero position $M_{i,j}$, create a spatial square mask with the center being $M_{i,j}$, the width, height being block_size and set all the values of M in the square to be zero;
Apply the mask: $A = A \times M$;
Normalise the features: $A = A \times$ **count**(M)/**count_ones**(M)

Algorithm 1: DropBlock.

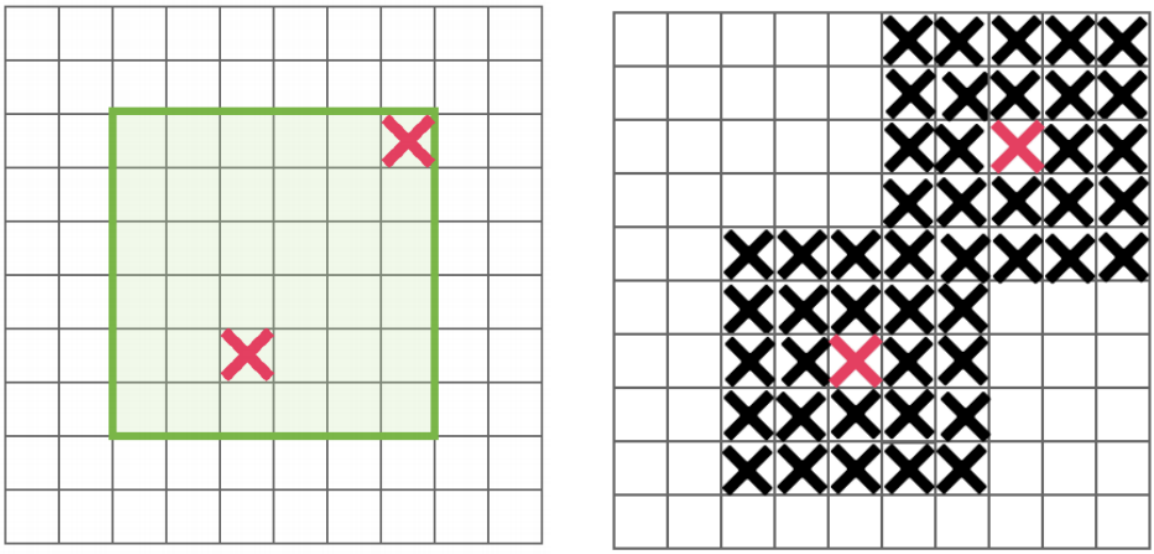


Figure 4.1: Mask sampling in DropBlock. On every feature map, sample a mask M. Every zero entry in M is expanded to block_size$\times$block_size zero block. The green region indicates the valid region from which each sample entry can be expanded to a mask fully contained in the feature map. Image is taken from [8].

The value of γ is computed based on the dropout probability set by the user. It is approximated by Equation 4.1, where feat_size is the size of the feature map. Every zero entry in the mask will be expanded by block_size^2. The size of the valid seed region is

feat_size − block_size + 1. The equation below is only an approximation because there will be some overlap in the dropped blocks.

$$\gamma = \frac{\text{drop\_prob}}{\text{block\_size}^2} \frac{\text{feat\_size}}{(\text{feat\_size} - \text{block\_size} + 1)^2} \tag{4.1}$$

In addition, the authors found DropBlock with a fixed *drop_prob* does not work well. Instead, they gradually increase the *drop_prob* from 0 to 0.5 as training progresses using a linear scheduling rate.

4.2 Dropping Feature Maps using Kernel Similarity

We experiment with two Dropout variants: one based on SpatialDropout and one based on DropBlock. Both modifications involve calculating the kernel similarity as described in Equation 2.16. In the ideal situation, we would use Equation 2.14 to determine the correlated units and drop them. However, we found this to be computationally expensive to run during training. Since we already know the spatial correlations exist between nearby units in a convolutional layer, we can just incorporate the kernel similarity and extend SpatialDropout and DropBlock.

In the first modification, we change SpatialDropout to drop a specific set of feature maps versus random selection of feature maps. For every example, we randomly choose one feature map and drop the top p^{th} percentile of similar feature maps during training. The features are then normalised by $\frac{1}{1-p}$ which is the same as in SpatialDropout. By using this method, correlated feature maps will be dropped during training. This method is a simple extension and does not require any changes to the hyperparameters.

We also propose a second modification involving DropBlock. Instead of sharing the mask across all feature maps, we pick a random feature map per example, find the top 50% of similar feature maps and apply the corresponding mask to the selected maps. Since the mask does not drop an entire feature map, the value of *drop_prob* needs to be multiplied by 2 at each step during the scheduling to keep a similar effective drop out probability to DropBlock. We call this method CorrDrop. To ensure this results from this method are effective, we also use this modification: instead of finding the most similar feature maps for every example, we choose 50% of the number of feature maps at random during training. This will tell us if the kernel similarity plays a role in the dropping of units.

4.2.1 Model Architecture & Dataset

We use the VGG-11 architecture [33] trained on the CIFAR-10 [21] dataset. The architecture is shown in Table 3.1. The network is trained using stochastic gradient descent with momentum = 0.9 and a weight decay of 0.001. The networks are trained for 50 epochs and the model with the best validation accuracy is saved for evaluation. The experiments in this section are implemented in PyTorch [29].

4.2.2 Results

We evaluate the test set accuracy as well as the accuracy on the common corruptions dataset on CIFAR-10 [13] across the methods described above implemented at different layers in the network. The results are listed in Table 4.1.

In our modifications to SpatialDropout (SpatialDropoutMod), we see no improvements when compared to SpatialDropout on test set accuracy. We believe this might be because by choosing feature maps randomly, the number of possible combinations of zeroed out maps is $\binom{n}{\frac{n}{2}}$, where n is the number of feature maps and the dropout probability is 50%. With our modification, the number of possible combinations will be lower. For example, kernels may not change as frequently at each optimisation step, and thus the same groups of feature maps will be dropped leading to fewer possible combinations of dropped feature maps. This also suggests kernel similarity may have no role to play in SpatialDropout in improving test set accuracy. On the common corruptions dataset, it improves performance in 8/19 cases in layer 1, 6/19 cases in layer 3, 10/19 cases in layer 5, 14/19 cases in layer 6, and 15/19 cases in layer 8. The performance improvements are significant in later layers and indicates that SpatialDropoutMod is more robust to corrupted images than SpatialDropout.

In our second modification to DropBlock, we see improvements in both CorrDrop and DropBlockRandom. This suggests applying the same mask to all feature maps may not be the ideal technique of DropBlock. However, it is still unclear whether the feature maps to which the mask is applied should be chosen at random (DropBlockRandom) or chosen based on kernel similarity (CorrDrop) to improve test set performance. This is because the improvements are split between the two methods when evaluated across layers with low differences in test set accuracy. On the common corruptions dataset, DropBlockRandom improves classification accuracy in 19/19 cases in layer 1, 19/19 cases in layer 3, 19/19 cases in layer 5, 18/19 cases in layer 6, and 18/19 cases in layer 8 over DropBlock. This suggests DropBlockRandom is a better technique for robust image classification than

	SpatialDropout		SpatialDropoutMod		DropBlock		DropBlockRandom		CorrDrop	
	Mean (%)	std (%)	Mean (%)	std (%)	Mean (%)	std (%)	Mean (%)	std (%)	Mean (%)	std (%)
Test Set	80.37	0.22	79.90	0.51	81.24	0.49	**84.64**	0.17	84.40	0.24
Brightness	78.54	0.31	78.28	0.45	78.38	0.43	**82.82**	0.21	82.34	0.39
Contrast	58.59	0.75	**61.05**	0.88	46.31	1.04	55.27	0.24	54.32	1.31
Defocus blur	74.32	0.32	**76.01**	0.14	67.67	0.67	75.79	0.28	74.43	1.02
Elastic transform	71.95	0.17	73.89	0.35	67.92	0.63	**74.97**	0.06	74.08	0.67
Fog	67.99	0.61	**70.96**	0.10	60.89	0.71	70.55	0.51	69.12	1.25
Frost	74.42	0.18	73.46	0.18	62.14	0.76	**76.72**	0.33	74.36	0.83
Gaussian blur	71.75	0.56	**73.90**	0.13	61.97	0.71	71.73	0.30	69.88	1.47
Gaussian noise	**73.62**	0.35	68.43	0.88	46.87	1.83	68.82	0.18	65.73	1.51
Glass blur	**71.92**	0.29	69.23	0.28	44.38	1.12	67.09	0.79	63.50	0.73
Impulse noise	**65.20**	0.08	59.22	0.90	44.00	2.07	61.63	0.40	60.35	1.77
Jpeg compression	78.12	0.24	78.13	0.48	74.50	0.91	**80.81**	0.18	80.01	0.33
Motion blur	69.79	0.57	**71.76**	0.25	60.26	0.79	70.97	0.37	68.72	1.26
Pixelate	78.99	0.20	78.41	0.26	75.54	0.51	**81.92**	0.22	81.27	0.36
Saturate	76.57	0.18	74.05	0.51	76.95	0.50	**80.67**	0.17	80.58	0.27
Shot noise	**75.25**	0.21	71.93	0.77	54.33	1.40	73.13	0.34	70.56	1.20
Snow	73.40	0.21	72.14	0.47	64.15	0.33	**75.86**	0.52	74.38	0.61
Spatter	72.05	0.39	69.60	0.54	68.41	0.32	**76.00**	0.12	75.77	0.50
Speckle noise	**74.75**	0.16	71.96	0.59	54.69	1.49	72.94	0.45	70.46	1.24
Zoom blur	70.43	0.32	**73.63**	0.10	61.35	0.93	71.06	0.30	69.50	1.35

Table 4.1: Test set accuracy and Corruption set accuracy on CIFAR-10 across all methods of structured dropout on layer 1 of VGG-11. SpatialDropoutMod refers to our modification to SpatialDropout and DropBlockRandom is only applying the selected mask to randomly selected feature maps. The results are averaged over 5 runs.

DropBlock. Between DropBlockRandom and CorrDrop, the performance is slightly better for DropBlockRandom but still comparable on the common corruptions dataset.

4.3 Summary

- We propose modifications to existing forms of spatially structured Dropout and evaluated them on the test set of CIFAR-10.
- The first modification involves using kernel similarity to target certain feature maps to be completely dropped in SpatialDropout. However, this does not yield any benefits over SpatialDropout on the test set. When applied to later layers, it yields benefits in image classification of commonly corrupted images.
- The second modification uses kernel similarity to apply the mask to only certain feature maps versus all of them as implemented in DropBlock. We also choose feature maps at random to apply the mask and leave the remaining as is. In both cases, the

	SpatialDropout		SpatialDropoutMod		DropBlock		DropBlockRandom		CorrDrop	
	Mean (%)	std (%)	Mean (%)	std (%)	Mean (%)	std (%)	Mean (%)	std (%)	Mean (%)	std (%)
Test Set	81.69	0.16	80.34	0.48	84.12	0.06	84.53	0.27	**84.78**	0.06
Brightness	79.79	0.06	78.81	0.43	81.65	0.12	82.79	0.31	**82.93**	0.14
Contrast	56.59	0.43	**60.43**	0.60	52.28	0.33	57.05	0.66	55.68	0.40
Defocus blur	73.69	0.13	72.14	0.44	71.90	0.17	**75.54**	0.54	75.28	0.12
Elastic transform	72.46	0.23	71.53	0.55	72.27	0.13	**74.87**	0.47	74.60	0.15
Fog	69.01	0.11	70.64	0.68	67.85	0.05	**71.14**	0.50	70.97	0.12
Frost	71.92	0.35	74.04	0.49	69.68	0.44	73.63	0.20	**74.15**	0.24
Gaussian blur	70.28	0.10	68.72	0.55	66.93	0.32	**71.23**	0.66	70.98	0.14
Gaussian noise	65.01	0.77	66.32	1.52	60.62	0.32	66.40	0.96	**66.57**	1.06
Glass blur	58.43	0.86	**62.96**	0.79	51.62	0.33	57.99	0.42	57.94	0.75
Impulse noise	57.54	0.73	55.98	2.51	55.73	1.42	59.01	1.19	**59.92**	1.03
Jpeg compression	77.91	0.12	77.05	0.36	78.95	0.14	**80.15**	0.11	79.94	0.13
Motion blur	68.81	0.18	68.61	0.31	64.43	0.38	**69.45**	0.94	69.11	0.18
Pixelate	77.70	0.25	74.00	1.02	76.86	0.28	**79.52**	0.15	78.88	0.22
Saturate	78.31	0.20	74.84	0.87	80.32	0.13	80.88	0.23	**81.25**	0.09
Shot noise	68.91	0.73	71.04	1.21	65.98	0.45	71.08	0.83	**71.16**	0.59
Snow	71.50	0.22	71.08	0.42	70.45	0.24	**74.13**	0.55	73.83	0.36
Spatter	72.41	0.26	68.81	0.99	74.30	0.21	74.56	0.29	**75.24**	0.39
Speckle noise	68.39	0.87	**71.30**	1.12	65.81	0.47	70.71	1.03	70.70	0.34
Zoom blur	69.14	0.22	68.48	0.61	65.92	0.08	**70.64**	0.53	69.83	0.32

Table 4.2: Test set accuracy and Corruption set accuracy on CIFAR-10 across all methods of structured dropout on layer 3 of VGG-11. SpatialDropoutMod refers to our modification to SpatialDropout and DropBlockRandom is only applying the selected mask to randomly selected feature maps. The results are averaged over 5 runs.

	SpatialDropout		SpatialDropoutMod		DropBlock		DropBlockRandom		CorrDrop	
	Mean (%)	std (%)	Mean (%)	std (%)	Mean (%)	std (%)	Mean (%)	std (%)	Mean (%)	std (%)
Test Set	83.18	0.16	82.29	0.24	84.47	0.27	85.15	0.23	85.02	0.05
Brightness	81.43	0.17	80.69	0.12	82.12	0.57	**83.12**	0.12	82.94	0.02
Contrast	59.83	0.91	55.44	0.42	53.79	1.45	59.62	0.67	**59.96**	0.64
Defocus blur	75.98	0.93	72.42	0.60	75.70	1.01	77.11	0.17	**77.52**	0.14
Elastic transform	74.96	0.69	72.02	0.23	75.47	0.62	76.32	0.05	**76.62**	0.25
Fog	72.74	0.88	68.49	0.33	69.03	1.11	73.09	0.32	**73.21**	0.55
Frost	72.28	0.57	73.57	0.80	68.23	1.70	**73.95**	0.32	73.53	0.42
Gaussian blur	72.41	1.15	67.89	0.76	71.39	1.25	73.35	0.46	**73.83**	0.26
Gaussian noise	61.97	1.01	**65.84**	2.72	57.40	0.88	65.44	0.91	65.20	1.03
Glass blur	56.10	0.97	**63.21**	1.58	51.84	2.81	58.98	0.89	59.44	1.67
Impulse noise	55.02	1.02	**60.72**	2.12	56.42	0.98	58.26	0.40	59.20	0.38
Jpeg compression	78.65	0.25	79.33	0.11	78.47	0.83	**80.58**	0.06	80.14	0.16
Motion blur	71.81	0.88	65.58	0.37	70.35	0.58	72.42	0.48	**72.50**	0.52
Pixelate	76.10	0.65	78.11	0.64	75.06	0.64	78.97	0.28	**79.03**	0.58
Saturate	79.36	0.10	77.25	0.54	80.25	0.38	81.10	0.09	**81.25**	0.03
Shot noise	67.81	0.63	70.12	2.02	63.61	0.98	**70.65**	0.77	70.22	0.90
Snow	71.05	0.63	**74.19**	0.48	69.70	1.42	74.06	0.04	73.82	0.19
Spatter	71.92	0.20	73.86	0.23	73.74	0.73	**74.77**	0.12	74.63	0.22
Speckle noise	67.94	0.58	69.97	1.80	64.06	0.96	**70.61**	0.84	70.19	0.99
Zoom blur	71.71	1.12	67.05	0.90	71.57	1.08	73.12	0.05	**73.23**	0.07

Table 4.3: Test set accuracy and Corruption set accuracy on CIFAR-10 across all methods of structured dropout on layer 5 of VGG-11. SpatialDropoutMod refers to our modification to SpatialDropout and DropBlockRandom is only applying the selected mask to randomly selected feature maps. The results are averaged over 5 runs.

	SpatialDropout		SpatialDropoutMod		DropBlock		DropBlockRandom		CorrDrop	
	Mean (%)	std (%)	Mean (%)	std (%)	Mean (%)	std (%)	Mean (%)	std (%)	Mean (%)	std (%)
Test Set	85.19	0.01	83.53	0.44	86.16	0.09	85.87	0.14	86.21	0.15
Brightness	82.55	0.41	81.73	0.38	83.87	0.07	83.92	0.09	**84.14**	0.09
Contrast	56.50	0.40	**62.88**	0.31	59.12	0.62	60.80	0.72	59.72	0.96
Defocus blur	74.93	0.35	75.48	0.85	76.85	0.51	77.58	0.35	**77.73**	0.39
Elastic transform	74.95	0.30	75.16	0.57	77.19	0.47	77.24	0.11	**77.62**	0.17
Fog	71.04	0.48	**74.79**	0.23	73.65	0.59	73.85	0.24	74.02	0.22
Frost	71.00	0.56	**75.70**	0.37	73.14	0.48	74.21	0.33	74.64	0.47
Gaussian blur	70.13	0.55	71.94	0.79	72.52	0.73	73.54	0.46	**73.63**	0.57
Gaussian noise	58.57	1.38	**67.20**	0.86	63.34	1.33	65.32	0.32	65.88	0.68
Glass blur	55.74	1.31	**64.57**	0.91	59.74	1.12	59.81	0.49	61.61	0.89
Impulse noise	57.69	1.49	54.32	2.05	60.46	1.53	61.12	0.77	**62.39**	0.55
Jpeg compression	79.78	0.19	79.88	0.39	81.23	0.11	81.17	0.08	**81.43**	0.19
Motion blur	69.27	0.45	71.94	0.93	71.47	0.92	72.48	0.49	**72.54**	0.43
Pixelate	78.19	0.12	75.48	0.32	79.26	0.21	**80.48**	0.31	80.20	0.49
Saturate	80.20	0.30	77.52	0.39	80.87	0.13	81.43	0.07	**81.65**	0.14
Shot noise	65.59	1.18	**71.91**	0.90	69.04	1.17	70.37	0.31	70.71	0.43
Snow	72.48	0.43	74.11	0.39	74.56	0.26	75.28	0.21	**75.78**	0.27
Spatter	75.84	0.45	71.63	0.69	76.10	0.42	76.10	0.19	**76.82**	0.26
Speckle noise	66.18	1.28	**71.79**	0.77	69.35	1.33	70.18	0.41	70.71	0.25
Zoom blur	70.06	0.36	72.07	1.23	72.79	0.82	73.54	0.31	**73.72**	0.19

Table 4.4: Test set accuracy and Corruption set accuracy on CIFAR-10 across all methods of structured dropout on layer 6 of VGG-11. SpatialDropoutMod refers to our modification to SpatialDropout and DropBlockRandom is only applying the selected mask to randomly selected feature maps. The results are averaged over 5 runs.

	SpatialDropout		SpatialDropoutMod		DropBlock		DropBlockRandom		CorrDrop	
	Mean (%)	std (%)	Mean (%)	std (%)	Mean (%)	std (%)	Mean (%)	std (%)	Mean (%)	std (%)
Test Set	83.70	0.15	82.88	0.33	84.99	0.03	85.27	0.14	85.45	0.05
Brightness	80.96	0.16	81.30	0.38	82.32	0.30	83.07	0.10	**83.23**	0.08
Contrast	54.71	0.31	**61.65**	0.21	58.78	0.84	58.77	0.11	60.24	0.43
Defocus blur	74.19	0.15	75.52	0.34	75.26	0.27	76.53	0.42	**76.67**	0.14
Elastic transform	73.68	0.10	75.08	0.25	75.13	0.17	75.96	0.12	**76.09**	0.05
Fog	69.98	0.34	**73.63**	0.35	72.16	0.80	72.50	0.07	73.43	0.50
Frost	70.25	0.60	73.68	0.39	72.08	0.56	**74.69**	0.62	74.65	0.21
Gaussian blur	69.90	0.21	72.13	0.55	70.85	0.45	72.49	0.54	**72.62**	0.21
Gaussian noise	60.22	1.77	64.35	0.78	61.84	1.81	**67.19**	1.46	65.58	1.44
Glass blur	56.99	0.85	**61.40**	0.84	58.59	0.90	61.30	1.46	60.21	0.61
Impulse noise	57.68	0.75	56.86	1.25	59.13	1.81	**61.05**	1.16	60.69	0.88
Jpeg compression	78.36	0.21	78.84	0.20	79.48	0.30	80.30	0.15	**80.31**	0.20
Motion blur	68.41	0.27	71.08	0.92	70.37	0.30	71.32	0.06	**71.92**	0.27
Pixelate	76.96	0.25	75.86	0.25	77.60	0.36	**79.98**	0.37	79.76	0.06
Saturate	78.38	0.12	77.18	0.47	79.32	0.04	**80.72**	0.06	80.62	0.16
Shot noise	65.99	1.35	69.03	0.54	67.26	1.41	**71.62**	1.16	70.38	1.11
Snow	71.81	0.24	73.73	0.22	73.26	0.08	74.81	0.50	**74.90**	0.14
Spatter	73.50	0.31	71.80	0.21	74.66	0.54	**75.04**	0.48	74.76	0.29
Speckle noise	66.36	1.08	68.95	0.43	67.67	1.42	**71.44**	1.26	70.42	1.10
Zoom blur	69.60	0.47	72.56	0.48	71.47	0.24	72.46	0.14	**72.68**	0.27

Table 4.5: Test set accuracy and Corruption set accuracy on CIFAR-10 across all methods of structured dropout on layer 8 of VGG-11. SpatialDropoutMod refers to our modification to SpatialDropout and DropBlockRandom is only applying the selected mask to randomly selected feature maps. The results are averaged over 5 runs.

test set accuracy and corrupted image classification accuracy improves but it remains to be explored which modification works better in practice.

Chapter 5

Conclusions & Future Directions

In this thesis, we have introduced new training strategies for convolutional neural networks. These strategies revolve around one of the differences between convolutional networks and the brain, namely, correlated variability. We have shown how convolutional networks can be trained with correlated noise (inspired from biological observations) and its role in improvement in robust image classification. In an effort to understand the role of correlated noise in input space, we compared correlations of activations of neural networks trained with dataset augmentations against biological correlations. Our results show the similarity, both quantitatively and qualitatively. We also presented two variants of Dropout suited for convolutional networks based on SpatialDropout and DropBlock. Our results show improvement only on the DropBlock variant in image classification on CIFAR-10. In the following, we provide future research directions to pursue in the area of correlated variability in convolutional networks.

- **Correlated Noise**
  - While our experiments were implemented on a modified version of AllConvNet for computational tractability, it would be interesting to implement this on popular architectures such as ResNets [12] trained on ImageNet [4]. This would involve developing an efficient method for sampling from very high-dimensional multivariate distributions.
  - The hyper-parameters for the correlation structure can be varied to control the dependence on distance and tuning. This may have an appreciable effect on image classification depending on which layer the noise is added to.

  - Poisson noise had worse image classification accuracy than all other models. We believe this might be due to the noisy straight-through estimator of the gradient of Poisson samples. Using the Gumbel-softmax relaxation, this problem may be alleviated. Neural networks with activations from a Poisson distribution are interesting because the inter-arrival times of spikes of cortical neurons follows a Poisson process.

- **CorrDrop**:

  - The next step to take is to implement CorrDrop, a variant of DropBlock using kernel correlations, on ResNets trained on ImageNet. Since CorrDrop involves minimal additional computation, it can scale to the ImageNet dataset and be used in larger neural network architectures.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[2] Larry F Abbott and Peter Dayan. The effect of correlated variability on the accuracy of a population code. *Neural computation*, 11(1):91–101, 1999.

[3] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[5] Terrance DeVries and Graham W Taylor. Dataset augmentation in feature space. *arXiv preprint arXiv:1702.05538*, 2017.

[6] Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[7] Apostolos P Georgopoulos, Andrew B Schwartz, and Ronald E Kettner. Neuronal population coding of movement direction. *Science*, 233(4771):1416–1419, 1986.

[8] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. Dropblock: A regularization method for convolutional networks. In *Advances in Neural Information Processing Systems*, pages 10727–10737, 2018.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

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

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

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

[13] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *arXiv preprint arXiv:1903.12261*, 2019.

[14] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[15] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *European conference on computer vision*, pages 646–661. Springer, 2016.

[16] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[17] Jörn-Henrik Jacobsen, Arnold Smeulders, and Edouard Oyallon. i-revnet: Deep invertible networks. *arXiv preprint arXiv:1802.07088*, 2018.

[18] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[19] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[20] Adam Kohn and Matthew A Smith. Stimulus dependence of neuronal correlation in primary visual cortex of the macaque. *Journal of Neuroscience*, 25(14):3661–3673, 2005.

[21] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[22] Alex Krizhevsky, I Sutskever, and G Hinton. Imagenet classification with deep convolutional neural. In *Neural Information Processing Systems*, pages 1–9, 2014.

[23] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. *arXiv preprint arXiv:1605.07648*, 2016.

[24] Choongkil Lee, William H Rohrer, and David L Sparks. Population coding of saccadic eye movements by neurons in the superior colliculus. *Nature*, 332(6162):357, 1988.

[25] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

[26] Rubén Moreno-Bote, Jeffrey Beck, Ingmar Kanitscheider, Xaq Pitkow, Peter Latham, and Alexandre Pouget. Information-limiting correlations. *Nature neuroscience*, 17(10):1410, 2014.

[27] Gergő Orbán, Pietro Berkes, József Fiser, and Máté Lengyel. Neural variability and sampling-based probabilistic representations in the visual cortex. *Neuron*, 92(2):530–543, 2016.

[28] MA Paradiso. A theory for the use of visual orientation information which exploits the columnar structure of striate cortex. *Biological cybernetics*, 58(1):35–49, 1988.

[29] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

[30] Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[31] Ben Poole, Jascha Sohl-Dickstein, and Surya Ganguli. Analyzing noise in autoencoders and deep networks. *arXiv preprint arXiv:1406.1831*, 2014.

[32] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[33] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[34] Matthew A Smith and Adam Kohn. Spatial and temporal scales of neuronal correlation in primary visual cortex. *Journal of Neuroscience*, 28(48):12591–12603, 2008.

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

[36] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[37] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

[38] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[39] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christoph Bregler. Efficient object localization using convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 648–656, 2015.

[40] Bryan P Tripp. Decorrelation of spiking variability and improved information transfer through feedforward divisive normalization. *Neural computation*, 24(4):867–894, 2012.

[41] Vikas Verma, Alex Lamb, Christopher Beckham, Aaron Courville, Ioannis Mitliagkis, and Yoshua Bengio. Manifold mixup: Encouraging meaningful on-manifold interpolation as a regularizer. *stat*, 1050:13, 2018.

[42] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International conference on machine learning*, pages 1058–1066, 2013.

[43] Xiaolong Wang, Abhinav Shrivastava, and Abhinav Gupta. A-fast-rcnn: Hard positive generation via adversary for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2606–2615, 2017.

[44] Daniel LK Yamins and James J DiCarlo. Using goal-driven deep learning models to understand sensory cortex. *Nature neuroscience*, 19(3):356, 2016.

[45] Daniel LK Yamins, Ha Hong, Charles F Cadieu, Ethan A Solomon, Darren Seibert, and James J DiCarlo. Performance-optimized hierarchical models predict neural responses in higher visual cortex. *Proceedings of the National Academy of Sciences*, 111(23):8619–8624, 2014.

[46] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[47] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

[48] Ehud Zohary, Michael N Shadlen, and William T Newsome. Correlated neuronal discharge rate and its implications for psychophysical performance. *Nature*, 370(6485):140, 1994.

[49] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.