

WAVELET FLOW: FAST TRAINING OF HIGH RESOLUTION NORMALIZING FLOWS

JASON JIASHENG YU

A THESIS SUBMITTED TO
THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

GRADUATE PROGRAM IN COMPUTER SCIENCE
YORK UNIVERSITY
TORONTO, ONTARIO

JULY, 2020

© Jason Jiasheng Yu, 2020

Abstract

Normalizing flows are a class of probabilistic generative models which allow for fast density computation, efficient sampling, and are effective at modelling complex distributions like images. A drawback among current methods is their significant training cost, sometimes requiring months of GPU training time to achieve state-of-the-art results. This thesis introduces Wavelet Flow, a multi-scale, normalizing flow architecture based on wavelets. A Wavelet Flow has an explicit representation of signal scale that inherently includes models of lower resolution signals and conditional generation of higher resolution signals, *i.e.*, super resolution. A major advantage of Wavelet Flow is the ability to construct generative models for high resolution data (*e.g.*, 1024×1024 images) that are impractical with previous models. Furthermore, Wavelet Flow is competitive with previous normalizing flows in terms of bits per dimension on standard (low resolution) benchmarks while being up to $15\times$ faster to train.

Acknowledgements

I thank my co-supervisors Marcus Brubaker and Kosta Derpanis for their guidance, support, and advice provided to me in my time as their student. I have been very fortunate to have supervisors that were involved in the project as much as I was, and who made time for frequent meetings.

I thank members of the academic community and individuals who freely share their articles, posts, lectures, and other educational material. Accessibility to such content has greatly reduced barriers to education, and benefited many people including myself.

I acknowledge the financial support of the Ontario Graduate Scholarship (OGS) awarded by the Province of Ontario. This research was undertaken as part of the Vision: Science to Applications (VISTA) program at York University, thanks in part to funding from the Canada First Research Excellence Fund. This research was also supported in part by the Mathematics of Information Technology and Complex Systems (MITACS) Accelerate program, in partnership with Borealis AI.

Contents

Abstract	ii
Acknowledgements	iii
Contents	iv
List of Tables	vi
List of Figures	vii
1 Introduction	1
1.1 Contributions	3
2 Background	5
2.1 Generative methods	5
2.1.1 Autoregressive models	6
2.1.2 Variational autoencoders	7
2.1.3 Generative adversarial networks	8
2.1.4 Normalizing flows	9
2.2 Multi-scale representations	17
2.2.1 Wavelets	19
2.2.2 Multi-scale generative methods	20

3	Wavelet Flow	24
3.1	Training	26
3.2	Sampling	26
3.3	Summary	32
4	Experimental Evaluation	33
4.1	Implementation and training	34
4.2	MCMC sampling procedure	35
4.3	Datasets and evaluation	37
4.4	Results	40
4.5	Discussion	51
5	Conclusion	55
5.1	Broader impact	56
5.2	Future work	57
	Bibliography	59
	Appendix	70
A.1	Annealed Gaussian distribution	70
A.2	Experimental setup	71
A.3	Additional samples	73

List of Tables

4.1	Parameter counts of Wavelet Flow and baselines	34
4.2	Wavelet Flow quantitative results with baselines	39
4.3	Wavelet Flow training times	40
A.1	Hyper-parameters for Wavelet Flow instances	72

List of Figures

2.1	Change of variables for probability density functions	11
2.2	Discrete distributions vs. dequantized distributions	15
2.3	A Laplacian pyramid	16
2.4	Haar transformation	18
2.5	Haar pyramid	21
3.1	Wavelet Flow architecture	25
3.2	Annealing a one dimensional distribution	28
4.1	Flow steps	35
4.2	Coupling network architecture	36
4.3	Low resolution annealed samples: LSUN	43
4.4	Low resolution annealed samples: ImageNet, CelebA-HQ, and FFHQ . .	44
4.5	Annealed sampels from additive and affine flows	45
4.6	High resolution annealed samples from CelebA-HQ	46
4.7	High resolution annealed samples from FFHQ	47
4.8	Wavelet Flow super resolution examples	48
4.9	Marginal distributions of detail coefficients	49
4.10	Wavelet Flow failure case with asymmetric eyes	52
4.11	Wavelet Flow failure case with inconsistent texture	53
A.1	Samples from ImageNet 32×32	74
A.2	Samples from ImageNet 64×64	75

A.3	Samples from LSUN bedroom 64×64	76
A.4	Samples from LSUN tower 64×64	77
A.5	Samples from LSUN church 64×64	78
A.6	Samples from CelebA-HQ 256×256	79
A.7	Samples from FFHQ 256×256	80
A.8	Annealed samples from ImageNet 32×32	81
A.9	Annealed samples from ImageNet 64×64	82
A.10	Annealed samples from LSUN bedroom 64×64	83
A.11	Annealed samples from LSUN tower 64×64	84
A.12	Annealed samples from LSUN church 64×64	85
A.13	Annealed samples from CelebA-HQ 256×256	86
A.14	Annealed samples from FFHQ 256×256	87
A.15	Super resolution on CelebA-HQ	88
A.16	Super resolution on FFHQ	89

Chapter 1

Introduction

Probabilistic modeling is a challenging and core problem in machine learning and statistics. Models designed for tractable sampling are directly useful for generating new data, such as images. Inference over these models can provide likelihoods, which are useful for detecting outliers. Many methods have been developed for probabilistic modeling, each having their own trade-offs. One class of these methods, *normalizing flows*, is the primary focus of this thesis.

Normalizing flows [11, 12, 33, 45] are a class of probabilistic generative models that use invertible transformations to convert simple distributions into complex ones. They enable fast density computation, efficient sampling, and have been shown to be effective at modelling complex distributions, *e.g.*, images [31, 21], videos [35], and audio [14, 29, 47]. Current approaches come with a significant computational cost, typically requiring months of graphics processing unit (GPU) training time to achieve state-of-the-art results.

In this thesis, we introduce *Wavelet Flow*, a multi-scale, conditional normalizing flow

architecture based on wavelets. Like other flow-based methods, Wavelet Flow is fast when sampling and computing probability densities, but are also efficient to train even with high resolution data. Further, the model has an explicit, wavelet-based, representation of scale which, among other benefits, includes consistent models of low resolution signals and conditional generation of higher resolutions (*i.e.*, super resolution). Wavelet Flow is applicable to any modality, *e.g.*, audio, images, videos, and volumetric data, where the natural scale structure of the signal can be exploited. Our experiments focus on images as they are the most widely considered domain with normalizing flows.

The computational cost of current approaches is partly driven by the requirement that normalizing flows be invertible and preserve dimensionality. Dinh et al. [12] identified this challenge and introduced a form of multi-scale flow which uses an identity transform on increasing subsets of dimensions to reduce computation. As part of this multi-scale flow, Ardizzone et al. [2] further proposed using a Haar wavelet [19] transform for reshaping and reducing spatial resolution within a normalizing flow. In either case, the structure progressively limits the complexity of the transformation in certain dimensions but neither exposes nor exploits the natural scale structure in the signal itself. In contrast, Wavelet Flow uses a wavelet transformation [42] of the data that naturally exposes its inherent scale structure. This transformation enables a factorization of the distribution across scale and a novel conditional architecture. Coarse structure is generated first with the generation of increasingly fine details, conditioned on the coarser structure.

Normalizing flows are typically trained using maximum likelihood estimation. In a limited capacity setting, this results in models that favour diversity over specificity. Spurious samples are often generated by normalizing flows, and image quality is reduced. One

method to alleviate this issue is to sample from an annealed version of the estimated distribution. Generally, this is not trivial, and requires the use of approximations via *Markov Chain Monte Carlo* (MCMC) methods. Applying MCMC methods to current normalizing flows is difficult due to model size. Qualitative results from normalizing flow papers (e.g. [31, 40]) avoid this issue by using a restricted model that is able to generate annealed samples directly. In contrast, Wavelet Flow is composed of multiple separate flows, which reduce the computational requirements needed during sampling; making the use of MCMC more tractable to allow sampling without having to sacrifice model expressiveness.

Wavelet Flow leverages the scale structure of images, as represented by wavelets, to allow for significantly more efficient training, and tractable annealed sampling without model restrictions. The results in this thesis demonstrate that Wavelet Flow is competitive with state-of-the-art normalizing flows on (typically low resolution) standard benchmarks while being up to 15 times more efficient to train in some cases. Further, we introduce the first normalizing flow model trained on high resolution images, *i.e.*, 1024×1024 .

1.1 Contributions

The contributions of this thesis centre around a proposed novel normalizing flow that directly exploits the scale structure of images. By leveraging this structure, the resulting model reduces barriers typically found when training and sampling from current normalizing flow methods. We make the following three contributions:

1. **Wavelet Flow** First, we propose a novel normalizing flow model based on a multi-scale signal representation produced using orthonormal wavelets. We show that

wavelets are convenient to use in conjunction with normalizing flows. Further, we show that the distribution over a wavelet representation of a signal has a natural coarse-to-fine conditional decomposition, which has a number of benefits including efficient and highly parallel training. Other benefits we explore include embedded, self-consistent distributions of lower resolution signals, and conditional generation of higher resolution signals (*i.e.*, super resolution).

2. **Novel MCMC sampling algorithm** Second, we propose a novel MCMC sampling algorithm for Wavelet Flow. Sampling from annealed normalizing flows without restrictions on the model requires the use of MCMC methods, which are typically not practical due to model size and dimensionality. The coarse-to-fine conditional decomposition of Wavelet Flow is exploited to introduce a novel sampling algorithm to draw samples from an annealed version of the resulting distribution.
3. **Quantitative evaluation** Finally, we compare against other normalizing flows on standard datasets, and demonstrate that Wavelet Flow has competitive performance while being up to $\sim 15\times$ faster to train. As a result, we are able to apply the Wavelet Flow architecture to images, including 1024×1024 high resolution imagery which has not been previously modeled with normalizing flows.

Chapter 2

Background

This chapter aims to summarize prominent generative methods in the context of image modeling, provide detailed background information required for the following chapters, and discuss other research into generative models with multi-scale representations.

2.1 Generative methods

Generative modeling is a problem in machine learning concerned with modeling distributions $p(\mathbf{X})$, where the random variable $\mathbf{X}$ describes a data-generating process over some modality, *e.g.*, images. Fitting these models to a finite number of data points provides an estimate for the true distribution. Models are generally designed to evaluate densities, draw samples, or both. Although these methods can be applied to many modalities, this section will focus on the basics of prominent generative methods used for modeling distributions of images.

2.1.1 Autoregressive models

Autoregressive models represent joint distributions over the dimensions of $\mathbf{x} \in \mathbb{R}^D$ by factoring them into a set of conditional distributions using the product rule of probability,

$$p(\mathbf{x}) = \prod_{i=1}^D p(x_i | x_1, \dots, x_{i-1}), \quad (2.1)$$

where x_i is the value of dimension i of $\mathbf{x}$. Each distribution of x_i is conditioned on values that precede it. The order of the dimensions is a model parameter, and is an important factor in the performance of autoregressive models [62].

Pixel recurrent neural networks (PixelRNNs) [61] are a class of discrete autoregressive models developed for images, where each pixel is a dimension of $\mathbf{x}$. Each conditional distribution is modeled using some form of neural network. The order of the pixels is arbitrary, but often takes a raster scan ordering, *i.e.*, starting from the top-left, and progressing left-to-right and top-to-bottom. It is not known whether the raster scan ordering used in PixelRNNs is optimal. Pixel intensities are usually stored as 8-bit integers, and the conditional distributions are probability mass functions over 2^8 bins. Training is performed by maximizing the likelihood of a set of observations. Image generation starts by sampling the first pixel, and conditionally sampling the remaining pixels. With careful design, probability estimation can be done in parallel; however, sampling must be done serially and thus is slow for large images. Autoregressive models [52, 46, 44, 50] are powerful, and significantly lead competing methods (*e.g.*, normalizing flows) in density estimation; they are currently the state-of-the-art in terms of average log likelihoods.

2.1.2 Variational autoencoders

Variational autoencoders (VAEs) [32] are a form of latent variable model. Each sample, $\mathbf{x} \in \mathbb{R}^n$, has a corresponding latent variable, $\mathbf{z} \in \mathbb{R}^m$, where typically $m \ll n$. The latent variable is typically drawn from an independent multi-variate Gaussian $p(\mathbf{z})$, and is used to sample $\mathbf{x}$ via the decoder distribution, $p(\mathbf{x}|\mathbf{z})$. In VAEs, the decoder is often realized as a parameterized Gaussian, $p(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}|\mu(\mathbf{z}), \sigma^2\mathbb{I})$, where μ is the mean parameterized by a neural network, $\mathbb{I}$ denotes the identity matrix, and the variance is a constant σ^2 . The likelihood of $\mathbf{x}$ is given by

$$p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z}. \quad (2.2)$$

Training involves maximizing the likelihood in Equation 2.2 over a set of observations; however, the integral makes direct optimization intractable. Instead, VAEs use approximate inference of the latent variable using an encoder distribution, $q(\mathbf{z}|\mathbf{x})$, also implemented as a Gaussian parameterized by a neural network. The intuition behind approximate inference is that most values of $\mathbf{z}$ will yield $p(\mathbf{x}|\mathbf{z}) \approx 0$. The encoder allows Equation 2.2 to be maximized efficiently by only optimizing where $p(\mathbf{x}|\mathbf{z})$ is not close to zero. Using Jensen’s inequality, $p(\mathbf{x})$ is maximized indirectly through the *Evidence Lower Bound* (ELBO):

$$\log p(\mathbf{x}) \geq \mathbb{E}_{q(\mathbf{z}|\mathbf{x})} [\log p(\mathbf{x}|\mathbf{z})] - D_{KL} [q(\mathbf{z}|\mathbf{x})||p(\mathbf{z})], \quad (2.3)$$

where D_{KL} is the Kullback-Leibler divergence. Sampling from a VAE starts by sampling a latent value $\mathbf{z}$, and then sampling from the decoder distribution, $p(\mathbf{x}|\mathbf{z})$. The approximate

inference of VAEs can make optimization challenging, and the quality of samples can be poor, *e.g.*, blurry.

2.1.3 Generative adversarial networks

Generative adversarial networks (GANs) [15] are a class of generative models trained via an adversarial objective. The main component is a generator function, $G(\cdot)$, which directly converts samples of a latent variable, $\mathbf{Z}$, into samples from $\mathbf{X}$. Instead of directly maximizing the likelihood over a set of observations from $\mathbf{X}$, training is performed through the adversarial objective, where a discriminator network, $D(\cdot)$, is tasked with the binary decision of whether samples are from the dataset or from the generator. The generator is optimized to produce samples that minimize the accuracy of the discriminator, while the discriminator is optimized to maximize it. This min-max objective is formally expressed as

$$\min_G \max_D \mathbb{E}_{\mathbf{x}} [\log(D(\mathbf{x}))] + \mathbb{E}_{\mathbf{z}} [\log(1 - D(G(\mathbf{z})))] , \quad (2.4)$$

where $\mathbf{x}$ is a sample from the dataset, $\mathbf{z}$ is a sampled latent value, and $D(\cdot)$ estimates the probability that its input is from the dataset. A fully trained model results in a discriminator that has an accuracy of 50%. Optimizing this objective approximately minimizes the Jensen-Shannon divergence between the true data distribution $\mathbf{X}$ and the estimated distribution of $G(\cdot)$. This adversarial framework is very flexible, and is capable of producing high quality images at high resolutions. A drawback of GANs is they only learn an implicit distribution which can only draw samples from the estimated distribution, but cannot

directly provide likelihoods. Training is only possible by using the adversarial objective, and is incompatible with maximum likelihood based training. They also do not guarantee full support over the domain; there are regions in the domain that cannot be produced by the model, even if those regions contain values from the training data. Further, GANs are known to experience instability during training, and there are instances where the model does not assign high likelihoods to values in the training data; the diversity is not captured in a phenomenon known as *mode collapse*.

2.1.4 Normalizing flows

Normalizing flows [33, 45] are an application of the change of variables formula for probability density estimation [10, 55]. If $\mathbf{X}$ is a random variable with values $\mathbf{x} \in \mathbb{R}^D$ then the density function of $\mathbf{X}$ can be written as

$$p_{\mathbf{X}}(\mathbf{x}) = p_{\mathbf{Z}}(\mathbf{f}_{\theta}(\mathbf{x})) |\det \mathbf{J}_{\theta}(\mathbf{x})|, \quad (2.5)$$

where $\mathbf{Z}$ is a random variable which takes on values $\mathbf{z} \in \mathbb{R}^D$, with probability density function $p_{\mathbf{Z}}(\mathbf{z})$ (*base distribution*), which we will assume to be Gaussian with zero mean and unit variance. $\mathbf{f}_{\theta}$ is an invertible, differentiable function parameterized by θ . The Jacobian of $\mathbf{f}_{\theta}$, $\mathbf{J}_{\theta} = \frac{\partial \mathbf{f}_{\theta}}{\partial \mathbf{x}}$, describes the instantaneous change in the probability mass of $\mathbf{x}$ after applying $\mathbf{f}_{\theta}$. Note that unlike VAEs, the dimensionality of both the latent variable, $\mathbf{z}$, and target distribution variable, $\mathbf{x}$, have the same dimensionality, D , due to the bijective requirement.

An example of a simple one dimensional normalizing flow is shown in Figure 2.1.

The base distribution, $p_{\mathbf{z}}(\mathbf{z})$, is a simple uniform distribution with support $[0, 2]$. Using the inverse of $\mathbf{f}(\mathbf{x}) = \frac{1}{2}\mathbf{x}^2$, the base distribution can be transformed into a more complex triangular distribution, $p_{\mathbf{x}}(\mathbf{x})$. In this one dimensional case, the Jacobian determinant is the absolute derivative, $\frac{d\mathbf{f}(\mathbf{x})}{d\mathbf{x}} = \mathbf{x}$. The number line plot in the centre of Figure 2.1 shows $\frac{d\mathbf{f}(\mathbf{x})}{d\mathbf{x}}$ as the dispersion of values in the mapping between $\mathbf{x}$ and $\mathbf{z}$.

Learning in normalizing flows consists of optimizing the parameters θ to maximize the log likelihood of a set of observations:

$$\arg \max_{\theta} \mathbb{E}_{\mathbf{x}} [\log p_{\mathbf{z}}(\mathbf{f}_{\theta}(\mathbf{x})) + \log |\det \mathbf{J}_{\theta}(\mathbf{x})|] . \quad (2.6)$$

The resulting distribution can be sampled from by drawing samples $\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})$ and then applying the inverse transformation $\mathbf{g} = \mathbf{f}^{-1}$ to yield $\mathbf{x} = \mathbf{g}(\mathbf{z})$, where the θ subscript will be dropped for clarity.

Designing $\mathbf{f}$ as a single component can be challenging. To aid in the design of invertible functions, it is convenient to define them as a composition

$$\mathbf{f}(\mathbf{x}) = \mathbf{f}_k \circ \mathbf{f}_{k-1} \circ \dots \circ \mathbf{f}_1(\mathbf{x}), \quad (2.7)$$

where $\mathbf{f}_i, i = 1, \dots, k$ is a simple invertible function. Conveniently, the inverse can be computed as the composition of the inverse functions in reverse order:

$$\mathbf{g}(\mathbf{z}) = \mathbf{g}_1 \circ \mathbf{g}_2 \circ \dots \circ \mathbf{g}_k(\mathbf{z}), \quad (2.8)$$

where $\mathbf{g}_i$ is the inverse of $\mathbf{f}_i$. The Jacobian determinant of this composition can also be

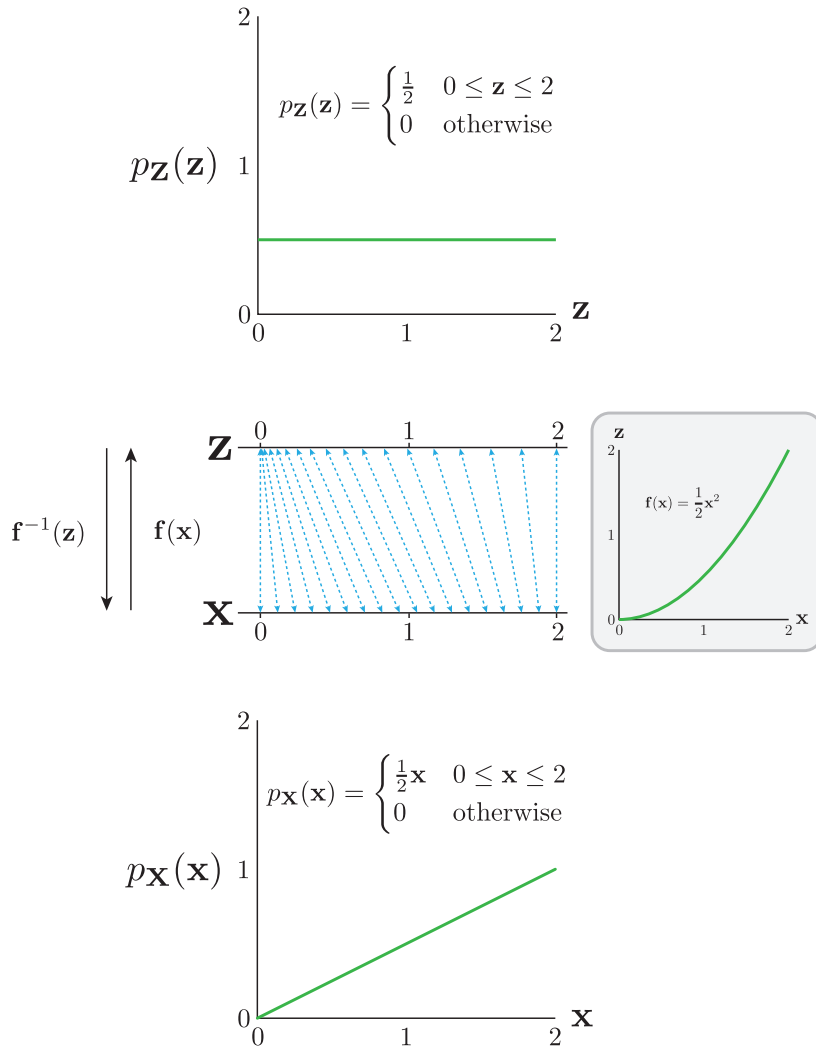


Figure 2.1: Change of variables for probability density functions. In this simple one dimensional normalizing flow the triangular distribution $p_{\mathbf{X}}(\mathbf{x})$ (bottom) can be transformed into the uniform distribution, $p_{\mathbf{Z}}(\mathbf{z})$ (top), through the invertible transformation, $\mathbf{f}$ (middle and middle-right). The inverse process is possible using inverse transformation, $\mathbf{f}^{-1}$. $\mathbf{f}$ is illustrated (middle) as an invertible mapping between the number lines of $\mathbf{z}$ and $\mathbf{x}$. The derivative of the transformation appears in the number line visualization as the dispersion of values in the mapping between $\mathbf{x}$ and $\mathbf{z}$. $\mathbf{f}$ is also illustrated (middle-right) using a standard 2D plot (middle-right), and is invertible within the support of the distributions, $[0, 2]$.

conveniently expressed as the product of the individual Jacobian determinants:

$$\det \mathbf{J}(\mathbf{x}) = \prod_{i=1}^k \det \mathbf{J}_i(\mathbf{x}_{i-1}), \quad (2.9)$$

where $\mathbf{x}_i = \mathbf{f}_i(\mathbf{x}_{i-1})$ are the outputs of the individual functions $\mathbf{f}_i$, $\mathbf{x}_0 = \mathbf{x}$ is the input to the flow, and $\mathbf{J}_i = \frac{\partial \mathbf{f}_i(\mathbf{x}_{i-1})}{\partial \mathbf{x}_{i-1}}$ is the Jacobian of $\mathbf{f}_i$.

Unlike the previously mentioned generative methods, normalizing flows do not require approximate inference of latent values, or auxiliary objectives. They also do not have any additional components (*e.g.*, discriminators or encoders) that are discarded. Since they are invertible, the model acts as its own encoder, which allows image manipulation in the latent space. However, as mentioned in Chapter 1, normalizing flows can be inefficient and require large amounts of compute to train.

Coupling layers The core challenge is constructing invertible and differentiable functions which are also efficient to compute and have easily computable Jacobian determinants. Two widely used functions are *additive* [11] and *affine* [12] *coupling* layers. An affine coupling layer, $\mathbf{f}_\theta$, splits an input into two disjoint subsets, $\mathbf{x} = (\mathbf{x}^A, \mathbf{x}^B)$, and applies a transformation

$$\mathbf{f}(\mathbf{x}) = (\mathbf{x}^A, \exp(\mathbf{s}(\mathbf{x}^A)) \odot \mathbf{x}^B + \mathbf{t}(\mathbf{x}^A)), \quad (2.10)$$

where $\odot$ denotes element-wise multiplication, and $\mathbf{s}_\theta$ and $\mathbf{t}_\theta$ are arbitrary functions, called *coupling networks*, which are typically parameterized deep networks of some form. The

Jacobian determinant of an affine coupling layer takes the form of

$$\frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}} = \begin{bmatrix} \mathbb{I} & 0 \\ \frac{\partial \mathbf{f}^B(\mathbf{x})^\top}{\partial \mathbf{x}^A} & \text{diag}(\exp[s(\mathbf{x}^A)]) \end{bmatrix}, \quad (2.11)$$

where $\mathbb{I}$ is an identity matrix, $\text{diag}(\exp[s(\mathbf{x}^A)])$ is a diagonal matrix with diagonal elements from $\exp[s(\mathbf{x}^A)]$, and $\mathbf{f}^B(\mathbf{x})$ is the second component of the tuple of Equation 2.10. The Jacobian is triangular, thus the determinant of an affine coupling layer is simply the product of the values of $\exp[s_\theta(\mathbf{x}^A)]$, *i.e.*, the diagonal of Equation 2.11. The inverse of a coupling layer is given by:

$$\mathbf{f}^{-1}(\mathbf{z}) = \left(\mathbf{z}^A, \frac{\mathbf{z}^B - \mathbf{t}(\mathbf{z}^A)}{\exp(\mathbf{s}(\mathbf{z}^A))} \right), \quad (2.12)$$

where $\mathbf{z} = (\mathbf{z}^A, \mathbf{z}^B)$. An additive coupling layer is simply an affine coupling layer without the scaling operation, *i.e.*, $\mathbf{f}_\theta(\mathbf{x}) = (\mathbf{x}^A, \mathbf{x}^B + \mathbf{t}_\theta(\mathbf{x}^A))$, and has unit Jacobian determinant.

Invertible 1×1 convolutions Coupling layers only transform part of their inputs. To ensure all dimensions are transformed, multiple coupling layers must be used while permuting the dimensions between each one. Dinh et al. [11, 12] used a fixed permutation, where the channels, c , in the input, $\mathbf{x} \in \mathbb{R}^{h \times w \times c}$, is reversed between coupling layers. Kingma and Dhariwal [31] proposed to learn the permutation using an invertible 1×1 convolution. The weights of the convolution have an equal number of input and output channels, which is equivalent to a square matrix. Convolution with the matrix inverse of $\mathbf{W}$ is used to compute the inverse. This layer has a determinant of $\det(\mathbf{W})^{h \cdot w}$. Computing

determinants of matrices can be expensive, but in practice this is typically not a limitation. When the number of dimensions is prohibitively large, $\mathbf{W}$ can be parameterized as its LU decomposition [56] to make determinants and inverses more efficient to compute.

Activation normalization Deep learning methods typically employ activation normalization methods. Batch normalization [24] is a common choice where an affine transformation is applied per channel, which uses the mean and variance of a minibatch. Kingma and Dhariwal [31] proposed the *actnorm* layer which performs a similar transformation, but using channelwise scale and bias parameters that are independent of the data. Initialization is performed using data dependent initialization where the parameters are set such that the output has zero mean and unit variance for an initial batch of training data. During training, the scale and bias parameters act as standard trainable parameters. The inverse is straightforward: given the input $\mathbf{x} \in \mathbb{R}^{h \times w \times c}$, the determinant is $\prod_{i=1}^c \alpha_i^{h \cdot w}$, where α_i is the i^{th} element of the scale parameter.

Dequantization The distribution over images stored using 8-bit integers is discrete. Despite this, continuous models can still be used to model discrete data; however, the density of a discrete random variable has a differential entropy of negative infinity [58], which can lead to arbitrarily high likelihoods during evaluation. Images must be *dequantized* [21, 23] to be compatible with normalizing flows, which are continuous models. Uniform dequantization [59, 60, 57] is a simple method where uniform noise is added to the data, as shown in Figure 2.2. When dequantizing integer data, the log likelihood is

$$\log p(\mathbf{x}) = \log \int p(\mathbf{x} + \mathbf{u}) d\mathbf{u}, \quad (2.13)$$

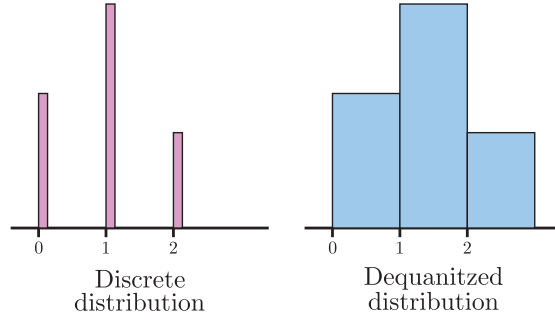


Figure 2.2: Discrete distributions vs. dequantized distributions. A discrete distribution over integer values (left), and the dequantized version using uniform dequantization (right). Without dequantization, the distribution would have a differential entropy of negative infinity, which would lead to arbitrarily high likelihoods during evaluation. Uniform dequantization is common and simple, but there exist more general methods [21, 23] that allow the dequantization distribution to be learned.

where $\mathbf{u}$ is the uniform dequantization noise with range $[0, 1)$. This integral is intractable, but a lower bound can be computed using importance sampling or Jensen’s inequality

$$\log p(\mathbf{x}) \geq \log \left[\frac{1}{k} \sum_{i=1}^k p(\mathbf{x} + \mathbf{u}_i) \right], \quad (2.14)$$

where k samples of $\mathbf{u}_i$ are taken from the uniform distribution. Results in normalizing flow research [12, 31] is typically reported using $k = 1$. Uniform dequantization is common and simple, but there exists more general methods [21, 23] that allow the dequantization distribution to be learned.

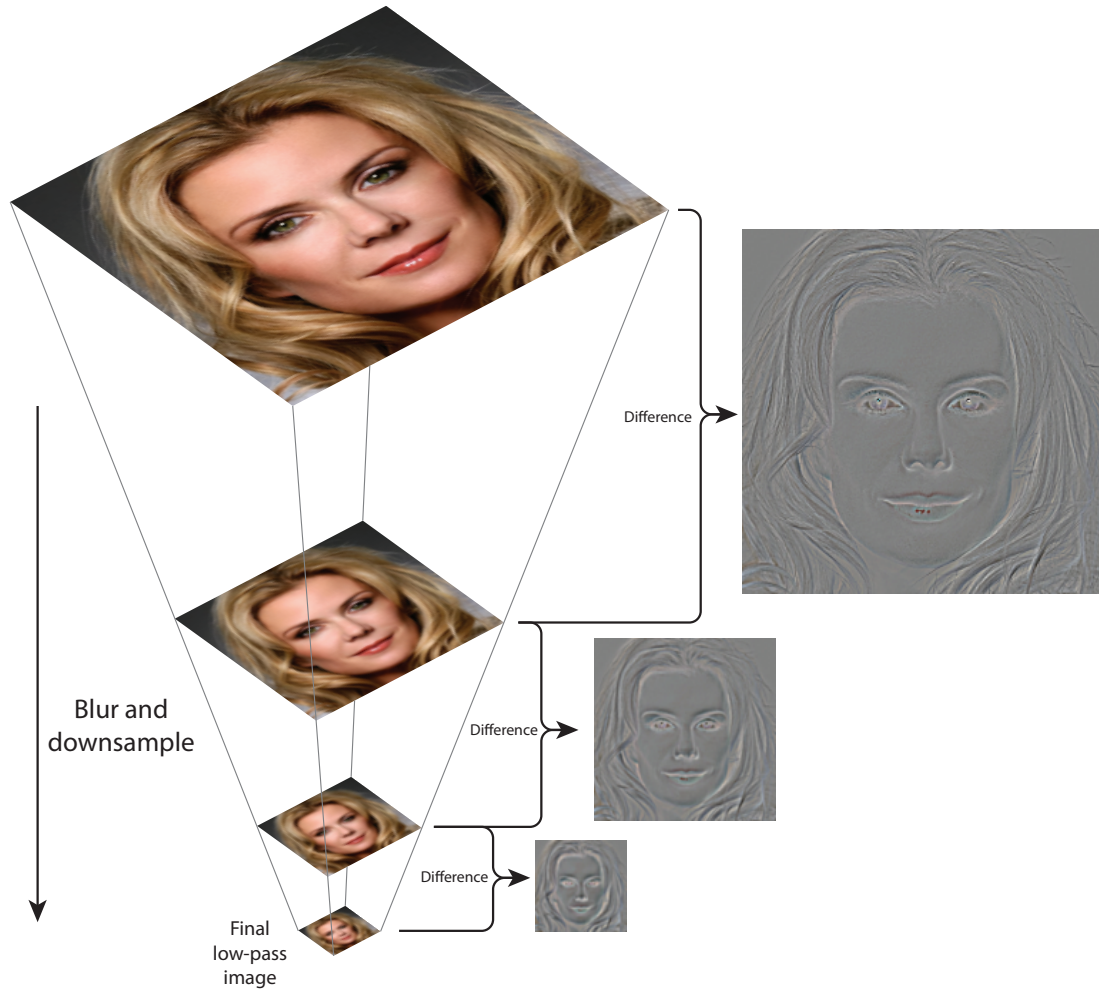


Figure 2.3: A Laplacian pyramid contains the difference images from each scale (right), and the final low-pass image (bottom). The difference images are created from the difference between the levels of a Gaussian pyramid (left). These difference images can be used to invert the decomposition and reconstruct the original image. Laplacian pyramids are $4/3$ overcomplete [54], which makes them surjective, but not bijective.

2.2 Multi-scale representations

Multi-scale representations have been an area of significant interest in computer vision for decades [7, 43, 65, 6, 8, 34, 1, 41, 37, 38, 4]. Their use is driven by the inherent scale structure that exists in natural signals [37]. The Fourier basis [3] is perhaps the best known multi-scale decomposition and is both invertible and orthonormal. The global basis functions of the Fourier transform render modelling in the Fourier domain challenging; the convolution operations in modern deep learning architectures are tailored for spatially structured signals.

Gaussian and Laplacian pyramids [7, 6, 8, 1, 37, 38] provide an overcomplete [54] but local representation for efficient coding and scale-sensitive representations. Gaussian pyramids repeatedly apply a Gaussian blur and downsample an image to capture multiple scales. The images at every scale are convenient for multi-scale analysis, but do not explain the information added at each scale. Laplacian pyramids are similar, but also contain a difference image between each scale, and the final low-pass image, as shown in Figure 2.3. This representation is $4/3$ overcomplete [54], and can reconstruct the original image. Starting with the initial low-pass image, it is upsampled and added to a difference image to create the low-pass image at the next scale. This is repeated until the original image is reconstructed. Since Laplacian pyramids are overcomplete, they are surjective and not bijective.

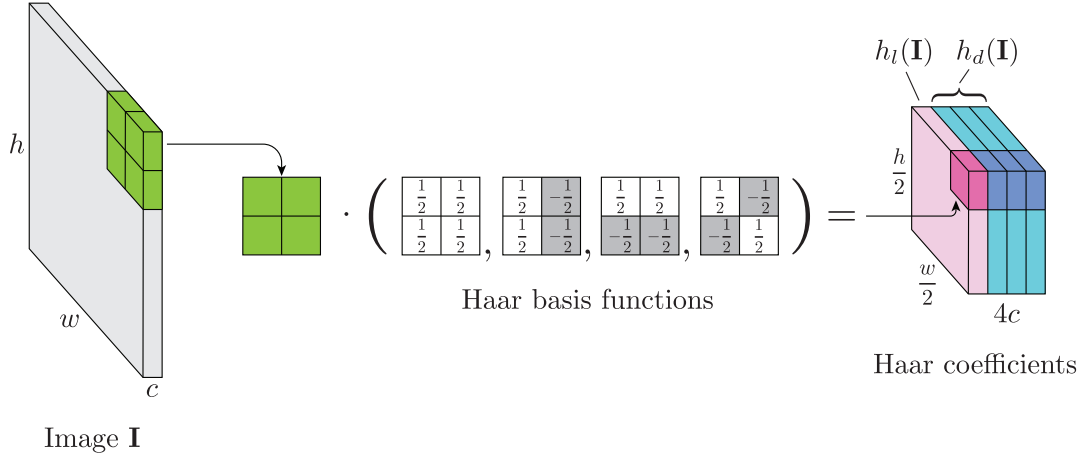


Figure 2.4: A Haar transformation operating on a single channel image. The transformation operates on 2×2 patches of an image, and can be implemented as a convolution with a 2×2 kernel with a stride of 2. The filters of the kernel acts as the basis functions of the Haar transformation. One of the basis functions acts as an low-pass averaging filter, while the remaining basis functions act as vertical, horizontal, and diagonal derivative filters. The outputs of the averaging filter and derivative filters constitute $h_l(\mathbf{I})$ and $h_d(\mathbf{I})$, respectively. For multi-channel images, the Haar transformation is applied independently across all channels.

2.2.1 Wavelets

Wavelets are a multi-scale decomposition of a signal similar in concept to a Fourier basis representation. Mallat [41] showed that wavelets could be used as a multi-scale, orthogonal and local representation of discrete signals which have since been widely used for a variety of applications including compression and restoration. The wavelets used in this thesis extract *detail* coefficients that explain the information added at each scale. While a Fourier basis is global in nature, wavelets are constructed to be localized, meaning that the value of a wavelet coefficient reflects the signal structure in a local region. This is particularly beneficial as modern deep learning architectures generally, including normalizing flows, are well tailored to spatially structured signal representations due to the widespread use of the convolution operation. Here, we briefly introduce wavelets; see [42] for a thorough introduction.

Consider an image $\mathbf{I} \in \mathbb{R}^{2^n \times 2^n \times C}$. The discrete wavelet transformation is constructed recursively as

$$\mathbf{I}_{i-1} = h_l(\mathbf{I}_i) \text{ and } \mathbf{D}_{i-1} = h_d(\mathbf{I}_i), \quad (2.15)$$

where $\mathbf{D}_{i-1} \in \mathbb{R}^{2^{i-1} \times 2^{i-1} \times 3C}$ are the detail coefficients at level $i - 1$, $h_l(\mathbf{I})$ applies the wavelet's low-pass filter, $h_d(\mathbf{I})$ applies the wavelet's high-pass filters, and both $h_l(\mathbf{I})$ and $h_d(\mathbf{I})$ use a stride of two, *i.e.*, spatial downsampling the result by a factor of two, and are applied channel-wise. Unlike Laplacian pyramids, which are overcomplete [54] and only surjective, wavelets are bijective and can be directly used with normalizing flows. The

wavelet transform is invertible and the original image can be recovered recursively by

$$\mathbf{I}_i = h^{-1}(\mathbf{I}_{i-1}, \mathbf{D}_{i-1}), \quad (2.16)$$

where h^{-1} is the inverse wavelet transform [42]. Applying the wavelet transform recursively until level 0 yields a complete representation of the image, where $\mathcal{H}(\mathbf{I}) = (\mathbf{I}_0, \mathbf{D}_0, \mathbf{D}_1, \mathbf{D}_2, \dots, \mathbf{D}_{n-1}) \in \mathbb{R}^{C2^{2n}}$ denotes the full wavelet transformation of an image. Note that $\mathcal{H}(\mathbf{I})$ has the same total dimensionality as $\mathbf{I}$. This representation decomposes the content of the image into a range of scales while retaining the dimensionality. At the coarsest scale, $\mathbf{I}_0 \in \mathbb{R}^{1 \times 1 \times C}$ represents the average intensity per channel, and $\mathbf{D}_0 \in \mathbb{R}^{1 \times 1 \times 3C}$ represents the global variations. At the finest scale, $\mathbf{D}_{n-1} \in \mathbb{R}^{2^{n-1} \times 2^{n-1} \times 3C}$ represents local variations.

There are many wavelets which have been designed and could be applied here. For simplicity, a Haar wavelet [19, 42] is used; however, any orthonormal wavelet could be used. Haar wavelets are fast and simple to apply and use only 2×2 filters. Each 2×2 patch yields $4C$ coefficients, $3C$ of them are comprised of the detail coefficients. The Haar wavelet transformation, shown in Figure 2.4, can be repeated to build a *Haar pyramid* as shown in Figure 2.5. A Haar pyramid decomposes the image in a manner similar to a Laplacian pyramid. Exploration of other wavelets remains a direction for future work.

2.2.2 Multi-scale generative methods

Multi-scale image representations have also been explored in other forms of generative modelling, including generative adversarial networks (GANs) [15] and variational auto-

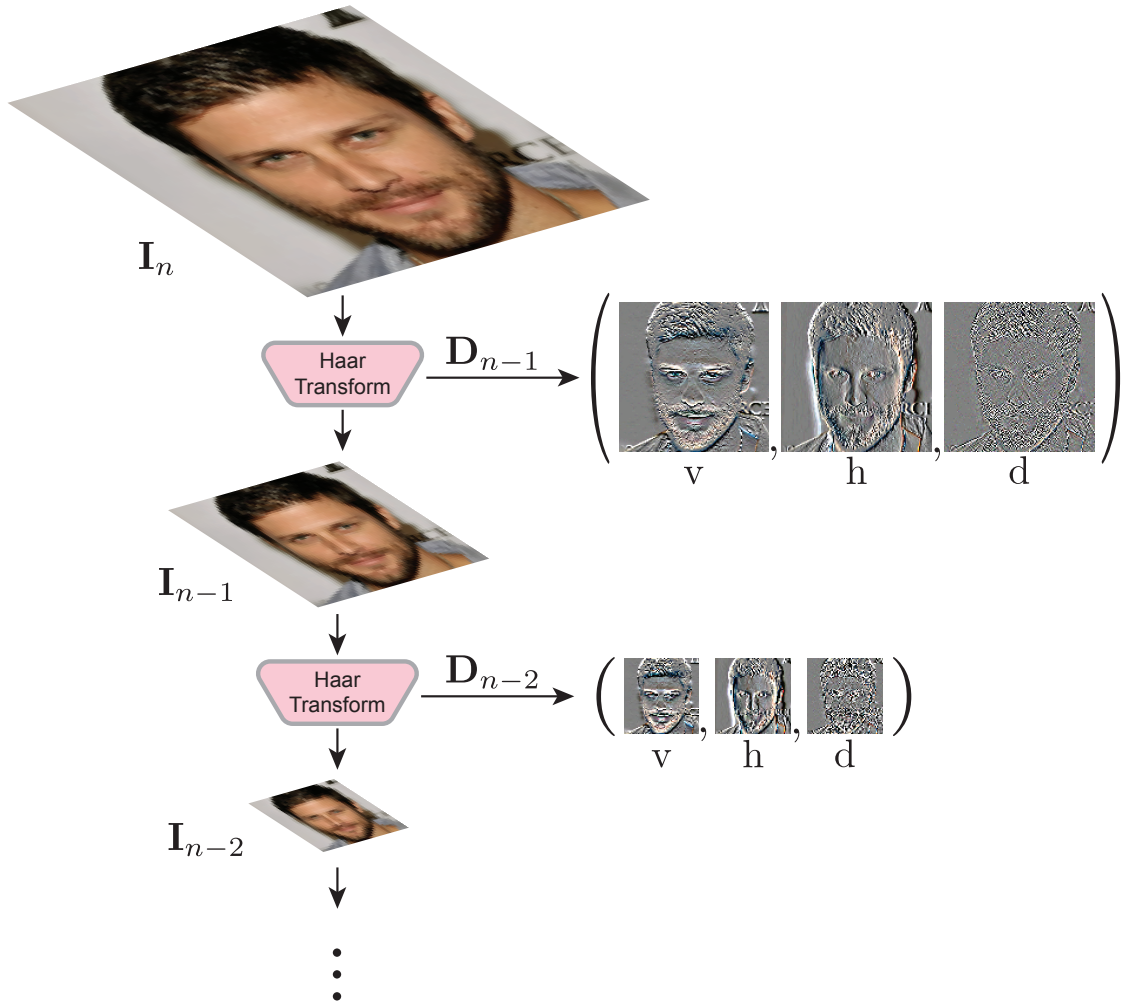


Figure 2.5: A Haar pyramid. By recursively applying a Haar transformation to an image, I_n , and extracting the details, D ., a Haar pyramid can be built. Like a Laplacian pyramid the information at each scale is factored out. In contrast to a Laplacian pyramid, the Haar pyramid is bijective. The Haar basis functions we have chosen result in the components of D . to represent vertical, horizontal, and diagonal derivatives of the image, labelled v , h , d , respectively.

encoders (VAEs) [32]. Their motivation is to simplify the modeling process by decomposing the difficult (holistic) generative process into separate model stages. Denton et al. [9] proposed a hierarchical GAN model consisting of a sequence of conditional generative modelling stages. Each stage models image structures at a particular scale (a residual image), as captured by a Laplacian pyramid [6], conditioned on the lower resolution image from the previous stage. The residual is then added to the lower resolution image from the previous stage and used as input for the subsequent generation of a higher resolution image. In addition, multiple discriminators were introduced, each dedicated to a particular image pyramid level. This is similar to our method which conditionally models the detail coefficients of a each scale of a wavelet representation. The Laplacian pyramid is an overcomplete representation making it unsuitable for use in a normalizing flow.

Various subsequent works [68, 27, 63, 26, 69, 53] also follow a stage-wise multi-scale image generation approach but forego the residual modeling and directly generate higher resolution images given lower resolutions inputs. More recently, Karras et al. [27] introduced progressive GAN training. Training of the model starts with a shallow architecture generating low resolution images and then adds layers to gradually increase the resolution of the images. In followup work, Karnewar et al. [26] removed the progressive training stage altogether, and consolidated the multiple discriminators dedicated to each image scale into a single one. Shaham et al. [53] also considered a multi-scale decomposition for image generation but with the goal of capturing the scale-dependent internal distribution of patches in a single image. PixelVAE [18] used a hierarchical structure in their latent space but did not explicitly represent signal scale. Razavi et al. [48] extended the vector quantized variational autoencoder (VQ-VAE) model [61], by introducing scale dependent

(discrete) latent codes, independently capturing local (*e.g.*, texture) and more global image aspects (*e.g.*, objects). Image generation was realized by a multi-stage PixelCNN [61] capturing the priors over the various scale-dependent latent maps. Dorta et al. [13] used a Laplacian pyramid with conditional VAEs in a manner similar to Wavelet Flow but primarily focused on image editing applications.

van den Oord et al. [61] described a multi-scale variant of the PixelRNN that generated a full higher resolution image conditioned on a lower resolution input; however, this structure did not constrain the conditional generation to the added content of the higher resolution image. Hence such a model can only be used for generation and does not allow for inference, *i.e.*, computing the probability density of a high resolution image. In contrast, Wavelet Flow conditionally generates only the detail coefficients of the wavelet representation at each scale and allows for inference of the probability density of an image at each modelled scale. Reed et al. [49] described a multi-scale autoregressive model, which consisted primarily of an alternative ordering of the pixels rather than a direct representation of image scale. The conditional structure of Wavelet Flow is reminiscent of autoregressive models [61, 52]. In these methods, conditioning is defined by an arbitrary pixel ordering, whereas the conditioning in Wavelet Flow is global but resolution limited.

Chapter 3

Wavelet Flow

To derive Wavelet Flow, we apply the change of variables [10, 55] to arrive at a distribution of the wavelet coefficients. Specifically, $p(\mathbf{I}) = p(\mathcal{H}(\mathbf{I}))|\det \frac{\partial \mathcal{H}}{\partial \mathbf{I}}|$, where we drop the random variable subscripts for clarity. To easily calculate the determinant of the transform, we require that the selected wavelet is orthonormal¹ and hence $|\det \frac{\partial \mathcal{H}}{\partial \mathbf{I}}| = 1$. Specifically, we use a Haar wavelet [19, 42]. For an image $\mathbf{I}_i$ at level $0 \leq i \leq n$, its distribution in terms of an orthonormal wavelet decomposition can be derived using the product rule of probability [10, 55], as follows:

$$p(\mathbf{I}_i) = p(h^{-1}(\mathbf{I}_{i-1}, \mathbf{D}_{i-1})) \quad (3.1)$$

$$= p(\mathbf{I}_{i-1}, \mathbf{D}_{i-1}) \quad (3.2)$$

$$= p(\mathbf{D}_{i-1}|\mathbf{I}_{i-1})p(\mathbf{I}_{i-1}), \quad (3.3)$$

¹As commonly implemented, the Haar wavelet filters are often unnormalized and hence orthogonal but not orthonormal. This can be rectified by a suitable rescaling.

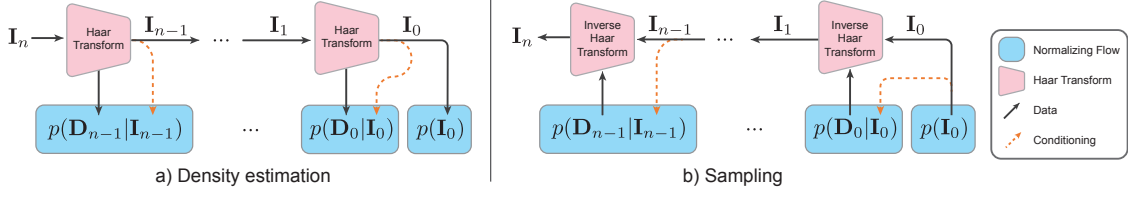


Figure 3.1: Wavelet Flow architecture. Wavelet Flow (a) density estimation, and (b) sampling. The normalizing flows at each level (blue) can be trained independently from each other for easy parallelization. Sampling is performed in a coarse-to-fine manner, where each flow at level i is conditioned (orange) on the lower resolution image, $\mathbf{I}_{i-1}$, to produce detail coefficients, $\mathbf{D}_{i-1}$. Images, $\mathbf{I}_i$, are reconstructed using the inverse Haar transform, $h^{-1}(\mathbf{I}_{i-1}, \mathbf{D}_{i-1})$.

where $h^{-1}(\cdot, \cdot)$ denotes the inverse wavelet transformation, and $\mathbf{D}_{i-1}$ are the detail coefficients of $\mathbf{I}_i$. This factorization can be applied recursively to build a multi-scale representation with a distribution of

$$p(\mathbf{I}) = p(\mathbf{I}_0) \prod_{i=0}^{n-1} p(\mathbf{D}_i | \mathbf{I}_i), \quad (3.4)$$

where $\mathbf{I} \in \mathbb{R}^{2^n \times 2^n \times C}$ with C channels. In other words, instead of modelling the entire image directly, a Wavelet Flow models the distribution of detail coefficients, conditioned on the lower resolution image. Each conditional distribution of detail coefficients, $p(\mathbf{D}_i | \mathbf{I}_i)$, and distribution of average intensity, $p(\mathbf{I}_0)$, is constructed using a normalizing flow. Note that inherent in this structure are distributions of all coarser resolution images, *e.g.*, $p(\mathbf{I}_k) = p(\mathbf{I}_0) \prod_{i=0}^{k-1} p(\mathbf{D}_i | \mathbf{I}_i)$ for all $k \leq n$. Hence, a Wavelet Flow model trained at a higher resolution can be used as a model of lower resolution images. The general structure of a Wavelet Flow is shown in Figure 3.1 and the specific normalizing flow architecture used for these distributions is described in Chapter 4.

3.1 Training

Similar to other normalizing flow models, a Wavelet Flow is trained by maximizing the log likelihood. Specifically, we seek to maximize

$$\log p(\mathbf{I}) = \log p(\mathbf{I}_0) + \sum_{i=0}^{n-1} \log p(\mathbf{D}_i | \mathbf{I}_i) \quad (3.5)$$

over a set of sample images. This reveals a significant advantage of a Wavelet Flow: the conditional distribution of detail coefficients for each level can be trained *independently*. This makes training more efficient, as it is easily parallelized with no communication overhead, gradient checkpointing or approximations, and the individual distributions are generally smaller and simpler, *i.e.*, they describe lower-dimensional distributions making them easier to fit in limited GPU memory.

3.2 Sampling

Sampling directly from a normalizing flow is efficient. However, maximum likelihood based training leads to an estimated distribution that favours diversity over specificity. Samples drawn directly from the model are often spurious and have reduce image quality. Qualitative results using normalizing flows (*e.g.*, [31, 40]) are typically produced by sampling from an *annealed* version of the estimated distribution.

Sampling from a normalizing flow without annealing starts with sampling $\mathbf{z} \sim p(\mathbf{z})$. A sample $\mathbf{x}$ is obtained by applying the inverse flow transformation $\mathbf{g}(\mathbf{z})$. Drawing annealed samples involves sampling from $\pi(\mathbf{x}) \propto p(\mathbf{x})^\gamma$ with $\gamma > 1$. The amount of annealing can

also be specified using *temperature* $T = 1/\sqrt{\gamma}$, which is the convention used by Kingma and Dhariwal [31]. Annealing tightens the distribution around its modes and reduces the amount of spurious samples along with limiting sample diversity. An example of annealing a simple bi-modal distribution is shown in Figure 3.2. If the normalizing flow has a constant Jacobian determinant term (*e.g.*, it consists only of additive coupling layers) then $p(\mathbf{x})^\gamma \propto p_{\mathbf{z}}(\mathbf{f}(\mathbf{x}))^\gamma$ and hence samples can be drawn from a Gaussian distribution with its standard deviation scaled by temperature T and applying the inverse of the flow transformation. A derivation for annealed Gaussian distributions is provided in the Appendix A.1.

This simple annealing approach does not work with more general flows including, *e.g.*, the affine coupling flows used here. Consequently, previous approaches [31, 40] resort to restrictive models with additive couplings when annealing the distribution for qualitative evaluation while using more expressive models with affine couplings for quantitative evaluation. In this work, we opt to use a single model with affine couplings for both quantitative and qualitative evaluation.

Sampling from normalizing flows using MCMC To sample from an annealed flow with affine coupling layers, Markov Chain Monte Carlo (MCMC) can be applied to the annealed, unnormalized distribution $p_{\mathbf{x}}(\mathbf{x})^\gamma$. Specifically, we use the No-U-Turn Sampler (NUTS) algorithm [22, 36] to generate samples. Using the change of variables formula, the target distribution for MCMC is the annealed normalizing flow, which can be written

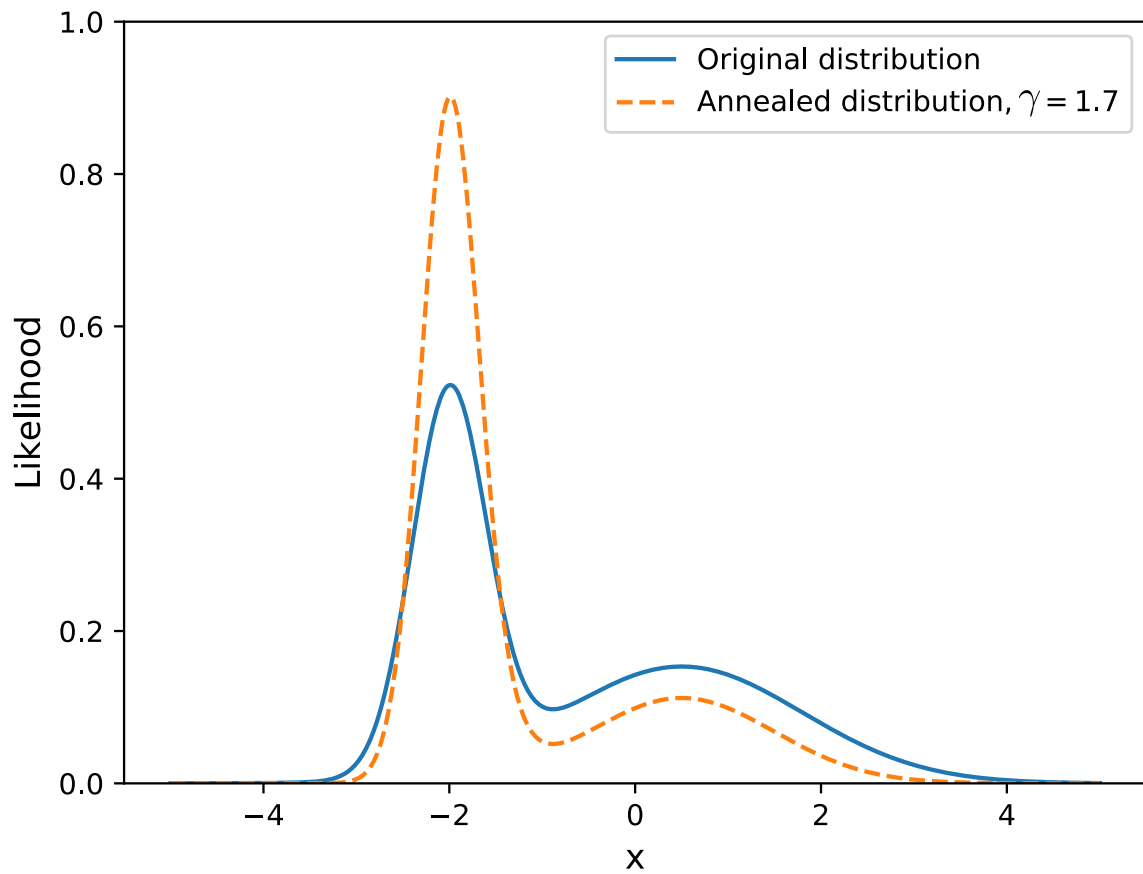


Figure 3.2: Annealing a one dimensional distribution. A one dimensional distribution with two modes (blue) is annealed with $\gamma = 1.7$ which results in the annealed distribution (orange). This tightens the distribution around the modes: making the values near the modes more likely.

as

$$\pi_{\mathbf{x}}(\mathbf{x}) \propto p_{\mathbf{x}}(\mathbf{x})^\gamma \quad (3.6)$$

$$= [p_{\mathbf{z}}(\mathbf{f}(\mathbf{x})) |\det \mathbf{J}(\mathbf{x})|]^\gamma \quad (3.7)$$

$$= p_{\mathbf{z}}(\mathbf{f}(\mathbf{x}))^\gamma |\det \mathbf{J}(\mathbf{x})|^\gamma, \quad (3.8)$$

where γ denotes the annealing parameter, $\mathbf{f}$ is the normalizing flow transformation, and $\mathbf{J}$ is the Jacobian of $\mathbf{f}$.

NUTS [22] requires only that the unnormalized probability density and its gradients can be evaluated and can be applied directly to $\pi_{\mathbf{x}}(\mathbf{x})$; however, the complex dependencies that exist between dimensions of $\mathbf{x}$ can produce a challenging geometry of the probability density which can be difficult for MCMC algorithms to sample from efficiently. Since we know the form of the density is closely related to a known normalizing flow, we can use the inverse of this flow, $\mathbf{g}$, to reparameterize the density such that it becomes exactly Gaussian (and hence easier to sample) when $\gamma = 1$. For $\gamma \neq 1$ the geometry should still be close to Gaussian and hence easier to sample from, particularly with values of γ close to 1. Reparameterizing the annealed distribution in terms of $\mathbf{z}$ gives:

$$\pi_{\mathbf{z}}(\mathbf{z}) \propto \pi_{\mathbf{x}}(\mathbf{g}(\mathbf{z})) |\det \mathbf{J}^{-1}(\mathbf{z})| \quad (3.9)$$

$$= p_{\mathbf{z}}(\mathbf{f}(\mathbf{g}(\mathbf{z})))^\gamma |\det \mathbf{J}(\mathbf{g}(\mathbf{z}))|^\gamma |\det \mathbf{J}^{-1}(\mathbf{z})| \quad (3.10)$$

$$= p_{\mathbf{z}}(\mathbf{z})^\gamma |\det \mathbf{J}^{-1}(\mathbf{z})|^{1-\gamma}, \quad (3.11)$$

where $\mathbf{J}^{-1} = \frac{\partial \mathbf{g}}{\partial \mathbf{z}}$ is Jacobian of $\mathbf{g}$, and the last line follows because $\mathbf{g}$ is the inverse of the

flow transformation $\mathbf{f}$, *i.e.*, $\mathbf{g} = \mathbf{f}^{-1}$, and hence their Jacobians (and their determinants) are also inverses of each other. NUTS is then used to perform MCMC on $\pi_{\mathbf{z}}(\mathbf{z})$. Samples from this Markov chain can then be transformed into samples from $\pi_{\mathbf{x}}(\mathbf{x})$ by computing $\mathbf{g}(\mathbf{z})$ like in a normalizing flow. In practice, we found that sampling in terms of $\mathbf{z}$ using the NUTS algorithm [22] is more efficient and can be done with larger steps and less divergences, compared to sampling in terms of $\mathbf{x}$. We use the implementation of NUTS provided in [36].

Sampling from Wavelet Flow Sampling from a Wavelet Flow starts with $\mathbf{I}_0 \sim p(\mathbf{I}_0)$ and proceeds recursively with

$$\mathbf{D}_{i-1} \sim p(\mathbf{D}_{i-1}|\mathbf{I}_{i-1}) \text{ and } \mathbf{I}_i = h^{-1}(\mathbf{I}_{i-1}, \mathbf{D}_{i-1}), \quad (3.12)$$

where sampling from each distribution is simply sampling from the corresponding normalizing flow, *i.e.*, sampling from the base distribution of that flow and applying its inverse flow transformation.

Using the distributive property of exponents, the annealed distribution of Wavelet Flow is

$$\pi(\mathbf{I}) \propto \left[p(\mathbf{I}_0) \prod_{i=0}^{n-1} p(\mathbf{D}_i|\mathbf{I}_i) \right]^\gamma \quad (3.13)$$

$$= p(\mathbf{I}_0)^\gamma \prod_{i=0}^{n-1} p(\mathbf{D}_i|\mathbf{I}_i)^\gamma \quad (3.14)$$

$$\propto \pi(\mathbf{I}_0) \prod_{i=0}^{n-1} \pi(\mathbf{D}_i|\mathbf{I}_i), \quad (3.15)$$

where $\pi(\cdot) \propto p(\cdot)^\gamma$ and $\pi(\cdot|\cdot) \propto p(\cdot|\cdot)^\gamma$ are the annealed versions of the distributions. The wavelet decomposition allows annealed samples to be drawn by individually sampling from the annealed detail distributions, which is more efficient than applying MCMC on the entire model at once. A sample is first drawn (using MCMC as described above) from $\pi(\mathbf{I}_0)$. Using the last sample from that chain, a sample from $\pi(\mathbf{D}_0|\mathbf{I}_0)$ is drawn and used to construct $\mathbf{I}_1 = h^{-1}(\mathbf{I}_0, \mathbf{D}_0)$. This last procedure is repeated for $\mathbf{D}_1, \mathbf{D}_2, \dots, \mathbf{D}_{n-1}$ until all detail coefficients have been generated, and a sample of $\mathbf{I}$ is drawn from the annealed distribution. The overall annealed sampling procedure is summarized in Algorithm 1.

Algorithm 1: Annealed sampling from Wavelet Flow using NUTS. $\gamma = 1/T^2$ is the annealing parameter, n is the number of levels in the Wavelet Flow, and m is the minimum number of NUTS steps. $\mathbf{z}_i$ is the base value of the flow $\mathbf{g}_i$ used to produce $\mathbf{D}_i$. Subscript b indicates components for the initial base image.

Given γ, n, m :

Sample $\hat{\mathbf{z}}_b \sim p_{\mathbf{z}_b}^\gamma(\mathbf{z}_b)$

Sample $\mathbf{z}_b \sim \pi_{\mathbf{z}_b}(\mathbf{z}_b)$ using NUTS, initialized with $\hat{\mathbf{z}}_b$, and m steps

Set $\mathbf{I}_0 \leftarrow \mathbf{g}_b(\mathbf{z}_b)$

for $i \leftarrow 0$ **to** $n - 1$ **do**

 Sample $\hat{\mathbf{z}}_i \sim p_{\mathbf{z}_i}^\gamma(\mathbf{z}_i)$

 Sample $\mathbf{z}_i \sim \pi_{\mathbf{z}_i}(\mathbf{z}_i|\mathbf{I}_i)$ using NUTS, initialized with $\hat{\mathbf{z}}_i$, and m steps

 Set $\mathbf{D}_i \leftarrow \mathbf{g}_i(\mathbf{z}_i|\mathbf{I}_i)$

 Set $\mathbf{I}_{i+1} \leftarrow h^{-1}(\mathbf{I}_i, \mathbf{D}_i)$

end

3.3 Summary

This chapter presented the formulation for Wavelet Flow along with a novel corresponding sampling algorithm. Using a Haar transformation, the distribution of an image was factored into the product of conditional distributions over the detail coefficients, and a distribution over base images; both were implemented as normalizing flows. The structure induced by this factorization allowed the distributions to be simpler, and enabled parallel training with no communication, gradient checkpointing or approximations.

Sampling from an annealed normalizing flow without restrictions to the model requires the use of MCMC methods. Since normalizing flows are invertible, the target distribution of an MCMC method can be reparameterized in terms of the latent variable, which we found makes sampling more efficient. Sampling from an annealed Wavelet Flow is also efficient since the conditional distributions can be sampled separately.

Chapter 4

Experimental Evaluation

We consider RealNVP [12] and Glow [31] as baselines for evaluation with uniform dequantization for all approaches. Comparison against these methods is only possible on lower resolution datasets as training on high resolution datasets is impractical. We note that improved architectures for a Wavelet Flow remains a direction of future work and could exploit newer flow architectures (*e.g.*, MaCow [40], Flow++ [21], and SoS Flow [25]) and alternative dequantization schemes [21, 23]. Hyper-parameters are set to produce models with parameter counts not exceeding those of Glow¹ [31] to enable a fair comparison. Controlling the number of parameters requires per-dataset hyper-parameter tuning because other approaches used dataset specific architectures. Specific hyper-parameter choices can be found in Appendix A.2. The resulting parameter counts from tuning our models are shown in Table 4.1 along with the parameter counts of the baselines.

¹Parameter counts for the Glow model for LSUN were determined from their provided logs and pre-trained model here: <https://github.com/openai/glow>. It should be noted that the hyper-parameters provided with their code result in a model with less parameters than the model used in their experiments.

Model	ImageNet 32×32	ImageNet 64×64	LSUN 64×64	CelebA-HQ & FFHQ 1024×1024
RealNVP [12]	46m	96m	96m	-
Glow [31]	66m	111m	111m	-
Wavelet Flow	64m	96m	47m	70m

Table 4.1: Comparison of the number of parameters used in RealNVP [12], Glow [31], and Wavelet Flow. Wavelet Flow always has less parameters than Glow, which is more comparable in terms of *bits per dimension* (BPD).

4.1 Implementation and training

As described in Section 3, the distributions $p(\mathbf{D}_i|\mathbf{I}_i)$ and $p(\mathbf{I}_0)$ are constructed using normalizing flows. Specifically, a Glow architecture [31] is used with slight modifications. This architecture is mainly comprised of a series of *flow steps*, each comprised of *act-norm*, *invertible 1×1 convolution*, and *coupling* layers, as shown in Figure 4.1. Other methods [12, 31] also include a multi-scale structure by reshaping and reducing spatial resolution within a normalizing flow. Wavelet Flows do not use the same internal multi-scale structure because they already leverage the natural scale structure of the signal itself.

Conditioning a coupling layer is performed by concatenating the lower resolution image to the input of its coupling network. Learning a conditional transformation of base distributions [64] was explored but not found to improve performance. We implement the coupling networks using a residual architecture [20], shown in Figure 4.2. A zero initialized convolutional layer is added at the end of this architecture, as recommended in [31]. This allows the coupling network to initialize as an identity function. As in [12], a *hyperbolic tangent* is applied to the scale component of the affine transform. To simplify the hyper-parameters, the number of output channels in all convolutional layers within a level,

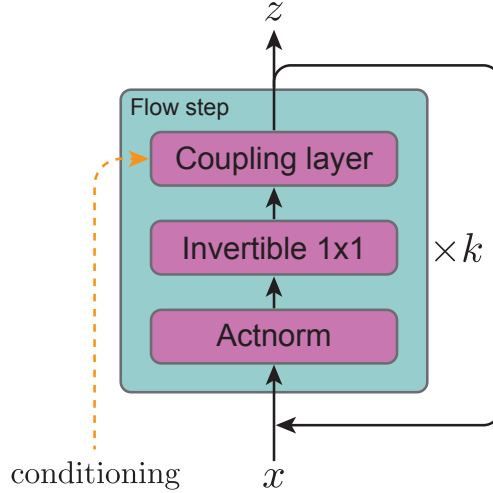


Figure 4.1: Flow steps. Flow steps are repeated k times, and each contain Actnorm, Invertible 1×1 , and Coupling layers. Conditioning is performed via a conditional coupling layer in each flow step, where the conditioning tensor is concatenated to the input of its coupling network.

is the same.

Training is done using the same Adamax optimizer [30] as in Glow [31]. The architecture of the flow is fully convolutional, and patch-wise training is used for the highest resolution conditional distributions. These implementation choices allow for distributions to be trained using a batch-size of 64 without gradient checkpointing on a single NVIDIA TITAN X (Pascal) GPU.

4.2 MCMC sampling procedure

In our experiments, the Markov chains are initialized with a sample from $p_{\mathbf{z}}(\mathbf{z})^\gamma$, which is straightforward since it is an annealed Gaussian and annealing simply scales the standard deviation, where $\gamma = 1/T^2$. This provides a reasonable initialization which is exact in

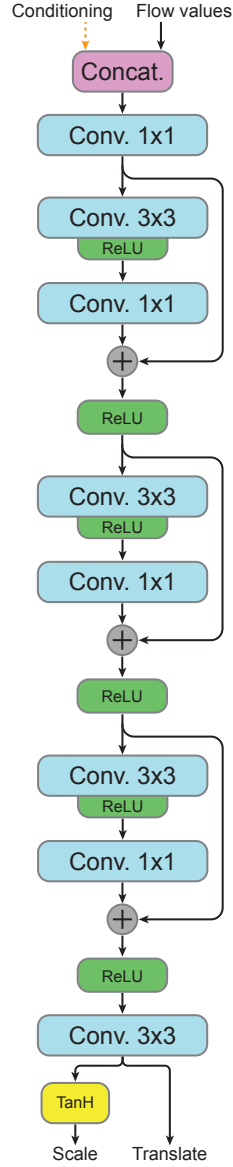


Figure 4.2: Coupling network architecture. The coupling network architecture consists of a 1×1 convolution, three residual blocks, and a final 3×3 convolution. Each residual block contains a 3×3 convolution followed by a 1×1 convolution. The final convolution is initialized with zeros so that the coupling layer initializes as an identity function. A hyperbolic tangent function is used to prevent the scale term from reaching extremely large or small values.

the case where the flow has a constant volume correction term or $T = 1$. Samples from this initialization are not proper samples from the annealed distribution but can still look reasonable, further suggesting that it is a good initialization. Dual averaging adaptation is used to adjust the step size for the first 10 steps of NUTS. [22]. A minimum of 30 steps are taken before the next accepted proposal is taken as the sample. These hyperparameters were chosen since they gave good qualitative performance without using too much compute. In our experiments, we observed that it takes roughly 30 minutes to sample an annealed image at 256×256 using $T = 0.97$ on CelebA-HQ. It roughly takes an additional 50 minutes and 35 hours to further reach resolutions of 512×512 and 1024×1024 , respectively. Levelwise sampling can exploit parallelism at coarser scales but GPU memory becomes a limit when sampling details at higher resolutions. The time it takes to draw samples using NUTS also is dependent on T with further from $T = 1$ generally taking longer. When $T \approx 1$, the target distribution is roughly Gaussian. For values of T that are far from 1, the curvature of the target distribution can become more complex, requiring smaller step sizes to avoid divergences. The times reported above are for $T = 0.97$.

4.3 Datasets and evaluation

To evaluate the performance of Wavelet Flow, we use several standard image datasets to directly compare against the reported results of previous methods. Specifically, we train and evaluate our model on natural image datasets at the commonly used resolutions and follow standard preprocessing procedures:

ImageNet [51] Two downsampled versions at resolutions of 32×32 and 64×64 are used. The training set consists of 1.28 million images, and the validation set contains 50,000 images. The validation set is used as the test set, which is also done in [12, 31].

LSUN [66] This dataset contains multiple categories, but experiments are only conducted on *bedroom*, *tower*, and *church* which are the most commonly used in generative modelling papers. The training sets of bedroom, tower, and church, contain 3 million, 700,000, and 126,000 images, respectively. The validation sets of all three categories contain 300 images each. Image dimensions in LSUN are not constant, pre-processing steps from [12] are performed to downsample the smallest side to 96 pixels before taking 64×64 random crops. No overfitting was observed on these datasets, and the validation set is used as the test set.

CelebA-HQ [27] This dataset is an extension of the original CelebFaces Attributes dataset [39]. The high resolution 1024×1024 images of CelebA-HQ are produced by applying inpainting and super resolution to aligned and cropped images. The 30,000 images in the dataset are separated into a training and test set using the same split as in [31]. The lower levels of the Wavelet Flow experienced overfitting, so a validation set is used to perform early stopping. The train, validation, and test sets contain 26,000, 1,000, and 3,000 images, respectively.

FFHQ [28] contains 70,000 high quality 1024×1024 aligned and cropped face images retrieved from Flickr. A training, validation, and test split were generated with 59,000, 4,000, and 7,000 images, respectively. As with CelebA-HQ lower levels of Wavelet Flow

Model	ImageNet [51]		LSUN [66]			CelebA-HQ [27]	FFHQ [28]
	32 × 32	64 × 64	bedroom	tower	church		
			64 × 64			1024 × 1024	1024 × 1024
RealNVP [12]	4.28	3.98	2.72	2.81	3.08	-	-
Glow [31]	4.09	3.81	2.38	2.46	2.67	-	-
Wavelet Flow	4.08	3.78	2.47	2.55	2.77	1.34	2.07

Table 4.2: Quantitative performance in bits per dimension (BPD). RealNVP [12] and Glow [31] are not evaluated on CelebA-HQ 1024 or FFHQ 1024 as training is impractical at those resolutions. Our method slightly outperforms Glow on both versions of ImageNet [51], but lags behind on LSUN [66]. We are still able to outperform RealNVP on LSUN.

experienced overfitting and early stopping is used.

Quantitative evaluations are based on average log likelihoods over a test set in the form of the commonly reported *bits per dimensions* (BPD) corresponding to intensity discretization into 2^8 bins. Given the probability of an image $p(\mathbf{I})$, the BPD is computed as:

$$\text{BPD}(\mathbf{I}) = \frac{-\log_2 p(\mathbf{I})}{hwc}, \quad (4.1)$$

where h, w, c are the dimensions of the image. All experiments are performed with the original 8-bit images. Kingma and Dhariwal [31] reported that reducing the bit-depth of the data to 5-bit can improve the quality of samples; however, no such effect was observed with Wavelet Flow and consequently we use the original bit-depth of the data.

Level	Resolution	FFHQ [28]	CelebA-HQ [27]	ImageNet [51]	LSUN [66] (bedroom)
0	1×1	6	6	6	6
1	2×2	6	6	96	12
2	4×4	6	6	48	14
3	8×8	6	6	168	36
4	16×16	18	12	168	48
5	32×32	120	36	168	120
6	64×64	120	120	168	264
7	128×128	120	120	-	-
8	256×256	120	120	-	-
9	512×512	156	120	-	-
10	1024×1024	168	120	-	-
Total		846	672	822	500

Table 4.3: Wavelet Flow training times in GPU hours on the evaluation datasets and across image scale. These times consider the reduced training time due to early stopping performed on FFHQ [28] and CelebA-HQ [27].

4.4 Results

Quantitative results in BPD are shown in Table 4.2. The Wavelet Flow models are competitive with other methods, outperforming Glow on ImageNet while somewhat underperforming Glow on LSUN but with significantly faster training, in both cases. Training times for each dataset and scale varied with the smallest scales taking a few hours and largest scales typically requiring five or six days. The total training times for ImageNet at 64×64 and CelebA-HQ at 1024×1024 were 822 and 672 GPU hours, respectively. Exact training times for Glow [31] were not reported in the paper but are estimated² to be approximately 6,700 GPU hours to train on CelebA-HQ at 256×256 and 3,700 GPU hours to train on LSUN bedroom at 64×64 . In contrast, training Wavelet Flow on those datasets and resolutions took approximately 430 and 500 GPU hours, respectively, $15\times$ and $7.5\times$

²Training times for Glow were estimated by inspecting the available training logs along with additional details reported by the lead author here: <https://github.com/openai/glow/issues/37/>

faster than Glow. It should be noted that our experiments on LSUN use significantly less parameters than Glow, shown in Table 4.1. A Wavelet Flow with more parameters would require more training time on LSUN, while potentially being more competitive in terms of BPD. More granular training times across image scale and datasets is shown in Table 4.3.

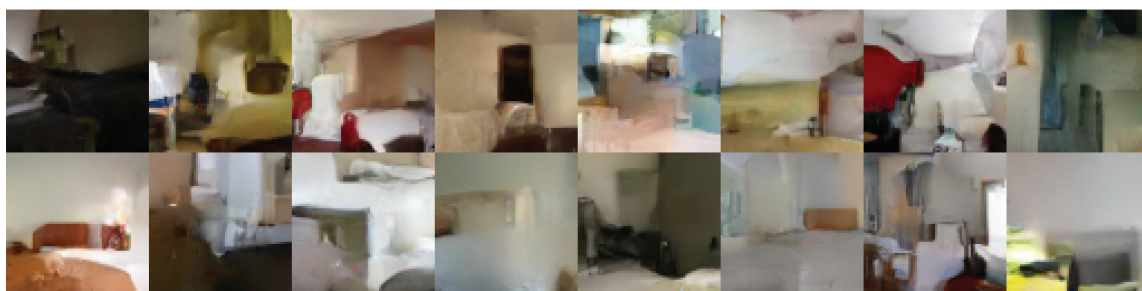
As described above, a Wavelet Flow can automatically be applied on lower resolution signals. To demonstrate this we use the model trained on ImageNet at 64×64 and truncate it to evaluate at 32×32 , getting a BPD of 4.16 vs. 4.08 with the model trained directly at 32×32 . This difference is due to dequantization [21, 23]. Specifically, dequantization noise is low-pass filtered in the wavelet transform along with the original signal and is no longer an independent and identically distributed uniform at lower resolutions and hence there are small (below a single pixel intensity increment) differences in the data distribution for the embedded lower resolution models. This was verified by using dequantization noise consistent with this low-pass filtered uniform noise during testing of the truncated model which achieved a BPD of 4.08, exactly the same as a Wavelet Flow trained directly on 32×32 images. Since the Haar transformation is linear, the consistent low-pass filtered uniform noise can be computed independently from the signal and added back at lower resolutions. Note that this does not occur when modelling a truly continuous signal, where dequantization is not necessary. Modern computers are discrete and so are models implemented on them. However, these models typically represent numbers using a 32-bit floating point representation which has enough precision to allow the model to be treated as continuous in practice. Further exploration of dequantization in Wavelet Flows remains a direction for future work.

To assess the qualitative performance of our model, we sampled from an annealed

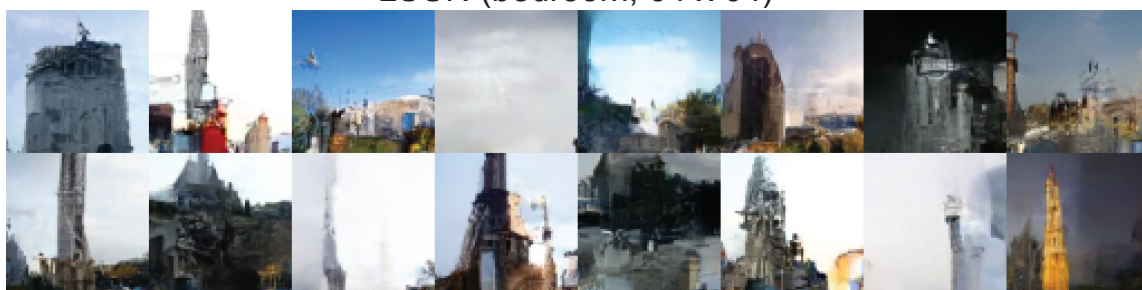
version of the estimated distribution using the MCMC approach described in Section 3. We explored a range of annealing parameters and found that only a small amount of annealing, *i.e.*, $T = 0.97$, was required to produce good visual results. In contrast, other methods have typically required a large amount of annealing to produce plausible images, *e.g.*, Glow [31] reported results using $T = 0.7$ for CelebA-HQ. Generated images for all datasets are shown in Figures 4.3, 4.4, and more are in Appendix A.3. Images generated at different values of T can be found in Figure 4.5c. Notably, extremely low temperatures become somewhat blurry, likely indicating that the annealed distribution of detail coefficients (see below) has become too peaked at zero at finer scales. Further, lower values of T may cause problems with convergence of the MCMC based sampling approach.

High resolution samples from Wavelet Flow trained on CelebA-HQ and FFHQ at 1024×1024 are shown in Figures 4.6 and 4.7, respectively. We also report a quantitative evaluation of Wavelet Flow on these datasets in Table 4.2 but note that comparison to other methods is not possible as training the baselines on high resolution imagery is impractical.

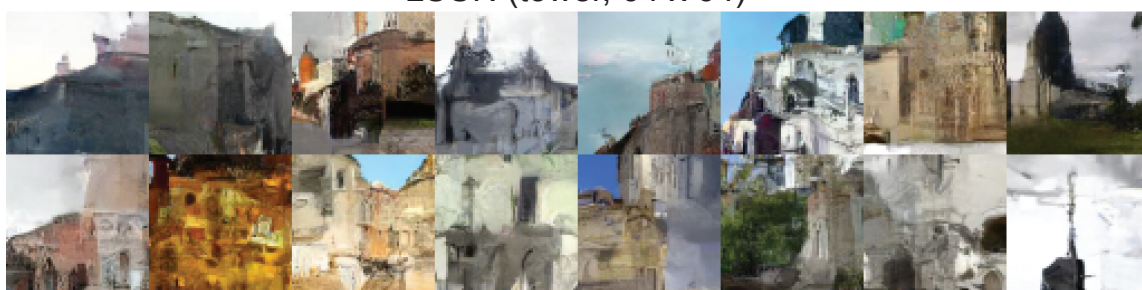
Super Resolution The conditional structure of a Wavelet Flow allows them to perform probabilistic super resolution by generating the detail coefficients given a lower resolution input image. That is, given an input image as $\mathbf{I}_i$ we can generate $\mathbf{D}_i$ by sampling from $p(\mathbf{D}_i|\mathbf{I}_i)$ and construct $\mathbf{I}_{i+1} = h^{-1}(\mathbf{I}_i, \mathbf{D}_i)$ to achieve a $2\times$ increase in resolution. This process can be applied iteratively to recover higher resolutions. An example of $8\times$ upsampling, from 128×128 to 1024×1024 , can be seen in Figure 4.8. Note that our model is not trained specifically for super resolution, rather this is a notable byproduct of the scale conditional structure of Wavelet Flow. We present these results only as a demonstration



LSUN (bedroom, 64 x 64)



LSUN (tower, 64 x 64)



LSUN (church, 64 x 64)

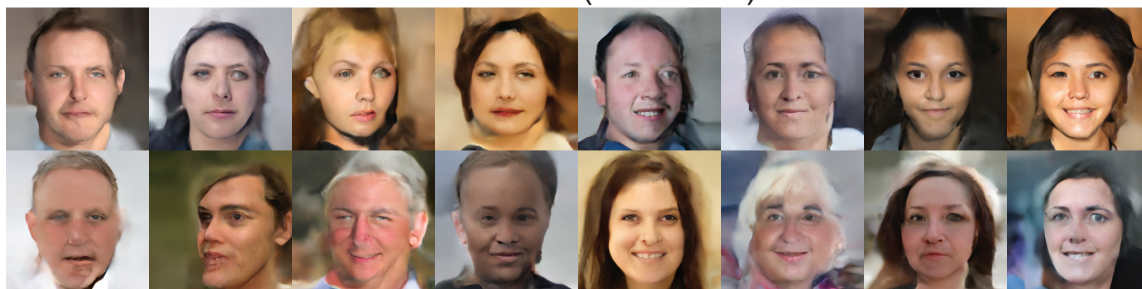
Figure 4.3: Low resolution annealed samples: LSUN. Wavelet Flow samples drawn using MCMC with $T = 0.97$, on LSUN [66].



Imagenet (64 x 64)



CelebA-HQ (256 x 256)



FFHQ (256 x 256)

Figure 4.4: Low resolution annealed samples: ImageNet, CelebA-HQ, and FFHQ. Wavelet Flow samples drawn using MCMC with $T = 0.97$, on ImageNet [51], CelebA-HQ [27], and FFHQ [28].

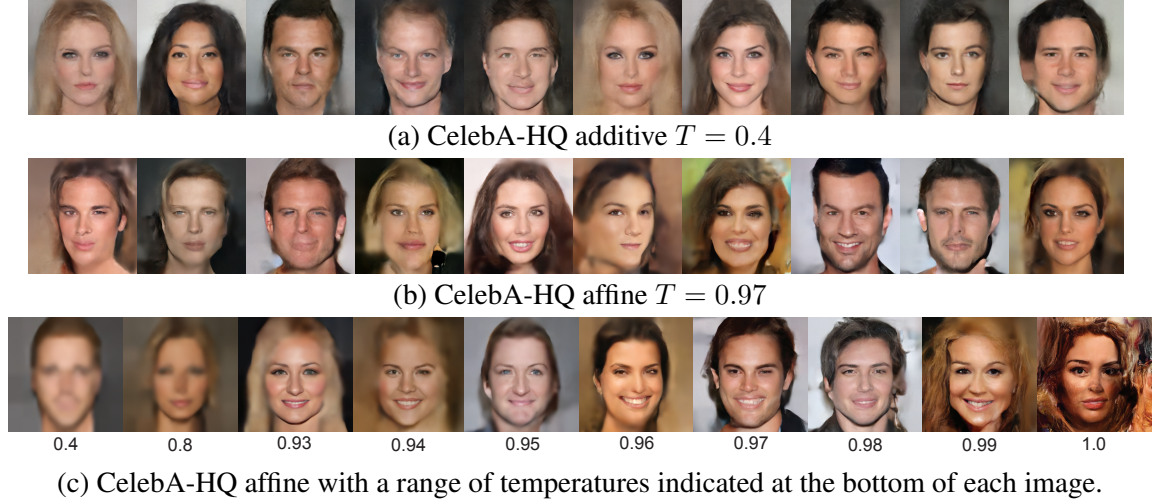


Figure 4.5: Annealed samples from additive and affine Wavelet Flows on CelebA-HQ.

and leave a full exploration of Wavelet Flow for super resolution as future work.

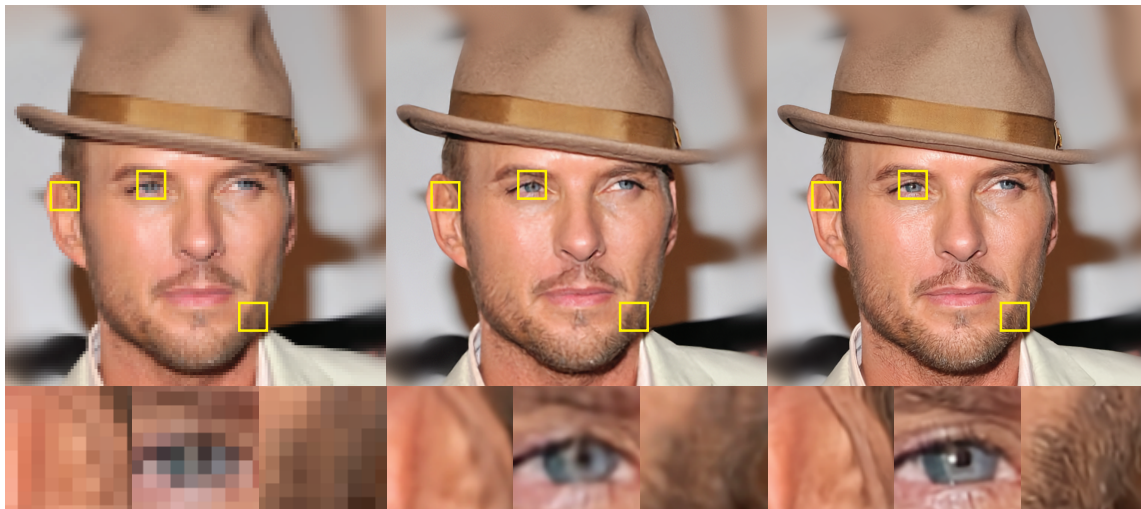
Effects of Annealing Training with maximum likelihood (ML) typically results in models that favour diversity rather than specificity due to its “mean-seeking” tendency. Kingma and Dhariwal [31] further speculated that flow-based models over-estimate the entropy of the distribution. As a result, images directly drawn from ML trained models tend to be less realistic and previous flow-based approaches (*e.g.*, [31, 40]) have instead shown samples drawn from an annealed version of the estimated distribution. We can see this over estimation of the entropy in the marginal distribution of the generated detail coefficients shown in Figure 4.9. Specifically, without annealing the peak of the learned distribution of wavelet details is lower. After annealing, the distribution of detail coefficients matches well with the data, further supporting the choice of $T = 0.97$. In contrast, with the additive model we see that even with $T = 0.4$ the distribution still does not match.



Figure 4.6: High resolution annealed samples from CelebA-HQ. Annealed high resolution (1024×1024) samples from CelebA-HQ with $T = 0.97$.



Figure 4.7: High resolution annealed samples from FFHQ. Annealed high resolution (1024×1024) samples from FFHQ with $T = 0.97$.



(a) CelebA-HQ $T = 0.97$



(b) FFHQ $T = 0.97$

Figure 4.8: Wavelet Flow super resolution examples. Samples are generated with $T = 0.97$ trained on (a) CelebA-HQ and (b) FFHQ from 128×128 (left) to 1024×1024 (middle), with ground truth (right) and zoomed regions (bottom) for comparison.

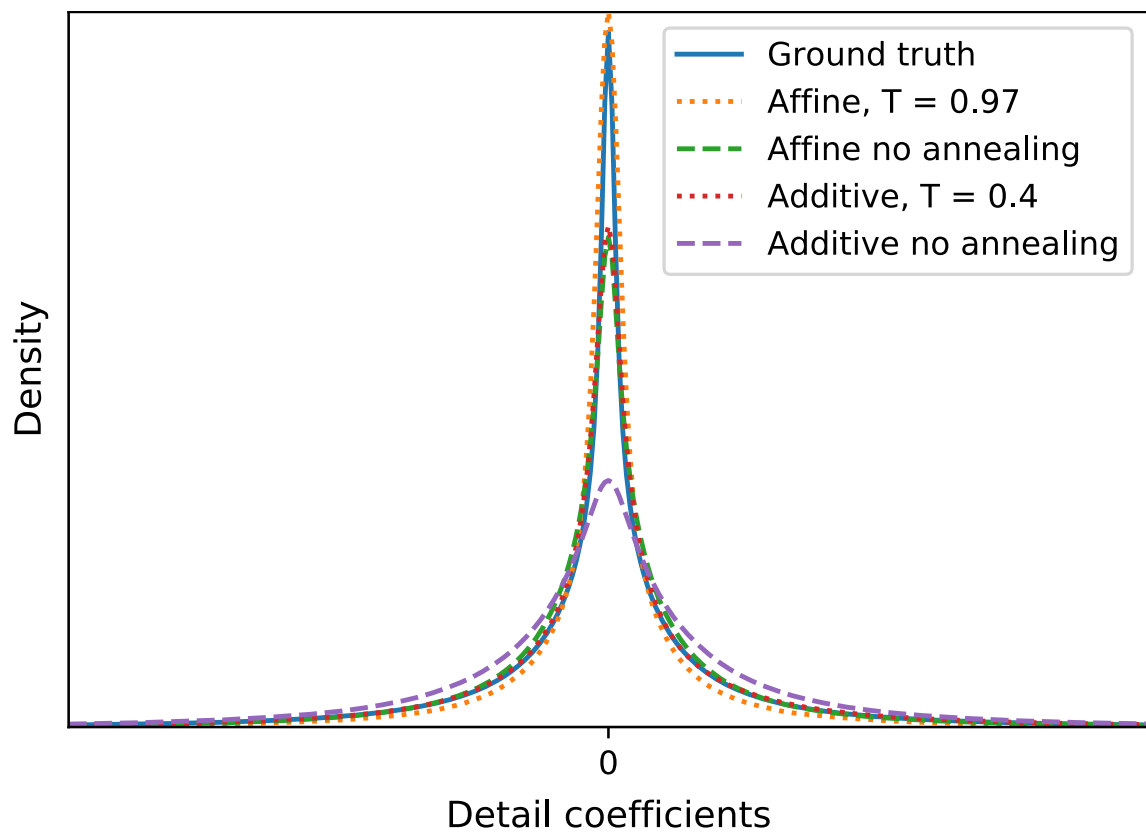


Figure 4.9: Marginal distributions of detail coefficients D_5 on CelebA-HQ. Wavelet Flows are considered with affine and additive coupling without annealing, and with annealing using $T = 0.97$ and $T = 0.4$ for affine and additive coupling, respectively. Annealing with affine coupling yields marginal detail coefficients that match the ground truth data most closely. Annealing with additive coupling makes the marginal detail coefficients more centrally peaked than without annealing, but not as much as with affine coupling.

Additive vs. Affine Coupling As discussed in Chapter 3, to draw samples from an annealed distribution in previous works (*e.g.*, [31, 40]), affine coupling layers were replaced with additive coupling layers to produce a model with constant Jacobian determinant. While affine layers generally produce better quantitative results [12, 31], it is unknown whether they are also qualitatively better when annealed. Here we present the first direct comparison between annealed additive and affine models. Figures 4.5a and 4.5b compare samples from Wavelet Flows with additive and affine layers. Our results demonstrate that, in the case of Wavelet Flow, additive and affine layers have similar image quality; however, more aggressive annealing of the additive model is required. Further, as shown in Figure 4.9, even with aggressive annealing, additive models still cannot match the marginal statistics of the detail coefficients as well as affine models. As a result we use a single, affine coupling-based model for both quantitative evaluation and sample generation.

Failure modes Example failure modes from a Wavelet Flow trained on CelebA-HQ are shown in Figures 4.10 and 4.11. Figure 4.10 is a sample generated with asymmetric eye colours. Although this is a possible trait in humans, these occurrences are far more frequent than in the training data. This asymmetry is difficult to see at lower resolutions, but begins to appear around 64×64 .

Figure 4.11 is an example of inconsistent details in two regions of hair. Hair typically has a regular structure over a long range. The model often generates regions of hair where the details become locally attenuated, and the texture of the hair becomes implausible. Inspecting the details at different scales reveals that these inconsistencies are mostly at the higher frequencies. Low frequency components, such as colour, appear to be consistent over larger distances.

We speculate that this failure is due to a reduction in the ability to capture long range dependencies as the resolution increases. A single coupling layer has a receptive field of nine pixels. The maximum number of steps used in our experiments is 16, giving a flow with a receptive field of 144 pixels. Patch-wise training also restricts the range further, especially at higher resolutions. These restrictions do not have a large impact on the flows at lower resolutions, which correspond to low frequencies where we do not observe long range inconsistencies.

4.5 Discussion

This chapter described our Wavelet Flow implementation, and provided both quantitative and qualitative evaluations. Our architecture was based on Glow [31], and hyperparameters were tuned such that the model parameter counts were similar or lower than the baseline methods. Standard datasets were used to quantitatively compare against baseline methods in terms of bits per dimension. Wavelet Flow was found to have comparable performance to the baselines while being more efficient, *i.e.*, using the same or fewer number of parameters, and training up to $15\times$ faster in wall-clock time. This efficiency also allowed for models that operate at resolutions up to 1024×1024 . Since this is the first normalizing flow to run at these resolutions, no comparisons could be made. Since Wavelet Flow is separable across scales, we demonstrated that it could also be used to perform super resolution, despite not being explicitly trained for this task.

Qualitative samples from normalizing flows are often drawn from annealed versions of the model. By analysing the marginal distribution of the detail coefficients, we observed

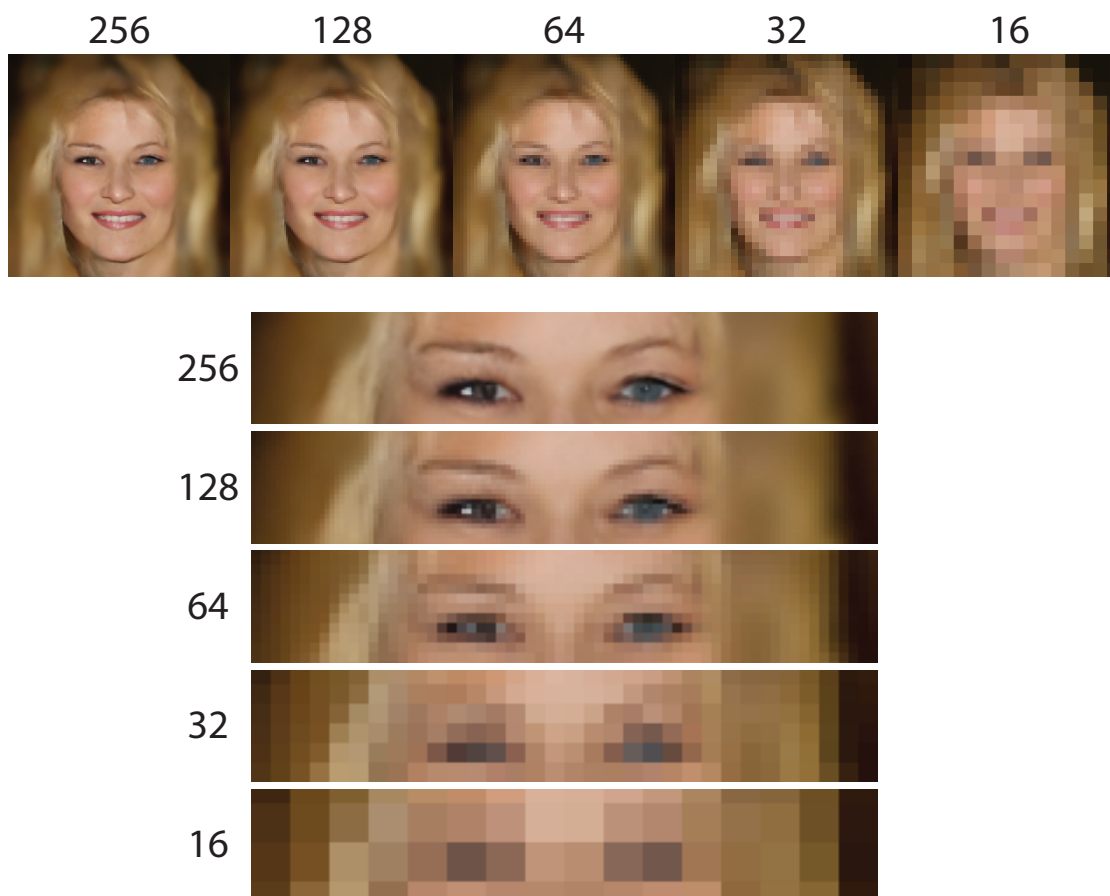


Figure 4.10: Wavelet Flow failure case with asymmetric eyes. A sample from Wavelet Flow trained on CelebA-HQ at different resolutions (top), and the same images enlarged around the asymmetrically coloured eyes (bottom). This kind of asymmetry is not common in the training data, and only becomes apparent at higher resolutions.

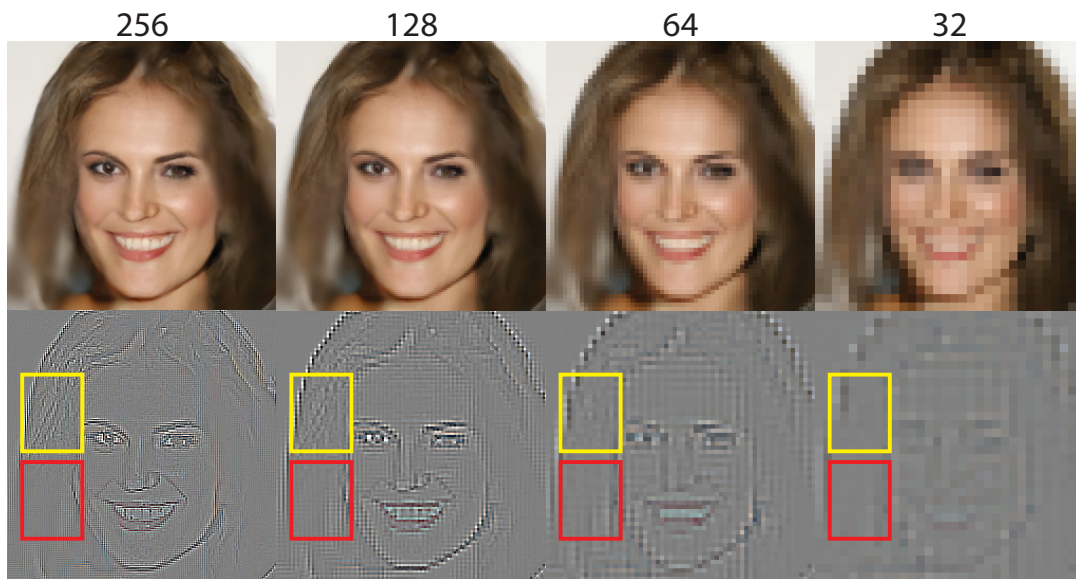


Figure 4.11: Wavelet Flow failure case with inconsistent texture. A sample from Wavelet Flow trained on CelebA-HQ at different resolutions (top), and corresponding visualizations of Haar detail coefficients (bottom). Two regions of interest are bounded by yellow and red boxes. In the image, the region in the yellow box has hair texture, while the region in the the red box is overly smooth. These regions in the Haar coefficient visualizations show the inconsistencies are mainly attributed to the high frequencies.

that Wavelet Flow generates detail coefficients with more variance than the data. By annealing the flow, the generated detail coefficients were much closer to the statistics of the data. This can be seen as improved quality in the annealed samples. In the case of Wavelet Flow, there is a drawback in restricting the model with additive coupling layers, *i.e.*, the marginal distribution of the detail coefficients do not match as closely to the data when using additive layers, compared to affine layers.

Wavelet Flow also experiences an interesting failure mode, where spatially distant dependencies are not captured well. This is most obvious when modeling human faces, and manifests as asymmetric eyes and inconsistent hair texture. Analysing the detail coefficients reveals that these inconsistencies mainly occur at higher frequencies. Low frequency features, such as hair colour, remain consistent across the image. We speculate that this could be due to a limited receptive field of the coupling layers, or the patch-wise training.

Chapter 5

Conclusion

In this thesis, we introduced Wavelet Flow, a multi-scale, conditional normalizing flow architecture based on wavelets. First, we showed that the distribution over a wavelet representation of a signal has a natural coarse-to-fine conditional decomposition, which has a number of benefits including efficient and highly parallel training. Second, we proposed a novel MCMC sampling algorithm for Wavelet Flow that does not require model restrictions. The algorithm is tractable in part due to the multi-scale decomposition. Third, our experiments demonstrated that the resulting model is up to $15\times$ faster to train, while still competitive with previous methods [12, 31] using a comparable number of parameters. The efficiency of the model enables the training of flow models for high resolution data which was impractical with previous approaches. Finally, we also showed how the multi-scale structure of the Wavelet Flow model can be used to extract consistent distributions of low resolution signals, as well as to perform super resolution.

Generative modeling is a core problem in machine learning. Normalizing flows can be

applied to generation, manipulation, and outlier detection of structured high dimensional data. Unlike other generative methods [61, 32, 15], approximate inference of latent variables, and disposable auxiliary components are not required, while still allowing efficient sampling. However, without improved computational efficiency, normalizing flows may never reach the performance and resolutions of these other methods.

5.1 Broader impact

This work contributes to the domain of generative image modeling, which constructs probabilistic models of images, allowing the generation, and manipulation of high resolution images. As this domain advances, these tasks become more accessible due to reduced labour, and skill barriers. On a positive side, this accessibility can empower artists and content creators with powerful tools for their craft. The computational costs of machine learning are of increasing concern as a potential future source of social and economic inequality. Reducing the need for specialized, large-scale hardware for training, as we have proposed, can lower the barrier to entry for individuals and groups which would otherwise be unable to afford large compute clusters.

As with all tools, generative models have no intentions of their own. Media platforms play an important role in the modern world, and as a result can be leveraged as a conduit for malice. Platforms are constantly engaged in a battle of attrition with disinformation. Our contributions primarily serve to reduce the costs of content generation at high resolutions. The reduced cost of creating malicious content can give malicious actors the edge. Even without intentionally malicious actors, reduced barriers to these tools could

lead to increased misuse. The nature of generative models encourages them to adopt biases from training data [5]. Without caution, generative models with harmful biases could be dangerously distributed at scale.

These concerns assume unopposed misuse of generative methods. Complementary research into the detection of generated content could mitigate these potential consequences. Much like how research into adversarial attacks continuously competes with research into robust recognition methods [16], generative methods can exist at an equilibrium with methods used to detect them. Additionally, research into methods of mitigating model bias [67] can directly reduce the misuse of generative models.

5.2 Future work

There remains several promising directions for future work. The choice of wavelet was not considered here; a Haar wavelet was used for simplicity. Numerous wavelets exist [42] and it is unclear how the choice of wavelet may impact performance. This thesis focused on images, but Wavelet Flow is directly applicable to other domains, *e.g.*, video, medical imaging, and audio. With images there remains work to explore applications of Wavelet Flow for visual tasks such as image restoration and super resolution. While we explored some architecture choices for the conditional flows, we expect that general improvements in performance will be found with other architectures, *e.g.*, [21, 40, 17, 25].

As noted in Chapter 4 uniform dequantization in a Wavelet Flow can interfere with certain desirable multi-scale properties. A general approach involves also learning a dequantizer in conjunction with the flow. Adapting dequantization for use in a Wavelet Flow

may rectify this and is likely to improve performance as it has with other normalizing flow models [21, 23]. Careful consideration must be made to ensure that the dequantization scheme does not interfere with the efficient parallelization properties of Wavelet Flow.

As shown empirically in Chapter 4 Section 4.4, the quality of samples is reduced from the model’s inability to capture long range dependencies at higher frequencies. Most of this capacity is hampered by the receptive field of the coupling layers, and the patch-wise training. The receptive field may be increased through increased flow steps, but this may be inefficient. A better approach may be to increase it through an additional conditioning network, instead of directly conditioning on the lower image. We explored a limited set of architectures for this, but improvements were minimal. Since the problem requires long range dependencies, convolutions may not be suitable: it may be worth exploring non-local operations, such as *transformers* [46], which have been successful with autoregressive models.

Bibliography

- [1] C. H. Anderson. A filter-subtract-decimate hierarchical pyramid signal analyzing and synthesizing technique, 1987. US Patent 4,718,104, Washington, DC. 17
- [2] Lynton Ardizzone, Carsten Lüth, Jakob Kruse, Carsten Rother, and Ullrich Köthe. Guided image generation with conditional invertible neural networks. *arXiv preprint arXiv:1907.02392*, 2019. 2
- [3] Ronald Newbold Bracewell and Ronald N Bracewell. *The Fourier transform and its applications*. McGraw-Hill New York, 1986. 17
- [4] Joan Bruna and Stéphane Mallat. Invariant scattering convolution networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 35(8):1872–1886, 2013. 17
- [5] Joy Buolamwini and Timnit Gebru. Gender shades: Intersectional accuracy disparities in commercial gender classification. In *Conference on fairness, accountability and transparency (FAccT)*, pages 77–91, 2018. 57
- [6] P. Burt and E. Adelson. The Laplacian pyramid as a compact image code. *IEEE Transactions on Communications*, 31(4):532–540, 1983. 17, 22

- [7] Peter J Burt. Fast filter transform for image processing. *Computer Vision, Graphics, and Image Processing (CVGIP)*, 16(1):20 – 51, 1981. 17
- [8] James L. Crowley and Alice C. Parker. A representation for shape based on peaks and ridges in the difference of low-pass transform. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 6(2):156–170, 1984. 17
- [9] Emily L Denton, Soumith Chintala, Arthur Szlam, and Rob Fergus. Deep generative image models using a laplacian pyramid of adversarial networks. In *Neural Information Processing Systems (NeurIPS)*, pages 1486–1494, 2015. 22
- [10] Jay Devore. *Modern mathematical statistics with applications*. Thomson Brooks/Cole, Belmont, CA, 2007. 9, 24
- [11] Laurent Dinh, David Krueger, and Yoshua Bengio. NICE: Non-linear Independent Components Estimation. In *Proceedings of the International Conference on Learning Representations (ICLR) Workshop*, 2015. 1, 12, 13
- [12] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density Estimation using Real NVP. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017. 1, 2, 12, 13, 15, 33, 34, 38, 39, 50, 55
- [13] Garoe Dorta, Sara Vicente, Lourdes Agapito, Neill DF Campbell, Simon Prince, and Ivor Simpson. Laplacian Pyramid of Conditional Variational Autoencoders. In *Proceedings of the Conference on Visual Media Production (CVMP)*, pages 1–9, 2017. 23

- [14] Philippe Esling, Naotake Masuda, Adrien Bardet, Romeo Despres, and Axel Chemla-Romeu-Santos. Flow synthesizer: Universal audio synthesizer control with normalizing flows. *Applied Sciences*, 10:302, 2019. 1
- [15] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative Adversarial Nets. In *Neural Information Processing Systems (NeurIPS)*, 2014. 8, 20, 56
- [16] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015. 57
- [17] Will Grathwohl, Ricky T Q Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. FFIORD: Free-form continuous dynamics for scalable reversible generative models. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019. 57
- [18] Ishaan Gulrajani, Kundan Kumar, Faruk Ahmed, Adrien Ali Taiga, Francesco Visin, David Vazquez, and Aaron Courville. Pixelvae: A latent variable model for natural images. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017. 22
- [19] Alfred Haar. Zur Theorie der orthogonalen Funktionensysteme. *Mathematische Annalen*, 69(3):331–371, 1910. 2, 20, 24
- [20] 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 (CVPR)*, 2016. 34
- [21] Jonathan Ho, Xi Chen, Aravind Srinivas, Yan Duan, and Pieter Abbeel. Flow++: Improving flow-based generative models with variational dequantization and architecture design. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2019. 1, 14, 15, 33, 41, 57, 58
- [22] Matthew D Hoffman and Andrew Gelman. The no-u-turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research (JMLR)*, 15(1):1593–1623, 2014. 27, 29, 30, 37
- [23] Emiel Hoogeboom, Taco S. Cohen, and Jakub M. Tomczak. Learning discrete distributions by dequantization. *arXiv preprint, arXiv:2001.11235*, 2020. 14, 15, 33, 41, 58
- [24] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2015. 14
- [25] Priyank Jaini, Kira A. Selby, and Yaoliang Yu. Sum-of-squares polynomial flow. In *Proceedings of the International Conference on Machine Learning (ICML)*, 5 2019. 33, 57
- [26] Animesh Karnewar, Oliver Wang, and Raghu Sessa Iyengar. MSG-GAN: Multi-scale gradient GAN for stable image synthesis. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, abs/1903.06048, 2020. 22

- [27] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of GANs for improved quality, stability, and variation. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018. 22, 38, 39, 40, 44
- [28] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4401–4410, 2019. 38, 39, 40, 44
- [29] Sungwon Kim, Sang gil Lee, Jongyoon Song, Jaehyeon Kim, and Sungroh Yoon. FloWaveNet: A Generative Flow for Raw Audio. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2018. 1
- [30] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2014. 35
- [31] Diederik P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Neural Information Processing Systems (NeurIPS)*, pages 10215–10224, 2018. 1, 3, 13, 14, 15, 26, 27, 33, 34, 35, 38, 39, 40, 42, 45, 50, 51, 55, 70
- [32] Diederik P. Kingma and Max Welling. An Introduction to Variational Autoencoders. *arXiv preprint, arXiv:1906.02691*, 2019. 7, 22, 56
- [33] Ivan Kobyzev, Simon Prince, and Marcus Brubaker. Normalizing flows: An intro-

- duction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2020. 1, 9
- [34] Jan J. Koenderink. The structure of images. *Biological Cybernetics*, 50(5):363–370, 1984. 17
- [35] Manoj Kumar, Mohammad Babaeizadeh, Dumitru Erhan, Chelsea Finn, Sergey Levine, Laurent Dinh, and Durk Kingma. VideoFlow: A Flow-Based Generative Model for Video. In *Workshop on Invertible Neural Nets and Normalizing Flows, International Conference on Machine Learning (ICML)*, 2019. 1
- [36] Junpeng Lao, Christopher Suter, Ian Langmore, Cyril Chimisov, Ashish Saxena, Pavel Sountsov, Dave Moore, Rif A. Saurous, Matthew D. Hoffman, and Joshua V. Dillon. tfp.mcmc: Modern Markov chain Monte Carlo tools built for modern hardware. *arXiv preprint, arXiv:2002.01184*, 2020. 27, 30
- [37] Tony Lindeberg. Scale-space for discrete signals. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 12(3):234–254, 1990. 17
- [38] Tony Lindeberg. Scale-space theory: A basic tool for analyzing structures at different scales. *Journal of Applied Statistics*, 21(1-2):225–270, 1994. 17
- [39] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2015. 38
- [40] Xuezhe Ma, Xiang Kong, Shanghang Zhang, and Eduard Hovy. MaCow: Masked

- convolutional generative flow. In *Neural Information Processing Systems (NeurIPS)*, pages 5891–5900, 2019. 3, 26, 27, 33, 45, 50, 57
- [41] Stéphane Mallat. A theory for multiresolution signal decomposition: The wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 11(7):674–693, 1989. 17, 19
- [42] Stephane Mallat. *A wavelet tour of signal processing: The Sparse Way*. Elsevier, 3rd edition, 2009. 2, 19, 20, 24, 57
- [43] David Marr. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. Henry Holt and Co., Inc., USA, 1982. 17
- [44] Jacob Menick and Nal Kalchbrenner. Generating high fidelity images with subscale pixel networks and multidimensional upscaling. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019. 6
- [45] George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing Flows for Probabilistic Modeling and Inference. *arXiv preprint, arXiv:1912.02762*, 2019. 1, 9
- [46] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Łukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. *Proceedings of the International Conference on Machine Learning (ICML)*, 2018. 6, 58
- [47] Ryan Prenger, Rafael Valle, and Bryan Catanzaro. WaveGlow: A flow-based generative network for speech synthesis. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2019. 1

- [48] Ali Razavi, Aäron van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with VQ-VAE-2. In *Neural Information Processing Systems (NeurIPS)*, pages 14837–14847, 2019. 22
- [49] Scott E. Reed, Aäron van den Oord, Nal Kalchbrenner, Sergio Gomez Colmenarejo, Ziyu Wang, Yutian Chen, Dan Belov, and Nando de Freitas. Parallel multiscale autoregressive density estimation. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 2912–2921, 2017. 23
- [50] Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. Efficient content-based sparse attention with routing transformers. *arXiv preprint arXiv:2003.05997*, 2020. 6
- [51] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. 38, 39, 40, 44
- [52] Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P. Kingma. PixelCNN++: Improving the PixelCNN with Discretized Logistic Mixture Likelihood and Other Modifications. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017. 6, 23
- [53] Tamar Rott Shaham, Tali Dekel, and Tomer Michaeli. SinGAN: Learning a generative model from a single natural image. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 4569–4579, 2019. 22

- [54] Eero P Simoncelli and William T Freeman. The steerable pyramid: A flexible architecture for multi-scale derivative computation. In *Proceedings of the IEEE International Conference on Image Processing (ICIP)*, volume 3, pages 444–447, 1995. 16, 17, 19
- [55] David Stirzaker. *Elementary probability*. Cambridge University Press, Cambridge, UK New York, 2003. 9, 24
- [56] Gilbert Strang. *Linear algebra and its applications*. Harcourt, Brace, Jovanovich, Publishers, San Diego, 1988. 14
- [57] Lucas Theis and Matthias Bethge. Generative image modeling using spatial lstms. In *Neural Information Processing Systems (NeurIPS)*, pages 1927–1935, 2015. 14
- [58] Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. *arXiv preprint arXiv:1511.01844*, 2015. 14
- [59] Benigno Uria, Iain Murray, and Hugo Larochelle. Rnade: The real-valued neural autoregressive density-estimator. In *Neural Information Processing Systems (NeurIPS)*, pages 2175–2183, 2013. 14
- [60] Aaron Van den Oord and Benjamin Schrauwen. Factoring variations in natural images with deep gaussian mixture models. In *Neural Information Processing Systems (NeurIPS)*, pages 3518–3526, 2014. 14
- [61] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel Recurrent Neural Networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2016. 6, 22, 23, 56

- [62] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016. 6
- [63] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-resolution image synthesis and semantic manipulation with conditional GANs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8798–8807, 2018. 22
- [64] Christina Winkler, Daniel Worrall, Emiel Hoogeboom, and Max Welling. Learning likelihoods with conditional normalizing flows. *arXiv preprint arXiv:1912.00042*, 2019. 34
- [65] Andrew P. Witkin. Scale-space filtering. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1019–1022, 1983. 17
- [66] Fisher Yu, Ari Seff, Yinda Zhang, Shuran Song, Thomas Funkhouser, and Jianxiong Xiao. LSUN: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*, 2015. 38, 39, 40, 43
- [67] Brian Hu Zhang, Blake Lemoine, and Margaret Mitchell. Mitigating unwanted biases with adversarial learning. In *Proceedings of the Conference on AI, Ethics, and Society*, pages 335–340, 2018. 57
- [68] Han Zhang, Tao Xu, and Hongsheng Li. StackGAN: Text to photo-realistic image synthesis with stacked generative adversarial networks. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 5908–5916, 2017. 22

- [69] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris N. Metaxas. StackGAN++: Realistic image synthesis with stacked generative adversarial networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 41(8):1947–1962, 2019. 22

Appendix

A.1 Annealed Gaussian distribution

For completeness, here we derive the annealed form of a Gaussian distribution and show the relationship between the parameters γ used in Section 3.2, and the *temperature* parameter T used by Glow [31]. The annealed probability density function of a distribution $p(x)$ is $\propto p(x)^\gamma$, where values of $\gamma > 1$ correspond to distributions tighter than the unannealed distribution. In the case of a univariate Gaussian, annealing is equivalent to reducing the

variance of the Gaussian:

$$\mathcal{N}(x; \mu, \sigma^2)^\gamma = \left[\frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{1}{2} \left(\frac{x-\mu}{\sigma} \right)^2} \right]^\gamma \quad (\text{A.1})$$

$$= \frac{1}{(\sigma \sqrt{2\pi})^\gamma} e^{-\frac{1}{2} \gamma \left(\frac{x-\mu}{\sigma} \right)^2} \quad (\text{A.2})$$

$$= \frac{1}{\left(\sigma \frac{\sqrt{\gamma}}{\sqrt{\gamma}} \sqrt{2\pi} \right)^\gamma} e^{-\frac{1}{2} \left(\sqrt{\gamma} \frac{x-\mu}{\sigma} \right)^2} \quad (\text{A.3})$$

$$= \frac{1}{\sqrt{\gamma}^\gamma \left(\sigma \frac{1}{\sqrt{\gamma}} \sqrt{2\pi} \right)^\gamma} e^{-\frac{1}{2} \left(\frac{x-\mu}{\sigma \frac{1}{\sqrt{\gamma}}} \right)^2} \quad (\text{A.4})$$

$$= \frac{1}{\sqrt{\gamma}^\gamma \left(\sigma \frac{1}{\sqrt{\gamma}} \sqrt{2\pi} \right)^{\gamma-1}} \frac{1}{\sigma \frac{1}{\sqrt{\gamma}} \sqrt{2\pi}} e^{-\frac{1}{2} \left(\frac{x-\mu}{\sigma \frac{1}{\sqrt{\gamma}}} \right)^2} \quad (\text{A.5})$$

$$= \frac{1}{\sqrt{\gamma}^\gamma \left(\sigma \frac{1}{\sqrt{\gamma}} \sqrt{2\pi} \right)^{\gamma-1}} \mathcal{N} \left(x; \mu, \sigma^2 \frac{1}{\gamma} \right) \quad (\text{A.6})$$

$$\propto \mathcal{N} \left(x; \mu, \sigma^2 \frac{1}{\gamma} \right). \quad (\text{A.7})$$

In the case of annealing a Gaussian with unit variance, T is the standard deviation of the annealed Gaussian.

A.2 Experimental setup

As stated in Chapter 4 of the thesis, hyper-parameters are set differently for each dataset to keep the number of parameters below or within the range of the compared methods. These hyper-parameters are shown in Table A.1.

Model/ Hyper-params.	Resolution										
	1	2	4	8	16	32	64	128	256	512	1024
ImageNet 32											
N Steps	8	8	16	16	16	16	-	-	-	-	-
Conv. channels	128	128	128	128	128	256	-	-	-	-	-
Patch size	1	2	4	8	16	16	-	-	-	-	-
Batch size	64	64	64	64	64	64	-	-	-	-	-
N Parameters	4m	4m	8m	8m	8m	32m	-	-	-	-	-
ImageNet 64											
N Steps	8	8	16	16	16	16	16	-	-	-	-
Conv. channels	128	128	128	128	128	256	256	-	-	-	-
Patch size	1	2	4	8	16	16	32	-	-	-	-
Batch size	64	64	64	64	64	64	48	-	-	-	-
N Parameters	4m	4m	8m	8m	8m	32m	32m	-	-	-	-
LSUN (bedroom/tower /church) 64											
N Steps	8	8	16	16	16	16	16	-	-	-	-
Conv. channels	16	64	64	64	64	128	256	-	-	-	-
Patch size	1	2	4	8	16	32	64	-	-	-	-
Batch size	64	64	64	64	64	64	64	-	-	-	-
N Parameters	68k	1m	2m	2m	2m	8m	32m	-	-	-	-
CelebA-HQ/ FFHQ 1024											
N Steps	8	8	16	16	16	16	16	16	16	16	16
Conv. channels	64	64	64	128	128	128	128	128	128	128	128
Patch size	1	2	4	8	16	32	32	64	64	64	64
Batch size	64	64	64	64	64	64	64	64	64	64	64
N Parameters	1m	1m	2m	8m	8m	8m	8m	8m	8m	8m	8m

Table A.1: Hyper-parameters used with Wavelet Flow per level, and across the evaluation datasets.

A.3 Additional samples

Additional samples without annealing are shown in Figures A.1, A.2, A.3, A.4, A.5, A.6, and A.7. Additional samples with annealing using the previously described MCMC method with temperature $T = 0.97$, are shown in Figures A.8, A.9, A.10, A.11, A.12, A.13, and A.14. Additional $8\times$ super resolution samples from CelebA-HQ, and FFHQ, are shown in Figures A.15, and A.16.



Figure A.1: Samples from ImageNet (32×32) without annealing.



Figure A.2: Samples from ImageNet (64×64) without annealing.



Figure A.3: Samples from LSUN bedroom (64×64) without annealing.

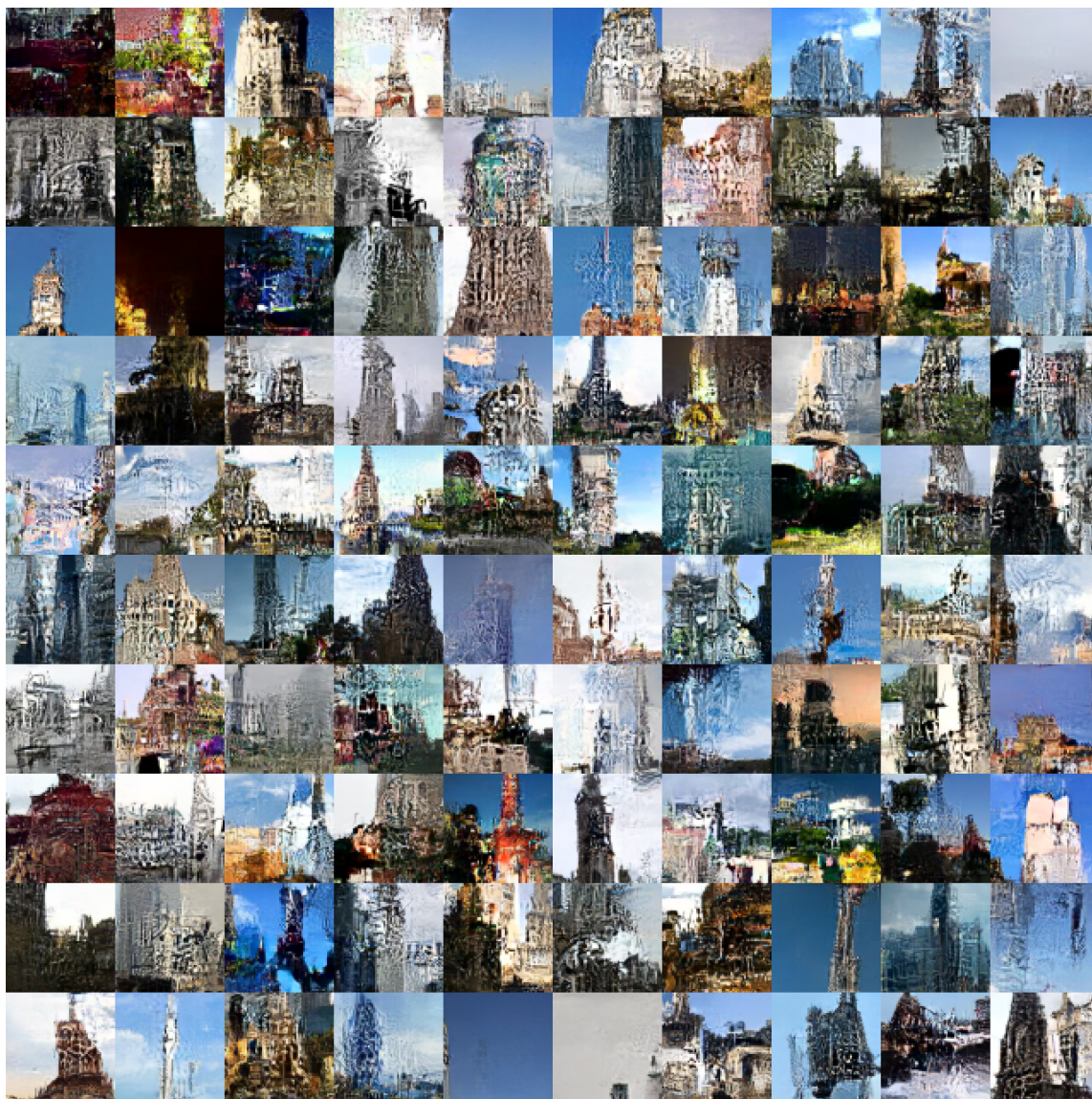


Figure A.4: Samples from LSUN tower (64×64) without annealing.



Figure A.5: Samples from LSUN church (64×64) without annealing.



Figure A.6: Samples from CelebA-HQ (256×256) without annealing.



Figure A.7: Samples from FFHQ (256×256) without annealing.



Figure A.8: Annealed samples from ImageNet (32×32) with $T = 0.97$.



Figure A.9: Annealed samples from ImageNet (64×64) with $T = 0.97$.



Figure A.10: Annealed samples from LSUN bedroom (64×64) with $T = 0.97$.

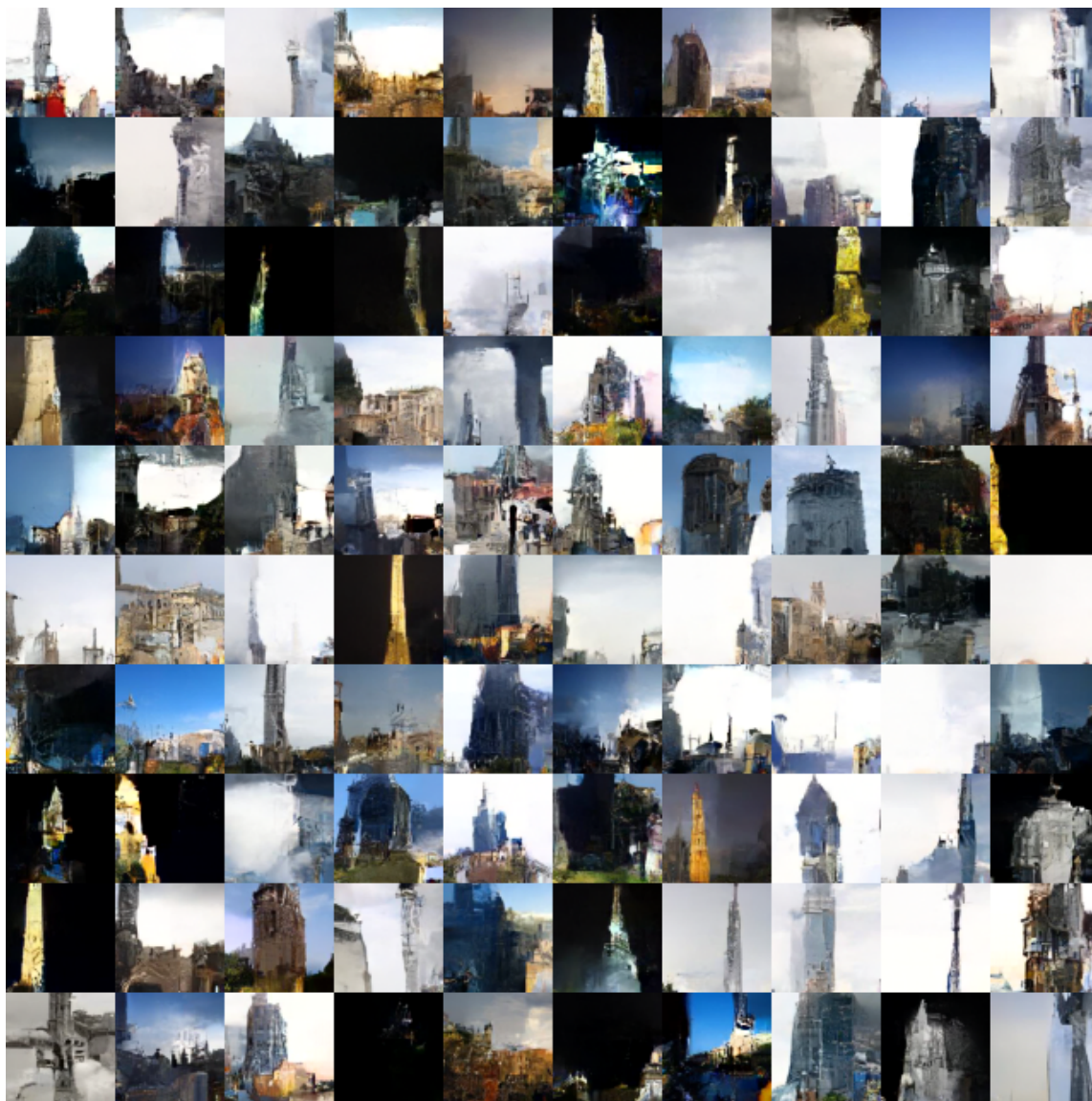


Figure A.11: Annealed samples from LSUN tower (64×64) with $T = 0.97$.



Figure A.12: Annealed samples from LSUN church (64×64) with $T = 0.97$.



Figure A.13: Annealed samples from CelebA-HQ (256×256) with $T = 0.97$.



Figure A.14: Annealed samples from FFHQ (256×256) with $T = 0.97$.



Figure A.15: $8\times$ super resolution on CelebA-HQ. The smaller images (top) are the original images at 128×128 , while the larger images (bottom) are the annealed super resolution results at 1024×1024 , with $T = 0.97$.



Figure A.16: $8\times$ super resolution on FFHQ. The smaller images (top) are the original images at 128×128 , while the larger images (bottom) are the annealed super resolution results at 1024×1024 , with $T = 0.97$.