Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

Heikki Timonen

Conditional Normalizing Flows for Imaging Inverse Problems

Master's Thesis
Espoo, April 27, 2020

Supervisor:	Prof. Jaakko Lehtinen
Advisor:	Prof. Jaakko Lehtinen

Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

ABSTRACT OF MASTER'S THESIS

Author:	Heikki Timonen		
Title:	Conditional Normalizing Flows for Imaging Inverse Problems		
Date:	April 27, 2020	**Pages:**	111
Major:	Machine Learning, Data Science and Artificial Intelligence	**Code:**	SCI3044
Supervisor:	Prof. Jaakko Lehtinen		
Advisor:	Prof. Jaakko Lehtinen		

Learning-based methods have provided powerful tools for solving classification and regression -related problems yielding far superior results to classical hand-crafted rule-based models. These models have proven to be efficient in multiple domains in many different fields. However, many common problems are inherently ill-posed and lack a unique answer hence requiring a regularization pass or alternatively a probabilistic framework for successful modeling. While many different families of models capable of learning distributions given samples exist, they commonly resort to approximations or surrogate training objectives.

In this thesis we solve image-related inverse problems with a family of probabilistic models known as conditional normalizing flows. A normalizing flow consists of repeated applications of invertible transformations on a simple prior distribution rendering it into a more complex distribution with direct and tractable probability density evaluation and efficient sampling. We show that a conditional normalizing flow is able to provide plausible, high-quality samples with visible benign variance from a conditional distribution in image superresolution, denoising and colorization tasks. We quantify the success of the model as well as its shortcomings and inspect how it internally addresses the conversion of white noise into a realistic image.

Keywords:	machine learning, generative models, conditional density estimation, inverse problems
Language:	English

Aalto-yliopisto
Perustieteiden korkeakoulu
Tieto-, tietoliikenne- ja informaatiotekniikan maisteriohjelma

DIPLOMITYÖN TIIVISTELMÄ

Tekijä: Heikki Timonen

Työn nimi:
Ehdolliset normalisoivat virtaukset kuvien käänteisongelmissa

Päiväys:	27.04.2020	**Sivumäärä:**	111
Pääaine:	Machine Learning, Data Science and Artificial Intelligence	**Koodi:**	SCI3044
Valvoja:	Prof. Jaakko Lehtinen		
Ohjaaja:	Prof. Jaakko Lehtinen		

Havainnoista oppimiseen optimoinnin avulla perustuvat mallit kykenevät ratkaisemaan monia ongelmia huomattavasti tehokkaammin, kuin klassiset staattisiin päätössääntöihin perustuvat mallit. Perinteisesti mallit antavat yleensä kuitenkin vain yhden vastauksen, vaikka useilla ongelmilla saattaa olla monta keskenään yhtä hyväksyttävää vastausta. Tämän takia on tarkoituksenmukaista mallintaa todennäköisyysjakaumaa kaikista mahdollisista vastauksista yksittäisen vastauksen sijaan.

Tässä diplomityössä tutkitaan normalisoivien virtausten malliluokan soveltamista digitaalisiin kuviin liittyviin käänteisongelmiin. Normalisoiva virtaus muuntaa yksinkertaisen todennäköisyysjakauman neuroverkoilla parametrosoiduilla kääntyvillä funktioilla monimutkaisemmaksi jakaumaksi, siten että havaintojen uskottavuudesta saadaan kuitenkin tarkka numeerinen arvo. Normalisoivat virtaukset mahdollistavat myös tehokkaan näytteiden ottamisen niiden mallintamasta monimutkaisesta todennäköisyysjakaumasta. Työssä määritetään, kuinka hyvin virtausmallit onnistuvat tehtävässään ja kuinka ne muodostavat uskottavia kuvia kohinasta. Työssä todetaan, että ehdollisten normalisoivien virtausten avulla voidaan tuottaa korkealaatuisia näytteitä useissa kuviin liittyvissä käänteisongelmissa.

Asiasanat: koneoppiminen, generatiiviset mallit, ehdollinen todennäköisyystiheyden arviointi, käänteisongelmat

Kieli: Englanti

Preface

I had the pleasure of meeting my thesis supervisor and advisor, professor Jaakko Lehtinen in the aftermath of the 2018 edition of the award-winning Programming Parallel Computers course that I attended that spring. He gently suggested me to take his introductory class to computer graphics the coming autumn, which I eventually ended up doing. Little did I know how far-reaching effects that little meeting would have in the future.

I would like to thank professor Lehtinen for his time and support, not only in the context of this project, but also in the broader scope of doing research in general. Tackling the unknown is difficult as is, so why make it even more so by neglecting simple practicalities like properly labeling your results and making your research code modular. I am also grateful for him letting me have such freedom on choosing what I want to focus on and letting me work so autonomously and independently.

I would like to extend special thanks to Dr. Miika Aittala for his many insights that multiple times lead into interesting results and me having a stronger understanding of the problem. I would also like to thank M.Sc. (Tech.) Tuomas Kynkäänniemi for his wonderful codebase which greatly facilitates the use of the computational cluster. I acknowledge the computational resources provided by the Aalto Science-IT project.

Lastly I would like to thank my friends, collegues at the lab, and family. This thesis was partially written during the corona-craze of spring 2020 rendering their support invaluable — even more so than usual.

Espoo, April 27, 2020

Heikki Timonen

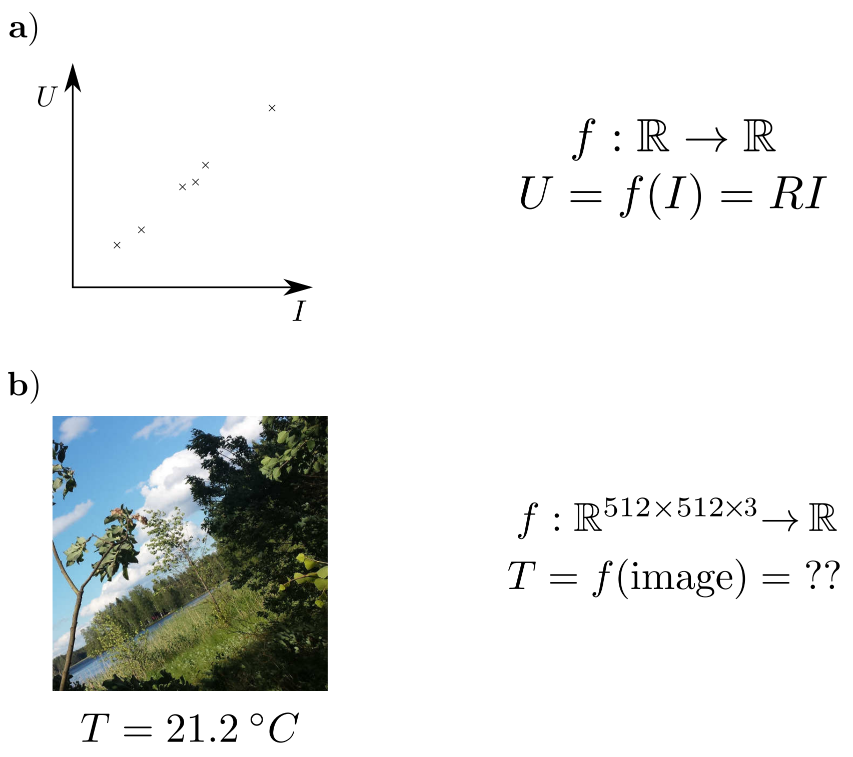
Contents

Chapter 1

Introduction

In very general terms, machine learning is the process of fitting a function to best match some observed or measured data. It is in principle not any different from, say, the familiar task of fitting a line on the relationship between current and voltage in an electrical circuit in order to estimate the resistance of a component. To find a good fit, two constraints must be satisfied: the observations must be explainable by the choice of the model (here, a straight line with an unknown slope and intercept) and some measure of error must be minimized (the line should interpolate the observations well). Not every problem is solved using a linear model in low-dimensional space. Indeed, often the dimensionality of the data can be in the order of millions (e.g. images or videos, see Figure 1.1), the relationships between the variables can be highly non-linear and no closed-form model derived using physical arguments and principles is available. It is reasonable to think a system (e.g. the electrical circuit) as a black box. We can make individual measurements and try to disassemble the box to reason about the system within (to find clues about $U = RI$). The system can, however, be so complex that our reasoning is of little use and measurements are all we have.

Even with a model with perfect predictive and expressive power, some problems remain unsolvable to a degree. Many problems naturally have only a single solution. The classification of an object can be either a teapot or a bunny but it cannot be both at the same time. Some problems, however, can have multiple solutions. If the task is to colorize a black-and-white image, plausible colors for an apple in the image can include red and green hues, but probably not blue colors. One can try to solve an ill-posed, multi-valued problem like the colorization via finding a single answer that minimizes some averaged error metric like the mean or mode. However, even though the acquired answer is the best with respect to some error measure, it might be uninteresting or somehow perceptually low-quality. The mean color of an



$$f : \mathbb{R} \to \mathbb{R}$$
$$U = f(I) = RI$$

$$f : \mathbb{R}^{512\times512\times3} \to \mathbb{R}$$
$$T = f(\text{image}) = ??$$

Figure 1.1: Function approximation. **a**) Finding the best fit line for the relationship between current and voltage is not difficult. Physical principles may help to find a closed-form formula. **b**) Finding a function that estimates the air temperature from an image requires a mapping from high-dimensional space to a single number. The system is much more complex and physical reasoning is of little use.

apple is probably some shade of brown but we do not want our model to only yield rotten apples! Another, more general means for solving an *inverse problem* such as the one defined above involves *probabilistic* modeling, that comprises of approximating the distribution of plausible solutions. Individual estimates minimizing various error metrics can still be obtained by finding, say the mean of the distribution or the most likely answer given the distribution.

Modeling probability distributions involves very similar requisites to those of ordinary function approximation. It is crucial to choose the right type of distribution for the right problem. Too simple a distribution may completely miss the intricacies of the data. One also needs to find the optimal parameters for the distribution, very much like finding the best slope in the resistance measurement. Fitting a distribution qualitatively, however, is much more difficult than finding the line of best fit and a quantitative measure for the quality of the fit is essential. The stochastic nature of the data poses additional challenges. Maybe the measurements are an extremely poor representation of the true underlying distribution. What if we have only ever seen rotten apples?

Machine learning provides tools that have empirically been found to be extremely versatile function approximators (some are even so-called universal approximators). In particular, this suggests that the models are able to present functions much more complicated than linear models. Solving the function approximation problem computationally also allows one to specify a much more well-defined measure of error than how "good" a line qualitatively looks to the eye with respect to the points of data. The solution of the machine learning model is found by minimizing the assigned error measure given the the observations, rendering the function approximation problem into an optimization problem with respect to the parameters of the model. In simple cases the optimization can be carried out in closed form by analytically finding the root of the derivative of the loss-function. However, one often needs to apply *iterative* optimization methods such as gradient descend to find even a local minimum of the loss function.

Versatile function approximation is also extremely useful in probabilistic modeling. It provides a remedy for the difficult modeling choice of deciding on the parametric form of the employed distribution. Models are often trained by maximizing the *likelihood* of the training data (or rather, minimizing the negative logarithmic likelihood) with gradient descent with respect to the parameters of the machine learning model. Some model types evaluate likelihoods explicitly and without approximations, while others resort to optimizing lower bounds, or choose only to work with likelihoods implicitly.

In this thesis we study the application of a family of models called *normalizing flows* (NFs) on several inverse imaging problems. Even though the roots of normalizing flows originate in the 1990's, they have only recently been rediscovered and found to be competitive in the field of generative modeling. A normalizing flow consists of a composition of *invertible* functions with tractable Jacobian determinants which can be efficiently computed. The flow model transforms a complex target probability distribution into a simple (usually Gaussian) distribution, or vice-versa a simple distribution into a complex target distribution, due to invertibility. Instead of minimizing some error metric, the training target of a NF-model is usually the maximization of the likelihood of the data under the parameters of the model. NF-models have in the past been applied to modeling the distributions of various toy-datasets in the conditional density estimation settings. Only very recently has there been reports of work on more complex, real-life datasets. In the unconditional setting, normalizing flows have been demonstrated to be able to generate photo-realistic human faces in high resolutions.

We employ a flow-model conditioned on images corrupted in various ways and show that our model is able to recover plausible, high-quality samples of the image with the corruption removed. Furthermore, we show that the samples have visible variation, yet remain faithful to the underlying corrupted image. Results of a flow model conditioned with grayscale-images tasked with colorization are given in Figure 1.2. We study extensively how the information conditioning should be introduced into the model. Based on previous

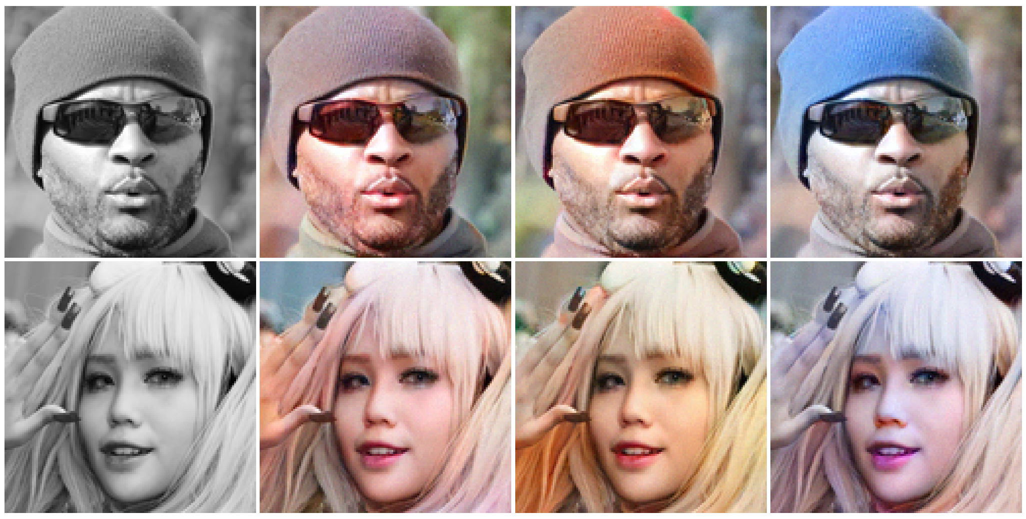


Figure 1.2: Colorization with conditional normalizing flows. The leftmost column is the network input and the other three correspond to samples from the conditional distribution.

work, we know that flow models by themselves alone are not particularly expressive compared to other methods. Hence it is crucial to find configuration that minimizes computational stress on the flow. We thoroughly examine the internal operation of the proposed conditional normalizing flow in order to understand how it converts image data into essentially white noise. In particular, we study which parts or layers of the flow are important for successful image restoration and whether sampling should be truncated. We aim to find differences between the operation between conditional and unconditional normalizing flows. We also quantify how well the flow succeeds in its task of decorrelating the elements of an image vector. We present examples on how to drastically affect the internal operation of the model and how to render the model more lightweight by reducing the number of required parameters.

The work in this thesis builds upon normalizing flows with coupling layers [16] and invertible 1×1 convolutions [38]. It is the most closely related to ideas of Ardizzone *et al.* [2, 3] who study the applicability of conditional normalizing flows to inverse problems. Our method slightly deviates from theirs by not employing a pretrained VGG-network [61] as a part of the flow. We also apply the flow model to other inverse problems than colorization. Closely related, parallel work concerning superresolution with normalizing flows [73] appeared during the writing of this thesis.

The structure of this thesis is as follows. We begin in Chapter 2 by taking a brief look at classical density estimation and probabilistic modeling. We also give a brief introduction to deep neural networks. We then proceed to describing some of the most common deep generative models including, for example, variational autoencoders and generative adversarial networks. We pay special attention to the family of normalizing flows, carefully depicting their features and various implementations, referring to a comprehensive collection of past work. In Chapter 3, we focus on *conditional* generative models and their connection and applicability to inverse problems. We present our methods and implementation details of the employed flow model and its variations in Chapter 4. We aggregate the results of the characterization of the model as well as its performance metrics in Chapter 5. Finally, we give concluding remarks and discuss future work in Chapter 6.

Chapter 2

Generative Modeling

Machine learning has successfully been applied to various problems with a single, deterministic solution. However, many problems are not one-to-one and hence instead of having a single solution, the answer to the problem is rather a probability distribution over a potentially infinite number of solutions. The task of an appropriate model is to produce samples from the distribution and potentially even estimate the underlying probability density function (PDF). More specifically, solving the problem requires *conditional* density estimation, since the solution is a distribution *given* some initial information. We will first introduce ordinary density estimation and afterwards move to the conditional variant in a later chapter.

In this section we introduce some of the most common methods for probabilistic modeling. We roughly divide the methods into classical models and deep learning-based models. The division is very similar to the example of function fitting in the previous chapter. While the modeling work with nearly all the models featured in this chapter includes optimization, classical probabilistic modeling often requires stronger assumptions and more expert knowledge about the system that is being modeled. In traditional function approximation this equals to the application of, say, physical reasoning about the system. Deep learning-based models assume very little about the function and rather employ measurements to approximate it. Deep probabilistic modeling operates exactly in the same way, merely replacing general functions with with more constrained probability density functions.

Generative modeling refers to the ability to generate novel observations $\boldsymbol{x}$ using the approximate probability distribution potentially conditioned with some label $\boldsymbol{y}$. It is in contrast to *discriminative modeling* where the aim is to find the most plausible label $\boldsymbol{y}$ given an observation $\boldsymbol{x}$.

2.1 Classical Probabilistic Modeling

A proper probability density $p(\boldsymbol{x})$ for a D-dimensional real-valued vector $\boldsymbol{x}$ has the following properties

$$p(\boldsymbol{x}) \geq 0 \quad \forall \boldsymbol{x} \in \mathbb{R}^D, \tag{2.1}$$

$$\int_X p(\boldsymbol{x})\mathrm{d}\boldsymbol{x} = \Pr(\boldsymbol{x}' \in X), \tag{2.2}$$

which also implies that the integral over the entire space must be unity due to the definition of probability. Having access to the PDF is not equivalent to being able to acquire samples from the distribution. Indeed, samples (measurements) are usually the only thing available from a distribution and the modeling task is to find the density function that has the strongest support of the measurements. This is known as *density estimation* [7].

Like with standard function fitting, essentially all natural systems are so complex that we simply cannot have all necessary information about their operation. This is illustrated in Figure 2.1. The system can be so overly complex, that we cannot control all the possible variables and hence probabilistic models are required to account for our lack of knowledge and control. The systems can again be seen as black boxes that allow individual measurements from the outside, but do no let us open the box and study the inside directly.

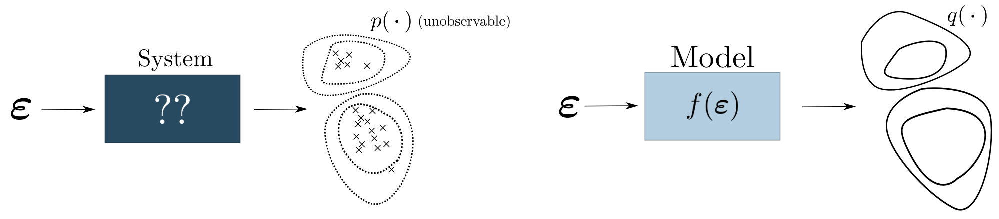
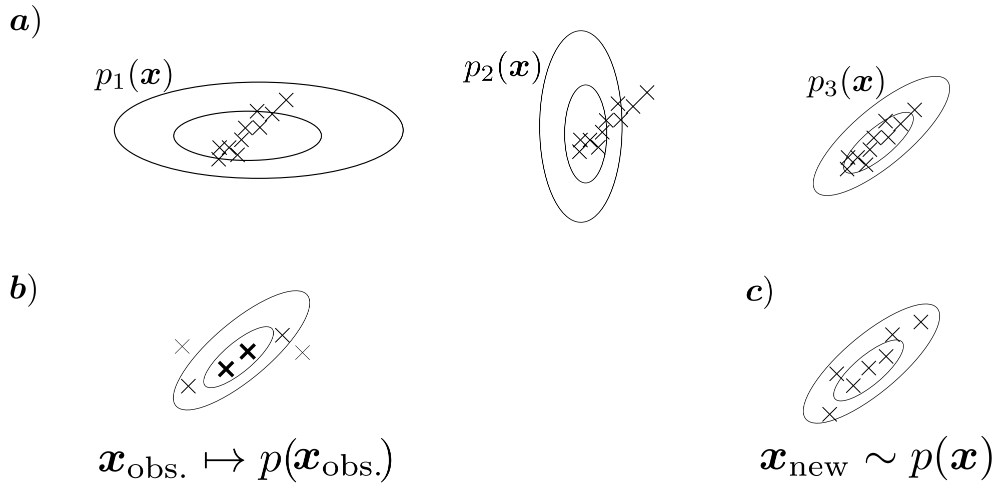

$p(\cdot)$ (unobservable)
System
$\boldsymbol{\varepsilon}$
??
Model
$\boldsymbol{\varepsilon}$
$f(\boldsymbol{\varepsilon})$
$q(\cdot)$

Figure 2.1: Distribution approximation. The variable $\boldsymbol{\varepsilon}$ is a random vector and denotes the lack of knowledge and control of the variables of the system. The contents of the system (left) are unknown and only samples of the true distribution p are available. The modeling task is to find an approximation q (right) that is as close to p as possible with some model f.

Properties of a good model A good density estimate or approximation $q(\boldsymbol{x})$ of an underlying true probability density $p(\boldsymbol{x})$ maximizes the (expected) *likelihood* of the observations. That is, the approximation q assigns high probabilities (or probability densities in continuous models) to the more common measurements. The modeling task involves choosing a family of distributions which hopefully contains the true $p(\boldsymbol{x})$. Using just any family is not enough, as it is reasonable to expect certain properties of the modeling distribution. We could, for example, merely assign tiny boxes of large probability density to the neighborhoods of all our data points (remembering proper normalization) to achieve large likelihood for the observations. However, it is almost certain that this setup fails to describe the true distribution since it is essentially discontinuous. A properly chosen model allows us to assess new measurements quantitatively. It allows us to assign a number describing the likelihood of the measurement under the chosen model by evaluating the PDF. Not all models, however, yield directly a likelihood as some only provide a lower bound and some do not even explicitly work with likelihoods. A probabilistic model can often also be sampled to find novel, unseen data points. Sampling usually happens by generating random numbers from a simple distribution (the uniform distribution) and applying a function to transform the uniform distribution into a more complex distribution. These elements of inference, density evaluation and sampling are illustrated in Figure 2.2.



a)
$p_1(\boldsymbol{x})$
$p_2(\boldsymbol{x})$
$p_3(\boldsymbol{x})$
b)
c)
$\boldsymbol{x}_{\text{obs.}} \mapsto p(\boldsymbol{x}_{\text{obs.}})$
$\boldsymbol{x}_{\text{new}} \sim p(\boldsymbol{x})$

Figure 2.2: Elements of probabilistic modeling. ***a***) Density estimation, inference. Choosing the PDF $p(\boldsymbol{x})$ that best describes the data (here, probably p_3). ***b***) Density evaluation. Measurement of likelihood of new observations under the model. Can be, for example, used to find outliers in the data. May not be possible to do exactly in all model types. ***c***) Sampling or generating new observations.

Assuming a family of distributions is parametrized by parameters $\boldsymbol{\theta}$, the best, likelihood-maximizing distribution if found by optimization. The optimization of likelihood is desirable in another sense as well. Namely, maximizing the data likelihood also minimizes the so-called Kullback-Leibler divergence between the approximation and the true model

$$\begin{aligned} D_{\mathrm{KL}}(p(\boldsymbol{x})||q(\boldsymbol{x})) &\equiv \int_X p(\boldsymbol{x}) \log\left(\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\right) \mathrm{d}\boldsymbol{x} \\ &= \int_X p(\boldsymbol{x}) \log\left(p(\boldsymbol{x})\right) \mathrm{d}\boldsymbol{x} - \int_X p(\boldsymbol{x}) \log\left(q(\boldsymbol{x})\right) \mathrm{d}\boldsymbol{x} \\ &= -H(p) - \mathbb{E}_{p(\boldsymbol{x})}\left[\log\left(q(\boldsymbol{x})\right)\right], \end{aligned} \tag{2.3}$$

where H is the entropy of p. The likelihood-maximizing distribution q^* minimizes the second term of Eq. 2.3 and hence minimizes the entire expression since the entropy H does not depend on q. The non-negative divergence vanishes only when $q(\boldsymbol{x}) = p(\boldsymbol{x}) \quad \forall \boldsymbol{x} \in X$. The duality between likelihood maximization and divergence minimization is demonstrated in Figure 2.3. We now introduce some of the most common classical models for density estimation.

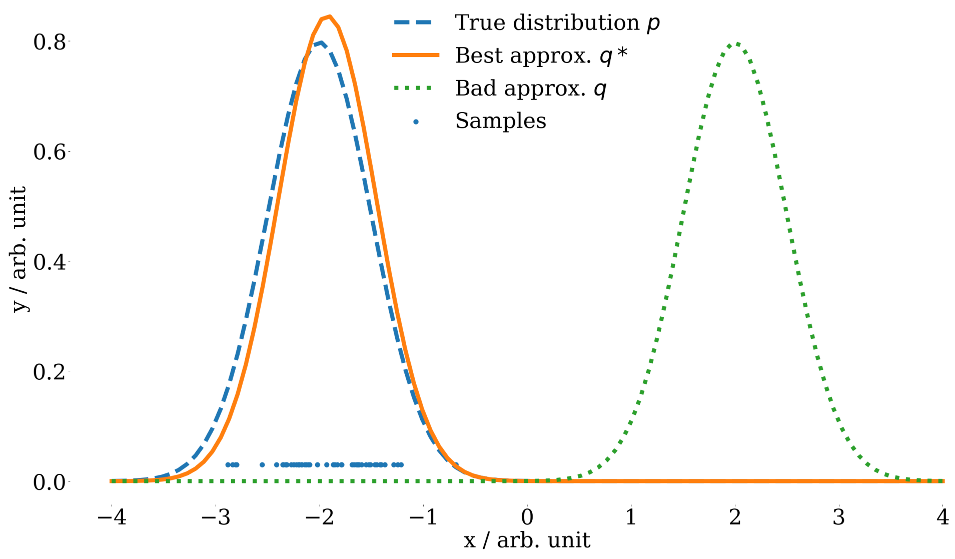


Figure 2.3: Likelihood and divergence. Samples from and the PDF of the true distribution p (blue dots and dashed line, respectively). The best approximation q^* from the Gaussian family given the data (orange solid line) assigns high likelihood to the samples. The KL-divergence is minimized. A bad approximation q (green dotted line) assigns vanishing likelihood to the samples and hence $D_{\mathrm{KL}}(p||q)$ is large.

Parametric models Simple parametric models provide a starting point for density estimation. Choosing, for example, the Gaussian family and finding the data-likelihood maximizing mean-vector and covariance matrix analytically yields the best density function q^* from the Gaussian family, like in Figure 2.3. However, for a more complex true distribution p, the Gaussian family might be completely inadequate and lack the required representational power. We visualize this in Fig. 2.4 with 2-dimensional toy-data. The chosen multivariate Gaussian fails to model the target distribution, completely missing the multimodality. The problem can remedied by expanding the search to more complex distributions, such as mixture models.

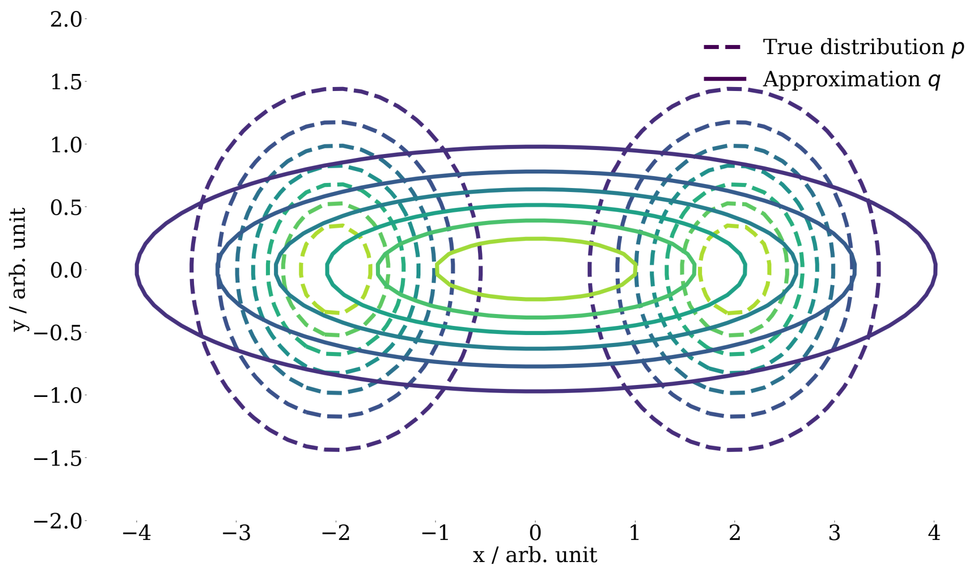


Figure 2.4: Failure case due to usage of a distribution that is not expressive enough. The contours represent points with equal values of the PDF. The simple multivariate Gaussian (solid lines) fails to capture the multimodality of the true mixture of Gaussians distribution (dashed lines).

Mixture models A mixture of Gaussians, a linear superposition of Gaussian components, introduces mixing coefficients π, measuring the probability of a sample coming from a particular component of the superposition. Some problems render themselves naturally to mixture models. For example, the distribution of the number of customers at a store at a given time of the day on a random day can be considered as superimposed distributions of the number of customers during the weekend and the weekdays. Mixture models can introduce so-called unobserved or *latent* variables if the mixing coefficients are not directly observed. Here, the missing information could be, for example, the day. If it is a weekday, the distribution most likely has more mass in the late afternoon and early evening, whereas during the weekend there is more mass during midday. Successful latent-variable models can yield interesting information about a system, if structure can be identified in the space of the latent variables. If we were completely unaware of the concept of a weekend, employing a latent-variable model to the customer data could potentially reveal us that there indeed is two sets of days that have very different properties.

The general Gaussian mixture model is given by

$$q(\boldsymbol{x}) = \sum_i \pi_i N(\boldsymbol{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i), \tag{2.4}$$

where π_i is mixing coefficient for which $\sum_i \pi_i = 1$. The mixing coefficients are in fact prior probabilities of latent variables since Eq. 2.5 can be rewritten as

$$q(\boldsymbol{x}) = \sum_i q(\boldsymbol{z}_i = 1) q(\boldsymbol{x}|\boldsymbol{z_i} = 1) = \sum_{\boldsymbol{z}} q(\boldsymbol{z}, \boldsymbol{x}), \tag{2.5}$$

that is, the latent variable $\boldsymbol{z}$ is *marginalized* out.

Graphical models The Gaussian mixture model is a special case of a more general family of probabilistic models called *Bayesian networks* or *directed graphical models*. Bayesian networks arise from a certain factorization of a joint probability distribution which can be represented using a directed graph. In general terms, the *chain rule of probability* states that every probability distribution p can be expressed as follows

$$p(\boldsymbol{x}) = \prod_i p(\boldsymbol{x}_i|\boldsymbol{x}_{<i}), \tag{2.6}$$

where $\boldsymbol{x}_{<i} = [x_1, \ldots, x_{i-1}]$. However, if some of the elements of the vector of variables are independent and there are no cyclical dependencies, the above equation reduces into

$$p(\boldsymbol{x}) = \prod_i p(\boldsymbol{x}_i|\text{parent}(\boldsymbol{x}_i)), \tag{2.7}$$

where parent$(\boldsymbol{x}_i)$ are the *parents* of $\boldsymbol{x}_i$ in the *directed acyclic graph* (DAG) that represents the distribution p. It is noteworthy that the complexity of the distribution greatly decreases if each $\boldsymbol{x}_i$ depends only on a constant number of other nodes. Due to the factorization, sampling from directed acyclic graphical models is also relatively simple. Starting from the nodes that have no parents, we can move through the graph, sampling from the corresponding conditional distribution $p(\boldsymbol{x}_i|\text{parents}(\boldsymbol{x}_i))$. This technique is called *ancestral sampling.* Samples from any marginal distribution of p can be acquired by simply discarding the elements over which the distribution is marginalized.

Not all systems can be modeled with directed graphs, which are only feasible in situations where there is a clear causal, one-directional interaction with variables. *Undirected models* also known as *Markov random fields* allow, as their name suggests, undirected edges in the graph modeling some probability distribution. Whereas undirected models are defined directly with proper probability distributions, undirected models work with more loosely constrained potential functions $\phi(\cdot)$. When multiplied together, the functions ϕ form an unnormalized probability distribution, which is normalized by finding the *partition function* $Z \equiv \int_{x \in X} \prod_i \phi_i dx$. Special attention needs to be paid to ensure that the partition functions exits (is finite) via choosing non-divergent functions ϕ. [7, 21]

Energy-based models (EBM) employ unnormalized density functions $\widetilde{p}$ with the form

$$\widetilde{p}(\boldsymbol{x}) = \exp\left(-E(\boldsymbol{x})\right), \tag{2.8}$$

where $E(\cdot)$ is the energy function. Such parametrization ensures the non-negativeness of the density function without any additional constraints and consequently allows Monte Carlo estimation of the gradient of the log-partition function of the unnormalized distribution $\widetilde{p}$. Distributions of the form $p(\boldsymbol{x}) \propto \exp\left(-E(\boldsymbol{x})\right)$ are known as *Boltzmann distributions* inherited — like many other concepts in probabilistic modeling — from statistical physics. Many energy-based models are hence called *Boltzmann machines.* [21]

Non-parametric models *Kernel density estimation* provides an alternative, non-parametric method of estimating the probability density function. The method is based on the observation that given enough samples N, $M = NP$ samples will be observed in the region X for which $P = \int_X p(\boldsymbol{x})\mathrm{d}x$. If the region X is small enough and with some volume V, the probability is simply $P \approx p(\boldsymbol{x})V$ and hence

$$p(\boldsymbol{x}) = \frac{M}{NV}. \tag{2.9}$$

We can define a *kernel function* k to act as an indicator function yielding one for values in the neighborhood of the origin and zero elsewhere. The kernel function centered at $\boldsymbol{x}$ can be used to count the number of observations near the kernel

$$M = \sum_{i=1}^{N} k(\boldsymbol{x} - \boldsymbol{x}_i). \tag{2.10}$$

Combining this with Eq. 2.9 yields an estimate for the density function

$$p(\boldsymbol{x}) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{V_k} k(\boldsymbol{x} - \boldsymbol{x}_i), \tag{2.11}$$

where V_k is the volume (integral) of the kernel. The functional form of the kernel is a design parameter, but is often chosen to be the Gaussian. [7]

Limitations Classical algorithms for solving probabilistic latent variable models often make strong assumptions about properties of the model. DAGs simplify a probability distribution by assuming a causal process which results into a more manageable factorization of the joint probability distribution. Unfortunately, building such a graphic model is not always possible if the process is too complex or not even directly observable.

The *expectation maximization* (EM) algorithm [14] — used to solve, for example, the Gaussian mixture model — assumes that the posterior $q(\boldsymbol{z}|\boldsymbol{x}) = q(\boldsymbol{z}, \boldsymbol{x})/q(\boldsymbol{x})$ is tractable (which it is with the Gaussian mixture model). However, in a more general setting this might not be the case. *Variational inference* (VI) [33] approximates an intractable log-probability with a tractable lower bound known as the *variational lower bound* or *evidence lower bound* (ELBO), which is maximized instead. Generally, a mean-field approximation is required, simplifying the intractable posterior distribution with a distribution which generally factorizes in a certain way. Resorting to variational approximation is a reason why some models fail to provide exact values for data likelihood, as only a lower bound can be computed.

Deep generative models combine classical probabilistic inference with modern deep learning based methods. A major problem with classical simple parametric models is the arbitrariness in the choice of the family of distributions used in the modeling. Alternatively, this can be seen as the requirement of having an expert to study the system and build a plausible, hand-tuned probabilistic model. Neural networks can provide a remedy by vastly expanding the family of distributions from which the optimal distribution is searched. Modern deep learning techniques are also designed to work with massive datasets efficiently, which is extremely important when modeling high-dimensional distributions due to the *curse of dimensionality*.

We dedicate the rest of this chapter to a brief look into deep neural networks as well as an introduction to some of the most common deep generative models and their evaluation, starting with variational autoencoders and generative adversarial nets. We pay special attention to the family of normalizing flow models, the principal method of this thesis.

2.2 Deep Neural Networks

Deep (feedforward) neural networks are, as their name suggests, loosely inspired by neuroscientific observations about biological neurons. Mathematically, they are essentially not much more than repeated matrix–vector multiplication (or in more general terms, affine transformations) combined with occasional non-linear *activation functions* such as the sigmoid or the hyperbolic tangent. A single *fully connected* or *dense* layer in a deep neural network hence computes

$$\boldsymbol{x}^{'} = \sigma\left(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}\right), \tag{2.12}$$

where $\sigma(\cdot)$ is some activation function. The layer is called dense as all elements of $\boldsymbol{x}$ affect each element in $\boldsymbol{x}^{'}$ via the weight matrix $\boldsymbol{W}$. The dimensionality of $\boldsymbol{x}^{'}$ does not necessarily need to be equal to that of $\boldsymbol{x}$. The *depth* of the network is determined by the number of such layers composed one after another. The *width* of the network depends on the dimensionality of the intermediate values $\boldsymbol{x}^{'}$ within the network. The non-linearities are crucial for the representational power of the network. Without the activations, the entire network could be reduced into a single affine transformation. It can be shown, that a feedforward network with at least one intermediate or *hidden* layer is a *universal approximator*, given that the hidden layer is wide enough [29]. This means that, in theory, a wide enough neural network can approximate any reasonably behaving (continuous, bounded) function arbitrarily well. The guarantee, however, only applies on the existence of such network (or a set of parameters parametrizing a network) and in practice we may fail to find the desired set of parameters.

Training a neural networks involves finding the optimal set of parameters $\boldsymbol{\theta}$ containing all the weights and biases in the layers of the network. The task in *supervised learning* is to have the function f_{NN} defined by network approximate the data-generating function f as well as possible with respect to a *cost* or *loss* -function ℓ, given paired training examples of $\boldsymbol{x}$ and $\boldsymbol{y} = f(\boldsymbol{x})$. We hope to minimize the expected loss over the data distribution, that is, the loss should vanish on datapoints that we are the most likely to encounter. Since we do not in general have a closed form for the true joint data distribution $p(\boldsymbol{x}, \boldsymbol{y})$, but rather only samples from it, we instead minimize a quantity known as the *empirical risk*

$$\mathbb{E}_{\boldsymbol{x},\boldsymbol{y}\sim p(\boldsymbol{x},\boldsymbol{y})}\left[\ell\left(f_{\mathrm{NN}}(\boldsymbol{x}_i), \boldsymbol{y}_i\right)\right] \approx \frac{1}{N}\sum_{i=1}^{N} \ell\left(f_{\mathrm{NN}}(\boldsymbol{x}_i), \boldsymbol{y}_i\right), \tag{2.13}$$

where the summation is over the paired training examples. It is not guaranteed that a network trained with empirical risk minimization does well with data not present in the paired training data. The dataset might be a poor representation of the underlying distribution, especially if the dataset is small in size. Additionally, the network can be overly expressive and learn to memorize the training dataset, misleadingly yielding vanishing expected loss. This is known as *overfitting*. Conversely, if the network is too simple and lacks the required representational power, *underfitting* may occur.

A good set of network parameters is found iteratively using *gradient descent*, a first order approximation of the high-dimensional loss landscape. At each step of the iteration, a small step is taken in the parameter space into the direction of the negative gradient of the cost-function with respect to the network parameters. The task is in general not convex and hence there are no guarantees of finding a global minimum for the cost. The gradient needed for the gradient descent is efficiently found and evaluated using *back-propagation* [57], a clever application of the chain rule of calculus.

The loss-function is in practice rarely evaluated for the entire dataset at a time. Instead, the data is split into smaller random subsets called *minibatches*. Computing a noisy estimate of the gradient over a minibatch is still correct on average, even though it is computationally much more lightweight, leading into more efficient training. Gradient descend using the stochastic gradients is still guaranteed to converge to a minimum given a couple of technical requirements for the step-size or *learning rate* used in the iteration [21]. The algorithm is known as the *stochastic gradient descent* (SGD). Extensions to SGD include algorithms with adaptive learning rates [17] and application of *momentum* — an exponentially decaying moving average of past gradients — in the weight update step [37].

Working with high-dimensional data quickly leads into memory problems due to the size of the weight matrices $\boldsymbol{W}$ scaling quadratically with the dimensionality. Allowing all elements of $\boldsymbol{x}$ to affect all elements of $\boldsymbol{x}'$ is not, however, strictly necessary. For example, modeling the relationship of a pixel in an image with all other pixels is mostly wasted effort, since the value of the pixel is probably mostly independent from pixels that are far away. Here, it is reasonable to model relationships only within a small, fixed neighborhood of a pixel. *Convolutional neural networks* (CNNs) [44] do exactly this with the convolution (or rather, the cross-correlation) operation. In addition to introducing sparsity to the connectivity, the convolution operation also brings equivariance to translation, meaning that a translated input yields a similarly translated output after the convolution. This is desirable, since an image shifted with a pixel or two is still more or less the same image. As a consequence of the equivariance, the shift does not introduce a large shift in the output of the convolution. This happens naturally and does not need to be learned, saving model capacity for other important aspects. With this compact primer to neural networks, we now move to probabilistic modeling with elements of deep learning.

2.3 Variational Autoencoders

An *autoencoder*, in general, is a function (often implemented with a neural network) that attempts to perform an identity operation via an encoding–decoding pass.

$$f(\boldsymbol{x}) = \text{Decode}(\text{Encode}(\boldsymbol{x})) \approx \boldsymbol{x}. \tag{2.14}$$

The operation is made non-trivial by restricting the model, for example, by forcing it to use a lower-dimensional intermediate representation of the input via the encoder [21].

The *variational autoencoder* (VAE) [40] employs variational methods on a simple latent-variable generative model $p(\boldsymbol{z})p(\boldsymbol{x}|\boldsymbol{z})$ to find an approximate posterior of the latent codes $\boldsymbol{z}$, $q(\boldsymbol{z}|\boldsymbol{x})$. That is, instead of employing deterministic functions, the encoder and decoder are probabilistic. The posterior is referred as the *recognition model* or the *encoder*, while the likelihood $p(\boldsymbol{x}|\boldsymbol{z})$ is the *decoder* (see Fig. 2.5). Each datapoint $\boldsymbol{x}$ produces a distribution of latent codes $\boldsymbol{z}$ from which the datapoint could have been generated.

A variational approximation is required since the true posterior $p(\boldsymbol{z}|\boldsymbol{x})$ is in general intractable due to the challenging marginalization integration required for finding $p(\boldsymbol{x})$. Applying the standard variational inference procedure and finding the variational lower bound L for the likelihood of the

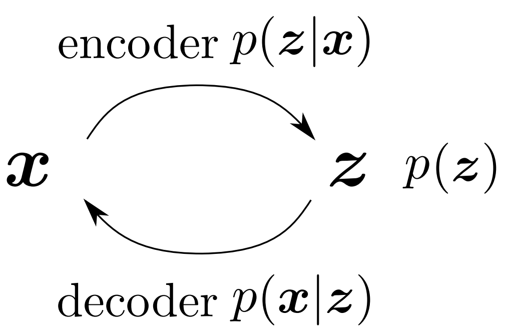

encoder $p(\boldsymbol{z}|\boldsymbol{x})$
$\boldsymbol{x}$
$\boldsymbol{z}$
$p(\boldsymbol{z})$
decoder $p(\boldsymbol{x}|\boldsymbol{z})$

Figure 2.5: Variational Autoencoder. The model is sampled by first sampling $\boldsymbol{z} \sim p(\boldsymbol{z})$ followed by $\boldsymbol{x} \sim p(\boldsymbol{x}|\boldsymbol{z})$. The density function of $\boldsymbol{x}$ alone, $p(\boldsymbol{x})$ is intractable due to the need of marginalization of $\boldsymbol{z}$ and hence variational methods are required.

observations $\boldsymbol{x}$ using Jensen's inequality yields

$$\begin{aligned}
\log(p(\boldsymbol{x})) &= \log \int p(\boldsymbol{x}, \boldsymbol{z}) \mathrm{d}\boldsymbol{z} \\
&= \log \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})} \left[\frac{p(\boldsymbol{x}|\boldsymbol{z}) p(\boldsymbol{z})}{q(\boldsymbol{z}|\boldsymbol{x})} \right] \\
&\geq \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})} \left[\log \left(\frac{p(\boldsymbol{z})}{q(\boldsymbol{z}|\boldsymbol{x})} \right) \right] + \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})} \left[\log(p(\boldsymbol{x}|\boldsymbol{z})) \right] \\
&= -D_{\mathrm{KL}}(q(\boldsymbol{z}|\boldsymbol{x}) || p(\boldsymbol{z})) + \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})} \left[\log \left(p(\boldsymbol{x}|\boldsymbol{z}) \right) \right] \\
&\equiv L(\boldsymbol{x}),
\end{aligned} \tag{2.15}$$

where $p(\boldsymbol{z})$ is a known prior for $\boldsymbol{z}$ (e.g. a multivariate Gaussian). The first term can intuitively be interpreted as a regularizer as it forces the recognition model to follow the simple form of the prior. The latter term maximizes the expected likelihood of the data over an encoder–decoder pass, that is, minimizes the reconstruction error. The model is trained by maximizing the lower bound L of the expected log-likelihood $p(\boldsymbol{x})$ with respect to the parameters of the likelihood $p(\boldsymbol{x}|\boldsymbol{z})$ and the approximate posterior $q(\boldsymbol{z}|\boldsymbol{x})$.

The variational lower bound is maximized using stochastic gradient descent which requires a (Monte Carlo) gradient estimate which can be shown to exhibit large variance [50]. This problem is amended by reparametrizing the recognition model with a differentiable function $g(\boldsymbol{x}, \boldsymbol{\epsilon})$, with $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$. The function g can be parametrized with a neural network, for example, letting $g(\boldsymbol{x}, \boldsymbol{\epsilon}) = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$, where $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are outputs of a neural network given $\boldsymbol{x}$ as the input. This technique is known as the *reparametrization trick.*

The variational autoencoder, like many other likelihood-based neural density estimators, use neural networks to parametrize known and easy to work with probability distributions. This allows the networks to be essentially unconstrained since there few strict requirements about the outputs. As long as the parametrized distributions and the prior are simple enough, sampling new points is readily achievable. Conversely, only the lower bound for the likelihood of the data is ever computed, which is not necessarily the case with other methods. While the structure of the latent $\boldsymbol{z}$ can readily be explored with the encoder and decoder, VAE-based models are generally inferior in terms of sample quality compared with generative adversarial nets, which are discussed next.

2.4 Generative Adversarial Nets

While most deep generative models approximate a probability distribution explicitly by maximizing the likelihood (or its lower bound) of the data, *generative adversarial nets* (GANs) [22] learn a distribution implicitly. A GAN is composed of two competing networks, a *generator* tasked with generation of samples resembling those of the data distribution, and a *discriminator* (or a *critic*) estimating the probability of a sample having been generated by the generator or having been sampled from the true distribution (see Fig. 2.6).

$$\begin{tikzcd}
& \boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x}) \arrow[dr] & & \\
& & D(\boldsymbol{x}) \arrow[r] & \text{fake}\big/\text{genuine} \\
\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z}) \arrow[r] & G(\boldsymbol{z}) \arrow[ur] & &
\end{tikzcd}$$

Figure 2.6: Generative adversarial nets. The generator network $G(\cdot)$ transforms latent vectors $\boldsymbol{z}$ into samples which are to be indistinguishable from the true samples $\boldsymbol{x}$. The task of the discriminator $D(\cdot)$ is to detect which samples are generated by the generator and which originate from the true distribution p_{data}. Note how no likelihood function is ever evaluated.

This results into a two-player minimax game where the generator attempts to maximize the probability of the discriminator making a mistake. The training objective can be expressed as

$$\min_G \max_D \quad \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}\left[\log(D(\boldsymbol{x}))\right] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}}\left[\log(1 - D(G(\boldsymbol{z})))\right], \tag{2.16}$$

where $D(\cdot)$ is the confidence of the discriminator for the sample coming from the true distribution and $G(\boldsymbol{z})$ is a sample generated by the generator with random seed $\boldsymbol{z}$. A GAN never evaluates the likelihood of the observed data directly and hence sidesteps many of the related challenges. Conversely, however, the latent code is not as accessible as it is, for example, in VAE-based models. One can back-propagate through the generator, but no direct inverse is available.

It can be shown, that the optimal generator and discriminator do exist and they are found at the global optimum of Eq. 2.16. At the optimum, the distribution of the samples produced by the generator network exactly matches that of the true data distribution and the discriminator assigns equal probability to the sample coming from the generator and the true distribution. Furthermore, given enough capacity for G and D, by alternatingly updating the weights of the generator and discriminator networks, the optimum is guaranteed to be reached in theory. However, in practice, training a GAN can be notoriously difficult due to instability in the training. Consequently, much work has been conducted to improve and stabilize the training process [4, 35, 47].

While GANs achieve impressive results in many generative image-related tasks [36, 74], they suffer from the well documented problem of *mode collapse*. That is, instead of properly assigning probability mass to all plausible points of data, they tend to underestimate the variance of the target distribution in exchange of being able to generate high-quality samples. Models trained with maximum likelihood (or equivalently KL-divergence minimization) make the opposite choice of forming a distribution covering all the data points but overestimating the variance and yielding low-quality samples much more frequently. This behavior is sometimes called *zero-avoidance*.

2.5 Autoregressive Models

An autoregressive (AR) model requires deciding an ordering of the variables of the problem. It models the probability distribution of each element of the random vector with an aggregate of the values of the previous elements. This family of techniques is well-suited for time-series analysis, where an inherent ordering for the data exists [9]. Unlike in variational autoencoders, autoregressive models do not contain latent variables. As a result, intractable marginalization is avoided and no variational methods are required. Instead, autoregressive models present a probability distribution exactly and without resorting to a variational lower bound using the chain rule of probability (Eq. 2.6). The decomposition can always be made and in itself, makes no assumptions about the conditional dependencies of the probability distribution. The model is only limited by the choice of family of distributions of the conditionals. There is also an inherent ambiguity that directed models solved with causality arguments: not all vectors $\boldsymbol{x}$ possess a natural ordering. Time-series data does, but individual images do not and hence it is not clear which ordering of the elements should be used in the expression of the joint probability distribution. The required conditional probabilities $p(x_i|\boldsymbol{x}_{<i})$ can be each realized with a neural network which takes as input the previously generated $\boldsymbol{x}_{<i}$ and (in case of $\boldsymbol{x}$ being discrete, like image pixel-values) outputs a probability distribution for x_i.

The neural network used in the formation of the conditionals can be realized by employing a masking scheme like in the *masked autoencoder distribution estimator* (MADE) [20]. The aforementioned architecture uses binary masking on a fully connected neural network to enforce the autoregressive property. Other suggested architectures employ restricted Boltzmann machines [43, 70] and convolutional or recurrent neural networks [49]. While autoregressive models tend to achieve better likelihood scores than other models capable of direct evaluation of the data likelihood, they suffer from slow $\mathcal{O}(D)$ sampling caused by the serial factorization of the probability density.

2.6 Normalizing Flows

Autoregressive models permit an exact evaluation of the model probability density function, but fail to naturally model unordered data. Normalizing flows are another family of models for density estimation that allow constructing an explicit PDF without resorting to variational methods. They, however, do not require the data to be ordered and are much more efficient at generating samples compared to autoregressive models.

Flow-models in the context of density estimation have their roots in *whitening transformations*, and *Gaussianization* [52]. Gaussianization attempts to iteratively render the components of a random vector $\boldsymbol{x}$ as independent as possible with an invertible transform f using an EM algorithm [10]. Normalizing flows in their current form were conceptualized by Tabak and Turner in 2013 [65]. They recognized that invertible transforms can be *composed* into more expressive transformations and that there exists a duality between density estimation and normalization (Gaussianization). The search of the density estimate $q(\boldsymbol{x})$ can be reformulated into the search of the best normalizing mapping $f(\cdot)$. If the mapping perfectly transforms the target distribution into a normal distribution, due to invertibility it also transforms a normal distribution into the target distribution.

A normalizing flow models a probability distribution using the *change of variables formula*

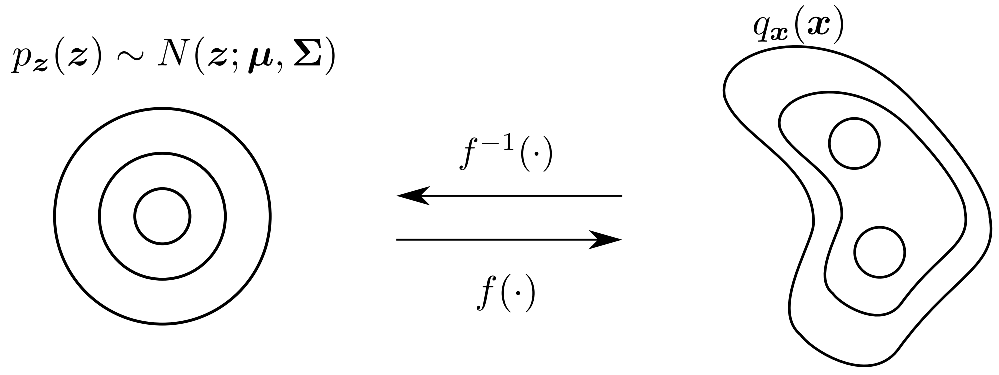
$$q_{\boldsymbol{x}}(\boldsymbol{x}) = p_{\boldsymbol{z}}(f^{-1}(\boldsymbol{x}))\, |J_{f^{-1}}(\boldsymbol{x})| \,, \tag{2.17}$$

where $\boldsymbol{x} = f(\boldsymbol{z})$ is an arbitrary invertible and differentiable function and $|J_{f^{-1}}(\boldsymbol{x})|$ is the determinant of the Jacobian of f^{-1} with respect to $\boldsymbol{x}$. The Jacobian determinant can be thought as a regularizer that ensures proper normalization of the PDF after the change of variables. It is a direct consequence of the change of variables in multivariate integration. Similar conversion factors appear, for example, in moving from the Cartesian coordinate system into polar coordinates in multivariable integration. If $f(\cdot) = (f_1 \circ f_2)(\cdot) \equiv f_1(f_2(\cdot))$ is a composite function as discussed earlier, the Jacobian is simply the product of the Jacobians of the individual functions due to the multivariate chain rule. Furthermore, the composition of invertible functions remains invertible. These properties are employed in essentially all modern normalizing flows. *The study of normalizing flows for density estimation usually concentrates in the search of families of functions f, which are both expressive and efficiently invertible, but also yield a tractable Jacobian determinant.* In order to see why those properties are necessary, we first need to understand how flow models are trained.

Training Similarly to classical parametric models for density estimation, the training goal of a flow models is the minimization of a divergence-metric, such as the KL-divergence (Eq. 2.3). As suggested previously, this is equivalent to maximizing the likelihood of the observations under the distribution spanned by the model. Evaluating the expectation analytically is not tractable but since the observations $\boldsymbol{x}_i$ are assumed to be samples from the true underlying distribution p, a Monte Carlo estimate for the expected likelihood L can be computed

$$\begin{aligned} L(q) &\approx \frac{1}{N}\sum_{i=1}^{N} \log\left(q(\boldsymbol{x}_i)\right) \\ &= \frac{1}{N}\sum_{i=1}^{N} \log\left(p_{\boldsymbol{z}}(f^{-1}(\boldsymbol{x}_i))\right) + \log\left(|J_{f^{-1}}(\boldsymbol{x}_i)|\right), \end{aligned} \tag{2.18}$$

where we have used Eq. 2.17 as the expression for q and $\boldsymbol{x}_i \sim p(\boldsymbol{x})$. If the function f is implemented by a neural network and expressions for f^{-1} and $J_{f^{-1}}(\boldsymbol{x})$ are available for evaluation, can the likelihood MC-estimate be minimized using standard stochastic gradient descend methods. Conversely, generating samples from the model requires evaluating f and being able to sample from the prior $p_{\boldsymbol{z}}(\boldsymbol{z})$. If the family of functions from which f is chosen is rich enough and a global optimum in the training is found q is exactly the same as p since the KL-divergence vanishes. Consequently, as the mapping between $\boldsymbol{x}$ and $\boldsymbol{z}$ is bijective, the observations mapped into $\boldsymbol{z}$ via f^{-1} perfectly follow the chosen prior distribution $p_{\boldsymbol{z}}(\boldsymbol{z})$. The flow model is *normalizing* since it maps a complex distribution into a simpler one (often chosen to be the Gaussian or the *normal* distribution). We visualize the normalization process in Fig. 2.7.


$p_{\boldsymbol{z}}(\boldsymbol{z}) \sim N(\boldsymbol{z}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$
$f^{-1}(\cdot)$
$f(\cdot)$
$q_{\boldsymbol{x}}(\boldsymbol{x})$

$$q_{\boldsymbol{x}}(\boldsymbol{x}) = p_{\boldsymbol{z}}(f^{-1}(\boldsymbol{x})) \left| J_{f^{-1}}(\boldsymbol{x}) \right|$$

Figure 2.7: Normalizing flow with a change of variables. The simple Gaussian distribution with density $p_{\boldsymbol{z}}(\cdot)$ is transformed with a invertible and differentiable transformation $f(\cdot)$ into $q_{\boldsymbol{x}}(\cdot)$, which minimizes some divergence with the (unknown) true data distribution p_{data}. Alternatively the inverse function $f^{-1}(\cdot)$ can be thought to transform the complex distribution $q_{\boldsymbol{x}}$ into a normal distribution, that is, normalize it.

As opposed to autoregressive models, the computational complexity of normalizing flows does not in general scale linearly with the number of dimensions D during sampling. Instead, we are free to make a modeling decision in the number of composed bijections K, which in general is much smaller than D. The training of the model, however, requires the computation of the Jacobian determinant, which in the general case is a restrictive $\mathcal{O}(D^3)$ operation. Most bijections used in flow models are specifically crafted to have at most a linear time complexity with respect to D in the evaluation of the Jacobian determinant. Another common goal is ensuring efficient invertibility, usually by employing simple bijections such as affine transformations.

We will now introduce in detail some methods for ensuring sub-cubic or otherwise more manageable Jacobian determinants from the literature. Many methods ensure that the Jacobian be triangular, significantly simplifying the computation of the determinant.

Residual flows Linear-time determinant can be achieved, for example, with clever use of the properties of the determinant. One can employ the matrix determinant lemma for a family of residual transformations of the form

$$f(\boldsymbol{x}) = \boldsymbol{x} + \boldsymbol{u}h\left(\boldsymbol{v}^{\mathrm{T}}\boldsymbol{x} + b\right), \tag{2.19}$$

where $\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^D$, and h is a smooth, differentiable element-wise nonlinearity [55]. According the the aforementioned lemma,

$$\det(J_f(\boldsymbol{x})) = \det\left(\boldsymbol{I} + \boldsymbol{u}\left[h^{'}(\boldsymbol{v}^{\mathrm{T}}\boldsymbol{x} + b)\boldsymbol{v}\right]^{\mathrm{T}}\right) = 1 + h^{'}(\boldsymbol{v}^{\mathrm{T}}\boldsymbol{x} + b)\boldsymbol{u}^{\mathrm{T}}\boldsymbol{v}, \tag{2.20}$$

which is clearly linear in D — only inner products between $\boldsymbol{x}$, $\boldsymbol{u}$ and $\boldsymbol{v}$ need to be computed. The matrix determinant lemma can readily be generalized for matrices to find

$$\det\left(\boldsymbol{A} + \boldsymbol{U}\boldsymbol{V}^{\mathrm{T}}\right) = \det\left(\boldsymbol{I}_m + \boldsymbol{V}^{\mathrm{T}}\boldsymbol{A}^{-1}\boldsymbol{U}\right)\det\left(\boldsymbol{A}\right), \tag{2.21}$$

where $\boldsymbol{A}$ in an invertible $D \times D$ matrix and $\boldsymbol{U}$ and $\boldsymbol{V}$ $D \times M$ matrices. In the special case where $\boldsymbol{A} = \boldsymbol{I}$, the equality is know as *Sylvester's theorem for determinants*. The theorem is utilized in *Sylvester normalizing flows* [6] to convert a prohibitive $D \times D$ matrix determinant computation into a more manageable $M \times M$ matrix determinant, assuming $M \ll D$. The determinant retains the cubic time complexity, but is only $\mathcal{O}(M^3 + M^2D)$, from the determinant and $\boldsymbol{V}^{\mathrm{T}}\boldsymbol{U}$ computations, respectively. Special attention needs to be paid to the matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ such that the equivalent of Eq. 2.19 with matrices instead of vectors, remains invertible.

Continuous-time flows The residual form of Eq. 2.19 is suggestive in the sense that it yields the state of the flow after one layer in the form

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + g(\boldsymbol{x}), \tag{2.22}$$

which can be seen as a step of the Euler method for solving ordinary differential equations (ODEs). In the limit of adding more layers and taking smaller steps the dynamics of the normalizing flow start to become specified by a differential equation defined by a neural network. *Continuous-time* normalizing flows do not have a well-defined notion of depth. They model the change of the probability density function as a function of continuous time using an ODE. The time derivative of the log-density is given by the *instantaneous change of variables* [11]

$$\frac{\partial \log(p(\boldsymbol{z}(t)))}{\partial t} = -\mathrm{Tr}\left(J_f(\boldsymbol{z}(t))\right), \tag{2.23}$$

where $\mathrm{Tr}\left(J_f(\boldsymbol{z}(t))\right)$ is the trace of the Jacobian. The log-probability of $\boldsymbol{z}$ at time t can be acquired by solving 2.23, which can be done efficiently with a standard numerical ODE solver. During training, gradient of the log-likelihood with respect to the parameters $\boldsymbol{\theta}$ of f is required. Fortunately, having a differentiable ODE-solver is not necessary. It can be shown, that for a scalar loss function $L(\boldsymbol{z}(t))$ (such as the expected likelihood) the derivative of the loss can be solved using the *adjoint sensitivity method* [8]:

$$\frac{dL}{d\theta} = -\int_{t_1}^{t_0} \left(\frac{\partial L}{\partial \boldsymbol{z}(t)}\right)^{\mathrm{T}} \frac{\partial f(\boldsymbol{z}(t), t; \theta)}{\partial \theta} \mathrm{d}t, \tag{2.24}$$

which can be computed with another forward pass of an ODE solver. As the gradients are given by another ODE, the solver can be treated as a black box and it *does not need to be differentiable.* Using continuous time dynamics lifts the prohibitive $\mathcal{O}(D^3)$ determinant computation and replaces it with the trace of the Jacobian. The Free-Form Jacobian of Reversible Dynamics (FFJORD) [23] model employs stochastic estimation of the trace of the Jacobian with Hutchinson's trace estimate [32]. The trace is given by computing the MC estimate of

$$\mathrm{Tr}(\boldsymbol{A}) = \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\boldsymbol{\epsilon}^{\mathrm{T}} \boldsymbol{A} \boldsymbol{\epsilon}\right], \tag{2.25}$$

where $\boldsymbol{\epsilon}$ is distributed with zero mean and identity covariance. The vector–Jacobian product $\boldsymbol{\epsilon}^{\mathrm{T}} \boldsymbol{J}$ is computed with reverse-mode automatic differentiation with $\mathcal{O}(D)$-time complexity instead of explicitly computing the diagonal of the Jacobian yielding an $\mathcal{O}(D)$ complexity for the trace computation without assuming anything about the form of the Jacobian matrix. While continuous-time flows lift the computation of the troublesome Jacobian determinant, they regardless somewhat suffer from scalability problems with high-dimensional data.

Autoregressive flows Normalizing flows can also employ the autoregressive property by mixing variables such that each component only depends on the previous components. In fact, the property is very fitting for flow models as it ensures that the Jacobian of the transformation is triangular and hence the determinant merely the product of the diagonal elements. In general, an *autoregressive flow* (AF) transforms a variable one element i at a time using

$$x_i = \tau\left(z_i, c(\boldsymbol{x}_{<i})\right), \tag{2.26}$$

where τ is an invertible *transformer* and c is the *conditioner*. The *masked autoregressive flow* (MAF) [53] transforms a random vector $\boldsymbol{z}$ using an affine transformer:

$$x_i = z_i \exp\left(f_{s_i}(\boldsymbol{x}_{<i})\right) + f_{b_i}(\boldsymbol{x}_{<i}), \tag{2.27}$$

where f_{s_i} and f_{b_i} are scalar functions usually implemented with neural networks. The inverse is simply

$$z_i = \left(x_i - f_{b_i}(\boldsymbol{x}_{<i})\right) \exp\left(-f_{s_i}(\boldsymbol{x}_{<i})\right). \tag{2.28}$$

Note how during mapping from $\boldsymbol{z}$ to $\boldsymbol{x}$ (i.e. sampling), the components of $\boldsymbol{x}$ need to be necessarily computed serially because of the dependency chain of previous x_i. However, computing the inverse (probability density evaluation) can be parallelized since the full vector $\boldsymbol{x}$ is available. The Jacobian is triangular by design and the log-determinant is simply the sum of the scales $f_{s_i}(\boldsymbol{x}_{<i})$. The *inverse autoregressive flow* (IAF) [39] makes an opposite choice in terms of the computational trade-off by having the bijections f_{s_i} and f_{b_i} depend on the vector $\boldsymbol{z}$ instead of $\boldsymbol{x}$. Now, during the forward pass (sampling), computation can be parallelized, since the full vector $\boldsymbol{z}$ is available. Conversely, the inverse needs to be computed serially.

Coupling layers The *non-linear independent components estimation* (NICE) [15] model uses *coupling layers*, somewhat resembling the operation of AF/IAF, where at each step of the flow, the vector $\boldsymbol{x}$ is split into two parts $\boldsymbol{x}_a, \boldsymbol{x}_b$, the first is used to compute bias-terms for the other and the other remains unchanged:

$$\begin{aligned}
[\boldsymbol{x}_a, \boldsymbol{x}_b] &= \mathrm{split}(\boldsymbol{x}) \\
\boldsymbol{x}_a^{'} &= \boldsymbol{x}_a + g(\boldsymbol{x}_b) \\
\boldsymbol{x}_b^{'} &= \boldsymbol{x}_b \\
f(\boldsymbol{x}) &= \mathrm{concat}(\boldsymbol{x}_a^{'}, \boldsymbol{x}_b^{'}).
\end{aligned} \tag{2.29}$$

The function g can be implemented with an unconstrained neural network. Regardless, the Jacobian retains a simple triangular form with ones on the diagonal. No serial computation is necessary either during training or inference and the inverse operation is nearly the same as the forward computation, with only a sign flipped. The follow-up work *real non volume preserving-flow* (RealNVP) [16] extends NICE by adding an additional neural network g_s for computing scales for $\boldsymbol{x}_a^{'}$

$$\boldsymbol{x}_a^{'} = \boldsymbol{x}_a \odot \exp\left(g_s(\boldsymbol{x}_b)\right) + g(\boldsymbol{x}_b). \tag{2.30}$$

Here, the Jacobian is again a triangular matrix, but with elements on the diagonal possibly deviating from unity. The block corresponding to $\boldsymbol{x}_b$ is the identity whereas the one corresponding $\boldsymbol{x}_a$ is diagonal with elements $\exp\left(g_s(\boldsymbol{x}_b)\right)$. The log-determinant is hence the sum of the elements of $g_s(\boldsymbol{x}_b)$. Attempting to solve the toy-problem visualized earlier in Fig. 2.4 with a simple RealNVP-like normalizing flow with ~ 150 learnable parameters yields much more promising results as shown in Fig. 2.8. The toy problem would still be readily solvable with other, more flexible classical methods, such as the EM-algorithm for a mixture of Gaussians. Figure 2.9 shows the results of another RealNVP-like normalizing flow applied to a more complex distribution. While the reconstruction is not perfect, the model manages to reproduce the distribution well. It is noteworthy, however, that the model in both examples — like other models trained with likelihood maximization — overestimates variance and generates samples that have vanishing likelihood under the true model. The opposite is penalized heavily and hence the approximation q attains high density wherever the true distribution p has high density.

Both NICE and RealNVP have the problem of choosing the partitioning of $\boldsymbol{x}$. While choosing equal sizes for the two parts is not guaranteed to be the best option, it is often done to ensure proper mixing of information. More involved partitioning schemes must be employed for example when dealing with image data. An image can, for example, be partitioned spatially using a checkerboard pattern or splitting along the channel-dimension. The *Glow*-architecture [38] attempts to remedy the partitioning problem with *invertible* 1×1 convolutions. An ordinary partitioning corresponds to applying a permutation matrix and then splitting. Glow offers a way to learn a *continuous* variation of this partitioning with the invertible convolutions which essentially are matrix–vector products along the channel dimension. As the convolution kernel is a learned parameter, the best partitioning can at least in theory be found. Recent work extended the 1×1 convolutions by introducing invertible $d \times d$ convolutions interpreting the convolution (cross-correlation) operation as matrix–vector multiplication and constraining the matrix to be triangular ensuring efficient Jacobian determinant computation

[27]. There has also been other work on masked convolutions applied to normalizing flows [45, 63].

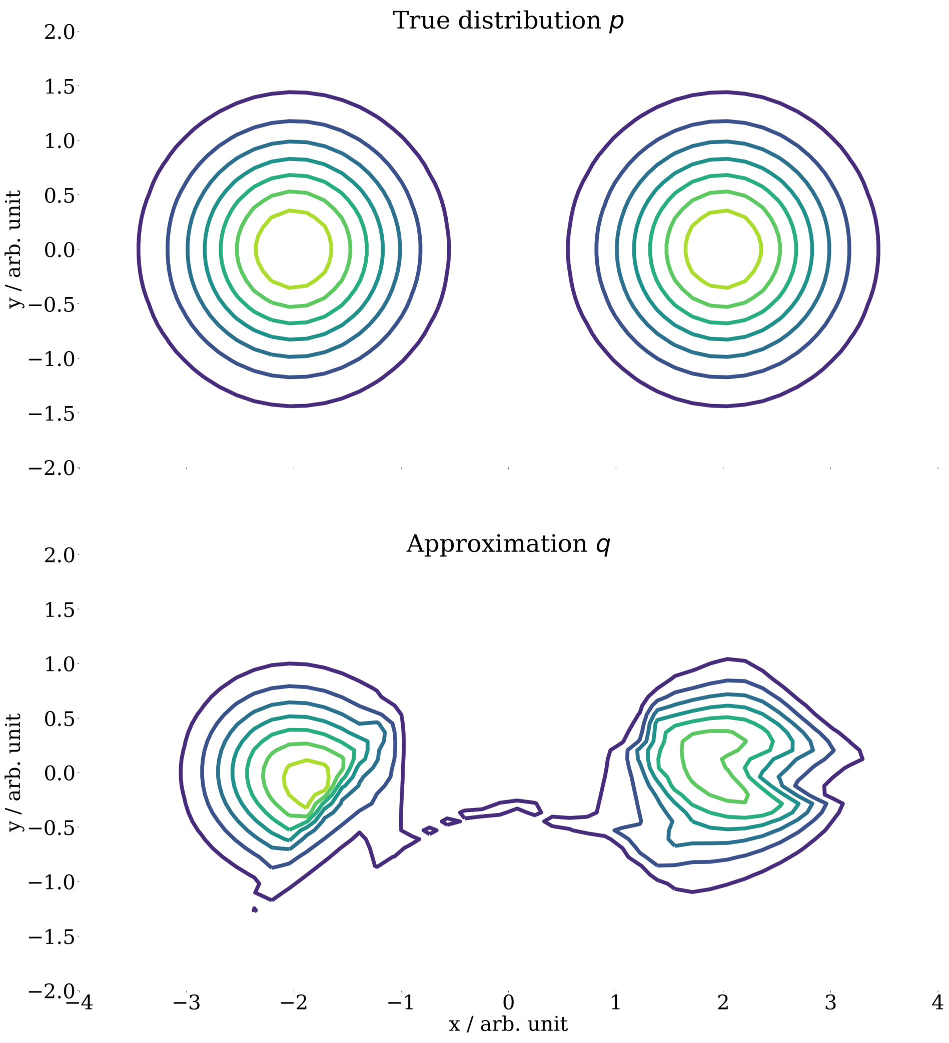


Figure 2.8: Mixture of multivariate Gaussians (top) approximated with a RealNVP-like normalizing flow (bottom). The contour-lines denote points with constant values of the PDF. While the flow-model is much more complex in terms of the number of parameters and the fact that it requires iterative optimization, it yields vastly superior results compared with simple parametric models.

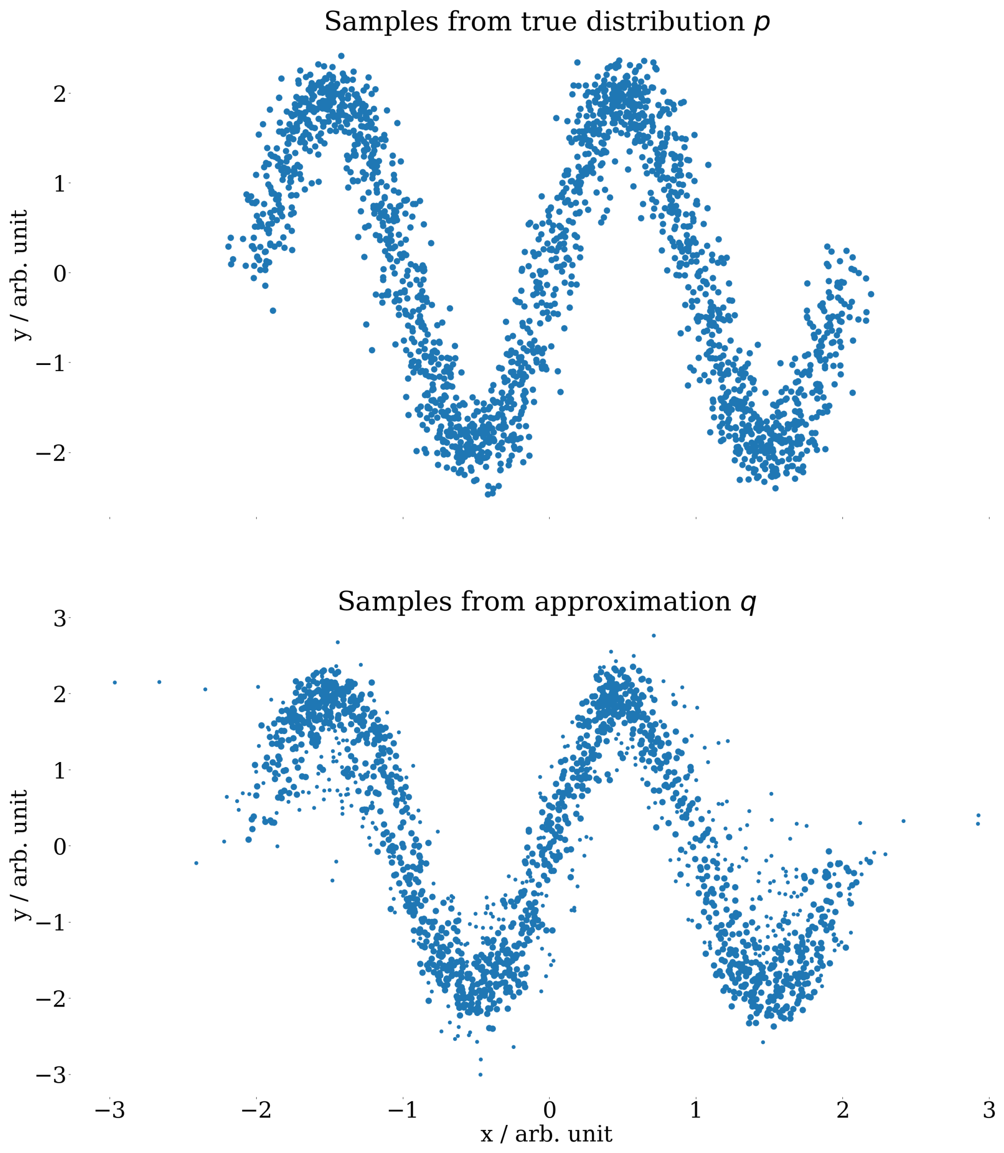


Figure 2.9: Approximation of a Gaussian with mean $\boldsymbol{\mu}(x) = [x, 2\sin(\pi x)]$, $x \sim \text{Uniform}(-2, 2)$ using a RealNVP-like normalizing flow. The top panel shows samples from the true distribution and the bottom panel samples from the approximation created with a normalizing flow. The model has high density wherever the true distribution has high density, but does generate samples that have very low likelihoods under the true distribution (qualitative outliers denoted with smaller dots in the bottom panel).

Numerical inversion While many flow models use transformations with deterministic analytic inverses and closed-form solutions for the Jacobian determinant/trace, are those not strictly necessary as already seen with the FFJORD trace estimate. In the literature, there are models that use iterative, numerical (fixed-point, bisection search) methods to find the inverse. The lack of analytic inverse often arises from using non-affine flows. Nearly all flow-models introduced earlier in this thesis have employed affine transformations which have very simple analytic inverses but which can cause problems with lacking expressivity. Non-linear activation functions, often found in neural networks, are absent in many flow models due to concerns about invertibility or training instability. Regardless, a flow-model could theoretically include non-linear (yet monotonically increasing) activation functions such as the hyperbolic tangent or the leaky rectified linear unit. Compositions and conic sums of such functions also remain monotonic and hence invertible. Employing transformations with non-trivial inverses in the context of normalizing flows can be lucrative due to a potential increase in their expressive power. Finding the inverse of the transformation is not necessarily the only computational challenge since the Jacobian determinant also becomes more complex.

The *neural autoregressive flow* (NAF) [30] replaces the affine transformer $\tau(\cdot)$ of AF/IAF with a deep neural network constrained to have non-negative weights and monotonic activations. NAF retains the conditioner network c, which is used to yield parameters for the transformer network. The *block neural autoregressive flow* (B-NAF) [13] removes the conditioner and hence renders the model more compact. It enforces the autoregressive property and monotonicity by directly applying constraints on the weights of the transformer network.

Another line of work deals with integration-based flows. The integral of any strictly positive function is clearly monotonic and hence integration can be used to generate bijections for normalizing flows. *Unconstrained monotonic neural networks* (UMNNs) [72] parametrize a strictly positive function with an unconstrained neural network with a biased exponential linear unit (ELU) activation at the end of the network enforcing positiveness. Evaluating the bijection requires integrating the output of the network which is carried out numerically using the Clenshaw–Curtis quadrature [12]. The inverse is found using bisection search. The Jacobian is constrained to be simple by employing autoregressivity and leveraging the simple partial derivatives of the integral-function. Müller *et al.* [48] build *piecewise continuous* first and second-order polynomials with neural networks and define the bijection with the integral of the polynomial. Here, the simple parametric form of the integrand yields a closed-form solution for the evaluation of the bijection and no numerical integration is necessary at the cost of reduced expressibility. Durkan *et al.* extend the above technique to piecewise cubic polynomials [18].

Discrete flows Despite essentially all data on a digital computer being discretized, discrete flows are not as prevalent in the literature. Discrete data is rather dequantized by adding noise. The change of variables formula (Eq. 2.17) applies only to continuous data and a different formula needs to be employed in the discrete case as follows

$$p(\boldsymbol{y}) = \sum_{\boldsymbol{x} \in f^{-1}(\boldsymbol{y})} p(\boldsymbol{x}), \tag{2.31}$$

where $f^{-1}(\boldsymbol{y})$ is the set of elements $\boldsymbol{x}$ subject to $f(\boldsymbol{x}) = \boldsymbol{y}$ [68]. If the function $f(\cdot)$ is invertible, there is only one element in the set and the summation vanishes. Even though moving to a discrete domain brings the benefit of the problematic Jacobian determinant vanishing from the equation, the transition does not come without problems. Back-propagation through non-differentiable discrete-output functions requires smooth approximations, like replacing the argmax-operation with a sharp yet differentiable softmax. Alternatively, the gradient of a discretization operation — like rounding — is effectively ignored by using the *straight through estimator* (STE) [5]. Discrete flows have seen use in *lossless compression* [28], where they are shown to perform significantly better than classical lossless compression methods (for image data) like PNG and moderately better than a VAE-based lossless compression method Bit-Swap [41]. The discrete flows are employed to find the probability mass function, which is in conjunction with a range-based asymmetric numerical system (rANS) used to try to approach the theoretical limit of Shannon's source coding theorem [60].

2.7 Comparison and Combinations

There is no method that is clearly better than others in all aspects. In fact, most of the methods introduced in this chapter are not even directly numerically comparable. Variational autoencoders only provide a lower bound for the data likelihood and GANs never even work with likelihoods explicitly. While autoregressive models usually yield superior likelihood scores compared to flow-models, they suffer from slow sampling and ambiguity in the ordering of the data-vector. Furthermore, a favorable likelihood-score does not necessarily translate into high-quality samples.

Relying only on the quality of the produced samples is equally questionable due to very different trade-offs between the models and the lack of a proper quantitative definition for "high quality". GANs generate qualitatively visually realistic samples, but often suffer from the absence of variation. Likelihood-based models offer worse sample quality but tend to have much higher variation. There has been work on creating metrics for evaluating the quality of generative model samples, such as the *Fréchet inception distance* (FID) [25] or the *inception score* (IS) [59]. Both of the metrics compact the quality into a single number immediately losing the obvious trade-off between realism and variation. More recent work has attempted to disentangle the aspects of quality into two separate dimensions via the notion of precision and recall, successfully demonstrating some of the shortcomings of one-dimensional measures of sample quality [42, 58].

There have also been efforts trying to combine some of the aforementioned methods or their basic ideas to obtain the best of both worlds. Autoregressive flows are an example of this, which has already been mentioned. Another method-combining approach, the Flow-GAN [24] mixes ideas from normalizing flows and GANs by suggesting a hybrid loss combining likelihood maximization with the adversarial loss given by Eq. 2.16. The generator of the GAN is invertible (a normalizing flow) and hence allow the evaluation of likelihood of data. Combining the two ideas, the authors show that with only using an adversarial loss, poor likelihoods are attained but the sample quality is high. Conversely, training with pure likelihood yields samples of worse quality. The hybrid loss is suspected to regularize the training in the sense of likelihood maximization and stabilized with respect to the adversarial objective. Interestingly, the hybrid objective is found to sometimes yield better test-likelihood scores than pure likelihood maximization.

Normalizing flows can also readily be combined with variational autoencoders. The representational power of a VAE is dependent on the form and expressibility of the approximate posterior $q(\boldsymbol{x}|\boldsymbol{z})$, which in the simple case is merely a multivariate Gaussian parametrized by a neural network. Since normalizing flows provide means for spanning highly flexible distributions, they are an excellent candidate for realizing the posterior distribution q. While employing normalizing flows in VAEs does not change the fact that only a lower bound is computed for the likelihood, they have been shown to bring clear improvements over simple parametrized distributions in several problems [55, 67].

Chapter 3

Inverse Problems and Bayesian Thinking

Sometimes two separate events are not *independent* and knowing the outcome of the first has an effect on the probability distribution of the other. The chain rule of probability states that the joint probability distribution of two random variables

$$p(\boldsymbol{x}, \boldsymbol{y}) = p(\boldsymbol{x}|\boldsymbol{y})p(\boldsymbol{y}), \tag{3.1}$$

where $p(\boldsymbol{x}|\boldsymbol{y})$ is the conditional probability of $\boldsymbol{x}$ given $\boldsymbol{y}$. The random variables are said to be independent if and only if

$$p(\boldsymbol{x}, \boldsymbol{y}) = p(\boldsymbol{x}|\boldsymbol{y})p(\boldsymbol{y}) = p(\boldsymbol{x})p(\boldsymbol{y}), \tag{3.2}$$

that is $p(\boldsymbol{x}|\boldsymbol{y}) = p(\boldsymbol{x})$, denoting that knowing the value of $\boldsymbol{y}$ yields no information on the distribution of $\boldsymbol{x}$.

The ability to estimate conditional distributions and sample from them is in the core of this thesis. In this chapter we draw a connection between conditional density estimation and inverse problems. Furthermore, we show how normalizing flows can readily be extended with conditioning to model conditional distributions without resorting to hand-crafted distributions and Bayes' theorem. In the end of the chapter, we provide a simple example of an implementation of a conditional normalizing flow on a toy problem.

3.1 Bayesian Methods

Rewriting Eq. 3.1 immediately leads into *Bayes' theorem*

$$\begin{aligned} p(\boldsymbol{x}|\boldsymbol{y})p(\boldsymbol{y}) &= p(\boldsymbol{y}|\boldsymbol{x})p(\boldsymbol{x}) \\ p(\boldsymbol{x}|\boldsymbol{y}) &= \frac{p(\boldsymbol{y}|\boldsymbol{x})p(\boldsymbol{x})}{p(\boldsymbol{y})}, \end{aligned} \tag{3.3}$$

where $p(\boldsymbol{y}|\boldsymbol{x})$ is the *likelihood*, $p(\boldsymbol{x})$ the *prior* and $p(\boldsymbol{x}|\boldsymbol{y})$ the *posterior*. Bayesian methods incorporate prior beliefs of the distribution of some variable $\boldsymbol{x}$ (the prior distribution) with observations of data $\boldsymbol{y}$ (the likelihood) to perform inference about the distribution of the variable given the observations. In simple cases, the posterior distribution can be solved analytically, given likelihood and prior distributions. This usually requires the prior and the likelihood to be *conjugate*, meaning they come from the same family of distributions. Conjugacy is necessary due to the $p(\boldsymbol{y})$-normalization term in the denominator, which, in general, is the (usually intractable) marginalization integral $\int p(\boldsymbol{y}|\boldsymbol{x})p(\boldsymbol{x})\mathrm{d}\boldsymbol{x}$. Conjugacy allows the normalization term to found *by inspection*, meaning that the (unnormalized) posterior follows the form of a known probability distribution allowing the normalization constant to be computed with a known formula given the parameters of the distribution. [19]

Conditional density estimation (CDE) is the task of finding the distribution of a variable $\boldsymbol{x}$ given the value of another variable $\boldsymbol{y}$. In Bayesian thinking, this equals to finding the posterior distribution of $\boldsymbol{x}$ given some observation $\boldsymbol{y}$. Unfortunately exact Bayesian treatment is in general intractable as even finding the likelihood model $p(\boldsymbol{y}|\boldsymbol{x})$, let alone the normalization constant may be too difficult. If there is a functional dependency $\boldsymbol{y} = g(\boldsymbol{x})$ (not necessarily a bijection), a method called *approximate Bayesian computation* (ABC) can be used. A simple form of ABC generates samples from the posterior using *rejection sampling*, by repeatedly sampling values of $\boldsymbol{x}$, mapping them to $\boldsymbol{y}$ via the known forward model g and accepting only those values of $\boldsymbol{x}$ which map within distance $\rho(\cdot,\cdot) \leq \varepsilon$ of the desired $\boldsymbol{y}$ [64]. ABC is a *likelihood-free* family of methods for generating samples from a posterior distribution [51], if a *simulator* for the *forward-model* $\boldsymbol{y} = g(\boldsymbol{x})$ is available. They are hence a way of generating solutions for *inverse problems* which are discussed next.

3.2 Inverse Problems

Inverse problems are usually encountered in situations where an unobserved quantity $\boldsymbol{x}$ is to be estimated based on measurements on a different observed quantity $\boldsymbol{y}$. The two quantities are related by some (possibly noisy) *forward model* as

$$\boldsymbol{y} = g(\boldsymbol{x}, \boldsymbol{\varepsilon}), \tag{3.4}$$

where $\boldsymbol{\varepsilon}$ denotes a set of noise variables, not of the primary interest. In the context of this thesis, image denoising can be given as an example of an inverse problem. Given a noisy image $\boldsymbol{y}$ "measured" by a camera, what are the corresponding clean, unobserved images $\boldsymbol{x}$? Since there are likely multiple clean pictures that could correspond to the noisy image, the solution is a *distribution* of images. Here, the values $\boldsymbol{\varepsilon}$ are parameters related to the unknown, stochastic photon capture process of the camera. Many inverse problems are extremely ill-posed simply due to an information losing nature of the forward process or due to the noise or uncertainty in the measurements. Various regularization methods can be employed to reduce the ill-posedness of the problem and to find a single, reasonable solution for the problem. In this thesis, however, we focus on *statistical* methods for inverse problems, and rather model uncertainty in the model with probability distributions. Classical statistical methods for inverse problems directly apply Bayes' law (Eq. 3.3) by finding an appropriate likelihood model for the forward process as well as specifying a prior distribution. In order to acquire samples, the likelihood and prior distributions need to be very simple to ensure a tractable posterior. Alternatively, expensive MCMC methods need to be employed. In contrast, we parametrize the conditional distribution $p(\boldsymbol{x}|\boldsymbol{y})$ *directly*, without applying Bayes' theorem, as explained in the following section. [34]

3.3 Deep Conditional Modeling

Alike to unconditional density estimation, classical conditional density estimation usually employs a fixed parametrized distribution for which optimal parameters are found. Contrary to unconditional density estimation, the outcome of the measurement is not completely determined by the unknown random vector $\boldsymbol{\varepsilon}$ as part of the input, denoted as $\boldsymbol{y}$ is controlled by us, as illustrated in Figure 3.1. Note how the modeling framework of Fig. 3.1 resembles the forward model of Eq. 3.4. It is exactly the *inverse*, implementing the mapping $\boldsymbol{x} = f(\boldsymbol{\varepsilon}, \boldsymbol{y})$!

Many methods and models introduced in Chapter 2 can readily be extended to perform conditional density estimation and yield conditional samples in the generative process. In practice this often means that the neural networks that are used to parametrize the distributions found in the density estimation models are given the conditioning information in addition to their regular inputs.

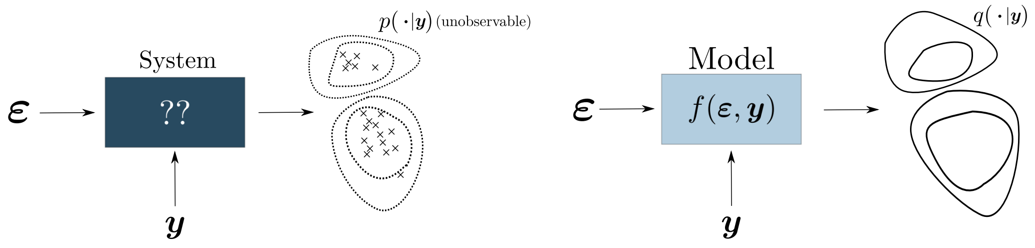


Figure 3.1: Conditional distribution approximation. The process is determined by vectors $\boldsymbol{\varepsilon}$ and $\boldsymbol{y}$, out of which the latter can be controlled. The modeling task is to approximate the true distribution $p(\cdot|\boldsymbol{y})$ (left), with q defined by the function f (right).

Conditioning in GANs GANs can be conditioned by concatenating the conditioning information into the latent noise of the generator network [46] or by applying conditional normalization layers. These layers operate by effectively setting the 1st and 2nd order feature statistics (mean and variance) of the image under generation into those of the conditioning image, within the hidden layers of the generator network. *Adaptive instance normalization* (AdaIN) [31] computes channel- and sample specific spatial mean and variance of a tensor, normalizes it with the computed values and finally rescales the normalized values with those computed using the conditioning tensor $\boldsymbol{y}$. The *spatially adaptive denormalization* (SPADE) [54] method computes the normalization similarly with AdaIN, but allows the scale and bias values computed using the conditioning tensor to be spatially varying. The above conditional normalization layers are not necessarily only to be used with GANs. In fact, conditional RealNVP-style coupling layers operate strikingly similarly by rescaling and shifting the input tensor based on the elements of the input tensor as well as the conditioning tensor. The conditional RealNVP layer lacks the explicit normalization step due to invertibility constraints, although it might be possible that the layer *learns* to perform an (invertible) operation close to that of spatially adaptive denormalization.

Conditioning in VAEs and AR-models The *conditional variational autoencoder* (CVAE) [62] augments the expression of the ELBO (Eq. 2.15) by conditioning each distribution with the conditioner $\boldsymbol{y}$. The conditional PixelCNN [71], a conditional autoregressive model, likewise simply adds a term dependent on the conditioning information into the *long short-term memory* (LSTM) layers employed by its unconditional predecessor PixelRNN [49].

Conditioning in NFs Flow-models have also recently been extended with conditioning. A conditional distribution can be realized by *partitioning* the flow into deterministic and sampled components $\boldsymbol{y}$ and $\boldsymbol{z}$, such that $\boldsymbol{x} = f(\boldsymbol{y}, \boldsymbol{z})$ [2] as illustrated in Fig. 3.2. The model is trained by applying a supervised loss for the part of $f^{-1}(\boldsymbol{x})$, corresponding to $\boldsymbol{y}$. The remaining part $\boldsymbol{z}$ is constrained to be distributed according to a simple prior, like the Gaussian. Sampling happens by concatenating the conditioning information $\boldsymbol{y}$ with the sampled $\boldsymbol{z}$ and applying $f(\cdot)$. At convergence, the model does not only approximate $p(\boldsymbol{x}|\boldsymbol{y})$, but also learns to approximate the unknown forward process. The method is shown to work with low-dimensional toy-data but its generalizability to high-dimensional data is unknown. The supervised loss causes additional stress to the already heavily constrained invertible transformation rendering work with more challenging data potentially troublesome. There is also an implicit assumption about the dimensionality of $\boldsymbol{y}$. Solving, for example, a denoising task is not possible since the forward process does not change the dimensionality, that is $\dim(\boldsymbol{x}) = \dim(\boldsymbol{y})$, and the sampled part $\boldsymbol{z}$ vanishes in the model.

Another line of work uses external neural networks to parametrize the bijections of the flow [3, 69, 73], sidestepping the need to partition the flow, but also losing the ability to learn the forward model. In this thesis, we choose to continue this line of work due to its roots residing in the state-of-the-art models of unconditional density estimation with normalizing flows [38]. We do not employ the partitioning scheme due to concerns about the its expressive limitations. Taking one step further, the work in [69] goes as far as to employ *Bayesian neural networks* in the generation of the flow-parametrizing values introducing regularization and applying priors to the parameter distributions. A mean field Gaussian approximation is used in conjunction with variational inference for the posterior of the network parameters.

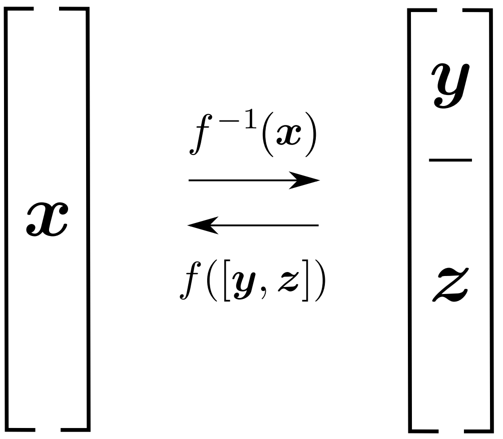


Figure 3.2: Normalizing flow conditioned via partitioning. The part $\boldsymbol{y}$ is the deterministic conditioning information while the rest of the vector (part corresponding to $\boldsymbol{z}$) is sampled. Samples are acquired by concatenating $\boldsymbol{y}$ and $\boldsymbol{z}$ and applying the transformation $f(\cdot)$.

3.4 Simple Conditional Flow

Conditioning can be readily implemented to the normalizing flow for the toy-dataset introduced in the previous chapter. The RealNVP-architecture computes the bias and scale vectors (Eq. 2.30) with neural networks $g(\cdot)$ and $g_s(\cdot)$. In the unconditional case its only argument is $\boldsymbol{x}_b$, that is the results of the split operation. The flow-model can be conditioned with some additional information $\boldsymbol{y}$, which can be given to the neural network as additional input. The conditional RealNVP-block simply computes

$$\begin{aligned}[][\boldsymbol{x}_a, \boldsymbol{x}_b] &= \mathrm{split}(\boldsymbol{x}) \\ \boldsymbol{x}_a^{'} &= \boldsymbol{x}_a \odot \exp\left(g_s(\boldsymbol{x}_b, \boldsymbol{y})\right) + g(\boldsymbol{x}_b, \boldsymbol{y}) \\ \boldsymbol{x}_b^{'} &= \boldsymbol{x}_b \\ f(\boldsymbol{x}) &= \mathrm{concat}(\boldsymbol{x}_a^{'}, \boldsymbol{x}_b^{'}), \end{aligned} \tag{3.5}$$

where the only difference with respect to the unconditional model is that the neural networks g_s and g additionally accept $\boldsymbol{y}$ as a parameter. Conditioning the mixture of Gaussians toy-model with the *discrete* cluster indices correctly recovers the individual Gaussians as visualized in Fig. 3.3. Recovering the two components is a rather simple task, since the base distribution of the flow

$p_{\boldsymbol{z}}$ only needs to be offset horizontally given the conditioning information $\boldsymbol{y}$ about the cluster membership.

Figure 3.4 illustrates a conditional variant of the more complex toy-data distribution introduced the the previous chapter. Here, we let the forward process be simply

$$\boldsymbol{y} = g(\boldsymbol{x}) = x_1, \tag{3.6}$$

that is, pick the first element of the vector $\boldsymbol{x}$. We hence seek to model the conditional distribution $p(\boldsymbol{x}|x_1)$. We find that the conditional model correctly yields samples at the correct location x_1 but that the values x_2 have too large a variance given the true distribution. The model, however, is quite simple and not too much time was spent on optimizing the results. A more expressive and tuned model would likely perform better.

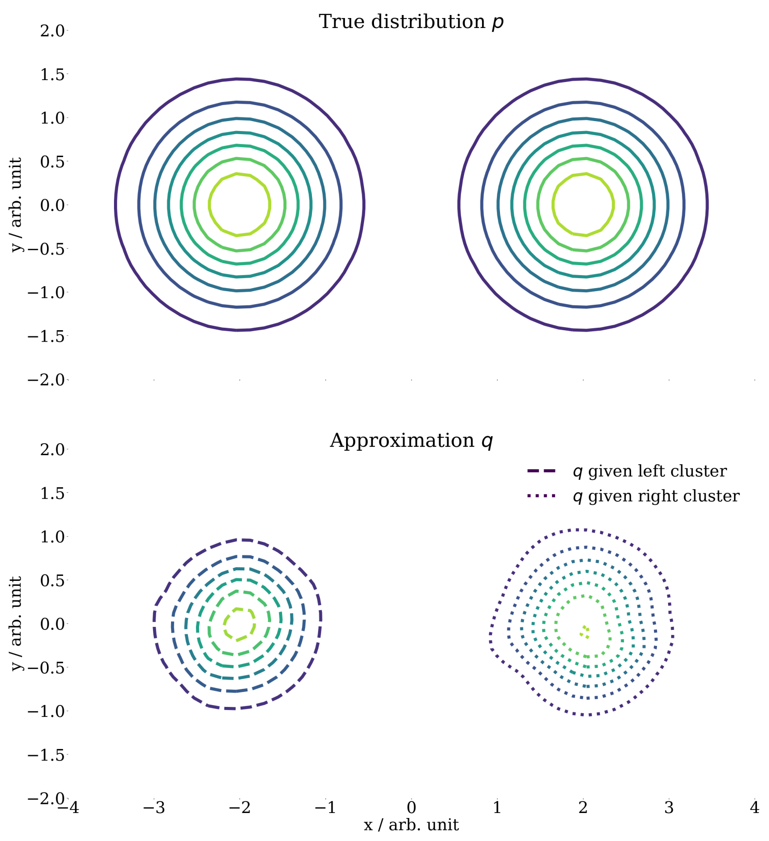


Figure 3.3: Mixture of multivariate Gaussians (top) approximated with a RealNVP-like conditional normalizing flow (bottom). Dashed lines correspond to contours of $q(\boldsymbol{x}|\text{cluster=left})$ and dotted lines to contours of $q(\boldsymbol{x}|\text{cluster=right})$.

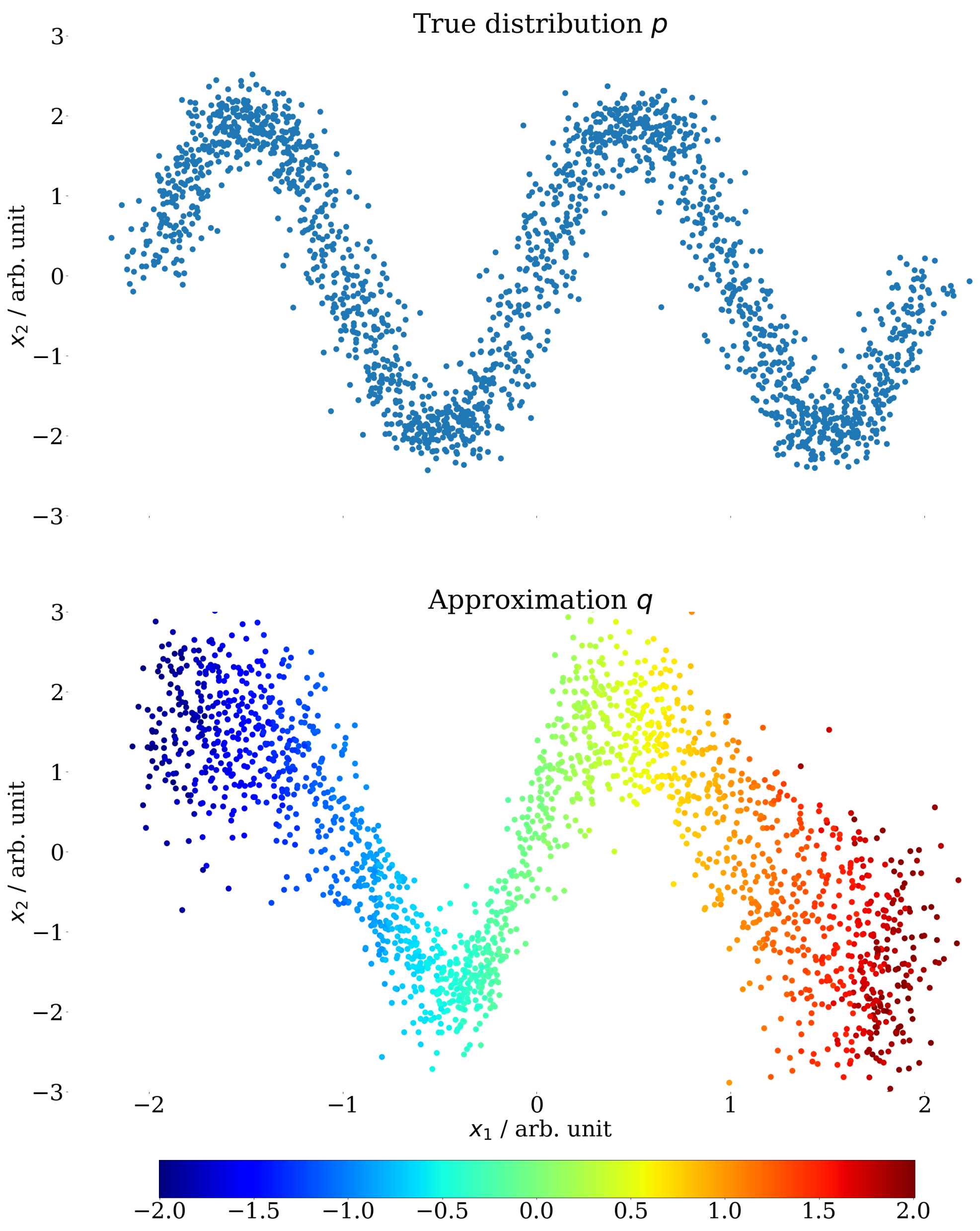


Figure 3.4: Conditional sampling in the toy-model with a 2-dimensional Gaussian with sinusoidal mean. The full marginalized distribution is plotted, with each color c in the colormap corresponding to samples from the conditional $q(\boldsymbol{x}|x_1 = c)$. Best viewed in color.

Chapter 4

Method

In this chapter we present our method for learning conditional probability distributions with normalizing flows in the context of finding solutions for inverse problems. The task of the model is to find the distribution of a "clean" or uncorrupted unobserved variable $\boldsymbol{x}$ using only a corrupted observation $\boldsymbol{y}$ of it. We assume that we are supplied with *paired* data of images $\boldsymbol{x} \in \mathbb{R}^{d \times d \times c_1}$ and corresponding images $\boldsymbol{y} \in \mathbb{R}^{m \times m \times c_2}$ with $d \times d \times c_1 \geq m \times m \times c_2$.

We wish to learn the conditional probability densities for three different tasks: superresolution, denoising and colorization. For each of the tasks, we use essentially the exact same architecture, only changing the training dataset. During inference, we need to be able to sample from the distribution given only the corrupted values $\boldsymbol{y}$ as specified above. The dimensionality of the tensor in the flow ($\boldsymbol{x}$ or $\boldsymbol{z}$) is necessarily of the same dimensionality as the uncorrupted samples, that is, of shape $d \times d \times 3$. Depending on the task, the shape of $\boldsymbol{y}$ is given in Table 4.1.

Table 4.1: Forward processes of the inverse problems. In the experiments, we use images with $h = w = d = 128$.

Task	dim($\boldsymbol{x}$)	dim($\boldsymbol{y}$)	Notes on forward process
Superresolution	$d \times d \times 3$	$d/4 \times d/4 \times 3$	4-times, Lanczos downsampling
Denoising	$d \times d \times 3$	$d \times d \times 3$	zero-mean additive Gaussian $\sigma \in [0.05, 0.3]$
Colorization	$d \times d \times 3$	$d \times d \times 1$	PIL-grayscale

The normalizing flow is trained using the standard maximum likelihood approach by minimizing the conditional variant of Eq. 2.18 (that is, the negative conditional log-likelihood) with respect to the parameters $\boldsymbol{\theta}$ of the flow-defining function f

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \; -\frac{1}{N}\sum_{i=1}^{N} \log p_{\boldsymbol{z}}\left(f^{-1}_{\boldsymbol{y}_i;\boldsymbol{\theta}}(\boldsymbol{x}_i)\right) + \log\left|J_{f^{-1}_{\boldsymbol{y}_i;\boldsymbol{\theta}}}(\boldsymbol{x}_i)\right|. \tag{4.1}$$

We build upon the RealNVP [16] and Glow [38] -models with the use of non-volume preserving coupling layers and invertible 1×1 convolutions. We do not, however, implement the so-called actnorm-layers used in the original Glow-architecture. There are also some differences in the reshaping operations employed in the flow. We also employ somewhat more complex U-Net-like autoencoders [56] in the scale and bias-term computing functions in the coupling layers. In their place Glow has a few convolutional layers with non-linearities in between. The most striking difference between our model and Glow is the conditioning information preprocessing network, which is discussed later. Ardizzone *et al.* employ a similar preprocessing network [3], which however, is pretrained (a truncated VGG-network), while ours is trained in conjuction with the flow.

4.1 Flow structure

Closely following the methodology of [38] we employ a *multi-scale* architecture consisting of blocks operating at different resolutions. Each block corresponds to an invertible function and stacking such blocks corresponds to composing several invertible functions into one. Since the function defining the flow is invertible, the total number of elements in the tensor cannot change during the flow and hence, changing the resolution comes with a corresponding change in the number of channels. To reduce computational complexity, we split the tensor into two at each resolution and pass one half directly to the end-result of the flow, as if there was a skip-connection. We visualize the flow in Fig. 4.1. Each block is composed of a set of other composable functions which we introduce next.

4.1.1 Coupling layers

We employ coupling layers following the ideas of [16]. The coupling layer splits the tensor in two using two alternative splitting strategies: the so-called checkerboard pattern, or channel-wise splitting. We visualize the split types in Fig. 4.2. One of the part of the tensor resulting from the split is used to compute shifts and biases for the other as defined in Eqs. 2.29 and 2.30. The shift and bias terms are computed by a single, fully convolutional but

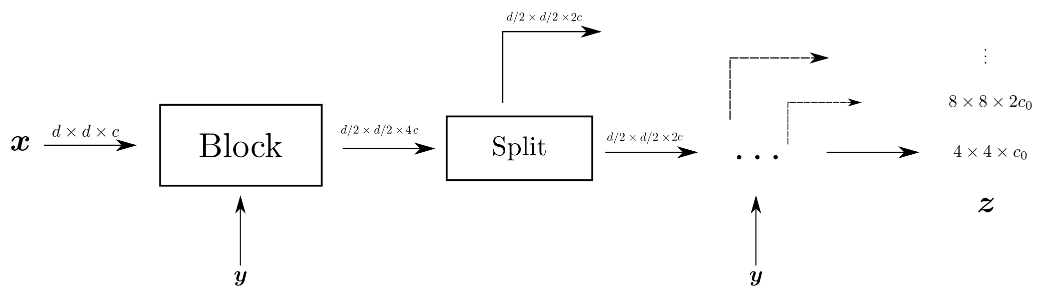


Figure 4.1: Flow composed of blocks and splits corresponding to $\boldsymbol{z} = f_{\boldsymbol{y}}^{-1}(\boldsymbol{x})$. Each applies an invertible transformation conditioned with $\boldsymbol{y}$ and reshapes the tensor. Finally, the tensor is split in two along the channel-axis.

otherwise unconstrained neural network. The shift term is exponentiated to ensure positiveness for the computation of the log-determinant. We initialize the weights of the last convolutional layer of the coupling network to zeros such that at the beginning of training the network performs the identity transformation. We visualize the coupling layer in Fig. 4.3.

4.1.2 1×1 invertible convolutions

We apply the idea of 1×1 invertible convolutions of the Glow-architecture to our normalizing flow. Invertibility implies that the convolution operation cannot change the number of channels in the input tensor and hence the convolution kernel is merely a $c \times c$ (invertible) matrix K. The convolution corresponds to multiplying each $d \times d$ $1 \times 1 \times c$ tensor with the convolution kernel. The inverse is naturally given by repeating the same operation but using the inverse matrix K^{-1}. The determinant of the Jacobian is given by the determinant of the kernel-matrix K. However, since the matrix is used to multiply each of the $d \times d$ vectors independently, the total determinant is the product of $d \times d$ such determinants. We do not constrain the values of K, since the computation of the determinant is only $\mathcal{O}(c^3)$, which is comparable to the convolution operations in the coupling layer neural networks which are $\mathcal{O}(d^2c^2)$ with usually $c < d^2$. It is possible, however, to parametrize K using a LU-decomposition to achieve $\mathcal{O}(c)$ time-complexity for the determinant [38].

4.1.3 Downsampling

We support two invertible means for changing the resolution of the tensor over the invertible transformation defining the flow. The simpler method is

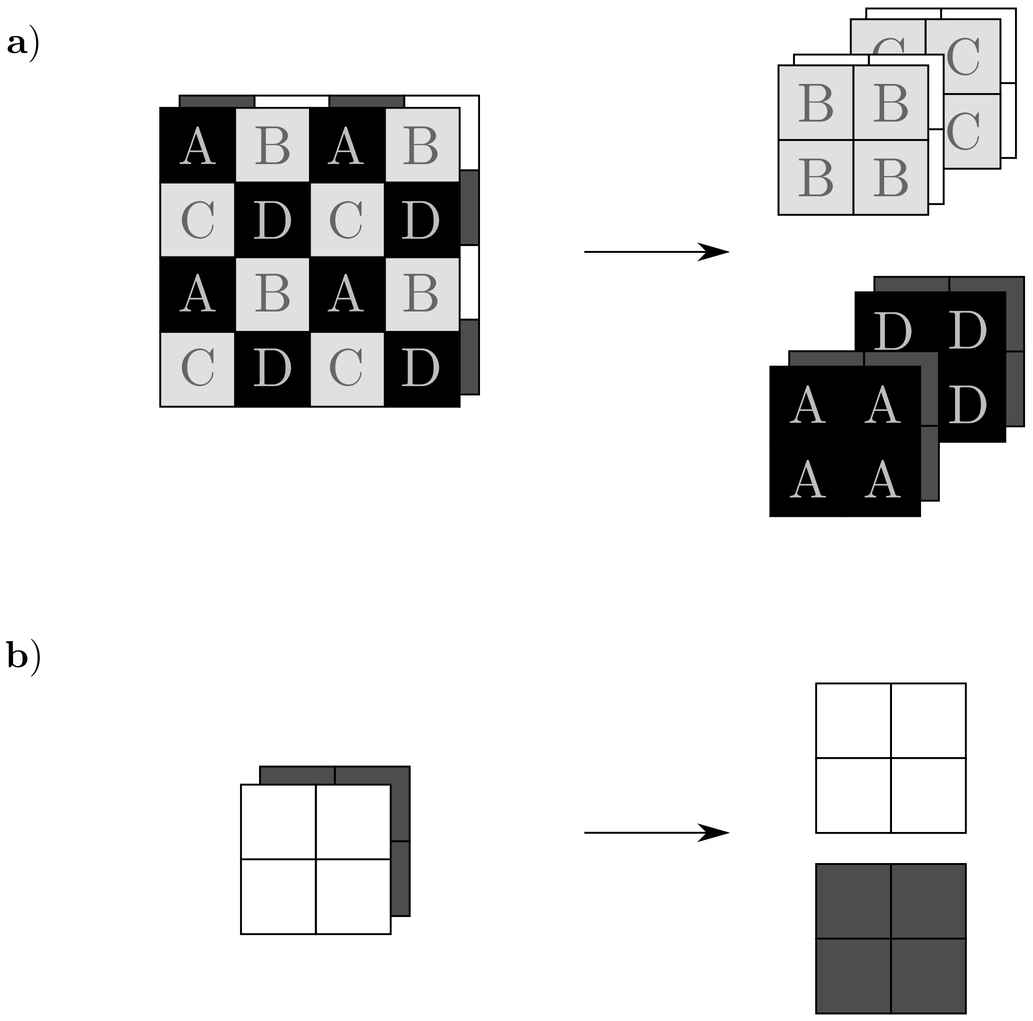


Figure 4.2: Split types for partitioning $\boldsymbol{x}$ for RealNVP layers. **a**) Checkerboard with concatenation. **b**) Split along the channel-dimension.

essentially a mere reshape. We reshape each $2 \times 2 \times 1$ neighborhood of the tensor into a $1 \times 1 \times 4$ tensor. The operation is repeated for each channel and the resulting c $1 \times 1 \times 4$ tensors that are concatenated along the channel dimension producing a tensor with half the width and height and four times the channels. The other methods involves Haar-wavelet downsampling, following the approach of [3]. The wavelet transform decomposes all 2×2 neighborhoods into their mean, horizontal, vertical and diagonal differences. This operation corresponds to using the first downsampling method and multiplying the result with an invertible 4×4 matrix as demonstrated in Fig.

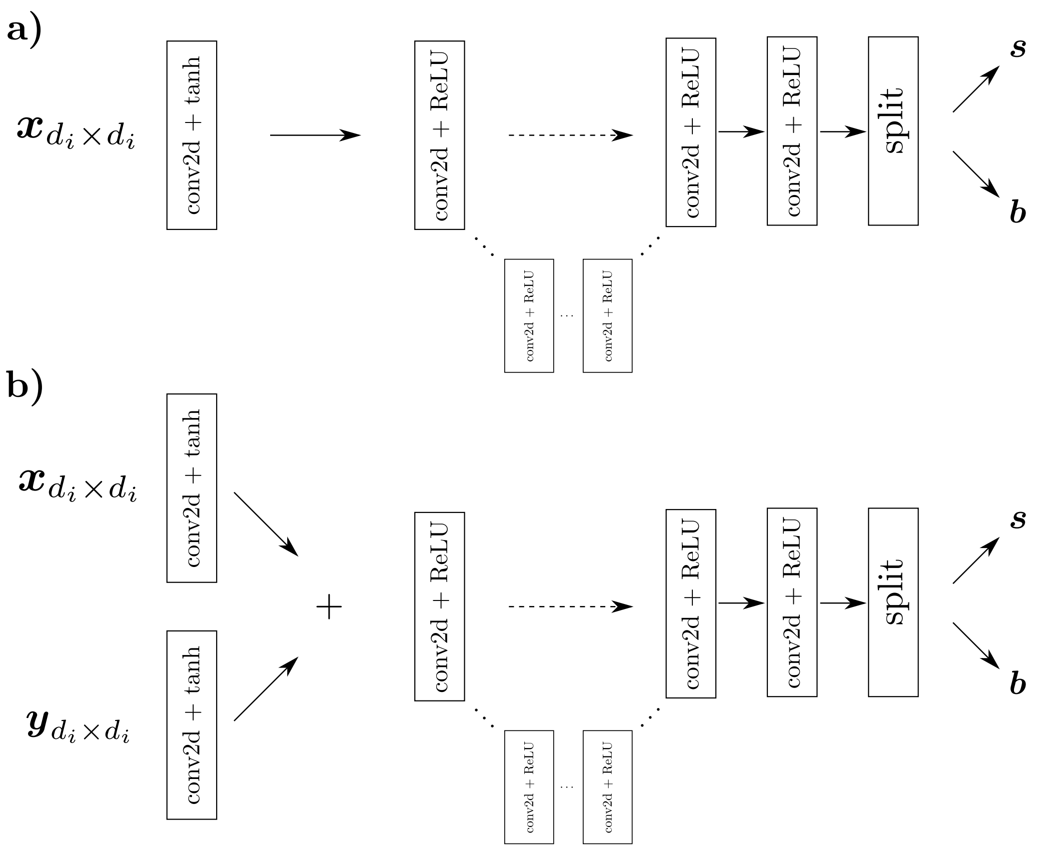


Figure 4.3: Convolutional neural network employed by the RealNVP-type coupling layer. d_i denotes the resolution of the tensors $\boldsymbol{x}$ and $\boldsymbol{y}$ at the ith level of the flow. We use a variable depth encoder–decoder structure with skip-connections in the middle of the coupling layer. The outputs of the network are the scale tensor $\boldsymbol{s}$ and the bias tensor $\boldsymbol{b}$. The scale-terms are exponentiated to ensure non-negativity for the computation of the log-determinant of the Jacobian. **a**) Unconditional variant. **b**) Coupling layer conditioned with $\boldsymbol{y}$.

4.4. The first method is essentially a permutation and hence has unit determinant. The matrix of the second method is orthonormal and hence also has unit determinant. Thus both the methods have vanishing log-Jacobian determinants.

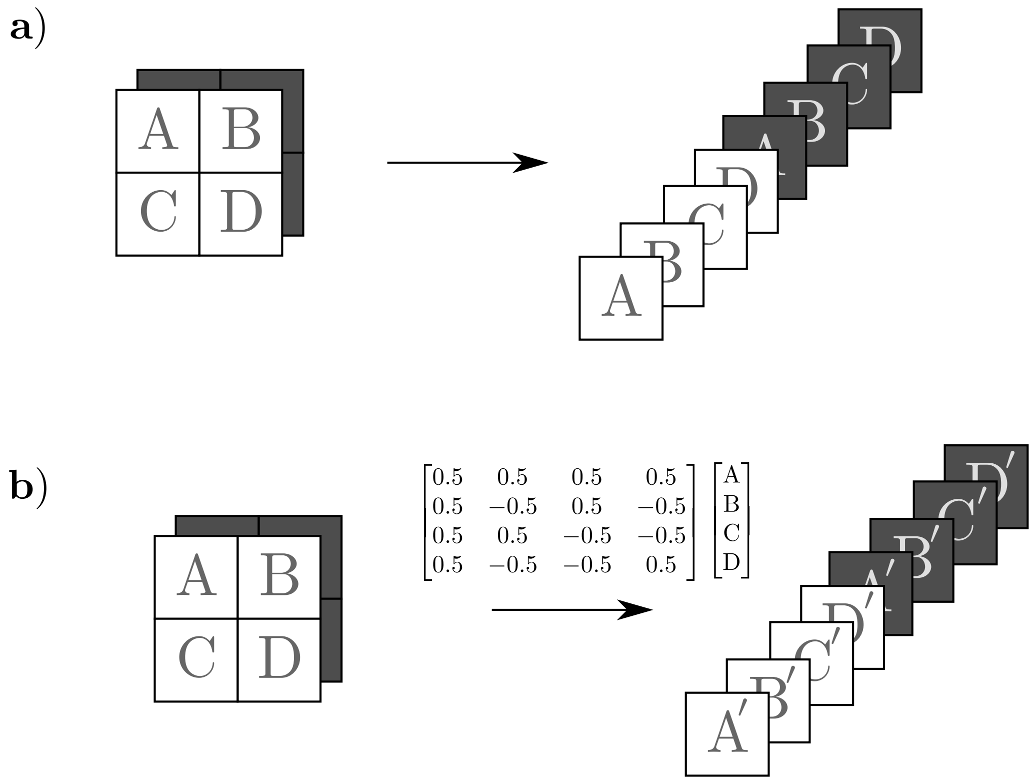


Figure 4.4: Downscaling types. **a**) Simple reshape. **b**) Haar-wavelet downsampling. The simple reshaping is essentially the same operation but with an identity matrix. Both types have unity Jacobian determinant.

4.1.4 Dequantization

Images are often stored on disk using 24-bit RGB values (8-bit unsigned integers per channel) per pixel. The flow-model, however, operates on continuous distributions and hence the images need to be dequantized to avoid arbitrarily high likelihoods [66]. Using Jensen's inequality and a change of variables $\boldsymbol{x}^{'} = \boldsymbol{x} + \boldsymbol{u}$, where $\boldsymbol{u} \sim \text{Uniform}(0, 1)$ we can show that

$$\begin{aligned}\mathbb{E}_{p_{\text{data}}(\boldsymbol{x}')}\left[\log p_{\text{model}}(\boldsymbol{x}^{'})\right] &= \sum_{\boldsymbol{x}} P(\boldsymbol{x}) \int\limits_{\boldsymbol{u}\in U} p(\boldsymbol{u}) \log p(\boldsymbol{x}+\boldsymbol{u}) \mathrm{d}\boldsymbol{u} \\ &\leq \sum_{\boldsymbol{x}} P(\boldsymbol{x}) \log \int\limits_{\boldsymbol{u}\in U} p_{\text{model}}(\boldsymbol{x}+\boldsymbol{u}) \mathrm{d}\boldsymbol{u} \\ &= \mathbb{E}_{P_{\text{data}}(\boldsymbol{x})}\left[\log \mathbb{E}_{p(\boldsymbol{u})}\left[p_{\text{model}}(\boldsymbol{x}+\boldsymbol{u})\right]\right] \\ &= \mathbb{E}_{P_{\text{data}}(\boldsymbol{x})}\left[\log P_{\text{model}}(\boldsymbol{x})\right], \end{aligned} \tag{4.2}$$

where P denotes the respective discrete density functions and $P_{\text{model}}(\boldsymbol{x}) \equiv \mathbb{E}_{p(\boldsymbol{u})}\left[p_{\text{model}}(\boldsymbol{x}+\boldsymbol{u})\right]$. By definition $p(\boldsymbol{u}) = 1 \quad \forall \boldsymbol{u} \in U$ and hence it vanishes in the equations. The inequality shows that maximizing the likelihood of the properly dequantized model maximizes a lower bound for the respective discrete model. We note that using the uniform distribution is not necessary and more expressive distributions can also be employed in the spirit of importance sampling [26].

As a preprocessing step we transform (normalize) the RGB-values from $[0, 255]$ to $[0, 1]$ by dividing with 256. Alternatively, if we choose some other discretization of colors (e.g. 5-bit, 32 possible values for each channels), we normalize using that value. This operation comes with a constant offset in the log-likelihood. In the standard case of 8-bit colors the change of variable formula Eq. 2.17 yields for elementwise division with 256 the log-Jacobian determinant

$$\log\left|J_{f_{[0,256]\to[0,1]}}(\boldsymbol{x})\right| = \log\left(\frac{1}{256}\right)^{D} = -D\log(256), \tag{4.3}$$

or eight bits-per-dimension. If the discretization level is something else, the offset changes correspondingly. Generally, training losses in likelihood-based models trained with image data are given in the dequantized, non-normalized space and hence adding the offset is important for allowing meaningful comparison.

4.2 Conditioning

We convert the flow model into a conditional flow model by feeding in the corrupted observation tensor $\boldsymbol{y}$ into the model via the coupling layers. The flow can be conditioned naively with simply using $\boldsymbol{y}$ as is. However, we find it beneficial to preprocess the conditioning information with a separate neural network following the U-Net [56] architecture. Ardizzone *et al.* take a similar approach in *colorization* of natural images with normalizing flows [3]. However, they use a pre-trained network for the preprocessing task. The preprocessing network can operate in two modes: one where only the final output of the network is used to condition the flow (downsampled for compatibility for lower-resolution layers) and other where the output is used at the highest resolution of the flow but the conditioning information for the lower flow-resolutions is obtained from the internal activations of the U-Net at respective resolutions. The full flow network together with the preprocessing U-Net are presented in Fig. 4.5.

4.3 Fully Regressive Variant

The invertible function $f(\cdot)$ (possibly parametrized with $\boldsymbol{y}$) is designed to operate on probability densities via the change of variables formula (Eq. 2.17). However, nothing prevents us from applying the function directly to map the corrupted observations into clean images. The mapping is naturally one-to-one (and still has a tractable Jacobian determinant) and hence is much more constrained than those parametrized by more standard neural network tasked with denoising, superresolution or colorization. Training the flow-model with the supervised task of finding

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, \mathbb{E}_{p_{\text{data}}(\boldsymbol{x},\boldsymbol{y})} \left[\ell(\boldsymbol{x}, f_{\boldsymbol{\theta}}(\boldsymbol{y}))\right], \tag{4.4}$$

where ℓ is a loss function such as the L_2-loss yields a baseline result and potentially some insight on how the network operates. We do not expect, however, the network to perform well in the task due to the aforementioned constraints. Due to the function being one-to-one, in the case of superresolution and colorization where the shape of the image changes, the input-tensor is naively (and deterministically) upscaled to the correct dimensions before the application of $f(\cdot)$.

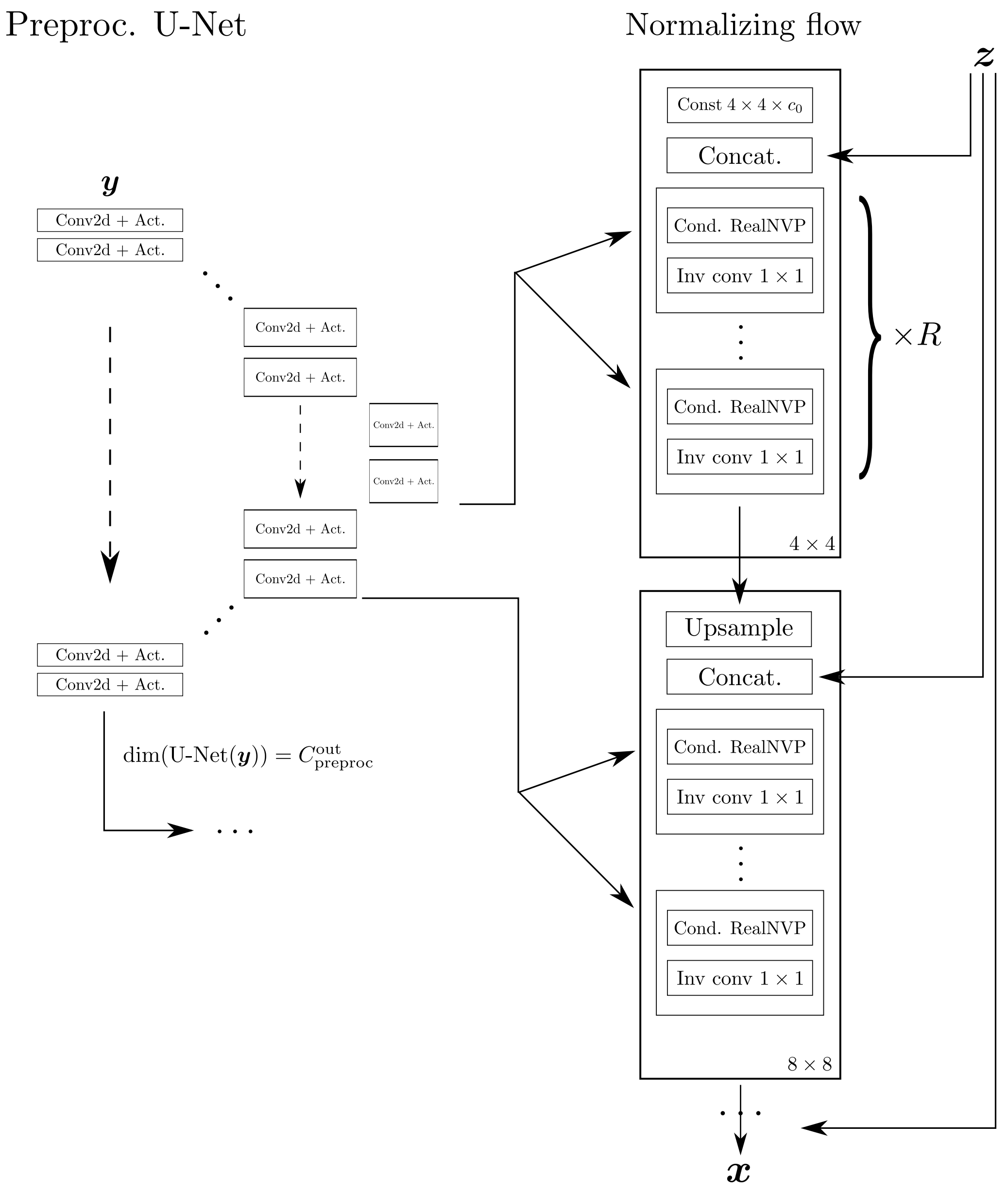


Figure 4.5: Normalizing flow transforming $\boldsymbol{z} \sim N(\mathbf{0}, \boldsymbol{I})$ into $\boldsymbol{x}$ conditioned on $\boldsymbol{y}$. The U-Net preprocesses the conditioning information which is used in the computation of the scale–bias terms in the RealNVP coupling layers. Each resolution of the flow has R separate RealNVP blocks that each can employ either the checkerboard or the channelwise -splitting scheme. Note that there are no non-linear activation functions in the flow itself. While most activation functions are invertible, they tend to introduce instability in the training process. The figure depicts the forward-operation of the flow. In inverse mode, the concatenations are replaced with splits, the convolutions with convolutions using the inverse of the convolution kernel and the RealNVP-layers with their inverses.

Chapter 5

Results

We test and validate the proposed method first by training fully regressive reference networks (that is, using training loss expressed by Eq. 4.4) and afterwards by training respective flow-models using the maximum likelihood objective. We mostly resort to comparing the results *qualitatively* by visual inspection due to the inherent ambiguity of the answers of the inverse problems. Using, for example, a pixel-wise L_2-loss makes little sense as the properties of a good answer are mostly perceptual. Our definition of visual quality is twofold. We expect the results to be plausible realizations of an inverse process, such that the sample, when corrupted again with the forward process, resemble the original corrupted image. Additionally, we hope to see variance in the samples, subject to the first visual quality criterion. The variance is expected to increase for more ill-posed problems or samples. In training, we use the FFHQ dataset [36], downscaled to 128×128 resolution. We train each network using 90% of the dataset and leave the remaining fraction for the testset. All the the experiments are performed with images that are not shown to the models during training (i.e. they are from the testset). We implement the normalizing flow and the accompanying neural networks with TensorFlow [1]. We also make use of the python library dnnlib, courtesy of NVIDIA[1]. We use the Adam-optimizer [37] with default parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\varepsilon = 1 \times 10^{-8}$. In the experiments, we use an exponetially decaying learning rate with respect to the epoch T of the form

$$\mathrm{LR}(T) = \mathrm{LR}_{\mathrm{st}} + \mathrm{LR}_{\mathrm{dec}} \exp(-T/\tau), \tag{5.1}$$

with values $\mathrm{LR}_{\mathrm{st}} = 1 \times 10^{-6}$, $\mathrm{LR}_{\mathrm{st}} = 3 \times 10^{-4}$ and $\tau = 4$. The batch size in our experiments is 5.

[1]Under CC BY-NC 4.0, available at `https://github.com/NVlabs/stylegan`

Experiments Rather than directly measuring the quality of the results, we pay significant effort on understanding the internal structure of the family of models. We study which parts of the flow contribute to producing the eventual output image and which parts serve towards the Gaussianization goal. We also study how well the model performs on its actual optimization task of Gaussianizing the data distribution. For the rest of this chapter we adopt nomenclature where we denote the flow layers with the lowest resolution as the bottom layers of the flow and the high-resolution layers as the top layers. Alternatively, we may call the bottom layers the latent-like layers and the top layers the image-like layers as the parts of the flow reaching the bottom layers have been processed more and have effectively lost their natural image look.

We start by quantifying the expressive power of the flow by employing it as a regressor, as introduced in Section 4.3. Standard likelihood maximization does not explicitly ask the model to produce pictures close to the ground truth images. Hence it is meaningful to ask, how well the flow model manages to restore corrupted images *if that is the primary optimization target.* We find that even direct optimization is not enough and hence it is perhaps too optimistic to expect that likelihood-maximization would yield any better results. This leads to the conclusion that additional model capacity is required in the form of an additional preprocessing network. We show that implementing such preprocessing network for the conditioning information is indeed beneficial. Later, we also observe how flexibly the flow model adapts to each of the three different tasks and how successful it is in Gaussianizing the input image distribution. This is interesting, since it is the primary optimization target, when optimizing for data likelihood. Finally, we identify which layers of the flow are the most relevant with respect to the appearance of the output image, both with respect to the conditioning information $\boldsymbol{y}$ and the normal-distributed variable $\boldsymbol{z}$.

5.1 Regressive Model

We train the regressive model using the multi-scale architecture down to 4×4 resolution. Each resolution has $R = 14$ conditional RealNVP-blocks each containing two RealNVP-layers and a 1×1 invertible convolution. An overview of the network parameters is given in Table 5.3 using the notation of Fig. 4.5. Four variants are trained using two different loss functions $L_1(\boldsymbol{a}, \boldsymbol{b}) \equiv |\boldsymbol{a} - \boldsymbol{b}|$ and $L_2(\boldsymbol{a}, \boldsymbol{b}) \equiv |\boldsymbol{a} - \boldsymbol{b}|^2$. The conditioning information is preprocessed either by using the supplementary U-Net and its internal activations or directly feeding the (possibly rescaled) conditioning tensor $\boldsymbol{y}$ into the conditional RealNVP-blocks (an identity transform). Each network is trained for a relatively short period of time, 12 hours. The results are visualized by task in Figs. 5.1, 5.2 and 5.3 for superresolution, denoising and colorization tasks, respectively.

Table 5.1: Regressive models. Preprocessing type refers to how the conditioning information is preprocessed before given to the RealNVP-layers. The fourth variant has a different value for R (number of RealNVP/invertible convolution blocks per flow-resolution) due to problems with convergence. The parameter $C^{\text{out}}_{\text{preproc}}$ denotes the number of channels in the output of the preprocessing network.

Model Variant	R	$C^{\text{out}}_{\text{preproc}}$	Preproc. type	Loss
Regr. 1	14	18	U-Net activations	L_2
Regr. 2	14	18	U-Net activations	L_1
Regr. 3	14	3	Identity	L_2
Regr. 4	12	3	Identity	L_1

To no surprise, we find that the function defining the normalizing flow $f(\cdot)$ is not particularly good at directly mapping a corrupted sample into a single clean estimate. It is highly constrained due to the invertibility and determinant constraints. Furthermore the pixel-wise distance-based losses introduce blurriness to the results. Halo-effects caused by the Lanczos downsampling procedure are greatly emphasized in the restored images in the superresolution task, as seen if Fig. 5.1. Interestingly, providing additional representational power in the form of the preprocessing U-Net, does not improve the results in any task. The colorization results (Fig. 5.3) are rather plain due to only a mean or median color being learned as a result of the loss-functions.

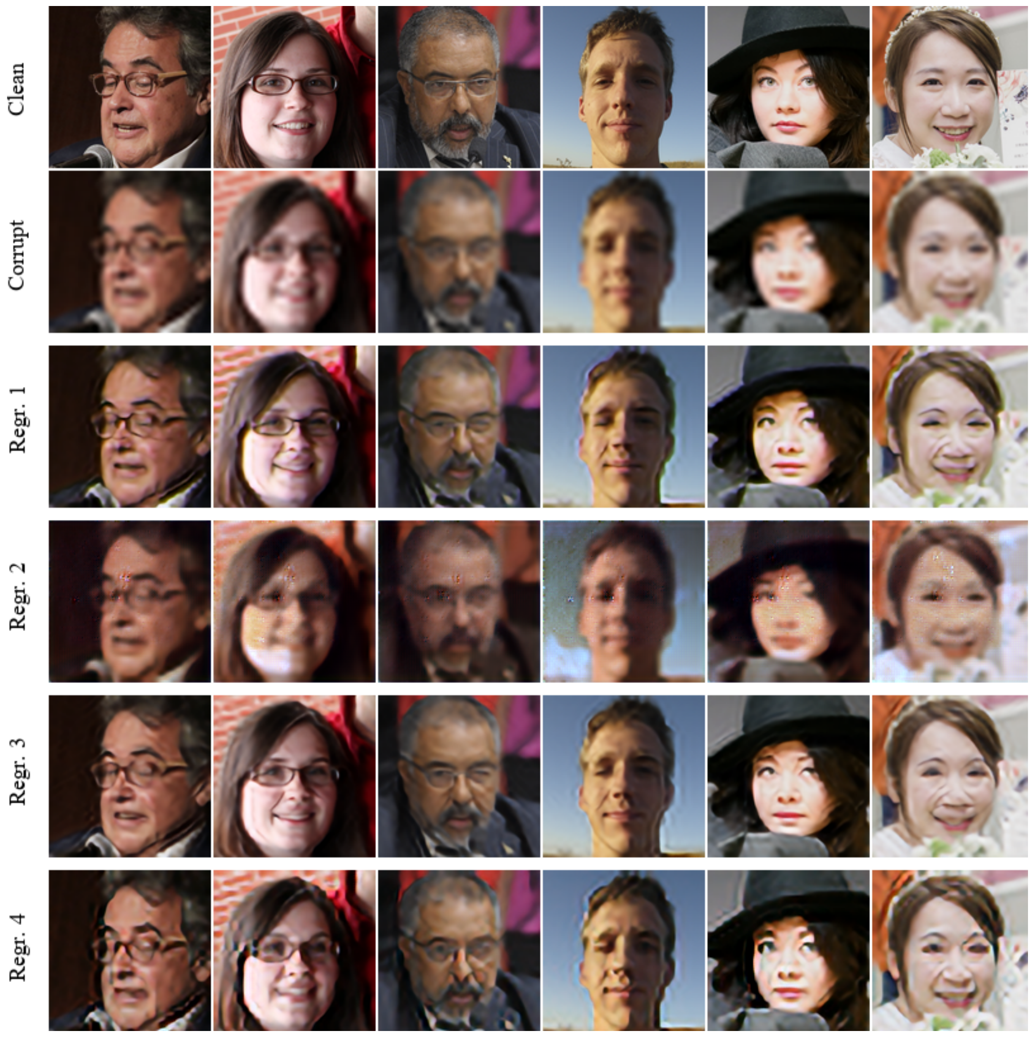


Figure 5.1: Regressive model variants for superresolution. The variants REGR.2 and REGR.4, trained with the L_1 loss seem to be performing worse than the ones employing the L_2 loss. The preprocessing network interestingly has negligible effect on the quality: models REGR.1 and REGR.2 do not yield better results than the remaining two.

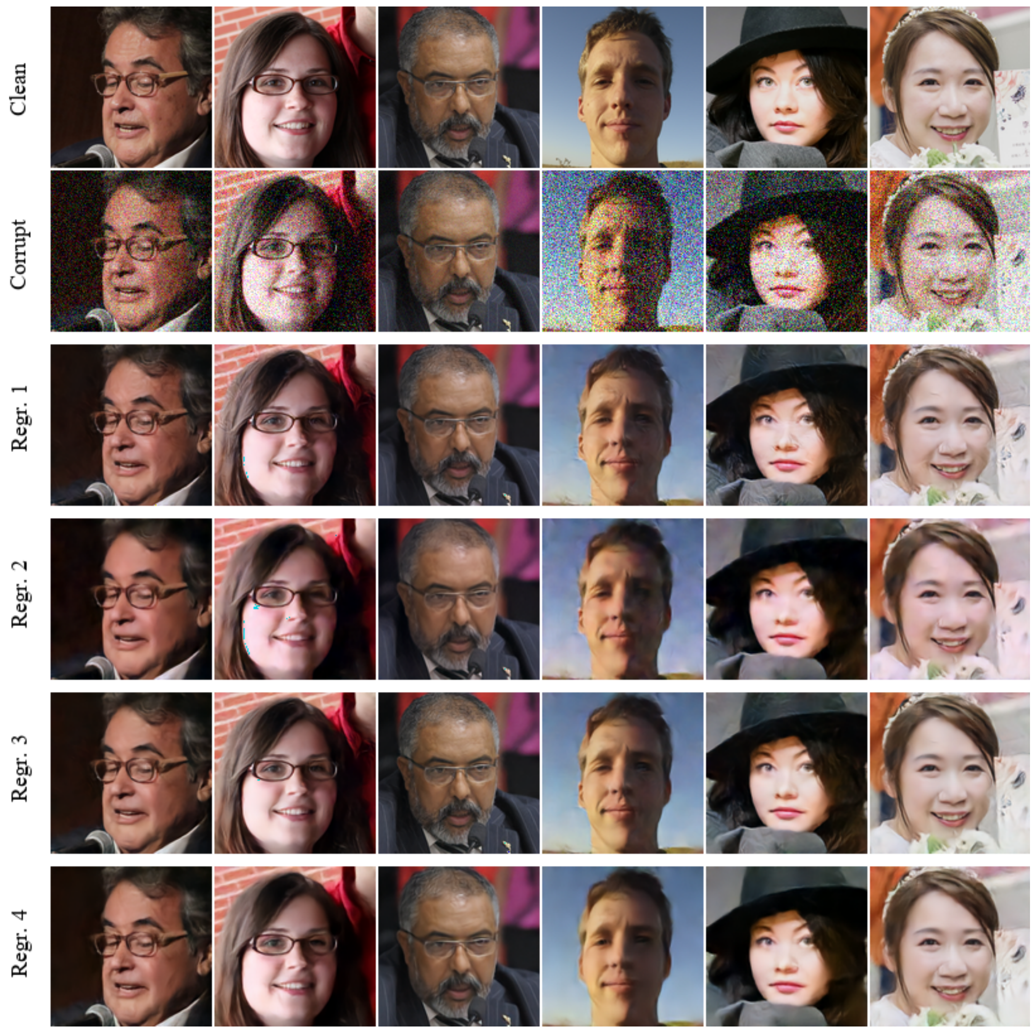


Figure 5.2: Regressive model variants for denoising. Minimizing the L_2 loss yields an average pixel value, which in the case of zero-mean Gaussian is the clean signal. Hence the results look decent with the L_2-loss variants (Regr.1 and Regr.3) seemingly slightly outperforming the other two.

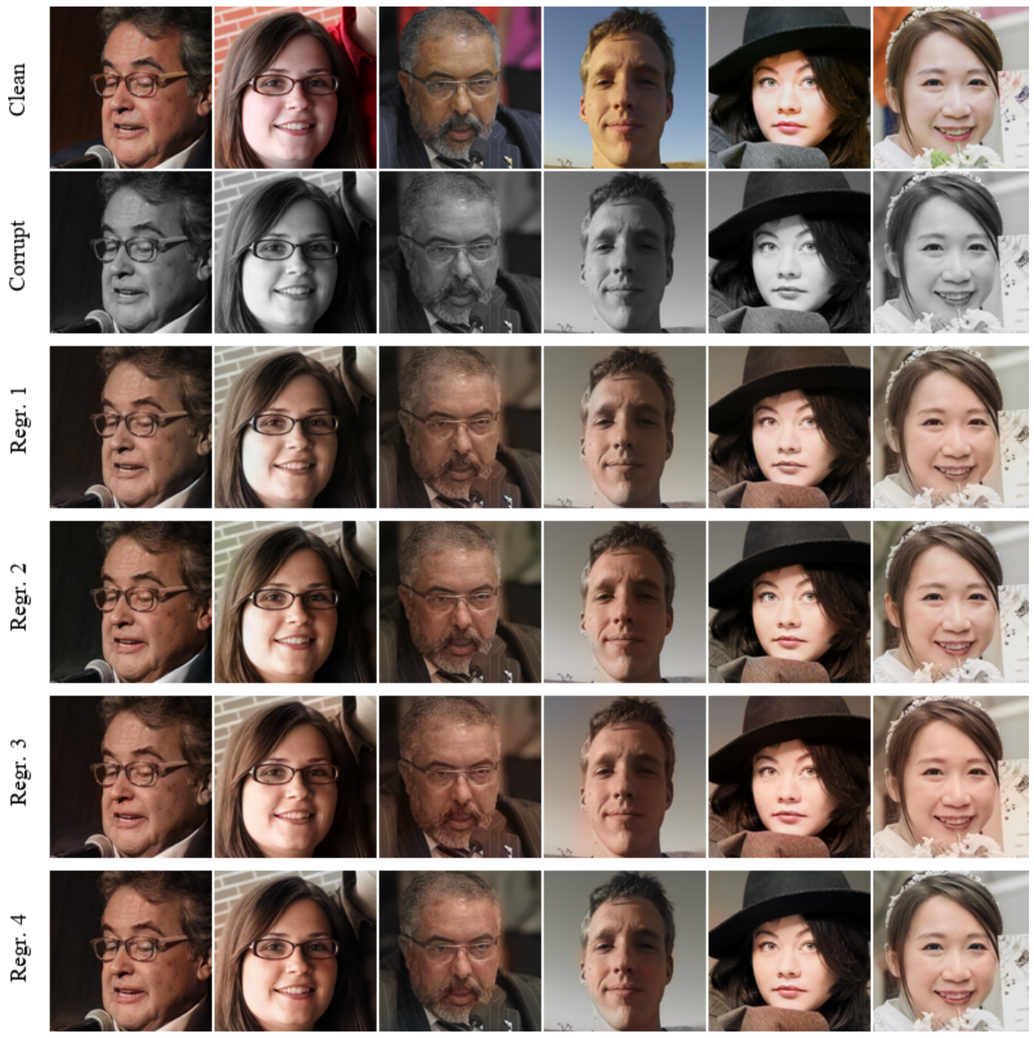


Figure 5.3: Regressive model variants for colorization. While the outputs look reasonable, they lack variance and are closer to some average color scheme, as expected.

5.2 Form of Conditioning

We now turn our attention towards flow-models trained by maximizing the likelihood of the training data under the distribution given by the change of variables formula. While the functional form of the condition info preprocessing network had little visible effect on the regressive models in the previous section, we extend similar comparison here. In addition to sampling from the conditional distribution $p(\boldsymbol{x}|\boldsymbol{y})$ for each of the tasks (Figs. 5.4, 5.6 and 5.8), we also find and estimate for the conditional mean

$$\boldsymbol{x}^* = \mathbb{E}_{\boldsymbol{x} \sim p(\boldsymbol{x}|\boldsymbol{y})} [\boldsymbol{x}] \approx \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{x}_i, \tag{5.2}$$

where $\boldsymbol{x}_i$ are sampled from $p(\boldsymbol{x}|\boldsymbol{y})$. We compare this single estimate with the values acquired from the regressive models in Fig. 5.5. We could alternatively find the *maximum a posteriori* (MAP) estimate via optimization. However, we found in preliminary experiments that the estimate is generally not that great, potentially due to convergence to a poor local minimum in the optimization process. Differences and similarities in the models are specified in Table 5.2. We sample using a reduced (truncated) standard deviation $\sigma = 0.85$ in the superresolution and denoising tasks as it is found to increase the quality of the samples. For colorization we use $\sigma = 1.06$ to increase variance. We find that using a lower sampling temperature rapidly decreases the variance and yields mostly brown images in the colorization task.

Model variants We employ three different conditioning information preprocessing schemes. The most direct way is to give the conditioning tensor as is to the RealNVP-layers (with possible naive up- or downsampling). This is the identity preprocessing transform used in the regressive case, which we will no denote as the DIRECT model variant. Alternatively, we can preprocess the conditioning information with a separate U-Net, again like in the regressive case. We can choose to either use the output of the U-Net and downscale the result for the lower-resolution flow-layers (denoted as U-NET DOWNSAMPLED or U-NET DS.) or use the internal activations of the U-Net at the respective resolutions (U-NET ACTIVATIONS or U-NET ACT.). We will continue to use the aforementioned names for the conditioning types throughout the rest of this chapter.

Observations We find that the preprocessing network is beneficial in terms of visual quality of the results. However, used only in conjunction with downscaling of the output of the U-Net, the results remain poor in the superresolution task (Fig. 5.4). The difference between U-NET DOWNSAMPLED and U-NET ACTIVATIONS is not as stark in the denoising task (Fig. 5.6), but the latter seems to reproduce high-frequency details such as hair or teeth marginally better. Using the preprocessing network also yields a significant improvement in the loss, especially in the colorization task, as seen on Table 5.2. The preprocessing network seems to improve the quality of individual samples but also the quality of the conditional mean as illustrated by Figures 5.5 and 5.7. The mean in the colorization task is perhaps not as interesting but is given in Figure 5.9 for the sake of completeness.

Table 5.2: Model conditioning variants. The minimum loss value is an expectation over the validation set. The loss is the negative log-likelihood and has units bits-per-dimension, smaller is better.

Model	R	$C_{\text{preproc}}^{\text{out}}$	Preproc. type	Task type	Loss
DIRECT	14	3	Identity	Superresolution	2.882
U-NET DOWNSAMPLED	14	18	U-Net ds.	Superresolution	2.866
U-NET ACTIVATIONS	14	18	U-Net act.	Superresolution	2.867
DIRECT	14	3	Identity	Denoising	2.911
U-NET DOWNSAMPLED	14	18	U-Net ds.	Denoising	2.902
U-NET ACTIVATIONS	14	18	U-Net act.	Denoising	2.896
DIRECT	14	3	Identity	Colorization	2.114
U-NET DOWNSAMPLED	14	18	U-Net ds.	Colorization	2.005
U-NET ACTIVATIONS	14	18	U-Net act.	Colorization	2.005

Judging the quality of the colorized images is somewhat difficult even quantitatively. None of the models clearly produces colorized images of inferior quality. All the model types have a tendency of recognizing and colorizing the face in the image plausibly. However, they all struggle with the context and continuity of the background. For example, the sky in the background can change color abruptly, if it is not a single continuous area in the image plane. This is highlighted in Figure 5.8.

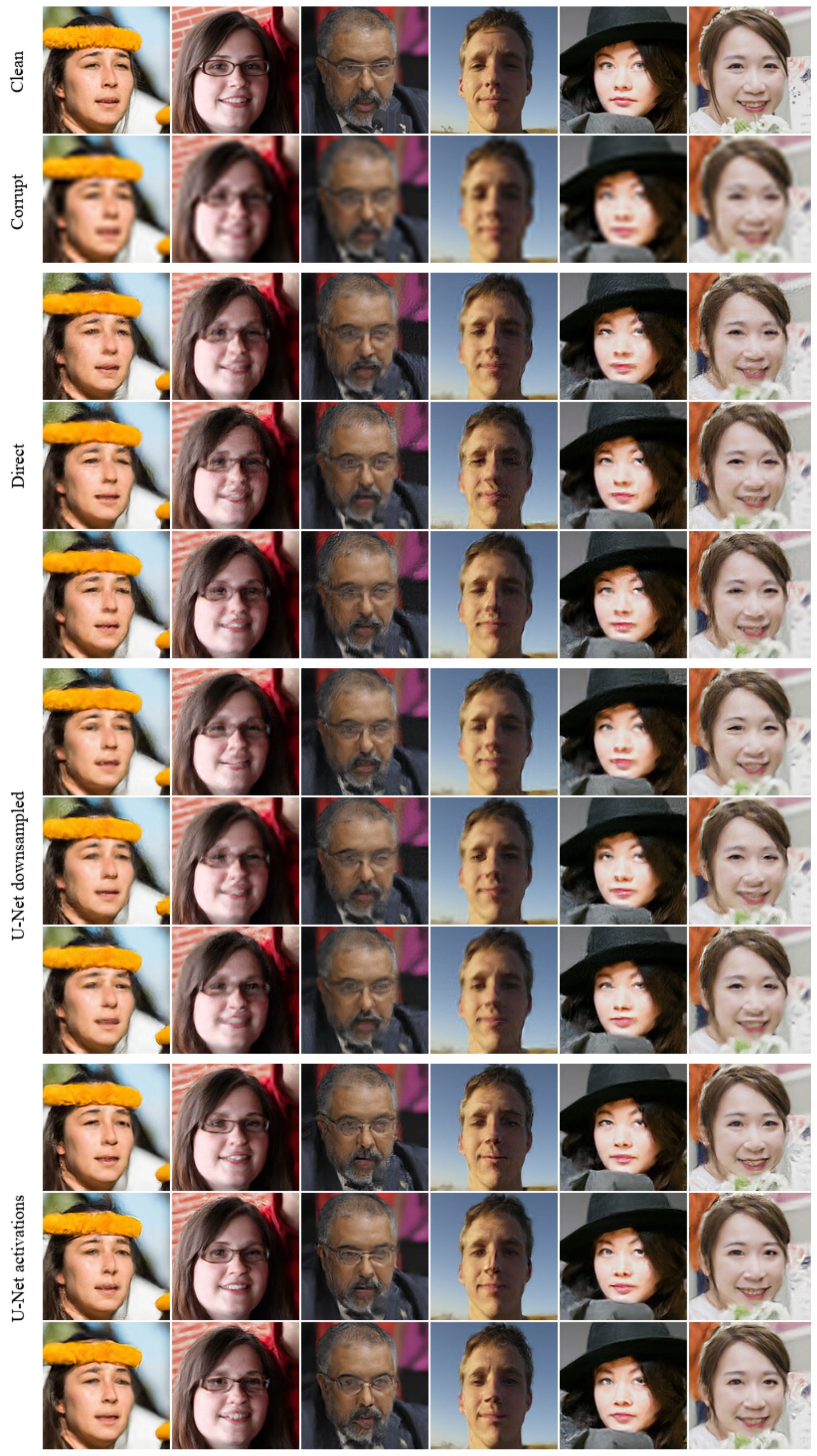


Figure 5.4: Effect of the form of conditioning on the superresolution task. Three samples for each low-resolution input image (columns) are given for each preprocessing technique Direct, U-Net Downsampled and U-Net Activations (grouped rows).

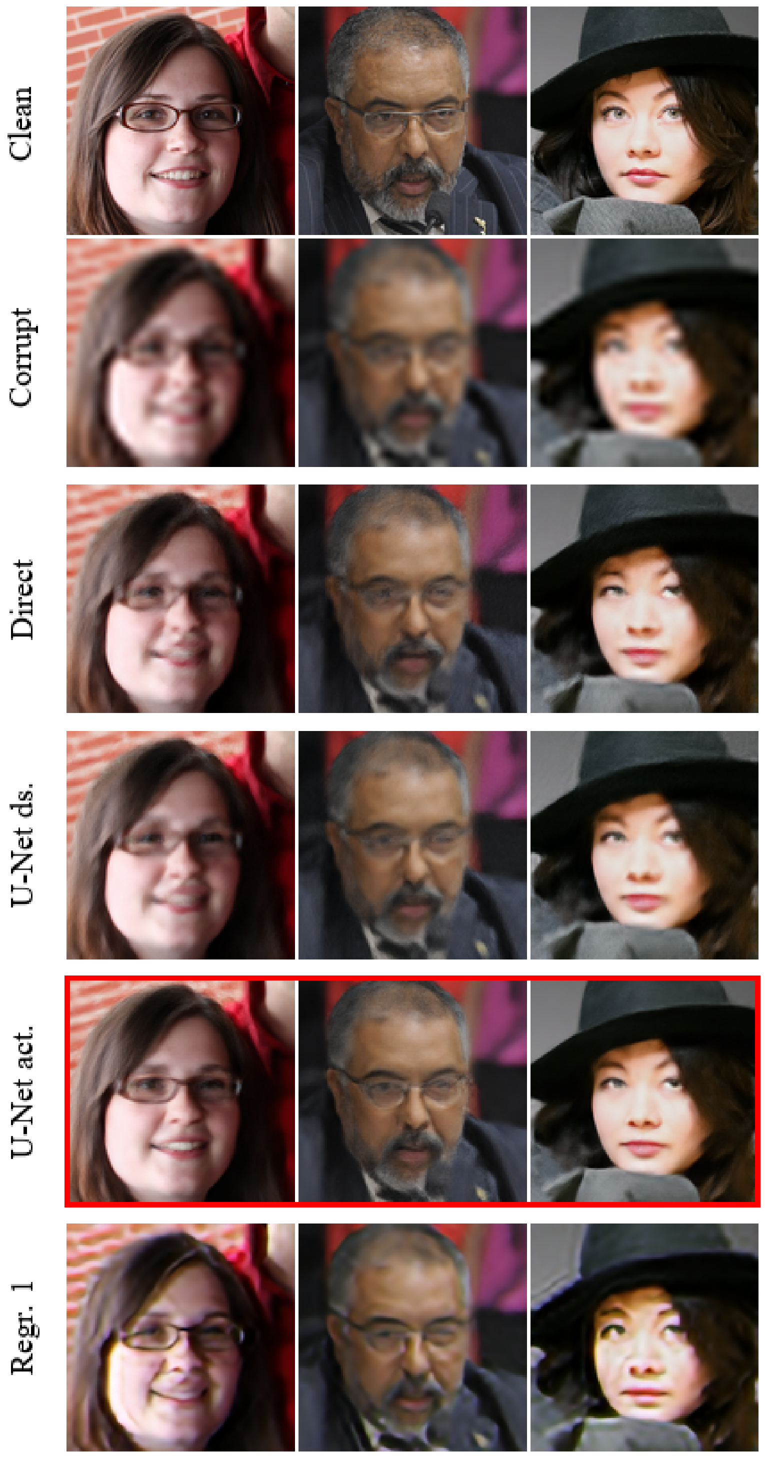


Figure 5.5: Effect of the form of conditioning on the superresolution task. Mean over 5 samples is given for each low-resolution input image (columns). The reference regressive model Regr. 1 (last row) only produces a single answer and hence taking the mean is not useful. The mean results of the U-Net Activations (highlighted with red) model are clearly the sharpest.

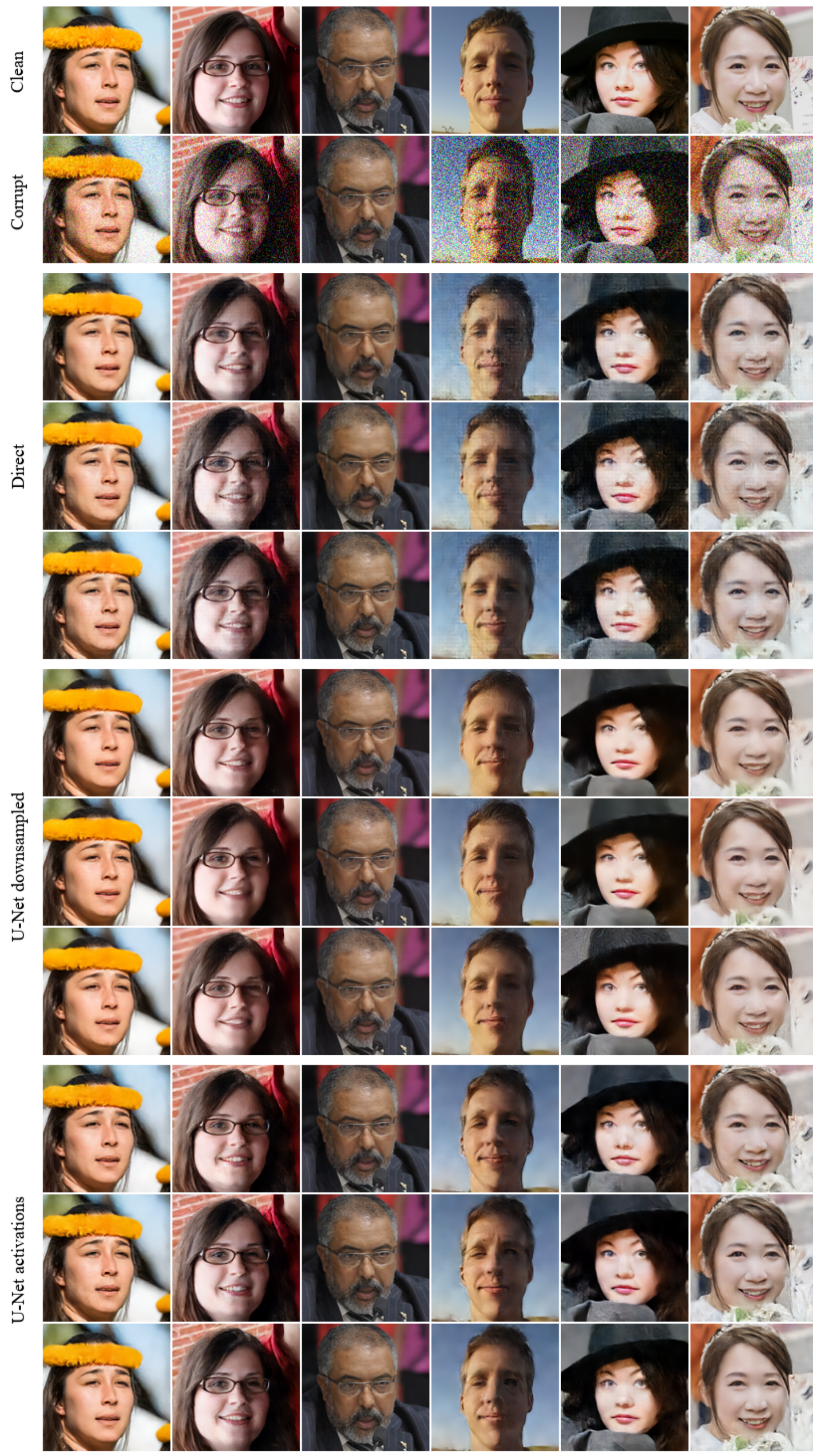


Figure 5.6: Effect of the form of conditioning on the denoising task for all three preprocessing techniques Direct, U-Net Downsampled and U-Net Activations (grouped rows).

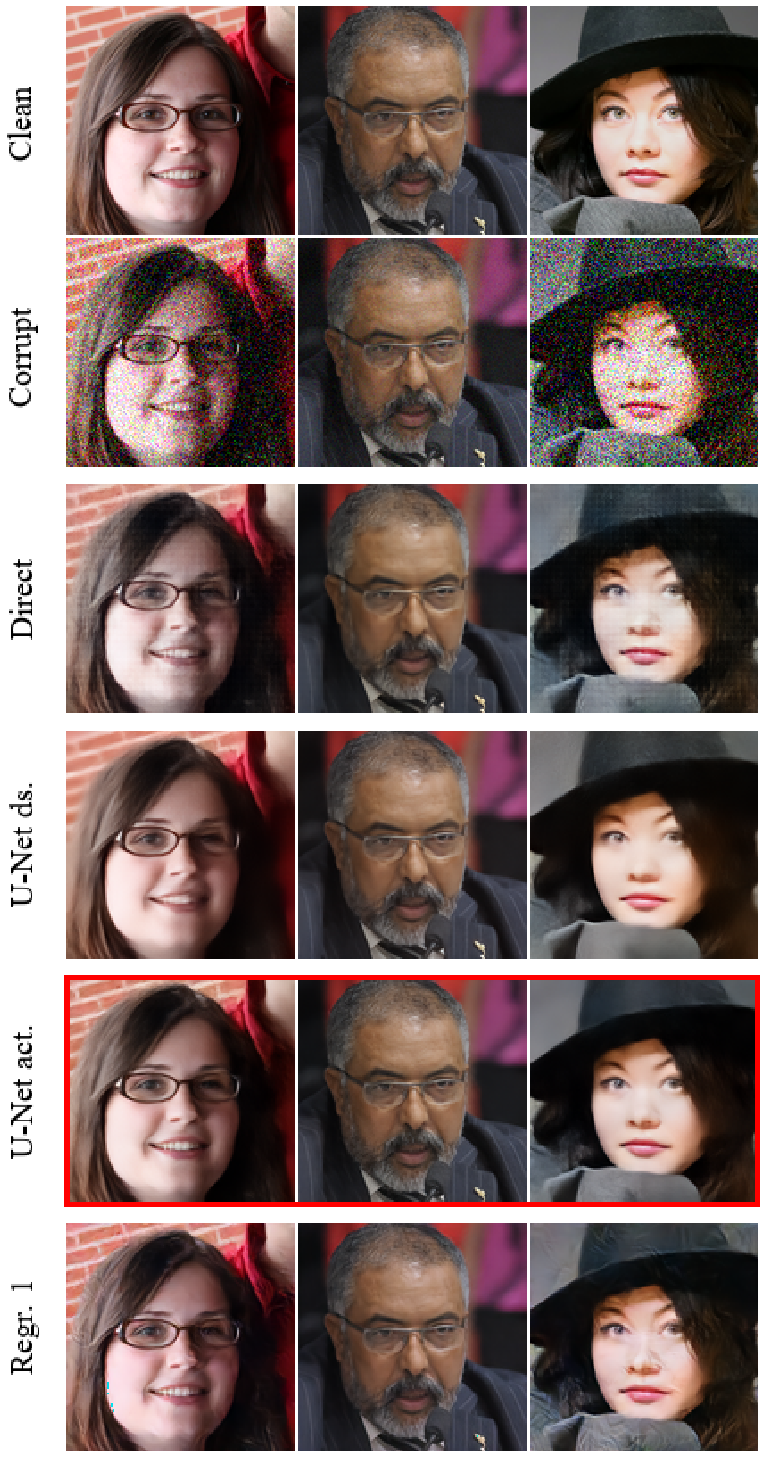


Figure 5.7: Effect of the form of conditioning on the denoising task. Mean over 5 samples is given for each corrupted input image (columns). The reference regressive model Regr. 1 (last row) only produces a single answer and hence taking the mean is not useful. The mean results of the U-Net Activations (highlighted with red) model are clearly the sharpest.

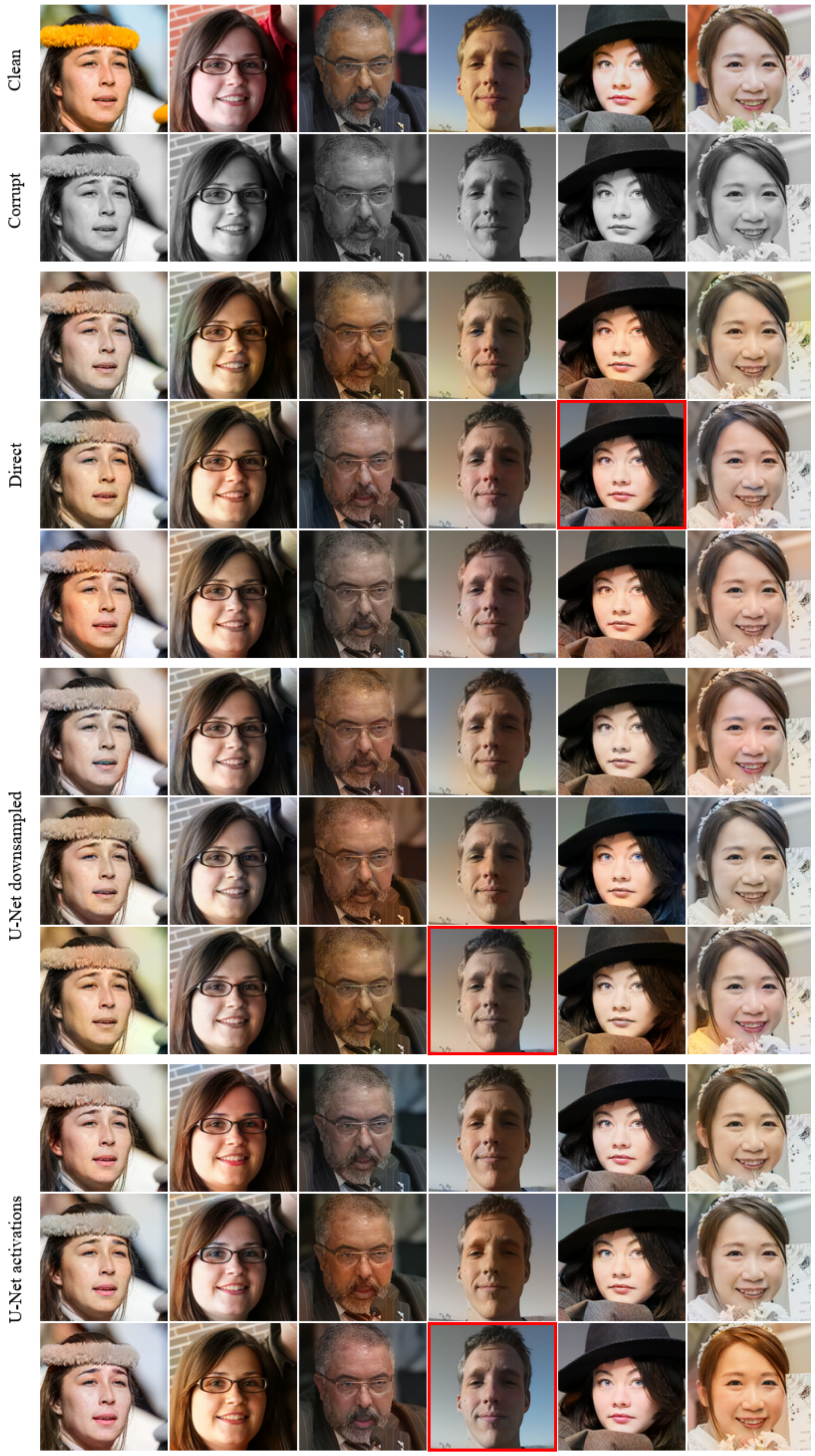


Figure 5.8: Effect of the form of conditioning on the colorization task for all three preprocessing techniques DIRECT, U-NET DOWNSAMPLED and U-NET ACTIVATIONS (grouped rows). All types share a similar failure case (highlighted with red), with discontinuous background coloring.

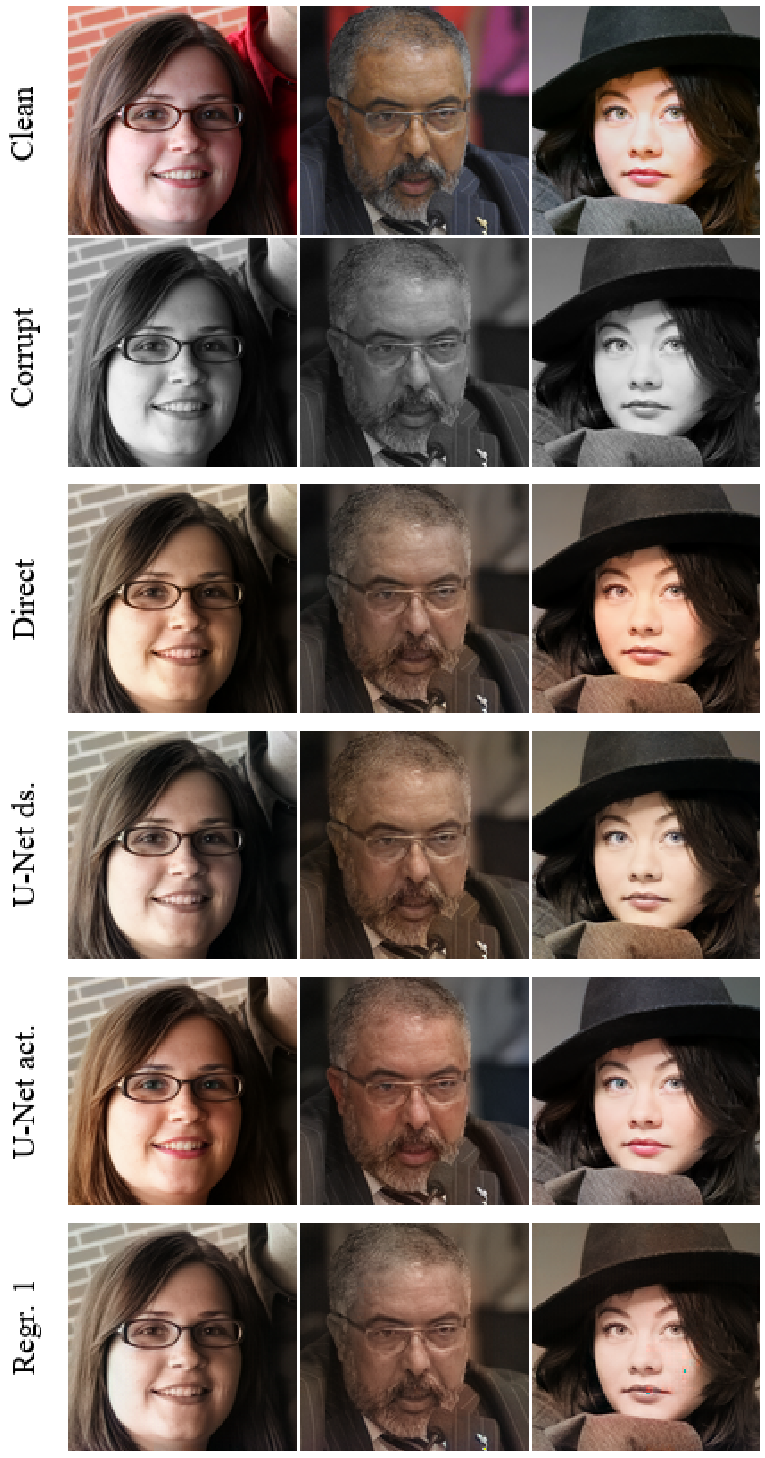


Figure 5.9: Effect of the form of conditioning on the colorization task. Mean over 5 samples is given for each gray-scale input image (columns). The reference regressive model Regr. 1 (last row) only produces a single answer and hence taking the mean is not useful.

5.3 Goodness of Fit

The regressive models explicitly minimize a distance between a ground truth image and the network output. Minimizing the negative log-likelihood, however, only minimizes a divergence between two distributions, but not directly between individual samples. Evaluating the quality of a handful of samples does not directly measure how well the model has learned the data. Vice-versa, successful Gaussianization does not necessarily guarantee high-quality samples, as we are about to see. However, since likelihood-maximization is the primary goal, a more direct question to ask is how well the model learns to Gaussianize the given data. If the Gaussianization is perfect, clean data transformed with the inverse flow $f^{-1}(\cdot)$ is distributed according to a unit Gaussian. Hence, mapping the validation dataset into $\boldsymbol{z}$-space via $\boldsymbol{z} = f^{-1}(\boldsymbol{x})$ and computing the empirical mean-vector and covariance-matrix should yield the zero-vector and a high-dimensional identity-matrix. Consequently, sampling from a Gaussian defined by the aforementioned empirical mean and covariance should yield results no different to sampling from the standard zero-mean, identity-covariance normal distribution. Visualizing an individual image in $\boldsymbol{z}$-space should also yield something resembling white noise since the components of the training target Gaussian are uncorrelated. Figure 5.10 shows that this is indeed the case, the values of $\boldsymbol{z}$ computed for an image from the validation set are effectively indistinguishable from those of actual white noise. We split the $\boldsymbol{z}$ vector into pieces corresponding to resolutions where the respective block of $\boldsymbol{z}$ is split/concatenated in the multi-scale flow.

We transform $N = 2000$ validation set images into the $\boldsymbol{z}$-space and compute blocks of the full covariance matrix. Even though the resolution of the images is relatively low, the size of the full covariance matrix would already be $(128^2 \times 3)^2 = 2.42 \times 10^9$ floats equaling 9 GB of memory assuming single precision. Any larger image would already start to cause memory problems on a standard workstation. Furthermore, since the computation of the empirical covariance matrix involves an outer product, the computation becomes restrictively slow for the full covariance matrix. Hence, we compute the blocks for each resolution of the flow, that is split the $\boldsymbol{z}$-vector into pieces corresponding to the parts that are split from the flow at each resolution exactly like in Fig. 5.10.

Observations Figure 5.12 visualizes the 4×4-resolution covariance matrix for the U-Net activations model variant as well as a corresponding *unconditional* model, where the RealNVP layers never receive any conditioning information. We observe that the whitening process is mostly successful, yet some traces of correlation remain as some structural patterns can be seen in the matrices outside the diagonal (insets of Fig. 5.12). Similar patterns can also be seen in the covariance matrices of other resolutions. This raises the question, whether these failure cases could be exploited in the sampling process. We study this later in section 5.4. Here, correlations between elements of different flow resolutions remain unexplored. They might, however, contain interesting information about the flow model. We leave this exploration for future work.

We also visualize the distribution of the covariance matrix values (Fig. 5.11) to understand how well the model manages to uncorrelate the dimensions in general. The three shown conditional model types are trained for the superresolution task. The unconditional variant is for reference. Vanishing covariance does not guarantee independence but the lack thereof signals a failure of the model. We find that all model types behave relatively similarly, with essentially all the mass of each histogram gathered in the interval $[-0.1, 0.1]$ signaling near desired behavior.

Finally, we compute the empirical mean and covariance values of the 2000 validation set samples and sample according to Gaussians parametrized by the mean and covariance of the test-data. We visualize the results in Figure 5.13. We find that that fitting a Gaussian to the empirical distribution via the maximum likelihood parameter estimation and sampling according to those distributions yields very similar results to those acquired using the standard sampling scheme. This is an indication that the empirical distribution is close to the unit Gaussian. Alternatively, it is possible that the conditional models simply are very resilient against the choice of the value of $\boldsymbol{z}$, as long as the values remain reasonably small in magnitude. We do, however, observe degradation of the results using the standard sampling method if the sampling temperature σ is close to or larger than unity. Figures illustrating this behavior are given in Appendix A.

The high-level conclusion of the tests in this section is that the optimization process succeeds quite well overall in terms of the Gaussianization and no obvious large faults are observed. Minor traces of correlation are, however, noticeable in the resulting values. This can potentially be a property of the FFHQ dataset, which is extremely homogeneous. It may be very difficult to find uncorrelated directions, since the pixels are so strongly correlated to begin with. The correlation, however, potentially allows one to find interesting directions in the $\boldsymbol{z}$-space. Directions, that are so strongly entangled that they are near impossible to separate, revealing intriguing properties of the dataset.

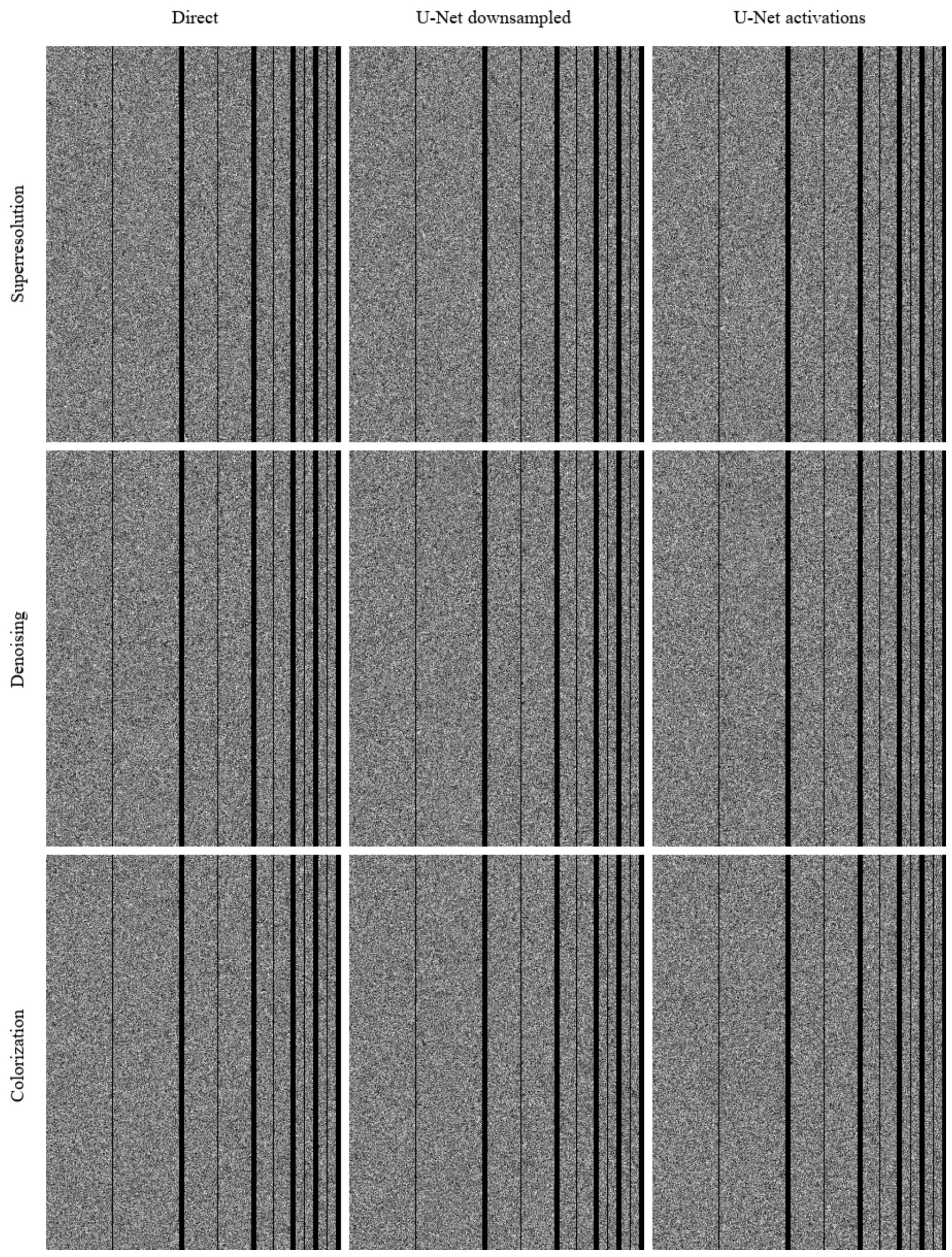


Figure 5.10: Visualization of a single validation set image mapped into $\boldsymbol{z}$ space for each forward process (rows) and each conditioning method (columns). Each image in the grid is divided into five columns of something resembling white noise with a narrow vertical black line splitting each column. The values on the left of each narrow black line correspond to simulated white noise using pseudorandom numbers. The values on the right of each black narrow line correspond to values of $\boldsymbol{z}_{r\times r}$ for resolutions $r = 64 \ldots 4$ from left to right.

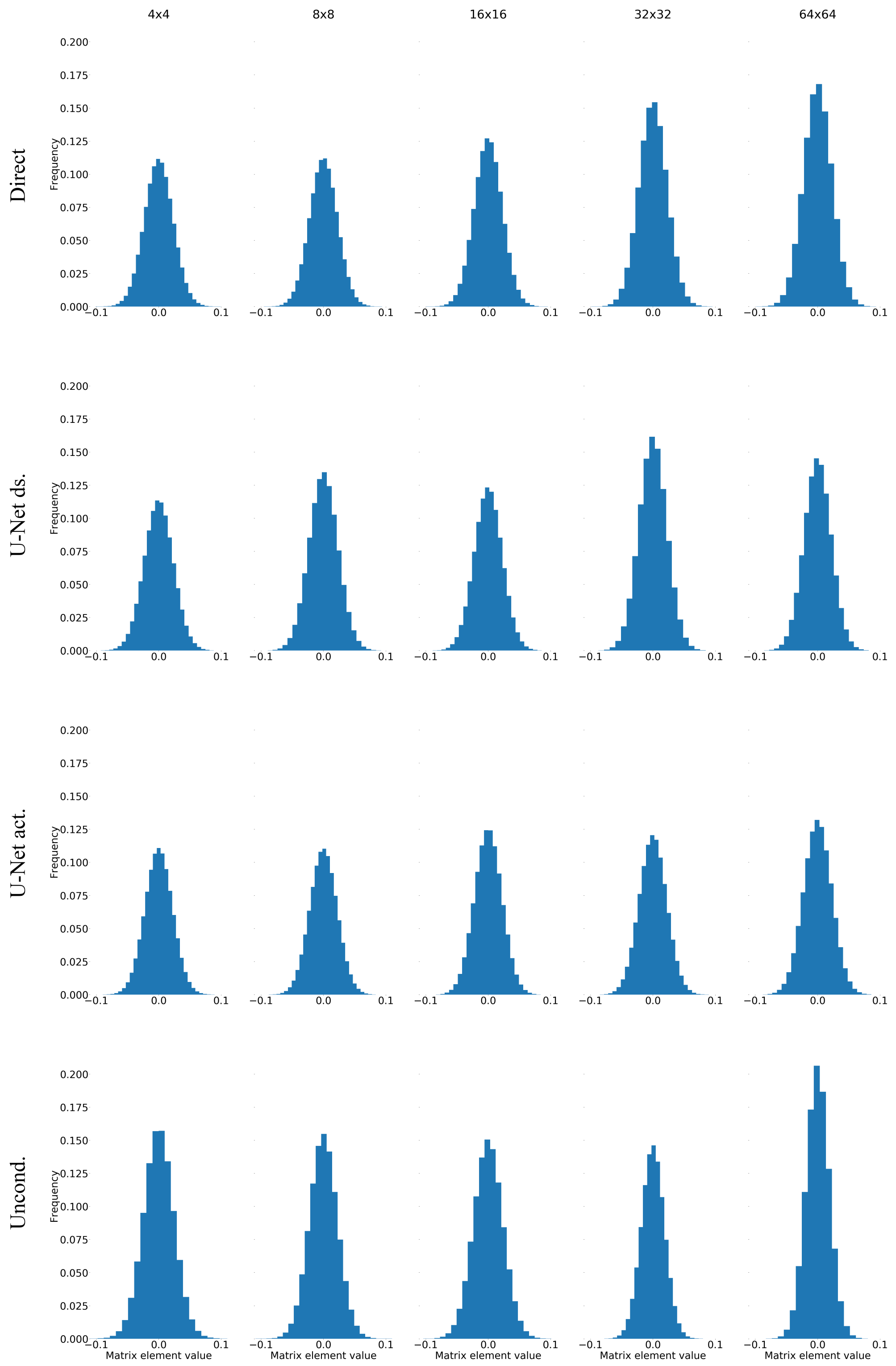
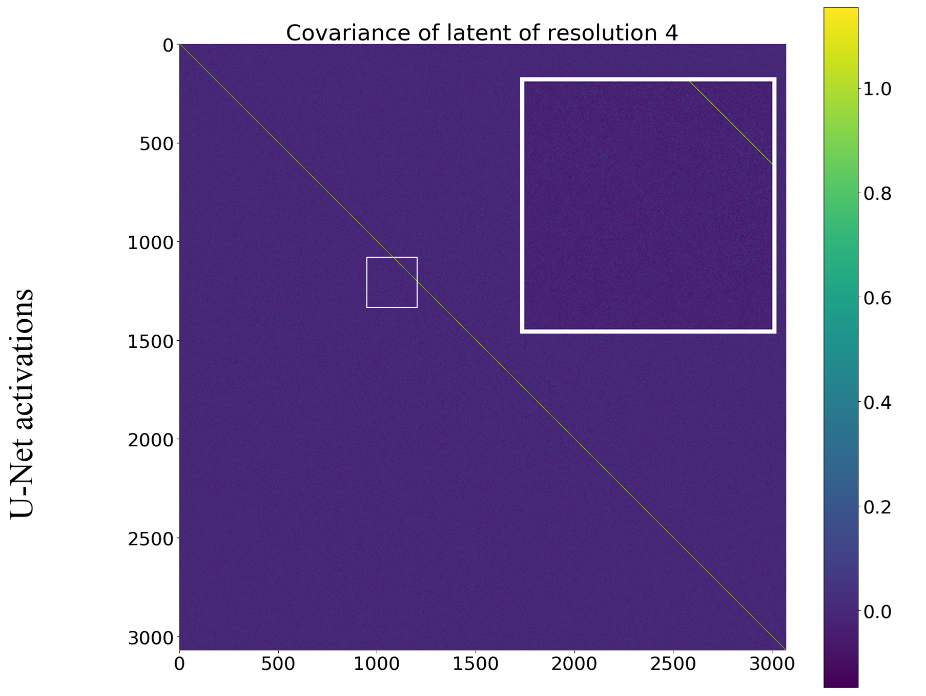


Figure 5.11: $r \times r$ -resolution $\boldsymbol{z}$-space covariance matrix values (columns) for four different model types Direct, U-Net downsampled, U-Net activations and Uncoditional (rows). Ideally, all but the diagonal elements should vanish and the distribution of the values is a spike around zero. All models are relatively well-behaved and have distributions concentrated around zero.



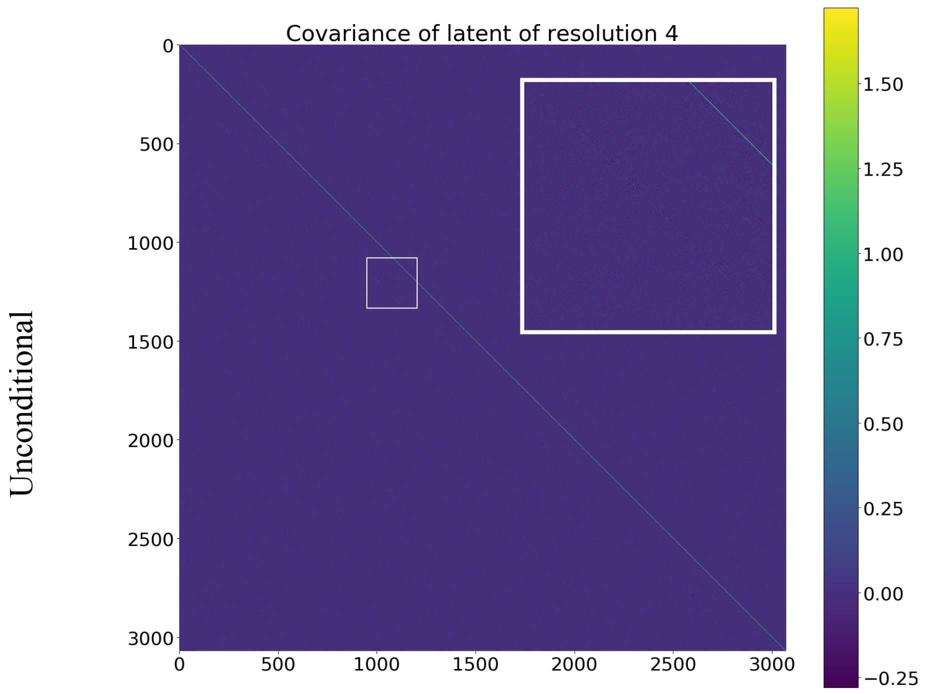


Figure 5.12: 4×4 flow resolution covariance matrices for the superresolution / unconditional models (rows). Minor patterns of undesired correlation can be observed in the covariance matrices of both models. Other resolution/task -pairs are found to have similar patterns.

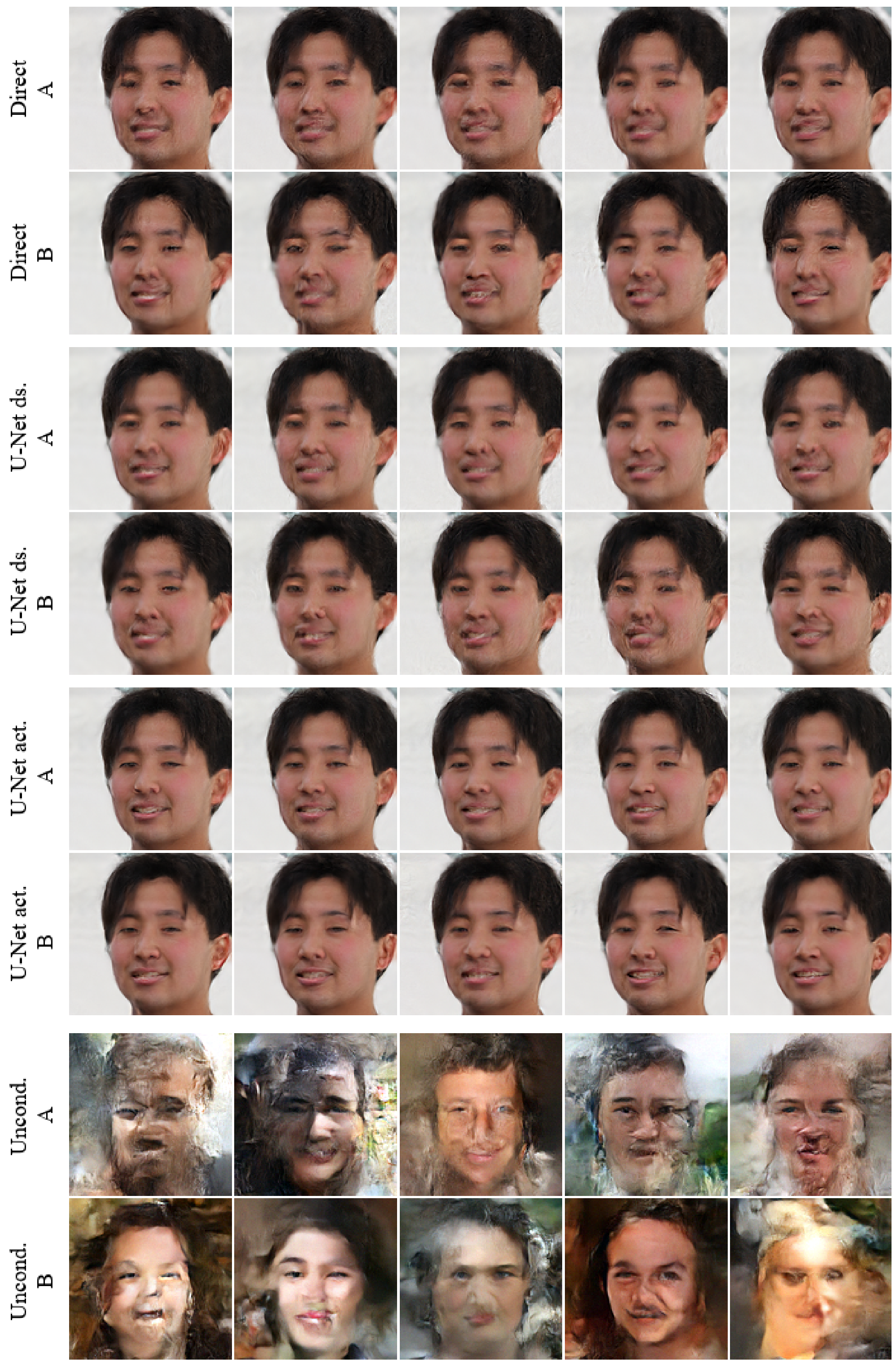


Figure 5.13: Comparison of sampling methods. Rows labeled with A are sampled using the standard zero-mean, identity-covariance Gaussian. Rows labeled with B use mean and covariance computed using validation data. In general, sampling according to the empirical parameter estimates yields results no different from sampling according to the zero-mean unit Gaussian.

5.4 Latent Structure

In the previous section we observed that even though the flow-model is tasked with normalizing the input data distribution, traces of correlation between the input vector elements remain. Since the components in the $\boldsymbol{z}$ space are not entirely independent, the choice of how $\boldsymbol{z}$ is sampled could have a non-insignificant effect on the resulting image. We now study the impact of the choice of its values. We continue the line of work of the computation of encoded validation data covariance matrix from the previous section by applying dimensionality reduction. We eigen-decompose the covariance matrices corresponding to the 4×4 layer, find the the eigenvectors corresponding to 5% of the largest largest eigenvalues, and sample along those axes normally while heavily truncating the other directions, by settings the smallest eigenvalues in the matrix decomposition to a small positive value ε.

We choose to only decompose the covariance matrix of the 4×4 -resolution layer for computational efficiency and because intuitively, the deepest layer and its input $\boldsymbol{z}_{4\times4}$ should have the most significant effect on the resulting image. We hypothesize this to be the case as this information is processed the most. One would expect, that the high-resolution layers only add detail to the output and the low-resolution layers define the general geometric shape. Interestingly, little observable difference can be seen when employing the aforementioned sampling scheme on the superresolution task as seen on Figure 5.14. Conversely, the truncation scheme has a large effect on the unconditional model, rendering the images much cleaner.

In addition to applying the alternative sampling scheme, we also attempt to see the effect of corrupting the values of $\boldsymbol{z}$ at various other layers of the flow. We again decompose the vector $\boldsymbol{z}$ into parts corresponding to values that are split/concatenated into the flow at various resolutions. We first encode a clean image via $\boldsymbol{z} = f^{-1}(\boldsymbol{x})$ and decompose the resulting $\boldsymbol{z}$ into components $\boldsymbol{z}_{4\times4}$, $\boldsymbol{z}_{8\times8}, \ldots, \boldsymbol{z}_{64\times64}$. We sample new values for values of corresponding to some layer $\boldsymbol{z}_{n\times n}$ while keeping the rest fixed at their computed, "true" values and observe the changes in the resulting decoded image via $\boldsymbol{x} = f(\boldsymbol{z})$. We do the re-sampling cumulatively starting from the lowest resolution layer and visualize the results in Fig. 5.15. We repeat the experiment in the inverse order in Fig. 5.16.

Observations Since only the deepest layers are affected by the change of sampling technique, it seems that models trained for the superresolution task mostly work on finer levels of detail, which considering the nature of the task, is intuitively rather pleasing. The low-resolution information from the flow is mostly ignored as it comes directly from the conditioner $\boldsymbol{y}$. In contrast, background artifacts are clearly suppressed in the images produced by the unconditional model suggesting that some directions in $\boldsymbol{z}$ at the deepest layers encode more relevant and structured information than others. Furthermore, we can conclude that the unconditional model is much more sensitive to changes of the low-resolution latent code due to the lack of additional information of $\boldsymbol{y}$.

We note again that the conditional and unconditional model types have very different behaviors. The image produced by the unconditional model is for the most part, defined by the lowest-resolution layers, supporting our previous results. This is seen the most clearly in Fig. 5.16, (rightmost column) where the identity of the face produced by the unconditional model remains the same up until all layers but the deepest have had their values of $\boldsymbol{z}$ randomized. Sampling new values for $\boldsymbol{z}_{4\times 4}$ is enough to drastically change the appearance of the image even though only a small fraction of values of $\boldsymbol{z}$ is modified, as seen in Fig. 5.15.

The conditional superresolution model exhibits the opposite behavior by being robust to changes and corruptions of $\boldsymbol{z}$ in the low-resolution layers. In fact, values of $\boldsymbol{z}$ up to 32×32 resolution can be corrupted with little effect on the decoded output image (Fig. 5.15) using the U-Net activations model type, further supporting our belief that the superresolution models mostly work at the high-resolution layers. Interestingly, the Direct model type is much more vulnerable to the corruption already at low resolution layers. In general, U-Net-based conditioning methods seemed to yield mostly blank (seemingly uninformative, not natural image-like) conditioning information into the deepest layers, signaling that the lower levels are completely ignored. We hypothesize that fact that the Direct model transfers meaningful image-information to the lowest resolution via naive downscaling might hurt its performance. According to the more expressive U-Net type models, the best action seems to be to ignore that information.

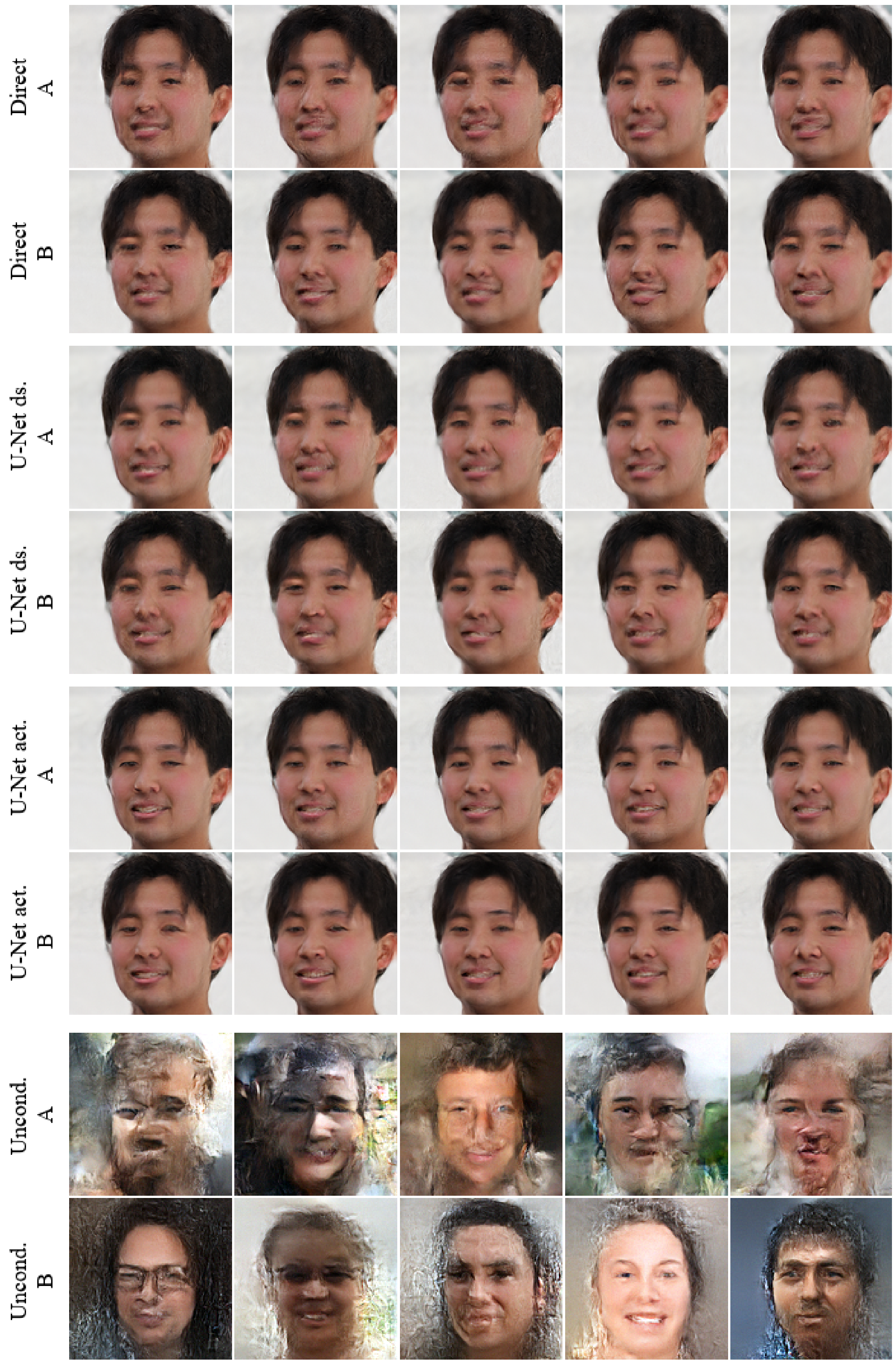


Figure 5.14: Selective truncation scheme for resolution 4×4. Sampling using $\sigma > 0$ only along principal axes with the top 5% largest eigenvalues. The remaining directions have their eigenvalues set to a small $\varepsilon > 0$. The conditional models are unaffected, but clear difference can be observed in the UNCONDITIONAL model (last grouped row). The conditional models are all conditioned with the same image, and columns represent different samples.

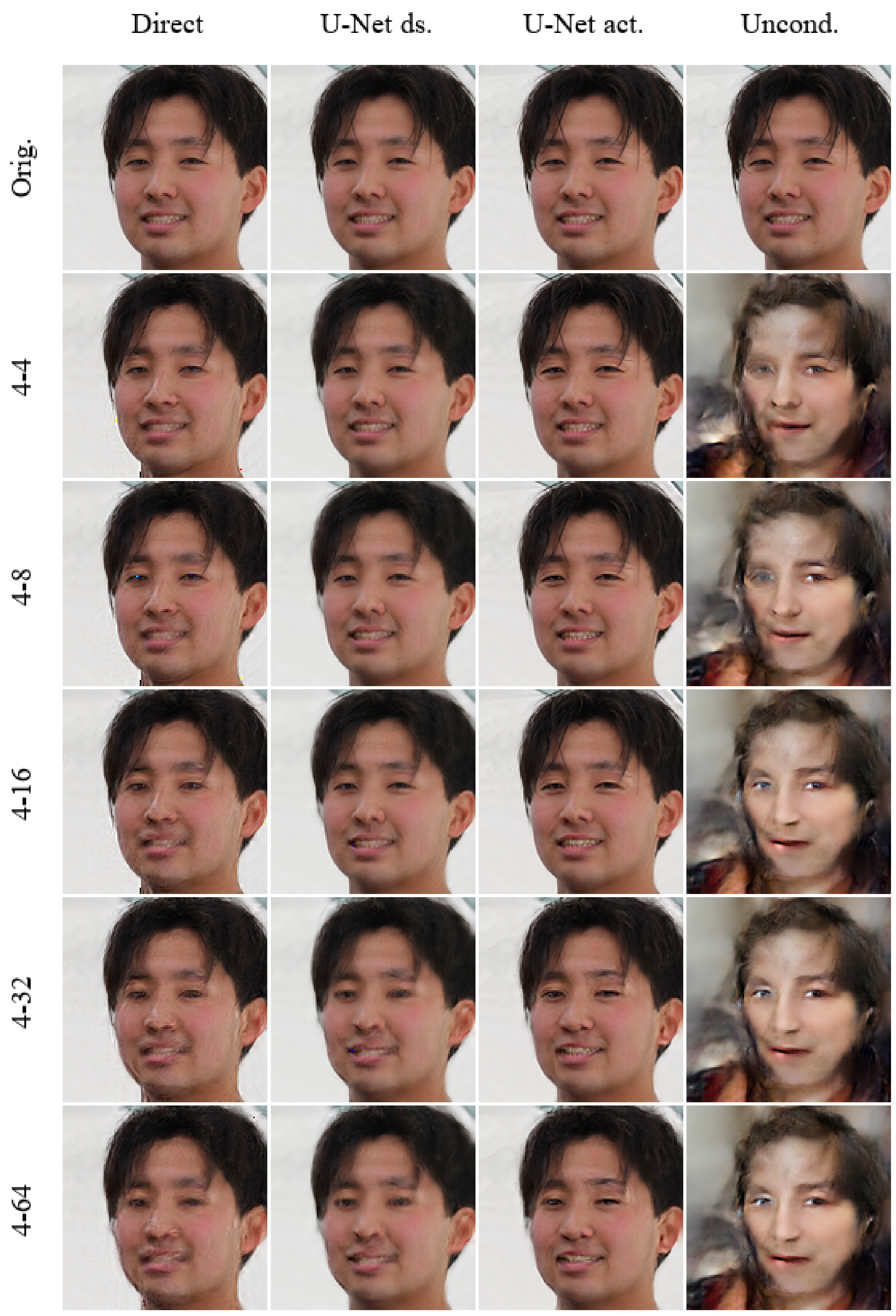


Figure 5.15: Layer-wise randomization of values $\boldsymbol{z}$ starting from the low-resolution layers. In the top row, no part of $\boldsymbol{z} = f^{-1}(\boldsymbol{x})$ has been corrupted resulting in the original image. In the second row (labeled 4-4), $\boldsymbol{z}_{4\times 4}$ has been replaced with random numbers distributed according to a zero-mean unit Gaussian. In the third row (labeled 4-8), $\boldsymbol{z}_{4\times 4}$ and $\boldsymbol{z}_{8\times 8}$ have been replaced and so forth.

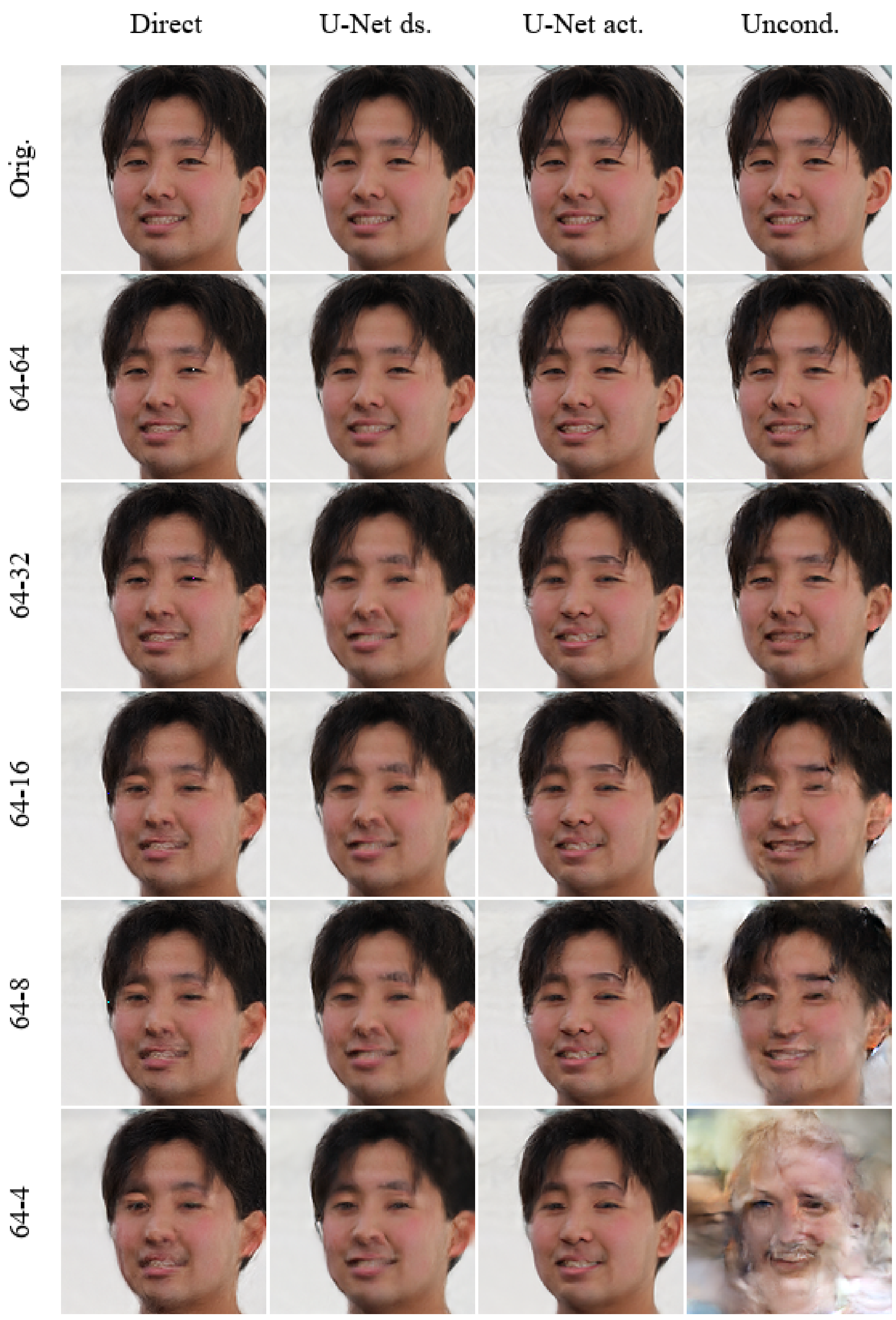


Figure 5.16: Layer-wise randomization of values $\boldsymbol{z}$ starting from the high-resolution layers. Same as Fig. 5.15 but in inverse order. The quality of the results of the conditional models suffer already after corruption of the 64-resolution layer (second row). The unconditional model (right-most column) encodes most of the information in the low-resolution layers and the identity of the face clearly changes only after corrupting the 4×4 resolution values (last row).

5.5 Image Formation

We showed in the last section that unconditional and conditional (superresolution) flow-models operate differently concerning where and how the final image is formed in the flow. We now present further evidence supporting the aforementioned observations in the form of additional characterizations of the normalizing flow. First, we visualize the internal activations of the flow on several resolutions in Figure 5.18 by interpreting the post-activation tensors as RGB-images (splitting along the channel dimension if more than three channels). The figure shows how the normalizing flow invertibly transforms the vector sampled from the Gaussian prior distribution into an image and specifically, where exactly the image is formed.

The flow-model seems to learn task-specific strategies for finding plausible solutions. In the superresolution task the model reasonably finds the sharp edges of the image staring in the mid-resolution layers. This is intuitively pleasing as the task mostly involves restoring sharp, high-resolution detail in the image. The low-resolution condition image can be upsampled naively with an interpolation technique or with a preprocessing neural network and the high-resolution detail restored using the information of the flow. The flow never has to construct the entire image only by itself. Similar behavior can be observed in the other two model variants as well. Interestingly, the lower-resolution layers begin to lack natural image-like properties. This is in agreement with the results of the previous section where we found the superresolution task to be quite robust to changes in parts of the latent corresponding to the low-resolution layers. This may also explain why the Direct model variant has the worst performance. Feeding image-like conditioning information into the low-resolution layers may be detrimental as the activations of the flow do not resemble natural images at the deepest layers. Using a intermediate network for preprocessing can help shape the conditioning into a form that is useful also at the lower levels of the flow.

Similar patterns can be seen in flow-models trained for the denoising task (second row of Fig. 5.18). Image-like properties are lost towards lower-resolution layers of the flow. Here, the objective of the model is not quite as clear intuitively as it is with the superresolution task. Furthermore, there is a clear difference between the Direct and the U-Net-based model variants which explains also the disparity in the quality of the results.

Conditional colorization and purely unconditional image generation seem to employ a different strategy compared with superresolution and denoising. They seem to transport image-like information in the activations all the way from low-resolution layers. This can be seen in last row of Fig. 5.18 as rather well-defined patterns in the activations visible already at the lowest shown resolution of 16×16. Furthermore, Figures 5.15 and 5.17 show that for unconditional image generation and conditional colorization the values of $\boldsymbol{z}$ corresponding to high-resolution layers have little effect on the final image — in complete contrast to superresolution and denoising. These results indicate that the overall color of an image and the general shape of a face can be encoded in a relatively low-dimensional space ($4 \times 4 \times 192$, or even lower) compared to the dimensionality of a full image ($128 \times 128 \times 3$). Color palette and general shape are also perhaps more low-level detail that affect the entire image. Superresolution and denoising mostly restore high-frequency detail, which can be applied locally on the outermost layers of the flow without detailed global context.

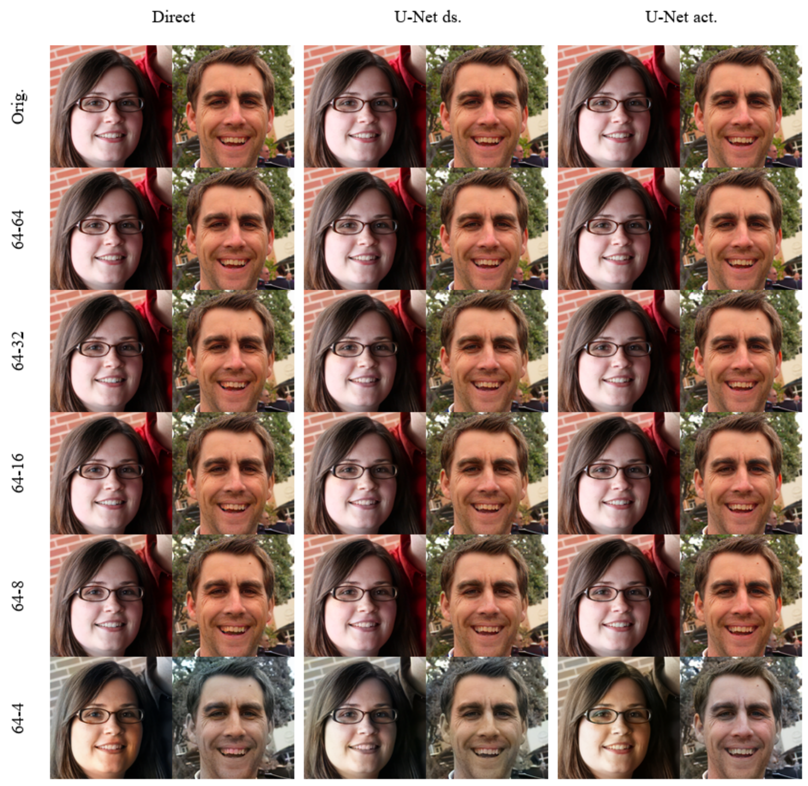


Figure 5.17: Layer-wise randomization of values $\boldsymbol{z}$ starting from high-resolution layers for the colorization task. Opposite to the superresolution task, the appearance of the final image is mostly decided by the low-resolution part of $\boldsymbol{z}$ for each model variant. The colorization remains close to the original (first row) up until the lowest-resolution values $z_{4\times4}$ (last row) are corrupted.

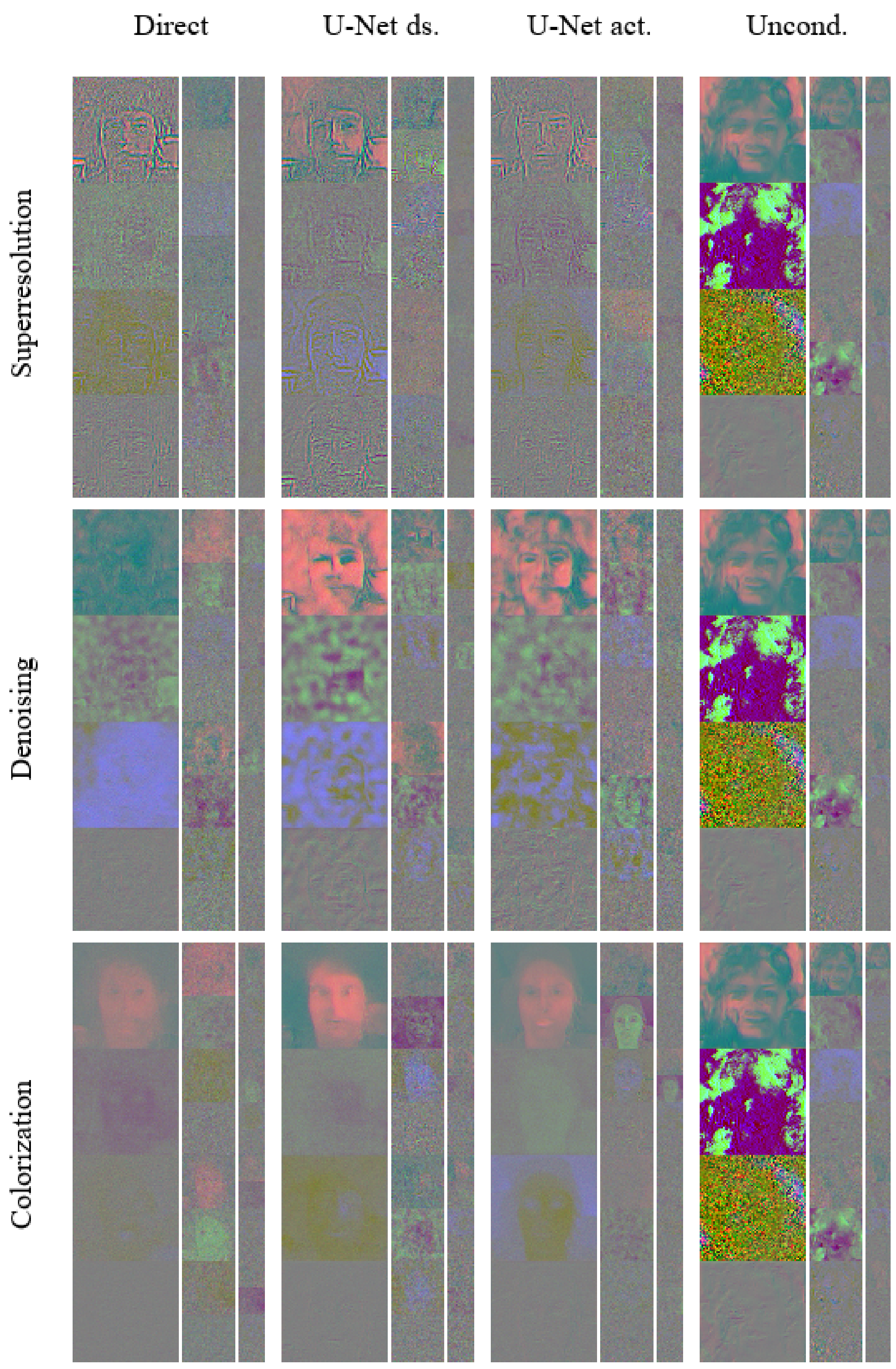
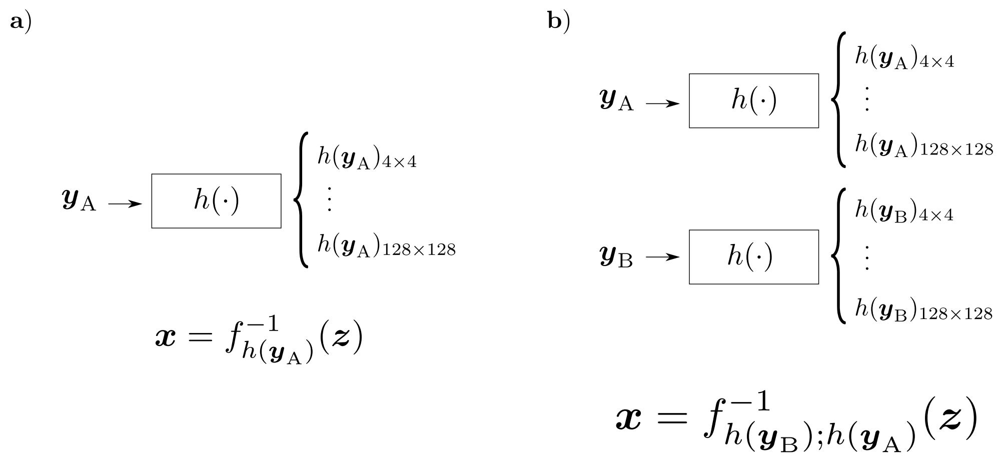


Figure 5.18: Flow intermediate activations. Each task-architecture pair in the grid consists of 3-channel activation maps of resolutions 64, 32 and 16 from left to right. For the unconditional architecture there is no difference between the task since the condition information is never shown and hence the leftmost column has the same activations repeated three times.

5.5.1 Unregularized Mixed Conditioning

We now take a different approach to studying the image formation process of the normalizing flow. Instead of mutating the encoded values of $\boldsymbol{z}$ and holding the condition information $\boldsymbol{y}$ static, we examine the opposite of using a constant $\boldsymbol{z}$ and varying $\boldsymbol{y}$. We *mix* condition information from two different images A and B by computing $h(\boldsymbol{y}_\mathrm{A})$ and $h(\boldsymbol{y}_\mathrm{B})$, where $h(\cdot)$ is a preprocessing method, and choosing the conditioning information for each layer of the flow from either that of image A or image B. Figure 5.19 demonstrates the difference between conditioning schemes.

Figures 5.20 and 5.21 show the results of condition information mixing cumulatively from low to high-resolution and high to low-resolution, respectively. We find that in addition to being robust against changes in the $\boldsymbol{z}$-code in the low-resolution layers, the flow mostly ignores the condition information $\boldsymbol{y}$ in the low-resolution layers for all three variants. The outlines of Image B start to appear only after the three bottom layers are conditioned with image B instead of image A. Conditioning in the last and highest-resolution layer is essentially the deciding factor in the final appearance, although minor traces (outlines) appear already on the previous resolutions. The same effect can be seen in the inverse order visualization.



a)
b)
$\boldsymbol{y}_\mathrm{A}$
$h(\cdot)$
$h(\boldsymbol{y}_\mathrm{A})_{4\times4}$
$h(\boldsymbol{y}_\mathrm{A})_{128\times128}$
$\boldsymbol{x} = f^{-1}_{h(\boldsymbol{y}_\mathrm{A})}(\boldsymbol{z})$
$\boldsymbol{y}_\mathrm{B}$
$h(\boldsymbol{y}_\mathrm{B})_{4\times4}$
$h(\boldsymbol{y}_\mathrm{B})_{128\times128}$
$\boldsymbol{x} = f^{-1}_{h(\boldsymbol{y}_\mathrm{B});h(\boldsymbol{y}_\mathrm{A})}(\boldsymbol{z})$

Figure 5.19: Conditioning with multiple images at inference time. **a**) Standard conditioning where the flow $f(\cdot)$ is parametrized only with image $\boldsymbol{y}_\mathrm{A}$ (possibly preprocessed with $h(\cdot)$). **b**) Conditioning with two images A and B. Both images are processed with the preprocessing network $h(\cdot)$ and the results are combined by choosing for each resolution r, whether to use $h(\boldsymbol{y}_\mathrm{A})_{r\times r}$ or $h(\boldsymbol{y}_\mathrm{B})_{r\times r}$.

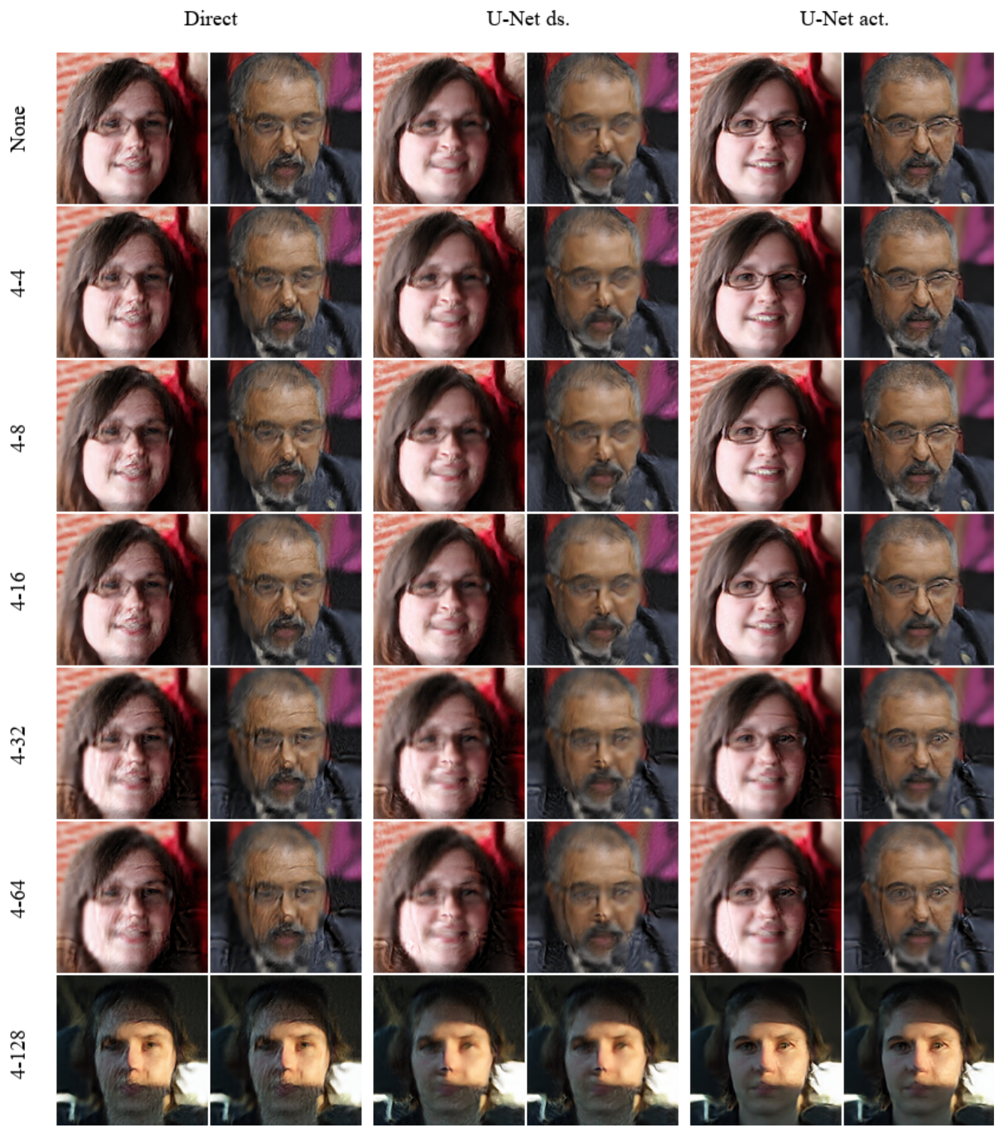


Figure 5.20: Condition information mixing in the superresolution task. The top row (labeled as None) is generated by only conditioning with the respective image (image A). The bottom row (4-128) uses a different image B for all the layers. The second row (4-4) is generated using image B on the 4×4 -resolution layer and image A elsewhere. We cumulatively add image B row-by-row to higher resolution layers until only image B is used for conditioning (bottom row). The output image is mostly defined by the conditioning information on the outermost layer, although outlines start already bleeding through few layers earlier.

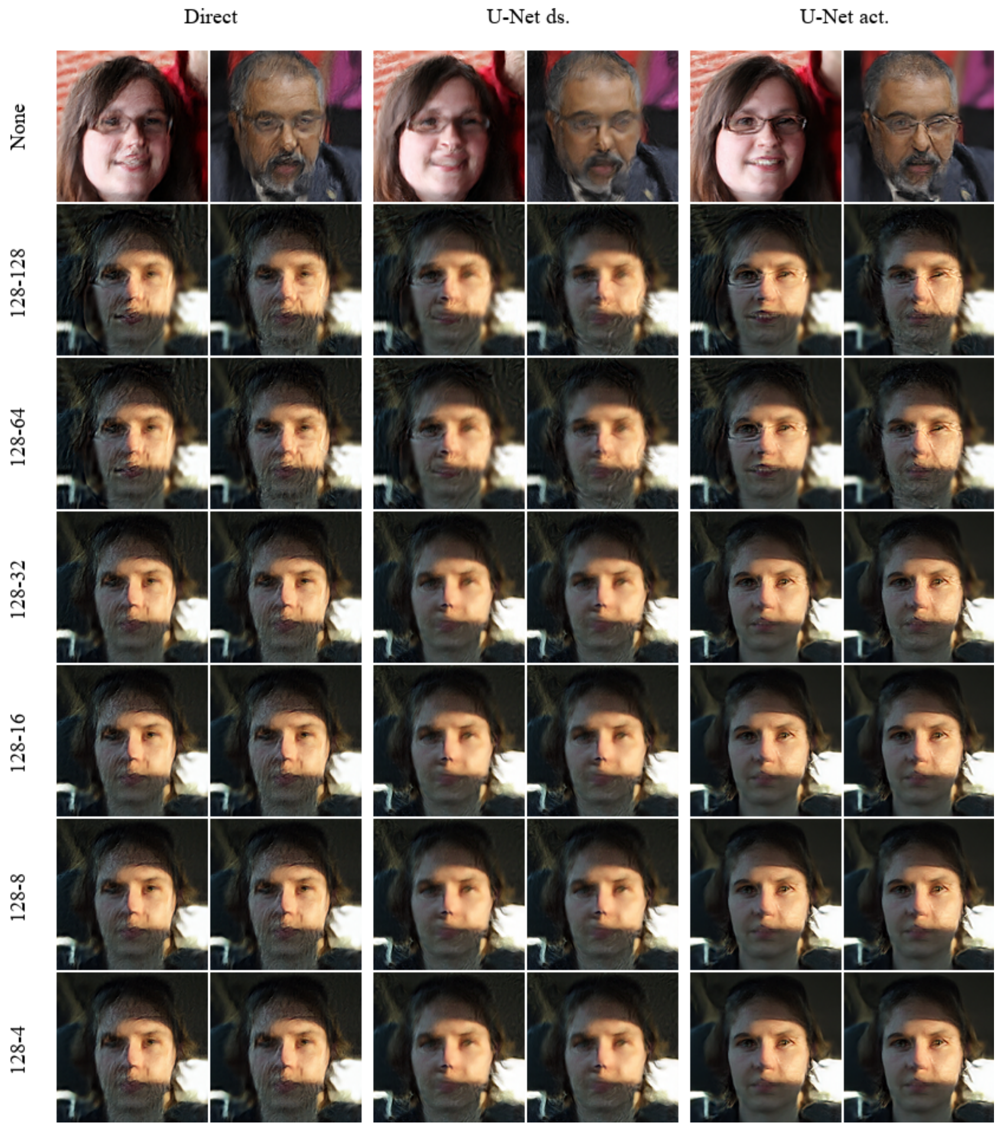


Figure 5.21: Condition information mixing in the superresolution task. Similar to Figure 5.20, but in reverse order. We cumulatively condition the image with information from image B starting from the high-resolution layers, starting from the second row, going downwards in the image grid. As the outermost layers mostly define the appearance of the output, conditioning layers of resolutions 128-64 is enough. Adding the conditioning information to the lower layers merely improves the output quality.

5.5.2 Regularized Mixed Conditioning

An interesting question is to ask whether it truly is a good solution for the flow in the superresolution and denoising tasks to form the image almost entirely at the top layers of the flow. We can try to *regularize* the flow by stochastically blocking the conditioning information from reaching the RealNVP-layers during training. In the previous experiments, all layers have always received the correct conditioning information during training. This should enforce the flow no to rely on being able to essentially copy the output of the preprocessing network, but to start the process, on average, already at lower layers of the flow. We employ the masking scheme during training as follows:

Algorithm 1 Masking scheme for regularization. The flow of conditioning information is blocked cumulatively to a number of layers ranging from none to all of them.

```
    Input:
    $\boldsymbol{y}$: corrupted conditioning image
    $N_{\text{layers}}$: Number of layers in the flow
1: procedure MASKING($\boldsymbol{y}$, $N_{\text{layers}}$)
2:     $h(\boldsymbol{y})_{4\times 4}, \ldots, h(\boldsymbol{y})_{128\times 128} \leftarrow \text{LayerwiseSplit}(h(\boldsymbol{y}))$
3:     $N_{\text{masked}} \leftarrow \text{RandInt}(0, N_{\text{layers}})$
4:     for $i, r$ in enumerate($[4, 8, \ldots 128]$) do
5:         if $i \geq N_{\text{masked}}$ then
6:             $h(\boldsymbol{y})_{r\times r} \leftarrow \mathbf{0}$
       return $[h(\boldsymbol{y})_{4\times 4}, \ldots, h(\boldsymbol{y})_{128\times 128}]$
```

Algorithm 1 cumulatively masks layers starting from the top layers of the flow. Alternatively, the masking can be started from the bottom by changing the direction of the inequality in the if-conditional. Starting the masking from the top should yield more drastic changes in the behavior of the flow if the image is formed on the top layers since the probability of masking the top-most layer is much higher than by starting the masking cumulatively from the low-resolution layers.

As expected, we see clear shift in the strategy of the flow-model in terms where and how the image is formed, as demonstrated in Fig. 5.22. In the superresolution task, the flow is no longer generating only sharp edges and superimposing those with the output of the preprocessing network. Instead, the activations resemble much more those of the denoising or the colorization tasks. There is less change in the intermediate activations in the other two tasks. However, a common characteristic in all the models is that the natural image-like look is retained also in the deeper layers of the flow.

Since there is a non-zero probability of having all conditioning information being blocked from entering the flow, the regularized network is in fact a hybrid between unconditional and conditional models. This can observed in the outputs by selectively masking conditioning inputs during inference. Instead of mixing the values of two separate natural images, we set image B to a flat gray color to emphasize the semi-unconditional nature of the model. The effect of employing a blank conditioning image for the mixing is visualized in Figures 5.25 and 5.26, where blocking of the layers with the most contribution results in an image of a face that no longer resembles the original conditioning image $\boldsymbol{y}$. Interestingly this does not seem to happen in the denoising task, perhaps due to too large average corruption (noise standard deviation) in the training data which on average produces blank images.

The artifacts created by the semi-unconditional nature of the model can also be observed in Fig. 5.23 where outlines of another image can be seen in cases where the result is, in fact, only conditioned by a single image (first and last rows). Similar artifacts are not present in the denosing task (Fig. 5.24), which is in line with the lack of unconditionally generated faces in Figure 5.25 for the denoising column.

On average half of the of the conditioning layers are masked when masking is employed following Algorithm 1. Hence it is not surprising to see a shift from image A into image B in the conditioning information mixing experiment in Figures 5.23 and 5.24, for the superresolution and denoising task, respectively.

Adding the masking regularization into the model does not seem to bring any benefits. The results look across all tasks significantly worse than those produced by unregularized models. This is not a total surprise as the regularized task is more demanding in difficulty since the flow can no longer rely on getting an almost correct answer from the preprocessing U-Net directly to the output-resolution conditional layer. Resources need to be allocated to ensure that similar information can be routed from the deeper layers as well. Since the flow function already suffers from the lack of representational power due to the constraints on invertibility and tractable Jacobian determinant computation, increasing the task difficulty hurts the performance significantly.

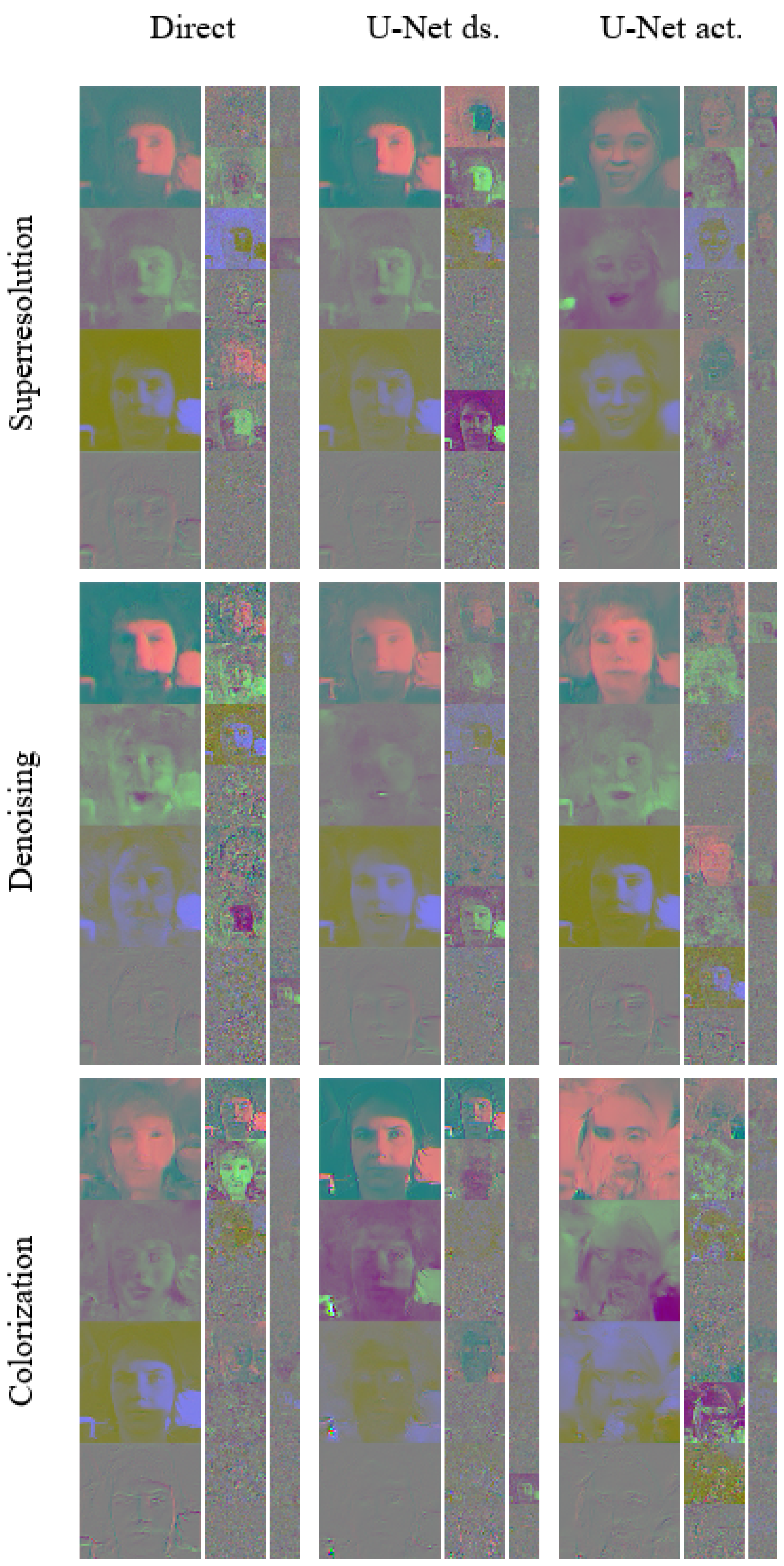


Figure 5.22: Flow intermediate activations for regularized models. The activations resemble natural images much more that they do in the unregularized models.

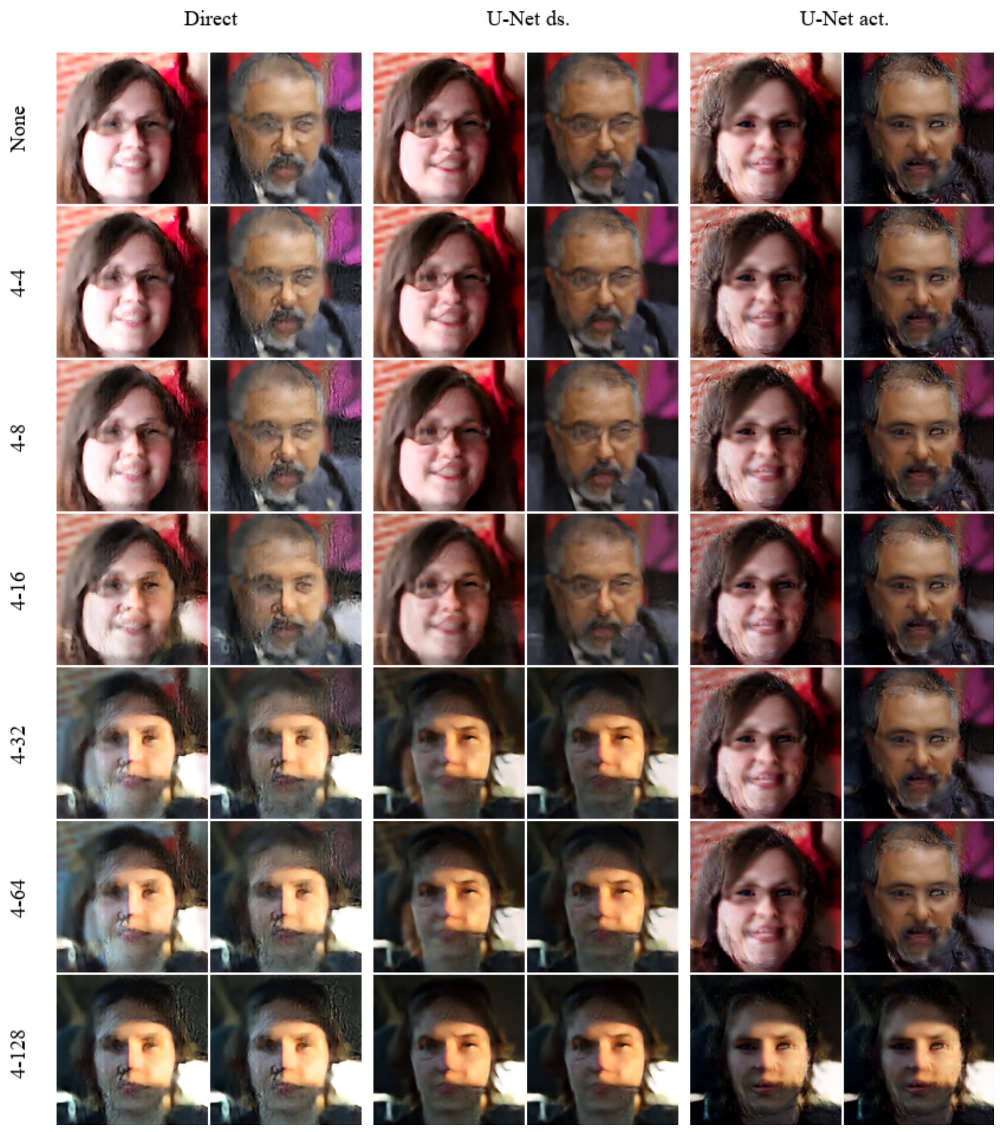


Figure 5.23: Condition information mixing in the superresolution task for the regularized model. The topmost layer of the flow no longer completely dictates the appearance of the output for the Direct and U-Net downsaled (first and second grouped columns) models, seen from the fact that the image starts to morph from condition image A (first row) to condition image B (last row) already after layers of resolution 4, 8 and 16. The U-Net activations (rightmost grouped columns) model seems to be more robust against the regularization and retains its old behavior.

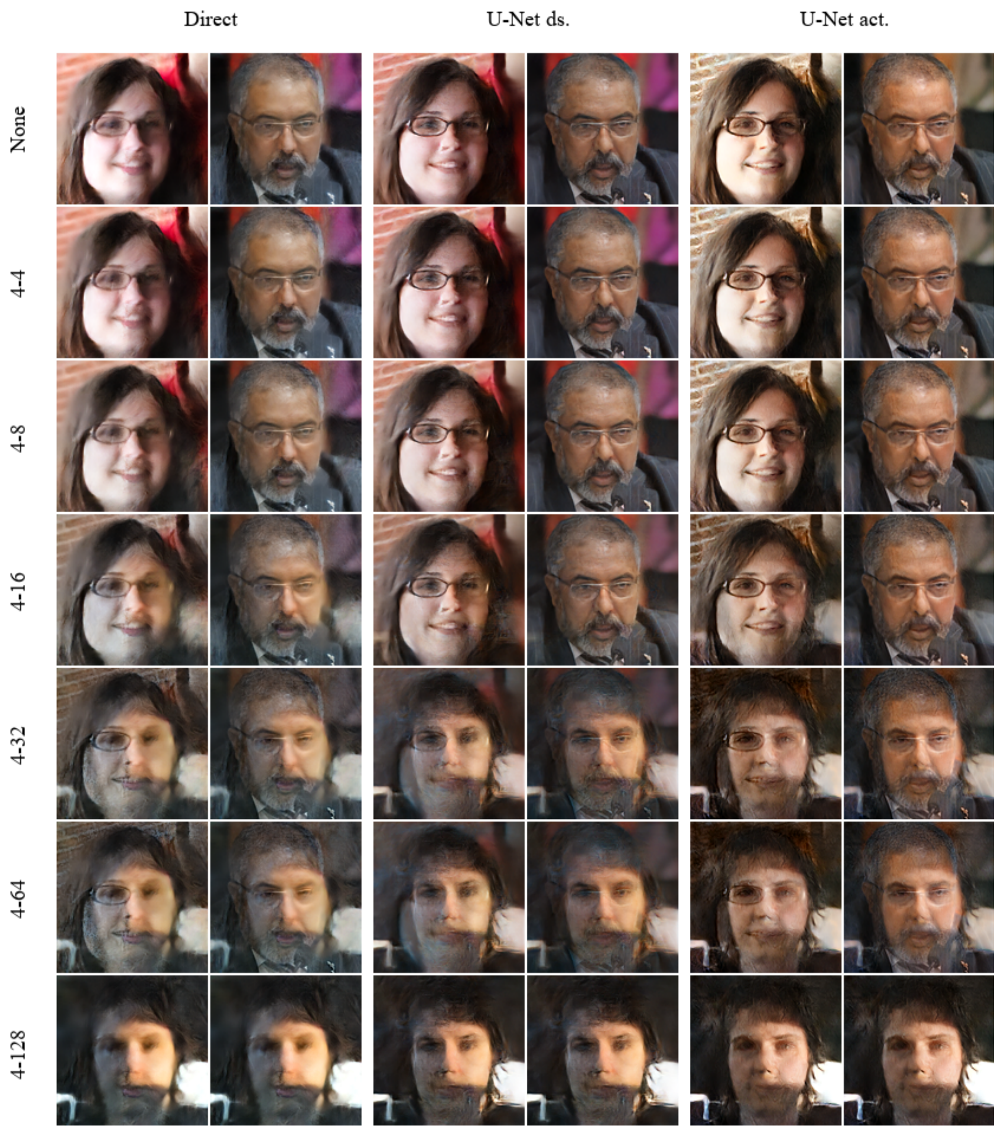


Figure 5.24: Condition information mixing in the denoising task for the regularized model. Like with the superresolution task, topmost layer of the flow no longer completely dictates the appearance of the output. Here, all the model types clearly have differing behavior from the unregularized case.

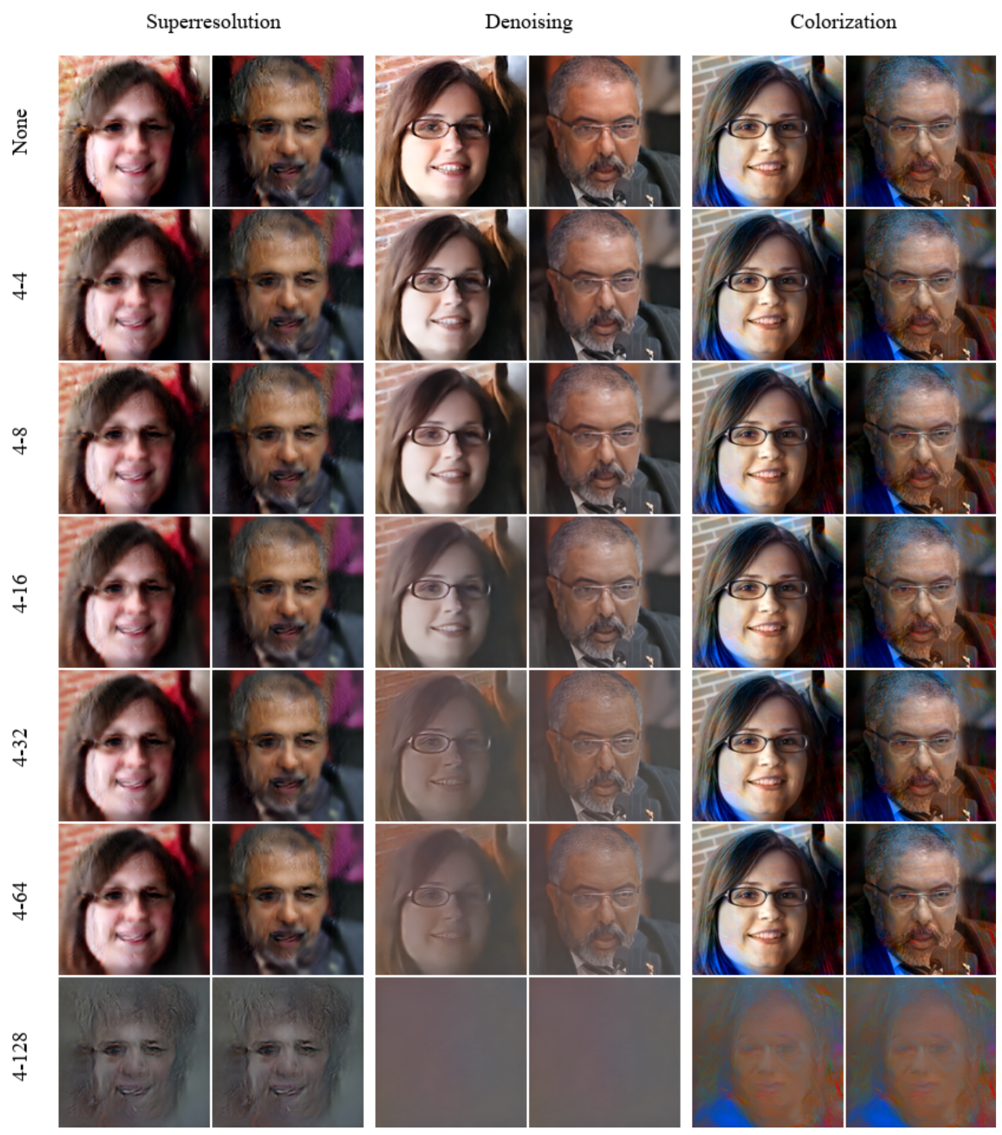


Figure 5.25: Condition information inference-time masking for all three tasks for U-Net activations model type. Masking starts cumulatively from the low-resolution layers (second row) until all conditioning information has been masked (bottom row). In the top row, all layers receive conditioning information (none is masked). Regardless of masking all conditioning information, an image resembling a face is generated in the last row in the superresolution and colorization tasks. The value of $\boldsymbol{z}$ is held constant in all images.

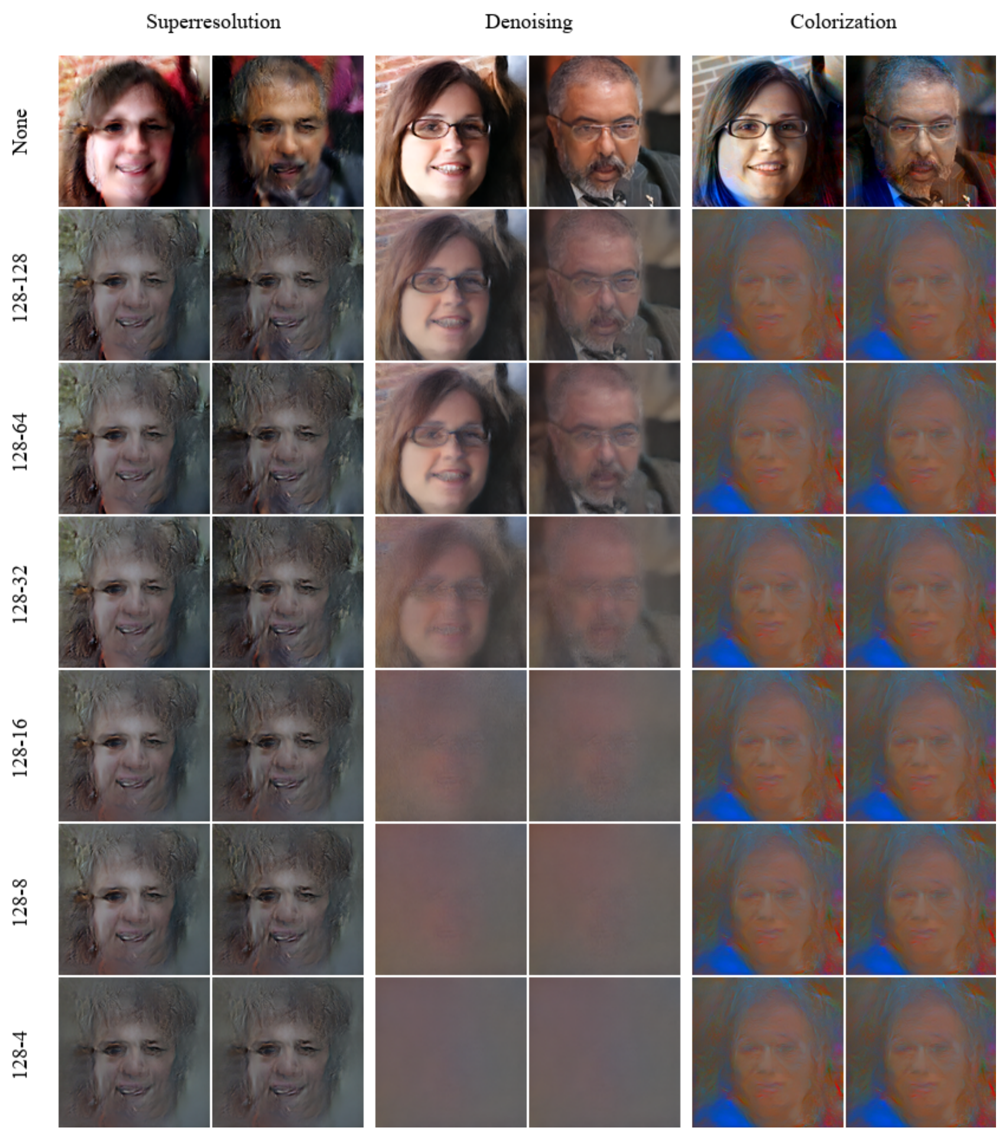


Figure 5.26: Condition information inference-time masking for all three tasks for U-Net activations model type. Masking starts cumulatively from the high-resolution layers (second row) until all conditioning information is masked in the last row. The value of $\boldsymbol{z}$ is held constant, hence the repeating images in the columns. As noted above, the U-Net activations model type retains its behavior of forming the image mostly based on the high-resolution conditioning information, regardless of regularization.

5.6 Model Compaction and Tuning

In the previous sections, we have seen much evidence that there exists one or two resolutions whose RealNVP-layers have by far the most influence on the resulting image. All models in the previous sections have also used a constant parameter R between resolutions. That is, each resolution has had the same number of RealNVP-layers. We now let the parameter vary between layers, such that there is a larger number of layers on the important resolutions and fewer on the resolutions of the least importance. Especially with models where the outermost layer has the most influence, this should reduce the number of parameters greatly. This is due to the fact that the coupling layers are convolutional and the lower resolutions have more features/channels than the high-resolution layers. We also modulate the image discretization level (models labeled with B are trained with 5-bit images), which greatly affects the negative log-likelihood values.

Table 5.3: Model compaction and discretization. U-Net activations is the same model that has been used throughout the experiments. R denotes the number of repeated RealNVP/invertible convolution layers per resolution. $R_{\text{outermost}}$ is the respective value for the outermost (128-resolution) layer.

Model Variant	R	$R_{\text{outermost}}$	Discretization	Num. Params.	NLL
U-Net act	12	12	8	88.2 M	2.867
U-Net act 2A	5	20	8	50.3 M	2.924
U-Net act 2B	5	20	5	50.3 M	1.253
U-Net act 3A	3	18	8	36.1 M	2.866
U-Net act 3B	3	18	5	36.1 M	1.252

We observe that the value loss-function, in itself, is not that good of an indicator of the sample quality. Model 3A achieves an equal negative log-likelihood value as the original model, despite having a significantly lower parameter count. Regardless, the results of the original model look significantly better regardless of the sampling temperature (Figs. 5.27 and 5.28). Despite not seeming to have a large contribution to the output in terms of changes of the values of $\boldsymbol{z}$ and $\boldsymbol{y}$, the lower layers of the flow remain important. Reducing the parameter count on those layers does affect the quality of the outputs. Conversely, using a lower discretization (5-bit images) level renders the problem easier, effectively reducing the number of bins in the underlying discrete distribution. The effect of 5-bit image discretization can be seen both in the decreased loss and the increased quality of the results.

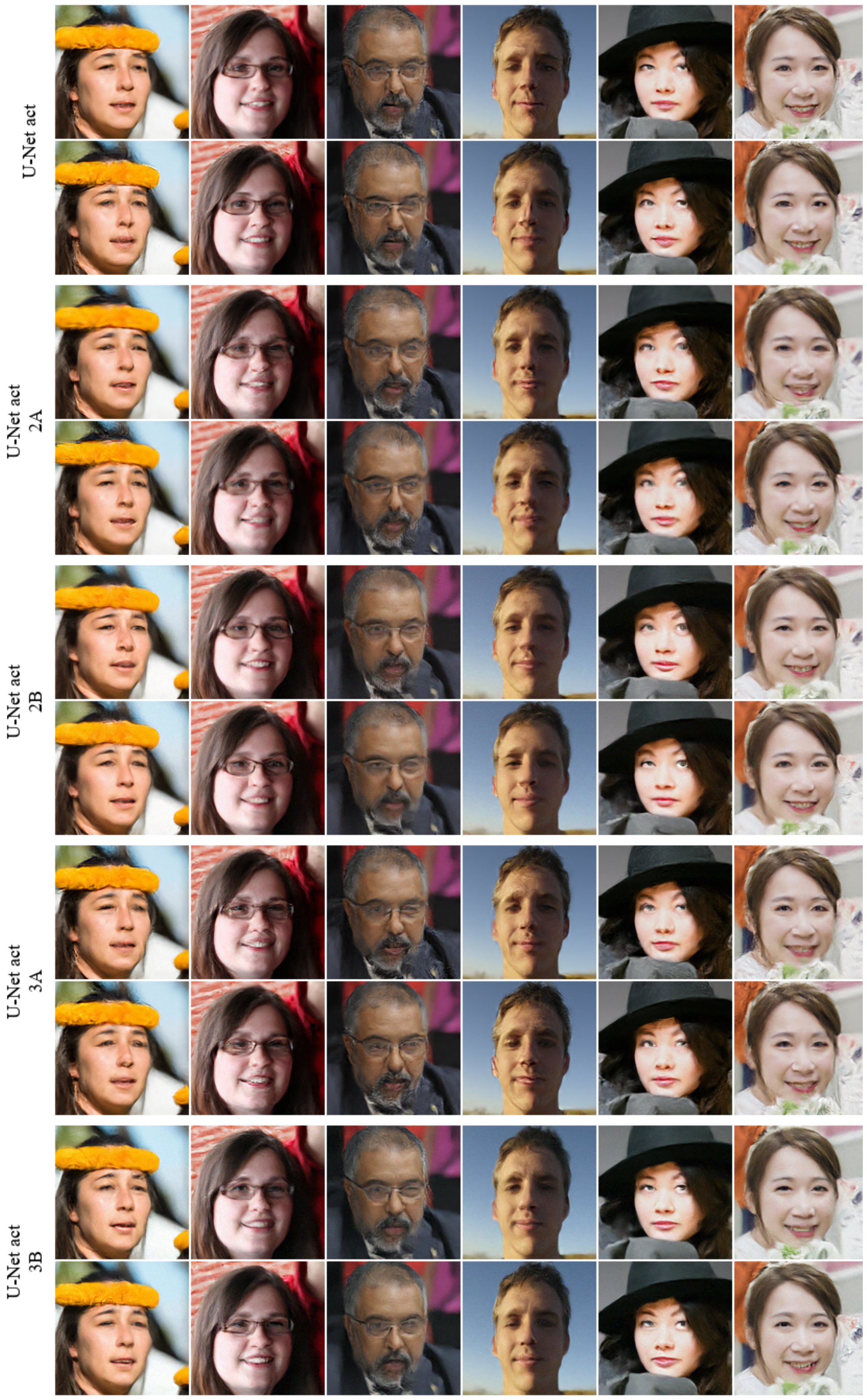


Figure 5.27: Model comparison for superresolution using sampling temperature $\sigma = 0.85$.

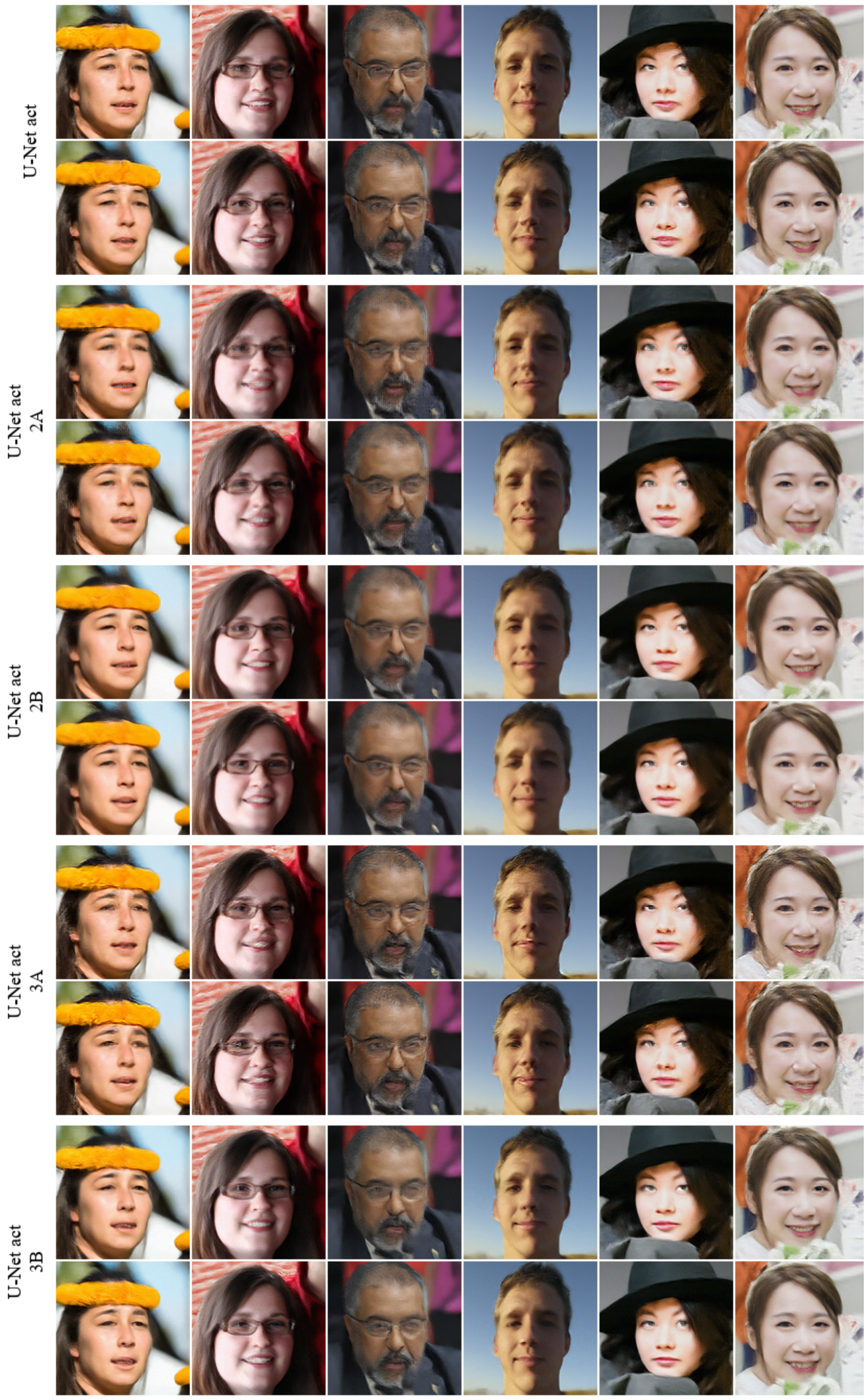


Figure 5.28: Model comparison for superresolution using sampling temperature $\sigma = 0.63$.

Chapter 6

Conclusion

Generative modeling has taken impressive leaps ahead during the past few years thanks to advances in algorithmic design and the abundance of sufficient computational resources. Regardless, no single model type is a silver bullet — better than all others in every aspect. Despite the recent success many open questions remain, for example concerning the evaluation metrics and regularization methods.

We have shown that normalizing flows are capable of directly modeling very high-dimensional conditional distributions, related to the solution space of various imaging-related inverse problems, without having explicit information of the underlying forward process. They hence provide an alternative for regression-based learning algorithms that provide only a single solution to the ill-posed problem. Using learning-based and data-driven techniques also avoids complex, hand-crafted likelihood-models and priors. Furthermore, there is no need to resort to expensive MCMC in the sampling. Training with likelihood maximization sidesteps the problem of choosing a meaningful similarity metric (pixel-wise distance, perceptual distance) in regressive models. We note that negative log-likelihood is not a very good direct measure of the quality of the output images, but that it does provide a meaningful guideline, at least asymptotically.

Our experiments have also shown interesting properties about both conditional and unconditional flow models. We have identified differences between how and where the model variants form the output images. We have shown which layers of the flow are vulnerable to changes in either the variable $\boldsymbol{z}$ or the conditioning information $\boldsymbol{y}$. We have also observed limitations in the representational power of the bijections of normalizing flows and found benefit in employing additional networks for preprocessing the conditioning information.

Future work We see two parallel paths of future work on conditional density estimation with normalizing flow. First is the question of expressive power of normalizing flows in general. Previous work has shown that flow models usually achieve worse negative log-likelihood scores than autoregressive models or VAEs. At the moment of writing, achieving high-resolution samples with high quality requires extremely deep flow models with hundreds of millions of parameters. Even then, the resolution of the samples lacks far behind the megapixel level achieved by the state-of-the-art GANs, implemented using only a fraction of the parameters required for a decent flow-model. Training deep flow-models is also quite unstable and requires dozens of high-end GPUs in order to be trained in a reasonable time. The second path is more directly related to this work. As optimizing pure likelihood does not necessarily directly translate into high-quality conditional output images, the training should be regularized, or biased in such a way that natural-looking images were more common. Our attempts at regularization did not quite manage to achieve this goal. Explicitly including some information about the underlying forward process would be an interesting continuation. Generalizing to domains other that pure image-to-image inverse problems would also be an interesting step forward. The framework of flow models is quite flexible and readily extendible for non-image data.

Bibliography

[1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467* (2016).

[2] Ardizzone, L., Kruse, J., Wirkert, S., Rahner, D., Pellegrini, E. W., Klessen, R. S., Maier-Hein, L., Rother, C., and Köthe, U. Analyzing inverse problems with invertible neural networks. *arXiv preprint arXiv:1808.04730* (2018).

[3] Ardizzone, L., Lüth, C., Kruse, J., Rother, C., and Köthe, U. Guided image generation with conditional invertible neural networks. *arXiv preprint arXiv:1907.02392* (2019).

[4] Arjovsky, M., Chintala, S., and Bottou, L. Wasserstein gan. *arXiv preprint arXiv:1701.07875* (2017).

[5] Bengio, Y., Léonard, N., and Courville, A. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432* (2013).

[6] Berg, R. v. d., Hasenclever, L., Tomczak, J. M., and Welling, M. Sylvester normalizing flows for variational inference. *arXiv preprint arXiv:1803.05649* (2018).

[7] Bishop, C. M. *Pattern recognition and machine learning.* Springer, 2006.

[8] Boltyanskiy, V., Gamkrelidze, R. V., MISHCHENKO, Y., and Pontryagin, L. Mathematical theory of optimal processes.

[9] Box, G. E., Jenkins, G. M., Reinsel, G. C., and Ljung, G. M. *Time series analysis: forecasting and control.* John Wiley & Sons, 2015.

[10] Chen, S. S., and Gopinath, R. A. Gaussianization. In *Advances in neural information processing systems* (2001), pp. 423–429.

[11] Chen, T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. Neural ordinary differential equations. In *Advances in neural information processing systems* (2018), pp. 6571–6583.

[12] Clenshaw, C. W., and Curtis, A. R. A method for numerical integration on an automatic computer. *Numerische Mathematik 2*, 1 (1960), 197–205.

[13] De Cao, N., Titov, I., and Aziz, W. Block neural autoregressive flow. *arXiv preprint arXiv:1904.04676* (2019).

[14] Dempster, A. P., Laird, N. M., and Rubin, D. B. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological) 39*, 1 (1977), 1–22.

[15] Dinh, L., Krueger, D., and Bengio, Y. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516* (2014).

[16] Dinh, L., Sohl-Dickstein, J., and Bengio, S. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803* (2016).

[17] Duchi, J., Hazan, E., and Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research 12*, Jul (2011), 2121–2159.

[18] Durkan, C., Bekasov, A., Murray, I., and Papamakarios, G. Cubic-spline flows. *arXiv preprint arXiv:1906.02145* (2019).

[19] Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., and Rubin, D. B. *Bayesian data analysis.* CRC press, 2013.

[20] Germain, M., Gregor, K., Murray, I., and Larochelle, H. Made: Masked autoencoder for distribution estimation. In *International Conference on Machine Learning* (2015), pp. 881–889.

[21] Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`.

[22] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. In *Advances in neural information processing systems* (2014), pp. 2672–2680.

[23] Grathwohl, W., Chen, R. T., Bettencourt, J., Sutskever, I., and Duvenaud, D. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367* (2018).

[24] Grover, A., Dhar, M., and Ermon, S. Flow-gan: Combining maximum likelihood and adversarial learning in generative models. In *Thirty-Second AAAI Conference on Artificial Intelligence* (2018).

[25] Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in neural information processing systems* (2017), pp. 6626–6637.

[26] Ho, J., Chen, X., Srinivas, A., Duan, Y., and Abbeel, P. Flow++: Improving flow-based generative models with variational dequantization and architecture design. *arXiv preprint arXiv:1902.00275* (2019).

[27] Hoogeboom, E., Berg, R. v. d., and Welling, M. Emerging convolutions for generative normalizing flows. *arXiv preprint arXiv:1901.11137* (2019).

[28] Hoogeboom, E., Peters, J., van den Berg, R., and Welling, M. Integer discrete flows and lossless compression. In *Advances in Neural Information Processing Systems* (2019), pp. 12134–12144.

[29] Hornik, K., Stinchcombe, M., White, H., et al. Multilayer feedforward networks are universal approximators. *Neural networks 2*, 5 (1989), 359–366.

[30] Huang, C.-W., Krueger, D., Lacoste, A., and Courville, A. Neural autoregressive flows. *arXiv preprint arXiv:1804.00779* (2018).

[31] Huang, X., and Belongie, S. Arbitrary style transfer in real-time with adaptive instance normalization. In *Proceedings of the IEEE International Conference on Computer Vision* (2017), pp. 1501–1510.

[32] Hutchinson, M. F. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communications in Statistics-Simulation and Computation 19*, 2 (1990), 433–450.

[33] Jordan, M. I., Ghahramani, Z., Jaakkola, T. S., and Saul, L. K. An introduction to variational methods for graphical models. *Machine learning 37*, 2 (1999), 183–233.

[34] Kaipio, J., and Somersalo, E. *Statistical and computational inverse problems*, vol. 160. Springer Science & Business Media, 2006.

[35] Karras, T., Aila, T., Laine, S., and Lehtinen, J. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196* (2017).

[36] Karras, T., Laine, S., and Aila, T. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2019), pp. 4401–4410.

[37] Kingma, D. P., and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[38] Kingma, D. P., and Dhariwal, P. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems* (2018), pp. 10215–10224.

[39] Kingma, D. P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., and Welling, M. Improved variational inference with inverse autoregressive flow. In *Advances in neural information processing systems* (2016), pp. 4743–4751.

[40] Kingma, D. P., and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114* (2013).

[41] Kingma, F. H., Abbeel, P., and Ho, J. Bit-swap: Recursive bits-back coding for lossless compression with hierarchical latent variables. *arXiv preprint arXiv:1905.06845* (2019).

[42] Kynkäänniemi, T., Karras, T., Laine, S., Lehtinen, J., and Aila, T. Improved precision and recall metric for assessing generative models. In *Advances in Neural Information Processing Systems* (2019), pp. 3929–3938.

[43] Larochelle, H., and Murray, I. The neural autoregressive distribution estimator. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics* (2011), pp. 29–37.

[44] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. Backpropagation applied to handwritten zip code recognition. *Neural computation 1*, 4 (1989), 541–551.

[45] Ma, X., Kong, X., Zhang, S., and Hovy, E. Macow: Masked convolutional generative flow. In *Advances in Neural Information Processing Systems* (2019), pp. 5891–5900.

[46] Mirza, M., and Osindero, S. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784* (2014).

[47] Miyato, T., Kataoka, T., Koyama, M., and Yoshida, Y. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957* (2018).

[48] Müller, T., McWilliams, B., Rousselle, F., Gross, M., and Novák, J. Neural importance sampling. *arXiv preprint arXiv:1808.03856* (2018).

[49] Oord, A. v. d., Kalchbrenner, N., and Kavukcuoglu, K. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759* (2016).

[50] Paisley, J., Blei, D., and Jordan, M. Variational bayesian inference with stochastic search. *arXiv preprint arXiv:1206.6430* (2012).

[51] Papamakarios, G. Neural density estimation and likelihood-free inference. *arXiv preprint arXiv:1910.13233* (2019).

[52] Papamakarios, G., Nalisnick, E., Rezende, D. J., Mohamed, S., and Lakshminarayanan, B. Normalizing flows for probabilistic modeling and inference. *arXiv preprint arXiv:1912.02762* (2019).

[53] Papamakarios, G., Pavlakou, T., and Murray, I. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems* (2017), pp. 2338–2347.

[54] Park, T., Liu, M.-Y., Wang, T.-C., and Zhu, J.-Y. Semantic image synthesis with spatially-adaptive normalization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2019), pp. 2337–2346.

[55] Rezende, D. J., and Mohamed, S. Variational inference with normalizing flows. *arXiv preprint arXiv:1505.05770* (2015).

[56] Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention* (2015), Springer, pp. 234–241.

[57] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning representations by back-propagating errors. *nature 323*, 6088 (1986), 533–536.

[58] Sajjadi, M. S., Bachem, O., Lucic, M., Bousquet, O., and Gelly, S. Assessing generative models via precision and recall. In *Advances in Neural Information Processing Systems* (2018), pp. 5228–5237.

[59] Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. Improved techniques for training gans. In *Advances in neural information processing systems* (2016), pp. 2234–2242.

[60] Shannon, C. E. A mathematical theory of communication. *Bell system technical journal 27*, 3 (1948), 379–423.

[61] Simonyan, K., and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[62] Sohn, K., Lee, H., and Yan, X. Learning structured output representation using deep conditional generative models. In *Advances in neural information processing systems* (2015), pp. 3483–3491.

[63] Song, Y., Meng, C., and Ermon, S. Mintnet: Building invertible neural networks with masked convolutions. In *Advances in Neural Information Processing Systems* (2019), pp. 11002–11012.

[64] Sunnåker, M., Busetto, A. G., Numminen, E., Corander, J., Foll, M., and Dessimoz, C. Approximate bayesian computation. *PLoS computational biology 9*, 1 (2013).

[65] Tabak, E. G., and Turner, C. V. A family of nonparametric density estimation algorithms. *Communications on Pure and Applied Mathematics 66*, 2 (2013), 145–164.

[66] Theis, L., Oord, A. v. d., and Bethge, M. A note on the evaluation of generative models. *arXiv preprint arXiv:1511.01844* (2015).

[67] Tomczak, J. M., and Welling, M. Improving variational autoencoders using householder flow. *arXiv preprint arXiv:1611.09630* (2016).

[68] Tran, D., Vafa, K., Agrawal, K., Dinh, L., and Poole, B. Discrete flows: Invertible generative models of discrete data. In *Advances in Neural Information Processing Systems* (2019), pp. 14692–14701.

[69] Trippe, B. L., and Turner, R. E. Conditional density estimation with bayesian normalising flows. *arXiv preprint arXiv:1802.04908* (2018).

[70] Uria, B., Murray, I., and Larochelle, H. A deep and tractable density estimator. In *International Conference on Machine Learning* (2014), pp. 467–475.

[71] Van den Oord, A., Kalchbrenner, N., Espeholt, L., Vinyals, O., Graves, A., et al. Conditional image generation with pixelcnn decoders. In *Advances in neural information processing systems* (2016), pp. 4790–4798.

[72] Wehenkel, A., and Louppe, G. Unconstrained monotonic neural networks. In *Advances in Neural Information Processing Systems* (2019), pp. 1543–1553.

[73] Winkler, C., Worrall, D., Hoogeboom, E., and Welling, M. Learning likelihoods with conditional normalizing flows. *arXiv preprint arXiv:1912.00042* (2019).

[74] Zhu, J.-Y., Park, T., Isola, P., and Efros, A. A. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision* (2017), pp. 2223–2232.

Appendix A

Supplementary Results

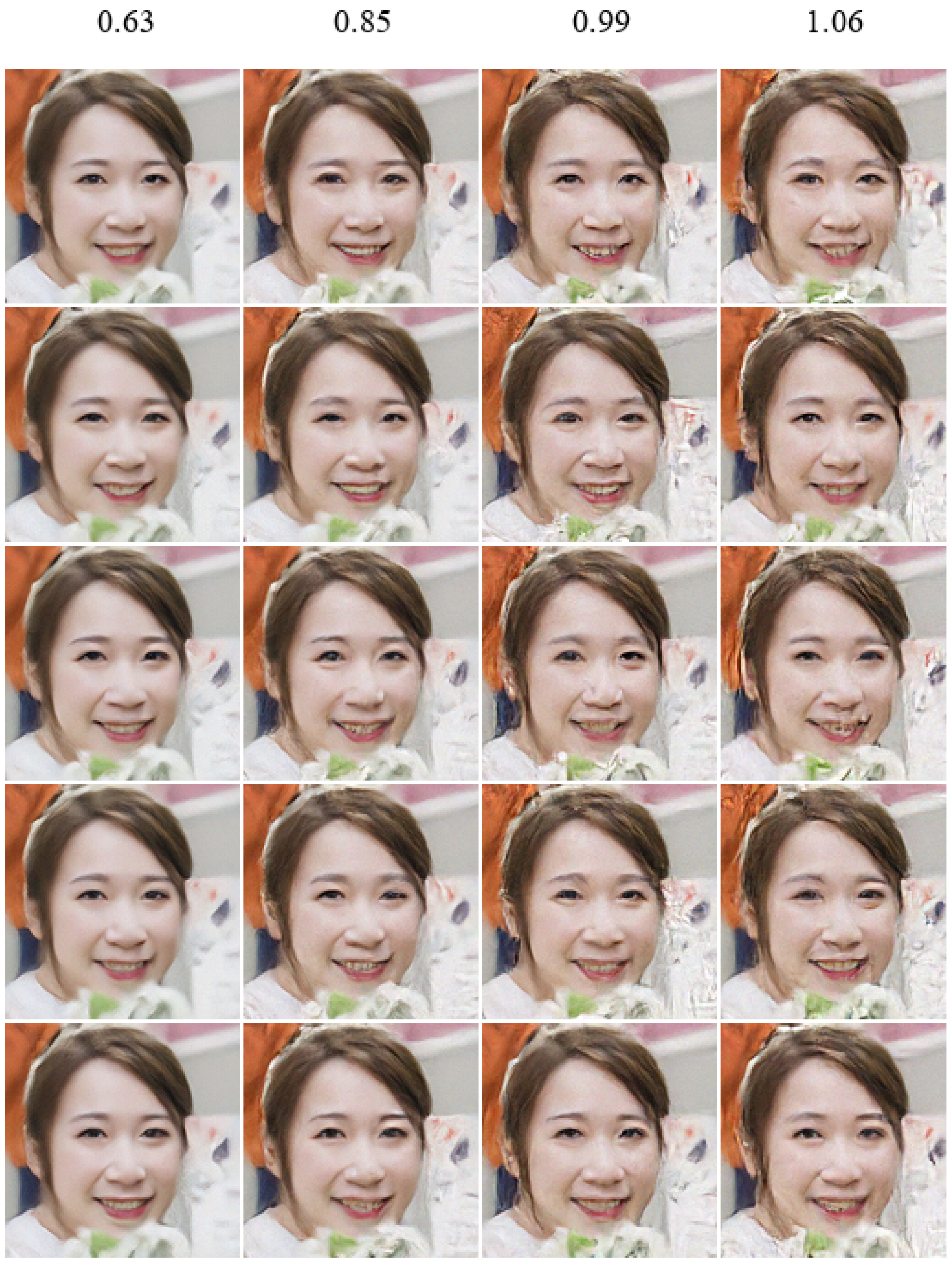


Figure A.1: Effect of sampling standard deviation in the superresolution task. Each row is a different sample from a model with a set sampling standard deviation (columns).

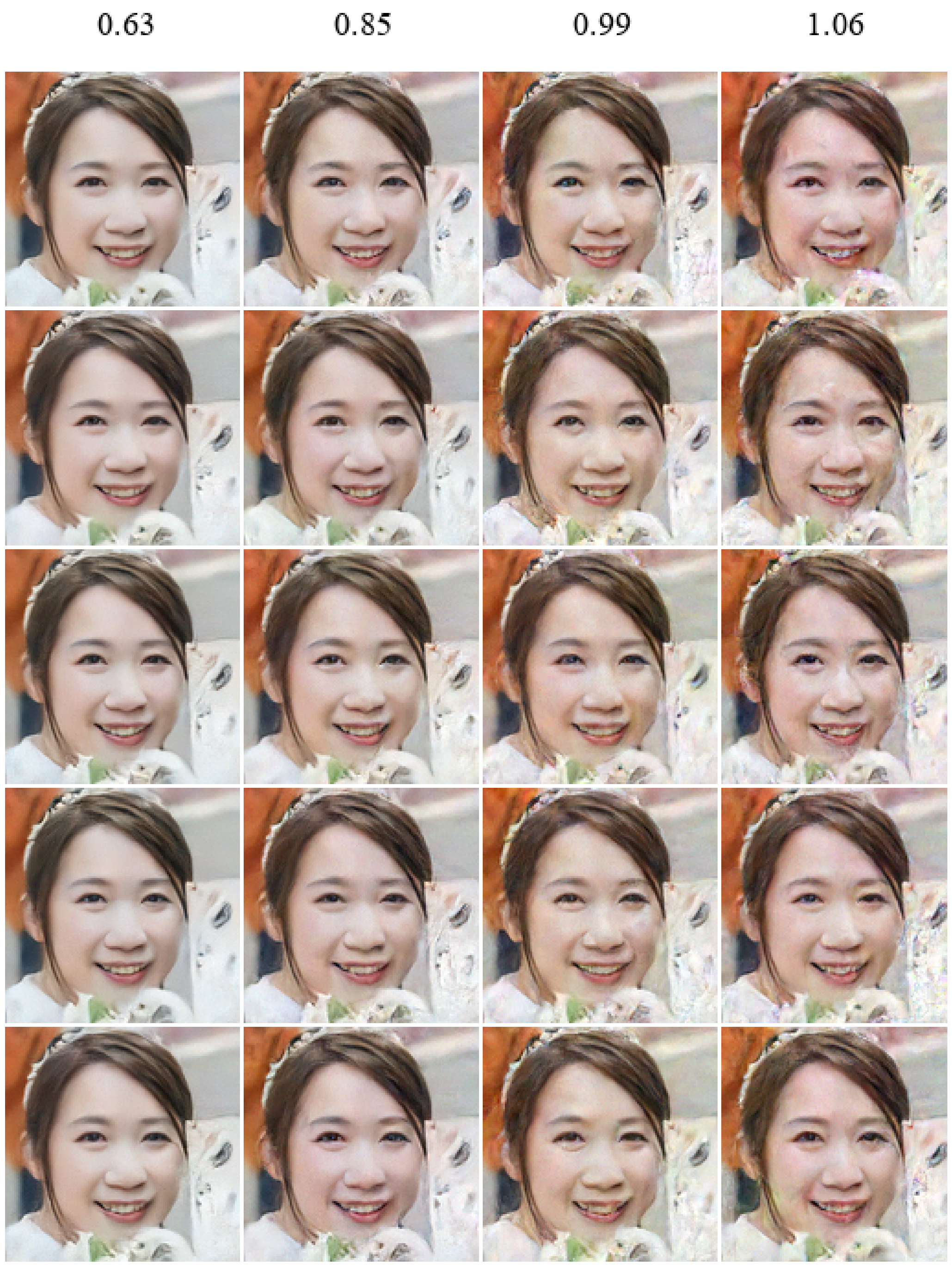


Figure A.2: Effect of sampling standard deviation in the denoising task. Each row is a different sample from a model with a set sampling standard deviation (columns).

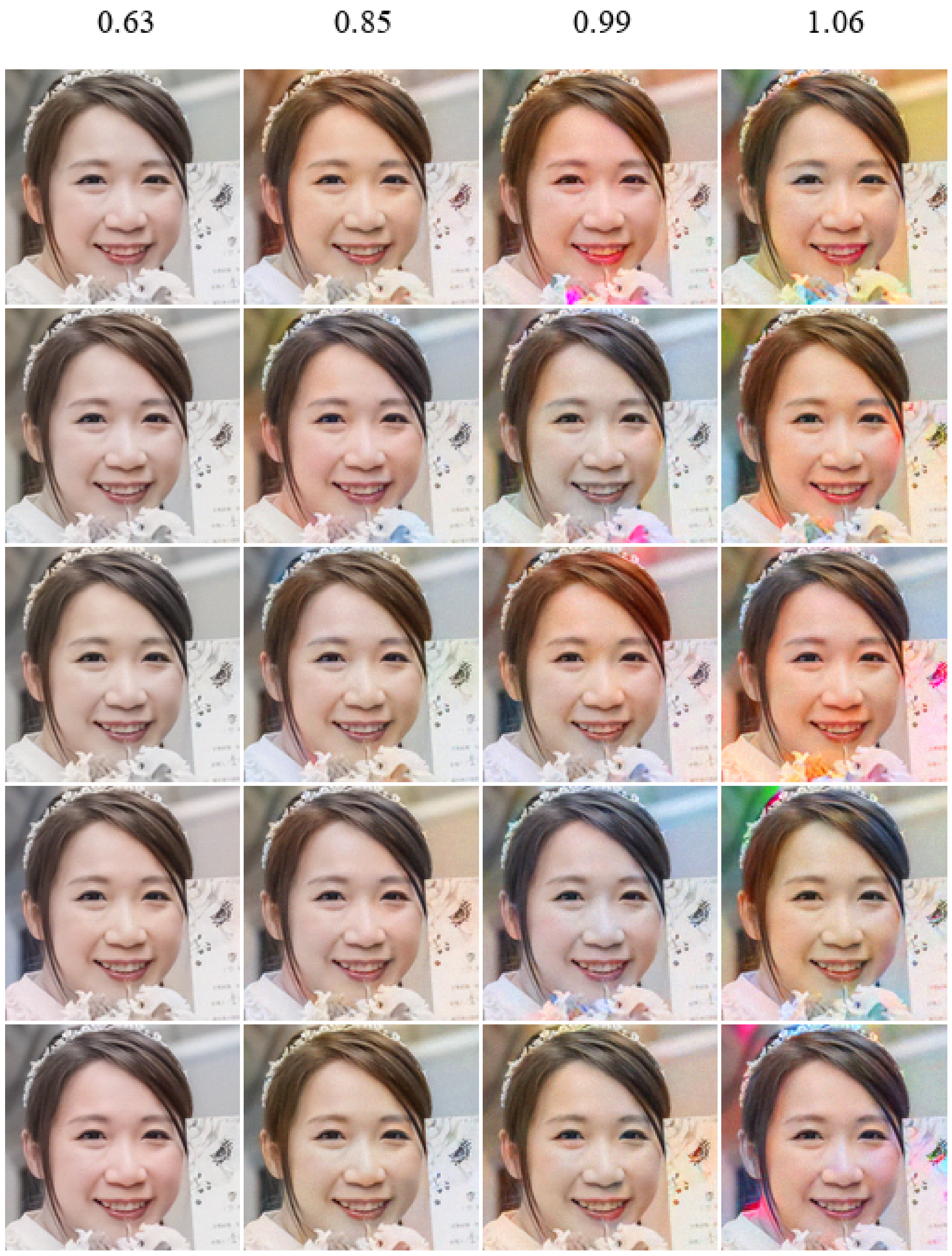


Figure A.3: Effect of sampling standard deviation in the colorization task. Each row is a different sample from a model with a set sampling standard deviation (columns).

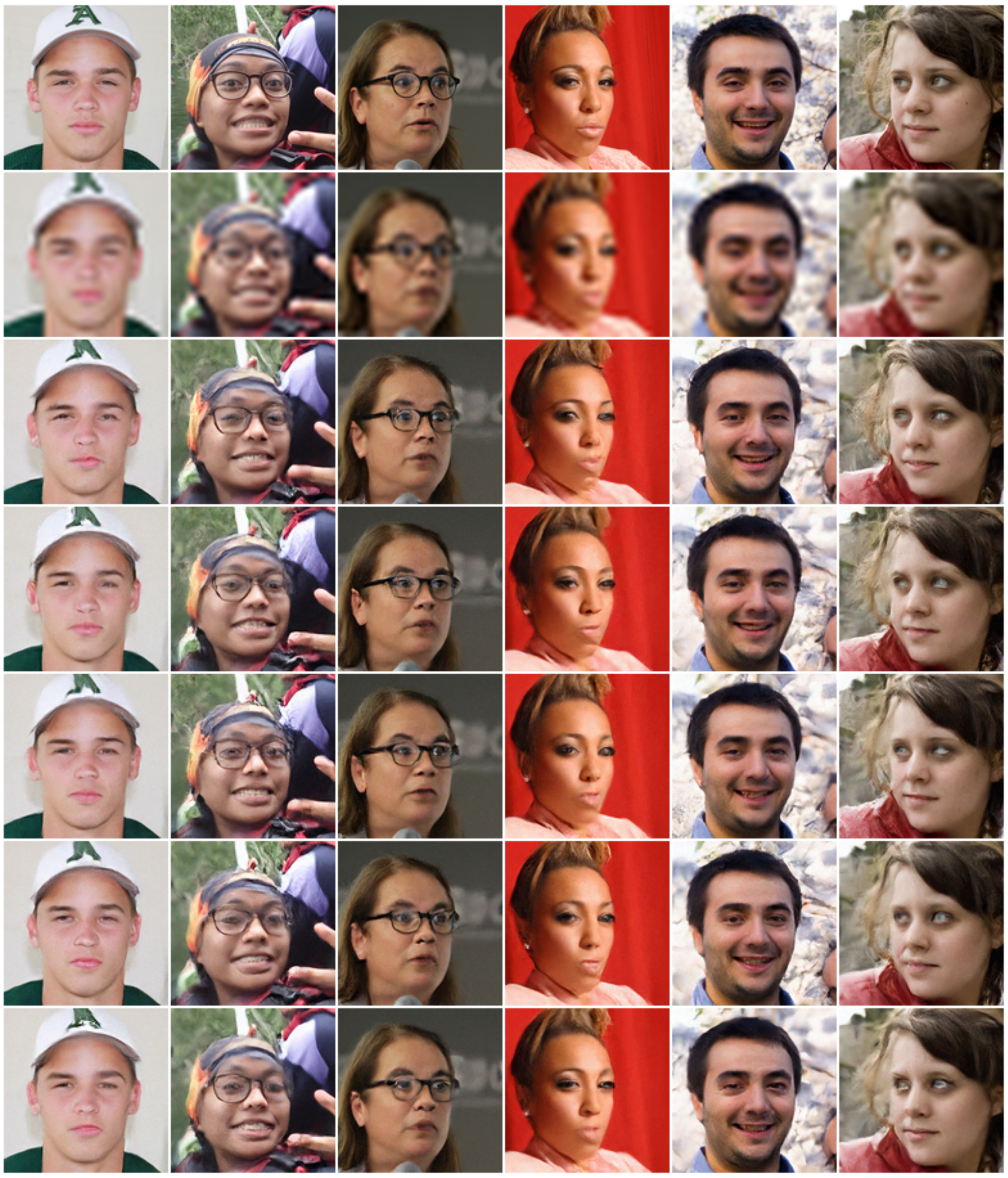


Figure A.4: Additional results for the superresolution task.

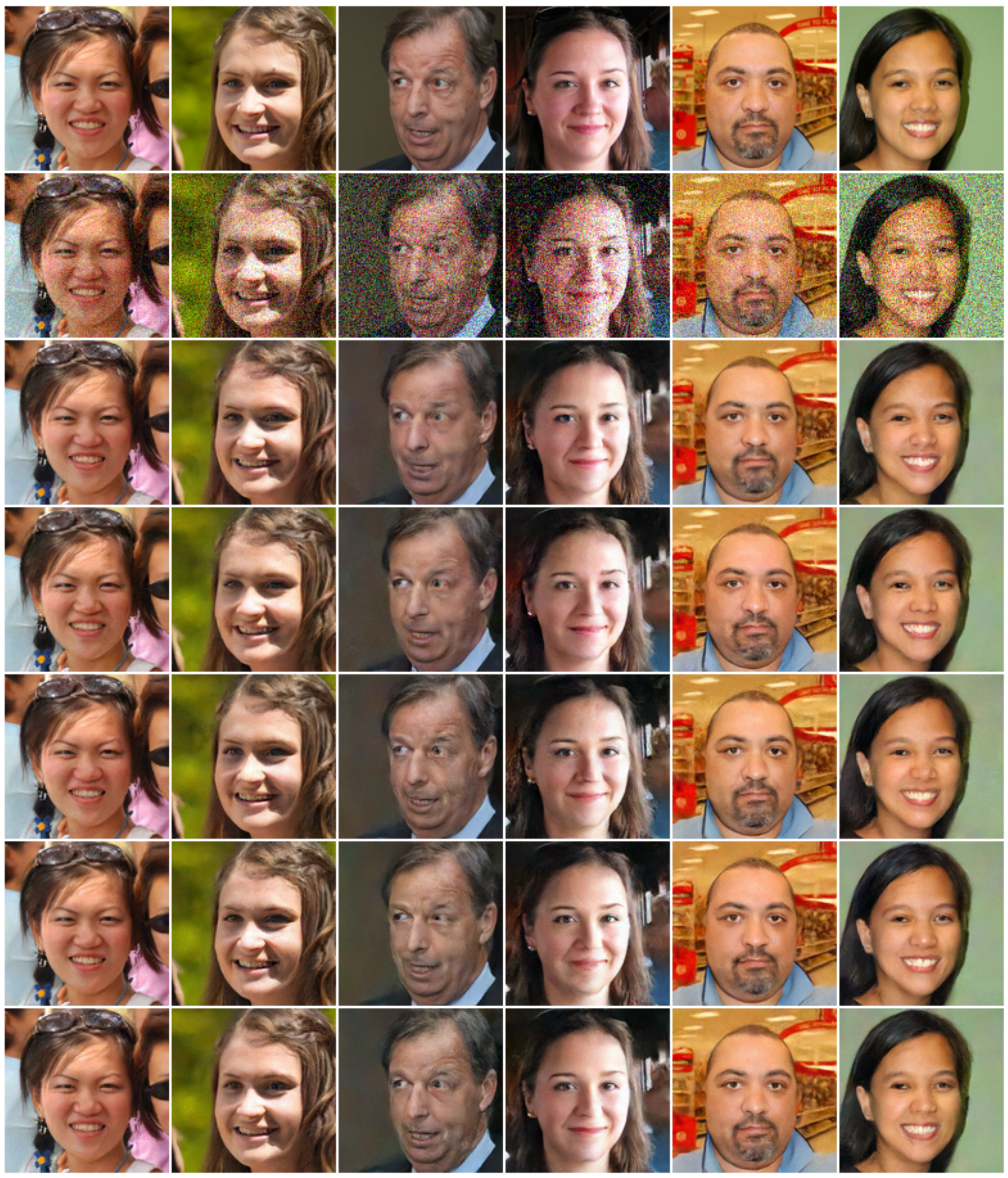


Figure A.5: Additional results for the denoising task.

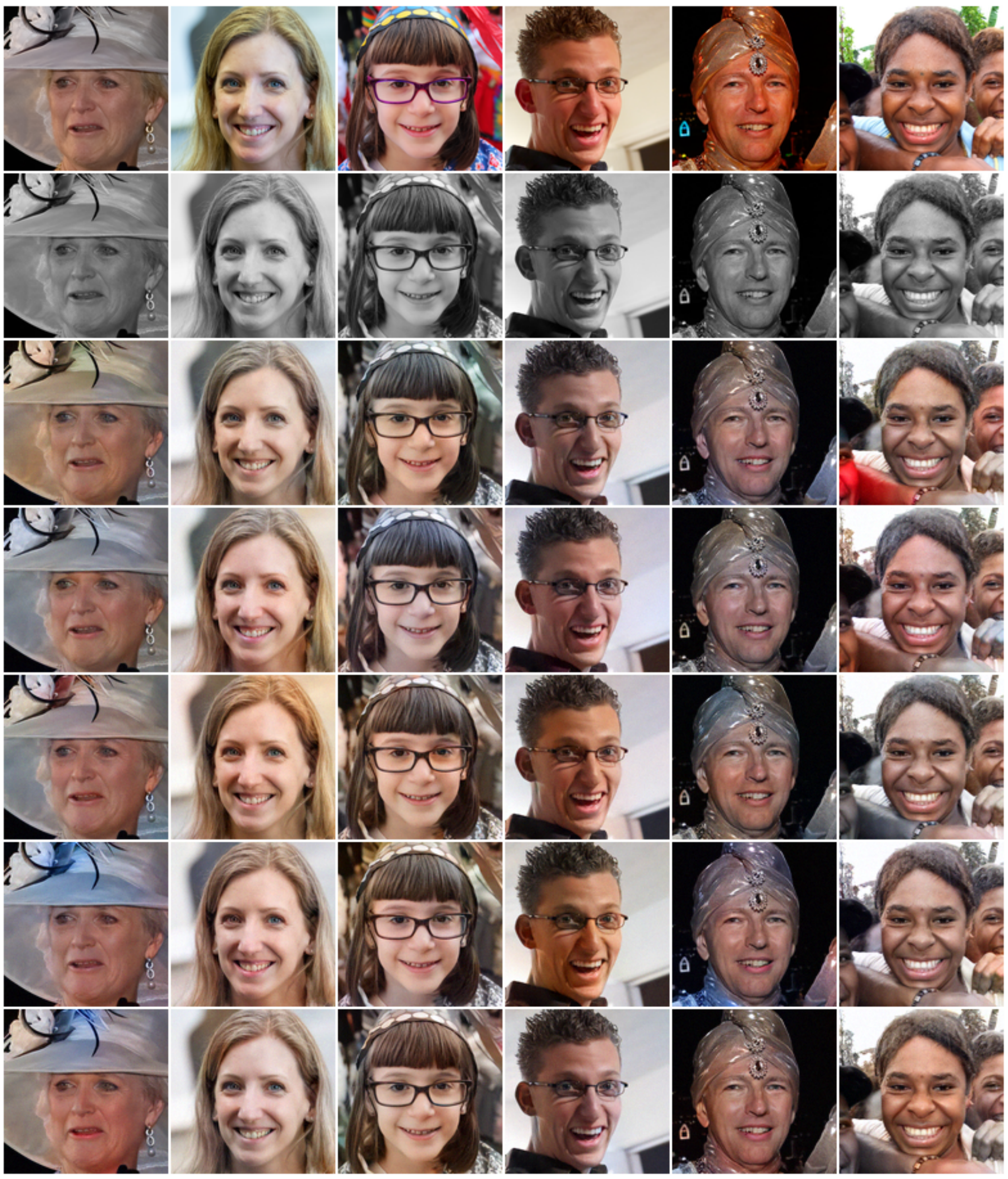


Figure A.6: Additional results for the colorization task.