

# Structure Search for Normalizing Flows

Felix Gonsior<sup>1</sup>, Sascha Mücke<sup>1</sup> and Katharina Morik<sup>1</sup>

<sup>1</sup>Technische Universität Dortmund, August-Schmidt-Straße 1, 44227 Dortmund

## Abstract

Normalizing Flows are deep generative models that allow for feasible exact inference by means of an invertible mapping between a simple prior and an unknown data distribution. Coupling Flows inject the expressive power of neural networks into this framework by allowing conditional transformations, where the conditioner can be any nonlinear function. Under the assumption of feature locality, e.g. in images, the conditional structure has been limited to locality preserving structures. We are interested in cases where the locality assumption does not hold and propose a novel structure search approach based on an evolutionary optimization scheme to find conditional structures. Our method can improve convergence on non-image datasets and lead to smaller models.

## Keywords

Normalizing Flow, Generative Model, Affine Coupling, Deep Learning, Evolutionary Algorithms

## 1. Introduction

Probabilistic inference is a central tool for many applications, like outlier detection [1], image processing [2, 3], gap filling [4] and natural language processing [5], to name a few.

Normalizing flows are a family of probabilistic models that can be used for both inference as well as data generation on continuous data. Their basic idea is to construct an invertible transformation between a tractable (e.g., Gaussian) prior probability distribution and the unknown data distribution. The result is a model that can transform (“flow”) data between the spaces of prior and data distribution. Deep normalizing flows may be composed of a series of simple transformations. A valid probability distribution is maintained by tracking the cumulative volume differentials throughout the flow. Data generation is performed by transforming samples from the prior into the data space. The likelihood of points in data space can be inferred by transforming them into the prior space. In an effort to exploit the high flexibility of deep neural architectures in normalizing flows, *coupling flows* were introduced by Dinh et al. [6, 7]. A coupling flow encapsulates an arbitrarily complex transformation in such a way that the result is invertible.

Across literature, normalizing flows are most commonly used on image data, where features exhibit strong locality properties: Pixels of an image correlate most strongly with neighboring pixels. For this reason, coupling flows that are designed for e.g. image data often use locality preserving masks to determine the conditioning structure. To the best of our knowledge,

---

LWDA’21: *Lernen, Wissen, Daten, Analysen* September 01–03, 2021, Munich, Germany


✉ felix.gonsior@tu-dortmund.de (F. Gonsior); sascha.muecke@tu-dortmund.de (S. Mücke);

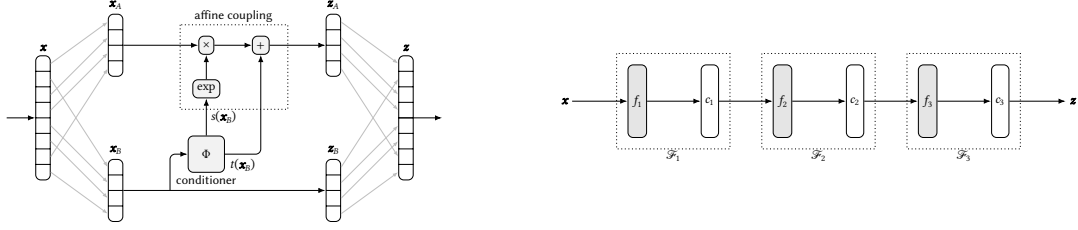
katharina.morik@tu-dortmund.de (K. Morik)

🌐 www.tu-dortmund.de (F. Gonsior); www.tu-dortmund.de (S. Mücke); www.tu-dortmund.de (K. Morik)



© 2021 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)



**Figure 1:** Structure of an affine coupling (left) and layer structure of our flow model (right).

there have been no substantial previous efforts to understand the effect that the structure of a normalizing flow has on convergence and quality of a coupling flow model. Searching the space of connectivity structures in complex, networked models can yield sparse models with few parameters, that still perform nearly as well as dense, hand-crafted models [8, 9, 10]. We devise an evolutionary search procedure that improves a given initial network structure iteratively. It works by generating offspring structures based on the current best guess and accepting only improving structures.

## 2. Background: Normalizing Flows

Given a dataset  $\mathcal{D} \subseteq \mathcal{X}$  with  $\mathcal{D} = \{\mathbf{x}^1, \dots, \mathbf{x}^N\}$ , the goal of unsupervised probabilistic modelling is to estimate the probability density  $p_{\mathbf{X}}(\mathbf{x})$  for any  $\mathbf{x} \in \mathcal{X}$ , such that  $\mathcal{D}$  is a plausible sample of the random variable  $\mathbf{X}$ . We use uppercase letters for random variables and lowercase letters for their respective realizations. Naturally, for  $p_{\mathbf{X}}$  to be a valid probability measure  $\int_{\mathcal{X}} p_{\mathbf{X}}(\mathbf{x}) d\mathbf{x} = 1$  must hold. From now on, we assume  $\mathcal{X} = \mathbb{R}^d$ . Let  $f : \mathbb{R}^d \mapsto \mathbb{R}^d$  be a bijective function with continuous first derivative. We use this function to transform a random variable  $\mathbf{X}$  into another variable  $\mathbf{Z} := f(\mathbf{X})$ . To obtain the density of  $\mathbf{Z}$ , we have to re-normalize the original density  $p_{\mathbf{X}}(\mathbf{z})$  using the change-of-variable formula, as  $f$  generally does not preserve volume:

$$p_{\mathbf{X}}(\mathbf{x}) = p_{\mathbf{Z}}(f_{\theta}(\mathbf{x})) \left| \det \frac{\partial f_{\theta}(\mathbf{x})}{\partial \mathbf{x}^{\top}} \right| \quad (1)$$

Here,  $\partial f / \partial \mathbf{x} =: \mathbf{J}$  is the  $d \times d$  Jacobian matrix with entries  $J_{ij} = \partial f(\mathbf{x})_i / \partial x_j$ . If  $f$  is a composition of transformations,  $f = f_L \circ \dots \circ f_2 \circ f_1$ , the absolute Jacobian determinants are multiplied, where  $\mathbf{z}_i$  is the intermediate result after applying  $f_i \circ f_{i-1} \circ \dots \circ f_2 \circ f_1$ , and  $\mathbf{z}_0 = \mathbf{x}$  and  $\mathbf{z}_L = \mathbf{z}$  accordingly. If we assume that  $p_{\mathbf{Z}}$  is a tractable prior density, e.g. of an isotropic Gaussian, it is easy to see that calculating the density for  $\mathbf{X}$  reduces to calculating the prior density and Jacobian determinant, which acts as a data-dependent scaling factor. If  $f_{\theta}$  is a parametric flow, then a normalizing flow model can be fit to data by minimizing the negative log likelihood:

$$-\ln L(\theta; \mathcal{D}) = -\ln \prod_{\mathbf{x} \in \mathcal{D}} p_{\mathbf{Z}}(f_{\theta}(\mathbf{x})) \left| \det \frac{\partial f_{\theta}(\mathbf{x})}{\partial \mathbf{x}^{\top}} \right| \quad (2)$$

In current literature, the unit of *bits per dimension* (bpd) is often used for comparing results. In the case of batch processing for stochastic optimization it is calculated for a minibatch  $\tilde{\mathcal{D}}$  by

expressing the negative log likelihood in base 2 and normalizing over data dimensions  $d$  and batch size  $|\mathcal{D}|$  as follows:  $\text{bpd} = -(\ln(2) \cdot d \cdot |\mathcal{D}|)^{-1} \ln L(\theta; \mathcal{D})$ . The choice of  $f_\theta$  is the determining factor of both a normalizing flow model’s expressivity and its computational efficiency. For a comprehensive overview see [11].

We use two types of affine flow transformations in our model. *Affine constant flows* consist of an affine transformation  $f(\mathbf{x}) = \mathbf{s} \odot \mathbf{x} + \mathbf{t}$  where  $\odot$  is element-wise multiplication and  $\mathbf{s}$  and  $\mathbf{t}$  are learnable constant vectors in  $\mathbb{R}^d$ . These flows transform each incoming data point by the same linear transformation. *Affine couplings* provide an easy way to design invertible flows from non-linear and non-invertible functions. Couplings are affine transformations that are dependent based on part of the incoming data. They can be built from neural network components, enabling nonlinear behavior of the overall transformation while being invertible wrt. each single data point. Let  $\mathbf{x}$  be an input vector to the coupling flow. It is partitioned into sub-vectors  $\mathbf{x}_A$  and  $\mathbf{x}_B$ . An *affine coupling*  $f$  is an affine transformation in  $\mathbf{x}_A$  and the identity in  $\mathbf{x}_B$ . Its scale and translational components are determined based on *conditioners*  $\mathbf{s}$  and  $\mathbf{t}$  which are computed only from  $\mathbf{x}_B$  (see Figure 1).

$$\begin{aligned} f(\mathbf{x}_A) &= \exp(\mathbf{s}(\mathbf{x}_B)) \odot \mathbf{x}_A + \mathbf{t}(\mathbf{x}_B) \\ f(\mathbf{x}_B) &= \mathbf{x}_B \end{aligned} \quad (3)$$

$$\ln|\mathbf{J}| = \sum_{i=1}^d s_i(\mathbf{x}_B) \quad (4)$$

As both parts of  $f$  are invertible wrt. their parameters, the whole function is invertible. The exponential function is applied to the scaling factor  $\mathbf{s}$ , keeping it greater than zero.

### 3. Conditional Structure Search

The best way to partition the input  $\mathbf{x}$  to a coupling into  $\mathbf{x}_A$  and  $\mathbf{x}_B$  is generally unknown. It poses a similar problem as feature selection, which in itself is NP-hard. The search space is of size  $2^d$  for one coupling and of size  $2^{L \times d}$  for a flow with  $L$  coupling layers.

A naive approach is to use a purely random partitioning, i.e. setting masking bits in a random fashion and thereby using random elements of the input as conditioning variables on each layer of the coupling flow. For data with feature locality, such as image data, coupling flows typically employ a checkerboard structure to ensure that the pixel transformations are conditioned on all neighboring pixels. Between layers the mask is alternated in an odd-even fashion, so that alternating regions of the image are used as conditioning features. This heuristic was developed for the RealNVP flow [7] and has also been used in GLOW [12].

Analogously, one-dimensional data can be partitioned according to index parity, such that in one layer the transformation of features with odd index are conditioned on those with even index, and the other way around in the next layer. We call this type of masking a *parity* masking and it will serve as one of the baselines for our experiments.

Instead of employing another heuristic, our goal is to find a conditional structure that naturally fits the data at hand: We expect that for each specific data distribution there exists a set of conditional structures that lead to better convergence in training as well as to improved performance in inference. The task of uncovering such structures is an optimization problem. Let  $\text{NLL}(\mathbf{x}; \theta, \xi)$  be the negative log likelihood from Equation 2, but parameterized also by  $\xi$  for the conditional structure. Let  $L$  be the number of coupling layers in the flow. For each layer

---

**Algorithm 1: Evolutionary Conditional Structure Search**

---

**Input** : Coupling flow model with  $L$  layers, budget  $T$

**Output**: Improved coupling structure  $\xi$

Initialize  $\xi^0$  and  $\theta^0$  randomly

$\xi^{*,0} \leftarrow \xi^0$

$c^* \leftarrow \text{evaluate}(\xi^0, \theta^0)$

$t \leftarrow 1$

**while**  $t \leq T$  **do**

$\xi^t \leftarrow \text{modify}(\xi^{*,t-1})$

$\theta^t \leftarrow \text{train}(\xi^t)$

$c \leftarrow \text{evaluate}(\xi^t, \theta^t)$

**if**  $c \leq c^*$  **then**

$\xi^{*,t} \leftarrow \xi^t$

$c^* \leftarrow c$

**end**

**else**

$\xi^{*,t} \leftarrow \xi^{*,t-1}$

**end**

$t \leftarrow t + 1$

**end**

**return**  $\xi^{*,T}$

---

$l \in \{1, \dots, L\}$  there is a bit mask  $\xi_l \in \{0, 1\}^d$  that indicates the partitioning through  $x_i \in A \leftrightarrow \xi_{li} = 1$  and  $x_i \in B \leftrightarrow \xi_{li} = 0$ . The size of the partitions  $A$  and  $B$  is fixed. In the following, let  $|B| = m$  be the number of input dimensions that are used for conditioning. The optimization problem we need to solve can be formulated as

$$\arg \min_{\theta, \xi} \text{NLL}(x; \theta, \xi) \quad \text{s.t. } \theta \in \mathbb{R}^p, \xi \in \{0, 1\}^{d \times L}, \forall l \in \{1, \dots, L\} : \sum_{i=1}^d \xi_{li} = m. \quad (5)$$

We obtain approximately optimal solutions to this mixed integer optimization problem by implementing an evolutionary optimization scheme with local search that optimizes  $\xi$  and  $\theta$  in an alternating fashion. The procedure is outlined in algorithm 1.

Structure search begins with a randomly initialized structure  $\xi^0$ . The corresponding  $\theta^0$  is then approximated by stochastic gradient descent on the negative log likelihood (*train*). Depending on the computational resources available, the gradient descent can be terminated early, e.g., after a small fixed number of iterations, to produce a coarse approximation. To evaluate the masking structures we use the negative log likelihood value obtained in the calculation of  $\theta^t$  on the last evaluated batch of training data (*evaluate*). In each iteration  $t \in \{1, \dots, T\}$  we generate a new offspring structure  $\xi^t$  by applying a modification on  $\xi^{*,t-1}$ , the best known structure so far. The modification consists of a random shuffle of layer masks  $\xi_l^t$  for a random selection layers  $l$  (*modify*). The number of layers to shuffle is sampled from a geometric distribution with  $p = 0.5$ . We output the best structure obtained in this way over all iterations,  $\xi^{*,T}$ .

**Table 1**

Overview of datasets that were used in the experiments, with number of features  $d$  and number of samples  $n$  as well as the batch size  $b$ .

dataset	$d$	$n$	$b$	description
<i>magic</i>	10	19020	380	MAGIC Gamma Telescope
<i>turbine</i>	10	22191	443	Gas Turbine CO and NOx Emission Data Set
<i>sonar</i>	60	208	16	Connectionist Bench (Sonar, Mines vs. Rocks)
<i>yeast</i>	8	1484	29	Yeast Data Set

At this point a more exhaustive optimization of the flow parameters  $\theta$  given  $\xi^{*,T}$  can be performed. This is necessary if during the structure optimization run only coarse approximations of  $\theta^{*,t}$  were obtained. We assume that the procedure has converged, if for given number of consecutive structure optimization iterations no improving structure has been found.

## 4. Experimental Evaluation

We evaluated our method empirically in a series of experiments. In order to verify if our proposed structure search yields improvements over common heuristics, we compare the performance of models with different conditional structures. We also vary the number of transformation layers, as the depth of a flow model also influences its expressivity.

To implement normalizing flows and structure search, we used the Python programming language<sup>1</sup> and the PyTorch[13] framework. We used four non-image datasets available on the UCI Machine Learning Repository [14]. The datasets are *magic*, *turbine*, *sonar* and *yeast* (see Table 1). All of them are real-valued, non-image datasets with numbers of features ranging from 8 to 60. Experiments were performed on coupling flow models with depths  $L \in \{2, 4, 6, 8\}$ . All flows are constructed with a repeating layer structure where a layer  $\mathcal{F}_\ell$  consists of an affine transform with constants, followed by an affine coupling flow  $f_\ell$  (see section 2), i.e.  $\mathcal{F}_\ell = c_\ell \circ f_\ell$  (see Figure 1). The entire flow model can then be written as  $\mathcal{F} = \mathcal{F}_L \circ \mathcal{F}_{L-1} \circ \dots \circ \mathcal{F}_2 \circ \mathcal{F}_1$ . For all flows, the prior is an isotropic Gaussian distribution,  $\mathbf{Z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ . As conditioners, we used single-layer neural networks rectified with element-wise tanh in all experiments:  $(s(\mathbf{x}_B), t(\mathbf{x}_B)) = \tanh(\mathbf{W}\mathbf{x}_B + \mathbf{b})$ . Its parameters are  $\mathbf{W} \in \mathbb{R}^{2m \times (d-m)}$  and  $\mathbf{b} \in \mathbb{R}^{2m}$  accordingly. As we cannot assume feature-locality for the datasets at hand, convolutional architectures are not a sensible choice. In all instances, we set  $m = \lfloor d/2 \rfloor$  to use half of the dimensions to condition the other half, in order for our results to be comparable to a *parity* structure.

We performed experiments on three different model layouts, which we will describe briefly. *Parity*: coupling flows have a fixed alternating masking structure, as described in section 3. *Random*: Coupling layers have fixed structure where the masking bits are randomized. *Opt*: Coupling layers have optimized structure, learned through our optimization algorithm. For the structure optimization runs we allowed 200 iterations of structure search, where the loss value of each offspring is taken from the 50th iterate of parameter search with Stochastic Gradient

<sup>1</sup>www.python.org

**Table 2**

Results for the *turbine* dataset. Training losses are given in bpd for differing flow depths. Lower values are better. Best results per row in train and test sets are bold.

depth	train			test		
	random	parity	opt	random	parity	opt
2	694.76 ± 521.66	798.37 ± 536.23	<b>27.16</b> ± 4.81	692.24 ± 519.61	796.22 ± 534.84	<b>26.71</b> ± 4.72
4	432.67 ± 791.98	522.31 ± 440.66	<b>10.81</b> ± 3.31	431.74 ± 791.77	520.53 ± 438.98	<b>10.70</b> ± 3.30
6	514.11 ± 831.73	457.47 ± 980.17	<b>9.20</b> ± 4.90	512.50 ± 831.39	458.50 ± 985.15	<b>9.25</b> ± 4.81
8	153.42 ± 139.00	236.30 ± 318.12	<b>7.12</b> ± 1.88	153.32 ± 139.21	239.35 ± 324.99	<b>7.27</b> ± 1.89

**Table 3**

Results for the *magic* dataset. Training losses are given in bpd for differing flow depths. Lower values are better. Best results per row in train and test sets are bold.

depth	train			test		
	random	parity	opt	random	parity	opt
2	12.73 ± 4.24	15.87 ± 6.92	<b>6.06</b> ± 0.74	13.39 ± 4.53	16.75 ± 7.40	<b>6.24</b> ± 0.80
4	8.94 ± 4.07	11.98 ± 6.52	<b>4.99</b> ± 0.41	9.35 ± 4.30	12.63 ± 6.97	<b>5.05</b> ± 0.42
6	8.98 ± 5.07	8.61 ± 8.61	<b>4.75</b> ± 0.30	9.46 ± 5.44	9.21 ± 10.15	<b>4.79</b> ± 0.30
8	6.31 ± 1.61	6.07 ± 1.07	<b>4.86</b> ± 0.34	6.52 ± 1.71	6.26 ± 1.15	<b>4.93</b> ± 0.36

Descent. Parameter optimization is performed on all models by applying Stochastic Gradient Descent with a cyclic learning rate schedule[15] that enables fast descent into a local optimum. We allowed 1250 iterations for SGD while sampling random batches from the data. By choice of the batch size, the whole dataset is presented to the algorithm within 50 iterations on average. This approximates a training duration of 25 epochs. For each combination of dataset, model and flow depth, 20 model instances were trained. Validation is performed on training and test datasets. For the *turbine* dataset, test data is given. For the other datasets we kept a holdout of 10% of the data points as test set. We report the mean over 20 trained model instances of the mean loss for a given model configuration and dataset. The training performance is measured in bits per dimension (see section 2), as it is in common use in related publications. The experiments have been run on 8 cores of an AMD EPYC 7742 CPU at 2.4 Ghz. The structure search phase at depth 8 took about 1.5 minutes at maximal depth while the postoptimization step using SGD took about 10 seconds.

In Table 2 and Table 3 results for the datasets *turbine* and *magic* are shown. On both of these datasets, the models which use an optimized structure outperform *parity* and *random*, often by more than an order of magnitude. The large means and deviations reported for the *turbine* dataset hint at slower convergence for the *random* and *parity* models. For the *sonar* and *yeast* datasets (see Table 4 and Table 5), the results are much more similar across all tested methods, with the *parity* layout actually having an edge over the other methods. For *sonar*, this can be explained by the fact that the features are taken from sensors that are arranged in a circle, which leads to some degree of locality. This gives *parity* an advantage, which serves to illustrate that this layout performs best on data with local features.

**Table 4**

Results for the *sonar* dataset. Training losses are given in bpd for differing flow depths. Lower values are better. Best results per row in train and test sets are bold.

depth	train			test		
	random	parity	opt	random	parity	opt
2	$-2.10 \pm 0.03$	<b><math>-2.25 \pm 0.00</math></b>	$-2.11 \pm 0.03$	$-2.05 \pm 0.03$	<b><math>-2.20 \pm 0.01</math></b>	$-2.07 \pm 0.04$
4	$-2.56 \pm 0.02$	<b><math>-2.63 \pm 0.02</math></b>	$-2.57 \pm 0.01$	$-2.50 \pm 0.03$	<b><math>-2.57 \pm 0.02</math></b>	$-2.50 \pm 0.01$
6	$-2.76 \pm 0.02$	<b><math>-2.79 \pm 0.02</math></b>	$-2.76 \pm 0.01$	$-2.70 \pm 0.02$	<b><math>-2.73 \pm 0.03</math></b>	$-2.70 \pm 0.02$
8	<b><math>-2.85 \pm 0.02</math></b>	$-2.77 \pm 0.26$	<b><math>-2.85 \pm 0.02</math></b>	$-2.79 \pm 0.03$	$-2.71 \pm 0.25$	<b><math>-2.80 \pm 0.02</math></b>

**Table 5**

Results for the *yeast* dataset. Training losses are given in bpd for differing flow depths. Lower values are better. Best results per row are bold.

depth	train			test		
	random	parity	opt	random	parity	opt
2	$-1.86 \pm 0.26$	<b><math>-2.41 \pm 0.08</math></b>	$-1.82 \pm 0.35$	$-1.98 \pm 0.26$	<b><math>-2.54 \pm 0.08</math></b>	$-1.95 \pm 0.35$
4	$-2.53 \pm 0.38$	<b><math>-2.98 \pm 0.07</math></b>	$-2.66 \pm 0.21$	$-2.64 \pm 0.39$	<b><math>-3.10 \pm 0.07</math></b>	$-2.78 \pm 0.21$
6	$-2.91 \pm 0.19$	<b><math>-3.13 \pm 0.06</math></b>	$-2.99 \pm 0.19$	$-3.02 \pm 0.18$	<b><math>-3.24 \pm 0.06</math></b>	$-3.11 \pm 0.19$
8	$-3.02 \pm 0.26$	<b><math>-3.15 \pm 0.18</math></b>	$-3.08 \pm 0.19$	$-3.12 \pm 0.26$	<b><math>-3.25 \pm 0.19</math></b>	$-3.19 \pm 0.19$

## 5. Conclusion

In this paper, we have taken a first step in the direction of exploring the effect that the conditional structure has on the quality of coupling flow models. We devised a local search heuristic inspired by structure learning that treats the conditional structure of coupling flows as a learnable parameter. It finds improving structures that can lead to better performance on datasets which exhibit no feature locality, compared to heuristically hand-picked structures. We could show that on the *turbine* and *magic* datasets, the resulting models performed significantly better. This leads to models that can be trained faster and that have fewer trainable parameters. On the remaining datasets, the results are not as convincing, as the bpd values are very close to each other and their uncertainty intervals are partly overlapping. Still, they illustrate that (1) conditional structure has a measurable influence on the performance of coupling flow models and (2) the best conditional structure strongly depends on the data at hand. Also, *parity* is evidently a strong heuristic for exploiting feature locality. These results open further avenues of research. Producing results on higher dimensional datasets and with sparser coupling structures opens up the possibility for even greater reduction in the number of parameters. The fact that parity structures work well on data with feature locality suggests that one may be able to derive an optimal structure directly from the data, rendering optimization altogether obsolete. This is something that should be further investigated, as well.

## Acknowledgments

This research has been funded by the Federal Ministry of Education and Research of Germany as part of the competence center for machine learning ML2R (01IS18038A).

## References

- [1] D. Peña, I. Guttman, Comparing probabilistic methods for outlier detection in linear models, *Biometrika* 80 (1993) 603–610.
- [2] S. Z. Li, Markov random field models in computer vision, in: *European conference on computer vision*, Springer, 1994, pp. 361–370.
- [3] G. R. Cross, A. K. Jain, Markov random field texture models, *IEEE Transactions on Pattern Analysis and Machine Intelligence* (1983) 25–39.
- [4] R. Fischer, N. Piatkowski, C. Pelletier, G. I. Webb, F. Petitjean, K. Morik, No cloud on the horizon: Probabilistic gap filling in satellite image series, in: *2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA)*, IEEE, 2020, pp. 546–555.
- [5] J. M. Conroy, D. P. O’leary, Text summarization via hidden markov models, in: *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, 2001, pp. 406–407.
- [6] L. Dinh, D. Krueger, Y. Bengio, NICE: non-linear independent components estimation, in: Y. Bengio, Y. LeCun (Eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*, 2015.
- [7] L. Dinh, J. Sohl-Dickstein, S. Bengio, Density estimation using real NVP, *CoRR abs/1605.08803* (2016). [arXiv:1605.08803](https://arxiv.org/abs/1605.08803).
- [8] K. O. Stanley, R. Miikkulainen, Evolving Neural Networks through Augmenting Topologies, *Evolutionary Computation* 10 (2002) 99–127.
- [9] B. Zoph, Q. V. Le, Neural architecture search with reinforcement learning, in: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, OpenReview.net, 2017.
- [10] E. Real, A. Aggarwal, Y. Huang, Q. V. Le, Regularized evolution for image classifier architecture search, in: *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI, AAAI Press*, 2019, pp. 4780–4789.
- [11] I. Kobyzev, S. Prince, M. A. Brubaker, Normalizing flows: Introduction and ideas, *CoRR abs/1908.09257* (2019). [arXiv:1908.09257](https://arxiv.org/abs/1908.09257).
- [12] D. P. Kingma, P. Dhariwal, Glow: Generative Flow with Invertible 1x1 Convolutions, [arXiv:1807.03039 \[cs, stat\]](https://arxiv.org/abs/1807.03039) (2018). [ArXiv: 1807.03039](https://arxiv.org/abs/1807.03039).
- [13] P. et al, PyTorch, 2019. URL: [www.pytorch.org](http://www.pytorch.org).
- [14] D. Dua, C. Graff, UCI machine learning repository, 2017. URL: <http://archive.ics.uci.edu/ml>.
- [15] L. N. Smith, Cyclical learning rates for training neural networks, in: *2017 IEEE Winter Conference on Applications of Computer Vision, WACV 2017, Santa Rosa, CA, USA, March 24-31, 2017, IEEE Computer Society*, 2017, pp. 464–472.