GlowTorch: A reproduction of Glow

Fredrik Lundkvist
flundkvi@kth.se

Pernilla Wikström
pwikst@kth.se

Harry Vuong
hvuong@kth.se

Abstract

Deep generative models is a field that has received much attention from the machine learning community in recent times. One of the more notable contemporary generative models is *Glow*, a flow-based model with state-of-the-art performance for image generation tasks. In this report, we attempt to implement Glow, and train our implementation on MNIST and CIFAR-10. Our results show on CIFAR-10 are in line with expectations, but MNIST leaves us with some doubts. We also find stabilizing training to be a major challenge.
Keywords: Normalizing flows, Generative models, Glow, Reproducibility

1 Introduction

Deep generative models is a field that has received much attention from the machine learning community in recent times. These models belong to the family of unsupervised techniques, and enable the generation of new data instances. Typically, generative models deal with modeling dependencies of high dimensional input data. Unlike standard discriminative models, which approximate a distribution $P(Y|X)$, generative models instead approximate $P(X)$ – or, in some cases– $P(X|Y)$. Application areas that can benefit from these generative models are, for instance, image editing or speech synthesis, as they enable predicting probabilities of given data. Image editing by deep generative models can turn a facial expression in a portrait from sad to happy. In short, generative models have a variety of possible use cases, with more to be found in the future.

As mentioned earlier, generative models attempt to calculate the *prior* distribution $P(X)$ in some way. However, the exact distribution is often too complex for an exact calculation to be computationally feasible. Flow-based models are a subset of generative methods where calculating the exact prior distribution actually is tractable, unlike other methods, where approximations are used instead. The idea of normalizing flows is to find a relationship between input data x and latent variables z, by forming a sequence of invertible transformations as seen in Equation 1

$$x \underbrace{\leftrightarrow}_{f_1} h_1 \underbrace{\leftrightarrow}_{f_2} h_2 \ldots \underbrace{\leftrightarrow}_{f_K} z \tag{1}$$

where K represents the depth of the architecture. The intermediate functions $f_{1\ldots K}$ are invertible, such that $h_i = f_i(h_{i-1})$ and $h_{i-1} = f_i^{-1}(h_i)$. During data generation, the perfect reconstruction of an image $x = f^{-1}(z)$ can be accomplished by inverting the projection of the latent variables z. Compared to other generative methods like GANs or VAEs, flow-based models have received relatively little attention from the research community. Seminal papers on the subject include NICE [1] and RealNVP[2]. These papers propose and expand a general framework for flow-based models. Glow [4] builds upon these papers, and proposes new layers that fit within the constraints of generative flows. One of the major contributions is the replacement of the fixed permutation layer used in RealNVP with a novel 1x1 convolutional layer; this has the same effect as the permutational layer, but makes the weights learnable, and thus optimizable. Other contributions include using normalization with data-dependent initialization, and affine coupling layers. Glow achieves impressive results, with generated image quality comparable to GANs.

Project report for DD2412 or FDD3412

1.1 Our Work

The aim of this project is to implement a somewhat reduced version of Glow (without conditional sampling) ourselves, using PyTorch[1]. The main goal is to reproduce the quantitative results of CIFAR-10 presented in the original paper. In addition, we attempt to produce quantitative results of MNIST, and generate images from these models. This work does not aim at defining new contexts of deep generative models; it is about learning to use deep learning development tools as PyTorch, and to attempt to implement the model described in the paper *Glow: Generative Flow with Invertible* 1×1 *convolution* [4].

Due to having less time and computational resources on our hands than the authors of the original paper, we do not expect to achieve the same quantitative results, and we do not expect the generated images to be as impressive, since we will not be able to train our model for as long, and will do it with fewer GPUs.

2 Related work

As mentioned in section 1, the main attraction of flow-based models are tractability of exact log-likelihood evaluation, efficient inference and synthesis, as well as more useful latent space [4]. The core of these methods is the *normalizing flow*, explained in [5]. Flow-based models, first proposed with NICE [1], and expanded upon with RealNVP [2] and Glow [4], achieve exact log-likelihood evaluation by using normalizing flows as described in equations 2 and 3

$$\log p_\theta(x) = \log p_\theta(z) + \log |det(\frac{\delta z}{\delta x})| \tag{2}$$

$$= \log p_\theta + \sum_{i=1}^{K} \log |det(\frac{\delta h_i}{\delta h_{i-1}})| \tag{3}$$

Where, as in equation 1, h_i is an intermediate variable in the flow. For conciseness, we define $h_0 = x$, and $h_K = z$. For a specific subset of transformations, namely those whose Jacobian matrix $\frac{\delta h_i}{\delta h_{i-1}}$ are triangular, these values are surprisingly simple to calculate.

As mentioned previously, flow-based models were first proposed in NICE [1]. RealNVP[2] built upon these methods, proposing a number of improvements, most notably the multi-scale architecture, seen on the right in Figure 1. Glow [4] in turn builds upon NICE and RealNVP, proposing novel layers to use within this architecture. These layers are explained in detail in section 3.2. Glow achieves impressive results, both with quantitative measures such as bits per dimension (bpd), and qualitative evaluation of generated images.

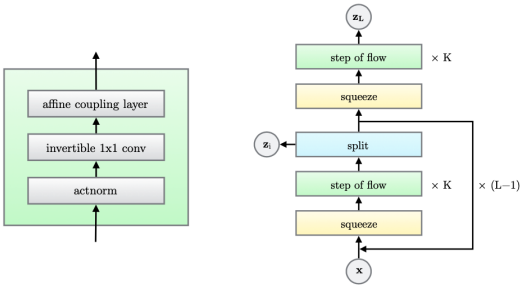


Figure 1: Left, one step of the Glow flow. Right, multi-scale architecture.

[1] https://pytorch.org/docs/stable/torchvision/index.html

3 Methods

3.1 Objective function

We used the same objective as the original paper, which is to minimize the following function:

$$\mathcal{L}(\mathcal{D}) = \frac{1}{N}\sum_{i=1}^{N} -\log p_\theta(\tilde{x}^{(i)}) + c \tag{4}$$

Where $\tilde{x}^{(i)} = x + u$, with u being drawn from a uniform distribution $\mathcal{U}(0, a)$, where a is the discretization level of the data. $c = -M \cdot \log a$, with M being the dimensionality of x. Thus, we are optimizing the parameters θ of some distribution p_θ to minimize the negative log-likelihood objective. The values of $\log p_\theta$ are calculated using equations 2 and 3.

3.2 Network Layers

In this section, we will provide an overview of the steps used, as well as the layers implemented. We will also discuss how our limited resources impacted our experiments. The main contribution of the original paper consists of two new layers suitable for invertible flows:

- Actnorm: a normalization layer using data-dependent initialization,
- Invertible 1x1 convolution: a learnable variant of the permutation layer proposed in [2].

Besides these two, the flow also uses an affine coupling layer, also used in [1] and [2]. In the following subsections, we will describe these layers in detail.

3.2.1 Actnorm

The authors of Glow propose to learn with mini-batches and thereby introducing an activation normalization, that performs an affine transformation of activations by using a scale and bias parameter per channel. Given an initial minibatch of data, these parameters are initialized such that the post-actnorm activations per channel have zero mean and unit variance. After initialization, the parameters are treated as data-independent trainable parameters.

3.2.2 Invertible 1x1 convolution

This layer replaces the fixed permutations used in RealNVP; while functionally equivalent, the 1×1 convolution uses an invertible weight matrix W, initialized as a random rotation matrix, with the network being able to learn the weights of the convolution. Additionally, W is used to compute the log determinant, which is either added or subtracted, depending on the desired direction of movement in the flow. The log determinant can be computed either by using the weight matrix as it is or by using the LU decomposition for faster calculation ($O(n^3)$ vs $O(n)$ complexity).

3.2.3 Affine coupling layer

The affine coupling layers have three important operations: zero initialization, splitting, and permutation. As a step of the affine coupling, there is a non-linear mapping that includes three convolutional layers. The last layer is initialized with zero weights to help training. Though not mentioned in the paper –but implemented in the author's code– weights are initialized using normal sampling and zero bias for the first and second convolution layers. Input data is split in two along the channel dimension. One half is processed by the non-linear mapping, followed by an exponential activation in the paper, which was replaced by a sigmoid activation as in the official code. The purpose for this is to bound the gradients, in order to avoid huge Jacobian determinants that might affect training stability[2]. The processed data is concatenated with the unprocessed into a single tensor. Finally, the permutation step used in RealNVP to reverse the order of the channels is replaced by a 1×1 convolution.

[2] `https://github.com/openai/glow/issues/62`

3.3 Limited resources

The original paper achieved very impressive results, backed by a huge amount of resources and time. While the paper does not specify the amount of training done for the experiments, the code in the provided GitHub repo[3] shows a default training period of 1 million epochs. This amount of training was deemed unfeasible for our project, as we did not have access to the same amounts of GPUs and time as the original authors. Thus, we settled for 100 epochs of training for the CIFAR-10 and MNIST datasets.

3.4 Code

The code used in this project is publicly available at: `https://github.com/Lunkers/DD2412-Project`.

4 Data

Given the large resource requirements of training and evaluating deep generative modeling, the datasets used need to be carefully chosen. In the end, we settled on evaluating our implementation on the MNIST and CIFAR-10 datasets, which were of a more manageable size. The original Glow paper does not present any quantitative results on MNIST; however, quantitative results on MNIST could be found in MintNet [6], for Glow and RealNVP. Hence, we chose to use MNIST in our project, with the additional argument that other datasets (save for CIFAR-10) were too big for us to be able to train and evaluate models for in time.

Quantitative results for normalizing flow models are evaluated on the metric bits per dimension(bpd); the best performing model is –to our knowledge– MintNet, which manages 3.32 bpd. Glow has a close result of 3.35 bpd, while RealNVP achieves 3.49 bpd. Thus, it is reasonable for us to expect to achieve bpd above 3.4, but lower than 4.

4.1 Preprocessing

The data was preprocessed in two steps. Initially, by using the python package torchvision, such that for each training sample (image), the data was translated, rotated, horizontally and randomly flipped, center cropped, and converted to a tensor. Secondly, for each batch, the data was dequantized before the intervention of the model as described in Equation 4. The dequantization is a method to get the log-likelihood of the continuous model on the dequantized data as close as possible to the discrete models log-likelihood, thereby compensating for high likelihoods [7]. For MNIST, all images were padded to be of size 32×32, and converted to RGB, as was done in the code for the original paper [4].

5 Experiments and findings

5.1 Experiment setup

As stated in section 4, we trained models on the CIFAR-10 and MNIST datasets. The models were trained for 100 epochs each; further optimization details can be found in table 1. These networks are

Table 1: Hyperparameters for the models used in our experiments.

Dataset	batch size	flow steps	levels	learning rate
CIFAR-10	16	8	4	10^{-4}
MNIST	16	8	4	10^{-4}

notably smaller than the ones used in the official paper, which used 3 levels of 32 flow steps each [4]. This change was made due to a lack of resources; the original paper trained on 8 GPUs for one week, which was infeasible for our project. We used bits per dimension as our quantitative evaluation

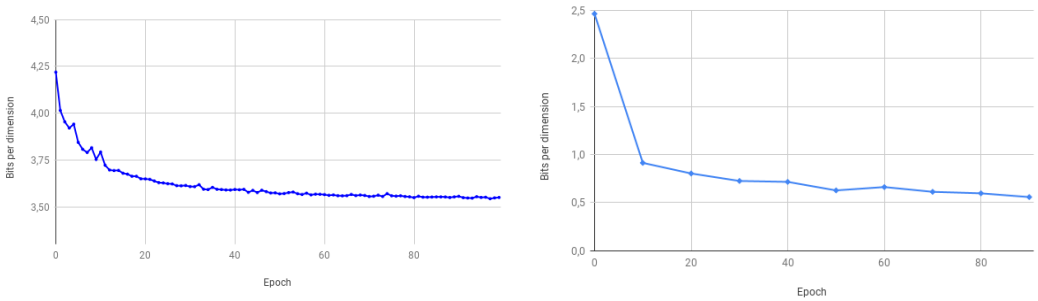
[3] `https://github.com/openai/glow`

[4] `https://github.com/openai/glow/blob/eaff2177693a5d84a1cf8ae19e8e0441715b82f8/data_loaders/get_mnist_cifar.py#L34`

metric, to enable direct comparisons with the original paper as well as RealNVP. We used the ADAM optimizer, [3] with $\alpha = 10^{-4}$, unlike the official implementation, which used a scheduled learning rate starting at 10^{-3}. Aside from learning rate, all optimizer settings were the same as in the original paper. As generative models often are evaluated qualitatively, we also generated samples every 10th epoch, with the final results seen in figure 4. To ensure consistency in weight initialization and enable reproducibility of results, we used the same random seed on both models.

5.2 Results

5.2.1 Quantitative results



(a) (b)

Figure 2: Average Loss in bits per dimension per epoch on the test sets for (a) CIFAR-10, (b) MNIST.

As we can see in Figure 2, the loss stabilized around 3.55 on CIFAR-10 after roughly 30 epochs. On MNIST, the loss values never really stabilized in the same way; we had to add batch normalization to the the nonlinear function in the affine coupling layer for the model to even finish training at all. Because of this, we believe that our MNIST results should be taken with a grain of salt. These doubts are amplified by the quantitative results on MNIST presented[5] in Table 2.

Table 2: Lowest loss values in bits per dimension on CIFAR-10 and MNIST for RealNVP, the original Glow, MintNet, and our implementation.

Model	CIFAR-10	MNIST
RealNVP	3.49	1.06*
Glow (original)	3.35	1.05*
MintNet	3.32	0.98
Glow (ours)	3.55	0.56

(*): data from [6]

As we can see from Table 2, our results of CIFAR-10 are about 0.2 bpd higher than the original implementation. This is in line with our expectations, since we used a much smaller network than the original paper (32 flow steps, compared to 96 in the original). It is plausible that longer training would have resulted in lower bpd, but we do not believe that we would match the results of the original paper, even when training for the same amount of epochs (1 million); since the values stabilized around 3.55, and stayed there during the entirety of training.

5.2.2 Generated images

Every 10th epoch, we generated images from the CIFAR-10 model by taking random samples from the latent space Z with a temperature of 0.7, which was highlighted as a "sweet spot" in the original paper [4]. Samples generated throughout the training process, as well as images generated by the final MNIST model can be found in appendix A.
Images generated by the model trained on CIFAR-10 are shown in Figure 4. Despite the relatively

[5]Quantitative results on MNIST found in MintNet [6]

low amount of training compared to the original paper, one can start to discern somewhat familiar shapes; for example, the second picture from the left on the bottom row looks like a bird if one squints hard enough. These samples show that even though the loss remained relatively stable, the model continuously learned to generate better images.

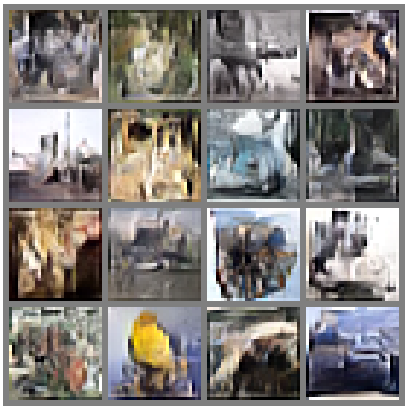


Figure 3: Images generated by the CIFAR-10 model . Sampling temperature 0.7.

6 Challenges

When implementing the model ourselves, we faced a number of challenges. Many implementation details were not mentioned in the paper, and some parts were changed in the code; for example, the affine coupling layer used sigmoid activations rather than `exp`, which was used in the paper. Due to the lack of details in the paper, we regularly had to resort to the source code posted on github [6] to make sure that we were on the right path. However, this was often not very helpful, since the code was not always very intuitive to read. From a reproducibility perspective, it is great that the authors provide their original code, but reimplementation will still be hard unless the code is well-formatted and legible.

The greatest challenge that we faced during this project was achieving stable training. We lost a considerable amount of time trying to remedy this issue, since it would occur quite far throughout the training process. After we discussed this issue with one of the TAs, we got the impression that stability seemed to be highly dependent on hyperparameter values such as amount of flow steps, levels, and learning rate, which was supported by our own attempts. The hyperparameters used in the experiments were the ones that we managed to achieve stable training with. The size and complexity of Glow also made debugging difficult. Luckily, the TA in question was of great help to us here, and we could conclude that our layers were correctly implemented (to the extent of our knowledge), and that hyperparameter sensitivity seemed to be the culprit. These implementation hurdles, combined with our lack of time and resources were the main factors for our choice to experiment on CIFAR-10 and MNIST, since other datasets would have needed even more time to train, which would have made the instability problems even more debilitating.

7 Conclusion

In this project, we have implemented the Glow model in pytorch, and evaluated it on CIFAR-10 and MNIST. The results on CIFAR-10 match our expectations, while we have reservations about our results on MNIST.

This project has given us experience with Pytorch, writing and debugging large and complicated models. As it turned out –perhaps somewhat expectedly–, it was a tough challenge to implement the model, but also to solve the optimization and stability problems that arose. As we expected, the results of our experiments are not as good as those of the original paper, but still within our expectations.

[6] `https://github.com/openai/glow`

There are many possible avenues for future research on flow-based models. In our opinion, investigating what can be done to improve stability across different hyperparameter constellations is one of the most interesting, since it would enable the use of flow-based models for more problems and datasets.

8 Ethical considerations

From an ethical perspective, generative models can be risky due to the exposed danger of deep fakes [8]. As the performance of generative models improves, the harder it will be to distinguish deep fakes from true images. Besides, these synthetic applications enable the possibility of spreading false information. Therefore, in our opinion, generative methods need to be used and provided carefully. Meaning that, if these kinds of applications become available for the society, it might counteract the UN SGD targets [7] of peace and justice (among others). Furthermore, developers should be aware of ethical perspectives when choosing datasets. Likewise to peace and justice, developers should aim for equality, such as gender, ethnicity, and social welfare when contributing to new technology.

9 Self assessment

We believe that this project is deserving of the grade **D**. Glow is a large and complicated model, and implementing it was a major challenge for us. The difficulty of implementation, combined with the restricted amount of resources and time available to us, meant we did not have time to conduct more or deeper experiments. Despite the somewhat lackluster results, we believe that the fact that we managed to implement a lesser version of such a complicated model, replicating basic experiments and produce both quantitative results and generated images makes the project worthy of at least a passing grade.

References

[1] Laurent Dinh, David Krueger, and Yoshua Bengio. NICE: Non-linear Independent Components Estimation. *arXiv:1410.8516 [cs]*, April 2015. arXiv: 1410.8516.

[2] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using Real NVP. *arXiv:1605.08803 [cs, stat]*, February 2017. arXiv: 1605.08803.

[3] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[4] Durk P Kingma and Prafulla Dhariwal. Glow: Generative Flow with Invertible 1x1 Convolutions. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 10215–10224. Curran Associates, Inc., 2018.

[5] Danilo Jimenez Rezende and Shakir Mohamed. *Variational Inference with Normalizing Flows*. 2015. _eprint: 1505.05770.

[6] Yang Song, Chenlin Meng, and Stefano Ermon. MintNet: Building Invertible Neural Networks with Masked Convolutions. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d\textquotesingle Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 11004–11014. Curran Associates, Inc., 2019.

[7] Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. *arXiv:1511.01844 [cs, stat]*, April 2016. arXiv: 1511.01844.

[8] Mika Westerlund. The Emergence of Deepfake Technology: A Review. *Technology Innovation Management Review*, 9(11):39–52, January 2019.

[7] `https://www.un.org/sustainabledevelopment/sustainable-development-goals/`

A Generated images

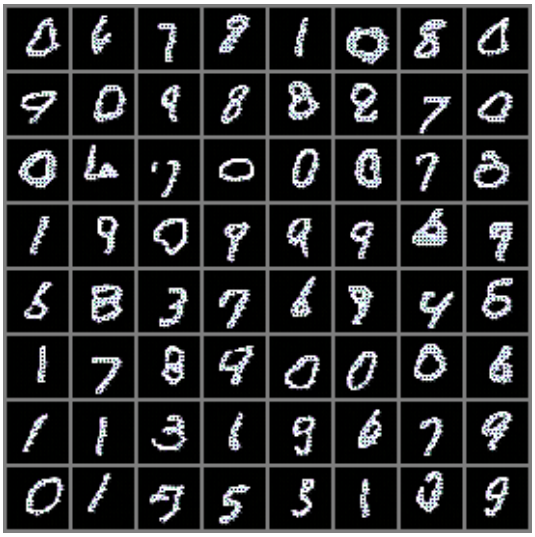
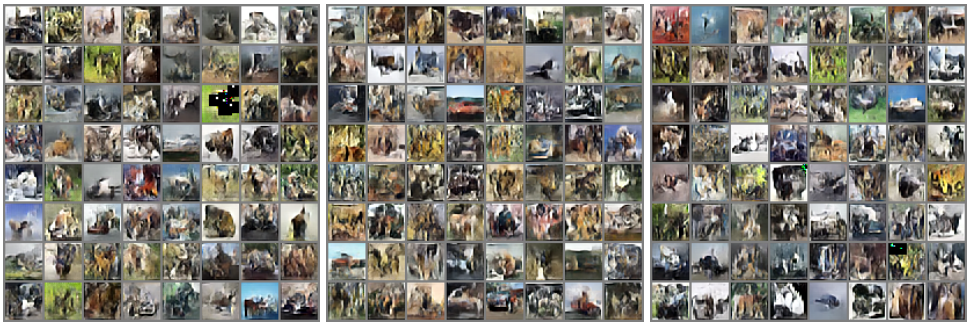
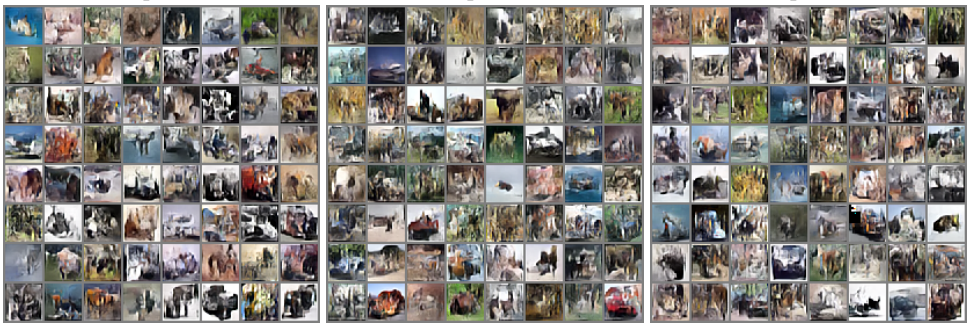


Figure 4: Images generated by the MNIST model. Sampling temperature 0.7.

(a) Epoch 0. (b) Epoch 10.



(c) Epoch 20. (d) Epoch 30. (e) Epoch 40.



(f) Epoch 50. (g) Epoch 60. (h) Epoch 70.

(i) Epoch 80. (j) Epoch 90. (k) Epoch 100.

Figure 5: Image generation evolution of CIFAR-10.