SCALE-SPACE DDPM TRAINING AND INFERENCE

Theo Rüter Würtzen[*]
Department of Computer Science
University of Copenhagen
theo@di.ku.dk

October 30, 2023

Keywords: Denoising Diffusion Probabilistic Model, DDPM, efficiency, compute, hierarchy, scale, decomposition, review, cascade

Abstract

DDPMs as originally presented in 2020 are computationally demanding, both during training and inference. Here, we[2] decomposing the training and inference into two semantically meaningful phases, which imposes an inductive bias into the image representation. Two separate DDPMs, $small$ and $upscaler$ are employed together (in a compound model referred to as $cascaded$) to realize a reverse process over differing scales in image space, akin to image scale spaces. $small$ produces a low-resolution image, while $upscaler$ upsamples it to the desired final size. We compare this approach to a classic single-DDPM generation $baseline$ model, and to the simpler approach of deterministically upscaling samples from $small$ with bilinear interpolation. This final simple approach empirically performs best, followed by the $baseline$ and then $cascaded$. This ranking is however highly dependent on implementation details such as architecture and training procedure, and can therefore only be a gentle hint towards some absolute measure of optimality.

Introduction

Denoising Diffusion Probabilistic Models[HJA20] (DDPMs) have emerged as a powerful class of generative models with diverse applications in machine learning, such as natural language, image, and sound synthesis, and their combined modalities. At their core, DDPMs leverage a 'forward' Markov chain to model a gradual transformation from complex data distributions to a simpler target distribution (usually diagonal covariance multivariate gaussian). In this chain of length T, the data is transformed through a sequence of T states, with each state generated from the previous one through a sample from a forward transition distribution $p(x_t|x_{t-1})$. By learning the reverse transition process $q(x_{t-1}|x_t)$ with a DNN, we can map the target distribution back to the complex distribution of the observed data. Sampling from the complex distribution is achieved through ancestral sampling from the 'reverse' process, starting from a sample from the target distribution and iteratively denoising $T-1$ times to the previous state using the learned q. By spreading the denoising process across multiple successive stages, DDPMs are capable of capturing intricate data dependencies and generating high-quality samples.

Prior Work

Although the theory is promising, harnessing the full potential of DDPMs comes with severe computational challenges. Training these models typically involves a substantial number of iterations[HJA20], and each iteration is be computationally expensive. Additionally, inference, which requires sampling sequentially from the model, can be slow. Different approaches have been presented, some more general than others. Multi-scale signal representations, or pyramid representations, are often employed in computer vision and signal processing to allow for convenient

[*]Accompanying code can be found at https://github.com/Whatisthisname/Scale-space-DDPM
[2]It is really just me, Theo, but I think it sounds neater saying 'we'.

information exchange at different scales of the image. This representation can circumvent some of the limitations involving the inherent locality of pixel-neighborhoods, allowing the model to better cross-reference features that are distant in pixel space. Approaches specific to DDPMs are covered below.
In [Ho+21], the original authors coining the DDPM propose a hierarchical decomposition of the Markov chain. The approach involves dividing the Markov chain into two (or more) distinct regimes – a low-resolution regime and a high-resolution regime. In this hierarchical framework, the low-resolution regime focuses on capturing global features and dependencies within the data, while the high-resolution regime hones in on fine-grained details, aiming to scale up and fill in remaining small local features. The authors of [Zha+22] had similiar ideas but present a signal-processing approach, in which the hierarchy does not consist of semantically meaningful steps, but a compression scheme is employed to reduce dimensionality.
[RY22] is similar to [Ho+21], but emply parameter sharing to increase efficiency and reduce complexity. Using the fact that the convolution operator does not depend on the size of the input signal, they employ the same architecture on all scales and stages of the reverse process. A positional embedding is attached to the image, which allows to model to 'detect' the scale at which it is operating, enabling the convolution operators to specialize to different scales.

In all cases, the motivation behind this hierarchical decomposition is clear – to enhance the efficiency of training and inference in DDPMs. By separating the modeling of global and local information, they aim to significantly reduce the overall computational complexity of the model. Their approaches lead to faster convergence during training. Moreover, it has shown to be a promising way towards improved sample quality, as the hierarchical formulation may make the task simpler. This study aims to reproduce and quantify the effect of a hierarchical decomposition, shedding light on aspects of the architecture, training process and sampling scheme required to enjoy the benefits of such a formulation of the forward process.

Methods

As training and sampling from DDPMs is compute-intensive, we base our results off of our work with the dataset MNIST [Den12] which consists of are 28x28 grayscale images of digits. The dataset consists of more than 50000 784-dimensional(1x28x28) objects, which is more than complicated enough for the publicly accessible GPUs provided by Google Colab [Bis19], which will serve as the provider of computational resources in this study. All DDPMs considered will be trained on the classwise-balanced 50k train subset of MNIST.
The $baseline$ DDPM is inspired in-spirit and adapted from [Guo22] and operates in the native image resolution (28×28 pixels). A U-net[RFB15] learns to map the noisy version of an image to the predicted mean of the added noise, given additional information of the current position in the markov process. The variance of the noise is fixed and given by a cosine-β schedule described in [ND21], with the exponent set to 1.5[3]. The U-net employs residual connections from the encoding stage to the decoding stage, and the timestep and (desired) image label is fed to all the encoding layers by means of a FiLM [Per+17] layer, which maps additional context to an affine transformation, which is applied across channels, pixel-wise. FiLM is a principled way of augmenting a network with additional task information besides the input, in this case the current timestep and desired sample class.

0.1 Experiments

To test whether a hierachical decomposition might be a helpful task formulation we will split the diffusion process into two distinct phases, the first phase We will take place in half the original resolution in $T_1 = 20$ steps, the second in the original resolution in $T_2 = 10$ steps. The joint model will be referred to as the $Cascaded$-model, to acknowledge that the approach followed that presented in [Ho+21], so this paper is partly a reproduction study and partly exploration. In the cited paper, the authors propose sampling full low resolution images $\mathbf{z_0}$ with a standard DDPM architecture (called $small$ in this paper, parametrized by θ) provided timestep t_1 and conditioning label. These low resolution samples $\mathbf{z_0}$ are then bilinearly upscaled to the original resolution (now denoted $\mathbf{z_0'}$) and subsequently fed into a second DDPM (named $upscaler$, parametrized by ϕ), whose training process involved sharpening images from the same distribution as $\mathbf{z_0'}$ to finally yield the MNIST sample $\mathbf{x_0}$. Adapted from [Ho+21], the underlying probabilistic notion of the described aggregate model is preserved as

$$p_{\theta,\phi}(\mathbf{x_0}) = \int p_\phi(\mathbf{x_0}|\mathbf{z_0'})p_\theta(\mathbf{z_0})\ \mathrm{d}\mathbf{z_0}$$

[3]The authors of the referenced paper fix the exponent to 2, but through a preliminary quantitative visual inspection by the writing author, this resulted in a forward process dwelling in the variance regime for longer than needed. The argument for this follows that from [ND21].

We will measure two attributes of the scale-space diffusion model $Cascade$, to estimate whether the inductive bias introduced by this decomposition is helpful. We quantify the sample quality of the generated model and compare it and the sample speed, to that of the $baseline$ and a second approach, using the upscaled samples of $small$, $\mathbf{z_0'}$ directly.

0.2 Quantifying Sample Quality

Sample quality of k generative models will be assessed using Classifier Accuracy Score (CAS) [RV19]. $(k+1) \cdot m$ classifiers are trained to predict the label of images. The first m are trained on "real" data from the true data+label distribution. For the remaining k generative models, m classifiers are trained on data+label sampled from the corresponding generative model. All are then evaluated on an unseen test-set from the real distribution, the results averaged within the m produced classifiers. A good generative model is expected to produce a competitive classifier, while a weak generative model will result in a classifier unable to generalize. The inital m classifiers trained on the real data act as a baseline, to anchor the resulting test-accuracies around the suspected maximum score attainable by the classifier chosen. See the appendix for classifier model description and training regime. In this article, each DDPM is trained three times until convergence, and 3334 samples from it to be collected and shuffled in a final balanced dataset D truncated to size 10000. Then, five convolutional neural networks are trained on D to predict the associated class.

0.3 Quantifying Sample Efficiency

Computational efficiency of the models will be assessed by using wall-clock time of producing a single sample, averaged over 1000 productions. Batching will be controlled for by setting the batch size to $n = 1$. Time spent loading and initializing the model is not tracked, only total time spent in all the iterated forward passes necessary to sample an image.

1 Results

1.1 Comment on Sample Quality

See the appendix for samples generated from each DDPM at the end of each three training session. A preliminary qualitative evaluation by a judge unfamiliar with MNIST showed that the models managed to capture the distribution sufficiently well, as the judge was mostly able to guess the associated label. However, they were also able to discern between real and generated samples with high accuracy, as there are speckles and artifacts left by all DDPMs denoising processes. These speckles can partly be attributed to the choice of loss function, MSE, which may give inexact results as the gradient is close to zero around the occupied optimum. A different loss function that aligns better with our perception may stop this phenomenon from occuring. Additionally, the author did not have time or resources for extensive hyperparameter-search, so all results are presented with some reservation for inaccuracies, as different design choices may lead to improved samples as shows in [Ho+21], which may lopside the final ranking.

1.2 Experiment Outcomes

Figure 1 shows the average classwise accuracy for each classifier trained on all datasets. In all classes, the classifier trained on D=MNIST has the highest accuracy, which is to be expected and reflected in the average accuracy presented in table 1. Deterministically upsampling images generated from the $small$ model yields a significant speedup, taking around $2/3^{\text{rds}}$ of the wall-clock time of the other models. Additionally, the resulting distribution achieves the highest CAS score among all the artificial datasets. Generally, the $cascade$ model performs poorly in comparison to the other models. Since it involves *two* DDPMs $small$ and $upscaler$, the mistakes made along the way are also greatly attenuated, as $upscaler$ is trained on downsampled MNIST images, but during inference receives out-of-distribution samples $\mathbf{z_0}$ from $small$. This distributional shift is conjectured to be responsible for the poor aggregate perfomance, as the $upscaler$ model itself achieved very low loss and visually successful denoising+upscaling on the true distribution. The performance across classes also varies highly; '4's seems to be easy to distinguish and learn, while '5's and '9's are difficult and often confused with other classes. The decomposed $cascade$ approach did not lead to higher sample-efficiency although it denoises $\frac{20 \cdot (14 \cdot 14) + 10 \cdot (28 \cdot 28)}{30 \cdot 28 \cdot 28} = 1/2$ the pixels in as the baseline. This can be due to bottlenecks and latency when transferring from the $small$ to the $upscaler$ model, due to the GPU working more efficiently with images of size 28×28, or due to the fact that the $upscaler$ model requires $\mathbf{z_0'}$ from $small$ as additional input, resulting in more computation.

	$D \sim$ MNIST	$D \sim Baseline$	$D \sim Small$, upscaled	$D \sim Cascaded$
CAS	0.9764 ± 0.0043	0.7300 ± 0.0401	$\mathbf{0.8188 \pm 0.0323}$	0.5997 ± 0.0477
Single-sample (s)	N/A	0.0607	**0.0412**	0.0671

Table 1: The results of the described experiments for the three trained models. The CAS score for a classifier trained on the data D sampled from the respective distributions given in the columns. Sample speed is evaluated by average wall-clok time of generating a single sample on a T4 GPU provided by Google Colab.

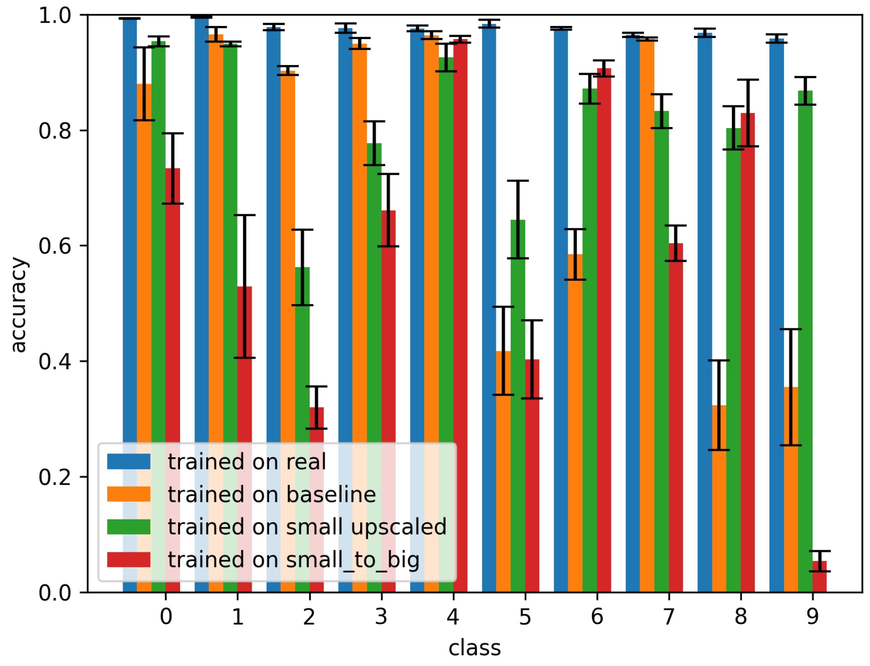

Figure 1: The full CAS results, which shows the classwise average accuracy for each classifier trained on respective samples from DDPM, with `real` representing the true MNIST dataset. `small_to_big` is the dataset generated from $cascaded$, and `small upscaled` is dataset consisting of $\mathbf{z}_0'$s generated from the $small$ model.

1.3 Exploring [& Explaining] Sampling Artifacts

When inspecting the reverse process of the models, some trajectories exhibited curious behaviour: There were localized patches in the samples of the reversed diffusion process that were at time bouncing back and forth between extreme values outside the image domain, see figure 2. To investigate this, we hypothesized that the model was unable to keep its samples within the image domain which is bounded, in contrast to the support of the gaussian forwards process. To investigate, we trained[4] a DDPM on one-dimensional input data, generated from a discrete $p = 0.5$ Bernoulli distribution over -1, 1. Trajectories from the target forward process are visualized in fig. 3, and sampled reverse trajectories are shown in the neighboring fig. 4. The model was quite consistently able to generate samples closely clustered around either -1 or 1, but with a few some deviations, as at times some reverse trajectories ventured between the two main trails, unable to decide on final datapoint. This is described in further detail in the figure. As DDPMs autoregressively bootstrap onto their own output, tiny errors are allowed to accumulate throughout the sampling process. This is an ever present issue in tasks like numerical integration and solving of ODE's. To constrain the trajectory to lie some convex subspace, an optimization construct like the $interiorpointsmethod$ could be used to additionally guide the model prediction. This would be applicable in the convex image domain, but not this toy example. This domain-specific loss would nudge samples into the space they belong to. Another take would be to optimize the n-step error during training of th DDPM, instead of the current standard 1-step error. This would attenutate the accumulating

[4]See the appendix for training procedure and architecture.

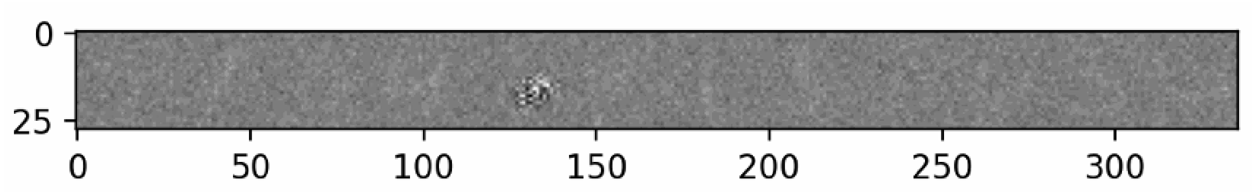

Figure 2: A reverse-process sample of 12 horizontally laid-out images from the *baseline* model. Notice the central high-contrast region in the fifth sample from the left, which motivated us to conduct a further described experiment exploring a 1-dimensional DDPMs.

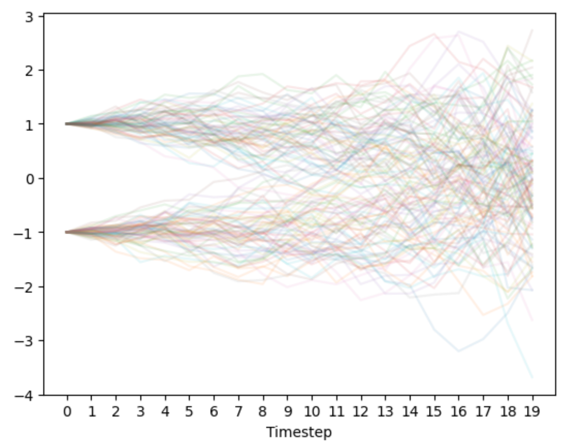

Figure 3: Trajectories from the forwards process. Samples from the scaled an shifted bernoulli distribution at t=0 are gradually randomly perturbed into into samples from a univariate standard gaussian at t=19.

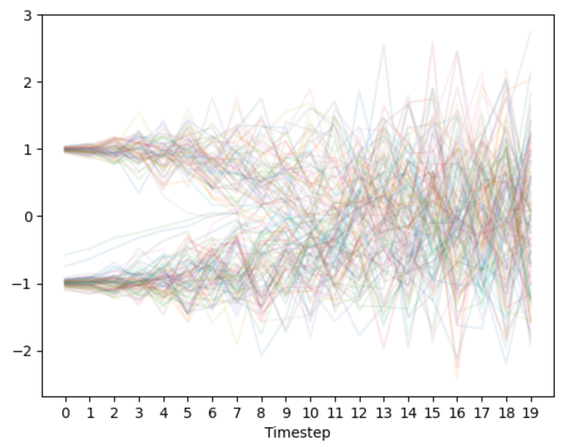

Figure 4: Trajectories from the trained reverse process. Notice the two deviating green and blue trajectories resulting in samples of approx. -0.55 and -0.75, which are far outside the intended support.

errors and allow the model to correct for them. The sub-experiment also underlined another importance when using DDPMs: The complex and target distributions (the marginal distributions at both ends of the diffusion markov chain) should have the same higher moments. When the complex distribution was linearly shifted from {-1, 1} to {0, 2}, many more trajectories landed around 0 than 2, although they should be equally likely. This justifies and motivates the preprocessing of MNIST to have pixelwise mean 0 and variance 1. In conclusion, the experiment strengthens the hypothesis that the high contrast patch is an unusual trajectory through image-space, which we have suggested some remedies for.

2 Conclusion

Scale-space decomposition proved succesful to some degree. DDPMs are able to generate (CAS-wise) useful low-resolution samples describing only the broader relationship between pixels, (yet again) proving that there is a lot of redundant information in images that can be left out at little cost. Operating in this lower resolution is also more computationally efficient, allowing more samples in less time. The attempt to adaptively upscale these smaller images with a second DDPM, reproducing the approach described in [Ho+21] did not turn out to be a success. Generally, DDPMs have proven to be highly sensitive to hyperparameter choices and network initialization, which requires a lot of engineering skill and time to optimize. This sensitivity and the accompanying variance has been controlled for by repeating training procedures and averaging away the outliers to best compare the approaches.

3 Future work

To further explore the limitations and scalability with more complex classes, one could devise an experiment using EMNIST [Coh+17] instead of MNIST. EMNIST, a superset of MNIST, consists of handwritten 28x28 grayscale latin letters and the arabic numerals and is therefore a natural consideration when testing the models capabilities. Questions left open by the presented research are whether the presented approach scales to more complex class-conditional generation. One consideration is whether all letters and digits are distinguishable in half the resolution, which may turn out to complicate the matter.

References

[Den12] Li Deng. “The mnist database of handwritten digit images for machine learning research”. In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.

[KB14] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. eprint: `arXiv:1412.6980`.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. eprint: `arXiv:1505.04597`.

[Coh+17] Gregory Cohen et al. *EMNIST: an extension of MNIST to handwritten letters*. 2017. eprint: `arXiv:1702.05373`.

[Per+17] Ethan Perez et al. *FiLM: Visual Reasoning with a General Conditioning Layer*. 2017. eprint: `arXiv:1709.07871`.

[Bis19] Ekaba Bisong. “Google Colaboratory”. In: *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners*. Berkeley, CA: Apress, 2019, pp. 59–64. ISBN: 978-1-4842-4470-8. DOI: `10.1007/978-1-4842-4470-8_7`. URL: `https://doi.org/10.1007/978-1-4842-4470-8_7`.

[RV19] Suman Ravuri and Oriol Vinyals. *Classification Accuracy Score for Conditional Generative Models*. 2019. eprint: `arXiv:1905.10887`.

[HJA20] Jonathan Ho, Ajay Jain, and Pieter Abbeel. *Denoising Diffusion Probabilistic Models*. 2020. eprint: `arXiv:2006.11239`.

[Ho+21] Jonathan Ho et al. *Cascaded Diffusion Models for High Fidelity Image Generation*. 2021. eprint: `arXiv:2106.15282`.

[ND21] Alex Nichol and Prafulla Dhariwal. *Improved Denoising Diffusion Probabilistic Models*. 2021. eprint: `arXiv:2102.09672`.

[Guo22] `bot66` Guocheng. *MNIST Diffusion*. `https://github.com/bot66/MNISTDiffusion`. 2022.

[RY22] Dohoon Ryu and Jong Chul Ye. *Pyramidal Denoising Diffusion Probabilistic Models*. 2022. eprint: `arXiv:2208.01864`.

[Zha+22] Han Zhang et al. *Dimensionality-Varying Diffusion Process*. 2022. eprint: `arXiv:2211.16032`.

Appendix

3.1 Forward processes from all trained models

Consult figure 5, 6, and 7.

3.2 Samples from all trained models

Samples were generated with the conditional labels $0, 1, \ldots, 8, 9$. Consult figure 8, 9, 10.

U-Net Architecture:

3.2.1 Encoder Block

Our encoder block is parametrized in the following way:

- **Input Channels** (in_ch)**:** This parameter specifies the number of input channels of the block, typically corresponding to the number of feature maps from the previous layer. Initially, this is 1 as the input image is grayscale.
- **Output Channels** (out_ch)**:** Specifies the number of output channels, which determines the depth of feature maps produced by this block.
- **Context Size** (ctx_sz)**:** This parameter indicates the size of the contextual information to be incorporated into the block. If $ctx\_sz > 0$, the FiLM (Feature-wise Linear Modulation) mechanism is employed for contextual information integration. ctx_sz = 11 throughout as explained in the main text.
- **Downsample** (d_smpl)**:** A boolean flag that determines whether the block includes max-pooling for downsampling. If `True`, the block performs max-pooling.

The layers within the Encoder Block are as follows:

- **FiLM Layer:** When $ctx\_sz > 0$, a FiLM layer is initialized an learned to incorporate contextual information into the block.
- **Convolution Layer 1:** A 2D convolutional layer with a kernel size of 3 and padding of 1, which processes the input feature maps.
- **Convolution Layer 2:** Another 2D convolutional layer with the same kernel size and padding as the first convolution layer.
- **Max Pooling:** If d_smpl is `True`, a max-pooling layer with a kernel size of 2 and stride of 2 is applied for downsampling.
- **GELU Activation:** The GELU (Gaussian Error Linear Unit) activation function is used to introduce non-linearity after each convolution operation.
- **Batch Normalization:** Batch normalization is applied after each convolution layer to stabilize and accelerate training.

3.2.2 Decoder Block

Our decoder block is parametrized in the following way:

- **Input Channels** (in_ch)**:** Specifies the number of input channels of the block, typically corresponding to the number of feature maps from the previous layer in the UNET architecture.
- **Output Channels** (out_ch)**:** Defines the number of output channels, determining the depth of feature maps produced by this block.
- **Up-sampling** (up_smpl)**:** A boolean flag indicating whether the block includes up-sampling. If `True`, the block performs up-sampling to increase spatial resolution.

The layers within the Decoder Block are:

- **Convolution Layer 1:** A 2D convolutional layer with a kernel size of 3 and padding of 1, which processes the input feature maps.

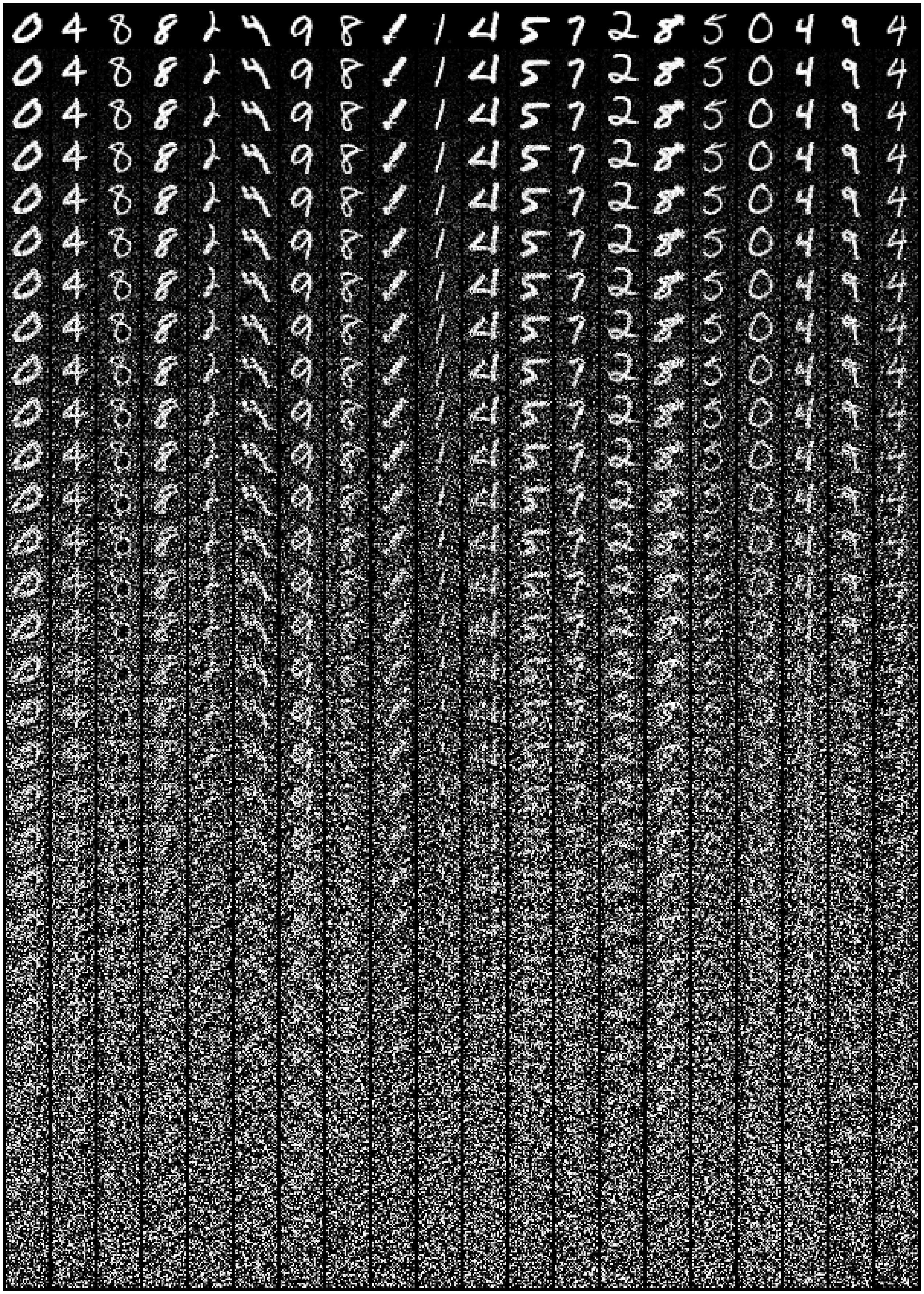

Figure 5: 30 state 28 × 28 forward process for $baseline$.

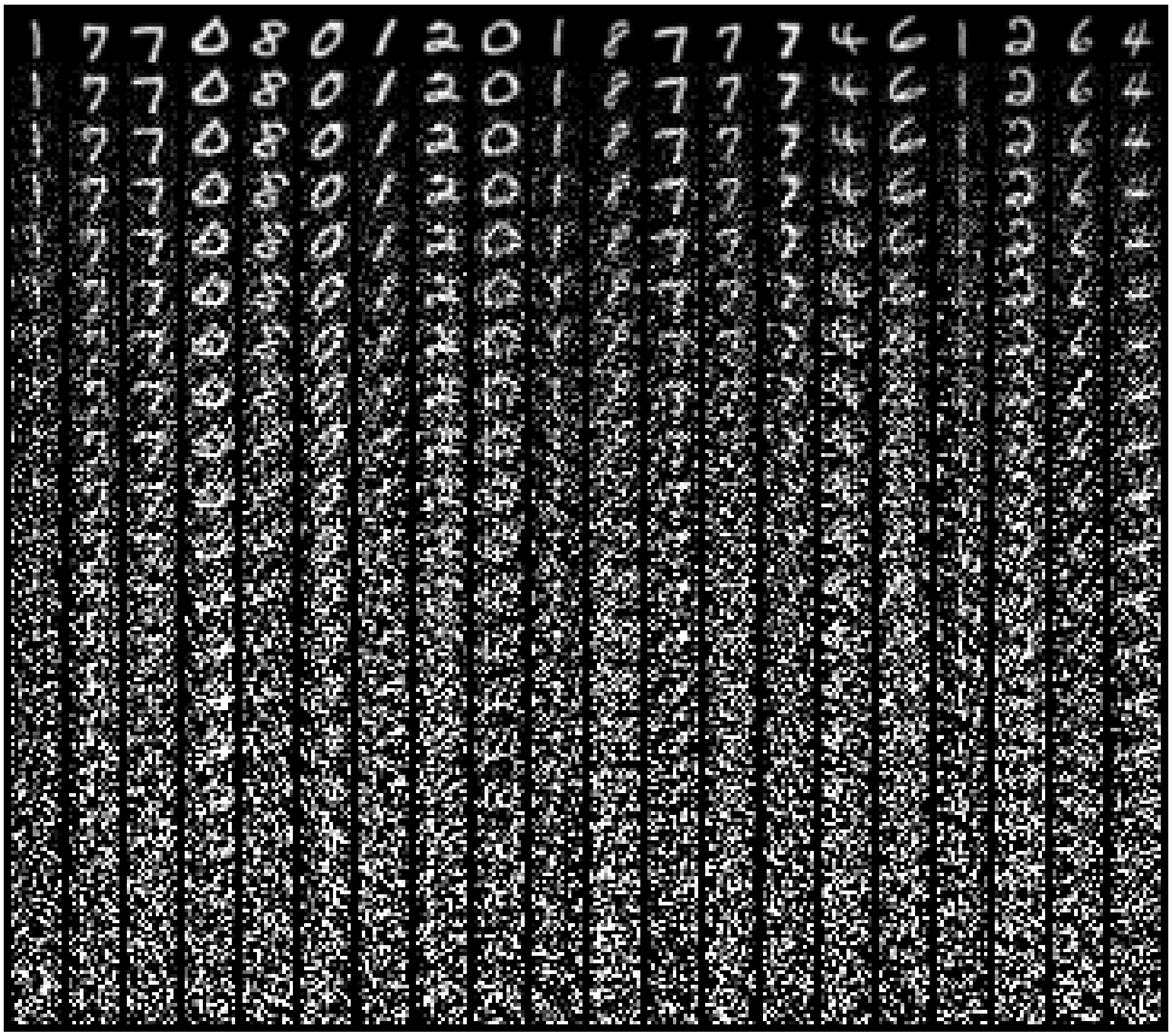

Figure 6: 20 state 14×14 forward process for $small$.

- **Convolution Layer 2:** Another 2D convolutional layer with the same kernel size and padding as the first convolution layer.
- **GELU Activation:** The GELU is used to introduce non-linearity after each convolution operation.
- **Batch Normalization:** Batch normalization is applied after each convolution layer.
- **Up-sampling:** If `up_smpl` is `True`, the block performs up-sampling. It uses bilinear interpolation to increase the spatial resolution of the feature maps to match a specified up-sampling target size, corners are aligned.

3.3 Baseline U-net architecture, $baseline$

The baseline model consists of three encoder blocks followed by three decoder blocks. There is an inner base channel scaling factor of 16 that is doubled after each encoder block and halved after each decoder block but the last such that the feature dimensionality after passing through the U-net is $(batch\_size, 16, img\_size, img\_size)$. The U-net is followed by a final padded 3x3 convolution to yield the ultimate desired output of size $(batch\_size, 1, img\_size, img\_size)$. To be explicit, the channel input/output of the 6+1 modules are:
Encoder 1,2, and 3: $(1 \rightarrow 16), (16 \rightarrow 32), (32 \rightarrow 64)$

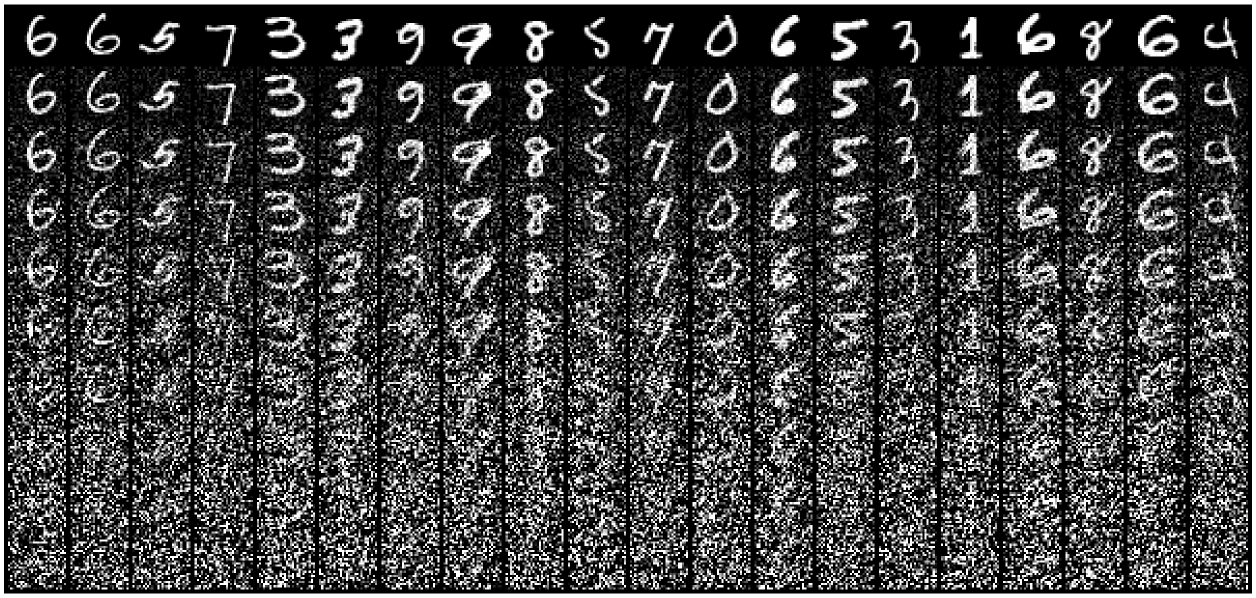
Figure 7: 10 state 28×28 forward process for upscaling, small-image-conditioned second half of $cascaded$. When denoising, this model is conditioned with additional information of the target image in $1/2$ resolution, so it effectively learns to upscale.

Decoder 1,2, and 3[5]: $(64 \to 32), (64 \to 16), (32 \to 16)$
Final convolution: $(16 \to 1)$

3.4 Compund model U-net architecture - low resolution model, a.k.a. $small$

The low resolution model is exactly the same as the Baseline architecture, but it operates on images of size 14x14 instead of 28x28.

3.5 Compund model U-net architecture - upscaling model, a.k.a. $upscale$)

The upscaling model is works like the baseline architecture, but takes 2 channel images as input. These 2-channel images are constructed from a noisy 28x28 image to be denoised stacked along the channel dimension onto the from 14x14 upscaled to 28x28 target context image sampled from the low resolution model.

3.6 Training Hyperparameters for Models in Experiments

3.6.1 General training procedure

During training of all models, we follow algorithm 1 from [HJA20], sampling t uniformly. The data is scaled and shifted to have mean 0 and standard deviation 1 to match the target gaussian first and second moments. The input noise level t is scaled by $1/T$ such that it lies in the unit interval. Image labels are one-hot encoded. This results in a $n \times (1 + 10)$ FiLM context matrix C (for 10-class MNIST):

$$C = \begin{bmatrix} t_1 & c_{1,0} & \cdots & c_{1,9} \\ \vdots & \vdots & \ddots & \vdots \\ t_n & c_{n,0} & \cdots & c_{n,9} \end{bmatrix}$$

where n is the batch size, and $c_{i,j} = \mathbb{1}(\text{image } i \text{ represents digit } j)$. C is provided to the model alongside the noised version of the data distribution both during training and sampling. During training, the label is set to be that of the true image. During sampling, any label can be chosen, and in practice it results in the model producing an image respecting the label. All models are trained to minimize the MSE of the predicted noise to the actual noise used in the forward process. We use Adam [KB14] with the default settings.

[5]The dimensionalities here might be surprising, but the input sizes must be twice as big as the previous output size, as the features of the previous corresponding same-depth encoding layer are concatenated channel-wise onto the input to the decoder.

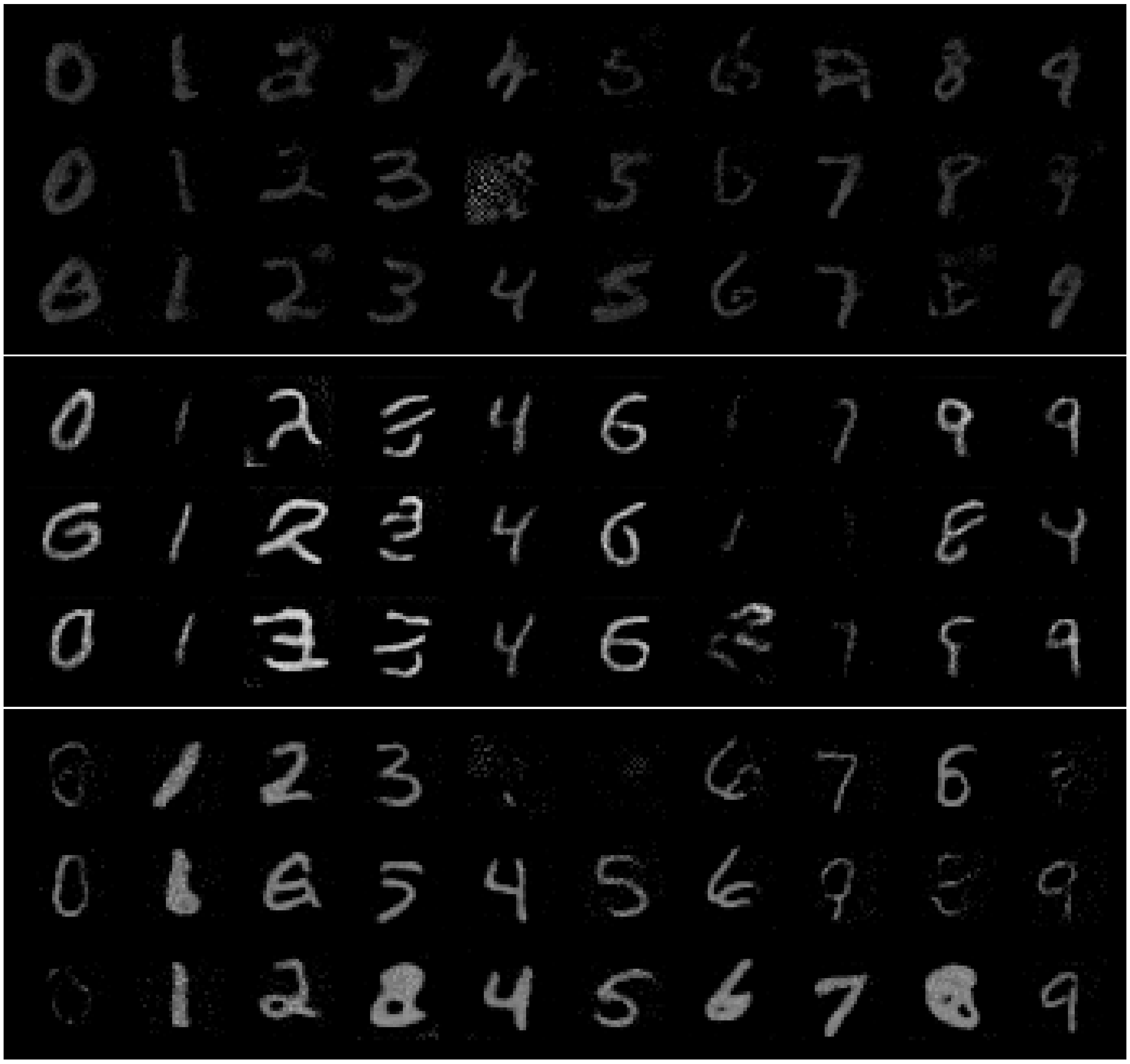

Figure 8: Images sampled from three independent training runs of the $baseline$ model. These images are 28×28.

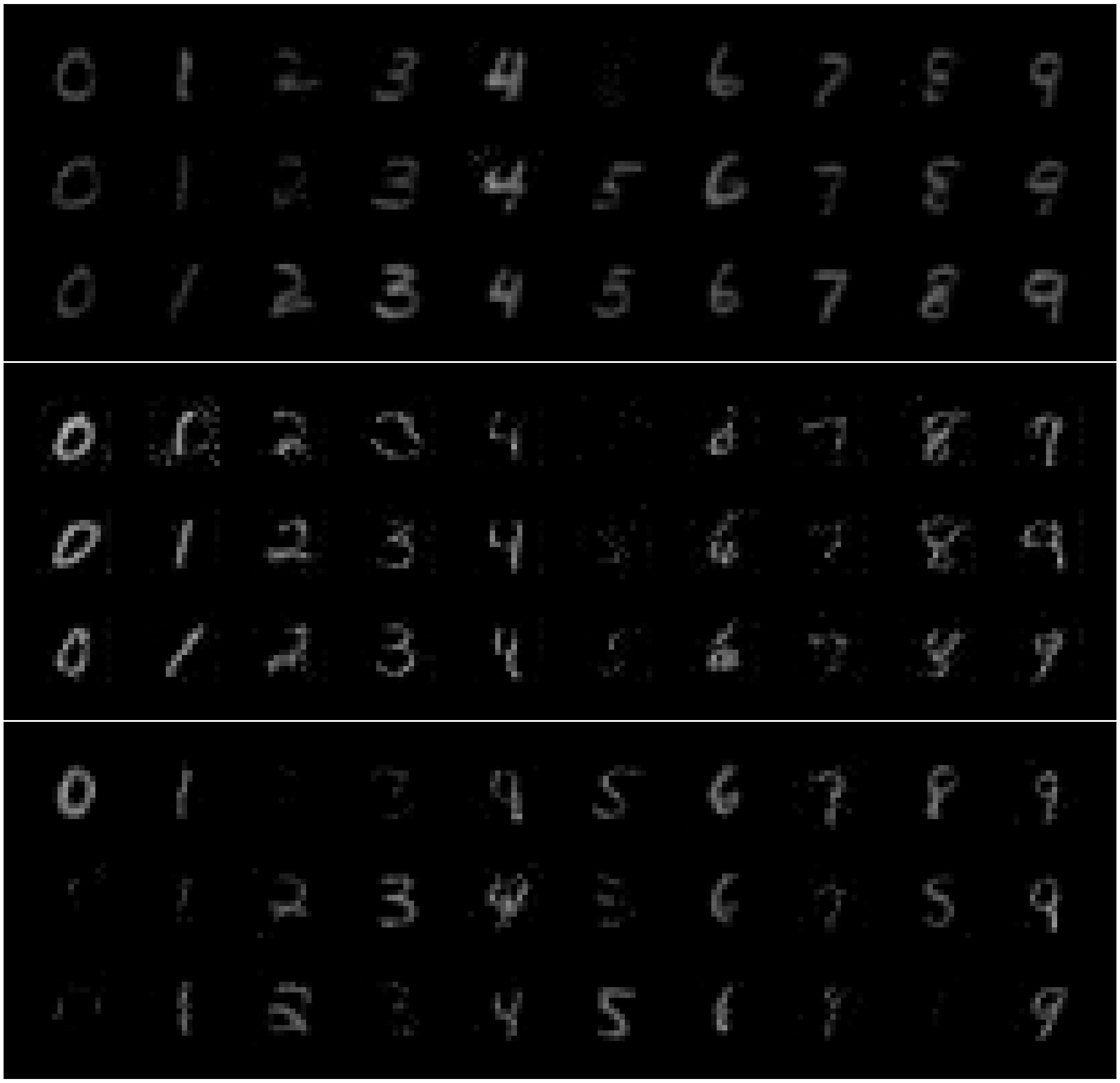

Figure 9: Images sampled from three independent training runs of the $small$ model. These images are 14×14.

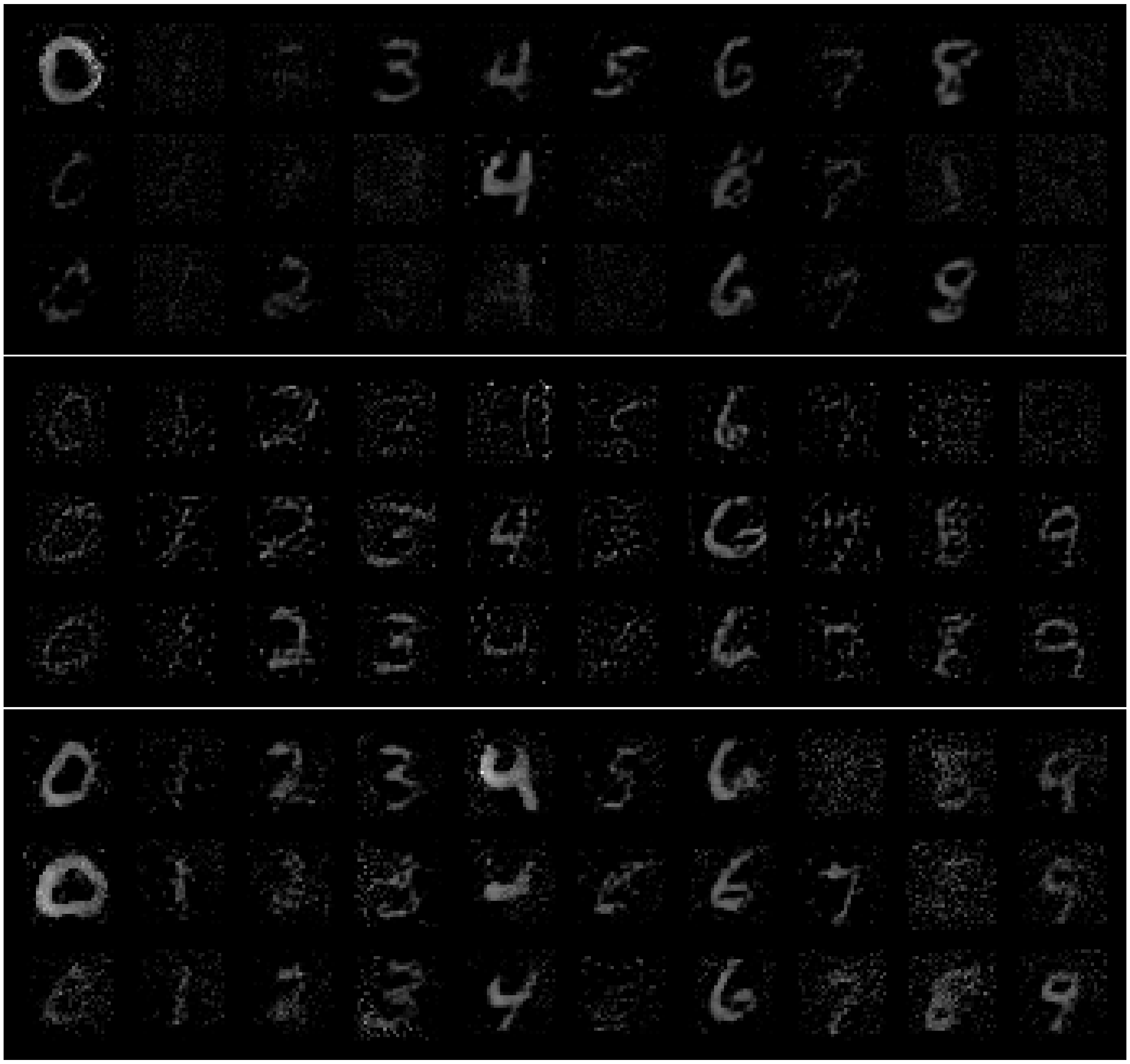

Figure 10: Images sampled from three independent training runs of the *cascaded* model. These images are 28×28.

3.7 DDPM forward process parametrization

The DDPMs $baseline$, $small$, and $upscale$ use respectively 30, 20, and 10 states in their forward process, with a fixed cosine variance schedule as suggested in [ND21], but with the exponent power reduced from 2 to 1.5. $upscaled$ composed with $small$ then also has 30 total states to match the $baseline$ to facilitate a comparison.

3.8 One-dimensional diffusion experiment model and training parameters

The model uses a fully connected (FC) layer ($2 \rightarrow 64$), GELU, FC ($64 \rightarrow 64$), GELU, FC ($64 \rightarrow 1$). As input it receives the current noisy data point and the normalized timestep $t \in \{0, ..., T\}$. For this experiment, $T = 19$ for a total of 20 states in the markov chain. The model is trained with Adam with a learning rate of 0.0001 on 200 epochs of 100×128 uniform samples from $\{-1, 1\}$, with a batch size of 128.

3.9 CAS MNIST classifier model and training parameters

The layers within the MNIST Classifier are

- **Convolution Layer 1:** A 2D convolutional layer with 1 input channel and 32 output channels, using a 3x3 kernel with padding of 1.
- **Convolution Layer 2:** Another 2D convolutional layer with 32 input channels and 64 output channels, again using a 3x3 kernel with padding of 1.
- **Max Pooling:** A max-pooling layer with a 2x2 kernel and stride of 2 is used for down-sampling.
- **Fully Connected Layer 1:** A fully connected layer with 64 * $(img\_size//4 - 1)^2$ input features and 128 output features
- **Fully Connected Layer 2:** The final fully connected layer with 128 input features and 10 output features, which correspond to the 10 possible MNIST digit classes.

Every layer is interleaved with the GELU activation. The forward pass is the following: conv1, pool, conv2, pool, fully connected 1, gelu, fully connected 2, which yields logits for the ten classes.