

Machine Visual Perception Course Project Report

Newer Architectures for Diffusion

Authors: Kai Cao, Haotian Shen, Haolong Wang

Group number: 5

Content:

Chapter 1: Introduction and Motivation	3
Section 1.1: Introduction to the problem [Author: Haolong]	3
Section 1.2: Background and related work [Author: Haolong]	3
Section 1.3: Overview of the idea [Author: Haolong]	4
Chapter 2: Method	5
Section 2.1: Baseline architecture [Author: Kai]	5
Section 2.2: Architectural improvements [Author: Haotian and Kai]	7
Section 2.3: Implementation details [Author: Haotian and Kai]	11
Section 2.4: Data pipelines [Author: Kai]	13
Section 2.5: Training procedures [Author: Kai]	14
Section 2.6: Testing and validation procedures [Author: Haotian]	14
Chapter 3: Experiments and Evaluation	16
Section 3.1: Datasets [Author: Kai]	16
Section 3.2: Training results [Author: Kai]	16
Section 3.3: Qualitative results [Author: Haotian]	17
Section 3.4: Quantitative results [Author: Haotian]	21
Section 3.5: Comparison to state-of-the-art [Author: Kai]	21
Chapter 4: Conclusions and Future Directions	23
Section 4.1: Conclusions [Author: Kai]	23
Section 4.2: Discussion of limitations [Author: Kai]	23
Section 4.3: Future directions [Author: Kai]	23
Section 4.4: Contributions of team members	24

Chapter 1: Introduction and Motivation

Section 1.1: Introduction to the problem [Author: Haolong]

Producing images and artwork using artificial intelligence is always an interesting topic. A range of generative models based on neural networks has been proposed. Generally, those methods can handle tasks such as image-to-image translation, semantic segmentation, classification, etc.

Among the different models, diffusion-based neural networks have shown promising results in many image-to-image translation tasks. The denoising diffusion probabilistic model (DDPM) is a good example [1]. The model mainly consists of three parts (noise scheduler, neural network, timestep encoding) and two processes (forward and backward). In the forward process, noises are gradually added to the images according to the encoded timestep. The backward process gradually removes the noises and is modelled by a relatively simple UNet [2].

In this project, we are going to discuss the performance of different neural networks used in the backward process. This is because other newly proposed networks have beaten the UNet in various machine perception tasks. New models can be applied to the diffusion model to improve the performance against a variety of metrics and test sets. The effects of different network architectures are also evaluated, which can help us better understand the mechanism of the diffusion model.

Section 1.2: Background and related work [Author: Haolong]

DDPM has two processes based on the Markov chain. All the states in both processes have the same dimensionality. The forward diffusion process adds random Gaussian noises to the images. This process can be captured by a compact equation:

$$q(x_t|x_0) = N(x_t; \sqrt{\bar{\alpha}_t}x_0, (1 - \bar{\alpha}_t)I) \quad (1)$$

, where x_t and x_0 are the images at time step t and 0 respectively, $\bar{\alpha}_t = \prod_{s=1}^t(1 - \beta_s)$ is the cumulative of the one minus the noise schedule $\beta_1 \dots \beta_t$, I is the identity matrix, and N is the normal distribution. The images become chaotic and full of noises after iterations. In the backward denoising process, the models recover the images from totally random noises. UNet is trained to predict the Gaussian noise at a specific time.

UNet is a convolutional neural network that works well with image regression tasks. The network mainly consists of two parts: encoder and decoder. The network reduces the size (spatial information) of input data and extends its feature information [2].

Many other neural networks specialized for machine perception and image processing have been developed. A few examples are listed: DDPM_AdditionalResLink, DDPM_RDN, DDPM_Deep, DDPM_ConvNeXt, DDPM_4ResBlocks, DDPM_BigGAN.

Section 1.3: Overview of the idea [Author: Haolong]

The DDPM can be improved in many different aspects. In this project, alternative solutions to the UNet are explored. The UNet can be completely replaced by other neural networks. It is also beneficial to augment the simple UNet with ideas from more advanced architectures such as ResNet and ConvNeXt.

Recent research has integrated such ideas into diffusion networks, but there still lacks a comprehensive analysis of the effect of different architectures. Our project aims to compare the improvements brought by these architectures and bridge the gap toward a better understanding of diffusion models.

Chapter 2: Method

Section 2.1: Baseline architecture [Author: Kai]

Our baseline is a simple UNet architecture, as described in the DDPM paper [1]. For its implementation, a GitHub repository¹ by @dome272 was adapted to this specific task. Our code can be found at: <https://github.com/kaicao233/mvp-diffusion> (a private repository, please contact us if you need access).

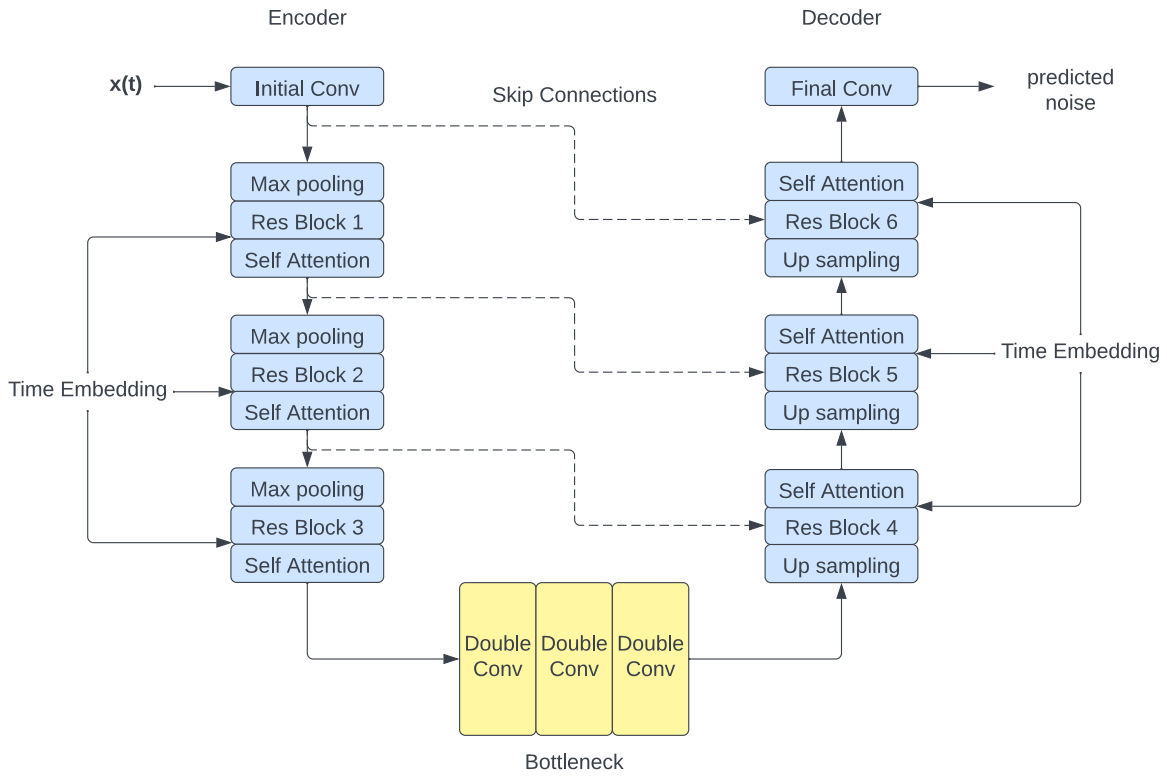


Figure 1: illustration of our baseline UNet architecture

As described in Chapter 1, this UNet takes the noised image $x(t)$ at timestep t and predicts the noise that is added at this timestep in the forward process. Its architecture is shown in Figure 1. It mainly consists of an Encoder (the down-sampling process), a Decoder (the up-sampling process), and the bottleneck in between. The down/up-sampling process can be further divided into 3 chunks respectively, each made up of a max pooling/upsampling layer, a residual block, and a self-attention layer. Each max pooling layer reduces the image resolution by half, and each up-sampling layer doubles the resolution. The residual block is composed of two double convolution blocks, where only the first one has a residual link (as shown in Figure 2). From the output images of size 32×32 , 16×16 and 8×8 during down-sampling, three skip links are connected to the input of the residual blocks in the decoder, concatenated with the up-sampled images from the up-sampling layers. As proposed by the

¹ <https://github.com/dome272/Diffusion-Models-pytorch>

UNet authors [2], these skip connections can be useful in recovering the information lost during downsampling. To adapt the UNet for the noise prediction task, the time embeddings (Transformer sinusoidal position embeddings [3] of the timestep) are inserted into the output of each residual block.

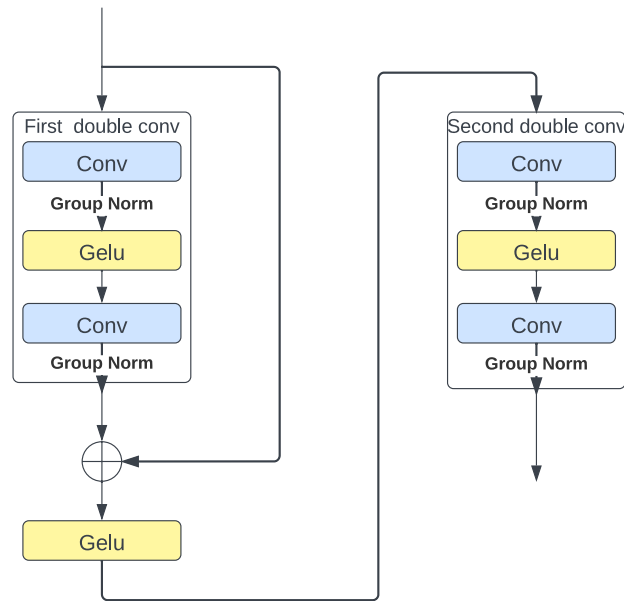


Figure 2: illustration of a typical Res Block in our baseline

This implementation of UNet is mostly the same as the original DDPM paper [1], with a few differences:

- No dropout layers
- No EMA (Exponential Moving Average)
- Down/up-sampling before residual block
- Max pooling for down-sampling instead of convolution or average pooling
- Bilinear upsampling
- Time embedding was added after each residual block, instead of between the two convolution layers
- Attention layers for all resolutions

Most of these differences are simplifications that allow us to highlight the essential parts of UNet and the DDPM model. Compared with the official implementation² which has a complicated structure, this implementation is easier to understand and manipulate. Therefore, we decided to use this one as the baseline, and all our following improvements were built upon this.

² <https://github.com/hojonathanho/diffusion>

Section 2.2: Architectural improvements [Author: Haotian and Kai]

We provide seven revised models for the image generation task: among them, DDPM_AdditionalResLink and DDPM_Deep are inspired by the Resnet [4]; DDPM_Anti_aliasing, DDPM_4DoubleConvBlocks and DDPM_BigGAN are inspired by the architecture improvements in the paper [5]. Apart from that, DDPM_RDN and DDPM_ConvNeXt take inspiration from recently proposed network structures: Residual Dense block and ConvNeXt block, for the computer vision field [6] [7]. All these improvements are built on the UNet backbone of the DDPM baseline.

DDPM_AdditionalResLink [Haotian]: DDPM baseline has one residual block containing 2 double convolution blocks for each resolution, yet only the first double convolution block, where the numbers of the input channels and output channels are the same, has the residual link. It is natural to think of adding a residual link to the second block, as more residual links will at least not harm the model's performance [4]. Inspired by He et al. [4], we find that there is a technique to add a shortcut even if the input and output are of different dimensions. Therefore, we add a residual link also to the second double convolution block.

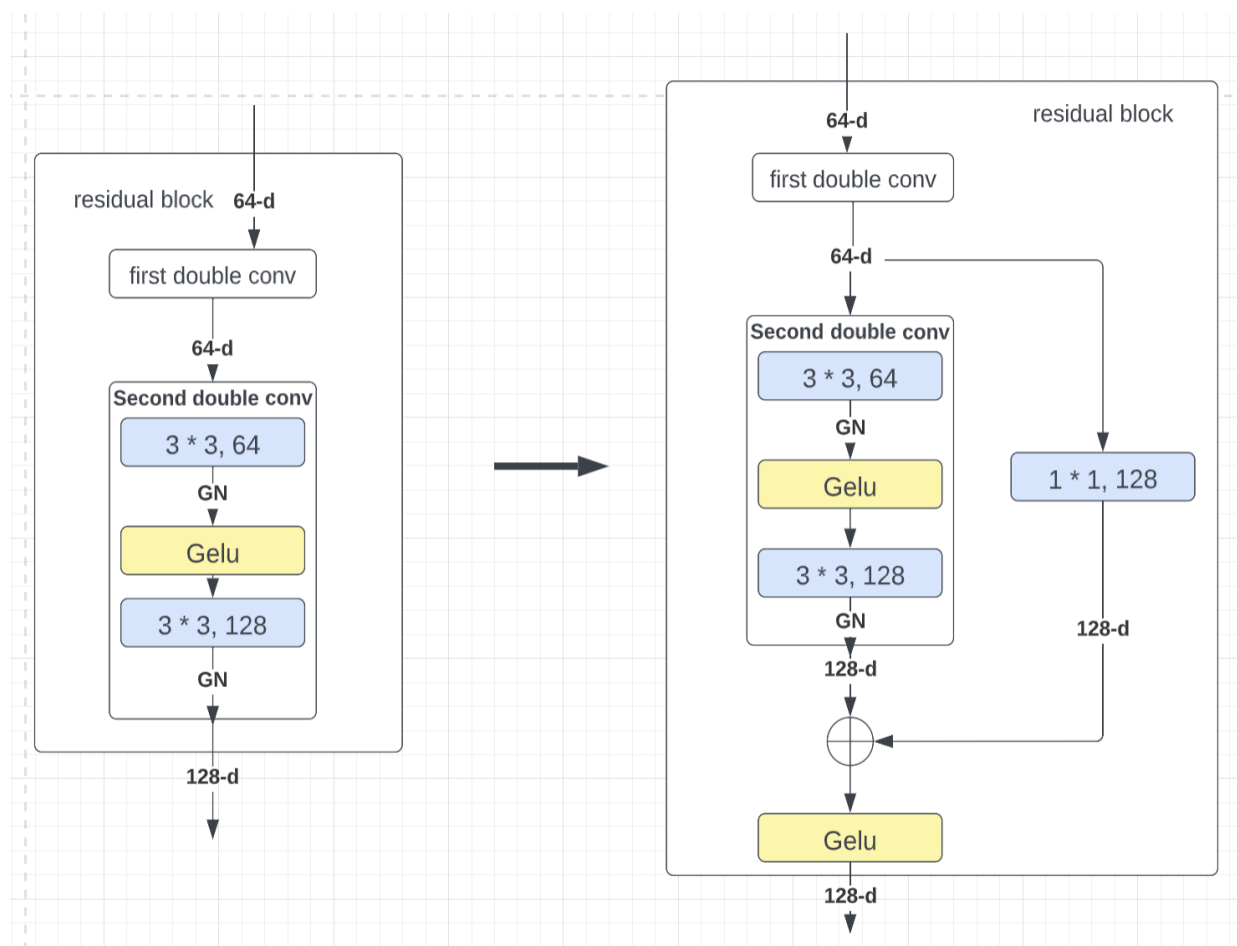


Figure 3: **Left:** original double convolution block. **Right:** double convolution block with an additional residual link.

DDPM_RDN [Haotian]: According to Zhang et al. [7], residual blocks do not fully exploit the information of each convolutional layer within. This issue can be solved by the recently proposed Residual Dense Block (RDB), which consists of both densely connected layers and local residual learning, extracting sufficient local features. In addition, a variant of the RDB: residual-in-residual Dense Block has been applied to a generative model to obtain the state-of-the-art result for the super-resolution imaging [8]. Considering all the above, we replace the residual block with RDB for better performance of the model.

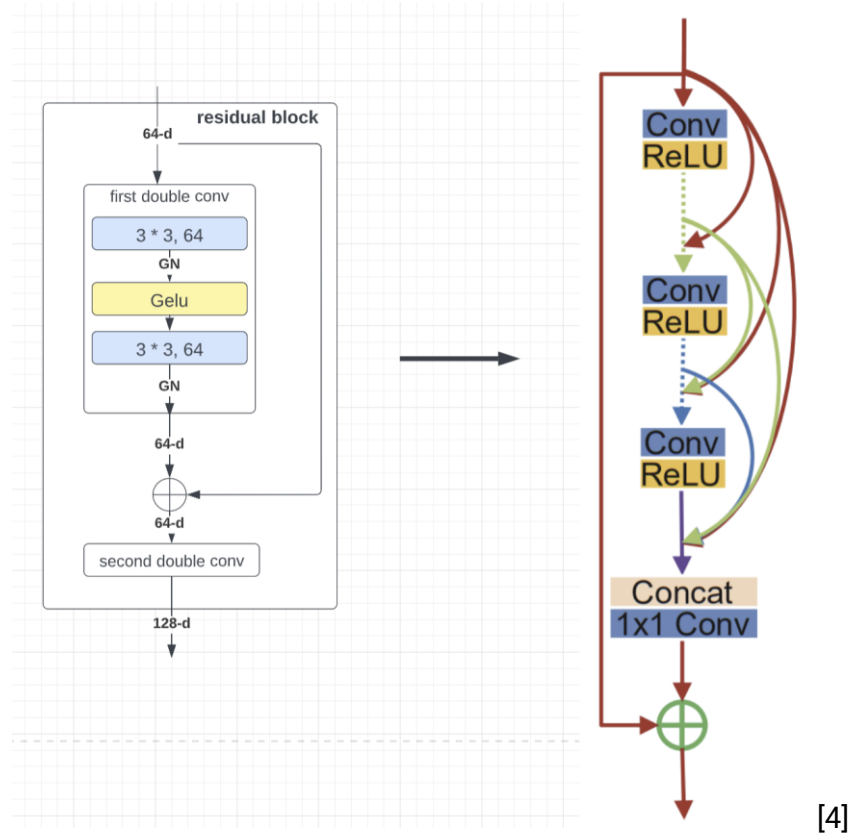


Figure 4: **Left:** residual block. **Right:** residual dense block.

DDPM_Deep [Haotian]: Residual structure can facilitate the training of the deeper neural network by alleviating issues such as vanishing gradient [4]. To fully utilize the benefits of the residual structure of the DDPM, one can make the network deeper. One possible idea is to introduce an additional resolution, and this is proved to be effective based on our empirical study on the DDPM_PureUNet model (def in 2.6), as we find that adding an additional resolution can reduce the FID. Therefore, we deepen the DDPM by introducing one more resolution.

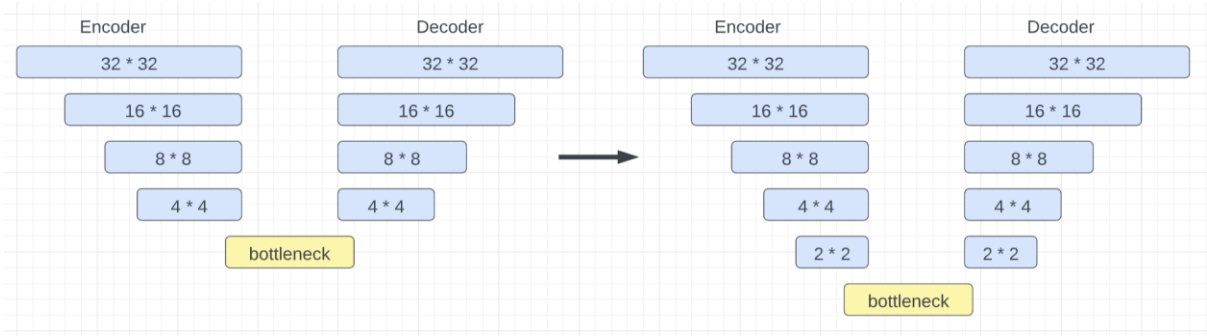


Figure 5: **Left:** 4 resolutions. **Right:** 5 resolutions.

DDPM_ConvNeXt [Haotian]: Built purely with convolutional layers, ConvNeXt is believed to be a perfect substitute for ConvNet, retaining the inductive bias of the convolution [6]. Compared to the vision-transformer-based model, ConvNeXt is simpler and more efficient and beat the former across many computer vision benchmarks [6]. Inspired by the above, we introduce the DDPM_ConvNeXt model, mainly by substituting the residual block into the ConvNext block.

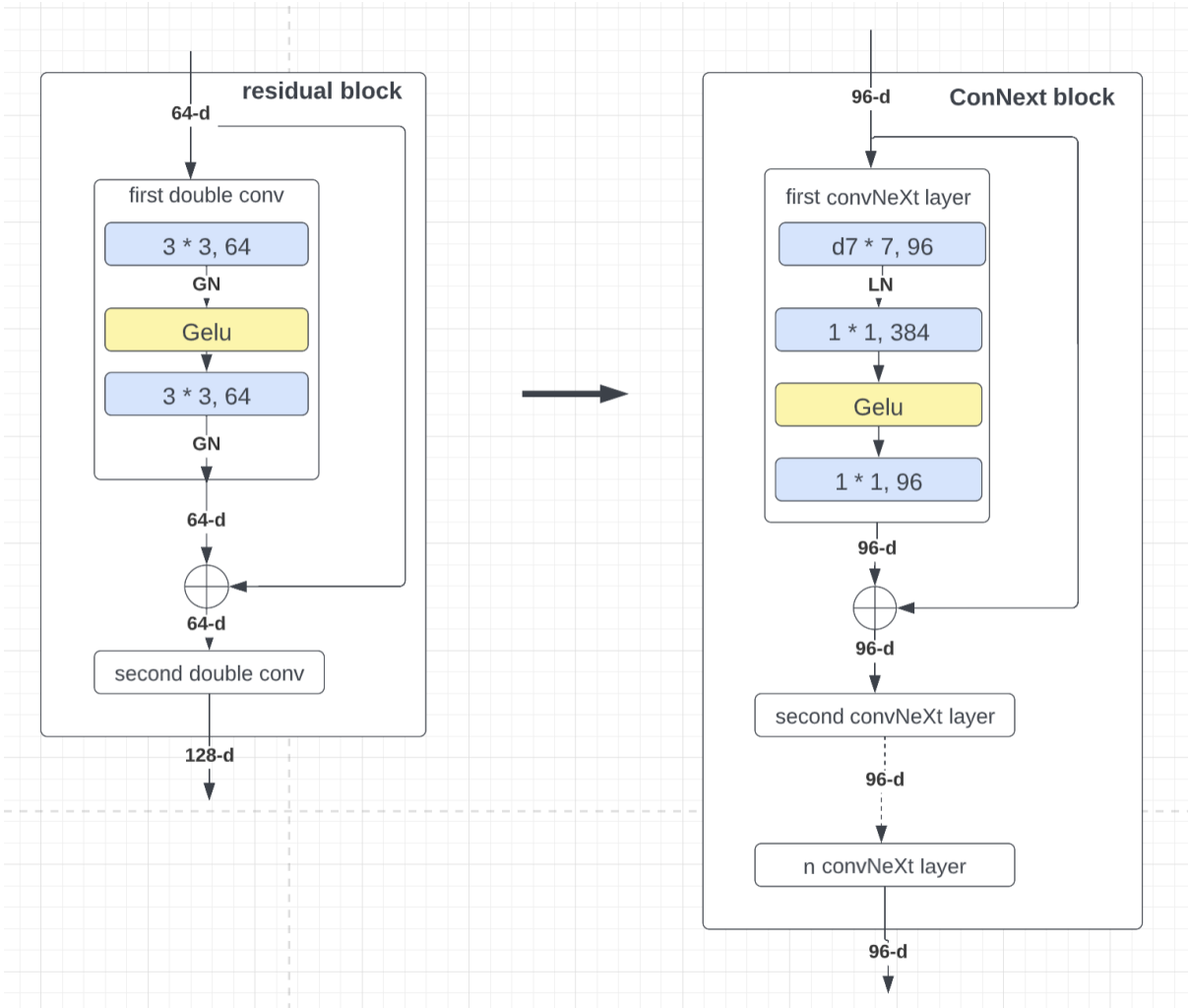


Figure 6: **Left:** residual block. **Right:** ConvNeXt block.

DDPM_Anti_aliasing [Kai]: Song et al. [5] suggest 5 architectural improvements to the U-Net in DDPM. The first one is to add anti-aliasing during up/down sampling by incorporating a low pass filter [9]. By applying a smoothing filter before downsampling, the process becomes less sensitive to small translations of the image input. The authors of [9] empirically show that this shift-invariant property benefits the performance of modern deep neural networks. Since the U-Net also contains a down-sampling and up-sampling process, we decided to also incorporate this as one variant model to be experimented on.

DDPM_4DoubleConvBlocks [Haotian]: Motivated by the [5], the number of the double convolution blocks in each residual block of the model has increased from 2 (number of blocks used in the baseline) to 4 to improve the performance of the model. In addition, the skip connections of each residual link are rescaled, since it has been found effective in several dominant GAN models including StyleGAN [10].

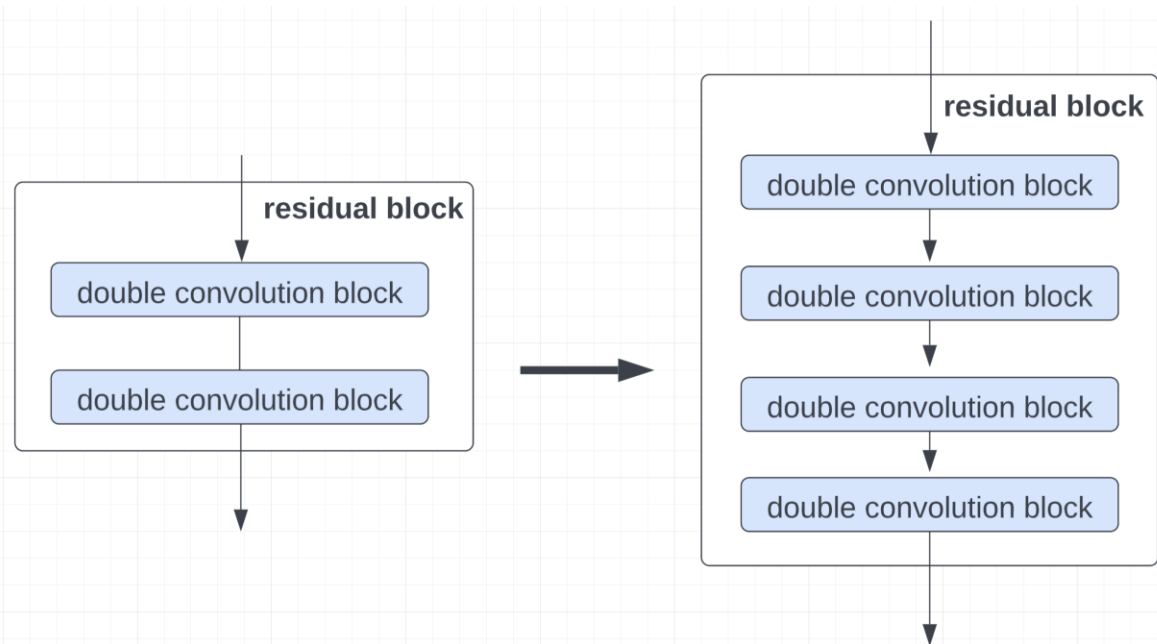


Figure 7: **Left:** 2 double convolution blocks. **Right:** 4 double convolution blocks.

DDPM_BigGAN [Kai]: Another architectural improvement proposed in [5] is the substitution of original residual blocks with residual blocks from BigGAN [11]. These new residual blocks are depicted in Figure 8. Their differences from the residual blocks in the DDPM baseline model are: 1) activation functions are ReLU instead of GELU; 2) average pooling instead of max pooling; 3) different orders or arranging layers; 4) different locations to add residual links.

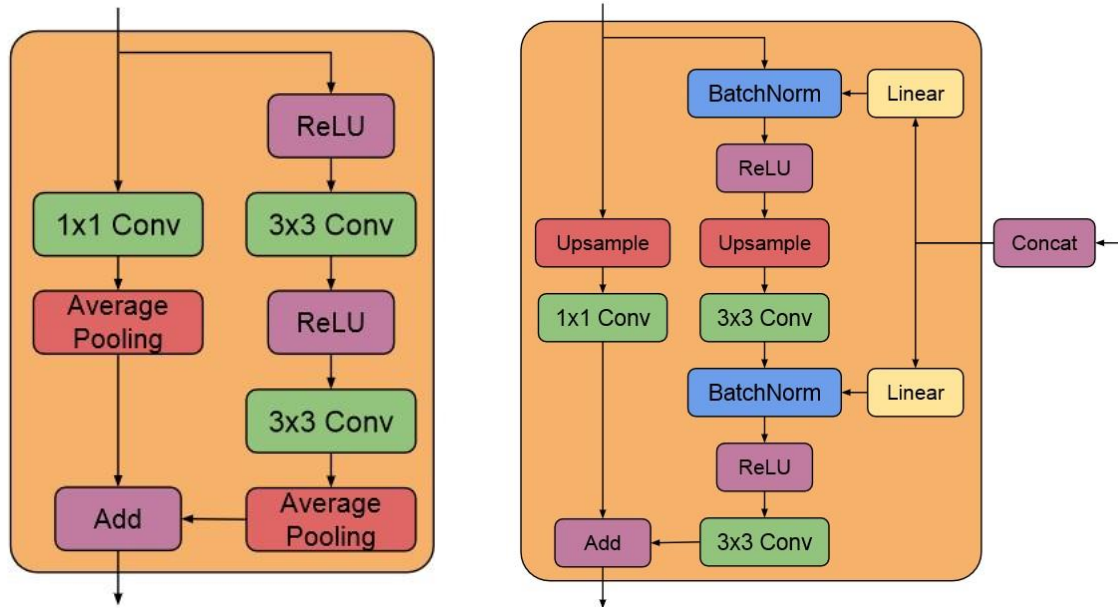


Figure 8: Residual blocks from BigGAN (figures taken from the original paper). **Left:** A residual block in BigGAN’s discriminator. **Right:** A residual block in BigGAN’s generator. Both have been adapted in our own code.

DDPM_Growing [Kai] (not implemented): The fifth improvement involves progressive growing architectures [12]. However, such architectures are used mostly when dealing with high-resolution images [12]. In this project, we are concerned with only the CIFAR-10 dataset containing 32x32 images, so progressive growing is not likely to be useful. Therefore, this improvement is not implemented in this project.

DDPM_RefineNet [Kai] (not implemented): Acknowledgement from Kai: this should have been my part of the job, but I have been ill for a while and cannot finish this in time. I have contacted the lecturer regarding this, and I am hoping the marking will take into account this. RefineNet [13] is used in NCSN [14] (Noise Conditional Score Network), where the same problem as DDPM is defined by a slightly different formulation and solved by score matching. The main difficulty of adapting it for DDPM is that it does not take in time embeddings by default. I was trying to find the appropriate place for inserting the timestep before falling ill but did not succeed.

Section 2.3: Implementation details [Author: Haotian and Kai]

All our improvements are built upon the baseline and retain the UNet skeleton. Time embedding structure is added to the last of each resolution of encoders (down sampling-structure) and decoders (up-sampling structure).

DDPM_AdditionalResLink [Haotian]: For each resolution in the encoder and decoder part of the UNet, we add a residual link for the second double convolution block which does not have one previously due to the input-output dimension mismatch. We use a 1×1 convolution layer to project the dimension of the input channels to the output channels first, then using the result as the skip connection to link with the block.

```
1. if self.in_channels != self.out_channels:
2.     return F.gelu(self.residual_conv(x) + self.double_conv(x))
3. else:
4.     return F.gelu(x + self.double_conv(x))
```

DDPM_RDN [Haotian]: For each resolution in the encoder and decoder part of the UNet, we replace the residual block with one RDN block. The number of convolutional layers in each RDN block follows the setting (8) proposed in the original paper [4] but is rescaled as a half (4) to reduce the computation. The growth rate, or the dimension of the output channels for each convolutional layer in the RDN block is not a constant as used in the original paper; Instead, we adapt the growth rate according to the number of the output channels of the baseline network for each resolution. The network structure of the UNet bottleneck is kept the same. We also naively attempt to replace the UNet backbone of the baseline with the Residual Dense Network for the image generation task, yet its performance is worse. Therefore, we retain the UNet skeleton and make changes for each resolution.

DDPM_Deep [Haotian]: The network is deepened by adding an additional resolution. The altered model has 5 resolutions, ranging from 32×32 to 2×2 . However, the max number of output channels is not increased in consideration of the computation cost. Instead, we change the output dimension from the first resolution from 64 to 32, and all other resolutions follow the original specification.

DDPM_ConvNeXt [Haotian]: For each resolution in the encoder and decoder part of the UNet, we replace the residual block with the ConvNeXt block. The stem cell (a network structure that processes the input image at the beginning) is also changed from a 4×4 , stride 4 convolutional layer to a 1×1 , stride 1 convolutional layer for not shrinking the input size (as the input size is as small as 32) [6]. We follow the tradition of the paper by using (96, 192, 384, 768) as the numbers of the output channels [6]. We keep the network structure of the UNet bottleneck the same. Additionally, we remove the self-attention block (which exists in the DDPM baseline) to follow the “pure convnet” idea [6].

DDPM_Anti_aliasing [Kai]: We followed the implementation in StyleGAN [15], which is to apply a smoothing filter before each down-sampling layer. It should be noted that our baseline model is already using bilinear sampling during upsampling, so there is no need to change there. The smoothing filter we chose is:

$$\begin{bmatrix} 0.0625 & 0.125 & 0.0625 \\ 0.125 & 0.25 & 0.125 \\ 0.0625 & 0.125 & 0.0625 \end{bmatrix}$$

, which is a normalised version of:

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

The code for achieving this can be found below. We first define the filter and save it during the initialisation of each down-sampling block:

```
1. # Precalculated and normalised filter
2. self.filter = torch.Tensor([[0.0625, 0.125, 0.0625],[0.125, 0.25, 0.125],[0.0625,
  0.125, 0.0625]])
3. self.filter = self.filter.repeat(in_channels, in_channels, 1, 1)
4. self.filter = self.filter.to(device)
```

, and then applying the filter in the forward function before sending the output to the down-sampling block, by calling “torch.nn.functional.conv2d”:

```
1. x = F.conv2d(x, self.filter, padding='same')
```

. The “same” padding and the construction of our filter ensure that the dimension of the input to the down-sampling block is not changed.

DDPM_4DoubleConvBlocks [Haotian]: For each resolution in the encoder and decoder part of the UNet, the baseline has 2 double convolution blocks in each residual block. Notice that only the first double convolution block has a residual link as its input and output dimensions are the same. We implement 2 more double convolution blocks before the previous first block and follow the specification of that block, making the newly added blocks have residual links. Moreover, the skip connection of each residual connection is rescaled by $\frac{1}{\sqrt{2}}$.

DDPM_BigGAN [Kai]: The residual blocks from BigGAN (as shown in Figure 8) were designed for GANs and need adaptation to be used for a diffusion model. The one for the discriminator resembles down-sampling since both reduce the size of feature maps, so it was adapted to be the residual blocks for down-sampling. For similar reasons, the residual blocks for the generator were adapted for the up-sampling process. Regarding the adaptation, we removed the class embedding input (as shown in light yellow with the label “Linear” in Figure 8), which is not useful for unconditional diffusion models, and added the time embedding input to each of the residual blocks. This incorporation of time embeddings is done by simply adding it to the output of each residual block, in the same fashion as in the DDPM baseline.

Section 2.4: Data pipelines [Author: Kai]

For building the dataset, the original code base uses a generic dataset class, “torchvision.datasets.ImageFolder”. We changed this to “torchvision.datasets.CIFAR10” which is dedicated to the CIFAR-10 dataset (details in Section 3.1), the one we chose for this project. This class automatically “unpickles” the batched images (in CIFAR-10 images are “pickled” and stored in batches). On top of it, a “DataLoader” was built, which randomly shuffles the data and outputs a batch of 128 images during each training step.

For pre-processing, we applied a random horizontal flip to the images, which is claimed to “improve sample quality slightly” [1]. The images are then converted to “Tensor” and normalized to be in the range [-1, 1], to prepare them as input to the network. Details about how data is consumed during training will be in the next section.

Section 2.5: Training procedures [Author: Kai]

Our models were implemented using PyTorch. Our loss function is a Mean Squared Error (MSE) loss, and our learning rate was set as $2e-4$. These are all consistent with the DDPM paper [1], with the exception that we used AdamW (Adam with decoupled weight decay) [16] optimizer as a better replacement for Adam. With the additional weight decay, AdamW could prevent our model from overfitting with the training data.

The number of training epochs was chosen as 100. We swept from 50 to 300 epochs and found 100 epochs to be a good balance between training time and sample image quality. All these training parameters are fixed for all models to ensure fairness during comparison.

During each epoch, a batch of 128 images is loaded, and a random timestep (1-1000) is sampled for each image. The forward diffusion process (as shown in Equation 1 in Section 1.2) generates the noise and the corresponding noised image for each image at the given timestep. The noised images and the timesteps are then taken by our model being trained as the input, yielding the predicted noise. The difference between the predicted noise and the ground truth noise gives the loss which is then backpropagated to the model weights.

Section 2.6: Testing and validation procedures [Author: Haotian]

The testing metric we used, following the original DDPM paper, is the Fréchet Inception Distance (FID), which can calculate the distance of the distributions between the ground truth and the generated images. To conduct the testing, we sample N (N is determined in the next paragraph) images from the model and store them in a directory, and then download the pre-calculated CIFAR-10 FID statistics and store it in another directory. By convention [1], we are comparing with the training set (containing 50000 images). Finally, we calculate the FID score by using the official implementation ported to PyTorch [17].

According to Heusel et al., the number of sampled images N should be at least 2048 and is desirable if greater than 10000 [18]. The DDPM paper samples 50000 images to get an accurate result. However, sampling images is time-consuming for diffusion models, especially when lacking computational resources. Therefore, we conduct an empirical experiment to find the best N for balancing the accuracy and time cost. We train the DDPM_PureUNet model, which is a variant of the DDPM baseline that has no self-attention block to lower the computation cost. Following the fashion in the training procedure, we train

DDPM_PureUNet for 100 epochs. We sample from 2048 images to 10240 images with a step of 2048 using the model and calculate the FID respectively.

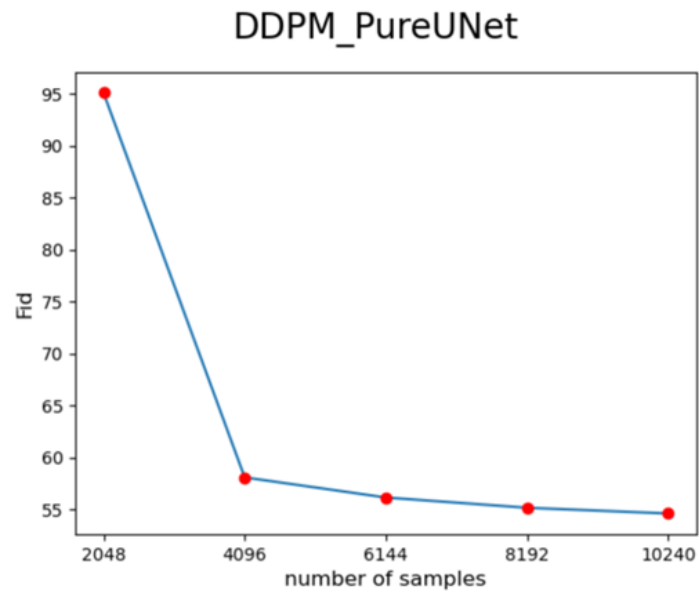


Figure 9: FID against the number of samples

As shown in the above plot, FID drops fast from 2048 to 4096 samples and remains steady afterwards. Therefore, we choose the number of image samples $N = 4096$.

Chapter 3: Experiments and Evaluation

Section 3.1: Datasets [Author: Kai]

Our dataset for training and testing is CIFAR-10 [19], which consists of 32x32 images from 10 classes. It was chosen because its small size and image resolution enable relatively short training time, which allows us to test multiple different architectures within the project timeline. Also, it is one of the benchmark datasets used in the DDPM paper [1].

Although the images come from 10 classes, this project focus on unconditional diffusion models which requires no class information. It should be expected that our model will not generate images all from the same class, but instead each from a random class.

The training set contains 50,000 images, while the test set contains 10,000. However, since we are following the standard practice of comparing the FID with the training set [1], the test set is discarded in this project.

One obvious limitation of this dataset is the low resolution of images. Some possible extensions on this can be the up-sampled version (64x64), or other high-resolution datasets used in the DDPM paper including CelebA-HQ and LSUM. Nonetheless, these would all require more training time which would exceed the time scope of this project.

Section 3.2: Training results [Author: Kai]

Figure 10 illustrates the training loss with respect to the number of training steps for two models, DDPM_baseline and DDPM_BigGAN, respectively. The baseline model is trained for a total of 300 epochs for this experiment only, to study the effect of the number of training epochs on performance. In the following sections, the results for baseline all come from the checkpoint where the model is trained for 100 epochs, if not noted otherwise.

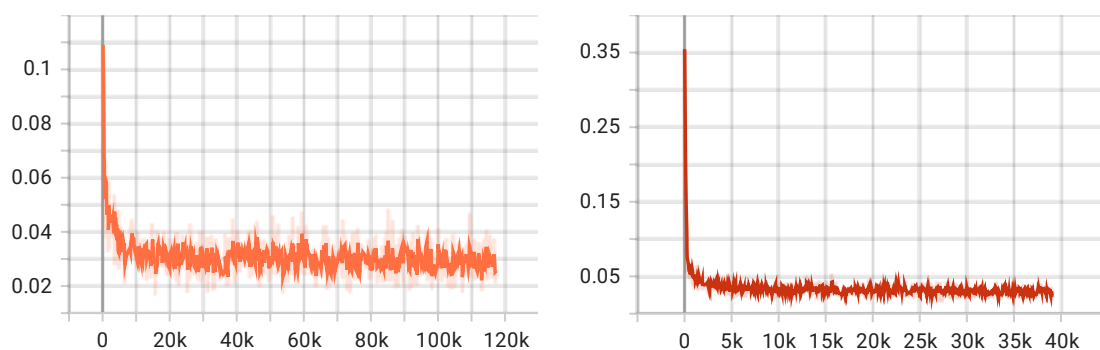


Figure 10: MSE loss for each step during training.

Left: baseline model trained for 300 epochs. **Right:** DDPM_BigGAN trained for 100 epochs.

For both plots, the darker colour denotes a smoothed curve with a factor of 0.6, and the lighter colour denotes the original data. Plots generated by tensorboard.





































































































As can be seen in the figure, the loss drops dramatically in just about 5000 steps (roughly 12 epochs, since each step here, corresponds with one 128-image batch and one epoch consists of 391 such batches). After the drop, the loss stabilizes at around 0.03, with small random

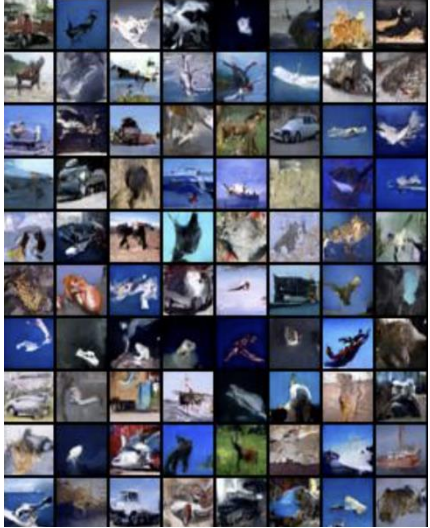


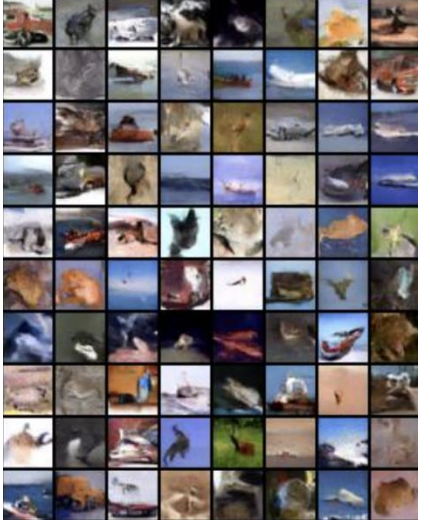


fluctuations. These fluctuations are to be expected since the training loss is calculated based on each step and the image batch in that step influences the actual loss calculation. This pattern exists for both the baseline and DDPM_BigGAN and the other models which are omitted here.

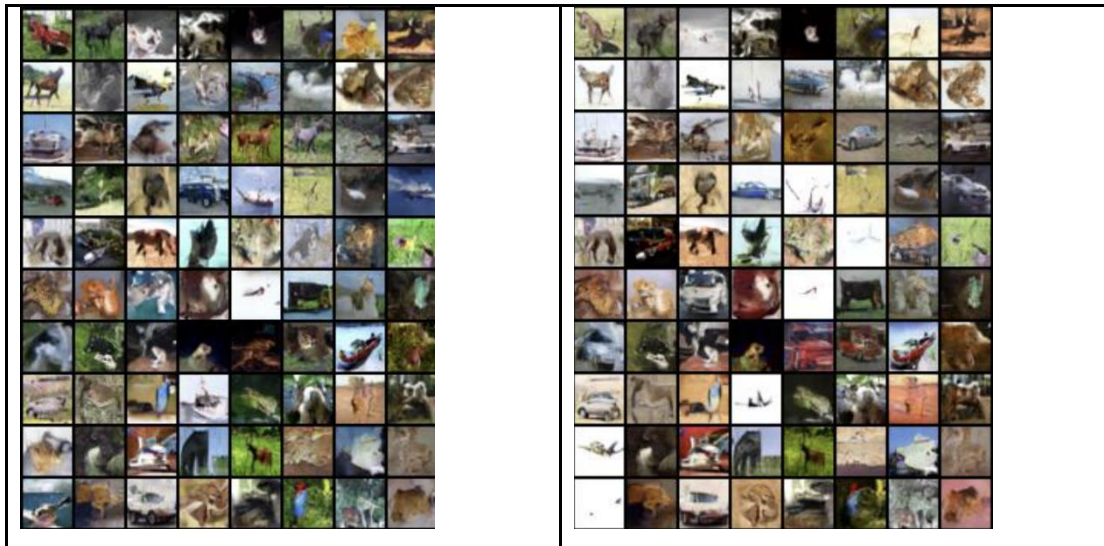
While the loss values suggest that the models have been trained properly, they may not be good indications of the models' performance. All our models have achieved a 0.03 loss after sufficient training. Therefore, our evaluation will mainly focus on the quality of sampled images and the FID scores, as shown in the following two sections.

Section 3.3: Qualitative results [Author: Haotian]

We generate 80 images for each diffusion model that are trained 100 epochs. For ease of comparison, we set the random seed to 0 to ensure the original gaussian noise is the same.

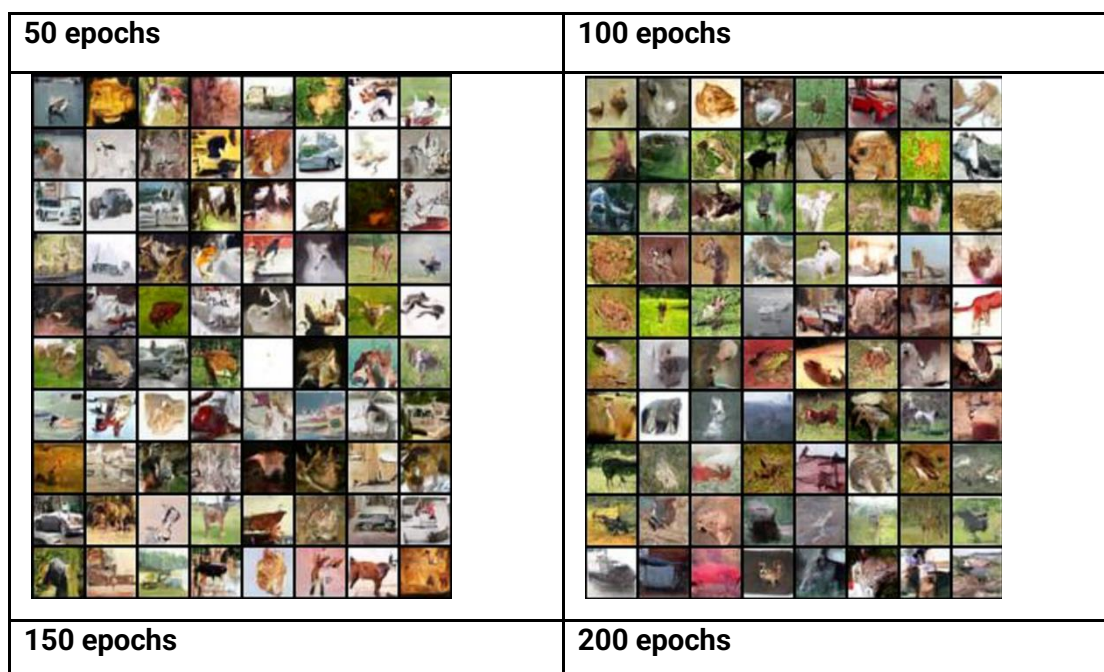
Original CIFAR-10 Dataset										
airplane										
automobile										
bird										
cat										
deer										
dog										
frog										
horse										
ship										
truck										
DDPM_Baseline						DDPM_AdditionalResLink				

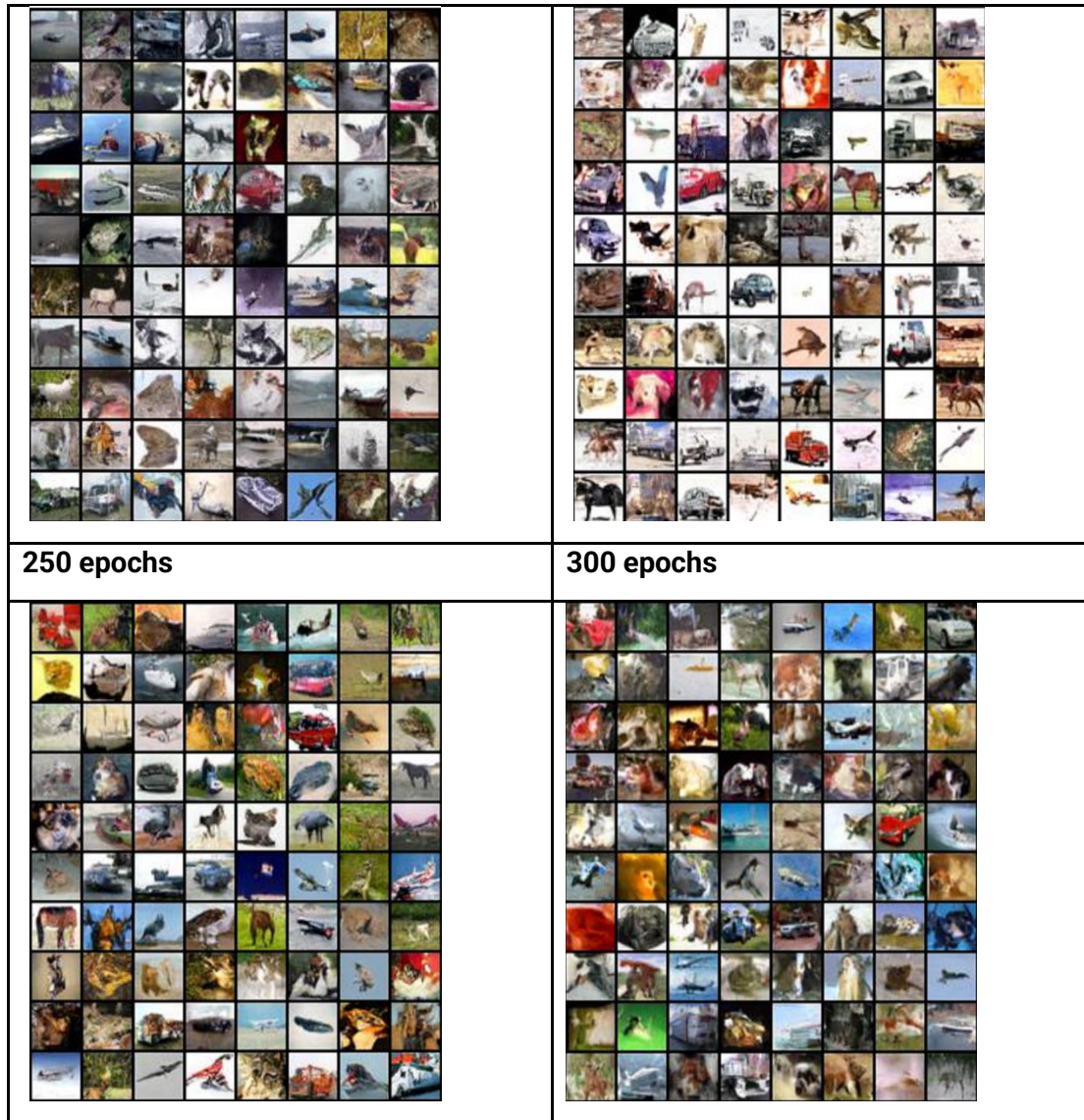
	
DDPM_RDN	DDPM_Deep
	
DDPM_ConvNeXt	DDPM_Anti_aliasing
	
DDPM_4DoubleConvBlocks	DDPM_BigGAN



From the chart above, we can find that the images generated from the diffusion models, though share a resemblance with the real-world images, cannot match the quality of the CIFAR-10 images. We can find that the generated images of different models in the same position are similar to each other as they share the shape or color. One interesting fact is that different models tend to generate different types of images with the same gaussian noise. For example, the top-left image generated by DDPM_AdditionalResLink is like a horse, while DDPM_Deep generates a truck.

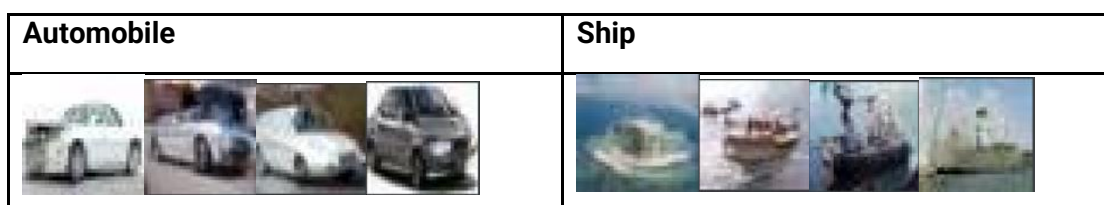
We also explore the behaviors of the models that are the same network trained with a different number of epochs. We choose the DDPM_BigGAN network and train it with 300 epochs, generating a model per 50 epochs. We also ensure the same gaussian noise is used in the sampling process. Below is the result:





We find that the types of images generated from the same gaussian noise vary significantly even if the underlying network is the same. This shows that the diffusion model does not converge and continues to change its parameters during training, which can be illustrated by the oscillation of the MSE loss in Figure 10.

We also notice that the generated images of man-made objects including automobiles and ships are more likely to be high-quality, thus easier to be identified. To prove this, we generated 1000 images from the DDPM_BigGAN network with 300 epochs training first. Then we use a pre-trained densenet121 model (used for CIFAR-10 classification) to classify the generated images [20]. Below are some examples:





We can find that automobile and ship images are more recognizable and can be distinguished from each other, whereas cat and dog images can be hardly identified. This is probably because man-made objects have simple and distinct patterns in the CIFAR-10 dataset, thus facilitating the learning process of models. For example, an object with four wheels (or two wheels on one side, as shown in the generated images) can be distinguished from other objects like ships, airplanes, and trucks (trucks have more wheels), whereas cats, dogs, deer, and horses share the pattern of 4 legs and one head. The diffusion model may merge the features of different animals, thus generating weird images.

Section 3.4: Quantitative results [Author: Haotian]

Models	FID	Num of epochs	Num of samples
DDPM_baseline	75.538	100	4096
DDPM_AdditionalResLink	55.932	100	4096
DDPM_RDN	60.524	100	4096
DDPM_Deep	67.579	100	4096
DDPM_ConvNeXt	69.575	100	4096
DDPM_Anti_aliasing	71.902	100	4096
DDPM_4DoubleConvBlocks	70.211	100	4096
DDPM_BigGAN	53.096	100	4096

Table 1: FID scores for our models

We find that all the variants of the baseline model have better performance compared to it. Among all the models, DDPM_BigGAN obtain the best performance, indicating the effectiveness of the BigGAN block. Intriguingly, DDPM_AdditionalResLink obtains the second-best performance with only a single addition of a residual link. It should be noticed that this residual link for every double convolutional block also exists in the BigGAN residual blocks. Therefore, the residual links might be the most important factor for improving the FID score, possibly due to its functionality of mitigating vanishing/exploding gradients.

We also did an ablation study and found the attention layers (already present in our baseline) to be important for improving results. For example, without the attention layers, the DDPM_BigGAN model can only achieve an FID score of 75.941 (this variant is not shown in the table), whereas its counterpart with attention layers achieves 53.096, the best score among the models.

Section 3.5: Comparison to state-of-the-art [Author: Kai]

While the authors of DDPM [2] achieve an FID of 3.17 on CIFAR-10 which is significantly better than ours, it should be noted that we have made simplifications (as described in Section 2.1) to save training time and ease the comparison between multiple variants built on top of the baseline. In particular, their model was trained for 800k steps (much longer than ours) and

thus should be expected to perform better. In addition, their FID score was calculated based on 50000 sampled images, whereas we only sampled 4096. As Figure 9 in Section 2.6 suggests, FID usually decreases with the number of sampled images, and it is recommended to have the same number of images as in CIFAR-10 (50000 as we are only using the training set) for an accurate FID score. However, we have to reduce the number of sampled images to save sampling time.

Despite the simplifications, the results we got are still useful since we are comparing against our baseline to study the effects of different architectures. The additional residual link and the residual blocks from BigGAN seem to be most beneficial to the performance, with FID scores of 55.9 and 53.1 respectively. The BigGAN residual blocks also implement the residual link for each up/down sampling blocks, which may explain the similar performance of the two models.

Chapter 4: Conclusions and Future Directions

Section 4.1: Conclusions [Author: Kai]

We explored seven different network architectures for replacing the simple U-Net in DDPM and all have been proven to yield better FID scores. The best model is the one that uses the residual blocks from BigGAN which has an FID score of 53.096 (approximately 22 lower than the baseline) and the second best is the one with additional residual links in all double convolution blocks (FID score: 55.932). Their common characteristic, the additional residual links, seems to have led to this significant improvement.

Qualitatively, we also find that while different models have their uniqueness, the images sampled by them (from a fixed noise image) still show similarity to some degree. In addition, they all tend to generate better-quality images for machines or vehicles and perform poorly for animals.

Section 4.2: Discussion of limitations [Author: Kai]

Some of the main limitations of this project are as follows.

- Isolated models: We tested seven different improvements separately but did not try to combine them. A combination of these techniques might lead to a better model since they have all been shown to improve performance.
- Simplified DDPM: As discussed in Section 2.1, our baseline is a simplification of the model in the original DDPM paper [1]. While these simplifications lighten the burden to understand the codebase, it can also be interesting to test the improvements on the original DDPM model.
- Limited training time: Restricted by the project timeline, we have to limit the number of training epochs, to explore more architectures. As the training loss plot in Figure 10 shows, the model may not have converged, and further training may be required.
- Sub-optimal performance: Both the qualitative and quantitative results indicate that our models are not optimal, even with the improvements. The sampled images cannot be convincingly distinguished by a human, and the FID scores are evidently higher than the one (3.17) reported in the DDPM paper.

Section 4.3: Future directions [Author: Kai]

The first extension to this project would be to combine the models to further boost performance. To achieve this, it may be necessary to study the exact effect of each improvement scheme. The insight gained from this can guide the choice of which improvements to incorporate for a given task.

Given sufficient time and computing resources, it would also be useful to experiment on the original DDPM model and train for as long as the original paper says (80k steps). This will improve the overall performance and may give rise to a better comparison between the models. From the analysis of sampled images by sending the same noise to DDPM_BigGAN

trained for different numbers of epochs in Section 3.3, further training will likely influence the models, which may bring potential improvements.

Section 4.4: Contributions of team members

[Kai]: I explored the current baseline code and made a comparison with other DDPM implementations (in other GitHub repositories and from Hugging Face), to choose the baseline to work on. I adapted the baseline code for our task (e.g., to directly use CIFAR-10 in its data pipeline and remove conditional DDPM parts) and implemented two improvements (DDPM_Anti_aliasing and DDPM_BigGAN). I trained and tested the baseline and the two improved models. Due to my illness, I was not able to finish DDPM_RefineNet.

[Haotian]: I designed and implemented the experiment and evaluation pipeline, studying and regulating the related parameters including the testing metric and the number of sampled images. I designed and implemented 5 networks, namely DDPM_AdditionalResLink, DDPM_RDN, DDPM_Deep, DDPM_ConvNeXt and DDPM_4DoubleConvBlocks, and trained and tested them. I conducted qualitative and quantitative evaluations of the experiment results of all the proposed models.

[Haolong]: I did the literature review about DDPM baseline and relevant models derived from DDPM. I helped with the implementation and testing of new architectures and models (DDPM_AdditionalResLink, DDPM_RDN, DDPM_Deep, DDPM_ConvNeXt and DDPM_4DoubleConvBlocks) on Colab. I worked as a project coordinator. I organised the content and format of the report and presentation slides.

References:

- [1] J. Ho, A. Jain, and P. Abbeel, 'Denoising Diffusion Probabilistic Models', *CoRR*, vol. abs/2006.11239, 2020, [Online]. Available: <https://arxiv.org/abs/2006.11239>
- [2] O. Ronneberger, P. Fischer, and T. Brox, 'U-Net: Convolutional Networks for Biomedical Image Segmentation', *CoRR*, vol. abs/1505.04597, 2015, [Online]. Available: <http://arxiv.org/abs/1505.04597>
- [3] A. Vaswani et al., 'Attention Is All You Need', *CoRR*, vol. abs/1706.03762, 2017, [Online]. Available: <http://arxiv.org/abs/1706.03762>
- [4] K. He, X. Zhang, S. Ren, and J. Sun, 'Deep residual learning for image recognition', in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
- [5] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole, 'Score-based generative modeling through stochastic differential equations', *arXiv preprint arXiv:2011.13456*, 2020.
- [6] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, 'A convnet for the 2020s', in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 11976–11986.
- [7] Y. Zhang, Y. Tian, Y. Kong, B. Zhong, and Y. Fu, 'Residual dense network for image super-resolution', in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2472–2481.
- [8] X. Wang et al., 'Esrgan: Enhanced super-resolution generative adversarial networks', in *Proceedings of the European conference on computer vision (ECCV) workshops*, 2018, pp. 0–0.
- [9] R. Zhang, 'Making Convolutional Networks Shift-Invariant Again'. *arXiv*, Jun. 08, 2019. doi: 10.48550/arXiv.1904.11486.
- [10] R. Abdal, Y. Qin, and P. Wonka, 'Image2stylegan: How to embed images into the stylegan latent space?', in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 4432–4441.
- [11] A. Brock, J. Donahue, and K. Simonyan, 'Large Scale GAN Training for High Fidelity Natural Image Synthesis'. *arXiv*, Feb. 25, 2019. doi: 10.48550/arXiv.1809.11096.
- [12] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila, 'Analyzing and Improving the Image Quality of StyleGAN', *CoRR*, vol. abs/1912.04958, 2019, [Online]. Available: <http://arxiv.org/abs/1912.04958>
- [13] G. Lin, A. Milan, C. Shen, and I. D. Reid, 'RefineNet: Multi-Path Refinement Networks for High-Resolution Semantic Segmentation', *CoRR*, vol. abs/1611.06612, 2016, [Online]. Available: <http://arxiv.org/abs/1611.06612>
- [14] Y. Song and S. Ermon, 'Generative Modeling by Estimating Gradients of the Data Distribution', *CoRR*, vol. abs/1907.05600, 2019, [Online]. Available: <http://arxiv.org/abs/1907.05600>
- [15] T. Karras, S. Laine, and T. Aila, 'A Style-Based Generator Architecture for Generative Adversarial Networks'. *arXiv*, Mar. 29, 2019. doi: 10.48550/arXiv.1812.04948.
- [16] I. Loshchilov and F. Hutter, 'Decoupled Weight Decay Regularization', *CoRR*, vol. abs/1711.05101, 2017, [Online]. Available: <http://arxiv.org/abs/1711.05101>
- [17] M. Seitzer, 'pytorch-fid: FID Score for PyTorch'. Aug. 2020. [Online]. Available: <https://github.com/mseitzer/pytorch-fid>

- [18] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, 'Gans trained by a two time-scale update rule converge to a local nash equilibrium', *Advances in neural information processing systems*, vol. 30, 2017.
- [19] A. Krizhevsky, G. Hinton, and others, 'Learning multiple layers of features from tiny images', 2009.
- [20] H. Phan, 'huyvnphan/PyTorch_CIFAR10'. Zenodo, Jan. 2021. doi: 10.5281/zenodo.4431043.