Image Processing Final Project

Team ID : 1

Team member : 110550090 王昱力 110550087 俞柏帆 110550067 簡秉霖

1 Introduction

In this project, we aim to achieve super resolution, enlarging from image sizes from 64×64 to 256×256 pixels. We combine two methods, HAT and ControlNet, where HAT is a state-of-the-art Transformer-based method, and ControlNet is a Diffusion-based method. By integrating these two methods, we generate images that are both highly consistent and smooth.

2 Method

2.1 HAT: Activating More Pixels in Image Super-Resolution Transformer [1] [2]

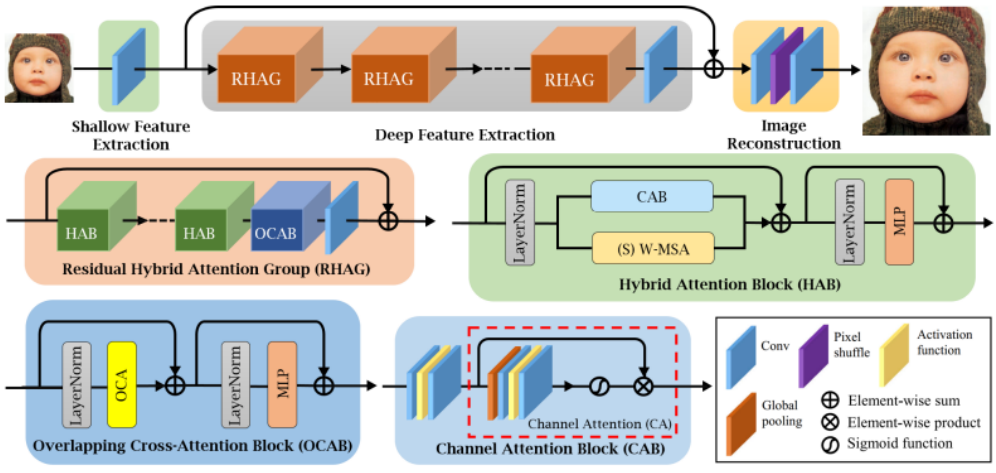


Figure 1: The pipeline begins with shallow feature extraction, followed by deep feature extraction through Residual Hybrid Attention Groups (RHAGs) containing Hybrid Attention Blocks (HAB) and Overlapping Cross-Attention Blocks (OCAB). Finally, deep and shallow features are combined and upsampled to reconstruct the high-resolution image.

2.2 ControlNet: Adding Conditional Control to Text-to-Image Diffusion Models [3]

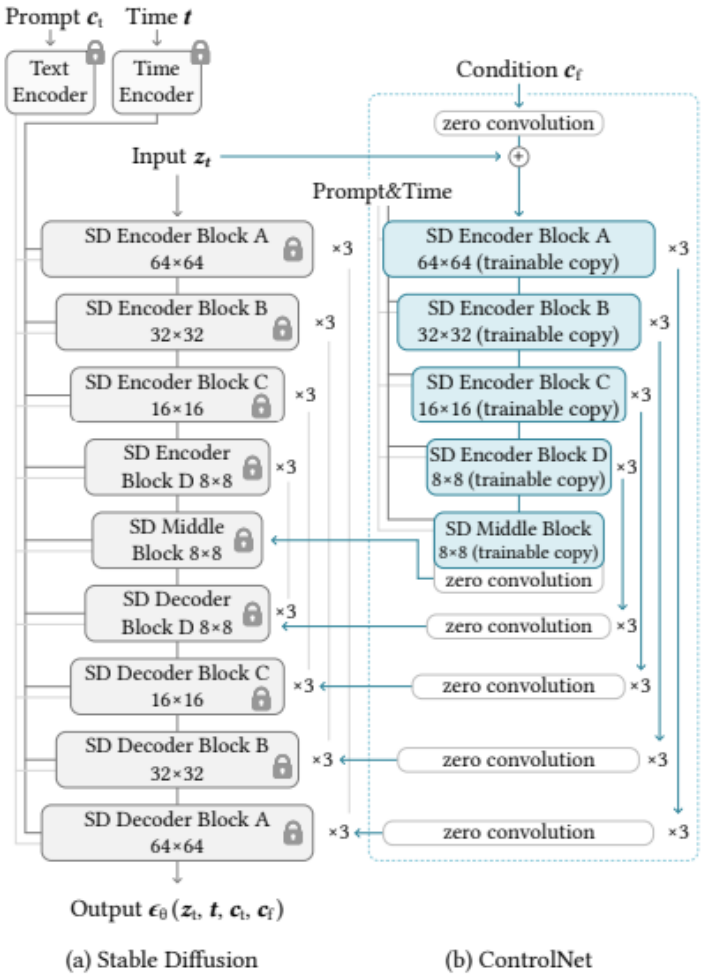


Figure 2: The U-net architecture of Stable Diffusion integrates with a ControlNet on the encoder blocks and middle block. Locked, gray blocks represent the Stable Diffusion structure, while trainable blue blocks and white zero convolution layers build the ControlNet. ControlNet uses an additional input image as a condition, with various types such as canny edge, user sketching, human pose, and depth. This provides greater control over image generation, simplifying the creation of specific images without extensive experimentation. Different conditions influence the final output uniquely, allowing for precise and diverse image modifications.

3 Our Method

As shown in Fig. 3, our method starts with inputting the original image into a transformer-based model, HAT. We discover that passing the images through the HAT model twice reduces background noise more effectively. After passing the images through the model twice, we resize the image back to 512x512. Then, we apply ControlNet diffusion model, which requires additional input as condition. We use edge map as condition, as it can directly obtained from OpenCV function. By

setting a threshold, the edges is detected by comparing the gradient to this threshold. Finally, we downscale the image to 256x256 for the final result.

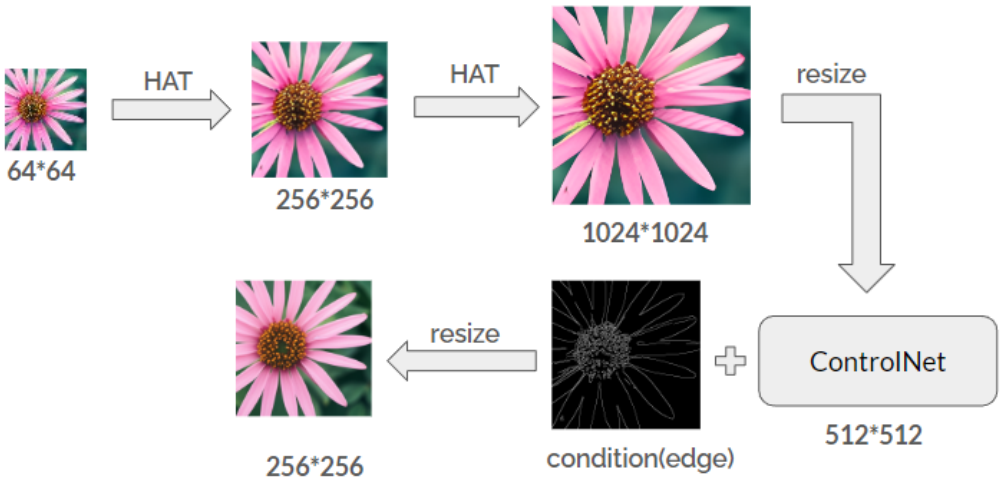
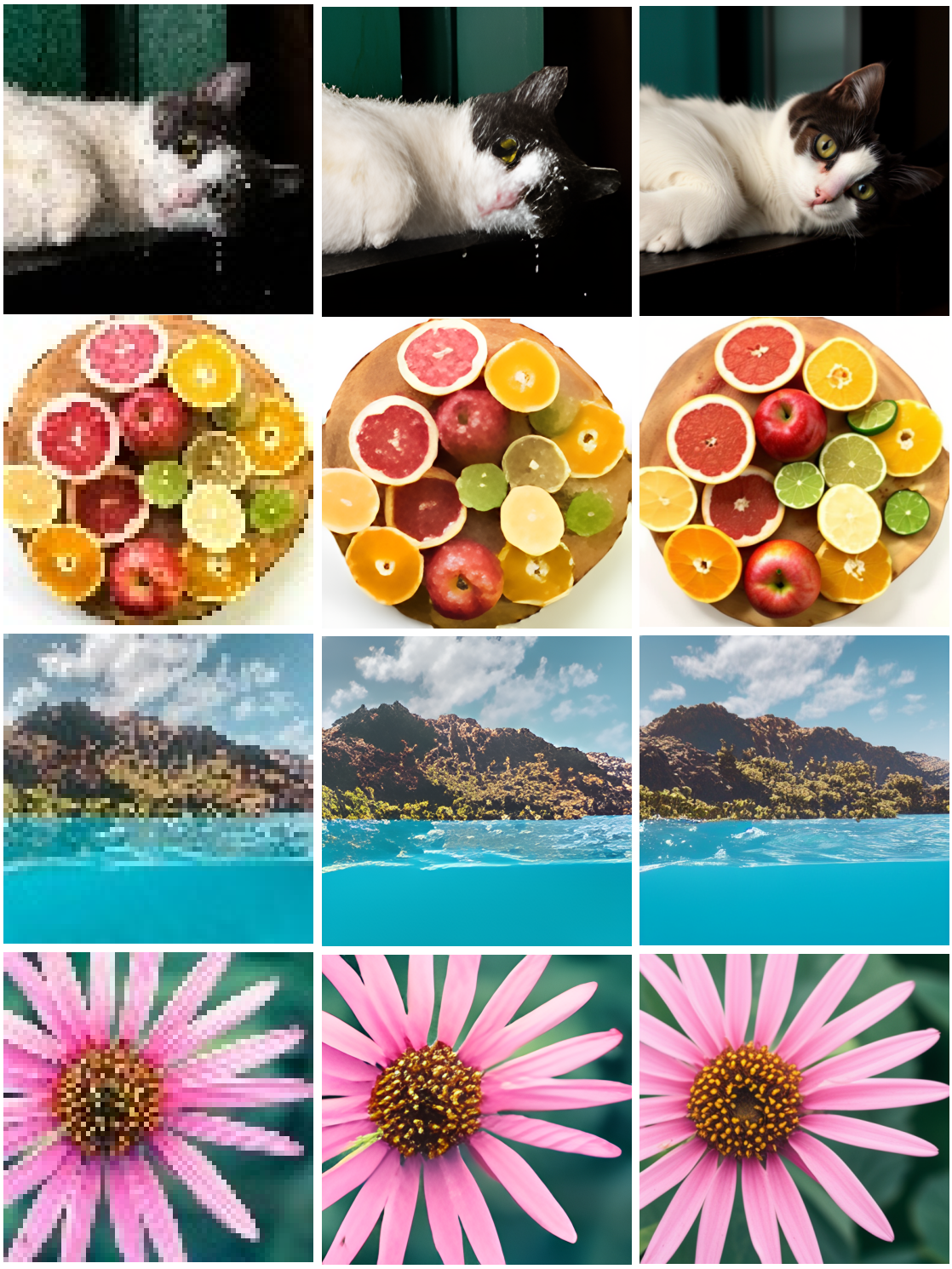


Figure 3: The pipeline of our method.

4 Innovation

Initially, we directly apply HAT to increase the resolution of the images, leveraging the ability of the Transformer to generate realistic and consistent images. However, the results of HAT still exhibit some artifacts and distortions. To address this issue, we use ControlNet as a post-processing step. Specifically, we use Canny-edge as the condition due to its simplicity and superior controllability compared to other common conditioning types such as Depth Map, Normal Map, and Segmentation Map. The Canny-edge condition method allows us to control the number of detected edges and therefore influence the level of detail in the final images by simply adjusting a threshold, while other condition types either need to utilize another deep learning model or provide less flexibility.



Original Image HAT-HAT HAT-HAT+ControlNet

Figure 4: The first column contains the original images, the second column shows the images after the HAT-HAT process, and the third column displays the images after applying both the HAT-HAT process and ControlNet.

5 Result

As shown in Fig. 4, Compared to the first column, the second column does not exhibit a mosaic effect at the same size. It adds many details without altering the original content of the images, resulting in a significant increase in resolution.However, some details in the second column appear unrealistic or slightly blurry. These issues are improved when using ControlNet. The cat's fur looks more realistic and smoother compared to using only HAT-HAT, the fruit appears more vibrant and its texture more closely resembles what is seen in everyday life, and the overall details of the mountain are enhanced. However, since the mountain is inherently more complex, the differences are more noticeable when the image is viewed at a larger size.

References

[1] X. Chen, X. Wang, J. Zhou, Y. Qiao, and C. Dong, "Activating more pixels in image super-resolution transformer," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2023, pp. 22 367–22 377.

[2] X. Chen, X. Wang, W. Zhang, *et al.*, "Hat: Hybrid attention transformer for image restoration," *arXiv preprint arXiv:2309.05239*, 2023.

[3] L. Zhang, A. Rao, and M. Agrawala, "Adding conditional control to text-to-image diffusion models," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2023.

Appendix A. ControlNet code

As shown below, our method shows promising result with concise code.

```
from diffusers import StableDiffusionControlNetImg2ImgPipeline,
   ControlNetModel, UniPCMultistepScheduler
import torch
import numpy as np
import cv2
from PIL import Image
import warnings
warnings.filterwarnings("ignore")

def lcg(seed, a=1664525, c=1013904223, m=2**32):
    """Linear Congruential Generator (LCG) to generate pseudo-
       random numbers."""
    while True:
        seed = (a * seed + c) % m
        yield seed

# generate condition canny edge map(you can adjust the
   low_threshold/high_threshold)
# to generate different number of edges
original_image = Image.open('wood_house.png').resize((512, 512))
image = np.array(original_image)

low_threshold = 75
high_threshold = 150

image = cv2.Canny(image, low_threshold, high_threshold)
image = image[:, :, None]
image = np.concatenate([image, image, image], axis=2)
canny_image = Image.fromarray(image)
canny_image.save('edges.png')

#load image and condition
canny_image = Image.open('edges.png')
image = Image.open('hat_real/15.png').resize((512, 512))

#load stable diffusion and controlnet
ckpt = "emilianJR/epiCRealism"
controlnet = ControlNetModel.from_pretrained("lllyasviel/sd-
   controlnet-canny", torch_dtype=torch.float16, use_safetensors
```

```
    =True)
pipe = StableDiffusionControlNetImg2ImgPipeline.from_pretrained(
     ckpt, controlnet=controlnet, torch_dtype=torch.float16,
        use_safetensors=True
)
pipe.scheduler = UniPCMultistepScheduler.from_config(pipe.
    scheduler.config)
pipe.enable_model_cpu_offload()

#add text prompt
quality = ", highest quality settings, sharp image, realistic,
    detailed"
prompt = "A photo of a beautiful coastal landscape with clear
    sky, mountain covered with trees" + quality
negative_prompt = "blurry, distortion, anime, cartoon, Bad
    quality, Jpeg artifacts, Signature, Username, Watermark"

#set your seed to reproduce the result
seed = 12345
random_generator = lcg(seed)
random_number = next(random_generator)
print(random_number)
generator = torch.manual_seed(random_number)

#generate images and save
output = pipe(
        prompt=prompt, image=image, control_image=canny_image,
            strength=0.2, negative_prompt=negative_prompt,
            generator=generator
    ).images[0]
output.save(f'{random_number}.png')
```