

A THE DETAILED STRUCTURE OF THE UNI-ADAPTER

The dimensions of each output layer from the image extractor and sketch extractor are the same. The input image for the image extractor is $3 \times 512 \times 512$, while for sketch extractor, it is $1 \times 512 \times 512$.

Table 1: Details of image extractor and sketch extractor.

Layer Name	Layer Output Size	Layer	
		Image Extractor	Sketch Extractor
PixelUnshuffle	$(b, c \times 64, 64, 64)$	downscale=8	
Conv in	$(b, 320, 64, 64)$	$\{3 \times 3, 192, 320\}$	$\{3 \times 3, 64, 320\}$
Block 1	$(b, 320, 64, 64)$	$\{Conv, ResBlock * 2, 320, 640\}$	
Block 2	$(b, 640, 32, 32)$	$\{Conv, ResBlock * 2, 640, 1280\}$	
Block 3	$(b, 1280, 16, 16)$	$\{Conv, ResBlock * 2, 1280, 1280\}$	
Block 4	$(b, 1280, 8, 8)$	$\{Conv, ResBlock * 2, 1280, 1280\}$	

The features from every block are extracted for texture and spatial information caption.

B THE PSEUDO CODE FOR TRAINING PROCESS

This section illustrates the pseudo-code for the training process, the image extractor is trained to extract the texture and spatial information while the Dual-LoRA module learned and saved real style and illustrative style separately during each training step.

Algorithm 1 Pseudo PyTorch code for training process

```

1  #load dataset and init model
2  dataloader = create_dataloader()
3  model = create_model()
4  model.init_layer(real_lora, illu_lora)
5
6  #set learnable parameters
7  parameters = model.image_extractor.parameters(), real_lora.parameters(), real_lora.parameters()
8  optimizer = Adam(parameters)
9
10 #training process
11 for step, batch in dataloader:
12
13     # train real lora module or illu lora module
14     lora_style = "real" if (step + 1) % 2 == 0 else "illu"
15     x_0 = batch['real_image'] if lora_style == "illu" else batch['illu_image']
16
17     # insert specific lora module
18     model.unet.set_lora_layer(lora_style)
19     image_feature = model.image_extractor(batch['condition'])
20     sketch_feature = model.sketch_extractor(batch['condition'])
21
22     # forward diffusion
23     t = randint(0, 1000, size=[batch_size])
24     x_t = forward(x_0, noise, t)
25
26     # diffusion and compute loss
27     model_pred = model.unet(x_t, t, c)
28     loss = F.mse_loss(model_pred, noise)
29     loss.backward()
30     optimizer.step()
31     optimizer.zero_grad()
32

```