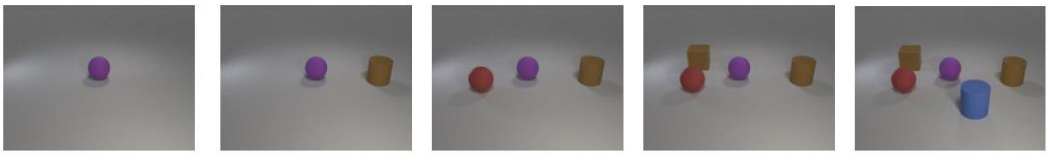
DL_LAB6_Report _ Let's Play DDPM

學號：312554004　姓名：林垣志

(1) Introduction:

In this lab, I will implement a conditional DDPM and generate synthetic images based on multi-labels conditions by given iclevr datasets. In detail, I need to choose a DDPM setting, design our noise schedule and UNet architecture, even a prediction type.

The number of training data is 18009, and two testing data which have same amount is 32 and give an object json file which has 24 classes (object type), the figures are shown below.



(2) Implementation details:

(a) Describe how you implement your model, including your choice of DDPM, UNet architectures, noise schedule, and loss functions.

The UNet model is using UNet2DModel which is an unconditional architecture, but I gave a extra input channels to accept multi-labels conditions. In this model, I set the in_channel as 27, because the input's channel is 3 and the label has 24 classes, so I will concat the image and label as a new_input (class labels as an additional argument), then feed this to the UNet. The down/up sampling block are setting the self-attention, and the layer_per_blocks is set to 2. Noise schedule is using DDPMSchedular given the parameter (num_train_timesteps = 1000), it's means that will reduction noise 1000 times in sampling loop. Last, I using MSELoss as my loss criterion.

```python
class ClassConditionedUnet(nn.Module):
    def __init__(self, args):
        super(ClassConditionedUnet, self).__init__()
        self.args = args
        self.batch_size      = args.batch_size
        self.mse_criterion   = nn.MSELoss()
        self.noise_scheduler = DDPMScheduler(num_train_timesteps=1000, beta_schedule='squaredcos_cap_v2')

        # Self.model is an unconditional UNet with extra input channels to accept the conditioning information (t
        self.model = UNet2DModel(
            sample_size=self.batch_size,     # the target image resolution
            in_channels=27,                  # Additional input channels for class cond.
            out_channels=3,                  # the number of output channels
            layers_per_block=2,              # how many ResNet layers to use per UNet block
            block_out_channels=(64, 128, 128),
            down_block_types=(
                "DownBlock2D",        # a regular ResNet downsampling block
                "AttnDownBlock2D",    # a ResNet downsampling block with spatial self-attention
                "AttnDownBlock2D",
            ),
            up_block_types=(
                "AttnUpBlock2D",
                "AttnUpBlock2D",      # a ResNet upsampling block with spatial self-attention
                "UpBlock2D",          # a regular ResNet upsampling block
              ),
        )
        self.optim           = optim.AdamW(self.parameters(), lr=self.args.lr)

    # Our forward method now takes the class labels as an additional argument
    def forward(self, x, t, class_labels):
        # Shape of x: ch ==> 3 , class_labels.shape ==> [128,24]
        bs, ch, w, h = x.shape

        class_labels = class_labels.view(bs, class_labels.shape[1], 1, 1).expand(bs, class_labels.shape[1], w, h)

        # Net input is now x and class cond concatenated together along dimension 1
        net_input = torch.cat((x, class_labels), 1) # # (bs, 27, 64, 64)

        # Feed this to the unet alongside the timestep and return the prediction
        return self.model(net_input, t).sample
```

The hyper parameters I use are shown below.

Batch_size	Learning rate	Epochs	Optimizer	Loss function
64	0.001	70	AdamW	MSE

(3) Results and discussion:

(a) Show your accuracy screenshot based on testind data.

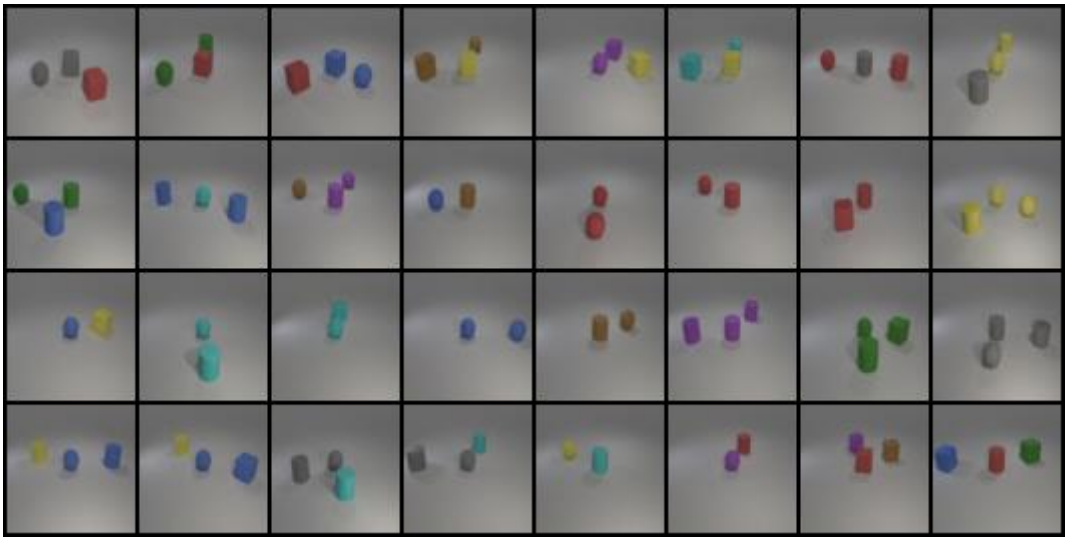
The two testing best accuracy are shown below.

test	0.7222
new_test	0.72619

(b) Show synthetic image grids and a progressive generation image.

- test.json image grids

- new_test.json image grids



(c) Discuss the results of different models architectures or methods.

Due to limited time for conducting further experiments, I utilized the same architecture for the model and experimented with and without the incorporation of conditions. Without adding conditions, in essence, giving it Gaussian noise alone is sufficient to generate images. The advantage is that the input is not constrained in any way, but the downside is that the generated results are uncontrolled due to the lack of supervision. However, in terms of experimental results, it becomes evident that the performance is notably poor. On the other hand, when conditions are introduced, it becomes possible to control the generated style, achieving image restoration capabilities similar to GANs, resulting in a significant improvement in model performance. The predicting type just choice predicting the noisy sample.