Deep Learning Lab7 (Let's Play DDPM)

311553060

周睦鈞

---

1. Introduction

這次的 Lab 要實作一個 conditional 的 DDPM , 透過指定顏色和形狀 的 One hot label 當做 Condition 最後生成我們要的結果圖

類似 Lab5 使用的 cVAE ，只是這次的生成模型改成使用 Diffusion model

2. Implement detail

2-1. Describe how you implement your model, including your choice of DDPM, UNet architectures, noise schedule, and loss functions

我所使用的 DDPM 是屬於 Pixel domain 的，UNet 架構則是使用 huggingface 所提供的 UNet2DModel ，我基於他們提供的 Model 做了一些修改加上了 condition 的資訊 (e.g. `self.model.class_embedding` 預設是 None , 這邊我將它改成 `nn.Linear` 的形式來對 one_hot vector 做 embedding)

UNet model architecture

```
class ConditionedUNet(nn.Module):
    def __init__(self, args, num_class = 24, embed_size = 512):
        super().__init__()

        self.model = UNet2DModel(
            sample_size = 64,
            in_channels =  3,
            out_channels = 3,
            layers_per_block = 2,
            class_embed_type = None,
            block_out_channels = (128, 256, 256, 512, 512, 1024),
            down_block_types=(
                "DownBlock2D",        # a regular ResNet downsampling block
                "DownBlock2D",
                "DownBlock2D",
                "DownBlock2D",
                "AttnDownBlock2D",    # a ResNet downsampling block with spatial self-attention
                "AttnDownBlock2D",
            ),
```

```
            up_block_types=(
                "AttnUpBlock2D",
                "AttnUpBlock2D",      # a ResNet upsampling block with spatial self-attention
                "UpBlock2D",
                "UpBlock2D",          # a regular ResNet upsampling block
                "UpBlock2D",
                "UpBlock2D",
            ),
        )

        self.model.class_embedding = nn.Linear(num_class, embed_size)

    def forward(self, x, t, y):
        return self.model(x, t, class_labels = y).sample
```

Loss funtion

```
# Loss functions
criterion = nn.MSELoss()
optimizer = optim.Adam(net.parameters(), lr = args.lr)
```

Noise

使用 Normal Distribution 來當作我的 noise

```
noise = torch.randn_like(img)
```

Dataloader

將助教提供的 json 檔透過 getData function 把對應的 Label data 轉成 one_hot vector

並且透過 Dataloader 在 training 的時候輸出 image 和 label

在 testing 的時候只輸出 label

```
class iclevrDataset(Dataset):
    def __init__(self, mode, root='iclevr', data_root = 'data', test_file='test.json'):
        self.filenames = None
        self.labels = None
        self.mode = mode
        self.root = root
        self.filenames, self.labels = getData(mode, data_root, test_file=test_file)
        self.transform = data_transform(mode)

    def __len__(self):
        return len(self.labels)

    def __getitem__(self, idx):
        if self.mode == 'train':
            image = Image.open(os.path.join(self.root, self.filenames[idx])).convert('RGB')
            image = self.transform(image)
            label = torch.Tensor(self.labels[idx])
            return image, label
        else:
            label = torch.Tensor(self.labels[idx])
            return label
```

```
def getData(mode, root, test_file='test.json'):
    label_map = json.load(open(os.path.join(root, 'objects.json')))
    filenames = None
    if mode == 'train':
        data_json = json.load(open(os.path.join(root, 'train.json')))
        filenames = list(data_json.keys())
        labels_list = list(data_json.values())
        one_hot_vector_list = []
        for i in range(len(labels_list)):
            one_hot_vector = np.zeros(24).astype(int)
            for j in labels_list[i]:
                one_hot_vector[label_map[j]] = 1
            one_hot_vector_list.append(one_hot_vector)
        labels = np.array(one_hot_vector_list)
```

```
    else:
        data_json = json.load(open(os.path.join(root, test_file)))
        labels_list = data_json
        one_hot_vector_list = []
        for i in range(len(labels_list)):
            one_hot_vector = np.zeros(24).astype(int)
            for j in labels_list[i]:
                one_hot_vector[label_map[j]] = 1
            one_hot_vector_list.append(one_hot_vector)
        labels = np.array(one_hot_vector_list)

    return filenames, labels
```

2-2. Specify the hyperparameters (learning rate, epochs, etc.)

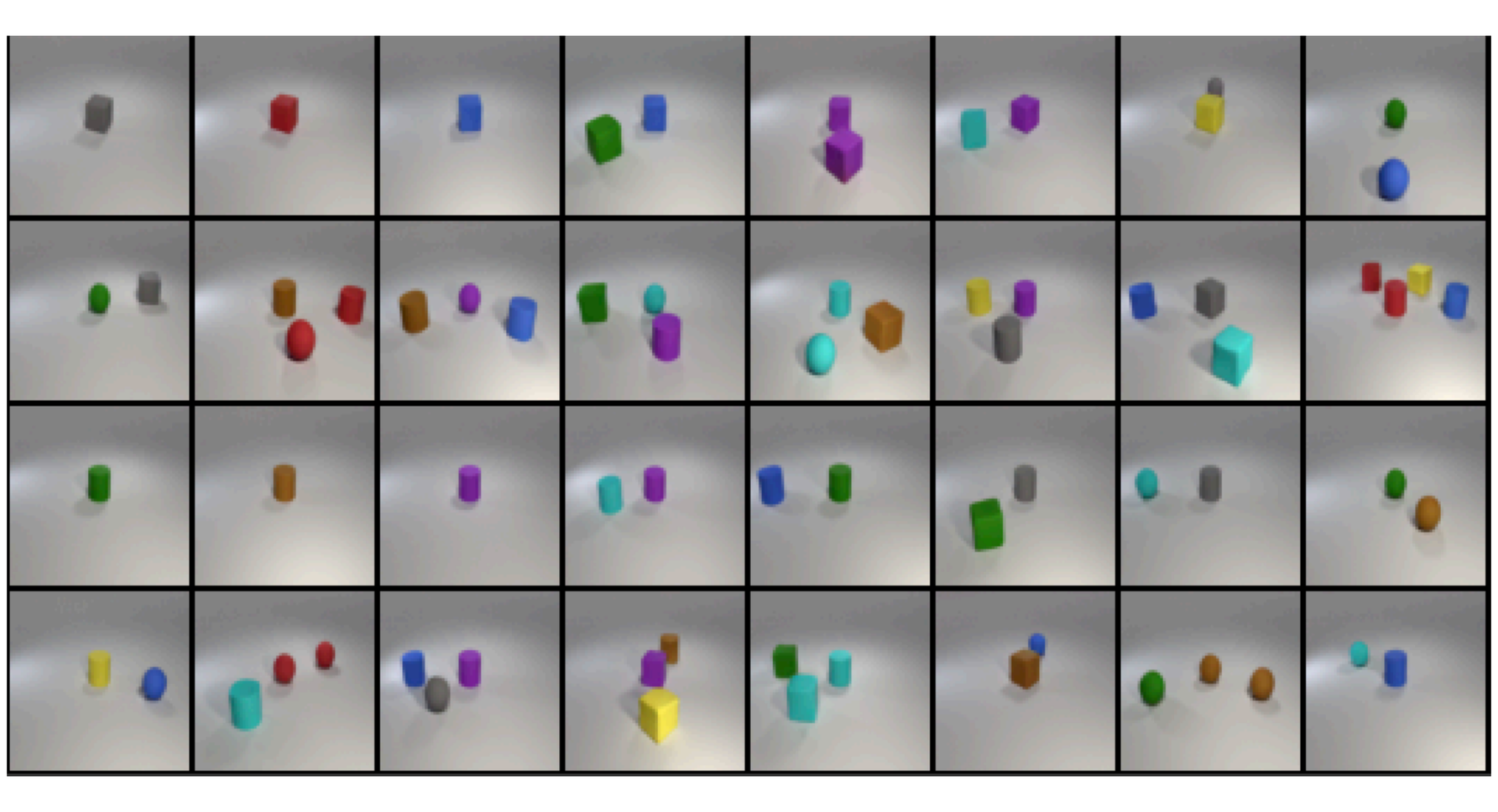
```
def parse_arg():
    parser = argparse.ArgumentParser()
    parser.add_argument('--epochs', default=60, help='Number of training epochs')
    parser.add_argument('--batch_size', default=28, help='Size of batches')
    parser.add_argument('--lr', default=0.0002, help='Learning rate')
    parser.add_argument('--c_dim', default=4, help='Condition dimension')
    parser.add_argument('--test_only', action='store_true')
    parser.add_argument('--model_path', default='ckpt', help='Path to save model checkpoint')
    parser.add_argument('--log', default='log/origin', help='Path to save log')
    parser.add_argument('--timesteps', default=1000, help='Time step for diffusion')
    parser.add_argument('--test_file', default='test.json', help='Test file')
    parser.add_argument('--test_batch_size', default=32, help='Test batch size')
    parser.add_argument('--figure_file', default='figure/origin', help='Figure file')
    parser.add_argument('--resume', default=False, help='Continue for training')
    parser.add_argument('--ckpt', default='net.pth', help='Checkpoint for network')

    return parser.parse_args()
```

3. Results and discussion

3-1. Show your results based on the testing data (including images)

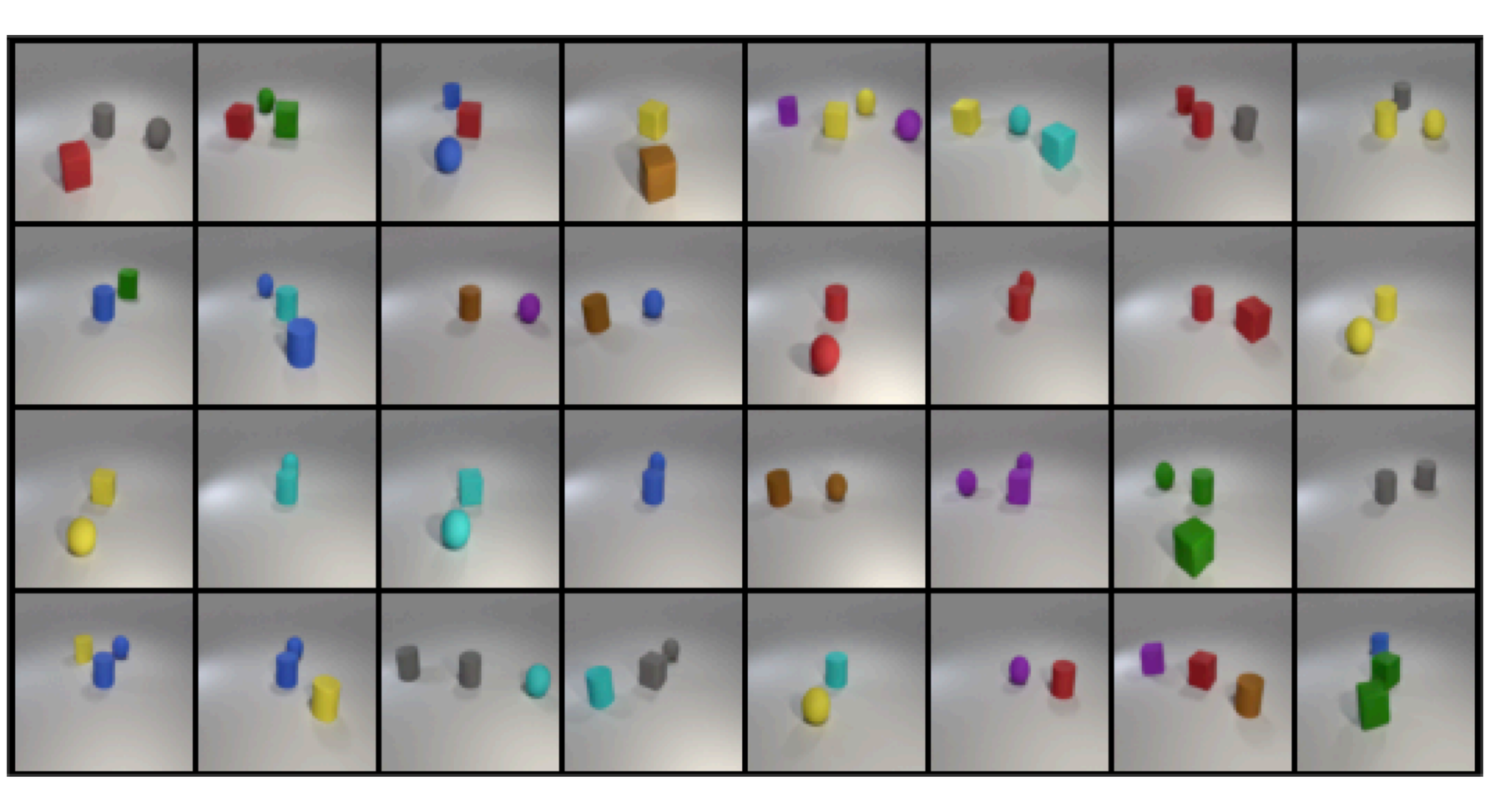
Test.json

```
(Lab) mcchou@VLLab_24_146:~/Lab/Lab7$ CUDA_VISIBLE_DEVICES=3 python main.py --te
st_only
/home/mcchou/anaconda3/envs/Lab/lib/python3.10/site-packages/torchvision/models/
_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since 0.13
and will be removed in 0.15, please use 'weights' instead.
  warnings.warn(
/home/mcchou/anaconda3/envs/Lab/lib/python3.10/site-packages/torchvision/models/
_utils.py:223: UserWarning: Arguments other than a weight enum or `None` for 'we
ights' are deprecated since 0.13 and will be removed in 0.15. The current behavi
or is equivalent to passing `weights=None`.
  warnings.warn(msg)
1000it [01:52,  8.91it/s]
acc: 0.8333333333333334
```



New_test.json

```
(Lab) mcchou@VLLab_24_146:~/Lab/Lab7$ CUDA_VISIBLE_DEVICES=6 python main.py --te
st_only --test_file 'new_test.json'
/home/mcchou/anaconda3/envs/Lab/lib/python3.10/site-packages/torchvision/models/
_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since 0.13
and will be removed in 0.15, please use 'weights' instead.
  warnings.warn(
/home/mcchou/anaconda3/envs/Lab/lib/python3.10/site-packages/torchvision/models/
_utils.py:223: UserWarning: Arguments other than a weight enum or `None` for 'we
ights' are deprecated since 0.13 and will be removed in 0.15. The current behavi
or is equivalent to passing `weights=None`.
  warnings.warn(msg)
1000it [01:51,  8.94it/s]
acc: 0.8095238095238095
```



3-2. Discuss the results of different model architectures

我總共測試了四種架構，但是都是基於 huggingface 的 UNet2DModel ，差別在於網路的 channel 數目以及 網路 down / up block type 使用 attention 的多寡，我發現網路的 channel 數目如果越以及 UNet 的深度越深效果會越好，且 attention 使用越多，對於記憶體的要求就會越大

a. One_hot:

不對 One hot label 做 embedding 直接當作 condition

```
            block_out_channels = (128, 128, 256, 256, 512, 512),
            down_block_types=(
                "DownBlock2D",        # a regular ResNet downsampling block
                "DownBlock2D",
                "DownBlock2D",
                "DownBlock2D",
                "AttnDownBlock2D",    # a ResNet downsampling block with spatial self-attention
                "DownBlock2D",
            ),
            up_block_types=(
                "UpBlock2D",
                "AttnUpBlock2D",      # a ResNet upsampling block with spatial self-attention
                "UpBlock2D",
                "UpBlock2D",          # a regular ResNet upsampling block
                "UpBlock2D",
                "UpBlock2D",
            ),
        )

    def forward(self, x, t, y):
        bs, c, w, h = x.shape

        class_cond = y.view(bs, y.shape[1], 1, 1).expand(bs, y.shape[1], w, h)
        net_input = torch.cat((x, class_cond), 1)

        return self.model(net_input, t).sample
```

Embedding:

Down / UP type 同上 , 只是使用 embedding

```
            block_out_channels = (128, 128, 256, 256, 512, 512),
            down_block_types=(
                "DownBlock2D",        # a regular ResNet downsampling block
                "DownBlock2D",
                "DownBlock2D",
                "DownBlock2D",
                "AttnDownBlock2D",    # a ResNet downsampling block with spatial self-attention
                "DownBlock2D",
            ),
            up_block_types=(
                "UpBlock2D",
                "AttnUpBlock2D",      # a ResNet upsampling block with spatial self-attention
                "UpBlock2D",
                "UpBlock2D",          # a regular ResNet upsampling block
                "UpBlock2D",
                "UpBlock2D",
            ),
        )

        self.model.class_embedding = nn.Linear(num_class, embed_size)

    def forward(self, x, t, y):
        return self.model(x, t, class_labels = y).sample
```

Embedding_DDDDAA

改變 Down / Up block type , 多使用一層 attention

```
            block_out_channels = (128, 128, 256, 256, 512, 512),
            down_block_types=(
                "DownBlock2D",        # a regular ResNet downsampling block
                "DownBlock2D",
                "DownBlock2D",
                "DownBlock2D",
                "AttnDownBlock2D",    # a ResNet downsampling block with spatial self-attention
                "AttnDownBlock2D",
            ),
            up_block_types=(
                "AttnUpBlock2D",
                "AttnUpBlock2D",      # a ResNet upsampling block with spatial self-attention
                "UpBlock2D",
                "UpBlock2D",          # a regular ResNet upsampling block
                "UpBlock2D",
                "UpBlock2D",
            ),
        )

        self.model.class_embedding = nn.Linear(num_class, embed_size)

    def forward(self, x, t, y):
        return self.model(x, t, class_labels = y).sample
```

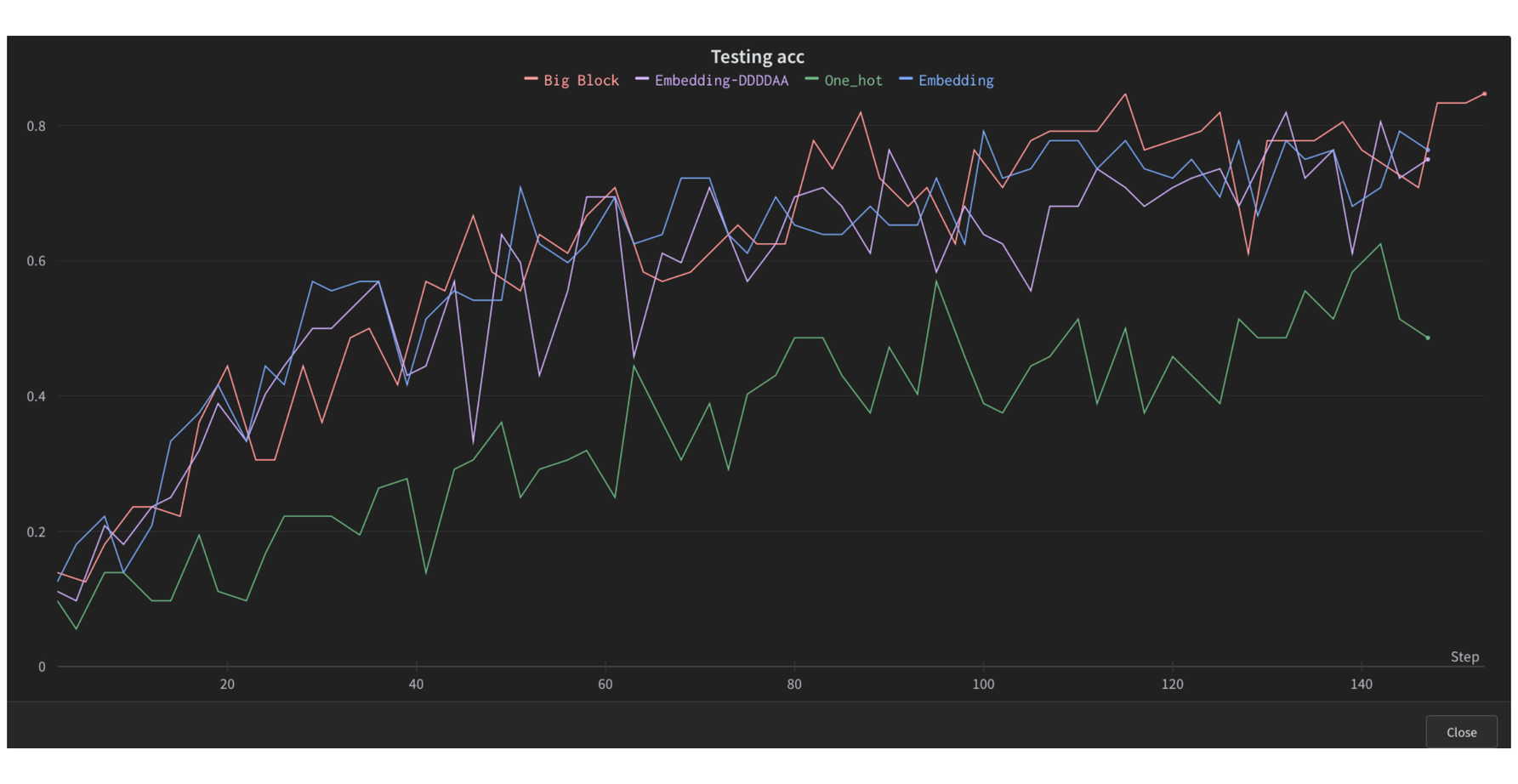
Big Block

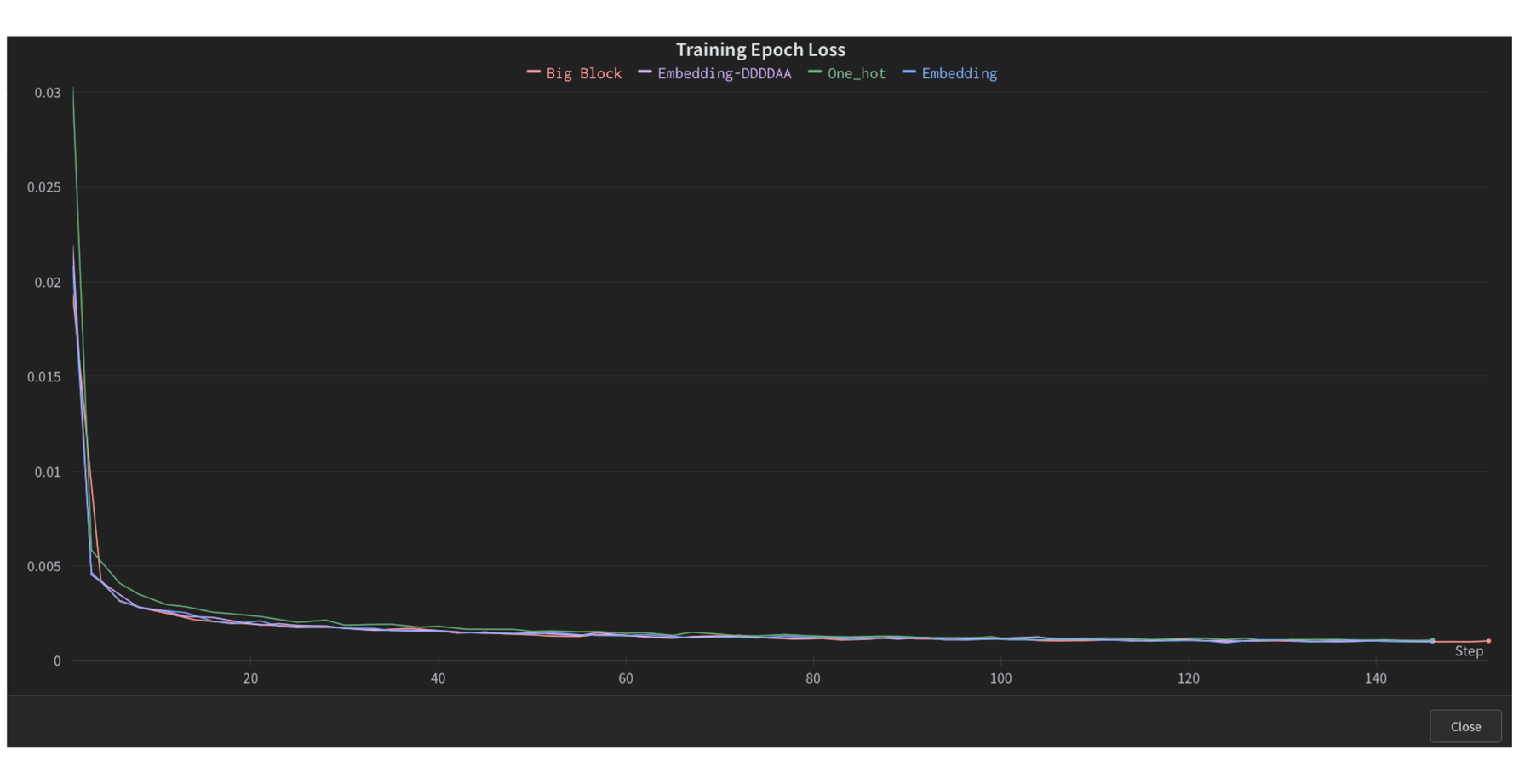
基本上架構同於 Embedding_DDDDAA 差別在於 channel 數目加大

```
            block_out_channels = (128, 256, 256, 512, 512, 1024),
            down_block_types=(
                "DownBlock2D",        # a regular ResNet downsampling block
                "DownBlock2D",
                "DownBlock2D",
                "DownBlock2D",
                "AttnDownBlock2D",    # a ResNet downsampling block with spatial self-attention
                "AttnDownBlock2D",
            ),
            up_block_types=(
                "AttnUpBlock2D",
                "AttnUpBlock2D",      # a ResNet upsampling block with spatial self-attention
                "UpBlock2D",
                "UpBlock2D",          # a regular ResNet upsampling block
                "UpBlock2D",
                "UpBlock2D",
            ),
        )

        self.model.class_embedding = nn.Linear(num_class, embed_size)

    def forward(self, x, t, y):
        return self.model(x, t, class_labels = y).sample
```

Comparison:

從結果圖來比較，可以發現單使用 one_hot vector 當作 condition 效果是最差的，另外對於網路多使用 attention block 且 channel 數目加大會有較好的效果，因此我最後使用的 model 就是採用 Big Block 的架構

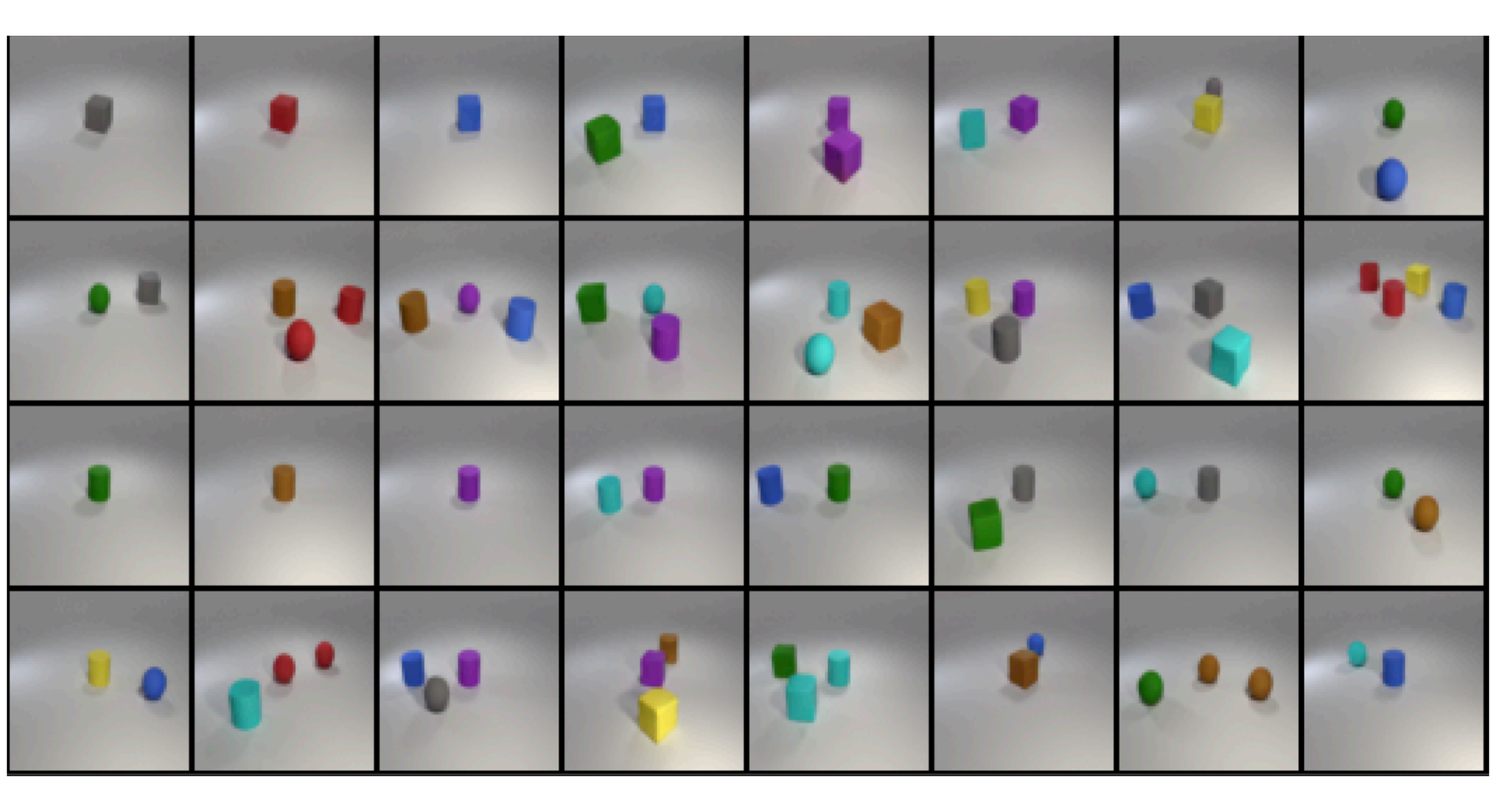




4. Experimental results (test.json and new_test.json)

Test.json


```
(Lab) mcchou@VLLab_24_146:~/Lab/Lab7$ CUDA_VISIBLE_DEVICES=3 python main.py --te
st_only
/home/mcchou/anaconda3/envs/Lab/lib/python3.10/site-packages/torchvision/models/
_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since 0.13
and will be removed in 0.15, please use 'weights' instead.
  warnings.warn(
/home/mcchou/anaconda3/envs/Lab/lib/python3.10/site-packages/torchvision/models/
_utils.py:223: UserWarning: Arguments other than a weight enum or `None` for 'we
ights' are deprecated since 0.13 and will be removed in 0.15. The current behavi
or is equivalent to passing `weights=None`.
  warnings.warn(msg)
1000it [01:52,  8.91it/s]
acc: 0.8333333333333334
```

New_test.json


```
(Lab) mcchou@VLLab_24_146:~/Lab/Lab7$ CUDA_VISIBLE_DEVICES=6 python main.py --te
st_only --test_file 'new_test.json'
/home/mcchou/anaconda3/envs/Lab/lib/python3.10/site-packages/torchvision/models/
_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since 0.13
and will be removed in 0.15, please use 'weights' instead.
  warnings.warn(
/home/mcchou/anaconda3/envs/Lab/lib/python3.10/site-packages/torchvision/models/
_utils.py:223: UserWarning: Arguments other than a weight enum or `None` for 'we
ights' are deprecated since 0.13 and will be removed in 0.15. The current behavi
or is equivalent to passing `weights=None`.
  warnings.warn(msg)
1000it [01:51,  8.94it/s]
acc: 0.8095238095238095
```