Segmentation

2025-1 Mobility UR

2025.05.08 소프트웨어학과 김유진

I. Problem Setting

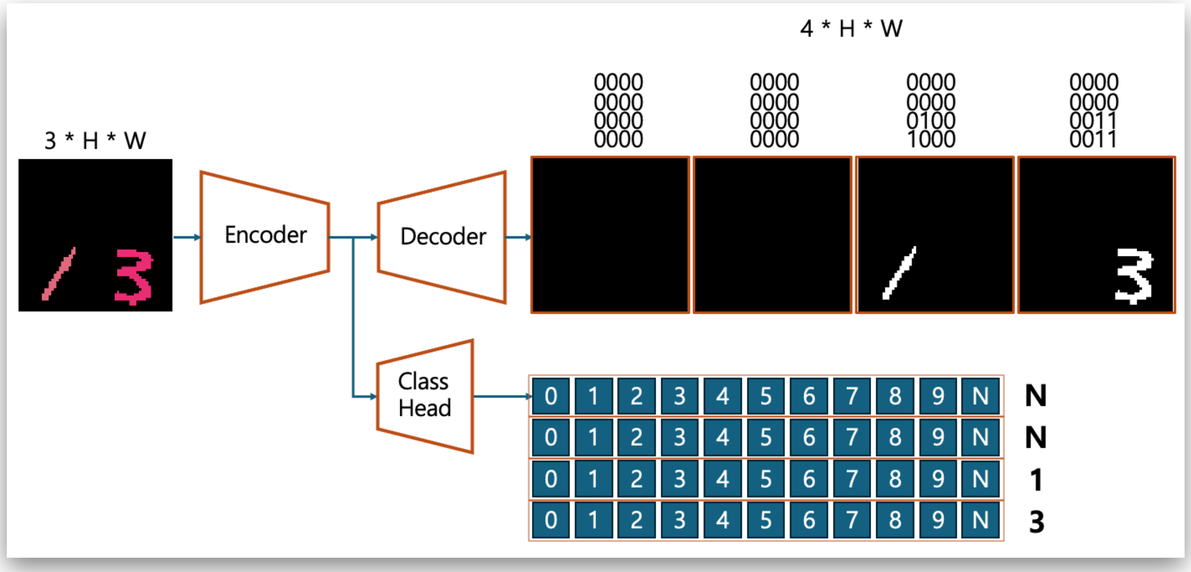
II. Encoder & Decoder

III. Classification

IV. Conclusion

Problem Setting

| 문제 정의



64x64 이미지 하나에 최대 4개의 숫자가 포함된 이미지에서

- 각 숫자의 segmentation mask 예측
- 해당 숫자 classification

Problem Setting

| Data Load

```python
def load_mask_and_label(self, path):
    mask = np.array(Image.open(path))
    binary_mask = (mask > 0).astype(np.float32)

    values = np.unique(mask) # 0 or 0 & 값+1
    values = values[values != 0] # 0이 아닌 것만 vales에 남음
    if len(values) == 0:
        label = -1
    elif len(values) == 1:
        label = values[0] - 1

    return binary_mask, label
```

```python
masks = []
labels = []
for i in range(4):
    mask_path = os.path.join(self.mask_dir, f"{base_name}_{i}.png")
    binary_mask, class_label = self.load_mask_and_label(mask_path)
    masks.append(torch.tensor(binary_mask))
    labels.append(class_label)

masks = torch.stack(masks, dim=0) # 4개를 하나의 텐서로 묶어서 (4, 64, 64) 텐서로 만듦
labels = torch.tensor(labels, dtype=torch.long)

return image, masks, labels
```

하나의 이미지에 관하여 binary mask, label 구분

- masks : 각 gt 이미지의 binary mask

  : (mask > 0).astype(np.float32)로 이진화(객체 있는 곳 0, 객체 없는 곳 1)

- labels : 각 객체의 정답 숫자

  : unique 함수를 사용해 0이 아닌 값 찾기(0만 존재 -> 객체 X)

Problem Setting

| Data Load

```
image, masks, labels = train_dataset[0]
print(image.shape)
print(masks.shape)
print(labels)
```

✓ 0.0s

```
torch.Size([3, 64, 64])
torch.Size([4, 64, 64])
tensor([ 4,  3, -1,  6])
```

Problem Setting

| Model & Train

Encoder & Decoder

- **Encoder** : CNN으로 feature 압축
- **Decoder** : CNN 기반의 4개의 디코더를 개별 학습
  - Hungarian Matching 방식 사용: [디코더 1개 : gt 1] -> 1:1 매칭 방식
- **Loss** : Encoder에서 입력받는 이미지를 Decoder에서 복원 시키는게 목적

  → binary mask와 Decoder의 결과를 비교하기 위해 BCELoss 사용

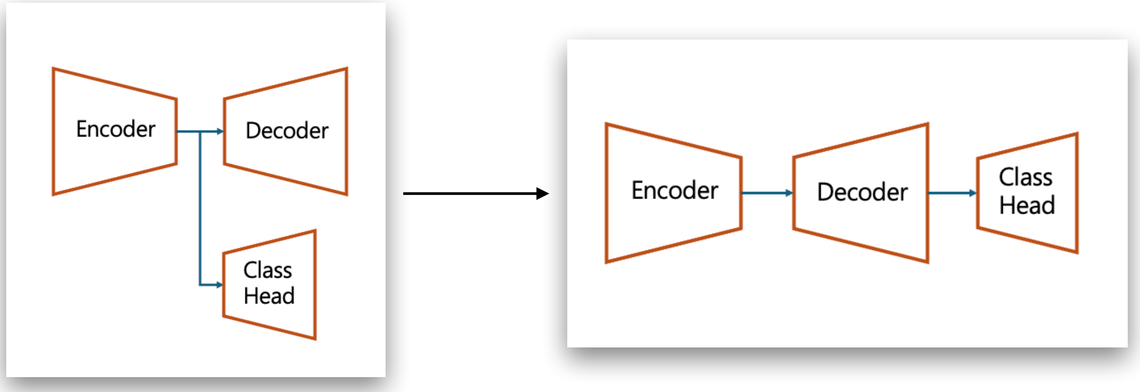
  (입력 이미지와 똑같이 복원 했는지 픽셀 단위 판별 가능)

Classification

- **Classification** : test loss 에 따른 Multi moel 모델을 저장 후

  → 해당 모델 Classification에 사용

- **Loss** : 클래스가 총 11개로 다중 분류 → Cross Entropy Loss 사용



I. Problem Setting

II. Encoder & Decoder

III. Classification

IV. Conclusion

Encoder & Decoder

| Model

```python
class Encoder(nn.Module):
    def __init__(self):
        super().__init__()
        self.encoder = nn.Sequential(
            nn.Conv2d(3, 16, 3, stride=2, padding=1),
            nn.ReLU(),
            nn.Conv2d(16, 32, 3, stride=2, padding=1),
            nn.ReLU(),
            nn.Conv2d(32, 64, 3, stride=1, padding=1),
            nn.ReLU(),
            nn.Conv2d(64, 128, 3, stride=1, padding=1),
            nn.ReLU(),
            nn.Conv2d(128, 128, 3, stride=1, padding=1),
            nn.ReLU(),
            nn.Conv2d(128, 32, 7),
        )

    def forward(self, x):
        return self.encoder(x)
```

```python
class Decoder(nn.Module):
    def __init__(self):
        super().__init__()
        self.decoder = nn.Sequential(
            nn.ConvTranspose2d(32, 128, 7),
            nn.ReLU(),
            nn.ConvTranspose2d(128, 64, 3, stride=1, padding=1),
            nn.ReLU(),
            nn.ConvTranspose2d(64, 32, 3, stride=1, padding=1),
            nn.ReLU(),
            nn.ConvTranspose2d(32, 16, 3, stride=2, padding=1, output_padding=1),
            nn.ReLU(),
            nn.ConvTranspose2d(16, 1, 3, stride=2, padding=1, output_padding=1),
            nn.Sigmoid()
        )

    def forward(self, x):
        return self.decoder(x)
```

```python
class MultiSegNet(nn.Module):
    def __init__(self, num_instances=4):
        super().__init__()
        self.encoder = Encoder()
        self.decoders = nn.ModuleList([Decoder() for _ in range(num_instances)]) # Decoder 최대 객체 수 만큼 분할
    def forward(self, x):
        f = self.encoder(x)
        return torch.stack([dec(f) for dec in self.decoders], dim=1)  # (B, 4, 1, 64, 64) -> B : batch
```

Encoder & Decoder

| Hungarian Matching

```
'''
    TODO : Hungarian Matching 방식의 유사도 측정
        : pred_mask, true_mask 사이 겹치는 정도 수치로 나타냄
        : IoU = 교집합 / 합집합 ( 0 ~ 1(완전 일치))
'''
def compute_iou(pred_mask, true_mask, eps=1e-6):
    intersection = (pred_mask * true_mask).sum()
    union = pred_mask.sum() + true_mask.sum() - intersection
    return (intersection + eps) / (union + eps)
```

pred_mask와 true_mask 사이의 겹치는 정도를 수치로 표현

```
# IoU 기반 cost matrix 생성 (4x4)
cost_matrix = torch.zeros(4, 4)
for i in range(4):
    for j in range(4):
        if labels[j] == -1:
            cost_matrix[i, j] = 1.0  # 객체가 없는 경우에는 큰 비용
        else:
            inter = (preds[i, 0] * masks[j]).sum()
            union = preds[i, 0].sum() + masks[j].sum() - inter
            iou = (inter + 1e-6) / (union + 1e-6)
            cost_matrix[i, j] = 1 - iou  # 비용: 1 - IoU(1 - 완전 일치)

row_ind, col_ind = linear_sum_assignment(cost_matrix.numpy())
```

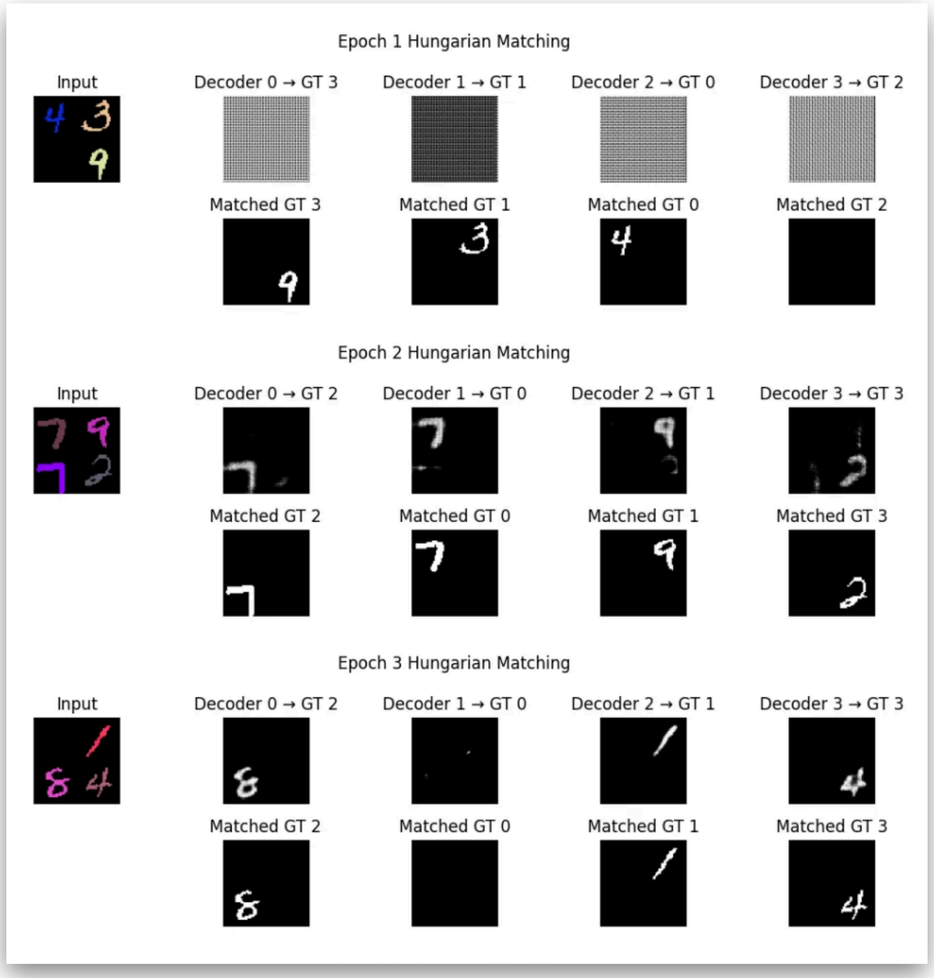
IoU 기반의 cost matrix 생성 → cost (1-IoU) 값이 작을수록 마스크 비슷

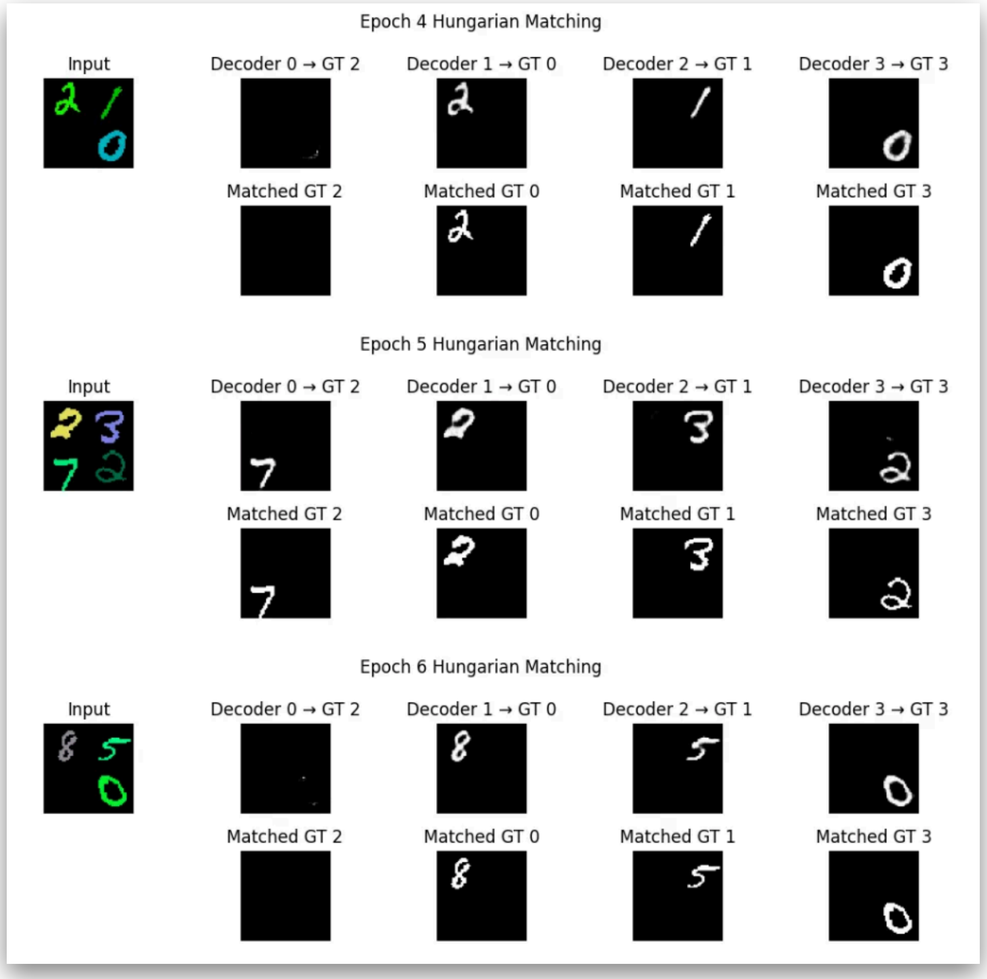
Hungarian Algorithm 구현한 linear_sum_assignment 사용

→ cost가 최소가 되도록 Decoder과 gt 1:1 매칭

Encoder & Decoder

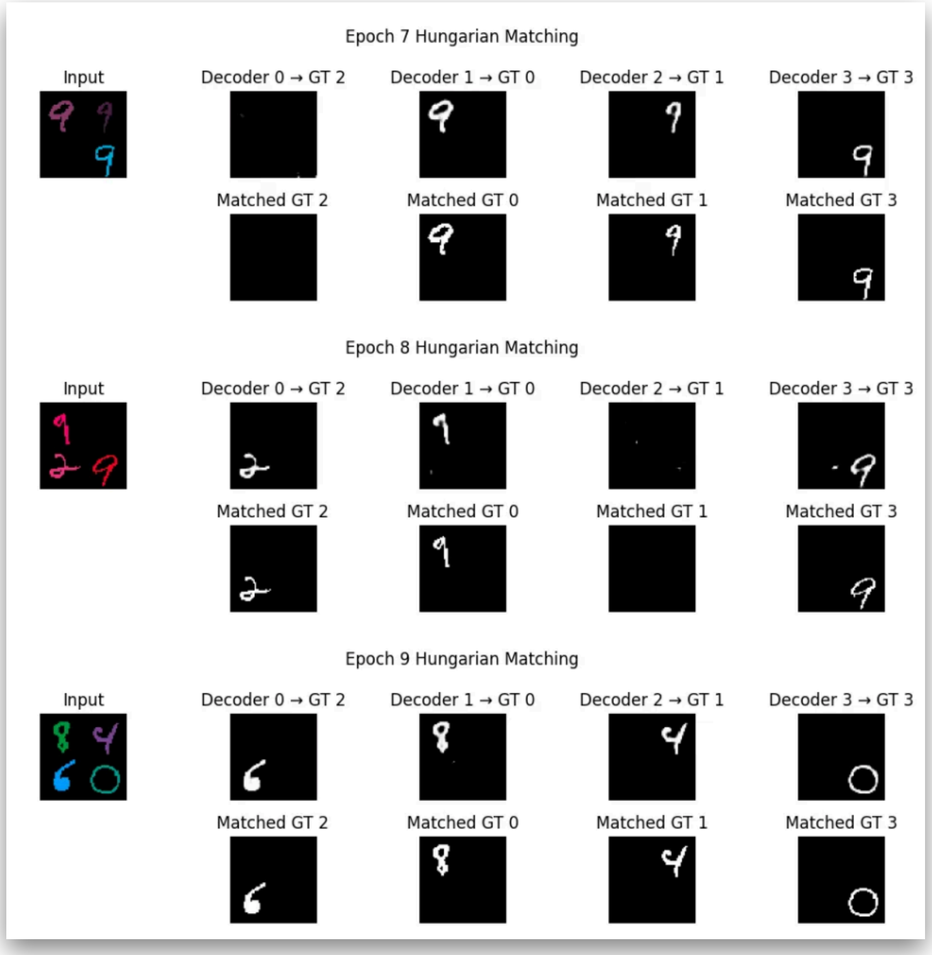
| Batch size = 4

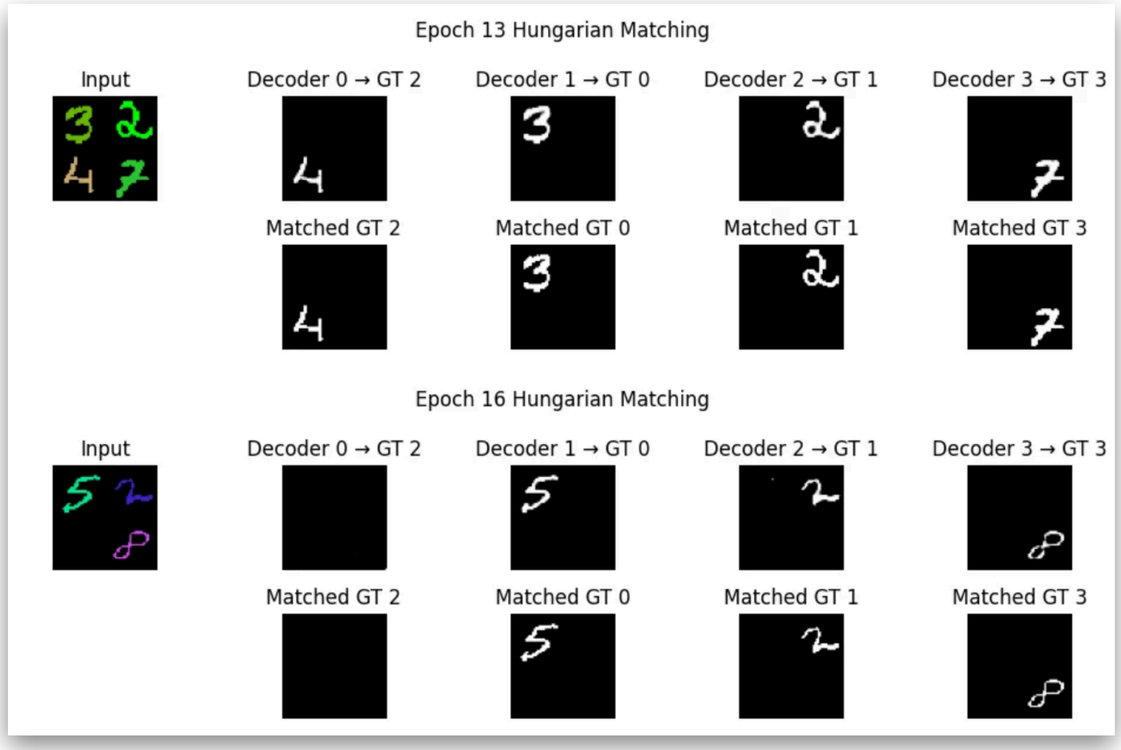




Encoder & Decoder

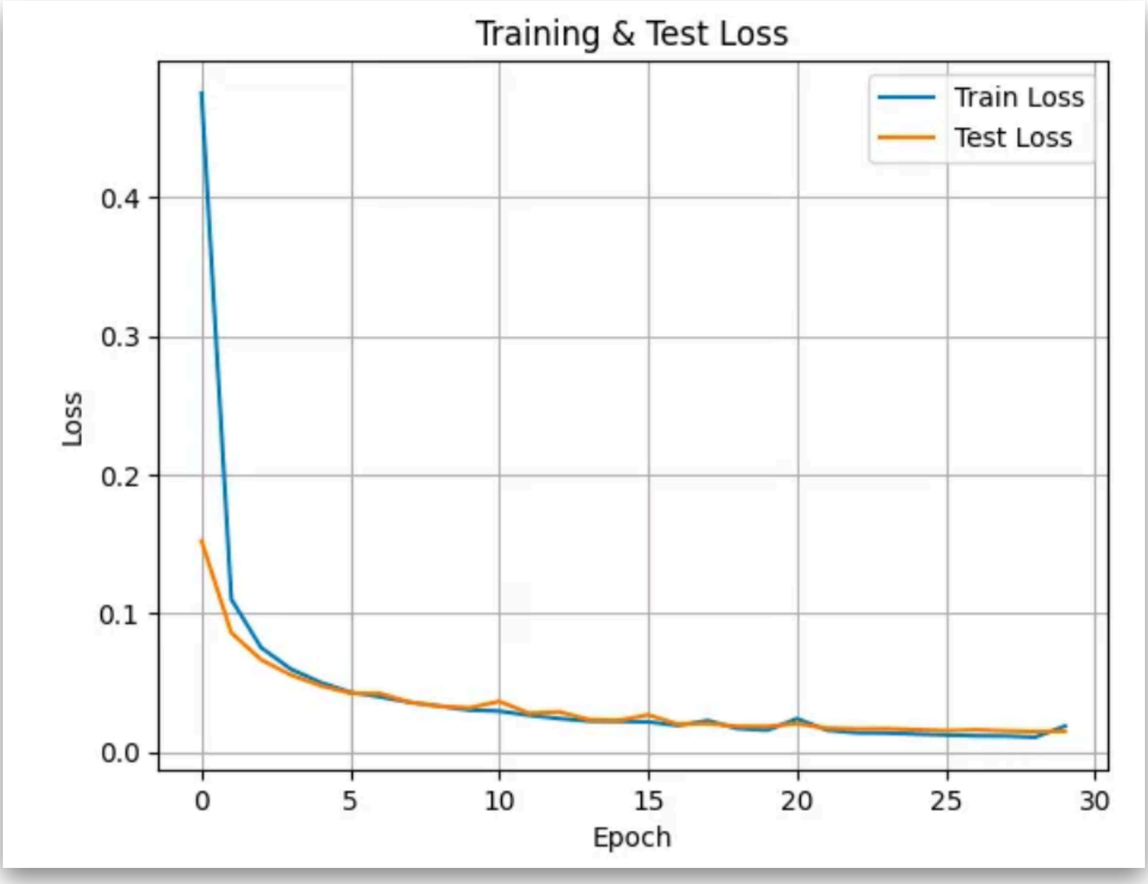
| Batch size = 4





Encoder & Decoder

| Batch size = 4

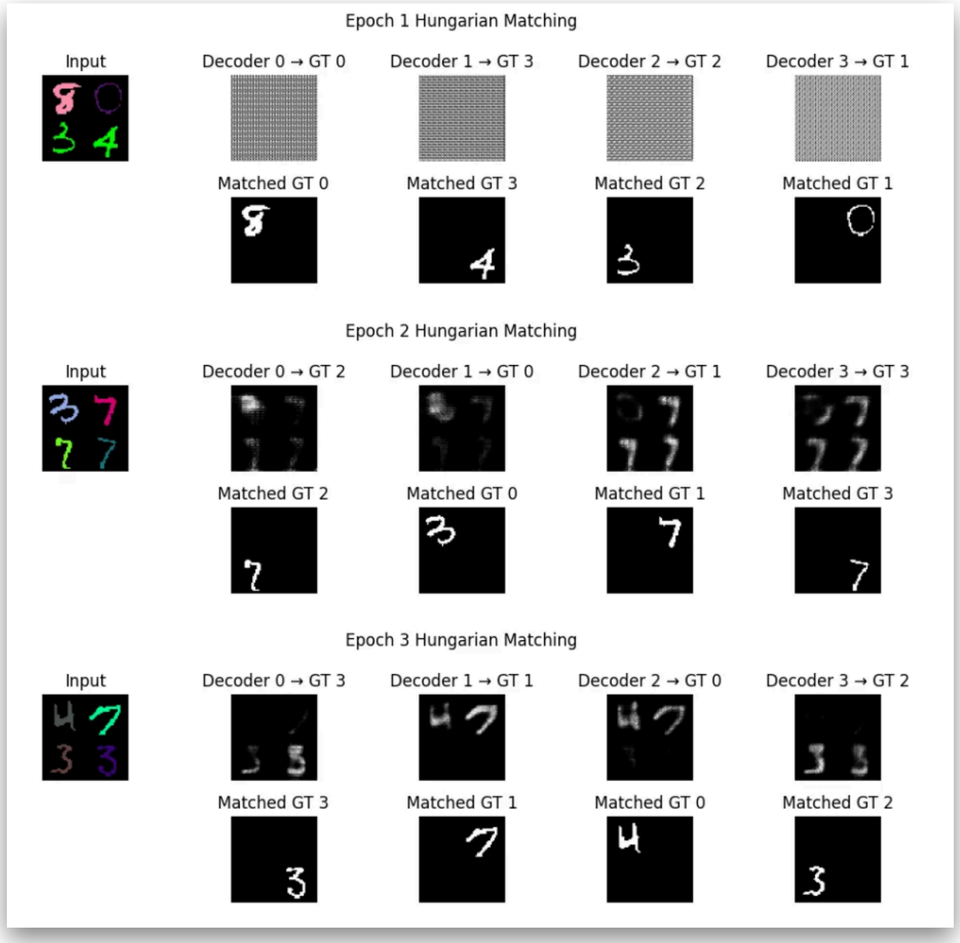


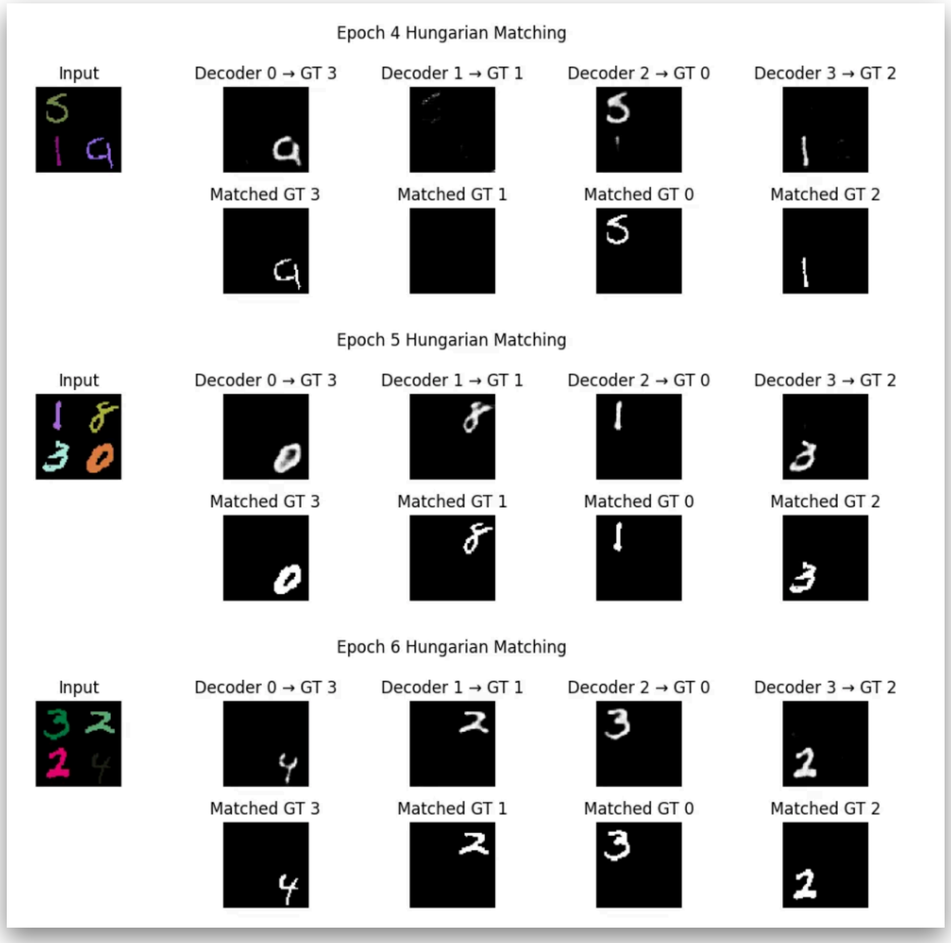
Min Train Loss : 0.0108

Min Test Loss : 0.0149

Encoder & Decoder

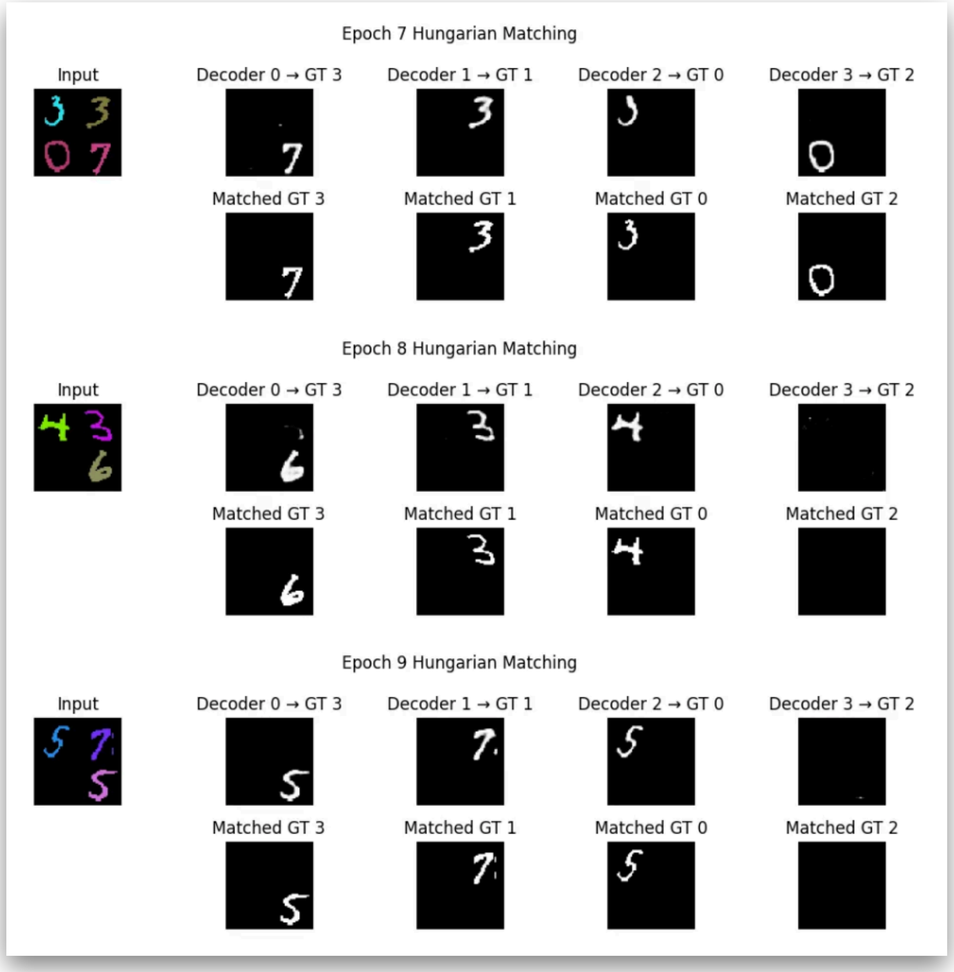
| Batch size = 8

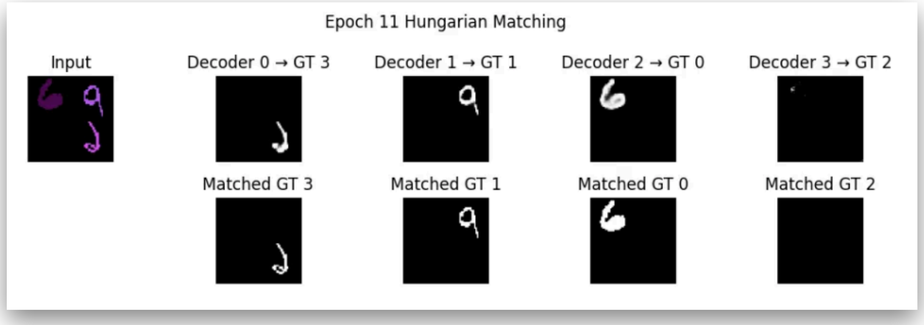


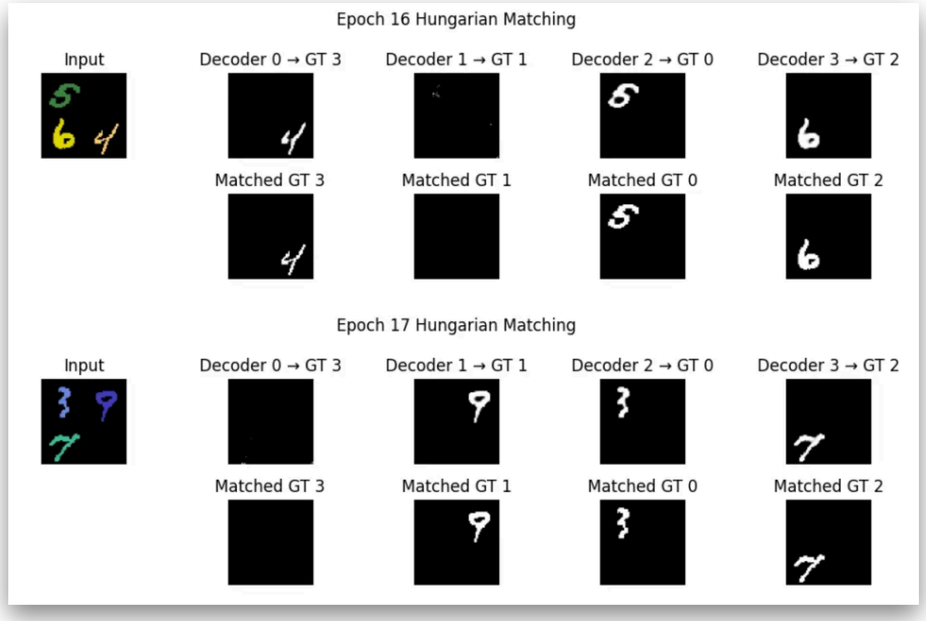


Encoder & Decoder

| Batch size = 8

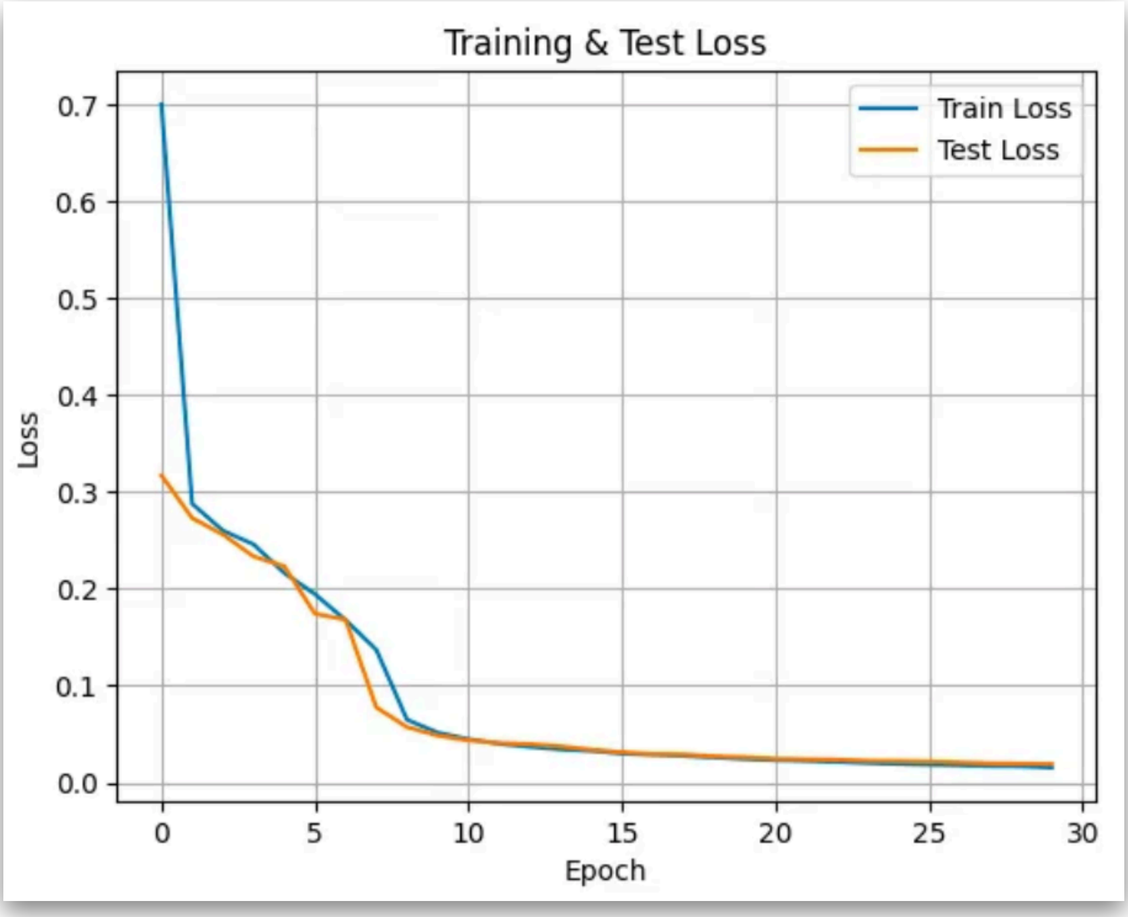






Encoder & Decoder

| Batch size = 8



Min Train Loss : 0.0156

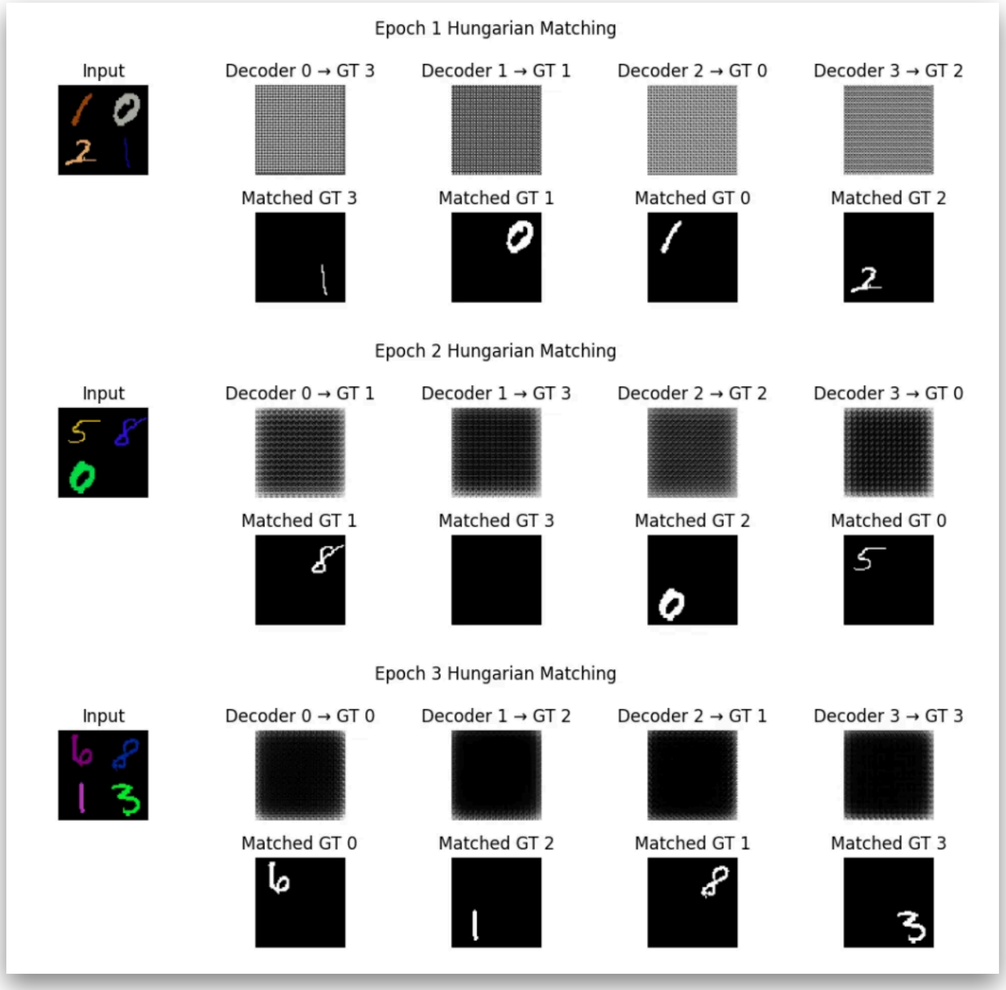
Min Test Loss : 0.0193

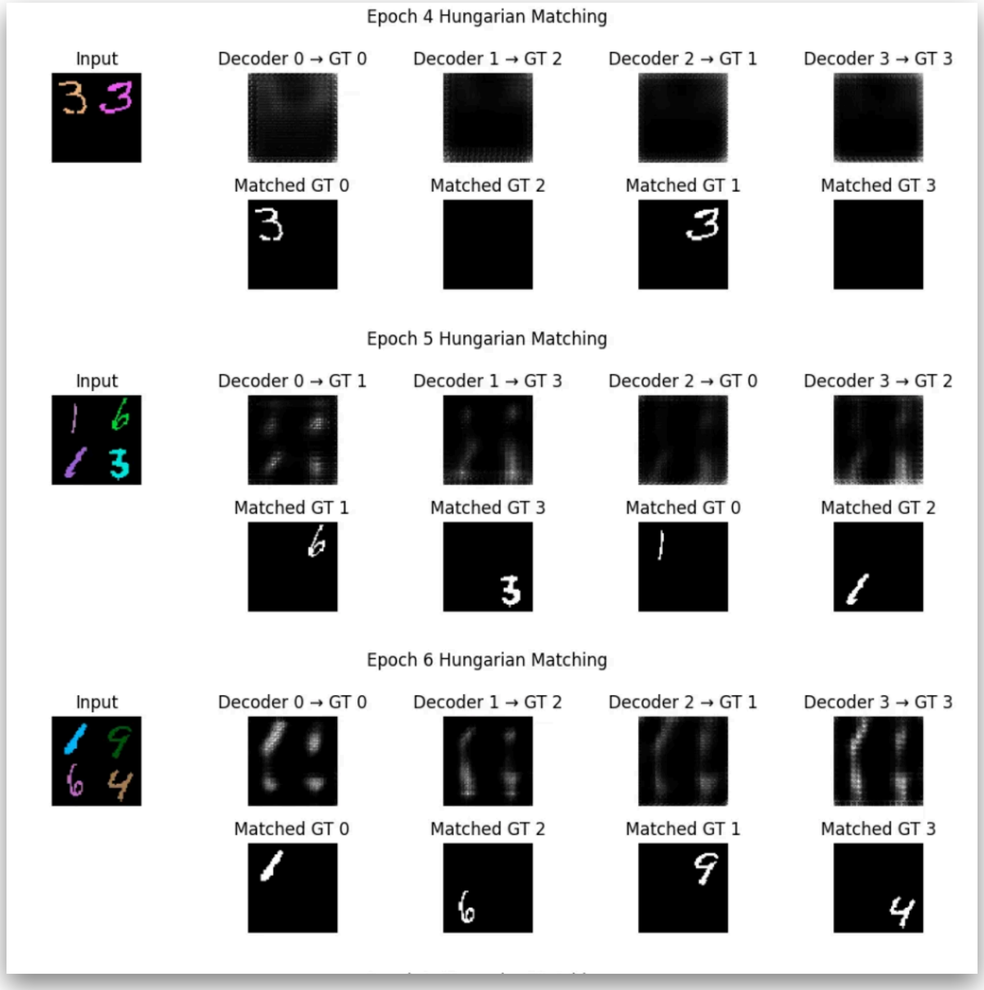
batch_size = 4

- Min Train Loss : 0.0108
- Min Test Loss : 0.0149

Encoder & Decoder

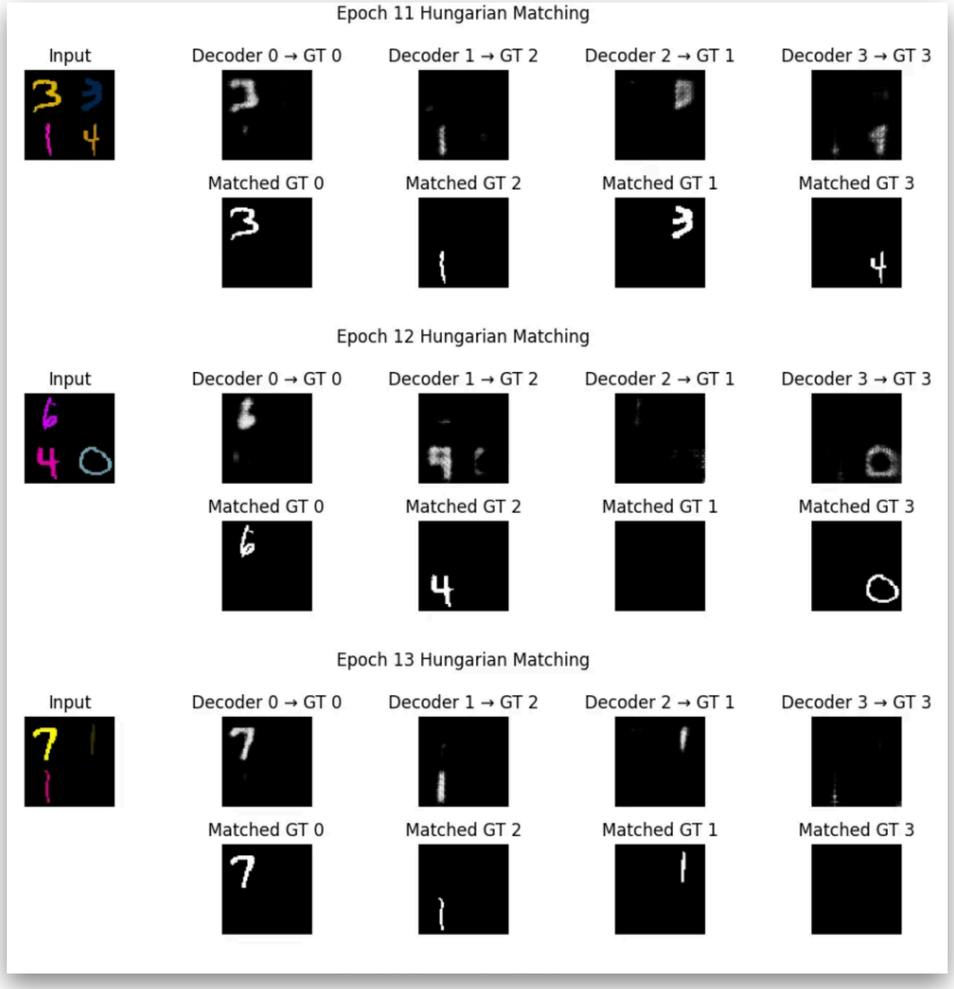
| Batch size = 64

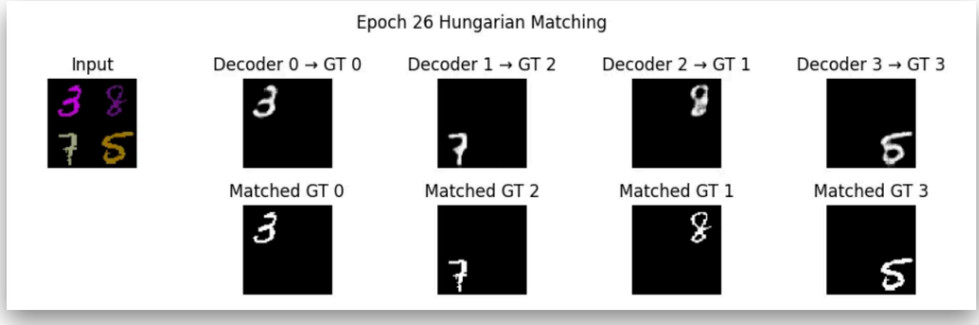


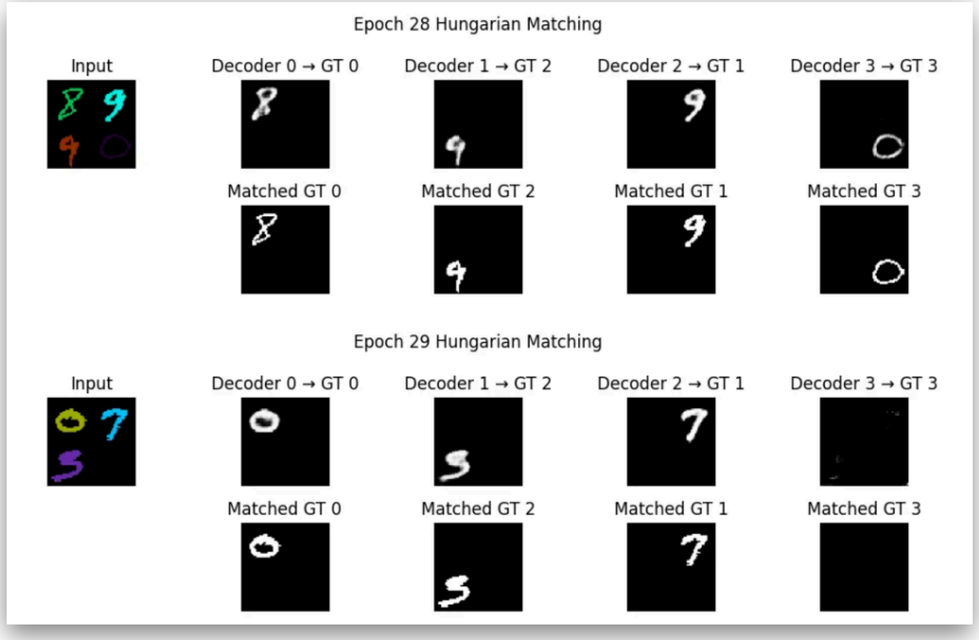


Encoder & Decoder

| Batch size = 64

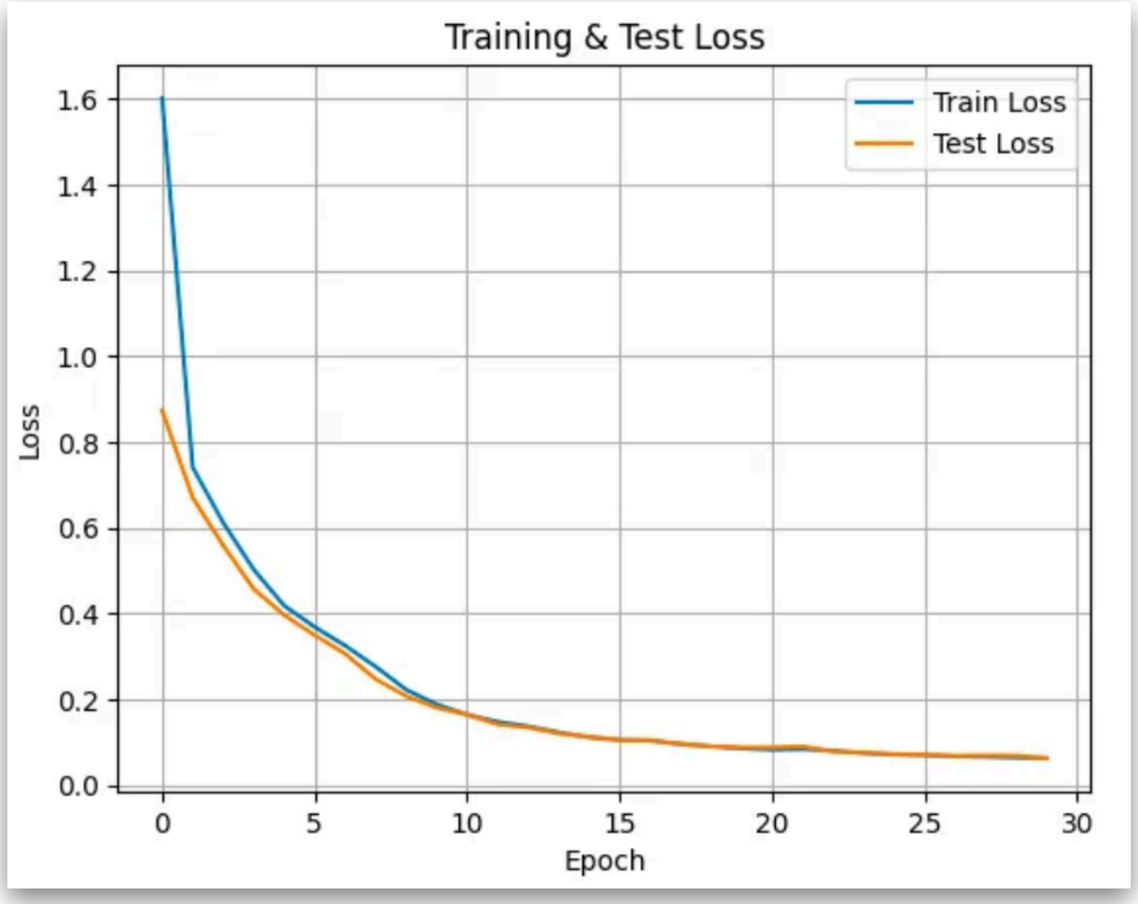






Encoder & Decoder

| Batch size = 64



Min Train Loss : 0.0642

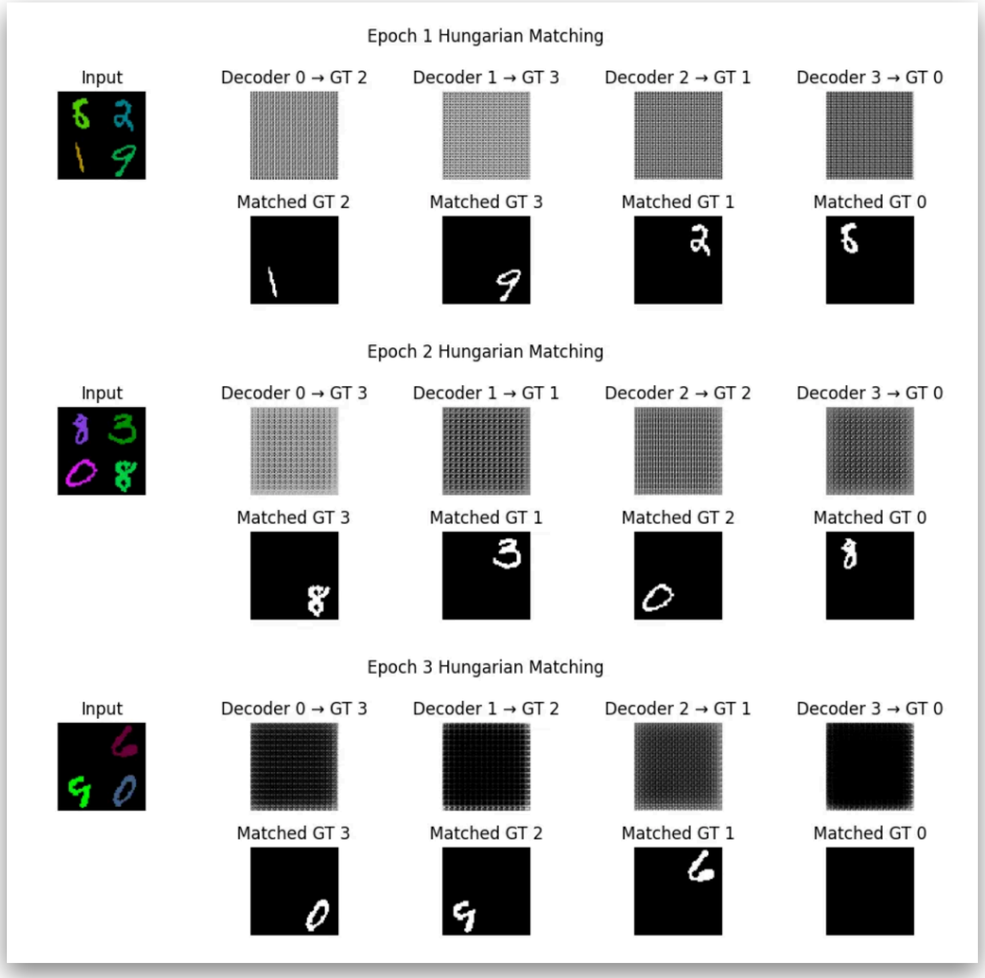
Min Test Loss : 0.0624

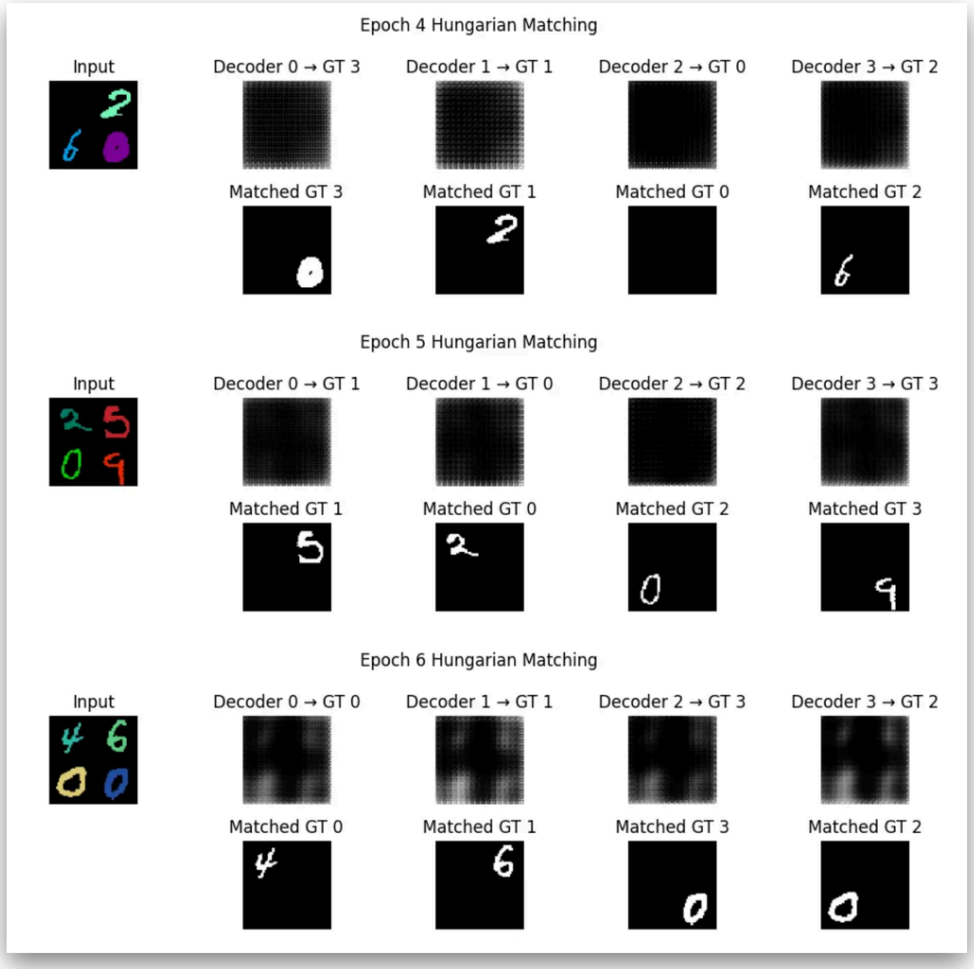
batch_size = 4

- Min Train Loss : 0.0108
- Min Test Loss : 0.0149

Encoder & Decoder

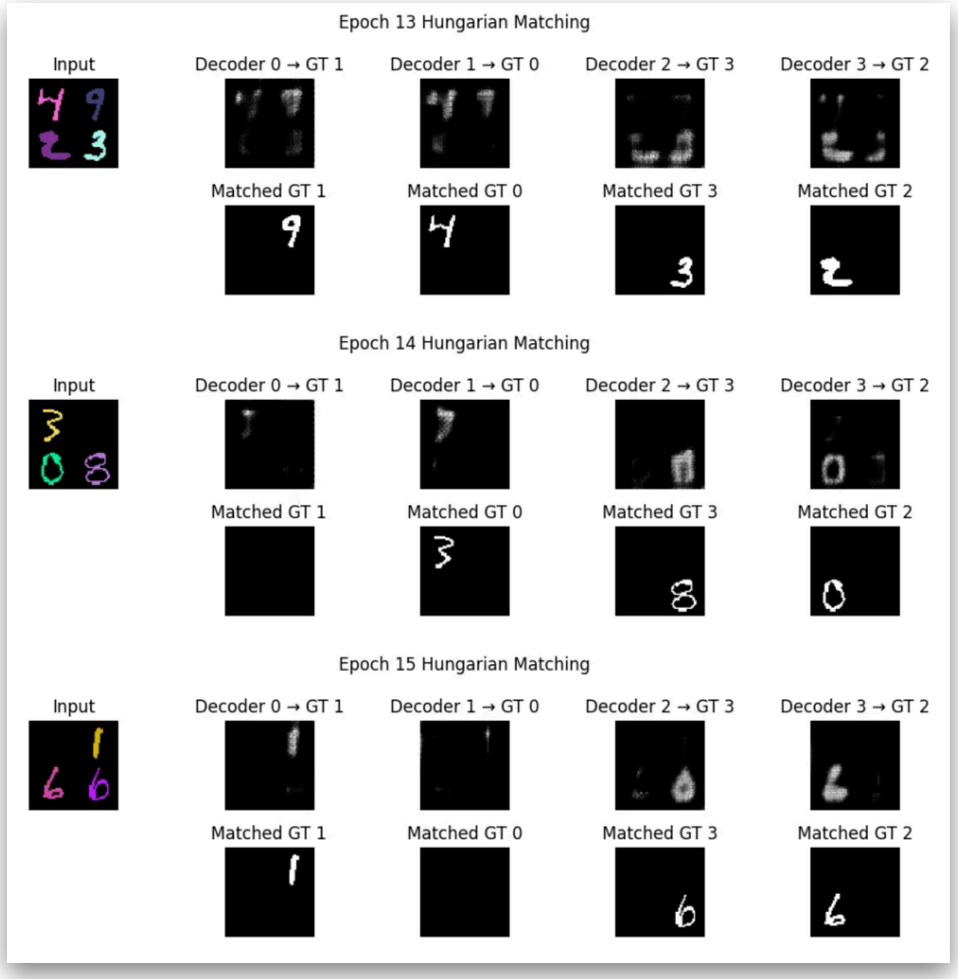
| Batch size = 128

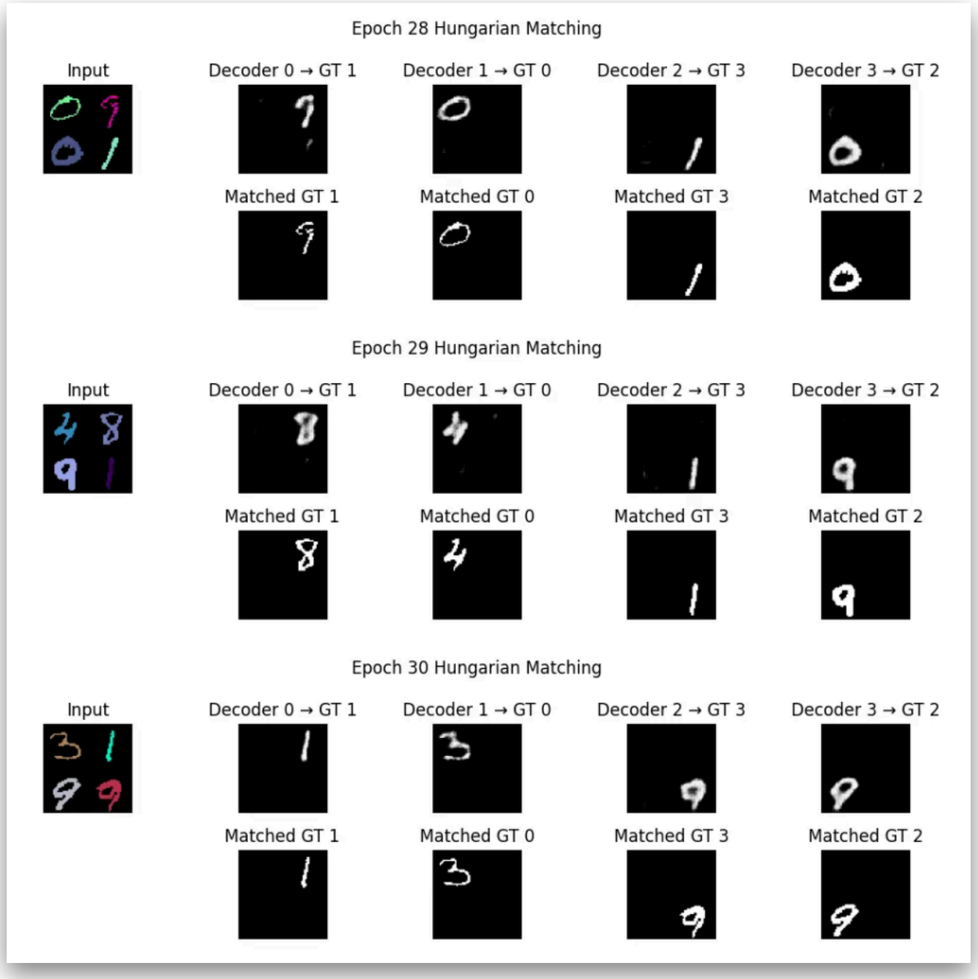




Encoder & Decoder

| Batch size = 128





Encoder & Decoder

| Batch size = 128

Min Train Loss : 0.0870

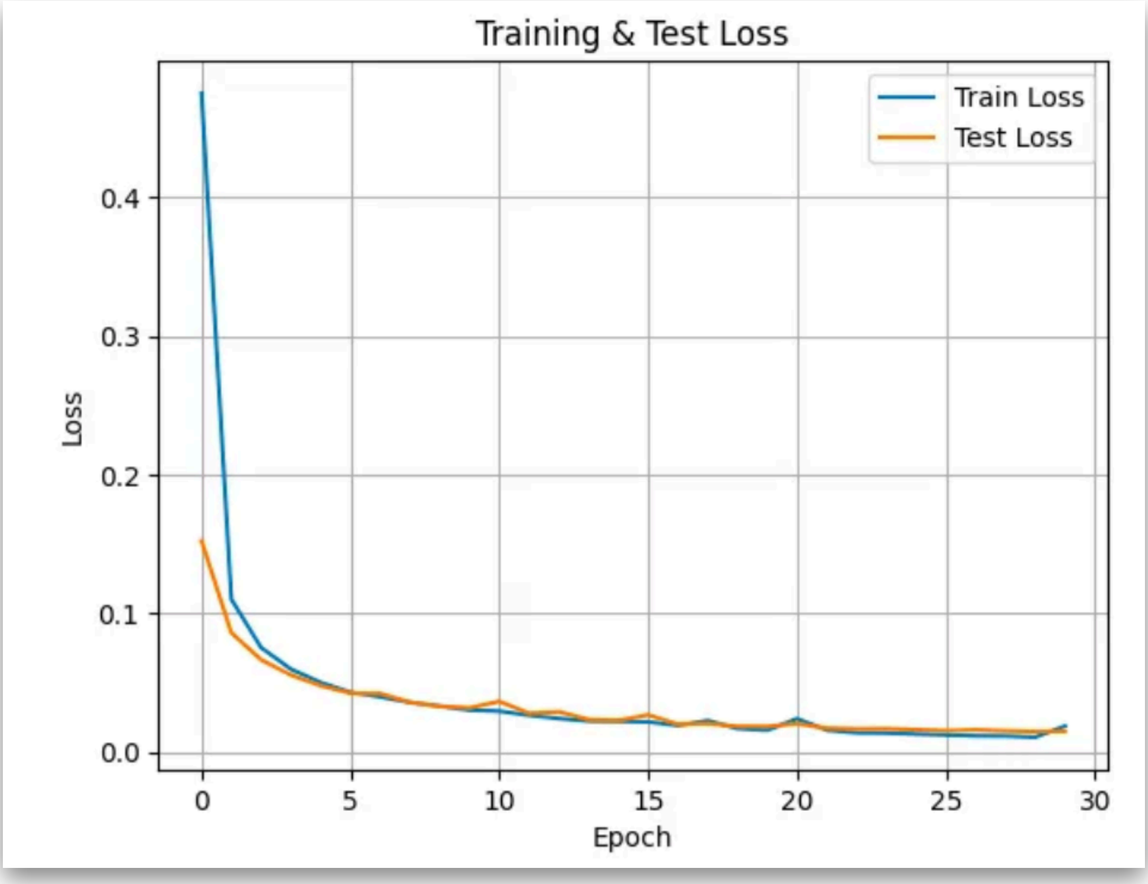
Min Test Loss : 0.0872

batch_size = 4

- Min Train Loss : 0.0108
- Min Test Loss : 0.0149

Encoder & Decoder

| Learning rate = 0.01

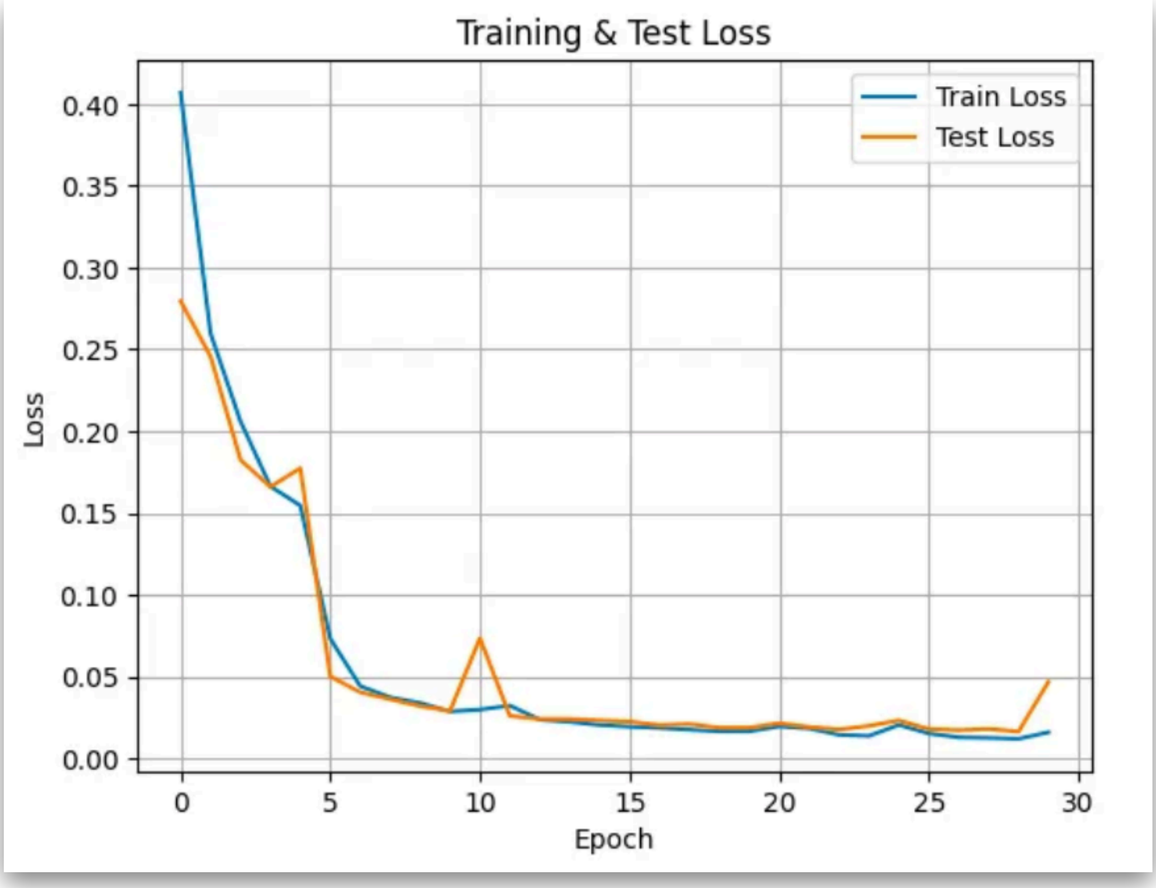


Min Train Loss : 0.0108

Min Test Loss : 0.0149

Encoder & Decoder

| Learning rate = 0.03



Min Train Loss : 0.0122

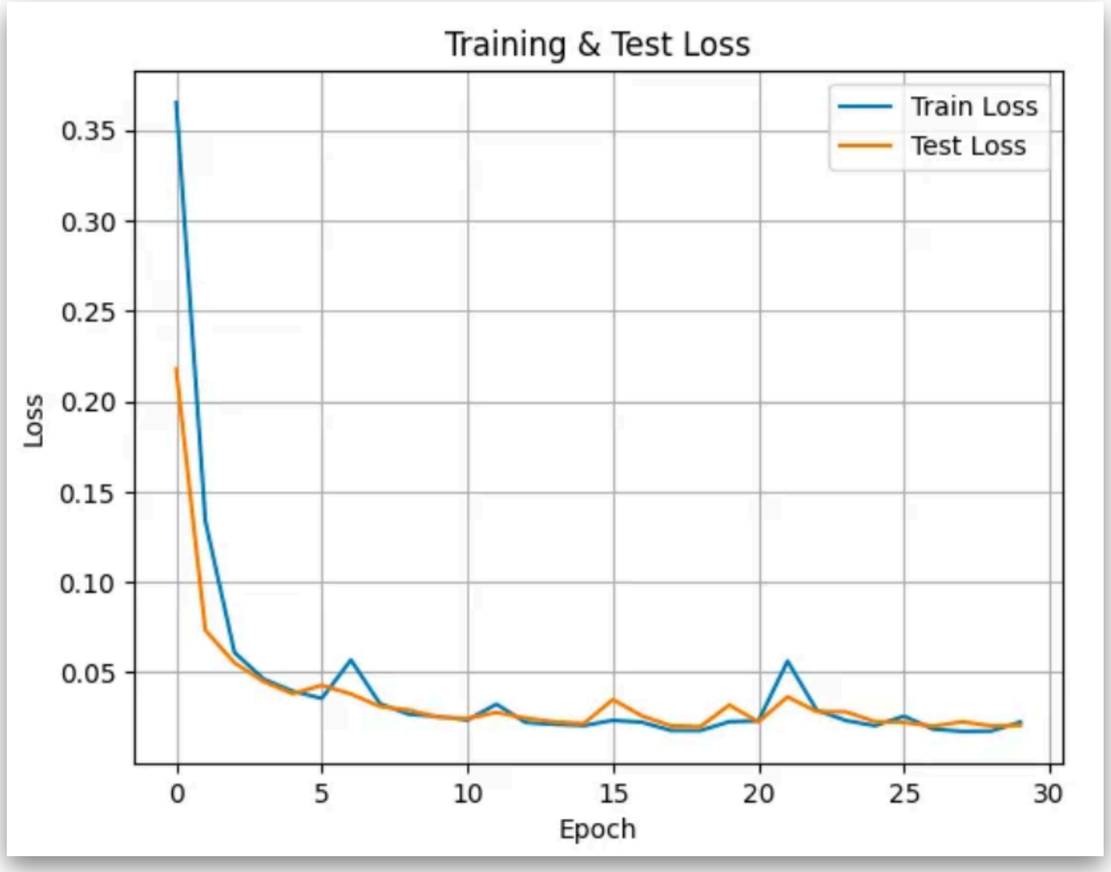
Min Test Loss : 0.0166

Learning rate = 0.01

- Min Train Loss : 0.0108
- Min Test Loss : 0.0149

Encoder & Decoder

| Learning rate = 0.05



Min Train Loss : 0.0174

Min Test Loss : 0.0200

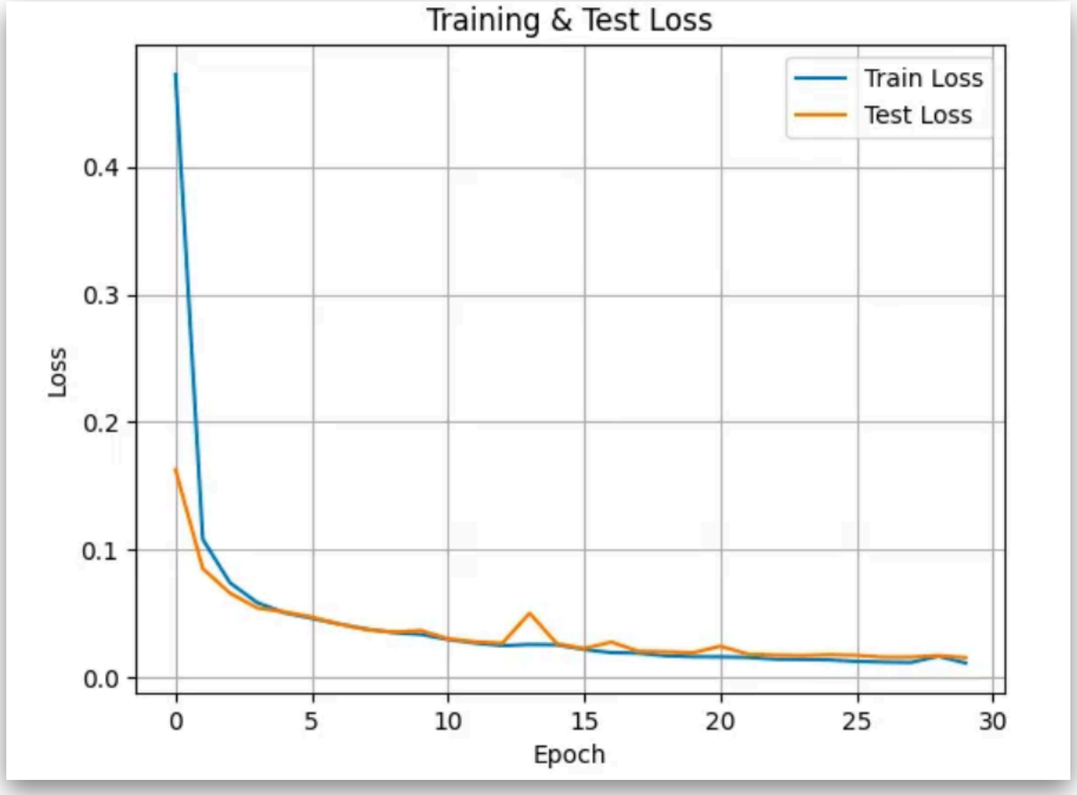
Learning rate = 0.01

- Min Train Loss : 0.0108
- Min Test Loss : 0.0149

Encoder & Decoder

| Learning rate = 0.01 & LR scheduler

ReduceLROnPlateau 방식 (lr - 0.001, patient : 5, factor = 0.5)



Min Train Loss : 0.0115

Min Test Loss : 0.0157

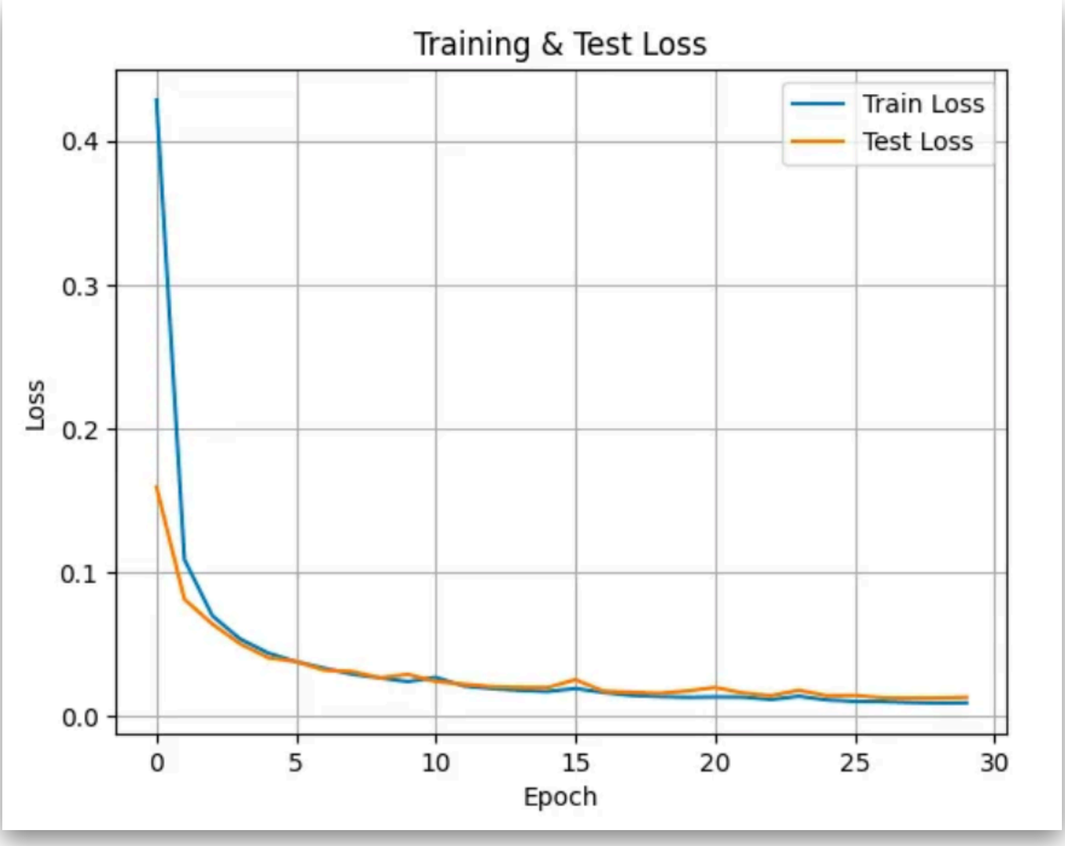
Learning rate = 0.01

- Min Train Loss : 0.0108
- Min Test Loss : 0.0149

Encoder & Decoder

| Learning rate = 0.01 & LR scheduler

ReduceLROnPlateau 방식 (lr - 0.001, patient : 3, factor = 0.5)



Min Train Loss : 0.0088

Min Test Loss : 0.0123

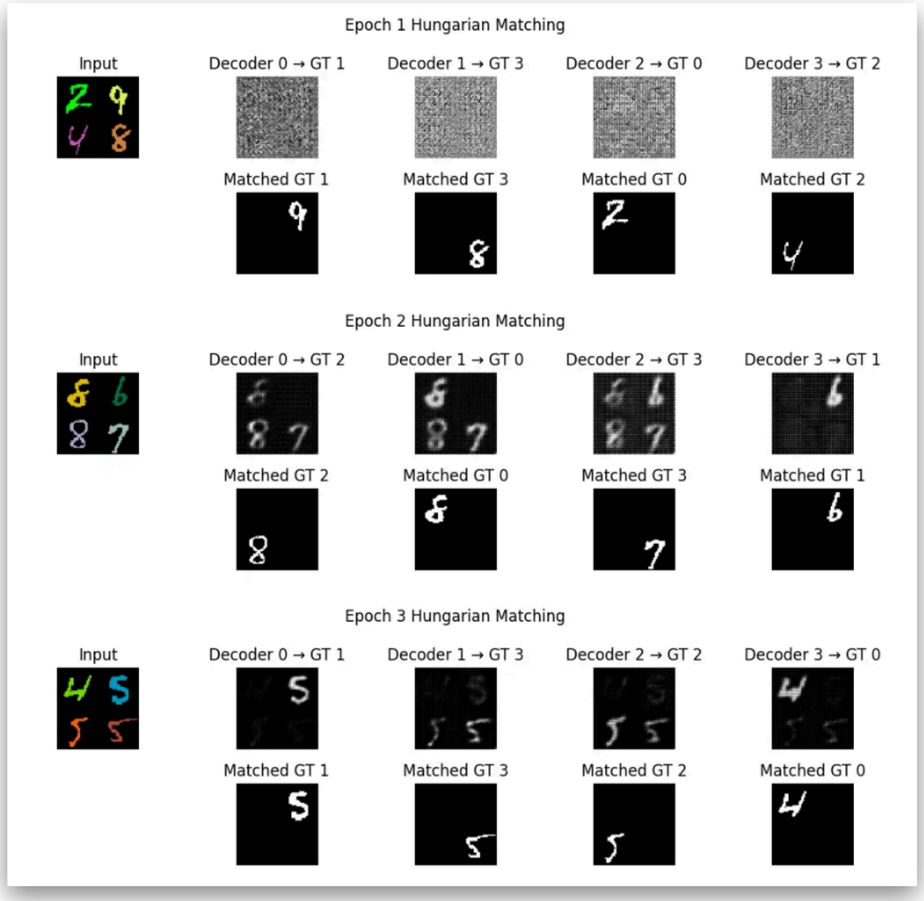
Learning rate = 0.01

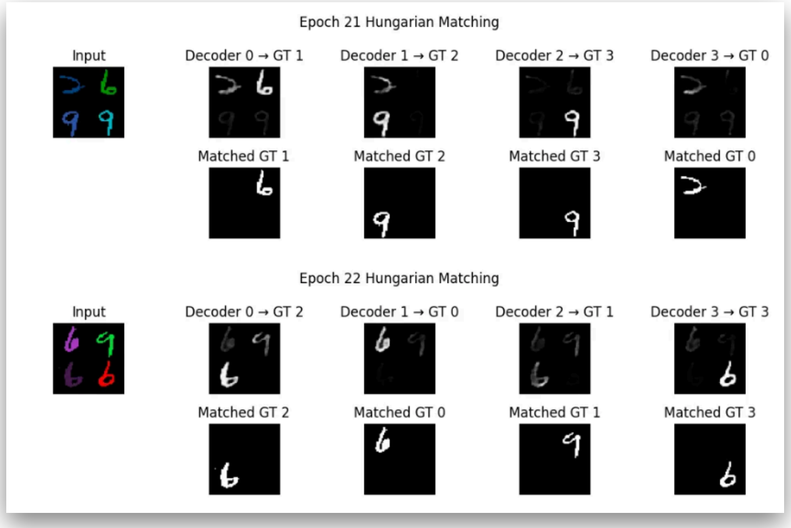
- Min Train Loss : 0.0108
- Min Test Loss : 0.0149

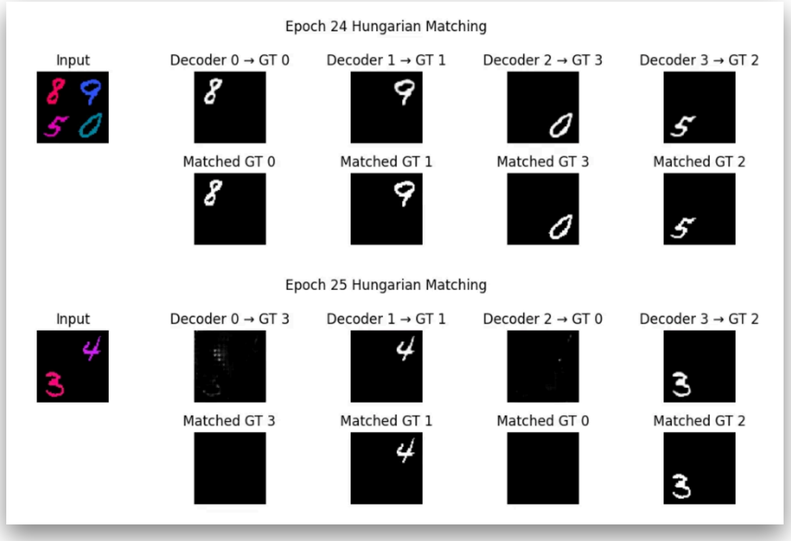
Encoder & Decoder

| Batch Normalize

Decoder Batch Normalize O



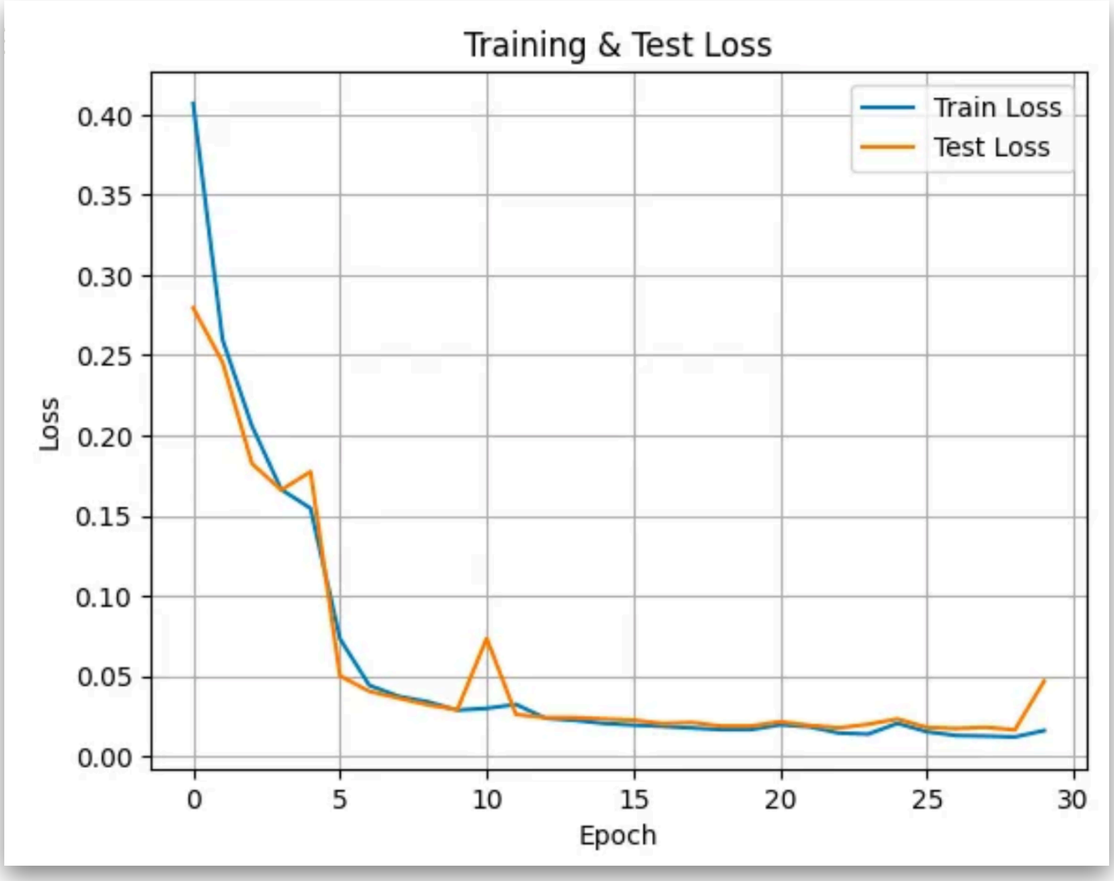




Encoder & Decoder

| Batch Normalize

Decoder Batch Normalize O



Min Train Loss : 0.0123

Min Test Loss : 0.0154

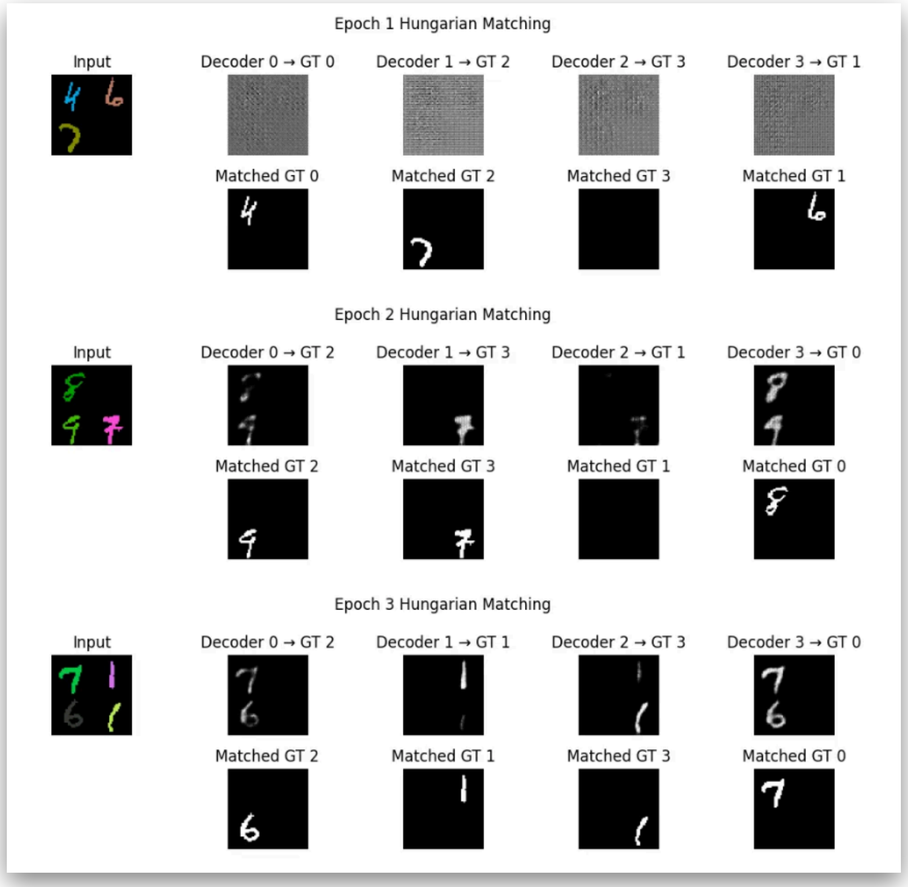
Learning rate = 0.01 & LR scheduler

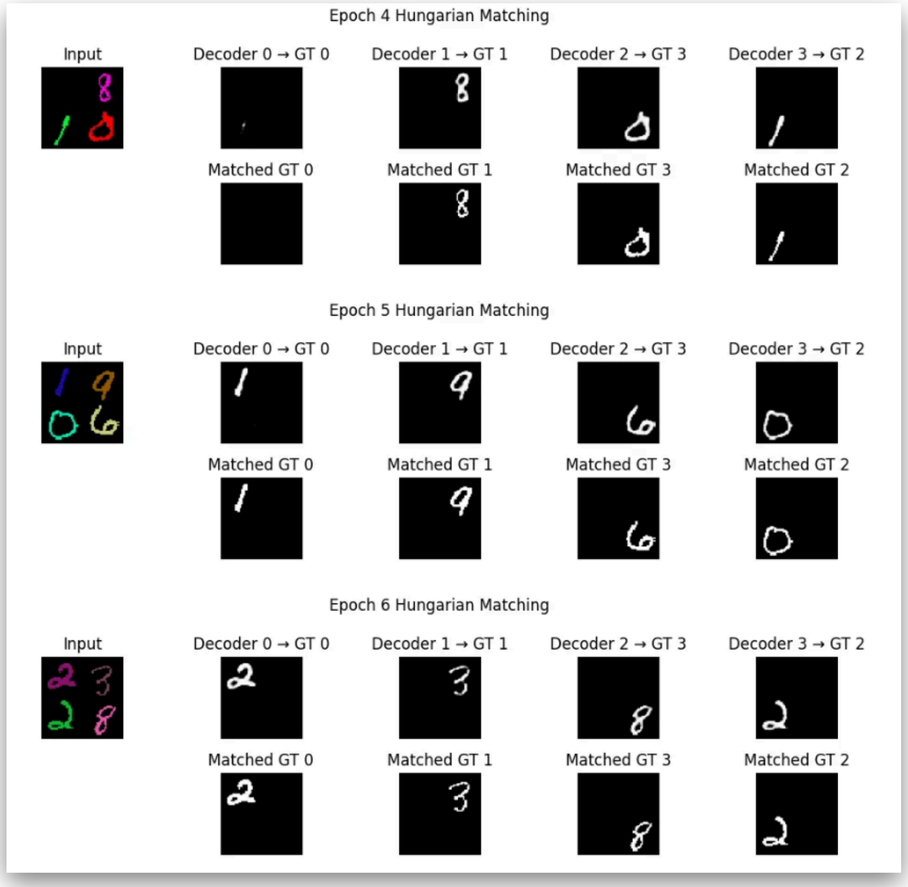
- Min Train Loss : 0.0088
- Min Test Loss : 0.0123

Encoder & Decoder

| Batch Normalize

Decoder Batch Normalize X

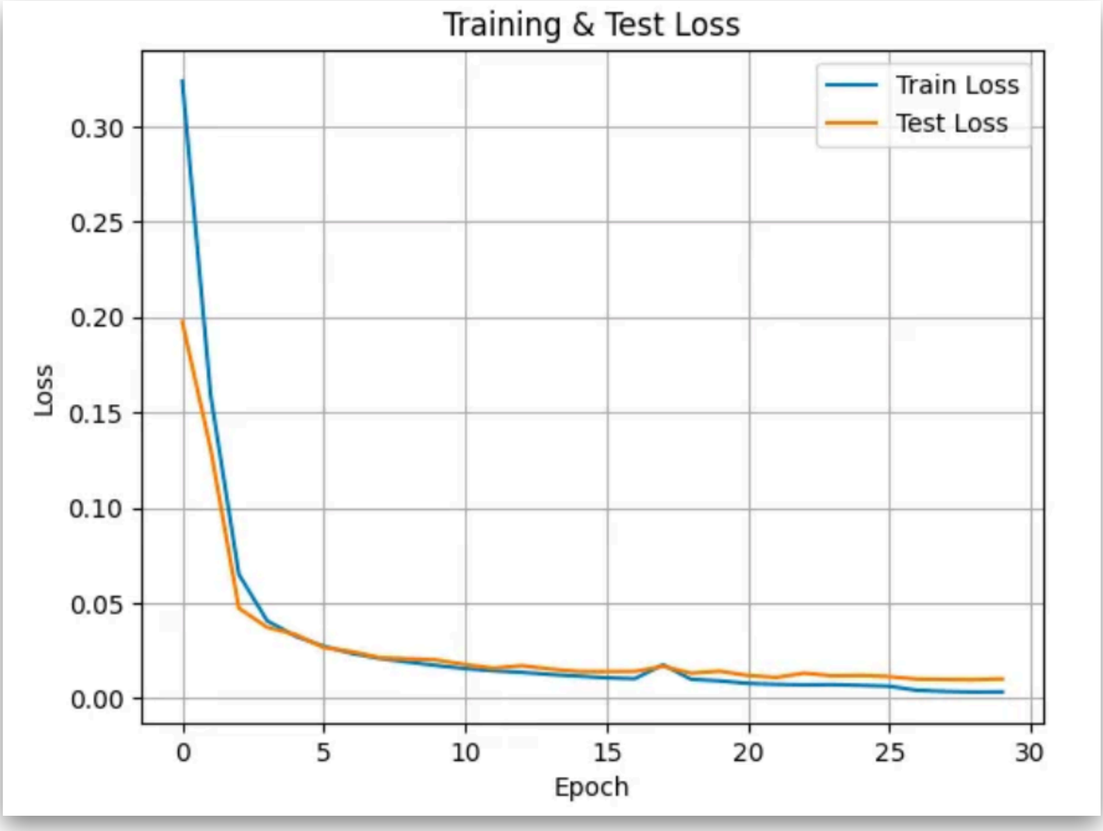




Encoder & Decoder

| Batch Normalize

Decoder Batch Normalize X



Min Train Loss : 0.0033

Min Test Loss : 0.0097

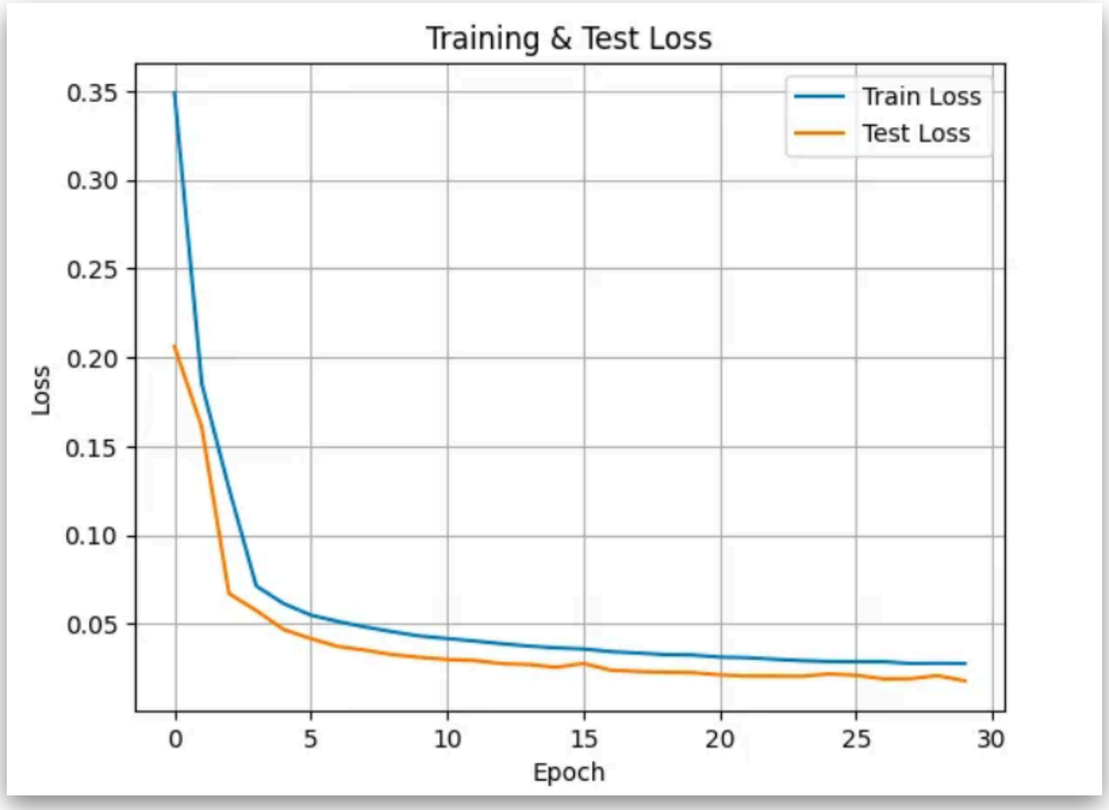
Learning rate = 0.01 & LR scheduler

- Min Train Loss : 0.0088
- Min Test Loss : 0.0123

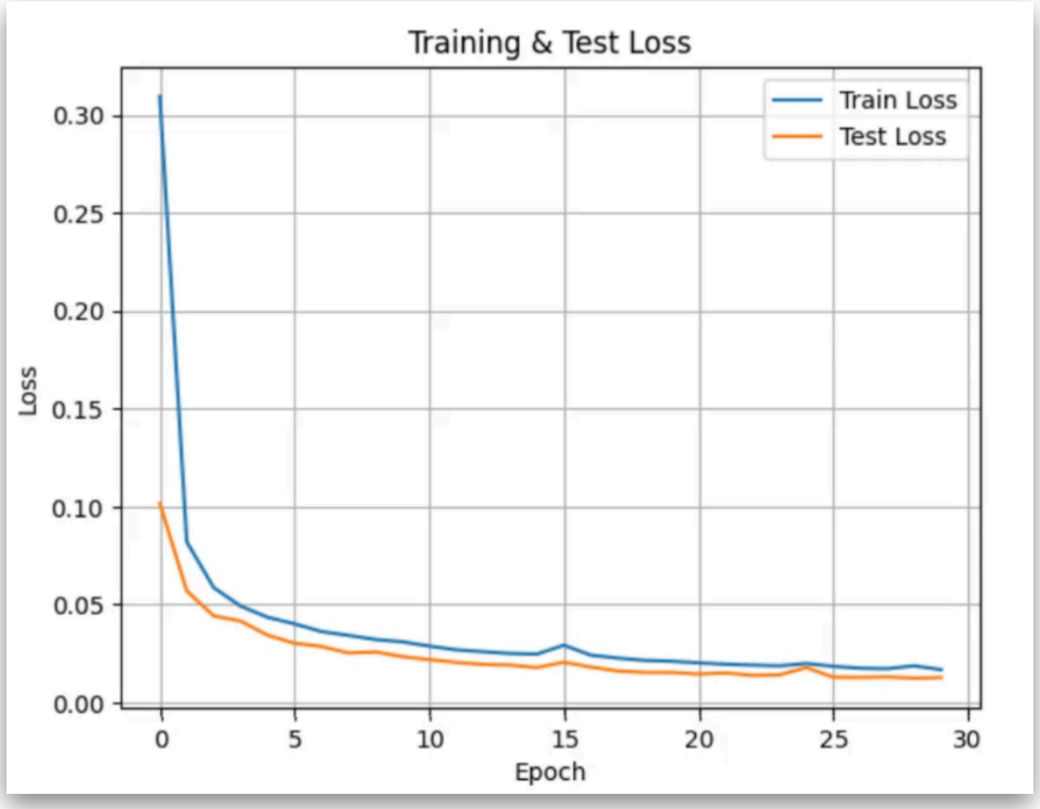
Encoder & Decoder

| Batch Normalize + Dropout(0.1) - (Decoder X)

Dropout = 0.2

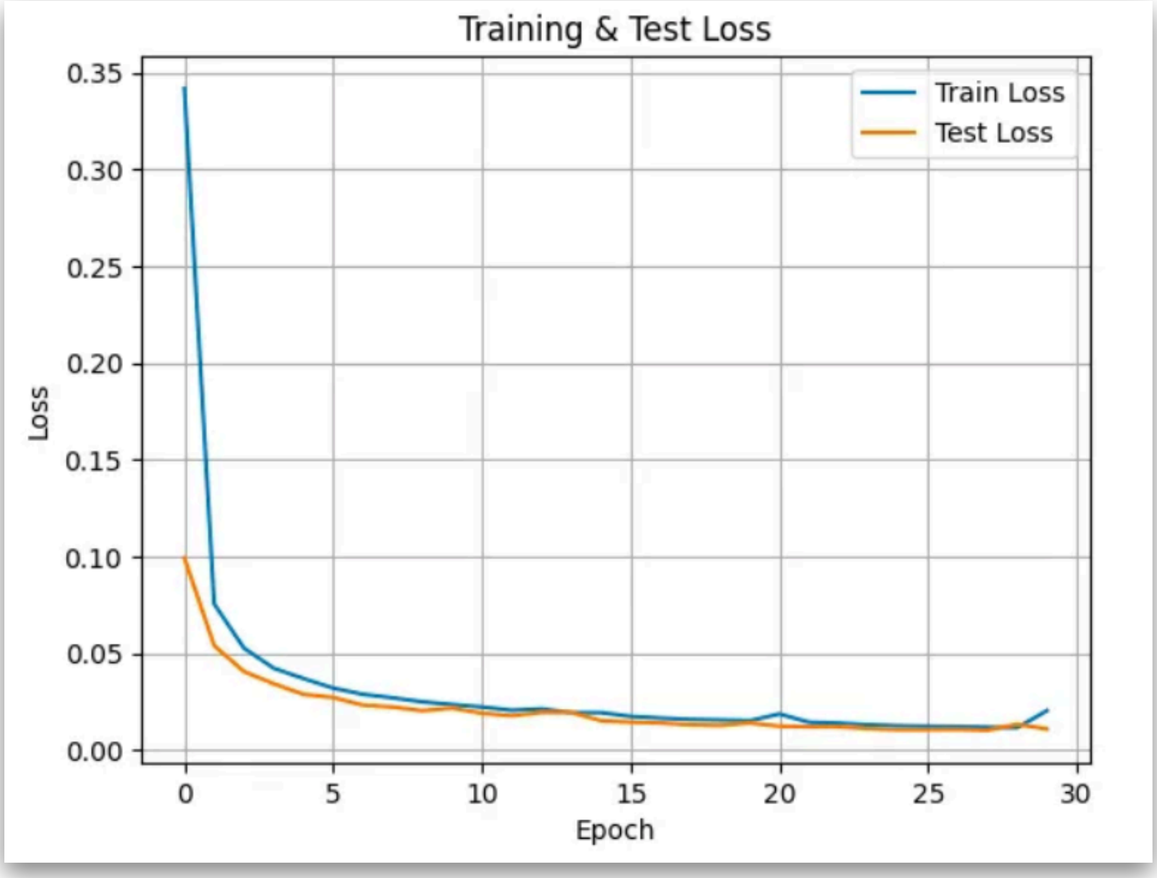


Dropout = 0.1



Encoder & Decoder

| Batch Normalize + Dropout(0.05) - (Decoder X)



Min Train Loss : 0.0116

Min Test Loss : 0.0103

Learning rate = 0.01 & LR scheduler & BN(Encoder)

- Min Train Loss : 0.0033
- Min Test Loss : 0.0097

I. Problem Setting

II. Encoder & Decoder

III. Classification

IV. Conclusion

Classification

| reordered gt label

```python
# ----- Hungarian Matching -----
def get_reordered_labels(pred_masks, gt_masks, labels, device, verbose=False, img=None, epoch=None, img_idx=None, shown_flag=None):
    B = pred_masks.size(0) # batch_size
    reordered_labels = torch.full_like(labels, 10) # reordered_labels: (B, 4)짜리 텐서 생성, 초기값은 모두 10(None)
    for b in range(B):
        cost_matrix = torch.zeros(4, 4, device=device)
        for i in range(4):
            for j in range(4):
                if labels[b, j] == -1:
                    cost_matrix[i, j] = 1.0
                else:
                    iou = compute_iou(pred_masks[b, i], gt_masks[b, j])
                    cost_matrix[i, j] = 1 - iou # cost가 낮을수록 좋은 매칭
        row_ind, col_ind = linear_sum_assignment(cost_matrix.detach().cpu().numpy()) # Hungarian 알고리즘을 구현한 함수
```

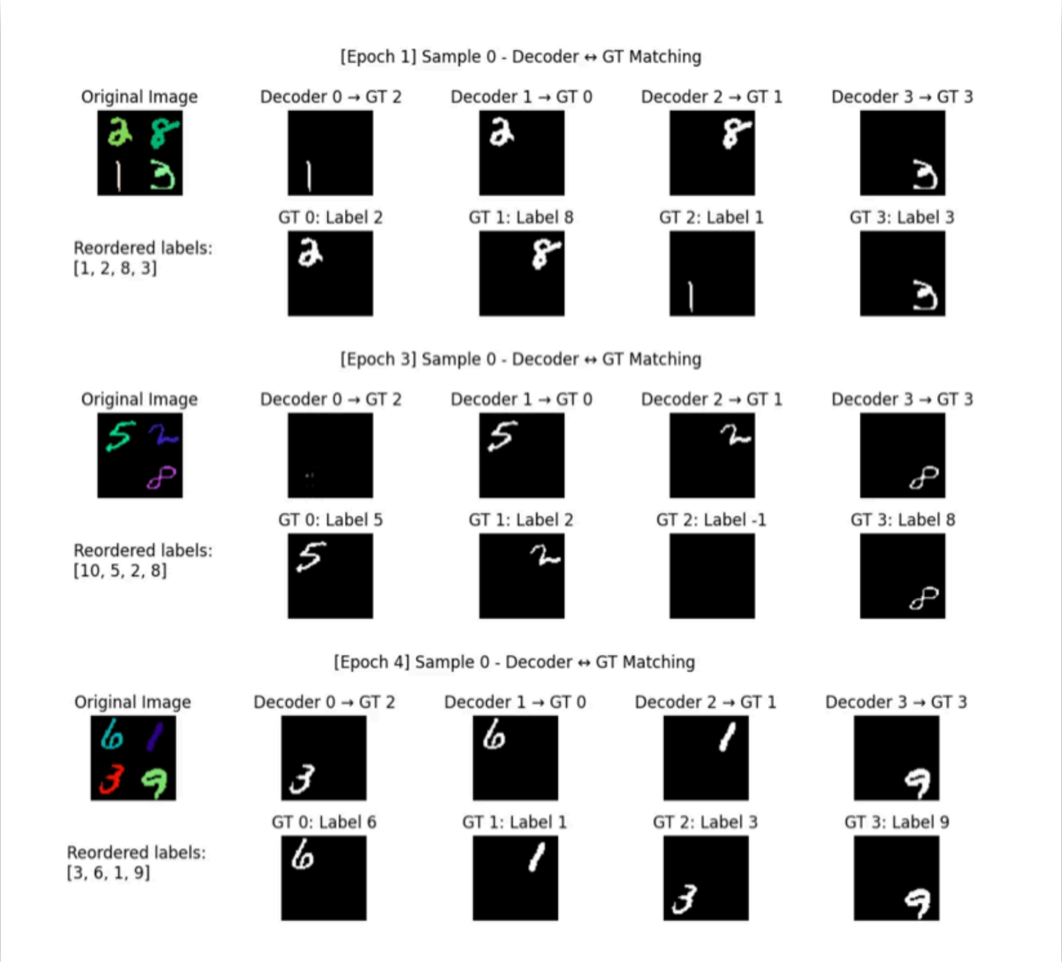
```python
for i, j in zip(row_ind, col_ind):
    if labels[b, j] != -1:
        reordered_labels[b, i] = labels[b, j]
```

- reordered_labels : 10(None)으로 구성되어있는 리스트
- row_ind, col_ind[i] : Decoder row_ind[i] 가 GT col_ind[i] 와 매칭되었다는 의미
  → i번째 디코더가 j번째 GT와 매칭되었으니 GT의 라벨을 reordered_labels[b, i]에 복사

- reordered_labels이 초기 10(None)으로 구성된 배열로, 라벨링된 텐서 값이 -1이면 reordered_labels 값을 대체하지 않으므로 10(None)으로 유지

Classification

| reordered gt label



Classification

| Model with MLP

```python
class Classifier(nn.Module):
    def __init__(self):
        super(Classifier, self).__init__()
        self.heads = nn.ModuleList([
            nn.Sequential(
                nn.Flatten(),               # (64, 64) → 4096
                nn.Linear(4096, 512),
                nn.BatchNorm1d(512),
                nn.ReLU(),

                nn.Linear(512, 256),
                nn.BatchNorm1d(256),
                nn.ReLU(),

                nn.Linear(256, 128),
                nn.BatchNorm1d(128),
                nn.ReLU(),

                nn.Linear(128, 64),
                nn.BatchNorm1d(64),
                nn.ReLU(),

                nn.Linear(64, 32),
                nn.BatchNorm1d(32),
                nn.ReLU(),

                nn.Linear(32, 11)  # 0~9 + N
                )
            for _ in range(4)
        ])

    def forward(self, masks):  # (B, 4, 64, 64)
        outputs = []
        for i in range(4):
            out = self.heads[i](masks[:, i])
            outputs.append(out)
        return torch.stack(outputs, dim=1)  # (B, 4, 11)
```

Min Train Loss : 0.5124

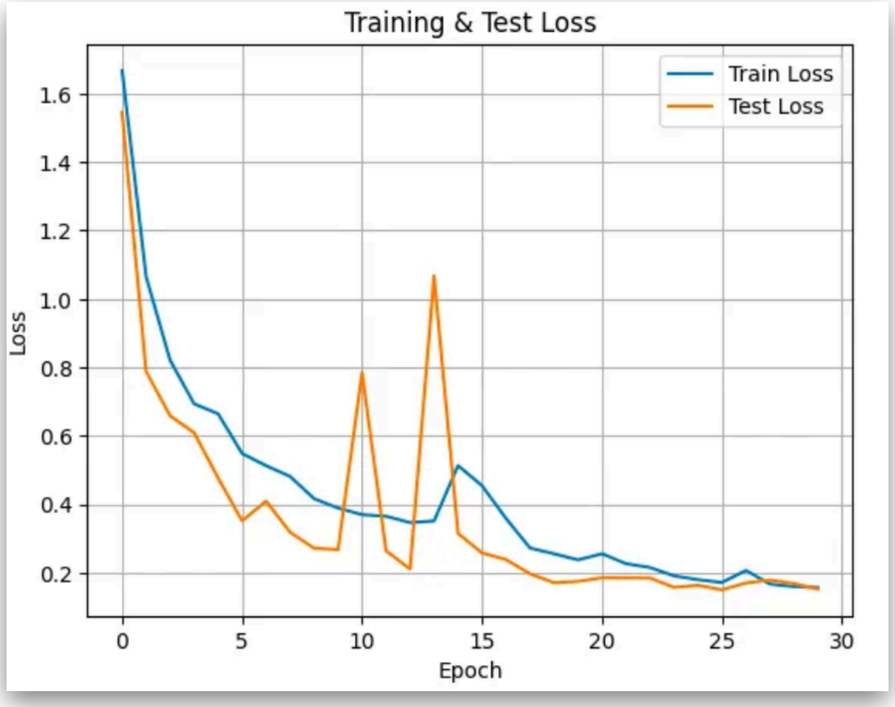
Min Test Loss : 0.6592

Best Test Acc : 83.29

Classification

| Model with CNN & MLP

```python
class Classifier(nn.Module):
    def __init__(self):
        super(Classifier, self).__init__()
        self.heads = nn.ModuleList([
            nn.Sequential(
                # --- Convolutional Feature Extractor ---
                nn.Conv2d(1, 16, kernel_size=3, padding=1),   # (1,64,64) → (16,64,64)
                nn.BatchNorm2d(16),
                nn.ReLU(),
                nn.MaxPool2d(2),                              # (16,32,32)

                nn.Conv2d(16, 32, kernel_size=3, padding=1),
                nn.BatchNorm2d(32),
                nn.ReLU(),
                nn.MaxPool2d(2),                              # (32,16,16)

                nn.Conv2d(32, 64, kernel_size=3, padding=1),
                nn.BatchNorm2d(64),
                nn.ReLU(),
                nn.AdaptiveAvgPool2d((4, 4)),                 # (64,4,4)

                nn.Flatten(),  # → 64×4×4 = 1024

                # --- MLP Classifier ---
                nn.Linear(1024, 256),
                nn.BatchNorm1d(256),
                nn.ReLU(),
                nn.Dropout(0.1),

                nn.Linear(256, 64),
                nn.BatchNorm1d(64),
                nn.ReLU(),
                nn.Dropout(0.1),

                nn.Linear(64, 11)  # 0~9 + N
            )
            for _ in range(4)
        ])

    def forward(self, masks):  # (B, 4, 64, 64)
        outputs = []
        for i in range(4):
            x = masks[:, i].unsqueeze(1)  # (B, 1, 64, 64)
            out = self.heads[i](x)
            outputs.append(out)
        return torch.stack(outputs, dim=1)  # (B, 4, 11)
```



Min Train Loss : 0.1575

Min Test Loss : 0.1509

Best Test Acc : 95.86

Classification

| Model with CNN & MLP - epoch 50

Min Train Loss : 0.0526

Min Test Loss : 0.1148

Best Test Acc : 96.44

Classification

| Model with CNN & MLP - epoch 50 + Dropout

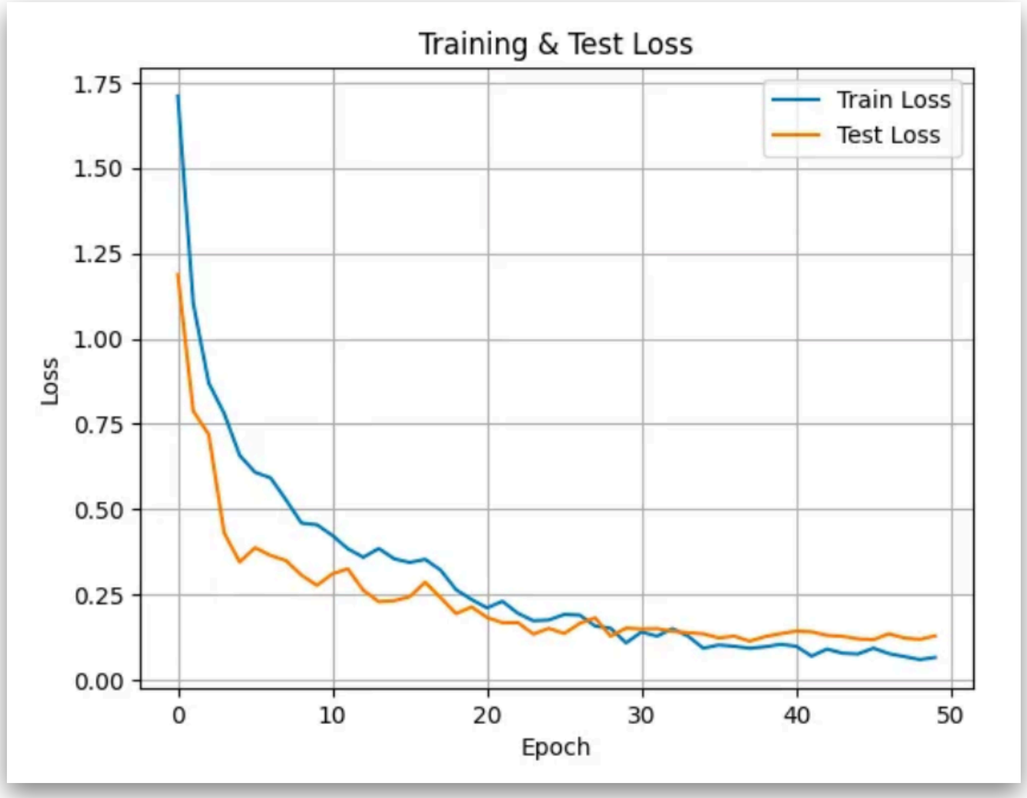
Dropout = 0.1

Min Train Loss : 0.0769

Min Test Loss : 0.1108

Best Test Acc : 96.16

Dropout = 0.05



Min Train Loss : 0.0603

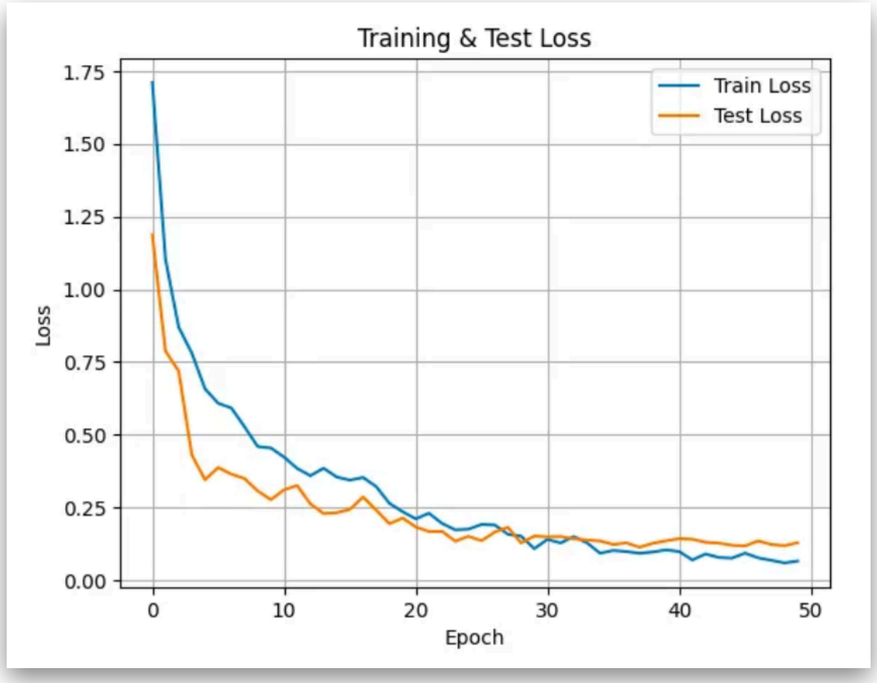
Min Test Loss : 0.1140

Best Test Acc : 96.99

Classification

| Model with CNN & MLP - epoch 50 + Dropout(0.05)

Dropout = 0.05

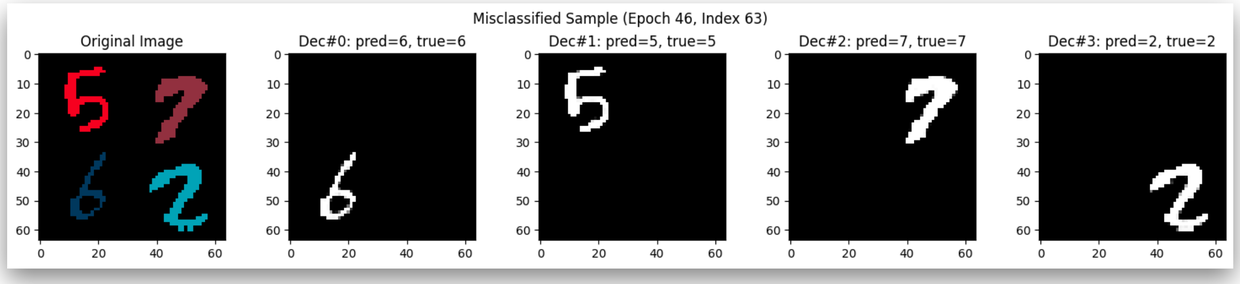


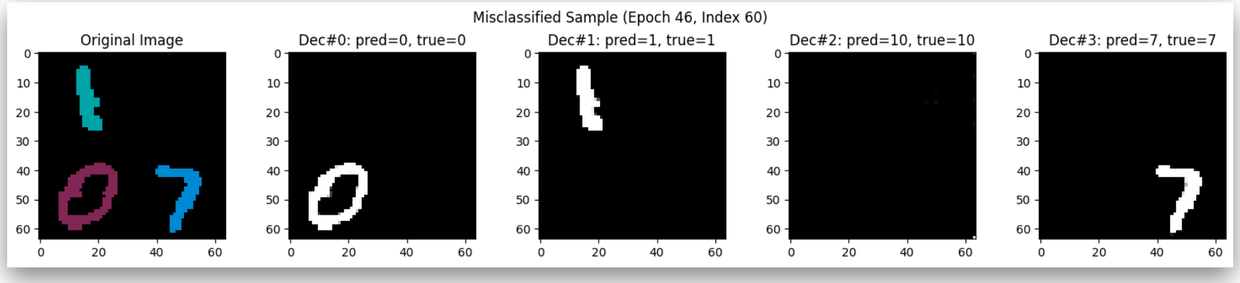
Min Train Loss : 0.0603

Min Test Loss : 0.1140

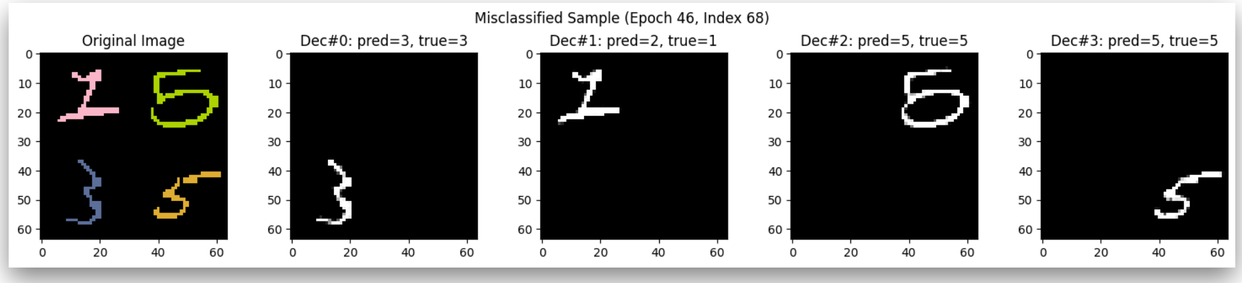
Best Test Acc : 96.99

Right





Wrong



I. Problem Setting

II. Encoder & Decoder

III. Classification

IV. Conclusion

결론

- **Segmentation 측면**

  : batch size 4, learning rate 0.01, ReduceLROnPlateau 스케줄러, BN을 적용한 Encoder,
  그리고 Decoder에는 BN 및 Dropout 제외 설정에서 가장 낮은 테스트 손실 0.0097을 기록

- **Classification 측면**

  : CNN 기반 Encoder와 MLP를 조합한 구조가 단순 MLP보다 우수한 성능을 보였으며,
  최고 테스트 정확도는 96.99%로 측정

Thanks for Listening

2025-1 모빌리티 UR 김유진

2025.05.08