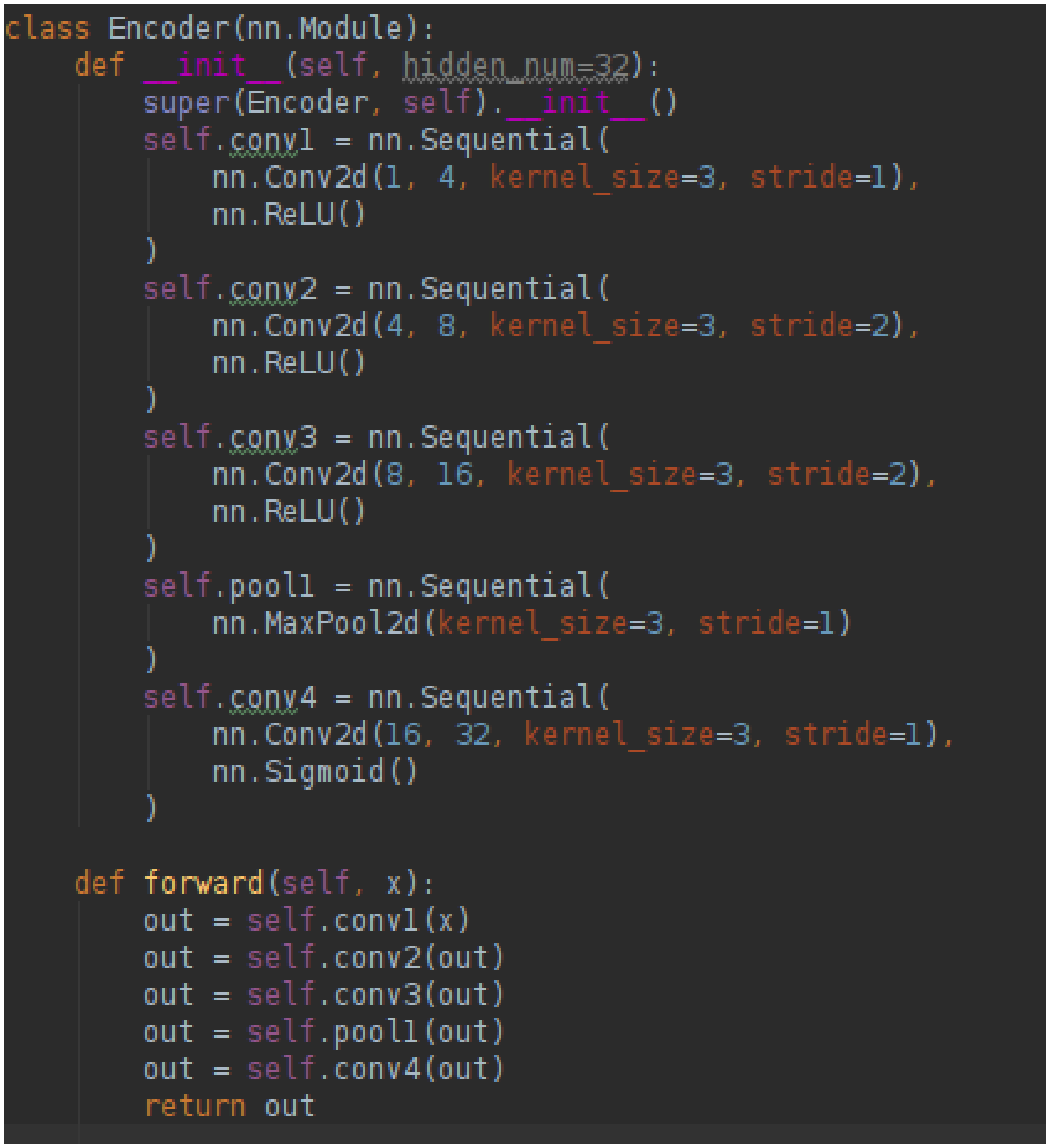
COMP 4211 - Machine Learning Programming Assignment 2 Report

Cheng Chi Fung
cfchengac@connect.ust.hk

1 CNN Classifier

1.1 Screen Shots of the CNN Classifier

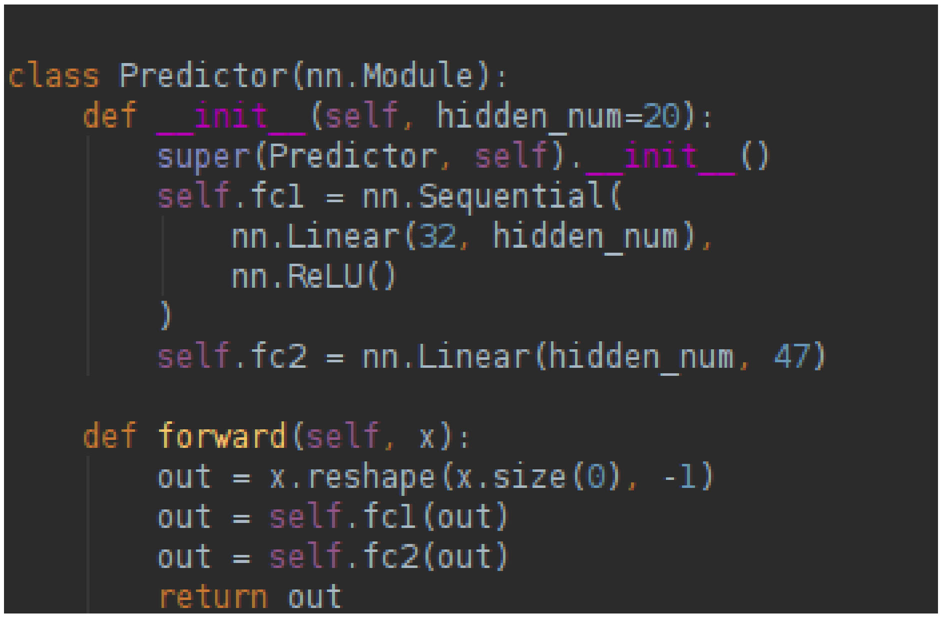
The following are the screen shot of the Predictor Network, CNN Encoder Network and loading the pretrained encoder.



```
class Encoder(nn.Module):
    def __init__(self, hidden_num=32):
        super(Encoder, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(1, 4, kernel_size=3, stride=1),
            nn.ReLU()
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(4, 8, kernel_size=3, stride=2),
            nn.ReLU()
        )
        self.conv3 = nn.Sequential(
            nn.Conv2d(8, 16, kernel_size=3, stride=2),
            nn.ReLU()
        )
        self.pool1 = nn.Sequential(
            nn.MaxPool2d(kernel_size=3, stride=1)
        )
        self.conv4 = nn.Sequential(
            nn.Conv2d(16, 32, kernel_size=3, stride=1),
            nn.Sigmoid()
        )

    def forward(self, x):
        out = self.conv1(x)
        out = self.conv2(out)
        out = self.conv3(out)
        out = self.pool1(out)
        out = self.conv4(out)
        return out
```

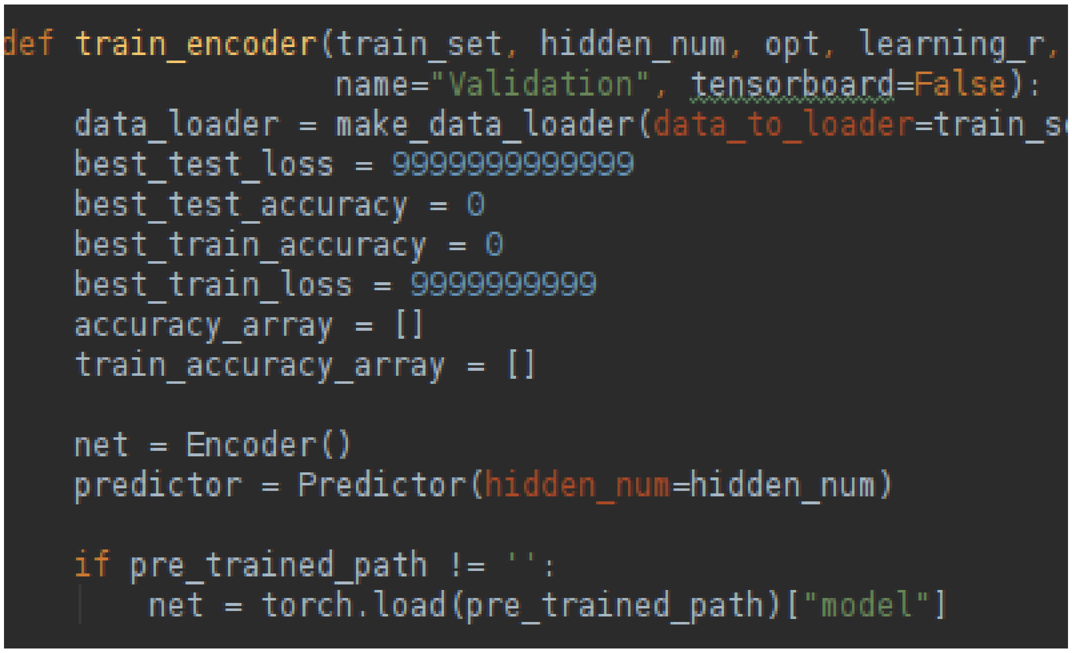
Figure 1: Screen Shots of Encoder Network



```
class Predictor(nn.Module):
    def __init__(self, hidden_num=20):
        super(Predictor, self).__init__()
        self.fc1 = nn.Sequential(
            nn.Linear(32, hidden_num),
            nn.ReLU()
        )
        self.fc2 = nn.Linear(hidden_num, 47)

    def forward(self, x):
        out = x.reshape(x.size(0), -1)
        out = self.fc1(out)
        out = self.fc2(out)
        return out
```

Figure 2: Screen Shots of Predictor Network



```
def train_encoder(train_set, hidden_num, opt, learning_r,
                  name="Validation", tensorboard=False):
    data_loader = make_data_loader(data_to_loader=train_s
    best_test_loss = 9999999999999
    best_test_accuracy = 0
    best_train_accuracy = 0
    best_train_loss = 9999999999
    accuracy_array = []
    train_accuracy_array = []

    net = Encoder()
    predictor = Predictor(hidden_num=hidden_num)

    if pre_trained_path != '':
        net = torch.load(pre_trained_path)["model"]
```

Figure 3: Screen Shots of Loading Pretrained Encoder Weights

1.2 Hold out validation result of CNN classifier from scratch

The following are the hold out validation results of CNN classifier from scratch. The network archiecture of using in the hold out validation was the same as the snapshot above. In the validation, we partitioned the training set into the training and validation sets with ratio 4:1. For each candidate set of hyperparameters, we trained the network for 20 epochs batch size 32 and validate it against the validation set. The cross entropy loss on validation set shown in the results table was the optimal one .

The cross entropy loss of parameters set 4 had the lowest optimal cross entropy loss 410.54719 after conducting the hold out validation, so we would choose the parameters set $[Adam, 0.001, 64]$ as the hyperparameters for the training in the testing phase.

Table 1: Hold out Validation Results of scratch CNN classifer

Parameters set	Optimizer	Learning Rate	Num of Hidden	Cross Entropy Loss
1	Adam	0.001	32	440.98172
2	SGD	0.1	32	2533.93441
3	SGD	0.01	32	2533.97749
4	Adam	0.001	64	410.54719
5	SGD	0.1	64	2533.96268
6	SGD	0.01	64	2534.04213

1.3 Hold out validation result of the CNN classifier with Pretrained Encoder Weights

The following are the hold out validation results of the CNN classifier with pretrained encoder weights. The network archiecture of using in the hold out validation was the same as the snapshot above. In the validation, we partitioned the training set into the training and validation sets with ratio 4:1. For each candidate set of hyperparameters, we trained the network for 20 epochs batch size 32 and validate it against the validation set. The cross entropy loss on validation set shown in the results table was the optimal one .

Table 2: Hold out Validation Results of CNN classifer with Pretrained Encoder Weights

Parameters set	Optimizer	Learning Rate	Num of Hidden	Cross Entropy Loss
1	Adam	0.001	32	735.95330
2	SGD	0.1	32	668.04426
3	SGD	0.01	32	964.40623
4	Adam	0.001	64	673.44847
5	SGD	0.1	64	636.57846
6	SGD	0.01	64	902.99894

The cross entropy loss of parameters set 5 had the lowest optimal cross entropy loss 636.57846 after conducting the hold out validation, so we would choose the parameters set $[SGD, 0.1, 64]$ as the hyperparameters for the training in the testing phase.

1.4 Testing Results of CNN Classifier

In the testing phase, for both CNN classifier from scratch and with Pretrained Encoder Weights, we trained using the entire training set with the best set of hyperparameters obtained from the hold out validation and the same network architecture as the hold out validation, and tested with the testing test. We used 32 as our batch size and trained with 20 epoch. And we had repeated the same process for 5 rounds. The below was the results.

Table 3: Testing Metric of CNN classifier from scratch

	Cross Entropy Loss	Top-1 Accuracy	Top-3 Accuracy
Mean	515.75526	79.27811	78.79027
Std	12.76076	0.34077	0.62663

Table 4: Testing Metric of CNN classifier with Pretrained Encoder Weights

	Cross Entropy Loss	Top-1 Accuracy	Top-3 Accuracy
Mean	768.54128	71.47568	70.88601
Std	14.14077	0.48995	0.37438

The following was the learning curve of the testing metric that we randomly selected only one run out of the total of five runs. The blue color curve was results on testing set and the orange color curve was the results on training set. (The diagram on the left hand side was the results drawn from CNN classifier from scratch and the right hand side was the results drawn from the CNN classifier with Pretrained Encoder Weights)

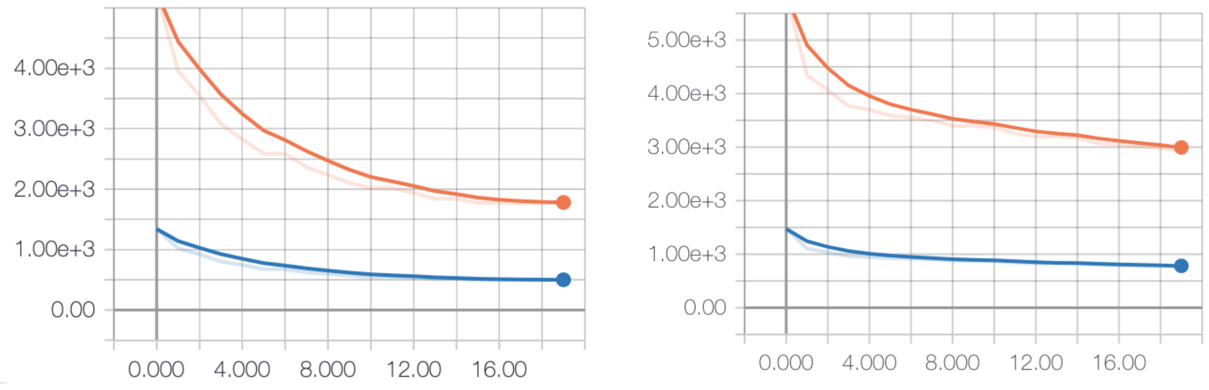


Figure 4: Learning Curve of Cross Entropy Loss

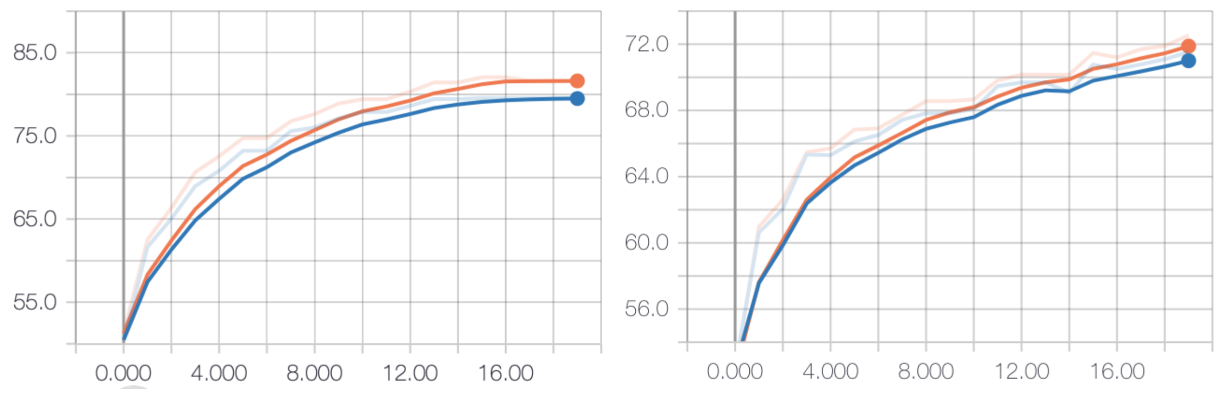


Figure 5: Learning Curve of Top-1 Accuracy

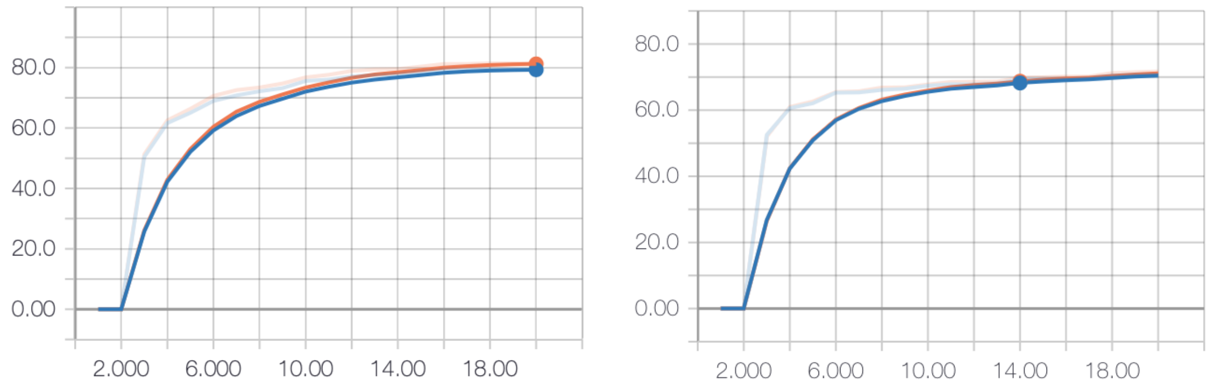


Figure 6: Learning Curve of Top-3 Accuracy

1.5 Analysis on the performance of the CNN classifiers

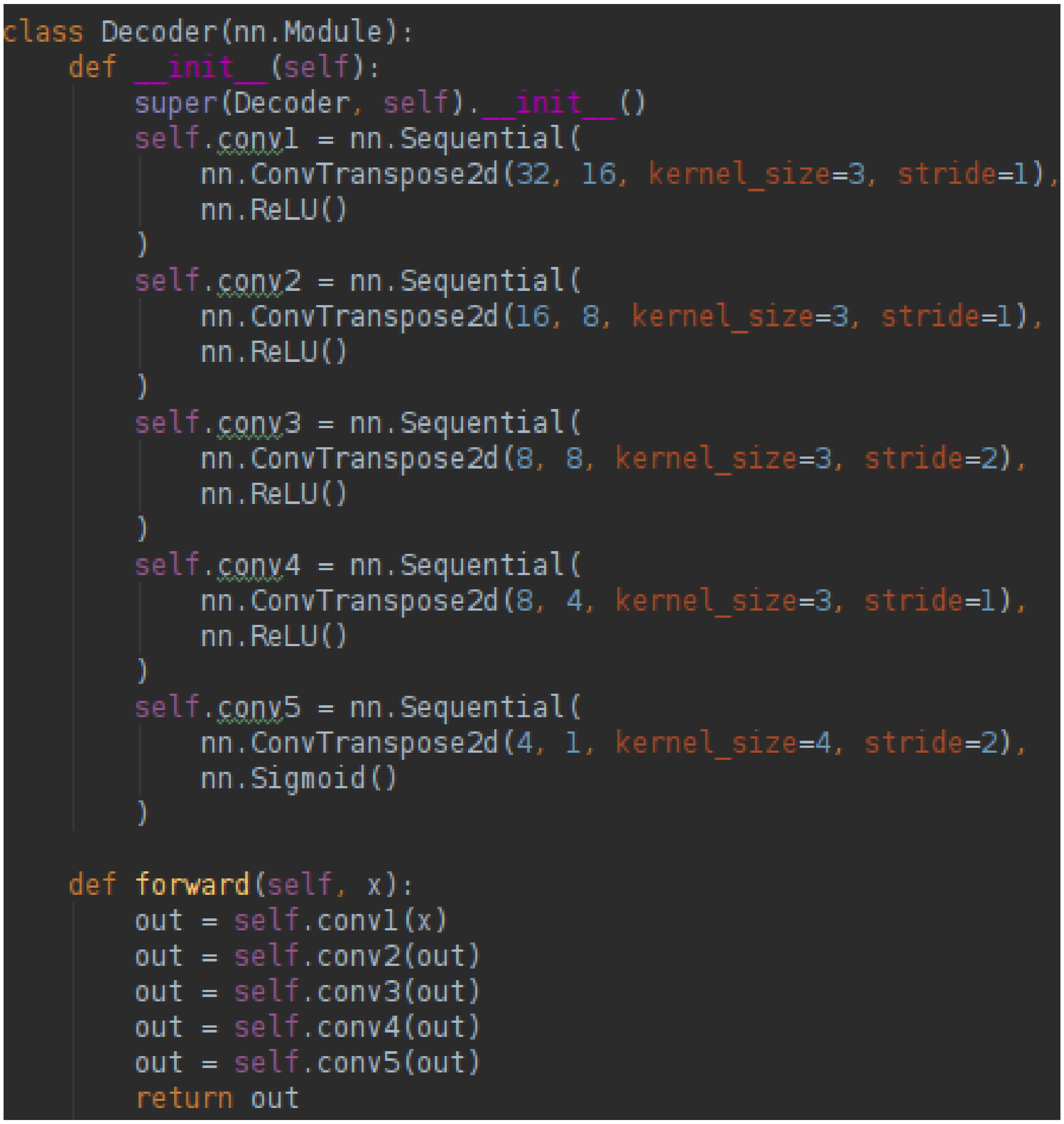
Base on the results of the empirical test results, we found out that the pretrained model does help in decreasing **the training time** and using less epoch. Since, from the results of hold out validation of CNN classifer with Pretrained Weights (Table 2), we could see the cross entropy loss of validation set that using different set of hyperparameters for training are quite near and low $(902.99 - 636.57)$. On the other hand, from the results of of hold out validation of CNN classifer without pretrain-ed Weights (Table 1), we could see the cross entropy loss of using different set of hyper-parameters are quite vary in range $(410.5471 - 2534.04213)$ and only little of them got low cross entropy loss. Therefore, by using pretrained model, it is far more easier to find good hyper-parameters for final training and hence it reduce for hold out validation and hence helps in decrasing the training time. And this may because the pre-training has already gave the network a head start.

However, the testing results of using the CNN classifier with pretrained weight was not as good as the CNN classifier without pretrained weight. We could see the mean of the **cross entropy loss, top-1 accuracy and top-3 accuracy** of the CNN without pretrained weight (Table 3) is better than the CNN with pretrained weight. (Table 4) with the same number of training epoch. Therefore, once if we could find a good hyparameters for training, the rate of convergence of CNN without pretrained weight is higher than with pretrained weight. This may because the pretraining model making the gradient decent stuck into the local miminma which lower the rate of convergence.

2 CAE with Pretrained Encoder

2.1 Screen Shots of the CAE with Pretrained Encoder

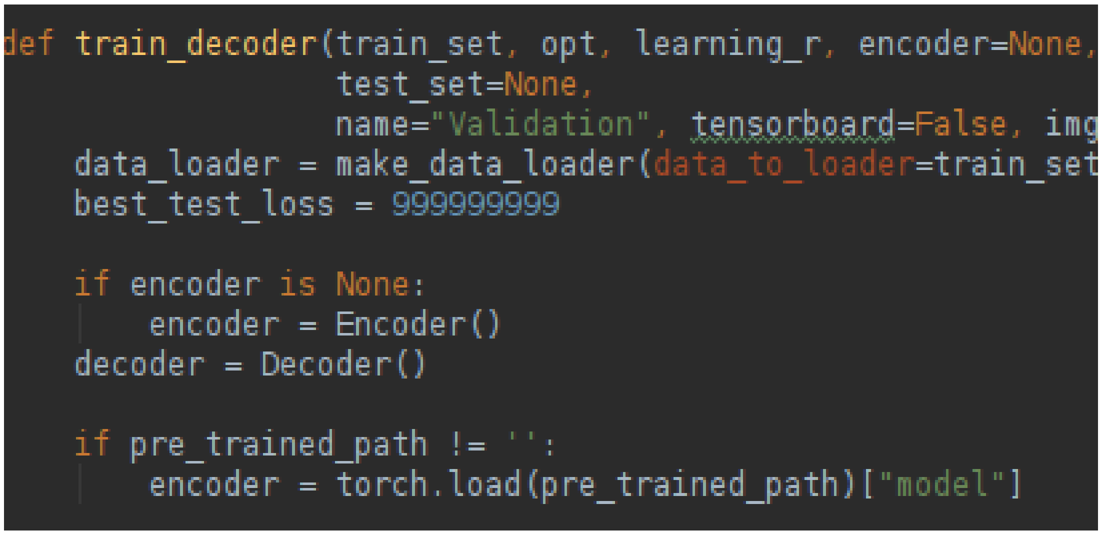
The following are the screen shots of the Network of CAE Decoder and loading the pretrained encoder.



```
class Decoder(nn.Module):
    def __init__(self):
        super(Decoder, self).__init__()
        self.conv1 = nn.Sequential(
            nn.ConvTranspose2d(32, 16, kernel_size=3, stride=1),
            nn.ReLU()
        )
        self.conv2 = nn.Sequential(
            nn.ConvTranspose2d(16, 8, kernel_size=3, stride=1),
            nn.ReLU()
        )
        self.conv3 = nn.Sequential(
            nn.ConvTranspose2d(8, 8, kernel_size=3, stride=2),
            nn.ReLU()
        )
        self.conv4 = nn.Sequential(
            nn.ConvTranspose2d(8, 4, kernel_size=3, stride=1),
            nn.ReLU()
        )
        self.conv5 = nn.Sequential(
            nn.ConvTranspose2d(4, 1, kernel_size=4, stride=2),
            nn.Sigmoid()
        )

    def forward(self, x):
        out = self.conv1(x)
        out = self.conv2(out)
        out = self.conv3(out)
        out = self.conv4(out)
        out = self.conv5(out)
        return out
```

Figure 7: Screen shot of the Network of CAE Decoder



```
def train_decoder(train_set, opt, learning_r, encoder=None,
                  test_set=None,
                  name="Validation", tensorboard=False, img
    data_loader = make_data_loader(data_to_loader=train_set
    best_test_loss = 999999999

    if encoder is None:
        encoder = Encoder()
    decoder = Decoder()

    if pre_trained_path != '':
        encoder = torch.load(pre_trained_path)["model"]
```

Figure 8: Screen shot of Loading the Pretrained Encoder

2.2 Hold out validation result of CAE with Pretrained Encoder

The following are the hold out validation results of the CAE with Pretrained Encoder. The network archiecture of using in the hold out validation was the same as the snapshot above. In the validation, we partitioned the training set into the training and validation sets with ratio 4:1. For each candidate set of hyperparameters, we trained the network for 20 epochs batch size 32 and validate it against the validation set. The MSE loss of the validation set shown in the results table was the optimal one .

Table 5: Hold out Validation Results of the CAE with Pretrained Encoder

Parameters set	Optimizer	Learning Rate	MSE Loss
1	Adam	0.01	0.02729
2	SGD	0.1	0.03881
3	SGD	0.01	0.11098

Some examples of the image recontructed by the decoder network by using different set of parameters during the hold out validation were shown below. (The left is the input image and the right is the recontructed image.)

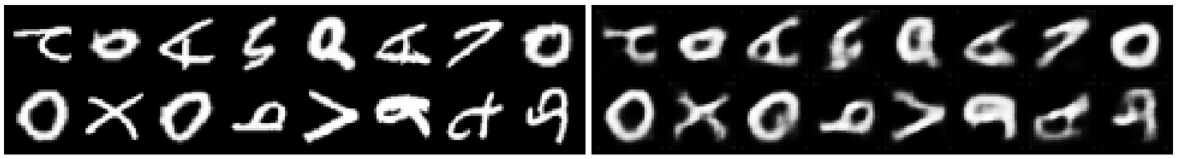


Figure 9: Parameters set 1

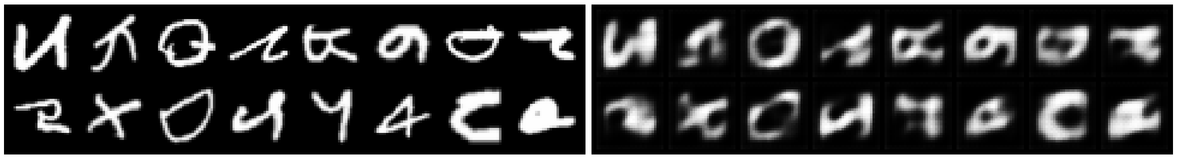


Figure 10: Parameters set 2

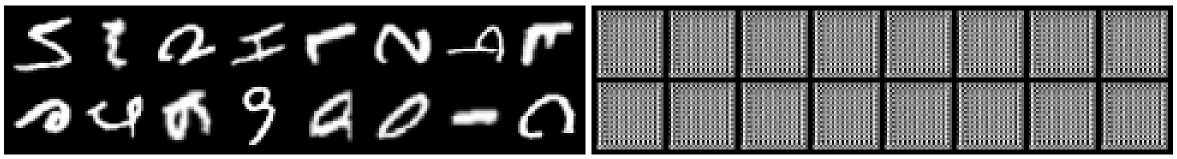


Figure 11: Parameters set 3

From the results of hold out validation, we could see for parameters set 1, it had the lowest MSE loss 0.02729 and the quality of the recontructed image was the best. Therefore, we would choose the parameters set $[Adam, 0.01]$ as the hyperparameters for the training in the testing phase.

2.3 Testing Results of CAE with Pretrained Encoder

In the testing phase, we trained using the entire training set with the best set of hyperparameters obtained from the hold out validation and the same network architecture as the hold out validation, and tested with the testing test. We used 32 as the batch size and trained for 20 epoch. The following was the results obtained from the testing phase. The MSE Loss shown in the table was the optimal MSE loss.

Table 6: Testing Result CAE with Pretrained Encoder

Best MSE Loss
0.02817

Some examples of the image recontructed by the decoder network during the test phase were shown below (In the epoch with minimum MSE loss).

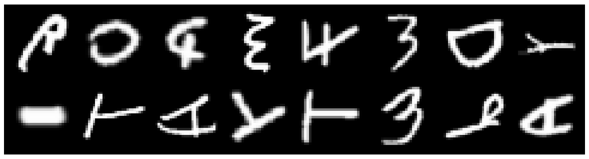


Figure 12: Input Image

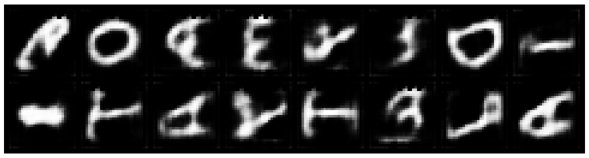


Figure 13: Recontructed Image

The following was the learning curve of the MSE loss. The blue color curve was results on testing set and the orange color curve was the results on training set.

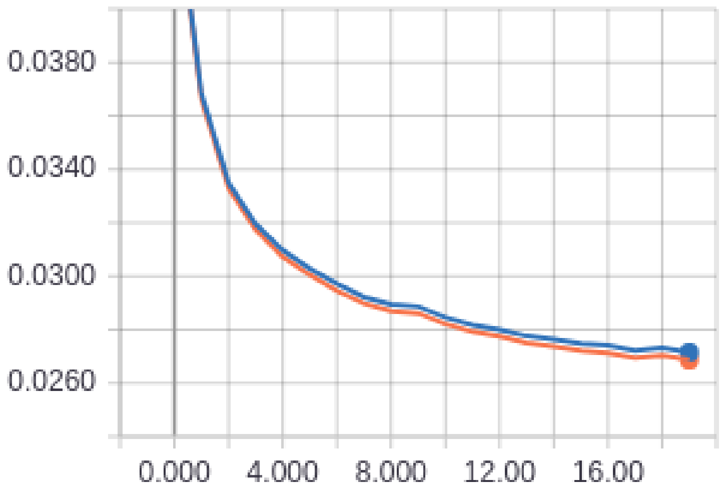


Figure 14: Recontructed Image