2018007956_assignment

[23-1 딥러닝 및 응용 과제 #1]

컴퓨터소프트웨어학부

2018007956 김채아

Task: Image Classification

Model: Encoder + Classifier (Semi-Supervised learning)

Dataset: Oxford flower 102 dataset

Preprocessing: Resize (100, 100), Normalize

Explain the Code (about model)

```
train_dataloader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size, shuffle=True, drop_last=True)
test_dataloader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size, drop_last=True)
```

- dataloader 생성시 마지막 배치 사이즈가 달라져서 연산 사이즈가 달라짐에 따라, 고정된 인풋 사이즈를 받는 classifier에서 오류가 발생할 수 있기 때문에 이를 막기 위해 drop_last=True를 사용한다

```
##### AutoEncoder 모델 코드 #####
class down_block(nn.Module):
    def __init__(self,in_channels, out_channels, kernel_size=(3,3),stride=1,padding=1, bias=False):
        super(down_block, self).__init__()
        self.block1 = nn.Conv2d(in_channels=in_channels, out_channels=out_channels, kernel_size=kernel_size, stride=stride,padding=pad
        self.activation = nn.ReLU()
        self.maxpool = nn.MaxPool2d(2)

    def forward(self,x):
        out = self.block1(x)
        out = self.activation(out)
        out = self.maxpool(out)
        return out

class Encoder(nn.Module):
    def __init__(self,):
        super(Encoder, self).__init__()
        self.encoder1 = down_block(in_channels=3, out_channels=16)
        self.encoder2 = down_block(in_channels=16, out_channels=32)

    def forward(self, input, test=False): # torch.Size([128, 3, 100, 100])
        out = self.encoder1(input) # torch.Size([128, 16, 50, 50])
        out = self.encoder2(out) # torch.Size([128, 32, 25, 25])
        if test==False:
          out = out.view(batch_size, -1) # torch.Size([64, ])
        else:
          x = out.shape
          out = out.view(-1, x[1], x[2], x[3])
        return out

class up_block(nn.Module):
    def __init__(self,in_channels, out_channels, kernel_size=4,stride=1,padding=14, bias=False): # padding=0: valid (no padding), padd
        super(up_block, self).__init__()
        self.block1 = nn.Conv2d(in_channels=in_channels, out_channels=out_channels, kernel_size=kernel_size, stride=stride,padding=pad
        self.activation = nn.ReLU()

    def forward(self,x):
        out = self.activation(self.block1(x))
        return out

class Decoder(nn.Module):
    def __init__(self, ):
        super(Decoder, self).__init__()
        self.decoder1 = up_block(in_channels=32, out_channels=16, kernel_size=4, stride=1, padding=14)
        self.decoder2 = up_block(in_channels=16, out_channels=3, kernel_size=3, stride=1, padding=26)

    def forward(self, x):
        x = x.view(-1, 32, 25, 25)
        out = self.decoder1(x)
        out = self.decoder2(out)
        return out
```

```
class AutoEncoder(nn.Module):
    def __init__(self):
        super(AutoEncoder, self).__init__()
        self.encoder = Encoder()
        self.decoder = Decoder()

    def forward(self, input, test=False):
        z = self.encoder(input, test)
        x_hat = self.decoder(z, test)
        return z, x_hat
```

- BatchNorm2d 사용 안 함
- Encoder의 forward() 부분에 있는 test=True/False 부분은 training 도중 test set 한 장에 대해서 확인을 해보려 하는데 batch size를 고정해버리면 size error 발생 → 테스트를 할 때는 데이터 개수가 batch size가 아니기 때문에 이미지 사이즈를 고정하고 나머지를 -1로 resize함
- Upsample 또는 interpolation 사용 안 함 → 대신 padding으로 이미지 사이즈 키우며 학습을 진행하였는데, 이 부분을 성능 부진의 원인으로 추측
- padding 시 kernel_size, stride, padding size는

  `Output size = (Input-Filter+2*Padding)/Stride+1`

  위 공식을 만족하며 output이 input의 2배가 되도록하는 임의의 숫자를 지정
- Encoder, Decoder 각각 2 layer 씩 쌓음
- Encoder는 linear로 구성된 Classifier와 연결되어 semi-supervised learning을 진행하기 위해, view를 사용햐여 flatten 시킨 형태로 output 생성
- Decoder에서는 view로 flatten 된 벡터를 다시 원래 형태로 되돌려 학습 진행

```
##### Classifier 모델 코드 #####
class Classifier(nn.Module):
    def __init__(self, ):
      super(Classifier, self).__init__()
      self.linear1 = nn.Linear(32*25*25,4096)
      self.linear2 = nn.Linear(4096, 1024)
      self.linear3 = nn.Linear(1024, 256)
      self.dropout = nn.Dropout(0.1)
      self.activation = nn.Sigmoid()
      self.cls = nn.Linear(256,102)

      self.bn1 = nn.BatchNorm1d(4096)
      self.bn2 = nn.BatchNorm1d(1024)
      self.bn3 = nn.BatchNorm1d(256)

    def forward(self, x):
      # print(x.shape)
      z1 = self.linear1(x)
      z1 = self.bn1(z1)
      a1 = self.activation(z1)
      a1 = self.dropout(a1)

      z2 = self.linear2(a1)
      z2 = self.bn2(z2)
      a2 = self.activation(z2)
      a2 = self.dropout(a2)

      z3 = self.linear3(a2)
      z3 = self.bn3(z3)
      a3 = self.activation(z3)
      a3 = self.dropout(a3)

      z4 = self.cls(a3)

      return z4
```

- Encoder의 output으로 나온 이미지 shape에 맞춰서 Linear함수로 받고, class 개수인 102개까지 dimension을 줄여나감
- linear-batch normalization-activation-dropout 순서로 세 층을 쌓음

To improve the performance of the model

1st try

1) Training setting

batch size: 128

[1] Autoencoder

model: 2 layer enc, dec

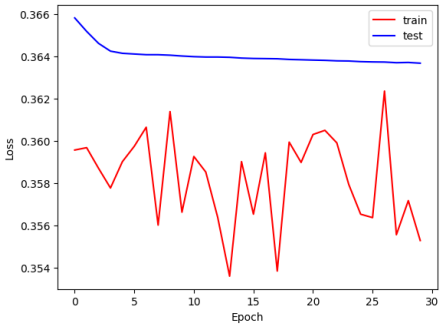
Epoch: 30

activation function: ReLU

loss: MSE

optimizer: Adam

learning rate: 0.0001



[2] Classifier

model: encoder+classifier (3 linear layer, dropout)

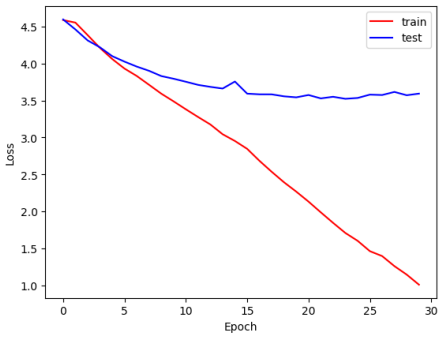
Epoch: 30

activation function: Sigmoid

loss: Cross Entropy

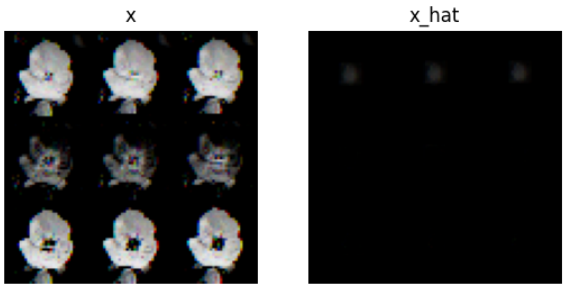
optimizer: Adam

learning rate: encoder [0.001], classifier [0.001]

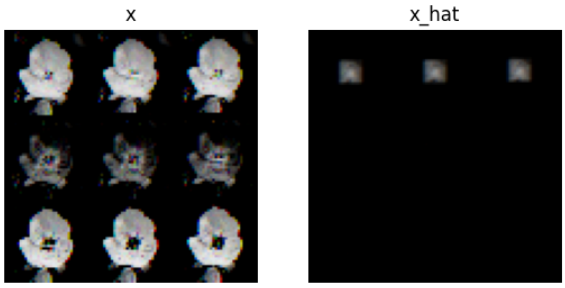


2) Result

Autoencoder trained image:



Before training



After training 25 epochs

Classifier Accuracy: **17.0898%**

2nd try

Classifier의 정확도 성능을 논하기 전에, Autoencoder학습 시 train loss가 불안정하며 training set에 대해 전혀 학습이 전혀 안되는 것을 볼 수 있었음

학습 안됨의 이유 중 하나를 upsampling 시 zero padding으로 이미지를 채운 것으로 추측하는데, 이 부분은 건드릴 수 없으므로 **model capacity를 높이고, learning rate를 줄여봄**

<autoencoder>

model: 3 layer enc, dec

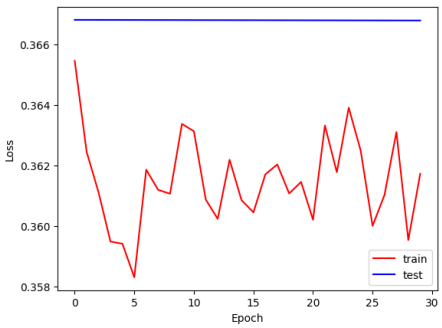
Epoch: 30

activation function: ReLU

loss: MSE

optimizer: Adam

learning rate: 0.000001



3rd try

Autoencoder 학습 시 learning rate 너무 줄여서 그런가 test loss가 너무 미세하게 조금 떨어져서, learning rate를 10배만 키우고, **Early stopping**을 걸어서 epoch을 더욱 길게 학습시켜봄

Classifier의 경우, test set에 대한 학습이 부진함. training set에 over fitting되는 경향을 감소시키기 위해 **L2 regularization** 적용, (4rd try에서 dropout 추가, 5th try에서 model capacity도 줄여봄)

1) Training setting

[1] autoencoder

model: 3 layer enc, dec

Epoch: 100

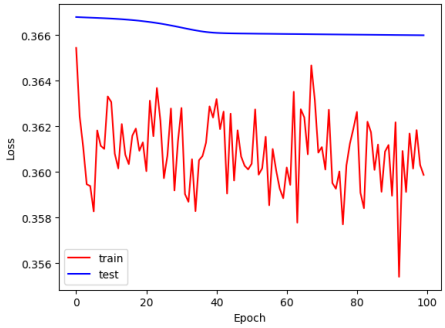
activation function: ReLU

loss: MSE

optimizer: Adam

learning rate: 0.00001

Early stopping (stop count: 10)



(early stopping을 test loss에 대해서 걸었더니, test loss는 꾸준하게 감소하는 추세이기 때문에 Early stop에 걸리지 않고 설정해 준 100 epoch 까지 학습함)

[2] Classifier

model: encoder+classifier (3 linear layer, dropout)

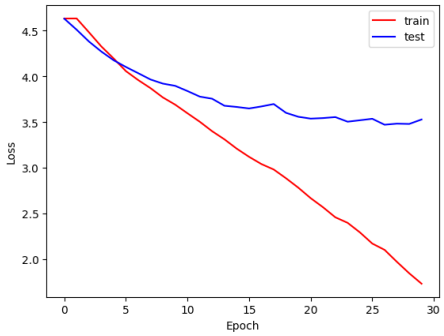
Epoch: 30

activation function: Sigmoid

loss: Cross Entropy + **L2 regularization**

optimizer: Adam

learning rate: encoder [0.001], classifier [0.001]



2) Result

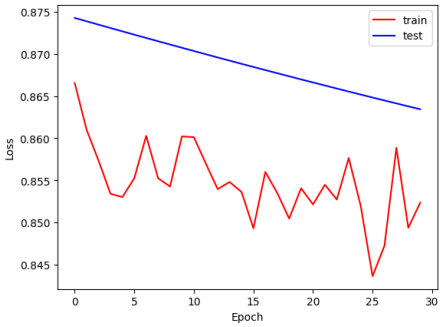
Classifier Accuracy: **15.9505%**

4rd try

model 구조를 조금 다르게 가져가보고자

[1] autoencoder

activation ReLU → sigmoid로 학습시 처참한 학습



activation ReLU로 다시 되돌리고 모델단에서 수정 진행

model: 3 layer enc, dec, **dropout layer (0.3) 추가**

Epoch: 30

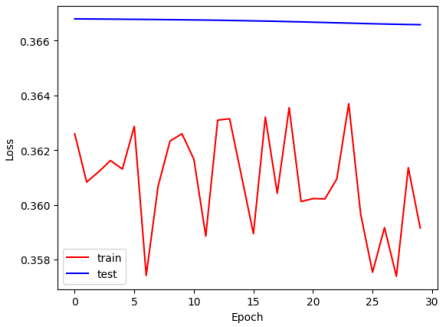
activation function: **LeakyReLU (negative_slope=0.01)**

loss: MSE

optimizer: Adam

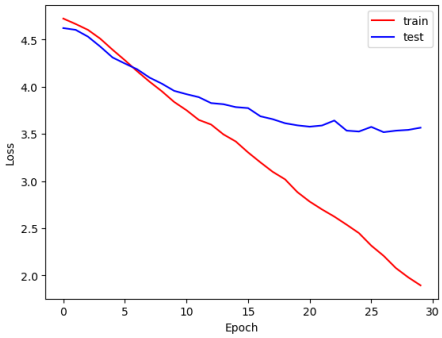
learning rate: 0.00001

Early stopping (stop count: 10)



[2] classifier

dropout rate 증가 (0.1→0.5)



2) Result

Classifier Accuracy: **14.4694%**

5th try

<classifier> **model capacity 낮춰보기 3 layer → 2 layer**

model: encoder+classifier (**2 linear layer**, dropout)

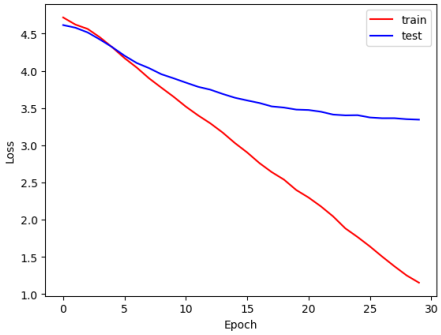
Epoch: 30

activation function: Sigmoid

loss: Cross Entropy + L2 regularization

optimizer: Adam

learning rate: encoder [0.001], classifier [0.001]



2) Result

Classifier Accuracy: **19.4499%**