Autoencoder 모델 (PCA 기반) 하이퍼파라미터 별 성능 기록

<1>

```
## Encoder
        self.encoder = tf.keras.Sequential([
            layers.Dense(17, activation="tanh"),
            layers.BatchNormalization(),
            layers.Dropout(0.3),
            layers.Dense(8, activation="relu"),
            layers.BatchNormalization(),
            layers.Dense(3, activation="relu"),
            layers.BatchNormalization()
        ])

        ## Decoder
        self.decoder = tf.keras.Sequential([
            layers.Dense(8, activation="relu"),
            layers.BatchNormalization(),
            layers.Dropout(0.3),
            layers.Dense(17, activation="tanh"),  #출력층
            layers.BatchNormalization()
        ])

optimizer = tf.keras.optimizers.Adam(learning_rate=1e-4, weight_decay=1e-5)
autoencoder.compile(optimizer=optimizer, loss='mae')

history = autoencoder.fit(normal_X_train, normal_X_train,
          epochs=50,
          batch_size=128,
          validation_data=(X_valid, X_valid),
          shuffle=True)
```

epoch 50 → val_loss 0.2189

Accuracy = 0.7889342554787101
Precision = 0.08014861995753715
Recall = 0.1587802313354364

<2>

```
self.encoder = tf.keras.Sequential([
            layers.Dense(64, activation="tanh"),
            layers.BatchNormalization(),
            layers.Dropout(0.4),
            layers.Dense(32, activation="relu"),
            layers.BatchNormalization(),
            layers.Dropout(0.4),
            layers.Dense(16, activation="relu"),
            layers.BatchNormalization(),
            layers.Dense(8, activation="elu"),
            layers.BatchNormalization(),
            layers.Dense(2, activation="relu"),
            layers.BatchNormalization()
        ])
```

```
        ## Decoder
        self.decoder = tf.keras.Sequential([
            layers.Dense(8, activation="elu"),
            layers.BatchNormalization(),
            layers.Dense(16, activation="relu"),
            layers.BatchNormalization(),
            layers.Dropout(0.4),
            layers.Dense(32, activation="relu"),
            layers.BatchNormalization(),
            layers.Dropout(0.4),
            layers.Dense(64, activation="tanh"),
            layers.BatchNormalization(),
            layers.Dense(17, activation="sigmoid"),  #출력층
            layers.BatchNormalization()
        ])

optimizer = tf.keras.optimizers.Adam(learning_rate=1e-4, weight_decay=1e-5)
autoencoder.compile(optimizer=optimizer, loss='mae')

history = autoencoder.fit(normal_X_train, normal_X_train,
          epochs=50,
          batch_size=128,
          validation_data=(X_valid, X_valid),
          shuffle=True)
```

epoch 49 → val_loss 0.2237

Accuracy = 0.7805182901424881
Precision = 0.08485446472619634
Recall = 0.1808622502628812

<3>

```
            ## Encoder
        self.encoder = tf.keras.Sequential([
            layers.Dense(64, activation="tanh"),
            layers.BatchNormalization(),
            #layers.Dropout(0.4),
            layers.Dense(32, activation="relu"),
            layers.BatchNormalization(),
            #layers.Dropout(0.4),
            layers.Dense(16, activation="relu"),
            layers.BatchNormalization(),
            layers.Dense(8, activation="elu"),
            layers.BatchNormalization(),
            layers.Dense(2, activation="relu"),
            layers.BatchNormalization()
        ])

        ## Decoder
        self.decoder = tf.keras.Sequential([
            layers.Dense(8, activation="elu"),
            layers.BatchNormalization(),
            layers.Dense(16, activation="relu"),
            layers.BatchNormalization(),
            #layers.Dropout(0.4),
            layers.Dense(32, activation="relu"),
            layers.BatchNormalization(),
```

```
            #layers.Dropout(0.4),
            layers.Dense(64, activation="tanh"),
            layers.BatchNormalization(),
            layers.Dense(17, activation="sigmoid"),  #출력층
            layers.BatchNormalization()
        ])

    ##forward pass 정의
    def call(self, x):
        #입력 x를 인코더에 전달하여 인코딩된 특성을 추출
        encoded = self.encoder(x)
        #인코딩된 특성을 디코더에 전달하여 디코딩된 출력을 생성
        decoded = self.decoder(encoded)
        return decoded


model_name = "anomaly.h5"
## ModelCheckpoint: 'val_loss'를 모니터링해, 이 값이 개선될 때마다 모델 가중치 저장
checkpoint = ModelCheckpoint(model_name,
                             monitor="val_loss",  #모니터링할 지표
                             mode="min",  #모델을 저장할 때 val_loss를 최소화하도록 설정
                             save_best_only = True,  #val_loss가 감소할 때만 모델을 저장
                             save_weights_only=True,  #모델 아키텍처와 관련된 정보를 저장하지 않고 모델 가
                             verbose=0)  #모델 저장 과정을 화면에 출력


## Early Stopping: 검증 손실값이 5에폭 동안 향상이 없으면 조기 종료
earlystopping = EarlyStopping(monitor='val_loss',  #모니터링할 지표
                              min_delta = 0,  #개선으로 간주하기 위한 최소 변화량
                              patience = 5,  #val_loss가 개선되지 않더라도 몇 번의 epoch를 기다릴지
                              verbose = 1,  #조기 종료 과정을 화면에 출력
                              restore_best_weights=True)  #조기 종료 후 최상의 성능을 보인 모델 가중치로


## 체크포인트와 조기 종료 콜백을 저장한다
callbacks = [checkpoint, earlystopping]

autoencoder = AnomalyDetector()

optimizer = tf.keras.optimizers.Adam(learning_rate=1e-4, weight_decay=1e-5)
autoencoder.compile(optimizer=optimizer, loss='mae')

history = autoencoder.fit(normal_X_train, normal_X_train,
          epochs=100,
          batch_size=128,
          validation_data=(X_valid, X_valid),
          callbacks = callbacks,
          shuffle=True)
```

epoch 97 → 0.1726

- Dropout 사용 안하는 게 오히려 성능이 더 높게 나온다.
- patience=8, batch_size=256, epoch=150

<4>

- Epoch 149/150

loss: 0.2070 - val_loss: 0.1884

Accuracy = 0.8000166652778935
Precision = 0.08145580589254767
Recall = 0.14826498422712933

<5>

- patience=8, batch_size=64

Epoch 74/150
loss: 0.2126 - val_loss: 0.1819

Accuracy = 0.7771852345637863
Precision = 0.08760172331258975
Recall = 0.19242902208201892

<6>

- patience=8, batch_size=32

Epoch 60/150
loss: 0.2332 - val_loss: 0.1927

<7>

```
        ## Encoder
    self.encoder = tf.keras.Sequential([
        layers.Dense(64, activation="tanh"),
        layers.BatchNormalization(),
        #layers.Dropout(0.4),
        layers.Dense(32, activation="leaky_relu"),
        layers.BatchNormalization(),
        #layers.Dropout(0.4),
        layers.Dense(16, activation="leaky_relu"),
        layers.BatchNormalization(),
        layers.Dense(8, activation="leaky_relu"),
        layers.BatchNormalization(),
        layers.Dense(2, activation="relu"),
        layers.BatchNormalization()
    ])

    ## Decoder
    self.decoder = tf.keras.Sequential([
        layers.Dense(8, activation="leaky_relu"),
        layers.BatchNormalization(),
        layers.Dense(16, activation="leaky_relu"),
        layers.BatchNormalization(),
        #layers.Dropout(0.4),
        layers.Dense(32, activation="leaky_relu"),
        layers.BatchNormalization(),
        #layers.Dropout(0.4),
        layers.Dense(64, activation="tanh"),
        layers.BatchNormalization(),
        layers.Dense(17, activation="sigmoid"),  #출력층
        layers.BatchNormalization()
    ])

##forward pass 정의
def call(self, x):
    #입력 x를 인코더에 전달하여 인코딩된 특성을 추출
    encoded = self.encoder(x)
    #인코딩된 특성을 디코더에 전달하여 디코딩된 출력을 생성
```

```
        decoded = self.decoder(encoded)
        return decoded


model_name = "anomaly.h5"
## ModelCheckpoint: 'val_loss'를 모니터링해, 이 값이 개선될 때마다 모델 가중치 저장
checkpoint = ModelCheckpoint(model_name,
                             monitor="val_loss",  #모니터링할 지표
                             mode="min",  #모델을 저장할 때 val_loss를 최소화하도록 설정
                             save_best_only = True,  #val_loss가 감소할 때만 모델을 저장
                             save_weights_only=True,  #모델 아키텍처와 관련된 정보를 저장하지 않고 모델 가
                             verbose=0)  #모델 저장 과정을 화면에 출력


## Early Stopping: 검증 손실값이 5에폭 동안 향상이 없으면 조기 종료
earlystopping = EarlyStopping(monitor='val_loss',  #모니터링할 지표
                              min_delta = 0,  #개선으로 간주하기 위한 최소 변화량
                              patience = 5,  #val_loss가 개선되지 않더라도 몇 번의 epoch를 기다릴지
                              verbose = 1,  #조기 종료 과정을 화면에 출력
                              restore_best_weights=True)  #조기 종료 후 최상의 성능을 보인 모델 가중치로


## 체크포인트와 조기 종료 콜백을 저장한다
callbacks = [checkpoint, earlystopping]

autoencoder = AnomalyDetector()


optimizer = tf.keras.optimizers.Adam(learning_rate=1e-4, weight_decay=1e-5)
autoencoder.compile(optimizer=optimizer, loss='mae')


history = autoencoder.fit(normal_X_train, normal_X_train,
          epochs=100,
          batch_size=128,
          validation_data=(X_valid, X_valid),
          callbacks = callbacks,
          shuffle=True)
```

- relu 대신 leaky_relu를 사용하면 성능이 낮아진다.

epoch 63 → val_loss: 0.1936

Accuracy = 0.7962669777518541
Precision = 0.08174692049272117
Recall = 0.15352260778128285

<8>

```
        ## Encoder
        self.encoder = tf.keras.Sequential([
            layers.Dense(64, activation="tanh"),
            layers.BatchNormalization(),
            #layers.Dropout(0.4),
            layers.Dense(32, activation="relu"),
            layers.BatchNormalization(),
            #layers.Dropout(0.4),
            layers.Dense(16, activation="leaky_relu"),
            layers.BatchNormalization(),
```

```
            layers.Dense(8, activation="relu"),
            layers.BatchNormalization(),
            layers.Dense(2, activation="relu"),
            layers.BatchNormalization()
        ])

        ## Decoder
        self.decoder = tf.keras.Sequential([
            layers.Dense(8, activation="relu"),
            layers.BatchNormalization(),
            layers.Dense(16, activation="leaky_relu"),
            layers.BatchNormalization(),
            #layers.Dropout(0.4),
            layers.Dense(32, activation="relu"),
            layers.BatchNormalization(),
            #layers.Dropout(0.4),
            layers.Dense(64, activation="tanh"),
            layers.BatchNormalization(),
            layers.Dense(17, activation="sigmoid"),  #출력층
            layers.BatchNormalization()
        ])

    ##forward pass 정의
    def call(self, x):
        #입력 x를 인코더에 전달하여 인코딩된 특성을 추출
        encoded = self.encoder(x)
        #인코딩된 특성을 디코더에 전달하여 디코딩된 출력을 생성
        decoded = self.decoder(encoded)
        return decoded


model_name = "anomaly.h5"
## ModelCheckpoint: 'val_loss'를 모니터링해, 이 값이 개선될 때마다 모델 가중치 저장
checkpoint = ModelCheckpoint(model_name,
                             monitor="val_loss",  #모니터링할 지표
                             mode="min",  #모델을 저장할 때 val_loss를 최소화하도록 설정
                             save_best_only = True,  #val_loss가 감소할 때만 모델을 저장
                             save_weights_only=True,  #모델 아키텍처와 관련된 정보를 저장하지 않고 모델 가
                             verbose=0)  #모델 저장 과정을 화면에 출력

## Early Stopping: 검증 손실값이 5에폭 동안 향상이 없으면 조기 종료
earlystopping = EarlyStopping(monitor='val_loss',  #모니터링할 지표
                              min_delta = 0,  #개선으로 간주하기 위한 최소 변화량
                              patience = 5,  #val_loss가 개선되지 않더라도 몇 번의 epoch를 기다릴지
                              verbose = 1,  #조기 종료 과정을 화면에 출력
                              restore_best_weights=True)  #조기 종료 후 최상의 성능을 보인 모델 가중치로


## 체크포인트와 조기 종료 콜백을 저장한다
callbacks = [checkpoint, earlystopping]

autoencoder = AnomalyDetector()


optimizer = tf.keras.optimizers.Adam(learning_rate=1e-4, weight_decay=1e-5)
autoencoder.compile(optimizer=optimizer, loss='mae')
```

```
history = autoencoder.fit(normal_X_train, normal_X_train,
          epochs=100,
          batch_size=128,
          validation_data=(X_valid, X_valid),
          callbacks = callbacks,
          shuffle=True)
```

Epoch 91/100
loss: 0.2079 - val_loss: 0.1824

Accuracy = 0.7949337555203733
Precision = 0.08333333333333333
Recall = 0.1587802313354364

<9>

```
        ## Encoder
         self.encoder = tf.keras.Sequential([
            layers.Dense(64, activation="tanh"),
            layers.BatchNormalization(),
            #layers.Dropout(0.4),
            layers.Dense(32, activation="relu"),
            layers.BatchNormalization(),
            #layers.Dropout(0.4),
            layers.Dense(16, activation="relu"),
            layers.BatchNormalization(),
            layers.Dense(8, activation="relu"),
            layers.BatchNormalization(),
            layers.Dense(2, activation="relu"),
            layers.BatchNormalization()
        ])

        ## Decoder
        self.decoder = tf.keras.Sequential([
            layers.Dense(8, activation="relu"),
            layers.BatchNormalization(),
            layers.Dense(16, activation="relu"),
            layers.BatchNormalization(),
            #layers.Dropout(0.4),
            layers.Dense(32, activation="relu"),
            layers.BatchNormalization(),
            #layers.Dropout(0.4),
            layers.Dense(64, activation="tanh"),
            layers.BatchNormalization(),
            layers.Dense(17, activation="sigmoid"),  #출력층
            layers.BatchNormalization()
        ])

    ##forward pass 정의
    def call(self, x):
        #입력 x를 인코더에 전달하여 인코딩된 특성을 추출
        encoded = self.encoder(x)
        #인코딩된 특성을 디코더에 전달하여 디코딩된 출력을 생성
        decoded = self.decoder(encoded)
        return decoded


model_name = "anomaly.h5"
```

```
## ModelCheckpoint: 'val_loss'를 모니터링해, 이 값이 개선될 때마다 모델 가중치 저장
checkpoint = ModelCheckpoint(model_name,
                             monitor="val_loss",  #모니터링할 지표
                             mode="min",  #모델을 저장할 때 val_loss를 최소화하도록 설정
                             save_best_only = True,  #val_loss가 감소할 때만 모델을 저장
                             save_weights_only=True,  #모델 아키텍처와 관련된 정보를 저장하지 않고 모델 가
                             verbose=0)  #모델 저장 과정을 화면에 출력


## Early Stopping: 검증 손실값이 5에폭 동안 향상이 없으면 조기 종료
earlystopping = EarlyStopping(monitor='val_loss',  #모니터링할 지표
                              min_delta = 0,  #개선으로 간주하기 위한 최소 변화량
                              patience = 5,  #val_loss가 개선되지 않더라도 몇 번의 epoch를 기다릴지
                              verbose = 1,  #조기 종료 과정을 화면에 출력
                              restore_best_weights=True)  #조기 종료 후 최상의 성능을 보인 모델 가중치로


## 체크포인트와 조기 종료 콜백을 저장한다
callbacks = [checkpoint, earlystopping]

autoencoder = AnomalyDetector()


optimizer = tf.keras.optimizers.Adam(learning_rate=1e-4, weight_decay=1e-5)
autoencoder.compile(optimizer=optimizer, loss='mae')


history = autoencoder.fit(normal_X_train, normal_X_train,
          epochs=100,
          batch_size=128,
          validation_data=(X_valid, X_valid),
          callbacks = callbacks,
          shuffle=True)
```

Epoch 98/100
loss: 0.2055 - val_loss: 0.1822

Accuracy = 0.7883509707524373
Precision = 0.08073878627968338
Recall = 0.1608832807570978