```
# read dataset
set.seed(10)
library(readxl)
creditcard <- read_excel("C:/Users/taeho/Documents/creditcard.xlsx")
# View(creditcard)
```

```
# number of predictors and observations
k=length(creditcard[1,])-2
n=nrow(creditcard)



crx = sapply(creditcard[,c(-1,-31)], function(x) (x - min(x, na.rm = T)) / (max(x, na.rm = T)
- min(x, na.rm=T)))
df = cbind(creditcard[,31], crx)
attach(df)
```

```
attach(df)
```

```
## The following objects are masked from df (pos = 3):
##
##     Amount, Class, V1, V10, V11, V12, V13, V14, V15, V16, V17, V18,
##     V19, V2, V20, V21, V22, V23, V24, V25, V26, V27, V28, V3, V4, V5,
##     V6, V7, V8, V9
```

```
glmAll =glm(Class~V1+V2+V3+V4+V5+V6+V7+V8+V9+V10+V11+V12+V13+V14+V15+
              V16+V17+V18+V19+V20+V21+V22+V23+V24+V25+V26+V27+V28+Amount,data=df,family="binomi
al")
summary(glmAll)
```

```
##
## Call:
## glm(formula = Class ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 +
##     V9 + V10 + V11 + V12 + V13 + V14 + V15 + V16 + V17 + V18 +
##     V19 + V20 + V21 + V22 + V23 + V24 + V25 + V26 + V27 + V28 +
##     Amount, family = "binomial", data = df)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -4.8731  -0.0292  -0.0194  -0.0125   4.6044
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  58.15527   15.37213   3.783 0.000155 ***
## V1            4.92458    2.44280   2.016 0.043804 *
## V2            1.23651    5.48206   0.226 0.821547
## V3            2.21832    2.62571   0.845 0.398196
## V4           15.92408    1.66253   9.578  < 2e-16 ***
## V5           15.14504    9.71207   1.559 0.118901
## V6          -12.21959    7.53733  -1.621 0.104973
## V7          -18.16698   10.85145  -1.674 0.094101 .
## V8          -15.68760    2.84679  -5.511 3.58e-08 ***
## V9           -7.57407    3.18935  -2.375 0.017558 *
## V10         -39.57652    4.68979  -8.439  < 2e-16 ***
## V11          -0.20716    1.27722  -0.162 0.871151
## V12           1.83959    2.28454   0.805 0.420683
## V13          -4.13369    1.05005  -3.937 8.26e-05 ***
## V14         -16.21095    1.83317  -8.843  < 2e-16 ***
## V15          -1.14106    1.12325  -1.016 0.309696
## V16          -6.08778    3.92551  -1.551 0.120943
## V17           0.08283    2.36226   0.035 0.972030
## V18          -0.54177    1.85626  -0.292 0.770394
## V19           0.97785    1.22659   0.797 0.425329
## V20         -42.00748    7.65480  -5.488 4.07e-08 ***
## V21          22.79634    3.59888   6.334 2.38e-10 ***
## V22          12.40505    2.74789   4.514 6.35e-06 ***
## V23          -6.06674    3.86913  -1.568 0.116884
## V24           1.02732    1.10677   0.928 0.353296
## V25          -0.79916    2.29141  -0.349 0.727265
## V26          -0.02063    1.15891  -0.018 0.985798
## V27         -43.62266    6.64190  -6.568 5.11e-11 ***
## V28         -14.50118    4.40100  -3.295 0.000984 ***
## Amount       23.46395    9.54434   2.458 0.013955 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 7242.5  on 284806  degrees of freedom
## Residual deviance: 2232.3  on 284777  degrees of freedom
## AIC: 2292.3
##
## Number of Fisher Scoring iterations: 12
```

```
library(caret)
```

```
## 필요한 패키지를 로딩중입니다: ggplot2
```

```
## 필요한 패키지를 로딩중입니다: lattice
```

```
library(InformationValue)
```

```
##
## 다음의 패키지를 부착합니다: 'InformationValue'
```

```
## The following objects are masked from 'package:caret':
##
##     confusionMatrix, precision, sensitivity, specificity
```

```
library(ISLR)

confusionMatrix(predict(glmAll, type="response") >= 0.5, df$Class)->tt
(tt[1,1]+tt[2,2])/sum(tt)*100
```

```
## [1] 99.92065
```

```
tt[1,1]/(tt[1,1]+tt[1,2])*100
```

```
## [1] 99.98523
```

```
tt[2,2]/(tt[2,1]+tt[2,2])*100
```

```
## [1] 62.60163
```

```
if (tensorflow::tf$executing_eagerly())
  tensorflow::tf$compat$v1$disable_eager_execution()

library(keras)
K <- keras::backend()
# training parameters
vae_batch_size = 160L
fnn_batch_size = 160L
epochs = 1L
vae_ep = 1L
fnn_ep = 7L
vae_flag = 0L

sel_pr_up = 1.0 # upper bound probability for VAE
sel_pr_dw = 0.0 # lower bound probability for VAE
sel_rate = 1.0

vae_w1 = 0.0
vae_w2 = 0.0
vae_w3 = 1.0

# latent and intermediate dimension
latent_dim = 2L
intermediate_dim = 10L
epsilon_std <- 0.1




# input image dimensions
input_shape = c(k+1)
```

```
model1 <- keras_model_sequential() %>%
    layer_dense(units = 1, activation = "sigmoid", input_shape = c(29))
  # %>%     layer_dense(units = 1, activation = "sigmoid")

  model1 %>% compile(
    optimizer = "rmsprop",
    loss = "binary_crossentropy",
    metrics = c("accuracy")
  )
```

```
history2 <- model1 %>% fit(
    as.matrix(df[,-1]), as.matrix(df[,1]),
    shuffle = TRUE,
    epochs = fnn_ep, batch_size = fnn_batch_size,
    verbose = 0
  )


temp3<-predict(model1,as.matrix(df[,-1]))
confusionMatrix(temp3>=0.5, df[,1]) -> tt3
tt3
```

```
##    FALSE TRUE
## 0 284315    0
## 1    433   59
```

```
(tt3[1,1]+tt3[2,2])/(sum(tt3))*100
```

```
## [1] 99.84797
```

```
tt3[1,1]/(tt3[1,1]+tt3[1,2])*100
```

```
## [1] 100
```

```
tt3[2,2]/(tt3[2,1]+tt3[2,2])*100
```

```
## [1] 11.99187
```

```
# 0의 범주를 갖는 행과 1의 범주를 갖는 행을 분리
df_class_0 <- df[df$Class == 0, ]
df_class_1 <- df[df$Class == 1, ]
nrow(df_class_0)
```

```
## [1] 284315
```

```
nrow(df_class_1)
```

```
## [1] 492
```

```
df_class_1 <- df_class_1[sample(nrow(df_class_1),nrow(df_class_0),replace=TRUE),]
oversample <- rbind(df_class_0, df_class_1)
oversample <- oversample[sample(nrow(oversample),nrow(oversample),replace=FALSE),]
```

```
glmOVER =glm(Class~V1+V2+V3+V4+V5+V6+V7+V8+V9+V10+V11+V12+V13+V14+V15+
              V16+V17+V18+V19+V20+V21+V22+V23+V24+V25+V26+V27+V28+Amount,data=oversample,family
="binomial")
```

```
## Warning: glm.fit: 적합된 확률값들이 0 또는 1 입니다
```

```
summary(glmOVER)
```

```
##
## Call:
## glm(formula = Class ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 +
##     V9 + V10 + V11 + V12 + V13 + V14 + V15 + V16 + V17 + V18 +
##     V19 + V20 + V21 + V22 + V23 + V24 + V25 + V26 + V27 + V28 +
##     Amount, family = "binomial", data = oversample)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -8.4904  -0.2606   0.0000   0.0000   3.0062
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -14.82449    3.12346  -4.746 2.07e-06 ***
## V1           36.69628    0.86111  42.615  < 2e-16 ***
## V2           56.23455    1.98542  28.324  < 2e-16 ***
## V3           22.60395    0.63529  35.581  < 2e-16 ***
## V4           17.40754    0.15664 111.131  < 2e-16 ***
## V5          103.23927    2.48786  41.497  < 2e-16 ***
## V6          -54.88292    1.12892 -48.616  < 2e-16 ***
## V7          -94.86630    3.34668 -28.346  < 2e-16 ***
## V8          -37.44538    0.67142 -55.771  < 2e-16 ***
## V9           -8.62804    0.29182 -29.566  < 2e-16 ***
## V10         -34.36475    0.65581 -52.401  < 2e-16 ***
## V11           9.54344    0.15810  60.362  < 2e-16 ***
## V12         -28.84214    0.37010 -77.930  < 2e-16 ***
## V13          -4.58916    0.08713 -52.669  < 2e-16 ***
## V14         -40.58004    0.47380 -85.648  < 2e-16 ***
## V15          -1.09393    0.09783 -11.181  < 2e-16 ***
## V16         -22.43792    0.45698 -49.100  < 2e-16 ***
## V17         -27.79148    0.68516 -40.562  < 2e-16 ***
## V18          -4.65782    0.17516 -26.592  < 2e-16 ***
## V19           3.56622    0.13216  26.984  < 2e-16 ***
## V20         -77.60390    2.04783 -37.896  < 2e-16 ***
## V21           1.41351    0.56000   2.524   0.0116 *
## V22          15.05053    0.25863  58.192  < 2e-16 ***
## V23          29.57722    1.31141  22.554  < 2e-16 ***
## V24          -0.42803    0.10137  -4.223 2.41e-05 ***
## V25           2.72799    0.25506  10.696  < 2e-16 ***
## V26          -2.54890    0.10173 -25.056  < 2e-16 ***
## V27           4.86659    1.23402   3.944 8.02e-05 ***
## V28          40.30486    1.52203  26.481  < 2e-16 ***
## Amount      215.12865    5.54059  38.828  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 788289  on 568629  degrees of freedom
## Residual deviance: 154721  on 568600  degrees of freedom
## AIC: 154781
##
## Number of Fisher Scoring iterations: 13
```

```
confusionMatrix(predict(glmOVER, type="response") >= 0.5, oversample$Class)->tt
(tt[1,1]+tt[2,2])/sum(tt)*100
```

```
## [1] 94.96808
```

```
tt[1,1]/(tt[1,1]+tt[1,2])*100
```

```
## [1] 97.69551
```

```
tt[2,2]/(tt[2,1]+tt[2,2])*100
```

```
## [1] 92.24065
```

```
model1 <- keras_model_sequential() %>%
    layer_dense(units = 1, activation = "sigmoid", input_shape = c(29))
  # %>%     layer_dense(units = 1, activation = "sigmoid")

  model1 %>% compile(
    optimizer = "rmsprop",
    loss = "binary_crossentropy",
    metrics = c("accuracy")
  )
```

```
history2 <- model1 %>% fit(
    as.matrix(oversample[,-1]), as.matrix(oversample[,1]),
    shuffle = TRUE,
    epochs = fnn_ep, batch_size = fnn_batch_size,
    verbose = 0
  )


temp3<-predict(model1,as.matrix(oversample[,-1]))
confusionMatrix(temp3>=0.5, oversample[,1]) -> tt3
tt3
```

```
##    FALSE   TRUE
## 0 280049   4266
## 1  33102 251213
```

```
(tt3[1,1]+tt3[2,2])/(sum(tt3))*100
```

```
## [1] 93.42842
```

```
tt3[1,1]/(tt3[1,1]+tt3[1,2])*100
```

```
## [1] 98.49955
```

```
tt3[2,2]/(tt3[2,1]+tt3[2,2])*100
```

```
## [1] 88.35728
```

```
### Oversampling with random copy########


##data partition##
df = df[sample(nrow(df),nrow(df),replace=FALSE),]
# 0의 범주를 갖는 행과 1의 범주를 갖는 행을 분리
df_class_0 <- df[df$Class == 0, ]
df_class_1 <- df[df$Class == 1, ]

# 0의 범주를 8대2 비율로 train과 test로 나눔
train_class_0_rows <- round(0.8 * nrow(df_class_0))
train_class_0 <- df_class_0[1:train_class_0_rows, ]
test_class_0 <- df_class_0[(train_class_0_rows + 1):nrow(df_class_0), ]

nrow(train_class_0)
```

```
## [1] 227452
```

```
# 1의 범주를 8대2 비율로 train과 test로 나눔

train_class_1_rows <- round(0.8 * nrow(df_class_1))
train_class_1 <- df_class_1[1:train_class_1_rows, ]
train_class_1 <- train_class_1[sample(nrow(train_class_1),nrow(train_class_0),replace=TRUE),]
test_class_1 <- df_class_1[(train_class_1_rows + 1):nrow(df_class_1), ]

# train과 test를 합쳐 최종 train_df와 test_df 생성
df <- rbind(train_class_0, train_class_1)
dfTS <- rbind(test_class_0, test_class_1)
dfTR0 = df[df$Class==0,]
overDF1 = df[df$Class==1,]
table(df$Class)
```

```
##
##      0      1
## 227452 227452
```

```
table(dfTS$Class)
```

```
##
##     0     1
## 56863    98
```

```
glmOver =glm(Class~V1+V2+V3+V4+V5+V6+V7+V8+V9+V10+V11+V12+V13+V14+V15+
               V16+V17+V18+V19+V20+V21+V22+V23+V24+V25+V26+V27+V28+Amount,data=df,family="binom
ial")
```

```
## Warning: glm.fit: 알고리즘이 수렴하지 않았습니다
```

```
## Warning: glm.fit: 적합된 확률값들이 0 또는 1 입니다
```

```
summary(glmOver)
```

```
##
## Call:
## glm(formula = Class ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 +
##     V9 + V10 + V11 + V12 + V13 + V14 + V15 + V16 + V17 + V18 +
##     V19 + V20 + V21 + V22 + V23 + V24 + V25 + V26 + V27 + V28 +
##     Amount, family = "binomial", data = df)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -8.4904  -0.2401   0.0000   0.0000   2.8418
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   9.5659     3.5986   2.658  0.00785 **
## V1           29.0386     0.9291  31.256  < 2e-16 ***
## V2           40.9521     2.2651  18.080  < 2e-16 ***
## V3           16.8902     0.7103  23.779  < 2e-16 ***
## V4           16.5279     0.1757  94.063  < 2e-16 ***
## V5           80.8187     2.7664  29.214  < 2e-16 ***
## V6          -52.3413     1.3074 -40.034  < 2e-16 ***
## V7          -85.9124     3.7874 -22.684  < 2e-16 ***
## V8          -37.3726     0.7619 -49.054  < 2e-16 ***
## V9          -11.1676     0.3253 -34.330  < 2e-16 ***
## V10         -26.5054     0.7250 -36.561  < 2e-16 ***
## V11           8.6032     0.1779  48.370  < 2e-16 ***
## V12         -29.5928     0.3911 -75.671  < 2e-16 ***
## V13          -4.2902     0.1047 -40.981  < 2e-16 ***
## V14         -34.6142     0.4923 -70.304  < 2e-16 ***
## V15          -0.7128     0.1149  -6.205 5.46e-10 ***
## V16         -19.5370     0.4883 -40.013  < 2e-16 ***
## V17         -17.5536     0.7100 -24.724  < 2e-16 ***
## V18          -3.5964     0.1912 -18.807  < 2e-16 ***
## V19           1.1534     0.1502   7.679 1.61e-14 ***
## V20         -54.4960     2.2966 -23.729  < 2e-16 ***
## V21           6.6504     0.6200  10.727  < 2e-16 ***
## V22          13.8135     0.3029  45.610  < 2e-16 ***
## V23          22.8353     1.4904  15.322  < 2e-16 ***
## V24           0.5545     0.1211   4.577 4.72e-06 ***
## V25           1.5636     0.3009   5.196 2.04e-07 ***
## V26          -2.2349     0.1157 -19.323  < 2e-16 ***
## V27          -6.5163     1.4373  -4.534 5.79e-06 ***
## V28          18.5212     1.6702  11.089  < 2e-16 ***
## Amount      173.5041     6.2163  27.911  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 630631  on 454903  degrees of freedom
## Residual deviance: 111461  on 454874  degrees of freedom
## AIC: 111521
##
## Number of Fisher Scoring iterations: 25
```

```
summary(glmOver)
```

```
##
## Call:
## glm(formula = Class ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 +
##     V9 + V10 + V11 + V12 + V13 + V14 + V15 + V16 + V17 + V18 +
##     V19 + V20 + V21 + V22 + V23 + V24 + V25 + V26 + V27 + V28 +
##     Amount, family = "binomial", data = df)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -8.4904  -0.2401   0.0000   0.0000   2.8418
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   9.5659     3.5986   2.658  0.00785 **
## V1           29.0386     0.9291  31.256  < 2e-16 ***
## V2           40.9521     2.2651  18.080  < 2e-16 ***
## V3           16.8902     0.7103  23.779  < 2e-16 ***
## V4           16.5279     0.1757  94.063  < 2e-16 ***
## V5           80.8187     2.7664  29.214  < 2e-16 ***
## V6          -52.3413     1.3074 -40.034  < 2e-16 ***
## V7          -85.9124     3.7874 -22.684  < 2e-16 ***
## V8          -37.3726     0.7619 -49.054  < 2e-16 ***
## V9          -11.1676     0.3253 -34.330  < 2e-16 ***
## V10         -26.5054     0.7250 -36.561  < 2e-16 ***
## V11           8.6032     0.1779  48.370  < 2e-16 ***
## V12         -29.5928     0.3911 -75.671  < 2e-16 ***
## V13          -4.2902     0.1047 -40.981  < 2e-16 ***
## V14         -34.6142     0.4923 -70.304  < 2e-16 ***
## V15          -0.7128     0.1149  -6.205 5.46e-10 ***
## V16         -19.5370     0.4883 -40.013  < 2e-16 ***
## V17         -17.5536     0.7100 -24.724  < 2e-16 ***
## V18          -3.5964     0.1912 -18.807  < 2e-16 ***
## V19           1.1534     0.1502   7.679 1.61e-14 ***
## V20         -54.4960     2.2966 -23.729  < 2e-16 ***
## V21           6.6504     0.6200  10.727  < 2e-16 ***
## V22          13.8135     0.3029  45.610  < 2e-16 ***
## V23          22.8353     1.4904  15.322  < 2e-16 ***
## V24           0.5545     0.1211   4.577 4.72e-06 ***
## V25           1.5636     0.3009   5.196 2.04e-07 ***
## V26          -2.2349     0.1157 -19.323  < 2e-16 ***
## V27          -6.5163     1.4373  -4.534 5.79e-06 ***
## V28          18.5212     1.6702  11.089  < 2e-16 ***
## Amount      173.5041     6.2163  27.911  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 630631  on 454903  degrees of freedom
## Residual deviance: 111461  on 454874  degrees of freedom
## AIC: 111521
##
## Number of Fisher Scoring iterations: 25
```

```
confusionMatrix(predict(glmOver, type="response") >= 0.5, df[,1])->tt
tt
```

```
##     FALSE   TRUE
## 0 222880   4572
## 1  15068 212384
```

```
(tt[1,1]+tt[2,2])/sum(tt)*100
```

```
## [1] 95.68261
```

```
tt[1,1]/(tt[1,1]+tt[1,2])*100
```

```
## [1] 97.98991
```

```
tt[2,2]/(tt[2,1]+tt[2,2])*100
```

```
## [1] 93.37531
```

```
confusionMatrix(predict(glmOver, as.data.frame(dfTS),type="response") >= 0.5, dfTS[,1])->tt1
tt1
```

```
##   FALSE TRUE
## 0 55682 1181
## 1    16   82
```

```
(tt1[1,1]+tt1[2,2])/sum(tt1)*100
```

```
## [1] 97.89856
```

```
tt1[1,1]/(tt1[1,1]+tt1[1,2])*100
```

```
## [1] 97.92308
```

```
tt1[2,2]/(tt1[2,1]+tt1[2,2])*100
```

```
## [1] 83.67347
```

```
table(dfTS$Class)
```

```
##
##     0     1
## 56863    98
```

```
#####VAE fitting#######

if (tensorflow::tf$executing_eagerly())
  tensorflow::tf$compat$v1$disable_eager_execution()

library(keras)
K <- keras::backend()

# training parameters
vae_batch_size = 160L
fnn_batch_size = 160L
epochs = 1L
vae_ep = 1L
fnn_ep = 7L
vae_flag = 0L

sel_pr_up = 1.0 # upper bound probability for VAE
sel_pr_dw = 0.0 # lower bound probability for VAE
sel_rate = 0.6

vae_w1 = 0.0
vae_w2 = 0.0
vae_w3 = 1.0

# latent and intermediate dimension
latent_dim = 2L
intermediate_dim = 10L
epsilon_std <- 0.1



# input image dimensions
input_shape = c(k+1)

# encoder
original_input_size = c(k+1)
inp <- layer_input(shape = original_input_size)
x <- layer_lambda(inp, f=function(x) {x[,2:(k+1)]})
y <- layer_lambda(inp, f=function(x) {x[,1:1]})

hidden_1 <- layer_dense(x, units=intermediate_dim, activation="relu")
dropout_1 <- layer_dropout(hidden_1, rate = 0.5)
hidden_2 <- layer_dense(dropout_1, units=intermediate_dim, activation="relu")
dropout_2 <- layer_dropout(hidden_2, rate = 0.5)

z_mean = layer_dense(dropout_2, units = latent_dim)
z_log_var <- layer_dense(hidden_2, units = latent_dim)

# sampling part
sampling <- function(args) {
  z_mean <- args[, 1:(latent_dim)]
  z_log_var <- args[, (latent_dim + 1):(2 * latent_dim)]

  epsilon <- k_random_normal(
    shape = c(k_shape(z_mean)[[1]]),
```

```
    mean = 0.,
    stddev = epsilon_std
  )
  z_mean + k_exp(z_log_var) * epsilon
}

z <- layer_concatenate(list(z_mean, z_log_var)) %>% layer_lambda(sampling)

# decoder + prediction model
output_shape = c(vae_batch_size, k)

decoder_hidden = layer_dense(units=intermediate_dim, activation="relu")
decoder_upsample = layer_dense(units = intermediate_dim, activation="relu")
decoder_reshape <- layer_reshape(target_shape = intermediate_dim)
decoder_hidden1 = layer_dense(units=k, activation="sigmoid")

pred_layer = layer_dense(units = 1, activation = "sigmoid")

hidden_decoded  = decoder_hidden(z)
up_decoded = decoder_upsample(hidden_decoded)
reshape_decoded <- decoder_reshape(up_decoded)
hidden1_decoded = decoder_hidden1(reshape_decoded)

y_pred =pred_layer(hidden1_decoded)


vae_loss <- function(y, y_pred) {
  x <- k_flatten(x)
  x_decoded_mean_squash <- k_flatten(hidden1_decoded)
  xent_loss <- 1.0 * # initial weight = 1
    loss_mean_squared_error(x, x_decoded_mean_squash) # loss_categorical_crossentropy도 시도해
볼 것
  kl_loss <- -0.5 * k_mean(1 + z_log_var - k_square(z_mean) -  # initial weight = -0.5
                           k_exp(z_log_var), axis = -1L)
  p_loss <- 1.0 * loss_binary_crossentropy(y, y_pred) # initial weight = 0 * 12000

  k_mean(xent_loss*vae_w1 + kl_loss*vae_w2 + p_loss*vae_w3)
}

vae <- keras_model(inp, y_pred)
optimizers <- keras::keras$optimizers
vae %>% compile(optimizer = optimizers$legacy$RMSprop(learning_rate=0.0001), loss = vae_loss,
                metrics = c("accuracy"))
# summary(vae)

## encoder: model to project inputs on the latent space
# encoder <- keras_model(inp, list(z_mean, z_log_var))

## build a digit generator that can sample from the learned distribution
# gen_decoder_input <- layer_input(shape = latent_dim)
# gen_hidden_decoded <- decoder_hidden(gen_decoder_input)
# gen_up_decoded <- decoder_upsample(gen_hidden_decoded)
# gen_hidden1_decoded <- decoder_hidden1(gen_up_decoded)

# generator <- keras_model(gen_decoder_input, gen_hidden1_decoded)
```

```
vae1 <- keras_model(inp, hidden1_decoded) # can be used for generating synthetic samples for ca
se 0 and 1

# summary(vae1)
```

```
model1 <- keras_model_sequential() %>%
    layer_dense(units = 1, activation = "sigmoid", input_shape = c(29))
  # %>%     layer_dense(units = 1, activation = "sigmoid")

  model1 %>% compile(
    optimizer = "rmsprop",
    loss = "binary_crossentropy",
    metrics = c("accuracy")
  )
```

```
history2 <- model1 %>% fit(
    as.matrix(df[,-1]), as.matrix(df[,1]),
    shuffle = TRUE,
    epochs = fnn_ep, batch_size = fnn_batch_size,
    validation_data = list(as.matrix(dfTS[,-1]), as.matrix(dfTS[,1]))
    , verbose = 0
  )

temp3<-predict(model1,as.matrix(df[,-1]))
confusionMatrix(temp3>=0.5, df[,1]) -> tt3
tt3
```

```
##    FALSE   TRUE
## 0 224590   2862
## 1  24233 203219
```

```
(tt3[1,1]+tt3[2,2])/(sum(tt3))*100
```

```
## [1] 94.0438
```

```
tt3[1,1]/(tt3[1,1]+tt3[1,2])*100
```

```
## [1] 98.74171
```

```
tt3[2,2]/(tt3[2,1]+tt3[2,2])*100
```

```
## [1] 89.34588
```

```
temp3 <- predict(model1, as.matrix(dfTS[,-1]))
confusionMatrix(temp3>=0.5, dfTS[,1]) -> tt3
tt3
```

```
##    FALSE  TRUE
## 0 56110   753
## 1    16    82
```

```
(tt3[1,1]+tt3[2,2])/(sum(tt3))*100
```

```
## [1] 98.64995
```

```
tt3[1,1]/(tt3[1,1]+tt3[1,2])*100
```

```
## [1] 98.67576
```

```
tt3[2,2]/(tt3[2,1]+tt3[2,2])*100
```

```
## [1] 83.67347
```

```
# j : number of epoch
# i : number of batchs for one epoch

for (j in epochs) {

  # FNN MODEL FITTING

  model1 <- keras_model_sequential() %>%
    layer_dense(units = 1, activation = "sigmoid", input_shape = c(29))
  # %>%     layer_dense(units = 1, activation = "sigmoid")

  model1 %>% compile(
    optimizer = "rmsprop",
    loss = "binary_crossentropy",
    metrics = c("accuracy")
  )

  # Insert VAE part here if needed

  if(vae_flag == 1){
    history = vae %>% fit(
      as.matrix(df), as.matrix(df[,1]),
      shuffle = TRUE,
      epochs = vae_ep,
      batch_size = vae_batch_size,
      validation_data = list(as.matrix(df_test), as.matrix(df_test[,1])),
      verbose = 0
    )
  }


  # whole train and test data preparation

  library(dplyr)

  temp0 <- predict(vae1, as.matrix(df))
  temp  <- predict(vae, as.matrix(df))
  temp1 <- as.data.frame(cbind(c(1), temp0[temp<=quantile(temp, sel_pr_up) & temp>=quantile(tem
p, sel_pr_dw),]))
  names(temp1) = names(dfTR0)
  samp_ind = sample(1:nrow(temp), size = round(nrow(temp)*sel_rate))
  temp1 <- temp1[samp_ind,]

  temp2 <- dfTR0 %>% sample_frac(nrow(temp1)/nrow(dfTR0), replace = TRUE)
  train_df <-rbind(temp2, dfTR0, overDF1, temp1)
  train_df <- train_df[sample(1:nrow(train_df)),]

  # print(i)

  ## ---- Fitting ------------------------------------------------------------

  history2 <- model1 %>% fit(
    as.matrix(train_df[,-1]), as.matrix(train_df[,1]),
    shuffle = TRUE,
    epochs = fnn_ep, batch_size = fnn_batch_size,
```

```
    validation_data = list(as.matrix(dfTS[,-1]), as.matrix(dfTS[,1]))
    , verbose = 0
  )

  print("FNN")
  print(history2)
  # plot(history2)
  print("VAE")
  if(vae_flag == 1){
    print(history)}
  #plot(history)
  print(j)

}
```

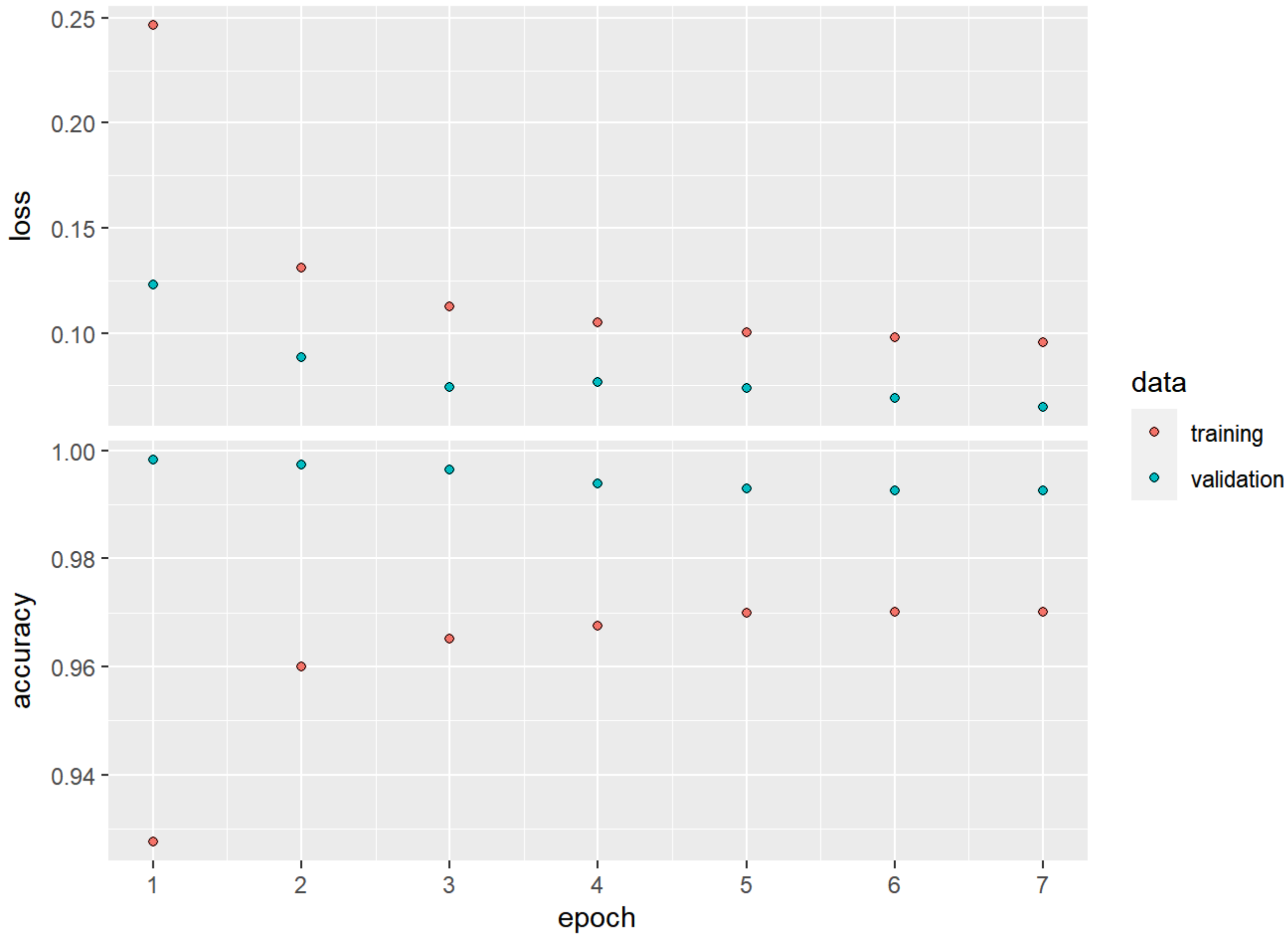
```
##
## 다음의 패키지를 부착합니다: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
## [1] "FNN"
## Trained on 1,000,788 samples (batch_size=160, epochs=7)
## Final epoch (plot to see history):
##         loss: 0.09578
##     accuracy: 0.9702
##     val_loss: 0.06468
## val_accuracy: 0.9927
## [1] "VAE"
## [1] 1
```

```
if(vae_flag == 1){
  plot(history)
}
plot(history2)
```




```
temp3 <- predict(vae, as.matrix(dfTS))
confusionMatrix(temp3>=0.5, dfTS[,1]) -> tt
tt
```

```
##   FALSE
## 0 56863
## 1    98
```

```
(tt[1,1]+tt[2,2])/(sum(tt))*100
```

```
## numeric(0)
```

```
tt[1,1]/(tt[1,1]+tt[1,2])*100
```

```
## numeric(0)
```

```
tt[2,2]/(tt[2,1]+tt[2,2])*100
```

```
## numeric(0)
```

```
temp3 <- predict(model1, as.matrix(dfTS[,-1]))
confusionMatrix(temp3>=0.5, dfTS[,1]) -> tt3
tt3
```

```
##    FALSE TRUE
## 0 56464  399
## 1    17   81
```

```
(tt3[1,1]+tt3[2,2])/(sum(tt3))*100
```

```
## [1] 99.26968
```

accuracy for case 0

```
tt3[1,1]/(tt3[1,1]+tt3[1,2])*100
```

```
## [1] 99.29831
```

accuracy for case 1

```
tt3[2,2]/(tt3[2,1]+tt3[2,2])*100
```

```
## [1] 82.65306
```