Untitled

2025-04-09

```
if(!require(readxl)){install.packages("readxl")}
library(readxl)
if(!require(dplyr)){install.packages("dplyr")}
library(dplyr)
if(!require(lubridate)){install.packages("lubridate")}
library(lubridate)
if(!require(tidyverse)){install.packages("tidyverse")}
library(tidyverse)
if (!require(glmnet)) install.packages("glmnet")
library(glmnet)
if (!require(keras)) install.packages("keras")
library(keras)
if (!require(elasticnet)) install.packages("elasticnet")
library(elasticnet)
if (!require(tidyr)) install.packages("tidyr")
library(tidyr)
if (!require(quantmod)) install.packages("quantmod")
library(quantmod)
if (!require(tidyquant)) install.packages("tidyquant")
library(tidyquant)
set.seed(123)
tensorflow::tf$random$set_seed(123)
```

```
# Use local drive address to load the data
load("data_ml.RData")
# preview first few rows of the dataset
head(data_ml, 6)
```

```
## # A tibble: 6 × 99
##   stock_id date       Advt_12M_Usd Advt_3M_Usd Advt_6M_Usd Asset_Turnover Bb_Yld
##      <int> <date>            <dbl>       <dbl>       <dbl>          <dbl>  <dbl>
## 1       13 2006-12-31         0.25        0.33        0.27           0.22   0.33
## 2       13 2007-01-31         0.25        0.32        0.28           0.22   0.4 
## 3       13 2007-02-28         0.26        0.3         0.3            0.22   0.15
## 4       17 2015-03-31         0.73        0.64        0.7            0.4    0.47
## 5       17 2015-04-30         0.72        0.62        0.66           0.4    0.46
## 6       17 2015-05-31         0.71        0.63        0.64           0.4    0.47
## # ℹ 92 more variables: Bv <dbl>, Capex_Ps_Cf <dbl>, Capex_Sales <dbl>,
## #   Cash_Div_Cf <dbl>, Cash_Per_Share <dbl>, Cf_Sales <dbl>, Debtequity <dbl>,
## #   Div_Yld <dbl>, Dps <dbl>, Ebit_Bv <dbl>, Ebit_Noa <dbl>, Ebit_Oa <dbl>,
## #   Ebit_Ta <dbl>, Ebitda_Margin <dbl>, Eps <dbl>, Eps_Basic <dbl>,
## #   Eps_Basic_Gr <dbl>, Eps_Contin_Oper <dbl>, Eps_Dil <dbl>, Ev <dbl>,
## #   Ev_Ebitda <dbl>, Fa_Ci <dbl>, Fcf <dbl>, Fcf_Bv <dbl>, Fcf_Ce <dbl>,
## #   Fcf_Margin <dbl>, Fcf_Noa <dbl>, Fcf_Oa <dbl>, Fcf_Ta <dbl>, …
```

```
# Download SP500 index (^GSPC) from Yahoo Finance
# Get daily data first
sp500_data <- tq_get("^GSPC",
                     from = "2016-12-31",
                     to = "2018-12-31",
                     get = "stock.prices")

# Calculate monthly simple returns (not log returns for consistency)
sp500_monthly_returns <- sp500_data %>%
  tq_transmute(
    select = adjusted,
    mutate_fun = periodReturn,
    period = "monthly",
    type = "arithmetic"  # simple returns
  ) %>%
  rename(sp500_return = monthly.returns)
```

```
# Clean and sort the dataset
target_leakage <- c("R3M_Usd", "R6M_Usd", "R12M_Usd")
data_ml <- data_ml %>%
  distinct() %>% #remove duplicates
  filter(date > "1999-12-31",         # Keep the date with sufficient data points
         date < "2019-01-01") %>%
  select(-all_of(target_leakage)) %>% # remove predictors that may cause target leakage
  arrange(stock_id, date)             # Order the data
```

```
data_ml <- data_ml %>%
  mutate(target_return = R1M_Usd) %>%
  filter(!is.na(target_return))  # Remove rows without a future return

# verify predictors are all scaled (standardized)
col_mean <- apply(data_ml[, 3:ncol(data_ml)], 2, mean)
col_sd <- apply(data_ml[, 3:ncol(data_ml)], 2, sd)

standardization_check <- data.frame(Column = colnames(data_ml[, 3:ncol(data_ml)]),
                                    mean = col_mean,
                                    sd = col_sd)

mean_range <- c(min(standardization_check$mean), max(standardization_check$mean))
sd_range <- c(min(standardization_check$sd), max(standardization_check$sd))

cat('Mean range:', mean_range[1], '-', mean_range[2], '\n')
```

```
## Mean range: 0.01273207 - 0.5056039
```

```
cat('Standard Deviation range:', sd_range[1], '-', sd_range[2], '\n')
```

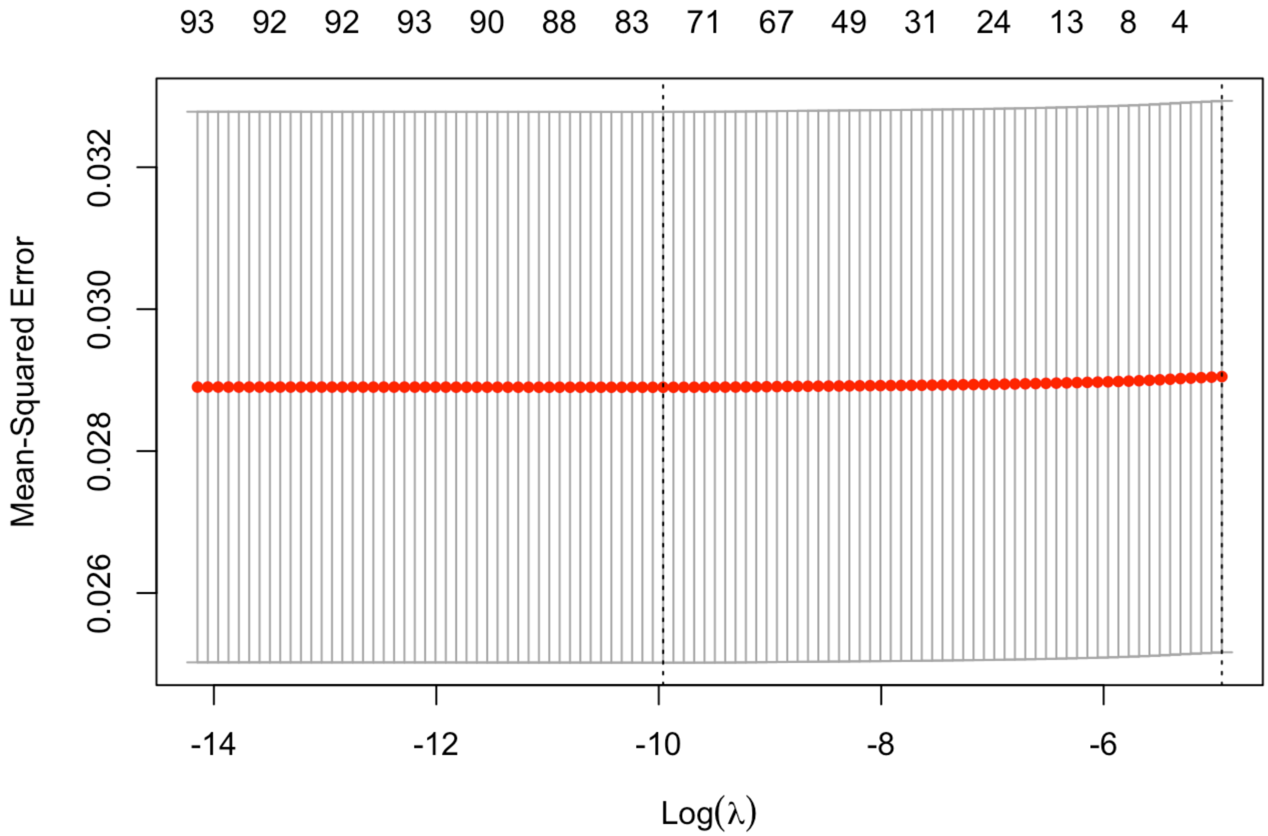
```
## Standard Deviation range: 0.1764311 - 0.2890281
```

```r
# standardize the predictors
data_ml <- data_ml %>% mutate(across(
  .cols = -all_of(c('stock_id', 'date', 'R1M_Usd', 'target_return')),
  .fns = ~scale(.)[,1]
))
```

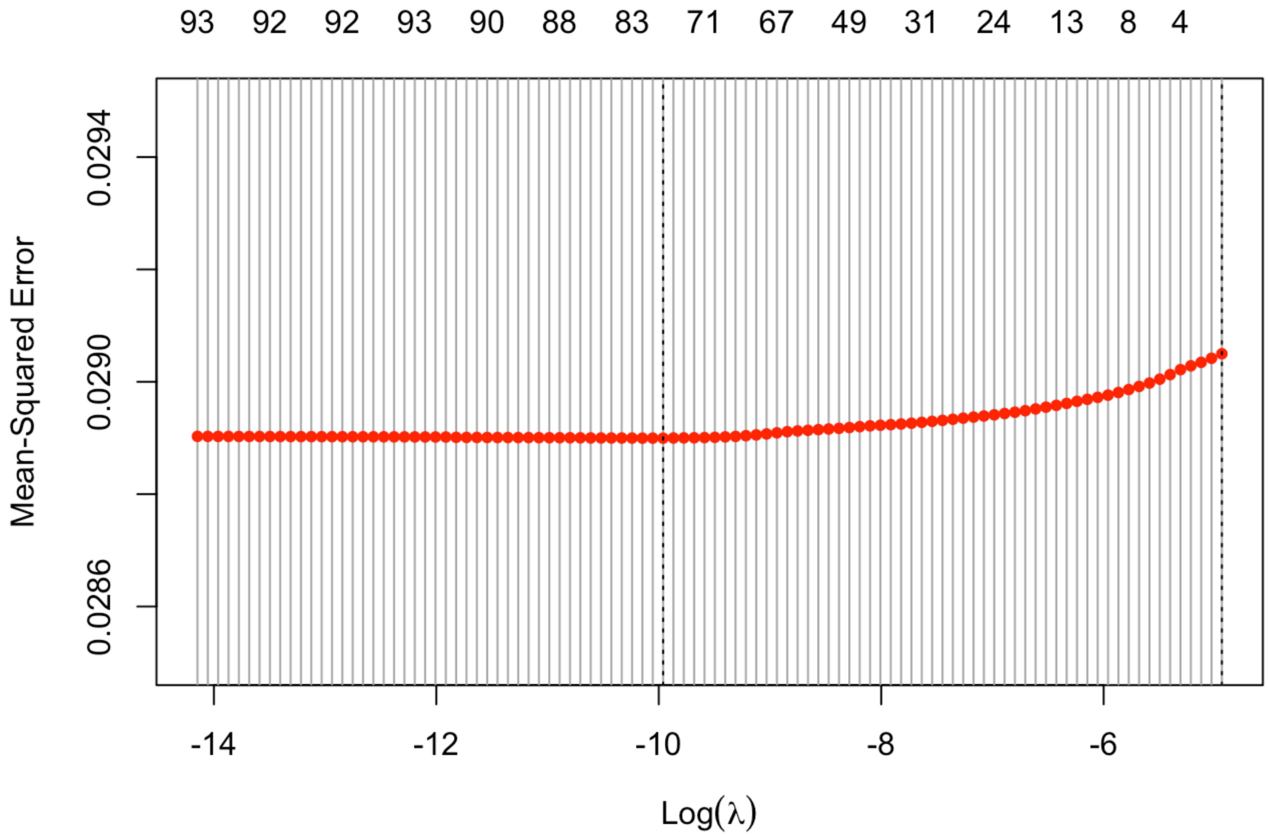
```r
# Split into training, validation, and test sets
training_set <- filter(data_ml, date < as.Date("2014-01-15"))
validation_set <- filter(data_ml, (date >= as.Date("2014-01-15")
                                   & date < as.Date("2017-01-15")))
testing_set <- filter(data_ml, date >= as.Date("2017-01-15"))
```

```r
# Define features to be used for LASSO and NN
features <- training_set %>%
  select(-stock_id, -date, -R1M_Usd, -target_return) %>%
  colnames()

# Create matrix inputs for glmnet
X_train <- as.matrix(training_set[, features])
y_train <- training_set$target_return

X_val <- as.matrix(validation_set[, features])
y_val <- validation_set$target_return

X_test <- as.matrix(testing_set[, features])
y_test <- testing_set$target_return
```

```r
# Train LASSO model with cross-validation
cv_lasso <- cv.glmnet(X_train, y_train, alpha = 1, nfolds = 10)

# plot CV error vs. lambda
plot(cv_lasso)
```




```
plot(cv_lasso, ylim=c(0.0285, 0.0295))
```




```
# Choose lambda with lowest CV error
best_lambda <- cv_lasso$lambda.min
cat("Best lambda:", best_lambda, "\n")
```

```
## Best lambda: 4.720529e-05
```

```
# Extract coefficients at best lambda
lasso_coef <- coef(cv_lasso, s = best_lambda)
selected_features <- rownames(lasso_coef)[which(lasso_coef[, 1] != 0)]
selected_features <- setdiff(selected_features, "(Intercept)")  # remove intercept
lasso_coef <- as.matrix(lasso_coef) %>%
  as.data.frame() %>%
  slice(-1) %>%
  filter(s1 !=0) %>%
  arrange(desc(abs(s1)))
lasso_coef
```

```
##                                          s1
## Mkt_Cap_3M_Usd                 -3.629189e-02
## Mkt_Cap_12M_Usd                 2.723486e-02
## Fcf_Bv                          5.963966e-03
## Ocf_Ta                          5.231126e-03
## Mom_5M_Usd                     -5.048371e-03
## Pb                             -4.829150e-03
## Fcf_Yld                        -4.545451e-03
## Ocf_Bv                         -4.478071e-03
## Mom_Sharp_11M_Usd               4.362510e-03
## Ni                              4.138691e-03
## Rev                             4.101435e-03
## Mom_11M_Usd                    -3.778312e-03
## Mom_Sharp_5M_Usd                3.566957e-03
## Ni_Oa                          -3.526356e-03
## Sales_Ps                       -3.470986e-03
## Ebitda_Margin                   3.230367e-03
## Fcf                             3.210102e-03
## Ocf_Margin                     -3.144972e-03
## Total_Debt                      2.865797e-03
## Vol1Y_Usd                       2.705398e-03
## Debtequity                     -2.623354e-03
## Vol3Y_Usd                       2.594227e-03
## Pe                             -2.570003e-03
## Mkt_Cap_6M_Usd                 -2.483954e-03
## Ni_Avail_Margin                 2.454994e-03
## Bv                             -2.236407e-03
## Roce                            2.165793e-03
## Eps                            -2.144026e-03
## Asset_Turnover                 -2.106678e-03
## Cash_Per_Share                 -2.101890e-03
## Ocf_Oa                          2.063179e-03
## Net_Margin                     -2.037736e-03
## Share_Turn_3M                   2.025437e-03
## Total_Liabilities_Total_Assets  2.017764e-03
## Fcf_Tbv                         1.919880e-03
## Share_Turn_12M                 -1.891544e-03
## Oper_Ps_Net_Cf                  1.882790e-03
## Ocf_Tbv                        -1.750312e-03
## Fa_Ci                           1.748320e-03
## Ni_Toa                          1.701317e-03
## Eps_Contin_Oper                -1.660257e-03
## Cf_Sales                       -1.589986e-03
## Ev_Ebitda                       1.577615e-03
## Ocf                             1.543912e-03
## Ebit_Noa                       -1.507896e-03
## Advt_3M_Usd                    -1.475341e-03
## Cash_Div_Cf                    -1.460924e-03
## Dps                             1.444972e-03
## Recurring_Earning_Total_Assets -1.292998e-03
## Div_Yld                        -1.255963e-03
## Capex_Sales                    -1.211795e-03
```

```
## Tev_Less_Mktcap                  1.183148e-03
## Roa                             -1.149997e-03
## Fcf_Ce                           1.121742e-03
## Capex_Ps_Cf                     -1.044732e-03
## Fcf_Margin                      -1.019915e-03
## Total_Debt_Capital               8.994458e-04
## Op_Margin                       -8.774057e-04
## Fcf_Toa                         -8.643440e-04
## Ebit_Oa                         -8.529555e-04
## Interest_Expense                 8.319396e-04
## Return_On_Capital                8.155614e-04
## Ebit_Ta                         -8.129246e-04
## Fcf_Oa                          -8.058481e-04
## Net_Debt                        -7.743785e-04
## Roc                              7.439507e-04
## Int_Rev                         -6.715502e-04
## Ebit_Bv                         -6.461071e-04
## Ptx_Mgn                          6.414147e-04
## Bb_Yld                           6.220725e-04
## Tot_Debt_Rev                    -6.179719e-04
## Eps_Basic_Gr                    -4.634775e-04
## Fcf_Ta                          -4.352807e-04
## Fcf_Noa                          2.200777e-04
## Eps_Dil                          1.469795e-04
## Ocf_Ce                           1.099358e-04
## Oa                              -8.445767e-05
## Advt_12M_Usd                     2.002129e-05
## Total_Capital                    1.956835e-05
## Free_Ps_Cf                      -7.786797e-06
## Eps_Basic                        4.753347e-06
```

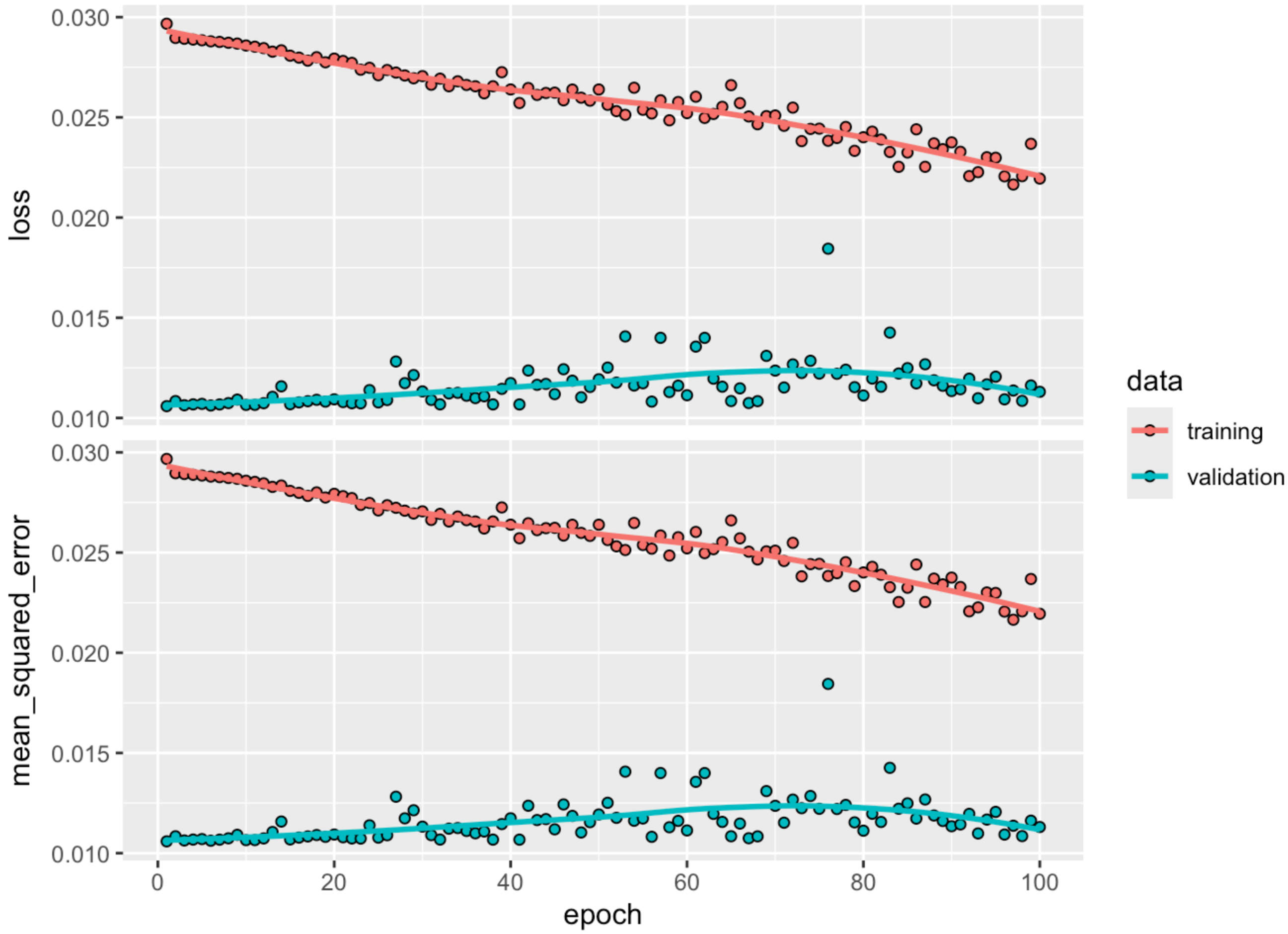
```
# Prepare training, validation, and test sets using only selected LASSO features
X_train_nn_lasso <- as.matrix(training_set[, selected_features])
X_val_nn_lasso   <- as.matrix(validation_set[, selected_features])
X_test_nn_lasso  <- as.matrix(testing_set[, selected_features])

y_train_nn_lasso <- training_set$target_return
y_val_nn_lasso   <- validation_set$target_return
y_test_nn_lasso  <- testing_set$target_return
```

```
# Define a grid of hyperparameters
units1_list <- c(16, 32, 64)
units2_list <- c(8, 16, 32)
learning_rates <- c(0.001, 0.0005)
batch_sizes <- c(64, 128)

results <- data.frame()

for (units1 in units1_list) {
  for (units2 in units2_list) {
    for (lr in learning_rates) {
      for (bs in batch_sizes) {

        # Build model
        # Starts a new model using a sequential stack of layers.
        # Layer 1.  x neurons in the first hidden layer.
        #     Applies ReLU activation (max(0, x)), good for non-linearity.
        #     Number of input features (i.e., number of selected LASSO predictors).
        # Layer 2.  x neurons, again with ReLU.
        # Layer 3.  1 output neuron: predicting a return.
        model <- keras_model_sequential() %>%
          layer_dense(units = units1, activation = "relu", input_shape = ncol(X_train_nn
_lasso)) %>%
          layer_dense(units = units2, activation = "relu") %>%
          layer_dense(units = 1)

        # Compile model
        # Mean Squared Error is the objective to minimize (standard for regression).
        # Uses the Adam optimizer, which is adaptive and works well with most problems.
        # Also tracks MSE while training, for reporting
        # Learning Rate x
        model %>% compile(
          loss = "mse",
          optimizer = optimizer_adam(learning_rate = lr),
          metrics = list("mean_squared_error")
        )

        # Train model
        # Train the model for 100 full passes through the data.
        # Use mini-batches of x rows for updates (tradeoff between speed and stability).
        # Monitors performance on validation set at the end of each epoch.
        history <- model %>% fit(
          x = X_train_nn_lasso,
          y = y_train_nn_lasso,
          epochs = 100,
          batch_size = bs,
          validation_data = list(X_val_nn_lasso, y_val_nn_lasso),
          verbose = 0
        )

        # Get final validation MSE
        val_mse <- tail(history$metrics$val_mean_squared_error, 1)
```

```
        # Record result
        results <- rbind(results, data.frame(
          units1 = units1,
          units2 = units2,
          learning_rate = lr,
          batch_size = bs,
          val_mse = val_mse
        ))
      }
    }
  }
}

# Find the best hyperparameters
best_result <- results[which.min(results$val_mse), ]
print(best_result)
```

```
##   units1 units2 learning_rate batch_size   val_mse
## 2     16      8         0.001        128 0.0106187
```

```r
# Build model
# Starts a new model using a sequential stack of layers.
# Layer 1.  x neurons in the first hidden layer.
#     Applies ReLU activation (max(0, x)), good for non-linearity.
#     Number of input features (i.e., number of selected LASSO predictors).
# Layer 2.  x neurons, again with ReLU.
# Layer 3.  1 output neuron: predicting a return.
model <- keras_model_sequential() %>%
  layer_dense(units = best_result$units1, activation = "relu",
              input_shape = ncol(X_train_nn_lasso)) %>%
  layer_dense(units = best_result$units2, activation = "relu") %>%
  layer_dense(units = 1)  # output layer

# Compile model
# Mean Squared Error is the objective to minimize (standard for regression).
# Uses the Adam optimizer, which is adaptive and works well with most problems.
# Also tracks MSE while training, for reporting
# Learning Rate x
model %>% compile(
  loss = "mse",
  optimizer = optimizer_adam(learning_rate = best_result$learning_rate),
  metrics = list("mean_squared_error")
)

# Train model
# Train the model for 100 full passes through the data.
# Use mini-batches of x rows for updates (tradeoff between speed and stability).
# Monitors performance on validation set at the end of each epoch.
history <- model %>% fit(
  x = X_train_nn_lasso,
  y = y_train_nn_lasso,
  epochs = 100,
  batch_size = best_result$batch_size,
  validation_data = list(X_val_nn_lasso, y_val_nn_lasso),
  verbose = 0
)
plot(history)
```




```
#title(main = "LASSO-Selected Features Neural Network")
```

```
# Predict and calculate RMSE
pred_nn_lasso_train <- model %>% predict(X_train_nn_lasso)
```

```
## 6192/6192 - 1s - 956ms/epoch - 154us/step
```

```
pred_nn_lasso_test <- model %>% predict(X_test_nn_lasso)
```

```
## 853/853 - 0s - 134ms/epoch - 158us/step
```

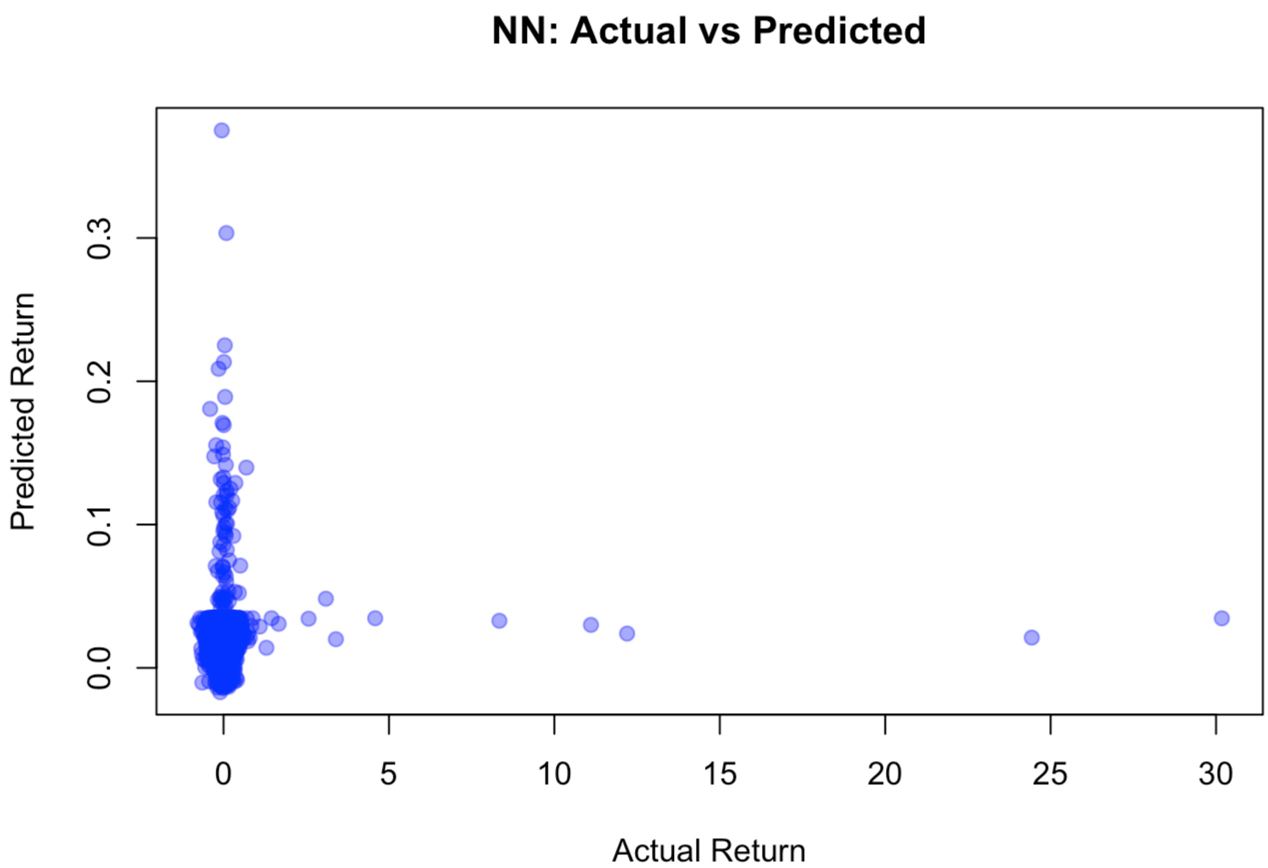
```
rmse <- function(actual, predicted) {
  sqrt(mean((actual - predicted)^2))
}

cat("LASSO + NN Train RMSE:", rmse(y_train_nn_lasso, pred_nn_lasso_train), "\n")
```

```
## LASSO + NN Train RMSE: 0.1466643
```

```
cat("LASSO + NN Test RMSE:", rmse(y_test_nn_lasso, pred_nn_lasso_test), "\n")
```

```
## LASSO + NN Test RMSE: 0.2803473
```

```
plot(y_test_nn_lasso, pred_nn_lasso_test,
     xlab = "Actual Return",
     ylab = "Predicted Return",
     main = "NN: Actual vs Predicted",
     pch = 19, col = rgb(0,0,1,0.4))
```




```
# SPCA transformed matrices for training, validation and test
X_train_spca <- as.matrix(training_set[, features])
X_val_spca <- as.matrix(validation_set[, features])
X_test_spca <- as.matrix(testing_set[, features])

y_train_nn_spca <- training_set$target_return
y_val_nn_spca  <- validation_set$target_return
y_test_nn_spca <- testing_set$target_return
```

```
# Run SPCA for dimensionality reduction
spca_result <- suppressMessages(spca(X_train_spca,
                    K = 5,                        # Number of components
                    type = "predictor",
                    sparse = "penalty",
                    para = rep(0.4, 5),           # Sparsity level per component
                    trace = FALSE))
```

```
## You may wish to restart and use a more efficient way
## let the argument x be the sample covariance/correlation matrix and set type=Gram
```

```
print(spca_result$loadings)
```

```
##                                      PC1          PC2          PC3
## Advt_12M_Usd                0.1443918664  0.074803695  0.197313054
## Advt_3M_Usd                 0.1453647325  0.073585996  0.195172438
## Advt_6M_Usd                 0.1453854598  0.074049701  0.196310695
## Asset_Turnover              0.0070275493 -0.116467774  0.188712406
## Bb_Yld                      0.0627610363 -0.014908479  0.011521453
## Bv                          0.1414199638  0.132924597  0.097593411
## Capex_Ps_Cf                 0.0362685657  0.100756628  0.077701138
## Capex_Sales                -0.0048421910  0.057384712  0.029679002
## Cash_Div_Cf                 0.0487712420  0.067106725 -0.143976401
## Cash_Per_Share              0.1163549738 -0.065378501 -0.035186736
## Cf_Sales                    0.0861390004  0.007892096 -0.149661562
## Debtequity                  0.0188107336  0.177876012 -0.084763453
## Div_Yld                     0.0484760464  0.084699955 -0.139216332
## Dps                         0.0812289456  0.094046419 -0.128904151
## Ebit_Bv                     0.1102990626  0.001504903 -0.141987633
## Ebit_Noa                    0.1003472917 -0.081890134 -0.127323472
## Ebit_Oa                     0.1045652317 -0.084249721 -0.147197825
## Ebit_Ta                     0.1088235140 -0.070489933 -0.150307267
## Ebitda_Margin               0.0829720224  0.042283296 -0.213115789
## Eps                         0.1493786050  0.014800086 -0.076313400
## Eps_Basic                   0.1494401192  0.015135417 -0.075270599
## Eps_Basic_Gr                0.0554218635 -0.051140637 -0.009316707
## Eps_Contin_Oper             0.1455306168  0.014134570 -0.072644232
## Eps_Dil                     0.1498037171  0.016052318 -0.076747672
## Ev                          0.1638489244  0.130455993  0.103878568
## Ev_Ebitda                   0.0092733667 -0.033197012  0.109561945
## Fa_Ci                       0.0123101004  0.073048750  0.063640665
## Fcf                         0.1701309168 -0.010708859  0.025548296
## Fcf_Bv                      0.1252168584 -0.099764429 -0.040543902
## Fcf_Ce                      0.0996747152 -0.091146248 -0.066451082
## Fcf_Margin                  0.1085252969 -0.079271845 -0.134407612
## Fcf_Noa                     0.1110305016 -0.155732269 -0.011485390
## Fcf_Oa                      0.1122866028 -0.168314556  0.026548882
## Fcf_Ta                      0.1183780406 -0.162364489  0.022322940
## Fcf_Tbv                     0.0631456714 -0.072839351 -0.031133502
## Fcf_Toa                     0.0892001696 -0.134482432  0.014514732
## Fcf_Yld                     0.0863521825 -0.057524016 -0.093387031
## Free_Ps_Cf                  0.1163430329 -0.065367112 -0.035181874
## Int_Rev                     0.0199472236  0.015701003 -0.030351640
## Interest_Expense            0.0897631078  0.183329132  0.046416501
## Mkt_Cap_12M_Usd             0.1689828873  0.091612461  0.134221385
## Mkt_Cap_3M_Usd              0.1694629889  0.089082165  0.132917089
## Mkt_Cap_6M_Usd              0.1696287311  0.089882924  0.133534330
## Mom_11M_Usd                 0.0124077208 -0.018709112 -0.009759618
## Mom_5M_Usd                  0.0014306416 -0.013135485 -0.003804307
## Mom_Sharp_11M_Usd           0.0279070289 -0.007758183 -0.029488238
## Mom_Sharp_5M_Usd            0.0163428548 -0.004230579 -0.021344923
## Nd_Ebitda                  -0.0182194135  0.167234858 -0.042196440
## Net_Debt                    0.0599277994  0.196624599 -0.033873660
## Net_Debt_Cf                 0.0008057218  0.038063636 -0.022378451
## Net_Margin                  0.1218463339 -0.045279817 -0.175551854
```

```
## Netdebtyield                     0.0010257760  0.039847310 -0.023751457
## Ni                               0.1903191379  0.043086860  0.033358177
## Ni_Avail_Margin                  0.1171068234 -0.047827547 -0.165733748
## Ni_Oa                            0.1148311590 -0.161202078  0.045779400
## Ni_Toa                           0.0947407177 -0.130068901  0.021172509
## Noa                              0.1298282112  0.185640487  0.050344208
## Oa                               0.1322683489  0.184173942  0.006830147
## Ocf                              0.1818058490  0.077256196  0.085913672
## Ocf_Bv                           0.1295613833 -0.057242258  0.024689021
## Ocf_Ce                           0.0989876217 -0.086462335 -0.022435992
## Ocf_Margin                       0.0973819349 -0.012772864 -0.141838874
## Ocf_Noa                          0.1067277896 -0.162566006  0.061387673
## Ocf_Oa                           0.0999405882 -0.170090288  0.097646731
## Ocf_Ta                           0.1095853622 -0.157464218  0.092160535
## Ocf_Tbv                          0.0661418375 -0.054908577  0.004449299
## Ocf_Toa                          0.0815152849 -0.136552302  0.061356923
## Op_Margin                        0.1212653819 -0.021751076 -0.170652221
## Op_Prt_Margin                    0.0500281001 -0.039210232 -0.026410298
## Oper_Ps_Net_Cf                   0.1320979793  0.052969769 -0.033168664
## Pb                               0.0752847218 -0.085722653  0.080225112
## Pe                              -0.0420691410 -0.023374809  0.084593838
## Ptx_Mgn                          0.1242740054 -0.049024495 -0.172843239
## Recurring_Earning_Total_Assets   0.0628972520 -0.110639684  0.064662067
## Return_On_Capital                0.1270517092 -0.134797174  0.038832862
## Rev                              0.1371442347  0.100146070  0.170131734
## Roa                              0.1202931797 -0.151958768  0.044026168
## Roc                              0.1270491881 -0.134818268  0.038839858
## Roce                             0.1242094958 -0.037302762  0.033975139
## Roe                              0.1469162770 -0.089396216 -0.023272007
## Sales_Ps                         0.0512094861  0.062660717  0.082281001
## Share_Turn_12M                   0.0076245886 -0.004981105  0.223696895
## Share_Turn_3M                    0.0107027229 -0.003099436  0.219080626
## Share_Turn_6M                    0.0096663381 -0.003887778  0.222076348
## Ta                               0.1356201154  0.179073015  0.018054573
## Tev_Less_Mktcap                  0.0589988590  0.169754458 -0.025170318
## Tot_Debt_Rev                     0.0079835132  0.177302184 -0.144321024
## Total_Capital                    0.1375999780  0.178208180  0.069858056
## Total_Debt                       0.1020459739  0.207041611  0.012017399
## Total_Debt_Capital               0.0169966187  0.173009910 -0.079271603
## Total_Liabilities_Total_Assets   0.0290710506  0.147287785 -0.119841024
## Vol1Y_Usd                       -0.0922334444 -0.055806512  0.129612383
## Vol3Y_Usd                       -0.0896649093 -0.058425129  0.136127322
##                                           PC4           PC5
## Advt_12M_Usd                     0.0018691304 -0.1075240770
## Advt_3M_Usd                      0.0019894222 -0.1090588524
## Advt_6M_Usd                      0.0017220989 -0.1091094580
## Asset_Turnover                  -0.0807416651  0.2187236257
## Bb_Yld                          -0.0002789474  0.0337026682
## Bv                               0.0202656943 -0.0265724126
## Capex_Ps_Cf                     -0.1504790716  0.0794760220
## Capex_Sales                     -0.1382373371 -0.1949035650
## Cash_Div_Cf                     -0.0158322271 -0.0036905277
```

```
## Cash_Per_Share                  0.2149921544  0.1497583904
## Cf_Sales                        0.0080620151 -0.2712193648
## Debtequity                      0.0516171391  0.0177911583
## Div_Yld                        -0.0159854699  0.0364210823
## Dps                            -0.0361133113  0.0469543646
## Ebit_Bv                        -0.0105149719  0.0923948202
## Ebit_Noa                       -0.0189197880  0.0545740124
## Ebit_Oa                        -0.0322310103  0.0322370551
## Ebit_Ta                        -0.0407047109  0.0509473723
## Ebitda_Margin                   0.0152448235 -0.1746970826
## Eps                            -0.1736210643  0.1432612092
## Eps_Basic                      -0.1719276349  0.1435808081
## Eps_Basic_Gr                   -0.1127970135  0.0026341340
## Eps_Contin_Oper                -0.1689683870  0.1424406862
## Eps_Dil                        -0.1722364367  0.1437921663
## Ev                              0.0117561599 -0.0722101773
## Ev_Ebitda                      -0.0312213939 -0.1728267632
## Fa_Ci                          -0.1065157711  0.0485329914
## Fcf                             0.1881395494  0.0502074441
## Fcf_Bv                          0.2221580584  0.0670444506
## Fcf_Ce                          0.1737042823 -0.0591950693
## Fcf_Margin                      0.2289880593 -0.0842778978
## Fcf_Noa                         0.1813715515  0.0373557898
## Fcf_Oa                          0.1658283543  0.0098156226
## Fcf_Ta                          0.1690907610  0.0235883534
## Fcf_Tbv                         0.1026586887 -0.0009732250
## Fcf_Toa                         0.1069857367 -0.0528281909
## Fcf_Yld                         0.2747876640  0.1324715681
## Free_Ps_Cf                      0.2148773513  0.1497529888
## Int_Rev                         0.0560438032 -0.0974830242
## Interest_Expense                0.0479204637  0.0090661163
## Mkt_Cap_12M_Usd                -0.0071728585 -0.0793161560
## Mkt_Cap_3M_Usd                 -0.0064710607 -0.0817917793
## Mkt_Cap_6M_Usd                 -0.0070571824 -0.0814351244
## Mom_11M_Usd                     0.0053970310 -0.0142456394
## Mom_5M_Usd                      0.0152312193 -0.0033259588
## Mom_Sharp_11M_Usd              -0.0031392832 -0.0155609850
## Mom_Sharp_5M_Usd                0.0069114396 -0.0046753246
## Nd_Ebitda                       0.0208535625  0.0216474014
## Net_Debt                        0.0243749749  0.0432810815
## Net_Debt_Cf                    -0.1088958467 -0.0553526208
## Net_Margin                     -0.1262617936 -0.1849044247
## Netdebtyield                   -0.1145235696 -0.0609111179
## Ni                             -0.1082443889  0.0182093648
## Ni_Avail_Margin                -0.1244354489 -0.1696689571
## Ni_Oa                          -0.1763604106 -0.0165566137
## Ni_Toa                         -0.1289435368 -0.0571699759
## Noa                             0.0323470062 -0.0012094250
## Oa                              0.0529746455  0.0201052957
## Ocf                             0.0728056732 -0.0024649718
## Ocf_Bv                          0.0865001338  0.0468217863
## Ocf_Ce                          0.0820281274 -0.0906416273
```

```
## Ocf_Margin                       0.0966603698 -0.2411669222
## Ocf_Noa                          0.0538191123  0.0125392299
## Ocf_Oa                           0.0200600588 -0.0414280182
## Ocf_Ta                           0.0144942133 -0.0218961369
## Ocf_Tbv                          0.0355202489  0.0003532267
## Ocf_Toa                          0.0216164300 -0.0820527058
## Op_Margin                       -0.0878388276 -0.1799745444
## Op_Prt_Margin                   -0.1022933249  0.0067622669
## Oper_Ps_Net_Cf                   0.0533033668  0.1359404007
## Pb                              -0.0618417929 -0.1260436279
## Pe                               0.0871731110 -0.1857895071
## Ptx_Mgn                         -0.1260240997 -0.1775555055
## Recurring_Earning_Total_Assets -0.1092690072  0.1132502137
## Return_On_Capital               -0.1390604769  0.0734466306
## Rev                              0.0048295306  0.1416106428
## Roa                             -0.1900218518 -0.0067675813
## Roc                             -0.1391459537  0.0734708099
## Roce                            -0.0523463752  0.0283726998
## Roe                             -0.1680083028  0.0419337564
## Sales_Ps                        -0.0407757561  0.3379715488
## Share_Turn_12M                   0.0184418138 -0.1146002644
## Share_Turn_3M                    0.0158386235 -0.1135380134
## Share_Turn_6M                    0.0166336745 -0.1147678473
## Ta                               0.0571156405  0.0054631763
## Tev_Less_Mktcap                  0.0310823977  0.0126227771
## Tot_Debt_Rev                     0.0541416893 -0.1103541994
## Total_Capital                    0.0404375949 -0.0236776514
## Total_Debt                       0.0474198256  0.0121654649
## Total_Debt_Capital               0.0507363994  0.0189526800
## Total_Liabilities_Total_Assets  0.0806651577  0.0985842225
## Vol1Y_Usd                        0.0440568506 -0.0358280218
## Vol3Y_Usd                        0.0515387797 -0.0495200248
```

```
# loadings < 0.05 are considered close to zero and insignficant
percentage <- mean(spca_result$loadings <0.05)*100
cat('Percentage of close to zero loadings (< 0.05):', percentage, "\n")
```

```
## Percentage of close to zero loadings (< 0.05): 62.15054
```

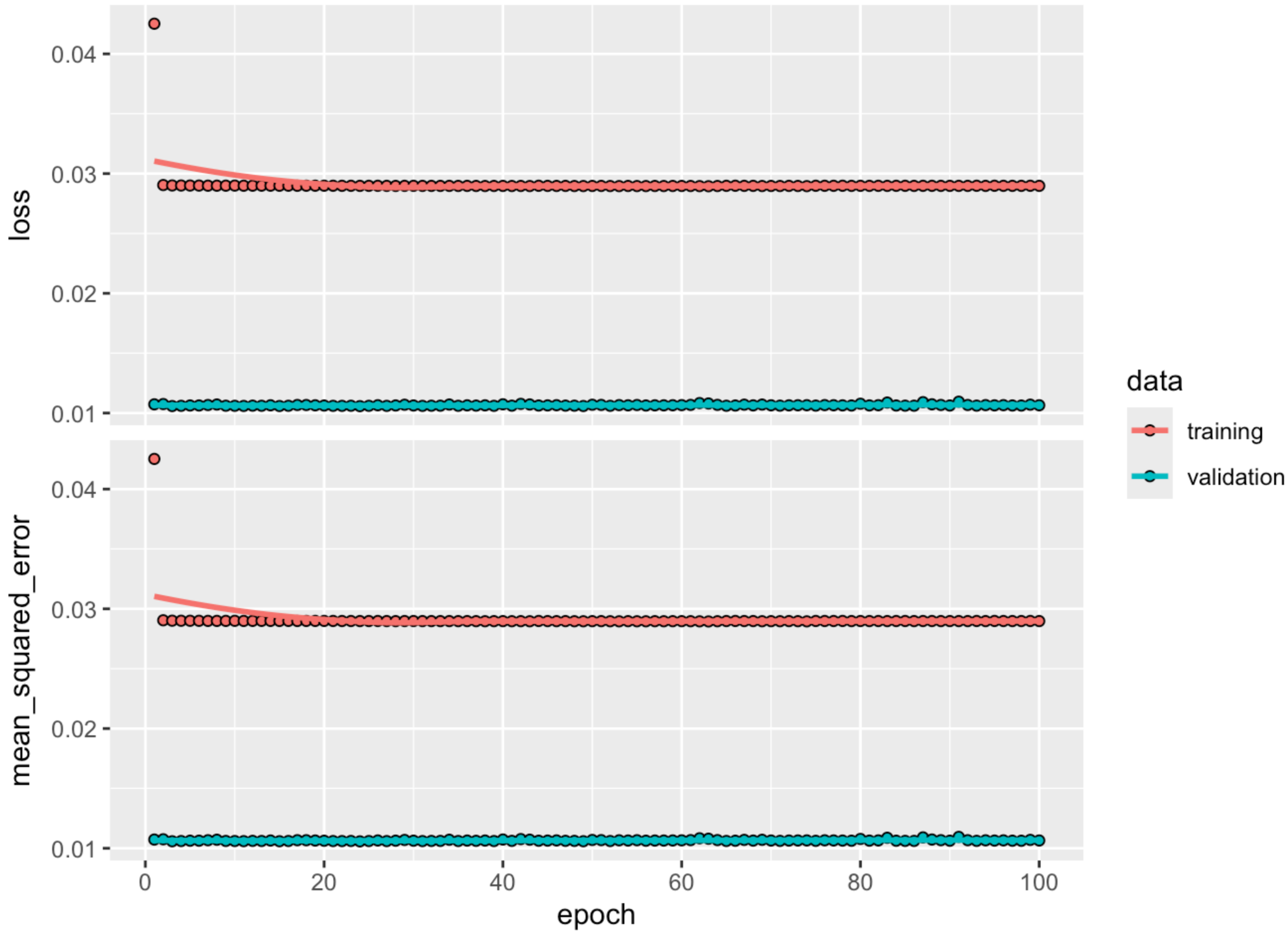
```
# SPCA transformed matrices for training, validation and test
Z_train_spca <- X_train_spca %*% spca_result$loadings
Z_val_spca   <- X_val_spca %*% spca_result$loadings
Z_test_spca  <- X_test_spca %*% spca_result$loadings
```

```r
# Define a grid of hyperparameters
units1_list <- c(16, 32, 64)
units2_list <- c(8, 16, 32)
learning_rates <- c(0.001, 0.0005)
batch_sizes <- c(64, 128)

results <- data.frame()

for (units1 in units1_list) {
  for (units2 in units2_list) {
    for (lr in learning_rates) {
      for (bs in batch_sizes) {

        # Build model
        model <- keras_model_sequential() %>%
          layer_dense(units = units1, activation = "relu", input_shape = ncol(Z_train_sp
ca)) %>%
          layer_dense(units = units2, activation = "relu") %>%
          layer_dense(units = 1)

        # Compile
        model %>% compile(
          loss = "mse",
          optimizer = optimizer_adam(learning_rate = lr),
          metrics = list("mean_squared_error")
        )

        # Train
        history <- model %>% fit(
          x = Z_train_spca,
          y = y_train_nn_spca,
          epochs = 100,
          batch_size = bs,
          validation_data = list(Z_val_spca, y_val_nn_spca),
          verbose = 0
        )

        # Get final validation MSE
        val_mse <- tail(history$metrics$val_mean_squared_error, 1)

        # Record result
        results <- rbind(results, data.frame(
          units1 = units1,
          units2 = units2,
          learning_rate = lr,
          batch_size = bs,
          val_mse = val_mse
        ))
      }
    }
  }
}
```

```
# Find the best hyperparameters
best_result <- results[which.min(results$val_mse), ]
print(best_result)
```

```
##    units1 units2 learning_rate batch_size    val_mse
## 13     32      8         0.001         64 0.01060999
```

```
# Train NN on SPCA components
model_spca <- keras_model_sequential() %>%
  layer_dense(units = best_result$units1,
              activation = "relu", input_shape = ncol(Z_train_spca)) %>%
  layer_dense(units = best_result$units2, activation = "relu") %>%
  layer_dense(units = 1)  # output layer for regression


# Compile model
model_spca %>% compile(
  loss = "mse",
  optimizer = optimizer_adam(learning_rate = best_result$learning_rate),
  metrics = list("mean_squared_error")
)

history_spca <- model_spca %>% fit(
  x = Z_train_spca,
  y = y_train_nn_spca,
  epochs = 100,
  batch_size = best_result$batch_size,
  validation_data = list(Z_val_spca, y_val_nn_spca),
  verbose = 0  # quiet training
)

plot(history_spca)
```




```
# Predict on test set
pred_spca_train <- model_spca %>% predict(Z_train_spca)
```

```
## 6192/6192 - 1s - 918ms/epoch - 148us/step
```

```
pred_spca_test <- model_spca %>% predict(Z_test_spca)
```

```
## 853/853 - 0s - 131ms/epoch - 154us/step
```

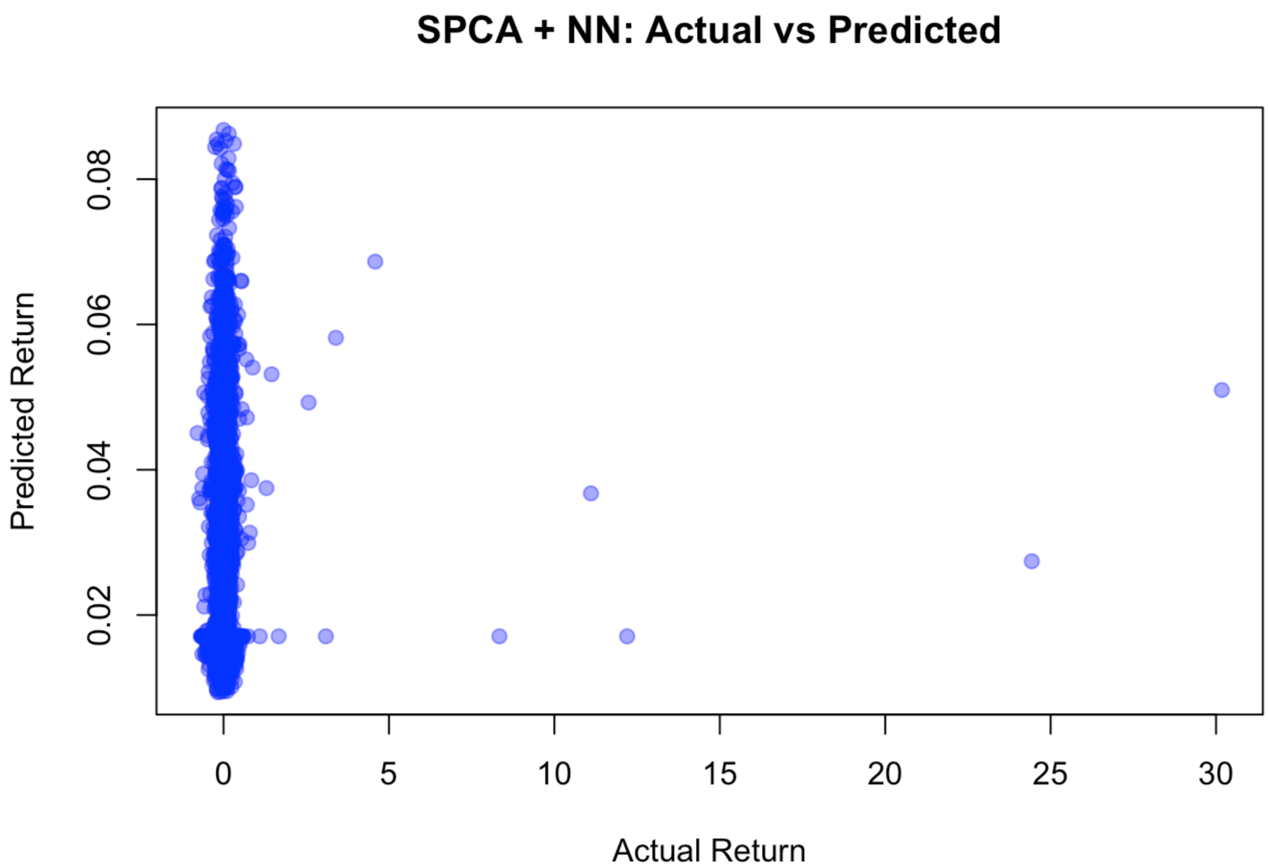
```
# Calculate RMSE
rmse <- function(actual, predicted) {
  sqrt(mean((actual - predicted)^2))
}

cat("SPCA + NN Train RMSE:", rmse(y_train_nn_spca, pred_spca_train), "\n")
```

```
## SPCA + NN Train RMSE: 0.1702056
```

```
cat("SPCA + NN Test RMSE:", rmse(y_test_nn_spca, pred_spca_test), "\n")
```

```
## SPCA + NN Test RMSE: 0.2802199
```

```
plot(y_test_nn_spca, pred_spca_test,
     xlab = "Actual Return",
     ylab = "Predicted Return",
     main = "SPCA + NN: Actual vs Predicted",
     #xlim = c(-0.05, 0.05),   # Zoom on X-axis: Actual returns between -10% and +10%
     # ylim = c(-0.1, 0.1),
     pch = 19, col = rgb(0, 0, 1, 0.4))
```



```
# Add predictions back to test set (already done if continuing)
testing_set_lasso <- testing_set %>%
  mutate(pred_return = as.numeric(pred_nn_lasso_test),
         model = "LASSO_NN")

testing_set_spca <- testing_set %>%
  mutate(pred_return = as.numeric(pred_spca_test),
         model = "SPCA_NN")

# Combine both models
portfolio_data <- bind_rows(testing_set_lasso, testing_set_spca)

# Rank stocks within each date and model
portfolio_data <- portfolio_data %>%
  group_by(model, date) %>%
  mutate(rank = percent_rank(pred_return)) %>%
  ungroup()

# Filter top 20% (long-only portfolio)
top_20_data <- portfolio_data %>%
  filter(rank >= 0.8)

# Calculate average monthly return
long_only_returns <- top_20_data %>%
  group_by(model, date) %>%
  summarise(top20_return = mean(target_return), .groups = "drop")

# Performance summary
long_only_summary <- long_only_returns %>%
  group_by(model) %>%
  summarise(
    avg_monthly_return = mean(top20_return, na.rm = TRUE),
    holding_period_return = prod(1+top20_return)-1,
    sd_monthly_return = sd(top20_return, na.rm = TRUE),
    sharpe_ratio = avg_monthly_return / sd_monthly_return,
    .groups = "drop"
  )

print(long_only_summary)
```

```
## # A tibble: 2 × 5
##   model  avg_monthly_return holding_period_return sd_monthly_return sharpe_ratio
##   <chr>               <dbl>                 <dbl>             <dbl>        <dbl>
## 1 LASSO…             0.0165                 0.414            0.0650        0.254
## 2 SPCA_…             0.0357                 1.05             0.109         0.326
```

```r
# Calculate
sp500_avg_return <- mean(sp500_monthly_returns$sp500_return, na.rm = TRUE)
sp500_sd_return <- sd(sp500_monthly_returns$sp500_return, na.rm = TRUE)
sp500_holding_period_return <- prod(1+sp500_monthly_returns$sp500_return) -1
sp500_sharpe <- sp500_avg_return / sp500_sd_return

# Show result
tibble(
  model = "SP500",
  avg_monthly_return = sp500_avg_return,
  holding_period_return = sp500_holding_period_return,
  sd_monthly_return = sp500_sd_return,
  sharpe_ratio = sp500_sharpe
)
```

```
## # A tibble: 1 × 5
##   model avg_monthly_return holding_period_return sd_monthly_return sharpe_ratio
##   <chr>              <dbl>                 <dbl>             <dbl>        <dbl>
## 1 SP500            0.00458                 0.101            0.0339        0.135
```

```
# cumulative returns for three strategies
long_only_returns_lasso <- long_only_returns[long_only_returns[,'model'] == 'LASSO_NN',]
long_only_returns_spca <- long_only_returns[long_only_returns[,'model'] == 'SPCA_NN',]

# Add cumulative returns
monthly_portfolio_return_lasso <- long_only_returns_lasso %>%
  mutate(cumulative_return_lasso = cumprod(1 + top20_return))

monthly_portfolio_return_spca <- long_only_returns_spca %>%
  mutate(cumulative_return_spca = cumprod(1 + top20_return))

sp500_monthly_returns <- sp500_monthly_returns %>%
  mutate(cumulative_return_sp500 = cumprod(1 + sp500_return))

# mutate sp500 date column to make sure it aligns with the date from lasso and spca
# Replace sp500 date with spca portfolio date
sp500_monthly_returns <- sp500_monthly_returns %>%
  mutate(date = long_only_returns_lasso$date)

compare_cumulative_returns <- monthly_portfolio_return_lasso %>%
  select(date, cumulative_return_lasso) %>%
  left_join(monthly_portfolio_return_spca %>% select(date, cumulative_return_spca), by =
"date") %>%
  left_join(sp500_monthly_returns %>% select(date, cumulative_return_sp500), by = "dat
e")


compare_cumulative_returns_long <- compare_cumulative_returns %>%
  pivot_longer(
    cols = starts_with("cumulative_return"),
    names_to = "model",
    values_to = "cumulative_return"
  )

ggplot(compare_cumulative_returns_long, aes(x = date, y = cumulative_return, color = mod
el)) +
  geom_line(size = 1.2) +
  labs(title = "Cumulative Returns: LASSO_NN vs SPCA_NN vs SP500",
       x = "Date",
       y = "Cumulative Return (Index Level)",
       color = "Strategy") +
  theme_minimal() +
  scale_color_manual(values = c("cumulative_return_lasso" = "red",
                                "cumulative_return_spca" = "blue",
                                "cumulative_return_sp500" = "black")) +
  theme(plot.title = element_text(hjust = 0.5))
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## ℹ Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```