in— title: “Neural_networks” author: “Yinuo Jin” output: pdf_document: default html_document: default —

### Load required library & dataset

```
library(torch)
library(hash)
```

```
## hash-2.2.6.1 provided by Decision Patterns
```

```
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
library(ggplot2)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

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

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

```
data_path <- "../data/"
```

```
data <- read.csv(paste0(data_path, "processed_counts.csv"))
# head(data)
```

```
label <- read.csv(paste0(data_path, "annotation.csv"))
# head(label)
```

### Feed-forward Neural Networks with 2 hidden layer (built with Torch)

```
# Map str labels to integers
encoding <- function(labels) {
  dict <- hash()
  unique_labels <- sort(unique(labels))
  for (i in 1:length(unique_labels)){
    l <- unique_labels[i]
    dict[[l]] <- i
  }

  # if one-hot encoding
  # encoded <- matrix(data = 0, nrow = length(labels), ncol = length(unique_labels))
  #
  # for (i in seq(1, length(labels))) {
  #   encoded[i, dict[[labels[i]]]] <- 1
  #
  # return (encoded)

  return (as.integer(lapply(labels, function(x) dict[[x]])))
```

```
}

label_binary <- ifelse(label$Type != "Normal", 1, 0) # binary classification; 0 - normal; 1 - cancerous
label_multi <- encoding(label$Type) # multi-label classification
```

### Train-test split

```
set.seed(6690)

train_idxs <- sample(seq(1:nrow(label)), size = round(nrow(label)) * 0.75)
total_train <- data[train_idxs, ]
total_test <- data[-train_idxs, ]

x_total <- torch_tensor(as.matrix(data[-1]), dtype = torch_float())
y_total_binary <- torch_tensor(as.double(as.matrix(label_multi)), dtype = torch_float())
y_total_multi <- torch_tensor(as.double(as.matrix(label_multi)), dtype = torch_long())

x_train <- torch_tensor(as.matrix(total_train[, -1]), dtype = torch_float())
x_test <- torch_tensor(as.matrix(total_test[, -1]), dtype = torch_float())

y_train_binary <- torch_tensor(as.double(as.matrix(label_binary[train_idxs])), dtype = torch_float())
y_test_binary <- torch_tensor(as.double(as.matrix(label_binary[-train_idxs])), dtype = torch_float())
y_train_multi <- torch_tensor(as.double(as.matrix(label_multi[train_idxs])), dtype = torch_long())
y_test_multi <- torch_tensor(as.double(as.matrix(label_multi[-train_idxs])), dtype = torch_long())
```

### Model

Vanilla model (flexibly with specified input & output channels)

```
# Reference:
# https://anderfernandez.com/en/blog/how-to-create-neural-networks-with-torch-in-r/
net <- nn_module(
  initialize = function(c_in, c_out, p = 0.2) {
    self$c_out <- c_out
    self$layer1 <- nn_sequential(
      nn_linear(c_in, 128),
      nn_relu(inplace = TRUE),
      nn_dropout(p = p)
    )

    self$layer2 <- nn_sequential(
      nn_linear(128, 64),
      nn_relu(inplace=TRUE),
      nn_dropout(p = p)
    )

    self$layer3 <- nn_sequential(
      nn_linear(64, c_out),
    )
  },

  forward = function(x) {
    if (self$c_out == 1) {
```

```
      x %>%
        self$layer1() %>%
        self$layer2() %>%
        self$layer3() %>%
        torch_flatten()
    } else{
      x %>%
        self$layer1() %>%
        self$layer2() %>%
        self$layer3() %>%
        nnf_softmax(dim = -1)
    }
  }
)
```

Sequential Model (fixed input & output features in declaration; simple for DeepLIFT explanation)

```
c_in <- ncol(x_train)
c_out <- length(unique(label_multi))

# The following module has reference & is taken from:
# https://cran.r-project.org/web/packages/innsight/vignettes/innsight.html
nn_flatten <- nn_module(
  classname = "nn_flatten",
  initialize = function(start_dim = 1, end_dim = -1) {
    self$start_dim <- start_dim
    self$end_dim <- end_dim
  },
  forward = function(x) {
    torch_flatten(x, start_dim = self$start_dim, end_dim = self$end_dim)
  }
)

# Sequential model for binary classification
SeqNet.bin <- nn_sequential(
  nn_linear(c_in, 128),
  nn_relu(inplace = TRUE),
  nn_dropout(0.2),

  nn_linear(128, 64),
  nn_relu(inplace = TRUE),
  nn_dropout(0.2),

  nn_linear(64, 1),
  nn_flatten(),
)

# Sequential model for pan-cancer classification
SeqNet.pan <- nn_sequential(
  nn_linear(c_in, 256),
  nn_relu(inplace = TRUE),
  nn_dropout(0.2),

  nn_linear(256, 128),
```

```
  nn_relu(inplace = TRUE),
  nn_dropout(0.2),

  nn_linear(128, 64),
  nn_relu(inplace = TRUE),
  nn_dropout(0.2),

  nn_linear(64, c_out),
  nn_softmax(dim = -1)
)
```

```
# Wrapper for model training
calcWeights <- function(y_train) {
  freq <- table(as.integer(y_train))
  weight <- sum(freq) / freq
  return (weight / min(weight))
}


# Calculate Matthews Correlation Coefficients (MCC)
calcMCC <- function(cm, is_binary = TRUE, eps = 1e-5) {
  if (is_binary) {
    tn <- cm[1, 1]
    fn <- cm[1, 2]
    fp <- cm[2, 1]
    tp <- cm[2, 2]

    d <-sqrt(tp+fp) * sqrt(tp+fn) * sqrt(tn+fp) * sqrt(tn+fn) + eps
    mcc <- (tp * tn - fp * fn) / d
  } else {
    t <- rowSums(cm)  # tot. samples in each category
    p <- diag(cm)     # tot. correct predicted. samples in each category
    s <- sum(cm)      # tot. samples in data
    c <- sum(p)       # tot. corrected samples

    n <- c * s - sum(p * t)
    d <- sqrt( (s^2  - sum(p^2)) * (s^2 - sum(t^2)) ) + eps
    mcc <- n / d
  }

  return (mcc)
}


trainModel <- function(x_train, y_train, c_in, epochs = 5, lr = 0.01, drop_rate = 0.2) {
  n_unique <- length(unique(as.integer(y_train)))
  c_out <- ifelse(n_unique == 2, 1, n_unique)
  print(paste("Output dimension:", c_out))

  model <- net(c_in, c_out, drop_rate)
  losses <- c()
  weights <- calcWeights(y_train)

  if (c_out == 1) {
```

```
    criterion <- nn_bce_with_logits_loss()
  } else {
    criterion <- nn_cross_entropy_loss(weight = weights)
  }

  optimizer <- optim_adam(model$parameters, lr = lr)

  for (i in 1:epochs) {
    optimizer$zero_grad()

    y_pred <- model(x_train)
    loss = criterion(y_pred, y_train)
    loss$backward()
    optimizer$step()

    # training-log
    if (i %% 10 == 0) {
      if (c_out == 1) {
        y_pred_binary <- ifelse(y_pred > 0.5, 1, 0)
        acc <- (y_pred_binary == y_train)$sum()$item() / y_train$size()
      }
      else {
        y_pred_cat <- y_pred %>% torch_argmax(dim = 2)
        acc <- (y_pred_cat == y_train)$sum()$item() / y_train$size()
      }
      print(paste0("Epoch=", i, " Loss=", loss$item(), " Acc=", acc))
      losses <- append(losses, loss$item())
    }
  }

  return (list(model = model, losses = losses))
}

trainSeqModel <- function(x_train, y_train, model, epochs = 5, lr = 0.01, is_binary = TRUE) {
  losses <- c()

  if (is_binary) {
    criterion <- nn_bce_with_logits_loss()
  } else{
    weights <- calcWeights(y_train)
    criterion <- nn_cross_entropy_loss(weight = weights)
  }

  optimizer <- optim_adam(model$parameters, lr = lr)

  for (i in 1:epochs) {
    optimizer$zero_grad()
    y_pred <- model(x_train)

    loss <- criterion(y_pred, y_train)
    loss$backward()
    optimizer$step()
```

```
    if (i %% 10 == 0) {
      if (is_binary) {
        y_pred_binary <- ifelse(y_pred > 0.5, 1, 0)
        acc <- (y_pred_binary == y_train)$sum()$item() / y_train$size()
      } else {
        y_pred_cat <- y_pred %>% torch_argmax(dim = 2)
        acc <- (y_pred_cat == y_train)$sum()$item() / y_train$size()
      }
      print(paste0("Epoch=", i, " Loss=", loss$item(), " Acc=", acc))
      losses <- append(losses, loss$item())
    }
  }

  return (list(model = model, losses = losses))
}
```

Helper functions for visualization

```
# Print confusion matrix
printConfusionMat <- function(y_true, y_pred, name, perc = FALSE) {
  true <- as.integer(y_true)
  pred <- as.integer(y_pred)
  mat <- table(factor(pred, levels=min(true):max(true)),
               factor(true, levels=min(true):max(true)))

  if (perc) {
    mat <- round(sweep(mat, 2, rowSums(mat), FUN = '/'), digits = 2)
  }
  row.names(mat) <- name
  colnames(mat) <- name

  return (mat)
}

#' Visualize confusion matrix
#' Reference:
#' https://stackoverflow.com/questions/37897252/plot-confusion-matrix-in-r-using-ggplot
#'
#' @y_true torch.Tensor ground-truth labels
#' @y_pred predictions
#' @name character of label names
#' @perc whether to show matrix in perentage
#' @title figure title
visualizeConfusionMat <- function(y_true, y_pred, name, perc = FALSE, savefig = TRUE,
                                  title = NULL) {
  y.true <- as.integer(y_true)
  y.pred <- as.integer(y_pred)

  cm <- confusionMatrix(factor(y.pred), factor(y.true), dnn = c("prediction", "reference"))

  if (perc) {
    plt <- as.data.frame(
      round(sweep(cm$table, 2, rowSums(cm$table), FUN = '/'), digits = 2)
    )
```

```
  } else {
    plt <- as.data.frame(cm$table)
  }

  plt$prediction <- factor(plt$prediction, levels=rev(levels(plt$prediction)))

  q <- ggplot(plt, aes(reference, prediction, fill= Freq)) +
        geom_tile(aes(fill = Freq), colour = "black") +
        geom_text(aes(label=Freq)) +
        scale_fill_gradient(low="white", high="purple") +
        labs(x = "Reference",y = "Prediction") +
        scale_x_discrete(labels = name) +
        scale_y_discrete(labels = rev(name)) +
        ggtitle(title) +
        theme(plot.title = element_text(hjust = 0.5),
              axis.text.x = element_text(angle = 45, vjust = 0.5, hjust = 0.5))

  return (q)
}
```

```
res_bin <- trainSeqModel(
  x_train = x_train,
  y_train = y_train_binary,
  model = SeqNet.bin,
  epochs = 50,
  lr = 0.001
)
```

### (1). Binary prediction (tumor vs. normal)

```
## [1] "Epoch=10 Loss=0.248016342520714 Acc=0.922862129144852"
## [1] "Epoch=20 Loss=0.134962633252144 Acc=0.951134380453752"
## [1] "Epoch=30 Loss=0.098906397819519 Acc=0.962303664921466"
## [1] "Epoch=40 Loss=0.0731233581900597 Acc=0.972774869109948"
## [1] "Epoch=50 Loss=0.0586078464984894 Acc=0.979581151832461"
```

```
y_pred <- ifelse(res_bin$model(x_test) > 0.5, 1, 0)
acc_test <- (y_pred == y_test_binary)$sum()$item() / y_test_binary$size()
cat(paste("Acc:", acc_test))
```

### Prediction on test set

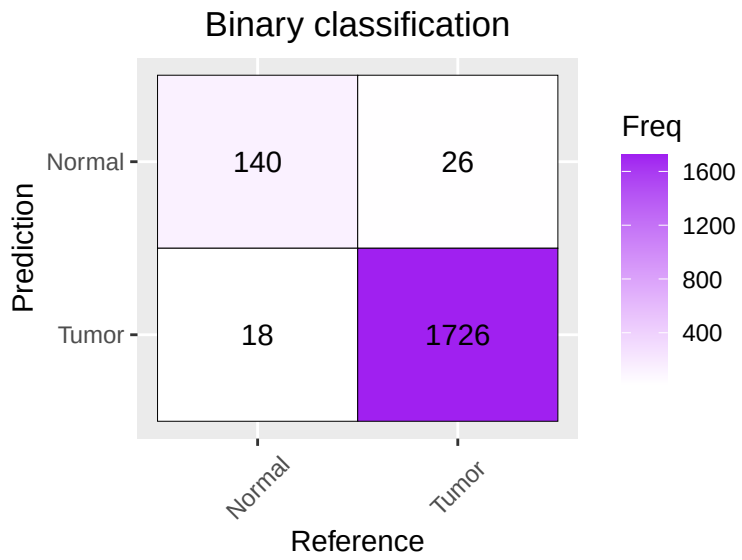
```
## Acc: 0.97696335078534
```

```
mat <- printConfusionMat(y_test_binary, y_pred, name = c("Normal", "Cancer"))
mat.perc <- printConfusionMat(y_test_binary, y_pred, name = c("Normal", "Cancer"), perc = TRUE)
mcc <- calcMCC(mat)
cat(paste("Mcc:", mcc))
```

### Confusion matrix

```
## Mcc: 0.851930185127783
```

```
q.bin <- visualizeConfusionMat(y_test_binary, y_pred,
                               name = c("Normal", "Tumor"),
                               title = "Binary classification")
q.bin
```




```
ggsave("../slides+report/nn_cm_bin.png", q.bin, width = 4, height = 3.5)
```

```
res_pan <- trainSeqModel(
  x_train = x_train,
  y_train = y_train_multi,
  model = SeqNet.pan,
  epochs = 400,
  lr = 0.001,
  is_binary = FALSE
)
```

**(2). Multi-label classification (pan-cancer)**

```
## [1] "Epoch=10 Loss=2.58004236221313 Acc=0.285514834205934"
## [1] "Epoch=20 Loss=2.36339235305786 Acc=0.461780104712042"
## [1] "Epoch=30 Loss=2.29297947883606 Acc=0.531064572425829"
## [1] "Epoch=40 Loss=2.20865511894226 Acc=0.617975567190227"
## [1] "Epoch=50 Loss=2.14625644683838 Acc=0.745026178010471"
## [1] "Epoch=60 Loss=2.09948205947876 Acc=0.790750436300174"
## [1] "Epoch=70 Loss=2.04787802696228 Acc=0.843455497382199"
## [1] "Epoch=80 Loss=2.02550029754639 Acc=0.86457242582897"
## [1] "Epoch=90 Loss=2.0058605670929 Acc=0.887085514834206"
## [1] "Epoch=100 Loss=1.99514067173004 Acc=0.894764397905759"
## [1] "Epoch=110 Loss=1.97311317920685 Acc=0.930191972076789"
## [1] "Epoch=120 Loss=1.94334053993225 Acc=0.9478184991274"
## [1] "Epoch=130 Loss=1.94403231143951 Acc=0.93630017452007"
## [1] "Epoch=140 Loss=1.92279827594757 Acc=0.964048865619546"
## [1] "Epoch=150 Loss=1.91823816299438 Acc=0.962303664921466"
## [1] "Epoch=160 Loss=1.91662001609802 Acc=0.962652705061082"
## [1] "Epoch=170 Loss=1.90990722179413 Acc=0.969808027923211"
```

```
## [1] "Epoch=180 Loss=1.91464555263519 Acc=0.963176265270506"
## [1] "Epoch=190 Loss=1.94113969802856 Acc=0.963525305410122"
## [1] "Epoch=200 Loss=1.92432141304016 Acc=0.971727748691099"
## [1] "Epoch=210 Loss=1.90514945983887 Acc=0.977137870855148"
## [1] "Epoch=220 Loss=1.90092039108276 Acc=0.978184991273996"
## [1] "Epoch=230 Loss=1.89945733547211 Acc=0.980104712041885"
## [1] "Epoch=240 Loss=1.89624607563019 Acc=0.982198952879581"
## [1] "Epoch=250 Loss=1.89731967449188 Acc=0.981675392670157"
## [1] "Epoch=260 Loss=1.89422380924225 Acc=0.983420593368237"
## [1] "Epoch=270 Loss=1.893195271492 Acc=0.983944153577661"
## [1] "Epoch=280 Loss=1.91651797294617 Acc=0.959685863874346"
## [1] "Epoch=290 Loss=1.90890038013458 Acc=0.967015706806283"
## [1] "Epoch=300 Loss=1.89795815944672 Acc=0.978359511343804"
## [1] "Epoch=310 Loss=1.89706087112427 Acc=0.984816753926702"
## [1] "Epoch=320 Loss=1.89841258525848 Acc=0.978184991273996"
## [1] "Epoch=330 Loss=1.91331613063812 Acc=0.979232111692845"
## [1] "Epoch=340 Loss=1.89357793331146 Acc=0.983071553228621"
## [1] "Epoch=350 Loss=1.89145517349243 Acc=0.984816753926702"
## [1] "Epoch=360 Loss=1.89079201221466 Acc=0.98586387434555"
## [1] "Epoch=370 Loss=1.9018212556839 Acc=0.983595113438045"
## [1] "Epoch=380 Loss=1.88897716999054 Acc=0.98760907504363"
## [1] "Epoch=390 Loss=1.88828361034393 Acc=0.988830715532286"
## [1] "Epoch=400 Loss=1.88779354095459 Acc=0.988830715532286"
```

```
y_pred <- res_pan$model(x_test) %>% torch_argmax(dim = 2)
acc <- (y_pred == y_test_multi)$sum()$item() / y_test_multi$size()
acc
```

### Prediction on test set

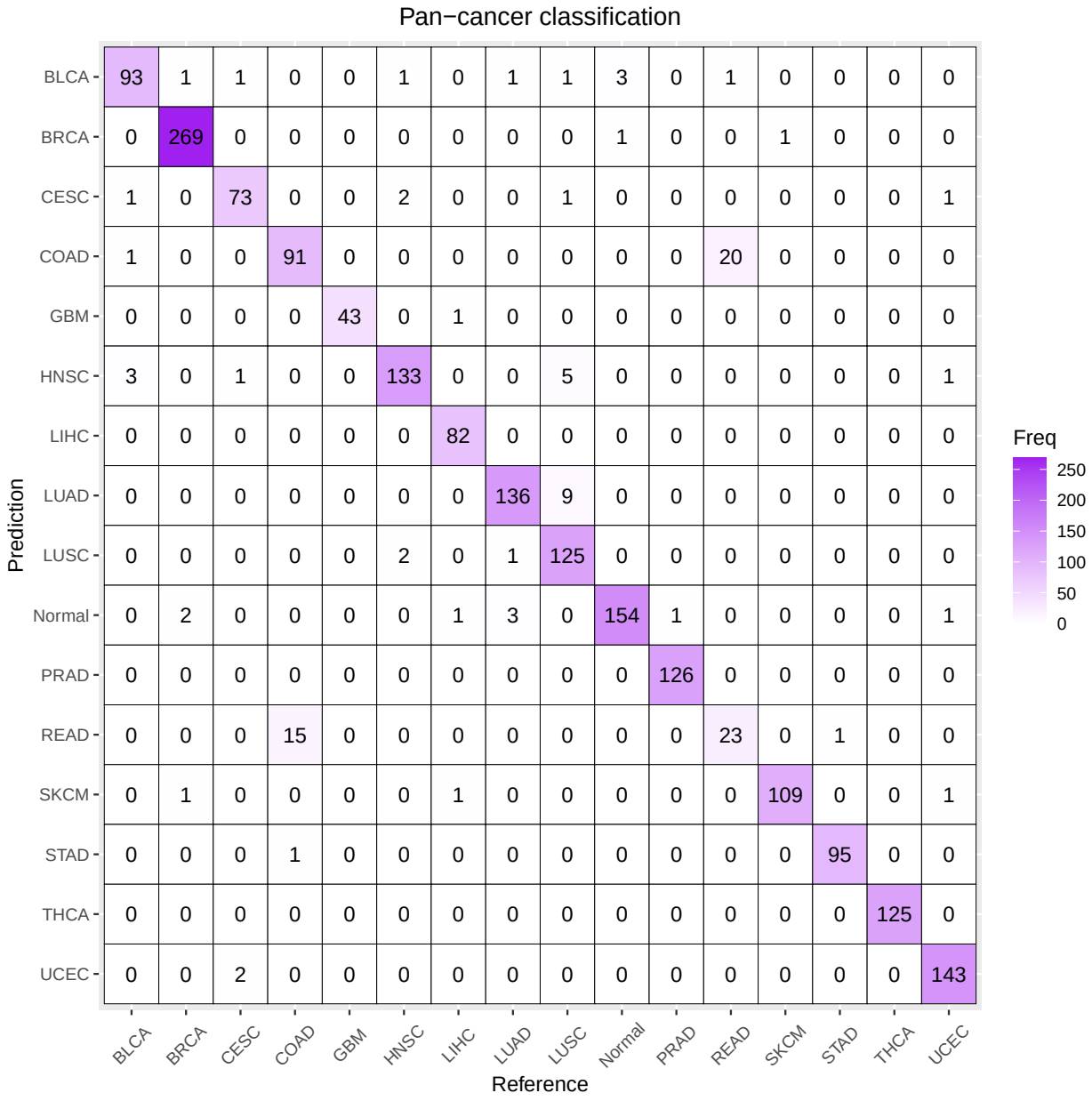
```
## [1] 0.9528796
```

```
mat <- printConfusionMat(y_test_multi, y_pred, name = sort(unique(label$Type)))
mat.perc <- printConfusionMat(y_test_multi, y_pred, name = sort(unique(label$Type)), perc = TRUE)
mcc <- calcMCC(mat, is_binary = FALSE)
mcc
```

### Confusion matrix

```
## [1] 0.9494026
```

```
q.pan <- visualizeConfusionMat(y_test_multi, y_pred,
                           name = sort(unique(label$Type)),
                           title = "Pan-cancer classification")
q.pan
```




```
ggsave("../slides+report/nn_cm_multi.png", q.pan, width = 8, height = 8)
```

### Save model

```
torch_save(res_bin$model$state_dict(), path = "../models/nn_bin.pt")
torch_save(res_pan$model$state_dict(), path = "../models/nn_multi.pt")
```

### Model explanation

```
library(innsight)
library(gridExtra)
```

```
##
```

```
## Attaching package: 'gridExtra'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
conv.pan <- Converter$new(res_pan$model, input_dim = c_in,
                          input_names = list(colnames(data)[-1]),
                          output_names = list(sort(unique(label$Type)))
)
```

**Extract pan-classification network features**

```
## Skipping nn_dropout ...
## Skipping nn_dropout ...
## Skipping nn_dropout ...
```

```
dl.pan <- DeepLift$new(conv.pan, x_train)
```

```
## Backward pass 'DeepLift':
```

```
##   |                                                                      |
```

```
dl_res.arr <- dl.pan$get_result()
```

For each gene, select top 20 genes for such prediction

```
pred_labels <- as.integer(res_pan$model(x_train) %>% torch_argmax(dim = 2))
unique_labels <- sort(unique(label$Type))

features <- list()
feature_idxs <- list()

for (i in seq(1:16)) {
  # Empty prediction in "READ" - skip
  if (unique_labels[i] == "READ") {
    next
  }

  print(paste0("Extracting features for ", unique_labels[i], "..."))
  idxs <- which(pred_labels == i)
  curr_dl <- dl_res.arr[idxs, , ]
  curr_dl.avg <- apply(curr_dl, c(2,3), mean)
  names(curr_dl.avg) <- unique_labels

  curr_feature <- curr_dl.avg[,i]
  features[[unique_labels[i]]] <- sort(curr_feature, decreasing = TRUE)[1:20]
  feature_idxs[[unique_labels[i]]] <- order(curr_feature, decreasing = TRUE)[1:20]
}
```

```
## [1] "Extracting features for BLCA..."
## [1] "Extracting features for BRCA..."
## [1] "Extracting features for CESC..."
## [1] "Extracting features for COAD..."
## [1] "Extracting features for GBM..."
## [1] "Extracting features for HNSC..."
## [1] "Extracting features for LIHC..."
```

```
## [1] "Extracting features for LUAD..."
## [1] "Extracting features for LUSC..."
## [1] "Extracting features for Normal..."
## [1] "Extracting features for PRAD..."
## [1] "Extracting features for SKCM..."
## [1] "Extracting features for STAD..."
## [1] "Extracting features for THCA..."
## [1] "Extracting features for UCEC..."
```

```
displayFeatures <- function(features, lbls) {
  q_list <- list()

  for (i in seq(1:length(lbls))) {
    lbl <- lbls[i]

    # Empty prediction for "READ"
    if (lbl == "READ") {
      next
    }

    df_lbl <- as.data.frame(features[[lbl]])
    df_lbl <- cbind(Gene = row.names(df_lbl), df_lbl)
    rownames(df_lbl) <- 1:nrow(df_lbl)
    names(df_lbl)[2] <- "DeepLIFT"

    q <- ggplot(data = df_lbl, aes_string(x = "Gene", y = "DeepLIFT", fill = "DeepLIFT")) +
                geom_bar(stat = "identity") +
                scale_fill_gradient( low="white", high="darkred", limits = c(0, max(df_lbl[["DeepLIFT"]]
                ylim(c(0, 0.9)) +
                ggtitle(lbl) +
                theme(plot.title = element_text(hjust = 0.5),
                axis.text.x = element_text(angle = 45, vjust = 0.5, hjust = 0.5))

    q_list[[lbl]] <- q
  }

  return (q_list)
}
```

```
q_list <- displayFeatures(features, unique_labels)
do.call("grid.arrange", c(q_list, nrow = 5, ncol = 3))
```


## Barplots

```
q.summary <- do.call("arrangeGrob", c(q_list, nrow = 5, ncol = 3))
ggsave("../slides+report/dl_summmary_v2.png", q.summary, width = 20, height = 15)
```

Comparison of BRCA "feature" genes, random genes in BRCA & importance of "feature" genes in other Tumor expression

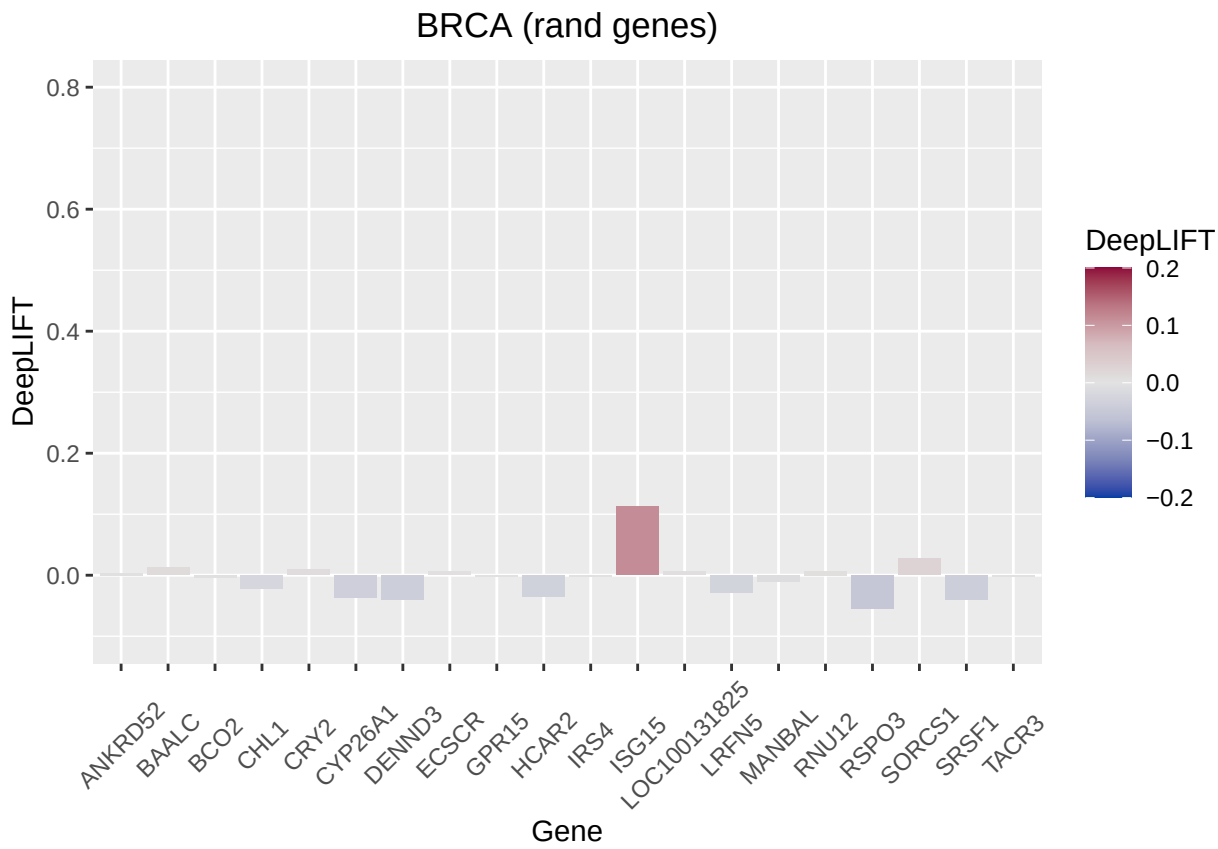
```
# Helper function: set  index of dataframe back as the 1st col.
reIndex <- function(df) {
    df <- cbind(Gene = row.names(df), df)
    rownames(df) <- 1:nrow(df)
    names(df)[2] <- "DeepLIFT"
    return (df)
}

brca_idxs <- which(pred_labels == 2)
dl_brca <- dl_res.arr[brca_idxs, , ]
dl_brca.avg <- apply(dl_brca, c(2,3), mean)
names(dl_brca.avg) <- unique_labels

brca_features <- dl_brca.avg[, 2]
df_brca_top_features <- as.data.frame(
  sort(brca_features, decreasing = TRUE)[1:20]
)
df_brca_top_features <- reIndex(df_brca_top_features)
```

```
brca_top_idxs <- order(brca_features, decreasing = TRUE)[1:20]

rand_idxs <- sample(1:length(brca_features), 20, replace = FALSE)
df_brca_rand_features <- as.data.frame(
  brca_features[rand_idxs]
)
df_brca_rand_features <- reIndex(df_brca_rand_features)
```

BRCA "feature gene" importance in Normal samples

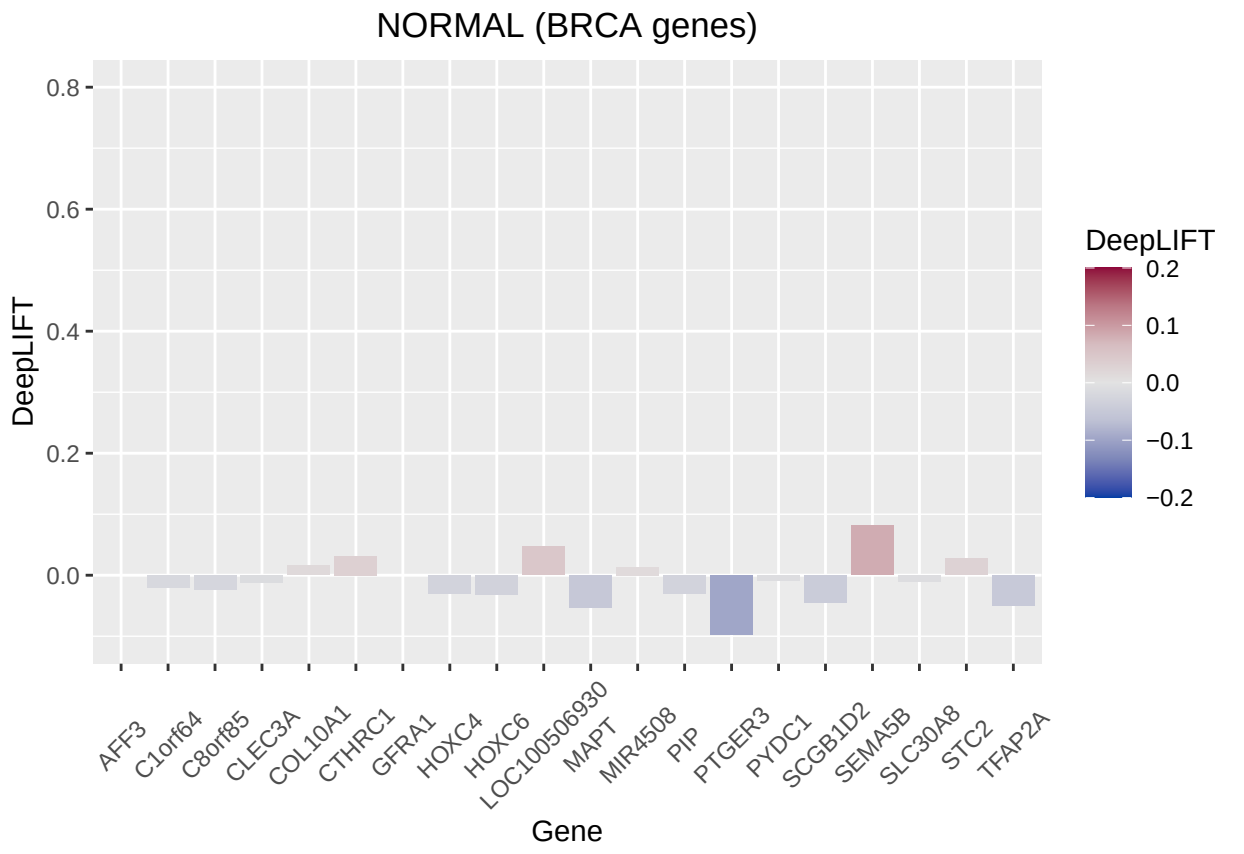
```
normal_idxs <- which(pred_labels == 10)
dl_normal <- dl_res.arr[normal_idxs, ,]
dl_normal.avg <- apply(dl_normal, c(2,3), mean)
names(dl_normal.avg) <- unique_labels

normal_features <- dl_normal.avg[, 8]
df_normal_brca_features <- as.data.frame(
  normal_features[brca_top_idxs]
)
df_normal_brca_features <- reIndex(df_normal_brca_features)
```

```
q.rand <- ggplot(data = df_brca_rand_features, aes_string(x = "Gene", y = "DeepLIFT", fill = "DeepLIFT"
                 geom_bar(stat = "identity") +
                 scale_fill_gradientn(colours = colorspace::diverge_hcl(7), limits = c(-0.2, 0.2)) +
                 ylim(c(-0.1, 0.8)) +
                 ggtitle("BRCA (rand genes)") +
                 theme(plot.title = element_text(hjust = 0.5),
                       axis.text.x = element_text(angle = 45, vjust = 0.5, hjust = 0.5))
q.rand
```



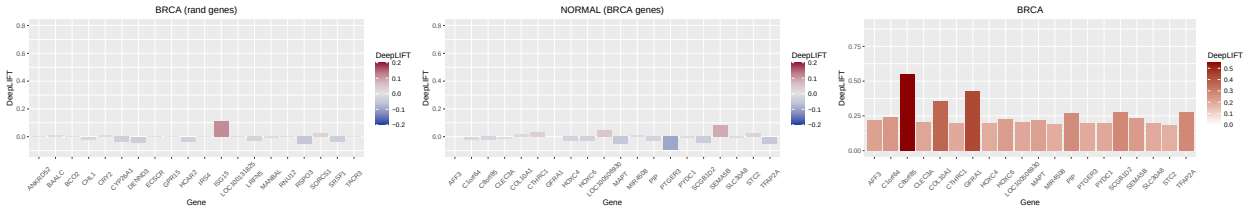

```
q.n_br <- ggplot(data = df_normal_brca_features, aes_string(x = "Gene", y = "DeepLIFT", fill = "DeepLIF
                 geom_bar(stat = "identity") +
                 scale_fill_gradientn(colours = colorspace::diverge_hcl(7), limits = c(-0.2, 0.2)) +
                 ylim(c(-0.1, 0.8)) +
                 ggtitle("NORMAL (BRCA genes)") +
                 theme(plot.title = element_text(hjust = 0.5),
                       axis.text.x = element_text(angle = 45, vjust = 0.5, hjust = 0.5))
q.n_br
```

```
## Warning: Removed 2 rows containing missing values (position_stack).
```




```
grid.arrange(q.rand, q.n_br, q_list$BRCA, ncol = 3)
```

```
## Warning: Removed 2 rows containing missing values (position_stack).
```




```
q_brca <- arrangeGrob(q.rand, q.n_br, q_list$BRCA, ncol = 3)
```

```
## Warning: Removed 2 rows containing missing values (position_stack).
```

```
ggsave("../slides+report/dl_brca_V2.png", q_brca, width = 20, height = 4)
```