The R torch codebook

Tengku Muhammad Hanis Bin Tengku Mokhtar, PhD

02/13/2026

Table of contents

Preface

Deep learning (DL) and artificial intelligence (AI) are very popular nowadays — at least in Malaysia at the time I am writing this chapter. There are numerous open-source and freely available resources to learn these topics, especially deep learning. The two most popular software ecosystems used in this area are Python and R. Python has been the industry standard; R, however, is catching up quickly.

In the Python ecosystem the two main libraries are TensorFlow and PyTorch. On the R side there are corresponding packages: tensorflow and torch. The tensorflow R package is essentially a wrapper around the Python TensorFlow API: although we write code in R, computations run in Python on the backend, and many error and warning messages therefore originate from Python. In recent years TensorFlow has reduced its GPU support. At the time of writing, macOS 12.0 (Monterey) or later (64-bit) does not have official GPU support, and Windows (native) — Windows 7 or later (64-bit) — also lacks GPU support for TensorFlow versions beyond 2.10. Despite these limitations, TensorFlow's earlier establishment means there is a large selection of tutorials and books for running deep learning with TensorFlow.

PyTorch (and the R torch package) was developed later than TensorFlow. Consequently, there are fewer tutorials and books for PyTorch and especially for torch in R. Nevertheless, a major advantage of PyTorch and torch is that GPU support is available and is often easier to set up than it used to be for TensorFlow. The relative scarcity of learning resources remains, however. At the time of writing, the most complete reference for torch is Deep Learning and Scientific Computing with R torch by Sigrid Keydana (Keydana 2023).

This book is my effort to compile R code related to torch — not only to consolidate my own understanding, but also to provide a practical resource for others. I hope this work will be a useful contribution.

Lastly, I want to express my gratitude and deep love to my late wife, Nurul Asmaq (al-Fatihah); without her I am not who I am today. To my son Hanif and to my parents, Tengku Mokhtar and Nor Malaysia, I love you all more than I can express. Thank you for always supporting me.

Tengku Muhammad Hanis Bin Tengku Mokhtar, PhD

About the book

Welcome to **The R torch codebook**.

This book is a consolidated repository of R code for deep learning using the `torch` ecosystem. Unlike traditional textbooks, this is not a theoretical guide. It is designed as a **functional compendium** for practitioners who want to move straight to implementation across diverse data modalities.

Core Philosophy

The aim of this book is to provide a single, searchable location for all R-based `torch` implementations.

- **Code-First:** Text is kept to a minimum. The value lies in the executable syntax.
- **Modality-Driven:** Chapters are organized by data type (Tabular, Image, Audio, Video), reflecting the progression of tensor dimensions and architectural complexity.
- **Standardized Workflow:** Every analysis follows a consistent logical flow:
    1. **Dataset Description:** A brief overview of the data used.
    2. **Objectives:** The specific goal of the deep learning task (e.g., classification, regression).
    3. **Implementation:** The complete `torch` code required to load data, build the model, and execute training.

How to Use This Book

This book assumes a working knowledge of the R programming language and a basic understanding of deep learning concepts (like backpropagation and gradient descent).

Readers are encouraged to:

1. **Search by Modality:** Use the sidebar to find the data type relevant to your project.

2. **Adapt the Recipes:** Copy the code blocks and swap the dataset description/loader with your own local data.
3. **Cross-Reference:** Observe how `torch` handles different tensor dimensions as you move from Tabular (2D) to Video (5D) data.

Technical Stack

All examples are built using the latest versions of the following R packages:

- `torch`: The core tensor and autograd library.
- `luz`: High-level API for model training and management.
- `torchvision` / `torchaudio`: Specialized utilities for imaging and signal processing.

License & Attribution

If you use code snippets or architectures from this book in your own research, software, or publications, please cite the author as:

> **Hanis, Tengku Muhammad (2026).** ***The R torch codebook.*** **https://tengku-hanis.github.io/r_torch_codebook/**

Part I

Chapter 1: Deep neural network for tabular data

1 Tutorial 1: Basic deep neural network (binary classification)

Aims: To predict a heart disease using a deep neural network.

Data: `heartdisease` data from `MLDataR` package.

Code description: This codes demonstrate the use of deep neural network through torch but at the same time, still using `tidymodels` functions for splitting data, preprocessing, and performance metrics.

Packages

```
library(torch)
library(luz)
library(tidyverse)
library(tidymodels)
library(MLDataR)
```

Data

```
heart_df <-
heartdisease %>%
mutate(across(c(Sex, RestingECG, Angina), as.factor))
```

Explore data.

```
skimr::skim(heart_df)
```

Table 1.1: Data summary

Name	heart_df

Number of rows	918
Number of columns	10
Column type frequency:	
factor	3
numeric	7
Group variables	None

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
Sex	0	1	FALSE	2	M: 725, F: 193
RestingECG	0	1	FALSE	3	Nor: 552, LVH: 188, ST: 178
Angina	0	1	FALSE	2	N: 547, Y: 371

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
Age	0	1	53.51	9.43	28.0	47.00	54.0	60.0	77.0	
RestingBP	0	1	132.40	18.51	0.0	120.00	130.0	140.0	200.0	
Cholesterol	0	1	198.80	109.38	0.0	173.25	223.0	267.0	603.0	
FastingBS	0	1	0.23	0.42	0.0	0.00	0.0	0.0	1.0	
MaxHR	0	1	136.81	25.46	60.0	120.00	138.0	156.0	202.0	
HeartPeakReading	0	1	0.89	1.07	-2.6	0.00	0.6	1.5	6.2	
HeartDisease	0	1	0.55	0.50	0.0	0.00	1.0	1.0	1.0	

Split data

```
set.seed(123)
split_ind <- initial_validation_split(heart_df, strata = "HeartDisease")
```

```
heart_train <- training(split_ind)
heart_val <- validation(split_ind)
heart_test <- testing(split_ind)
```

Preprocessing

```
heart_rc <-
recipe(HeartDisease ~., data = heart_train) %>%
step_normalize(all_numeric_predictors()) %>%
step_dummy(all_factor_predictors())

heart_train_processed <- heart_rc %>% prep() %>% bake(new_data = NULL)

heart_val_processed <- heart_rc %>% prep() %>% bake(new_data = heart_val)

heart_test_processed <- heart_rc %>% prep() %>% bake(new_data = heart_test)
```

Conver to dataloader

Convert to torch dataset

```
dat_train_torch <-
tensor_dataset(
    # Features
    heart_train_processed %>%
    select(-HeartDisease) %>%
    as.matrix() %>%
    torch_tensor(dtype = torch_float()),
    # Outcome
    heart_train_processed$HeartDisease %>%
    torch_tensor(dtype = torch_float()) %>%
    torch_unsqueeze(2)
)

dat_val_torch <-
tensor_dataset(
    # Features
    heart_val_processed %>%
    select(-HeartDisease) %>%
```

```
    as.matrix() %>%
    torch_tensor(dtype = torch_float()),
    # Outcome
    heart_val_processed$HeartDisease %>%
    torch_tensor(dtype = torch_float()) %>%
    torch_unsqueeze(2)
)

dat_test_torch <-
tensor_dataset(
    # Features
    heart_test_processed %>%
    select(-HeartDisease) %>%
    as.matrix() %>%
    torch_tensor(dtype = torch_float()),
    # Outcome
    heart_test_processed$HeartDisease %>%
    torch_tensor(dtype = torch_float()) %>%
    torch_unsqueeze(2)
)
```

Dataloader

```
train_dl <- dataloader(dat_train_torch, batch_size = 10, shuffle = TRUE)
val_dl <- dataloader(dat_val_torch, batch_size = 10, shuffle = FALSE)
test_dl <- dataloader(dat_test_torch, batch_size = 10, shuffle = FALSE)
```

Specify the model

```
net <- nn_module(
    initialize = function(d_in){
        self$net <- nn_sequential(
            nn_linear(d_in, 32),
            nn_relu(),
            nn_dropout(0.5),
            nn_linear(32, 64),
            nn_relu(),
            nn_dropout(0.5),
            nn_linear(64, 1),
            nn_sigmoid()
```

```
            )
        },
    forward = function(x){
        self$net(x)
    }
)
```

Fit the model

Set parameters

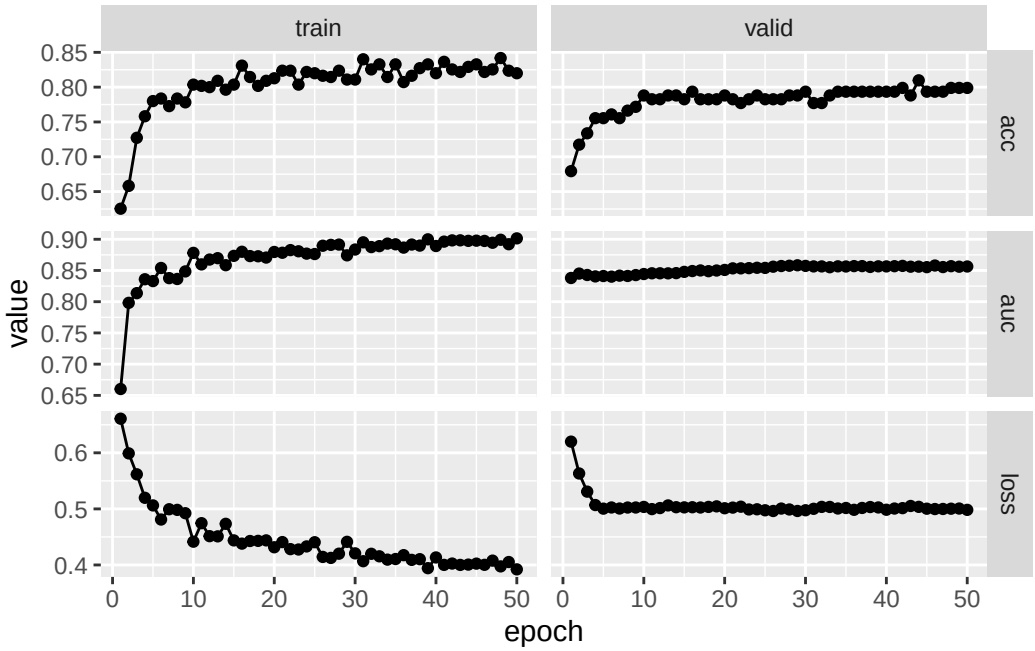
```
d_in <- length(heart_train_processed) - 1 # no of features minus the outcome
```

Fit

```
fitted <-
net %>%
setup(
    loss = nn_bce_loss(),
    optimizer = optim_adam,
    metrics = list(
        luz_metric_binary_accuracy(),
        luz_metric_binary_auroc())
) %>%
set_hparams(d_in = d_in) %>%
fit(
    train_dl,
    epoch = 50,
    valid_data = val_dl
)
```

Training plot

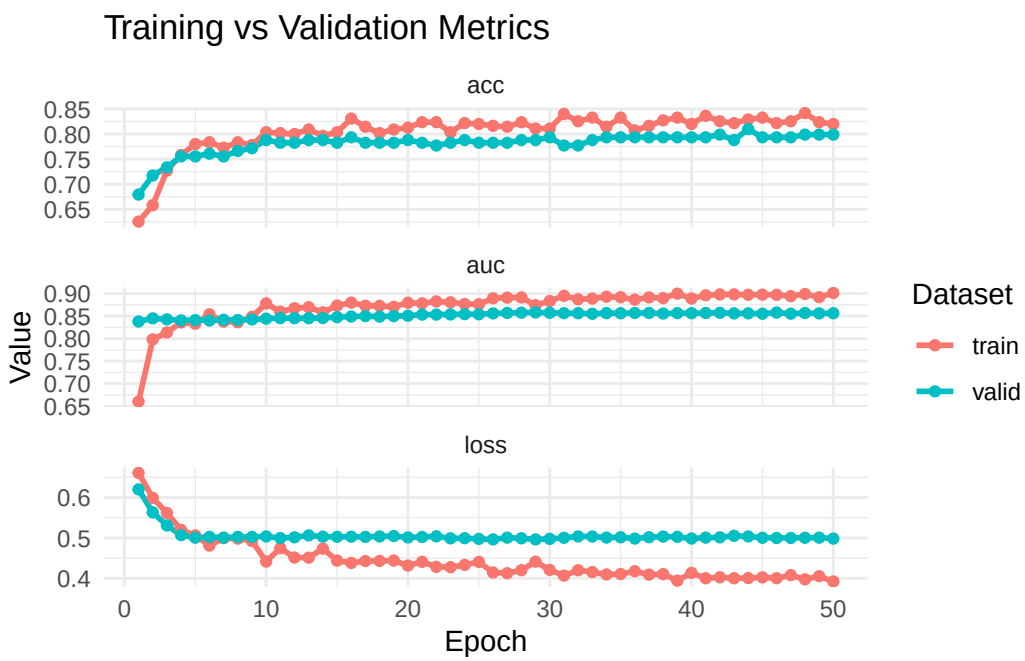
```
fitted %>% plot()
```



Better plot

```
hist <- get_metrics(fitted)

hist %>%
ggplot(aes(x = epoch, y = value, color = set)) +
geom_line(linewidth = 1) + # Draw lines
geom_point(size = 1.5) + # Add points for clarity
facet_wrap(~ metric, scales = "free_y", ncol = 1) + # Stack metrics vertically
theme_minimal() +
labs(
    title = "Training vs Validation Metrics",
    y = "Value",
    x = "Epoch",
    color = "Dataset"
)
```



Re-fit the model

Fit

```
fitted2 <-
net %>%
setup(
    loss = nn_bce_loss(),
    optimizer = optim_adam,
    metrics = list(
        luz_metric_binary_accuracy(),
        luz_metric_binary_auroc() )
) %>%
set_hparams(d_in = d_in) %>%
fit(
    train_dl,
    epoch = 5,
    valid_data = val_dl
)
```

Predict testing set

```
y_pred <- fitted2 %>% predict(test_dl)

dat_pred <-
y_pred %>%
as_array() %>%
as_tibble(.name_repair = "unique") %>%
 rename(prob = 1) %>%
mutate(
    pred = factor(ifelse(prob > 0.5, 1, 0)),
    true = factor(heart_test$HeartDisease)
    )
```

```
New names:
* `` -> `...1`
```

```
dat_pred
```

```
# A tibble: 184 x 3
    prob pred  true
   <dbl> <fct> <fct>
 1 0.271 0     1
 2 0.293 0     0
 3 0.712 1     1
 4 0.579 1     1
 5 0.235 0     0
 6 0.213 0     0
 7 0.247 0     0
 8 0.492 0     0
 9 0.705 1     0
10 0.172 0     0
# i 174 more rows
```

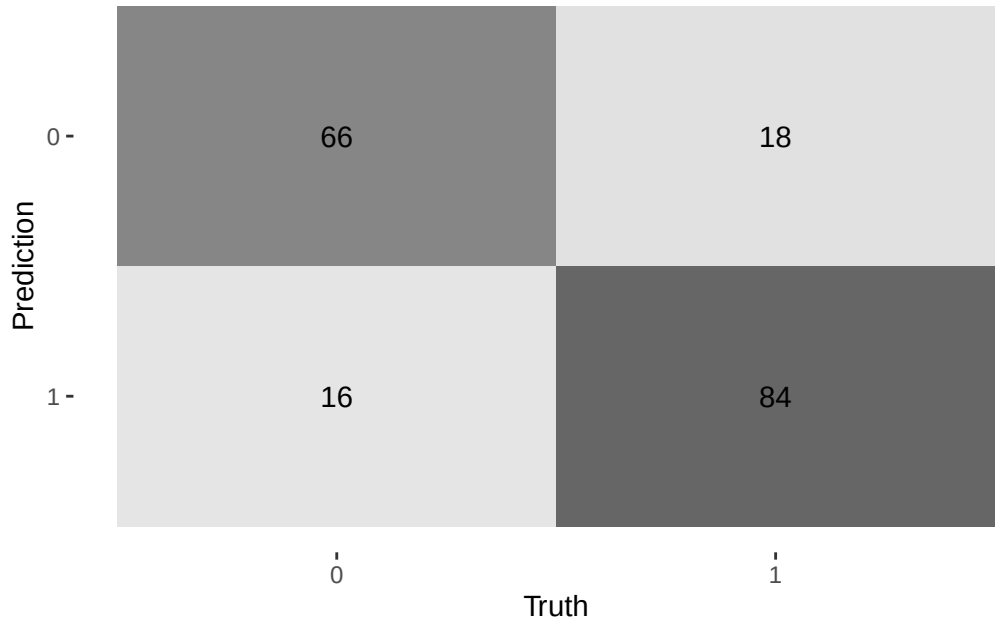
Evaluate

```
fitted %>% evaluate(test_dl) # Less accurate
```

```
A `luz_module_evaluation`
-- Results ---------------------------------------------------------------------
loss: 0.3917
acc: 0.8261
auc: 0.9009
```

Confusion matrix

```
dat_pred %>%
conf_mat(true, pred) %>%
autoplot("heatmap")
```



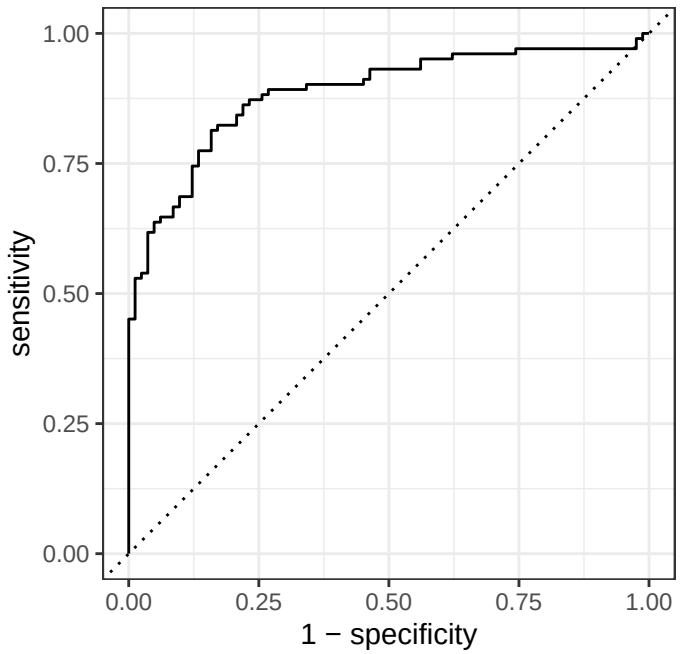
Accuracy

```
dat_pred %>%
accuracy(truth = true, estimate = pred)
```

```
# A tibble: 1 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.815
```

Plot ROC

```
dat_pred %>%
roc_curve(true, prob, event_level = "second") %>%
autoplot()
```




```
# ROC-AUC
dat_pred %>%
roc_auc(true, prob, event_level = "second")
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 roc_auc binary         0.886
```

2 Tutorial 2: Deep neural network with dataset functions and callbacks (binary classification)

Aims: To predict heart disease using a deep neural network with custom dataset functions and training callbacks.

Data: `heartdisease` data from `MLDataR` package.

Code description: This code demonstrates the use of torch with custom dataset functions, dataset subsets, and training callbacks including early stopping and best model checkpointing.

Packages

```
library(torch)
library(luz)
library(tidyverse)
library(tidymodels)
library(MLDataR)
```

Data

```
heart_df <-
  heartdisease %>%
  mutate(across(c(Sex, RestingECG, Angina), as.factor))
```

Explore data.

```
skimr::skim(heart_df)
```

Table 2.1: Data summary

Name	heart_df

Number of rows	918
Number of columns	10
Column type frequency:	
factor	3
numeric	7
Group variables	None

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
Sex	0	1	FALSE	2	M: 725, F: 193
RestingECG	0	1	FALSE	3	Nor: 552, LVH: 188, ST: 178
Angina	0	1	FALSE	2	N: 547, Y: 371

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
Age	0	1	53.51	9.43	28.0	47.00	54.0	60.0	77.0	
RestingBP	0	1	132.40	18.51	0.0	120.00	130.0	140.0	200.0	
Cholesterol	0	1	198.80	109.38	0.0	173.25	223.0	267.0	603.0	
FastingBS	0	1	0.23	0.42	0.0	0.00	0.0	0.0	1.0	
MaxHR	0	1	136.81	25.46	60.0	120.00	138.0	156.0	202.0	
HeartPeakReading	0	1	0.89	1.07	-2.6	0.00	0.6	1.5	6.2	
HeartDisease	0	1	0.55	0.50	0.0	0.00	1.0	1.0	1.0	

Dataset function

```
heart_dataset <- dataset(
  initialize = function(df) {
```

```
    # Pre-process and store as tensors
    self$x_num <- df %>%
      select(Age, RestingBP, Cholesterol, FastingBS, MaxHR, HeartPeakReading) %>%
      mutate(across(everything(), scale)) %>%
      as.matrix() %>%
      torch_tensor(dtype = torch_float())

    self$x_cat <- model.matrix(~ Sex + RestingECG + Angina, data = df)[, -1] %>%
      as.matrix() %>%
      torch_tensor(dtype = torch_float())

    self$y <- torch_tensor(as.matrix(df$HeartDisease), dtype = torch_float())
  },
  .getitem = function(i) {
    list(x = list(self$x_num[i, ], self$x_cat[i, ]), y = self$y[i])
  },
  .length = function() {
    self$y$size(1)
  }
)

# Convert to torch dataset
ds_tensor <- heart_dataset(heart_df)
ds_tensor[1]
```

```
$x
$x[[1]]
torch_tensor
-1.4324
 0.4107
 0.8246
-0.5510
 1.3822
-0.8320
[ CPUFloatType{6} ]

$x[[2]]
torch_tensor
 1
 1
 0
 0
```

```
[ CPUFloatType{4} ]


$y
torch_tensor
 0
[ CPUFloatType{1} ]
```

Split data with dataset subsets

```
set.seed(123)
n <- nrow(heart_df)
train_size <- floor(0.6 * n)
valid_size <- floor(0.2 * n)

# Create indices
all_indices <- 1:n
train_indices <- sample(all_indices, size = train_size)

remaining_indices <- setdiff(all_indices, train_indices)
valid_indices <- sample(remaining_indices, size = valid_size)

test_indices <- setdiff(remaining_indices, valid_indices)

# Create Subsets
train_ds <- dataset_subset(ds_tensor, train_indices)
valid_ds <- dataset_subset(ds_tensor, valid_indices)
test_ds  <- dataset_subset(ds_tensor, test_indices)
```

Convert to dataloader

```
train_dl <- train_ds %>%
  dataloader(batch_size = 10, shuffle = TRUE)

valid_dl <- valid_ds %>%
  dataloader(batch_size = 10, shuffle = FALSE)

test_dl <- test_ds %>%
  dataloader(batch_size = 10, shuffle = FALSE)
```

Specify the model

```
net <-
  nn_module(
    initialize = function(d_in){
      self$net <- nn_sequential(
        nn_linear(d_in, 32),
        nn_relu(),
        nn_dropout(0.5),
        nn_linear(32, 64),
        nn_relu(),
        nn_dropout(0.5),
        nn_linear(64, 1),
        nn_sigmoid()
      )
    },
    forward = function(x){
      # x is currently a list of two tensors (numeric and categorical)
      # Concatenate them along the feature dimension (dim=2)
      input <- torch_cat(x, dim = 2)
      self$net(input)
    }
  )
```

Fit the model

Set parameters

```
d_in <- length(ds_tensor[1]$x[[1]]) + length(ds_tensor[1]$x[[2]]) # total number of features
```

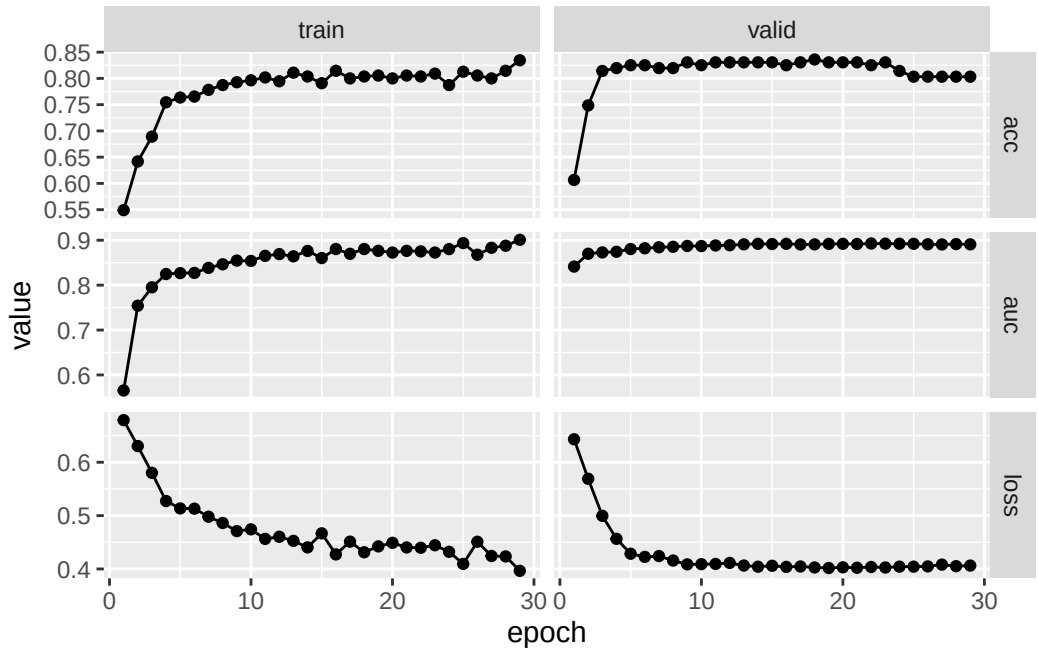
Fit with callbacks

```
fitted <-
  net %>%
  setup(
    loss = nn_bce_loss(),
    optimizer = optim_adam,
    metrics = list(
      luz_metric_binary_accuracy(),
      luz_metric_binary_auroc()
```

```
    )
  ) %>%
  set_hparams(d_in = d_in) %>%
  fit(
    train_dl,
    epoch = 50,
    valid_data = valid_dl,
    callbacks = list(
      luz_callback_early_stopping(patience = 10),
      luz_callback_keep_best_model()
    )
  )
```

Training plot

```
fitted %>% plot()
```

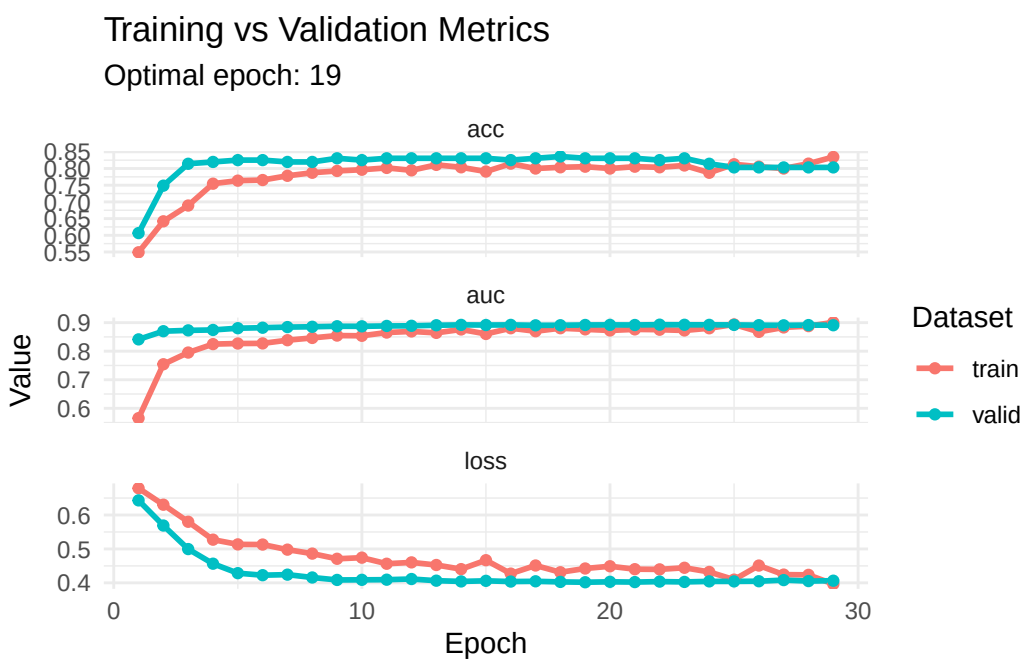


Better plot

```r
hist <- get_metrics(fitted)

optimal_epoch <- hist %>%
  filter(metric == "loss", set == "valid") %>%
  slice_min(value, n = 1) %>%
  pull(epoch)

hist %>%
  ggplot(aes(x = epoch, y = value, color = set)) +
  geom_line(linewidth = 1) +
  geom_point(size = 1.5) +
  facet_wrap(~ metric, scales = "free_y", ncol = 1) +
  theme_minimal() +
  labs(
    title = "Training vs Validation Metrics",
    subtitle = paste("Optimal epoch:", optimal_epoch),
    y = "Value",
    x = "Epoch",
    color = "Dataset"
  )
```



Re-fit the model

Note: No need to refit manually since we use ***`luz_callback_keep_best_model()`*** *to automatically save the best model based on validation loss.*

Predict testing set

```
y_pred <- fitted %>% predict(test_dl)
y_true <- ds_tensor$y[test_ds$indices] %>% as_array()

dat_pred <-
  y_pred %>%
  as_array() %>%
  as_data_frame() %>%
  rename(prob = V1) %>%
  mutate(
    pred = factor(ifelse(prob > 0.5, 1, 0)),
    true = factor(y_true)
  )
```

```
Warning: `as_data_frame()` was deprecated in tibble 2.0.0.
i Please use `as_tibble()` (with slightly different semantics) to convert to a
  tibble, or `as.data.frame()` to convert to a data frame.
```

```
Warning: The `x` argument of `as_tibble.matrix()` must have unique column names if
`.name_repair` is omitted as of tibble 2.0.0.
i Using compatibility `.name_repair`.
i The deprecated feature was likely used in the tibble package.
  Please report the issue at <https://github.com/tidyverse/tibble/issues>.
```

```
dat_pred
```

```
# A tibble: 185 x 3
     prob pred  true
    <dbl> <fct> <fct>
 1 0.101  0     0
 2 0.238  0     0
 3 0.0734 0     0
 4 0.963  1     1
```

```
 5 0.0776 0     0
 6 0.178  0     0
 7 0.344  0     0
 8 0.323  0     0
 9 0.780  1     0
10 0.129  0     1
# i 175 more rows
```

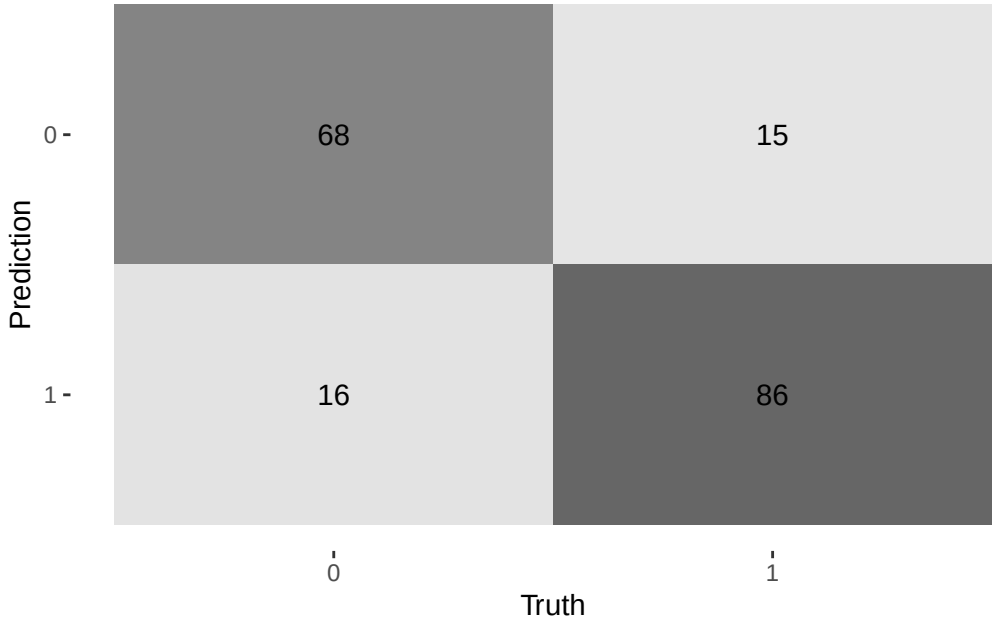
Evaluate

```
fitted %>% evaluate(test_dl)
```

```
A `luz_module_evaluation`
-- Results ---------------------------------------------------------------------
loss: 0.4049
acc: 0.8324
auc: 0.8881
```

Confusion matrix

```
dat_pred %>%
  conf_mat(true, pred) %>%
  autoplot("heatmap")
```



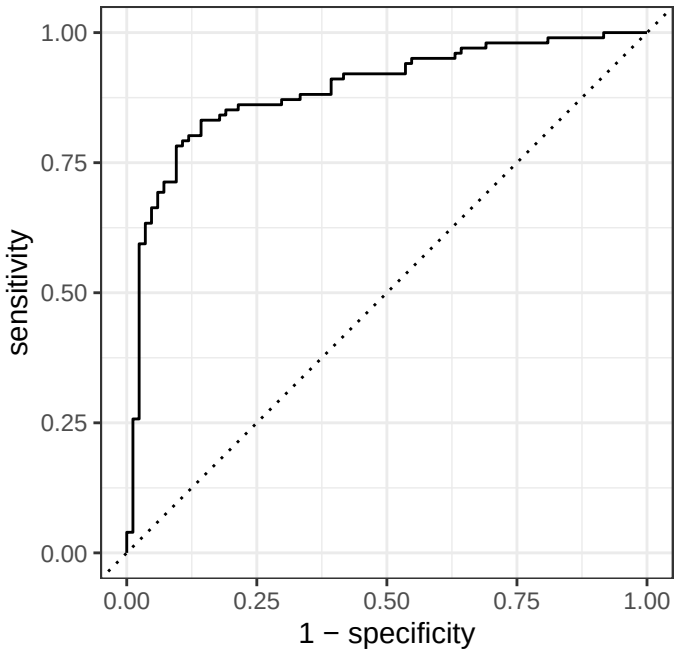
Accuracy

```
dat_pred %>%
  accuracy(truth = true, estimate = pred)
```

```
# A tibble: 1 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.832
```

Plot ROC

```
dat_pred %>%
  roc_curve(true, prob, event_level = "second") %>%
  autoplot()
```




```
# ROC-AUC
dat_pred %>%
  roc_auc(true, prob, event_level = "second")
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 roc_auc binary         0.890
```

3 Tutorial 3: Deep neural network with explainable method (binary classification)

Aims: To predict heart disease using a deep neural network - To explain model predictions using Layer-wise Relevance Propagation (LRP) - To visualize feature importance at both individual and global levels

Data: `heartdisease` data from `MLDataR` package.

Code description: This code demonstrates the use of torch with custom dataset functions, training callbacks, and model explainability using the `innsight` package for binary classification with LRP.

Packages

```
library(torch)
library(luz)
library(tidyverse)
library(tidymodels)
library(MLDataR)
library(innsight)
```

Data

```
heart_df <-
  heartdisease %>%
  mutate(across(c(Sex, RestingECG, Angina), as.factor))
```

Explore data.

```
skimr::skim(heart_df)
```

Table 3.1: Data summary

Name	heart_df
Number of rows	918
Number of columns	10
_____	
Column type frequency:	
factor	3
numeric	7
_____	
Group variables	None

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
Sex	0	1	FALSE	2	M: 725, F: 193
RestingECG	0	1	FALSE	3	Nor: 552, LVH: 188, ST: 178
Angina	0	1	FALSE	2	N: 547, Y: 371

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
Age	0	1	53.51	9.43	28.0	47.00	54.0	60.0	77.0	
RestingBP	0	1	132.40	18.51	0.0	120.00	130.0	140.0	200.0	
Cholesterol	0	1	198.80	109.38	0.0	173.25	223.0	267.0	603.0	
FastingBS	0	1	0.23	0.42	0.0	0.00	0.0	0.0	1.0	
MaxHR	0	1	136.81	25.46	60.0	120.00	138.0	156.0	202.0	
HeartPeakReading	0	1	0.89	1.07	-2.6	0.00	0.6	1.5	6.2	
HeartDisease	0	1	0.55	0.50	0.0	0.00	1.0	1.0	1.0	

Dataset function

```
heart_dataset <- dataset(
  initialize = function(df) {
    # Pre-process and store as tensors
    self$x_num <- df %>%
      select(Age, RestingBP, Cholesterol, FastingBS, MaxHR, HeartPeakReading) %>%
      mutate(across(everything(), scale)) %>%
      as.matrix() %>%
      torch_tensor(dtype = torch_float())

    self$x_cat <- model.matrix(~ Sex + RestingECG + Angina, data = df)[, -1] %>%
      as.matrix() %>%
      torch_tensor(dtype = torch_float())

    self$y <- torch_tensor(as.matrix(df$HeartDisease), dtype = torch_float())
  },
  .getitem = function(i) {
    list(x = list(self$x_num[i, ], self$x_cat[i, ]), y = self$y[i])
  },
  .length = function() {
    self$y$size(1)
  }
)

# Convert to torch dataset
ds_tensor <- heart_dataset(heart_df)
ds_tensor[1]
```

```
$x
$x[[1]]
torch_tensor
-1.4324
 0.4107
 0.8246
-0.5510
 1.3822
-0.8320
[ CPUFloatType{6} ]

$x[[2]]
torch_tensor
 1
```

```
 1
 0
 0
[ CPUFloatType{4} ]


$y
torch_tensor
 0
[ CPUFloatType{1} ]
```

Split data with dataset subsets

```
set.seed(123)
n <- nrow(heart_df)
train_size <- floor(0.6 * n)
valid_size <- floor(0.2 * n)

# Create indices
all_indices <- 1:n
train_indices <- sample(all_indices, size = train_size)

remaining_indices <- setdiff(all_indices, train_indices)
valid_indices <- sample(remaining_indices, size = valid_size)

test_indices <- setdiff(remaining_indices, valid_indices)

# Create Subsets
train_ds <- dataset_subset(ds_tensor, train_indices)
valid_ds <- dataset_subset(ds_tensor, valid_indices)
test_ds  <- dataset_subset(ds_tensor, test_indices)
```

Convert to dataloader

```
train_dl <- train_ds %>%
  dataloader(batch_size = 10, shuffle = TRUE)

valid_dl <- valid_ds %>%
  dataloader(batch_size = 10, shuffle = FALSE)
```

```
test_dl <- test_ds %>%
  dataloader(batch_size = 10, shuffle = FALSE)
```

Specify the model

```
net <-
  nn_module(
    initialize = function(d_in){
      self$net <- nn_sequential(
        nn_linear(d_in, 32),
        nn_relu(),
        nn_dropout(0.5),
        nn_linear(32, 64),
        nn_relu(),
        nn_dropout(0.5),
        nn_linear(64, 1),
        nn_sigmoid()
      )
    },
    forward = function(x){
      # x is currently a list of two tensors (numeric and categorical)
      # Concatenate them along the feature dimension (dim=2)
      input <- torch_cat(x, dim = 2)
      self$net(input)
    }
  )
```

Fit the model

Set parameters

```
d_in <- length(ds_tensor[1]$x[[1]]) + length(ds_tensor[1]$x[[2]]) # total number of features
```

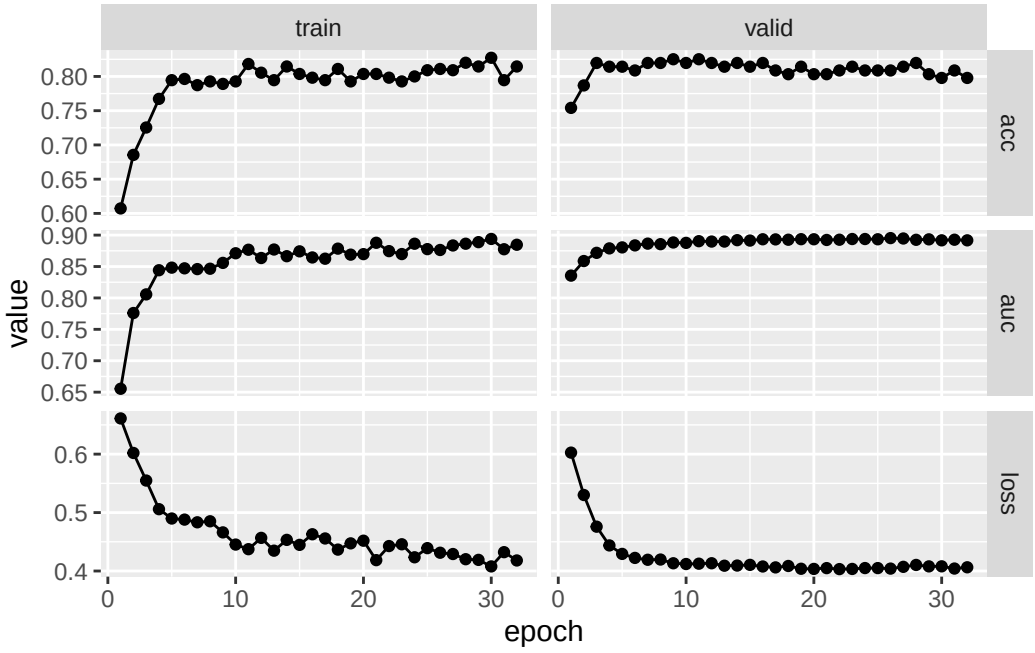
Fit with callbacks

```
fitted <-
  net %>%
  setup(
```

```
    loss = nn_bce_loss(),
    optimizer = optim_adam,
    metrics = list(
      luz_metric_binary_accuracy(),
      luz_metric_binary_auroc()
    )
  ) %>%
  set_hparams(d_in = d_in) %>%
  fit(
    train_dl,
    epoch = 50,
    valid_data = valid_dl,
    callbacks = list(
      luz_callback_early_stopping(patience = 10),
      luz_callback_keep_best_model()
    )
  )
```

Training plot

```
fitted %>% plot()
```

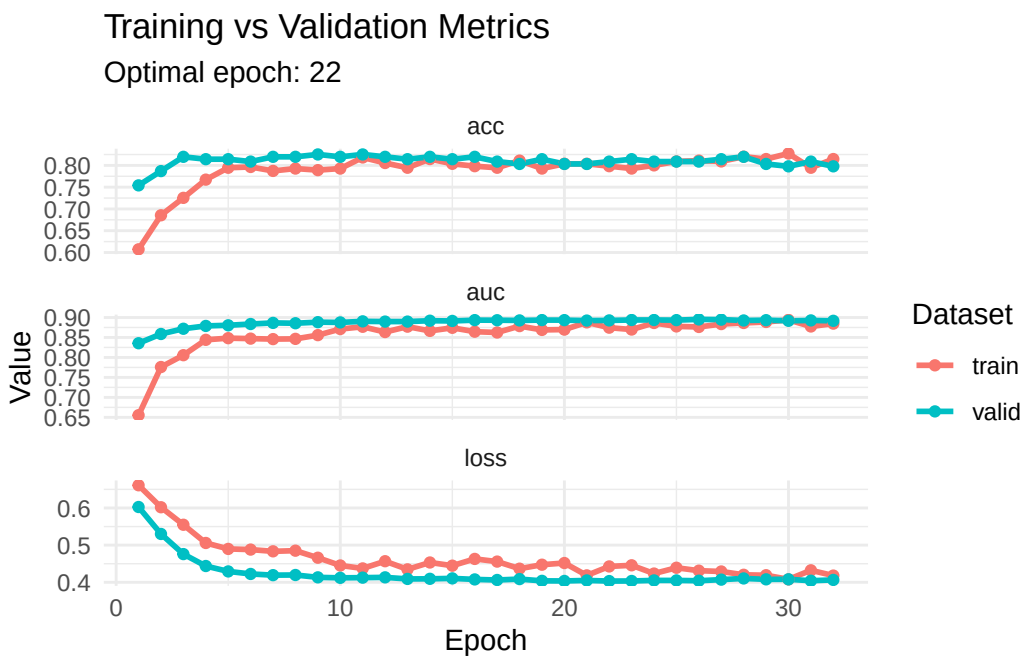


Better plot

```
hist <- get_metrics(fitted)

optimal_epoch <- hist %>%
  filter(metric == "loss", set == "valid") %>%
  slice_min(value, n = 1) %>%
  pull(epoch)

hist %>%
  ggplot(aes(x = epoch, y = value, color = set)) +
  geom_line(linewidth = 1) +
  geom_point(size = 1.5) +
  facet_wrap(~ metric, scales = "free_y", ncol = 1) +
  theme_minimal() +
  labs(
    title = "Training vs Validation Metrics",
    subtitle = paste("Optimal epoch:", optimal_epoch),
    y = "Value",
    x = "Epoch",
    color = "Dataset"
  )
```



Predict testing set

```
y_pred <- fitted %>% predict(test_dl)
y_true <- ds_tensor$y[test_ds$indices] %>% as_array()

dat_pred <-
  y_pred %>%
  as_array() %>%
  as_data_frame() %>%
  rename(prob = V1) %>%
  mutate(
    pred = factor(ifelse(prob > 0.5, 1, 0)),
    true = factor(y_true)
  )
```

```
Warning: `as_data_frame()` was deprecated in tibble 2.0.0.
i Please use `as_tibble()` (with slightly different semantics) to convert to a
  tibble, or `as.data.frame()` to convert to a data frame.
```

```
Warning: The `x` argument of `as_tibble.matrix()` must have unique column names if
`.name_repair` is omitted as of tibble 2.0.0.
i Using compatibility `.name_repair`.
i The deprecated feature was likely used in the tibble package.
  Please report the issue at <https://github.com/tidyverse/tibble/issues>.
```

```
dat_pred
```

```
# A tibble: 185 x 3
     prob pred  true
    <dbl> <fct> <fct>
 1 0.102  0     0
 2 0.343  0     0
 3 0.0887 0     0
 4 0.963  1     1
 5 0.0971 0     0
 6 0.160  0     0
 7 0.312  0     0
 8 0.316  0     0
 9 0.806  1     0
10 0.147  0     1
# i 175 more rows
```

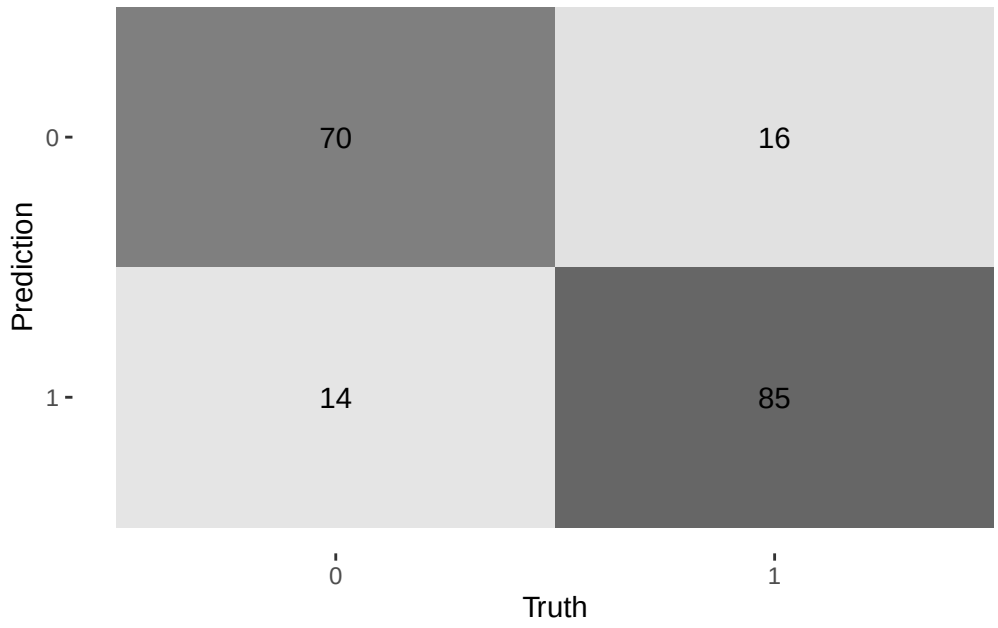
Evaluate

```
fitted %>% evaluate(test_dl)
```

```
A `luz_module_evaluation`
-- Results ---------------------------------------------------------------------
loss: 0.4054
acc: 0.8378
auc: 0.888
```

Confusion matrix

```
dat_pred %>%
  conf_mat(true, pred) %>%
  autoplot("heatmap")
```



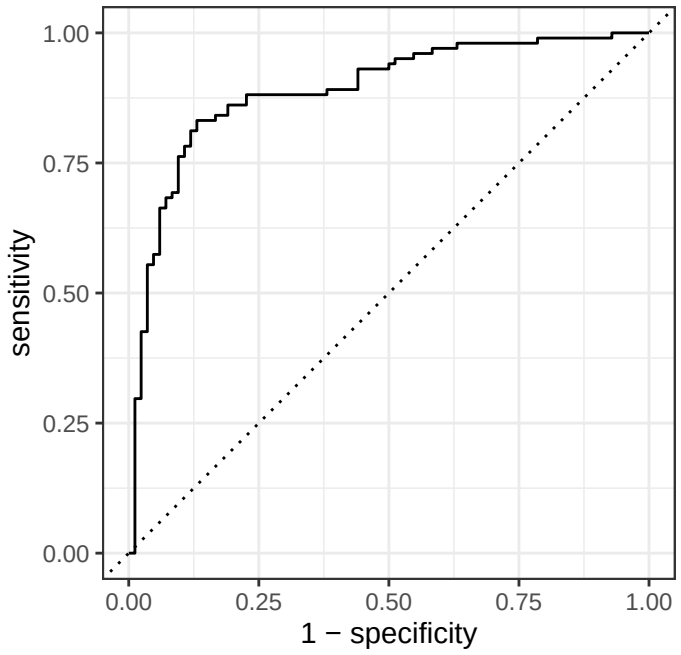
Accuracy

```
dat_pred %>%
  accuracy(truth = true, estimate = pred)
```

```
# A tibble: 1 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.838
```

Plot ROC

```
dat_pred %>%
  roc_curve(true, prob, event_level = "second") %>%
  autoplot()
```




```
# ROC-AUC
dat_pred %>%
  roc_auc(true, prob, event_level = "second")
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 roc_auc binary         0.890
```

Model explainability with LRP

Prepare model for interpretation.

```
# Extract the sequential model
model <- fitted$model$net$cpu()

# Define input and output names
input_names <- c(
  # Numeric variables
  "Age", "RestingBP", "Cholesterol", "FastingBS", "MaxHR", "HeartPeakReading",
  # Categorical dummies (from model.matrix ~ .-1)
  "Sex_M", "RestingECG_Normal", "RestingECG_ST", "Angina_Y"
)

output_names <- c("Probability of heart disease")
```

Create converter and prepare test data.

```
# Create the Converter object
converter <- convert(
  model,
  input_dim = 10,
  input_names = input_names,
  output_names = output_names
)
```

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

```
# Manually extract and concatenate the test data
idxs <- test_ds$indices
x_num <- ds_tensor$x_num[idxs, ]
x_cat <- ds_tensor$x_cat[idxs, ]

# Combine into one tensor and convert to R array
input_tensor <- torch_cat(list(x_num, x_cat), dim = 2)
input_data <- as_array(input_tensor)
```

Apply Layer-wise Relevance Propagation (LRP).

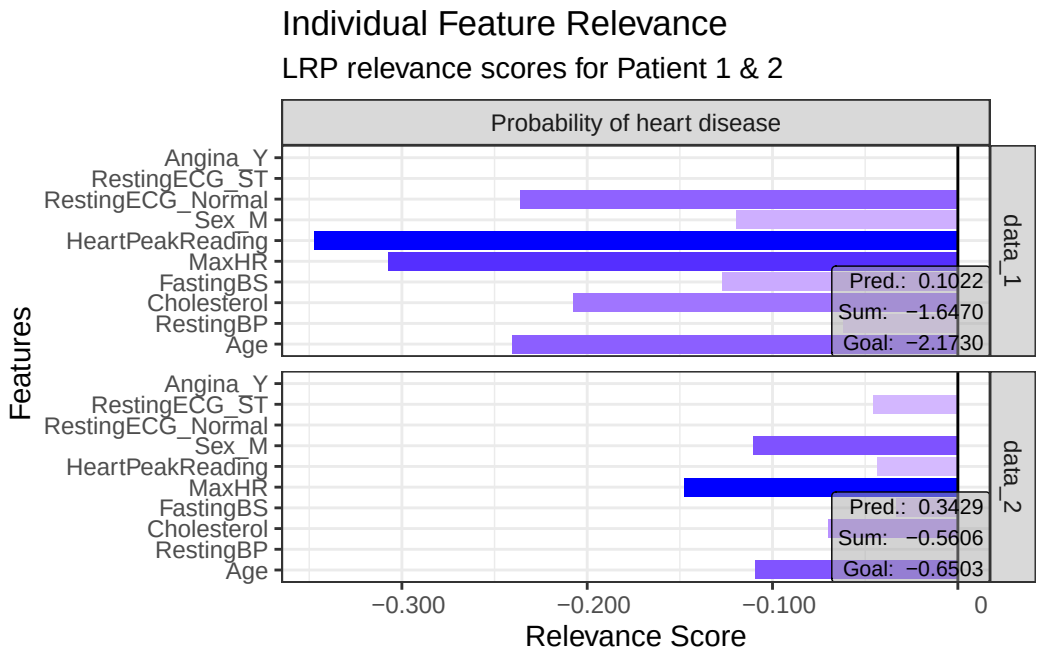
```r
# Run LRP with alpha-beta rule
lrp_result <- run_lrp(converter, input_data, rule_name = "alpha_beta", rule_param = 1)

# Check dimensions: Instances x Features x Outputs
dim(get_result(lrp_result))
```

```
[1] 185  10   1
```

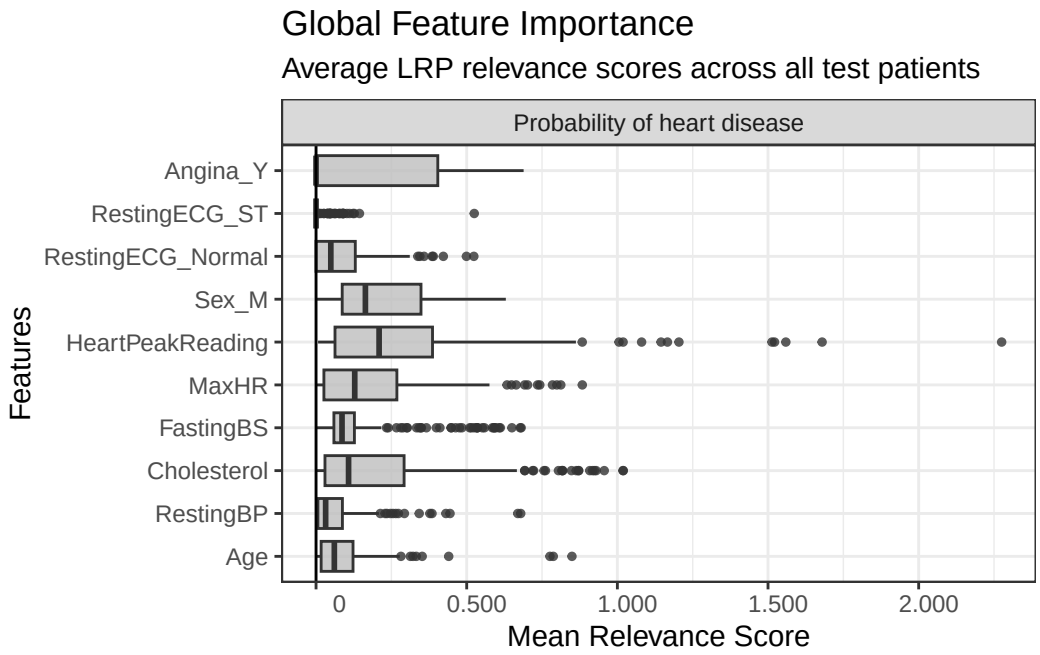
Individual explanations.

```r
# Individual plots for the first two test instances
plot(lrp_result, data_idx = c(1, 2)) +
  theme_bw() +
  coord_flip() +
  labs(
    title = "Individual Feature Relevance",
    subtitle = "LRP relevance scores for Patient 1 & 2",
    x = "Features",
    y = "Relevance Score"
  )
```



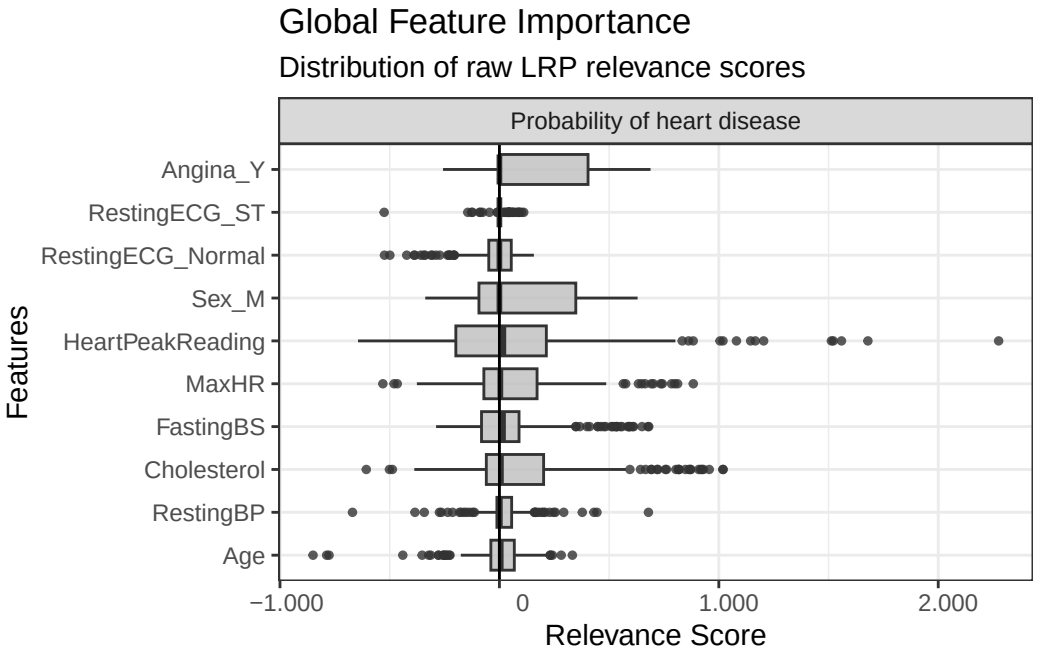
Global explanations.

```r
# Global boxplot - overall feature importance across the entire test set
boxplot(lrp_result) +
  theme_bw() +
  coord_flip() +
  labs(
    title = "Global Feature Importance",
    subtitle = "Average LRP relevance scores across all test patients",
    x = "Features",
    y = "Mean Relevance Score"
  )
```



Global explanation with raw relevance scores.

```r
# Another version of global boxplot with identity transformation
boxplot(lrp_result, preprocess_FUN = identity) +
  theme_bw() +
  coord_flip() +
  labs(
    title = "Global Feature Importance",
    subtitle = "Distribution of raw LRP relevance scores",
    x = "Features",
    y = "Relevance Score"
  )
```


Global Feature Importance
Distribution of raw LRP relevance scores
Probability of heart disease
Angina_Y
RestingECG_ST
RestingECG_Normal
Sex_M
HeartPeakReading
MaxHR
FastingBS
Cholesterol
RestingBP
Age
Features
−1.000
0
1.000
2.000
Relevance Score

4 Tutorial 4: Deep neural network with categorical embedding (binary classification)

5 Tutorial 5: Deep neural network with resampling technique (binary classification)

Part II

Chapter 2: Deep learning for text data

6 Tutorial

Part III

Chapter 3: Deep learning for image data

7 Tutorial

Part IV

Chapter 4: Deep learning for audio data

8 Tutorial

Part V

Chapter 5: Deep learning for video data

9 Tutorial

Part VI

Chapter 6: Deep learning for other types of data

10 Tutorial

References

Keydana, Sigrid. 2023. *Deep Learning and Scientific Computing with r Torch*. Chapman; Hall/CRC.