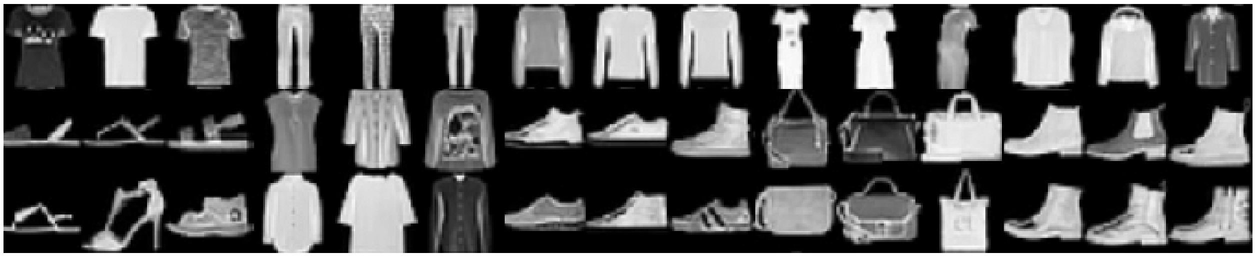
Deep Neural Networks - Fashion Mnist



Set up - load libraries and data set

```r
# Load libraries
library(keras)
library(ggplot2)
library(dplyr)

# Load data set
fashion_mnist <- dataset_fashion_mnist()
```

Data

Fashion-MNIST is a data set of Zalando's article images - it consists 60000 pictures for training and 10000 pictures for testing purposes.

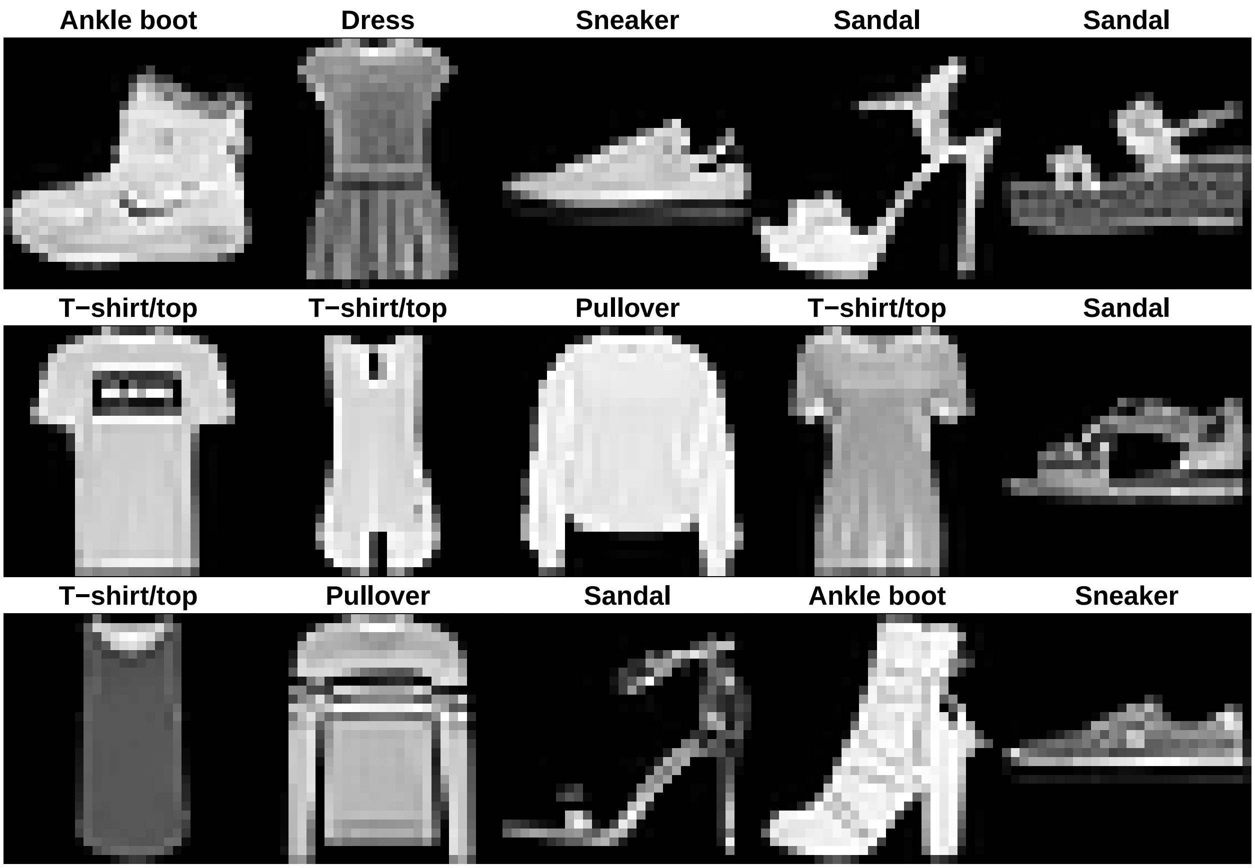
Each image is 28px in width and 28px in height, each pixel is represented by a greyscale value from 0 to 255 (where 0 means white and 255 - black).

Column 1 (train_labels / test_labels) is the class label, where numbers 0-9 represent one from 10 categories:

- 0 T-shirt/top
- 1 Trouser
- 2 Pullover
- 3 Dress
- 4 Coat
- 5 Sandal
- 6 Shirt
- 7 Sneaker
- 8 Bag
- 9 Ankle boot.

Data set preparation

```r
# Train data set
c(train_images, train_labels) %<-% fashion_mnist$train

#Test data set
c(test_images, test_labels) %<-% fashion_mnist$test

# Rescale data set
train_images <- train_images / 255
test_images <- test_images / 255

# Create classes vector
class_names = c('T-shirt/top',
                'Trouser',
                'Pullover',
                'Dress',
                'Coat',
                'Sandal',
                'Shirt',
                'Sneaker',
                'Bag',
                'Ankle boot')

# Show examples from data set
par(mfcol = c(3, 5))
par(mar = c(0, 0, 1.5, 0), xaxs = 'i', yaxs = 'i')

for (i in 1:15) {
  img <- train_images[i, , ]
  img <- t(apply(img, 2, rev))
  image(1:28, 1:28, img, col = gray((0:255)/255), xaxt = 'n', yaxt = 'n',
        main = paste(class_names[train_labels[i] + 1]))
  }
```




```r
# Reshape each image from 28 x 28 to 1 x 28 (single-line array)
train_images <- array_reshape(train_images, c(nrow(train_images), 28 * 28))
test_images <- array_reshape(test_images, c(nrow(test_images), 28 * 28))

# Change to categorical
train_labels <- to_categorical(train_labels, 10)
test_labels <- to_categorical(test_labels, 10)
```

DNN models

```r
# Model 1: 3 dense layers with dropout, big batch size (480)
model_1 <- keras_model_sequential() %>%
  layer_dense(units = 512, activation = 'relu', input_shape = 28 * 28) %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 256, activation = 'relu') %>%
  layer_dropout(rate = 0.2) %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dropout(rate = 0.2) %>%
  layer_dense(units = 10, activation = 'softmax')

summary(model_1)
```

```
## Model: "sequential"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                     Param #
## ================================================================================
##  dense_3 (Dense)                    (None, 512)                      401920
##  dropout_2 (Dropout)                (None, 512)                      0
##  dense_2 (Dense)                    (None, 256)                      131328
##  dropout_1 (Dropout)                (None, 256)                      0
##  dense_1 (Dense)                    (None, 128)                      32896
##  dropout (Dropout)                  (None, 128)                      0
##  dense (Dense)                      (None, 10)                       1290
## ================================================================================
## Total params: 567,434
## Trainable params: 567,434
## Non-trainable params: 0
## ________________________________________________________________________________
```

```
model_1 %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_adam(),
  metrics = 'accuracy')

model_1 %>% fit(train_images,
              train_labels,
              epochs = 50,
              validation_split = 0.2,
              batch_size = 480, ) -> model_1_dnn

# Model 2: 2 dense layers w/o dropout, smaller batch size (128)
model_2 <- keras_model_sequential() %>%
  layer_dense(units = 256, activation = 'relu', input_shape = 28 * 28) %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dense(units = 10, activation = 'softmax')

summary(model_2)
```

```
## Model: "sequential_1"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                     Param #
## ================================================================================
##  dense_6 (Dense)                    (None, 256)                      200960
##  dense_5 (Dense)                    (None, 128)                      32896
##  dense_4 (Dense)                    (None, 10)                       1290
## ================================================================================
## Total params: 235,146
## Trainable params: 235,146
## Non-trainable params: 0
## ________________________________________________________________________________
```

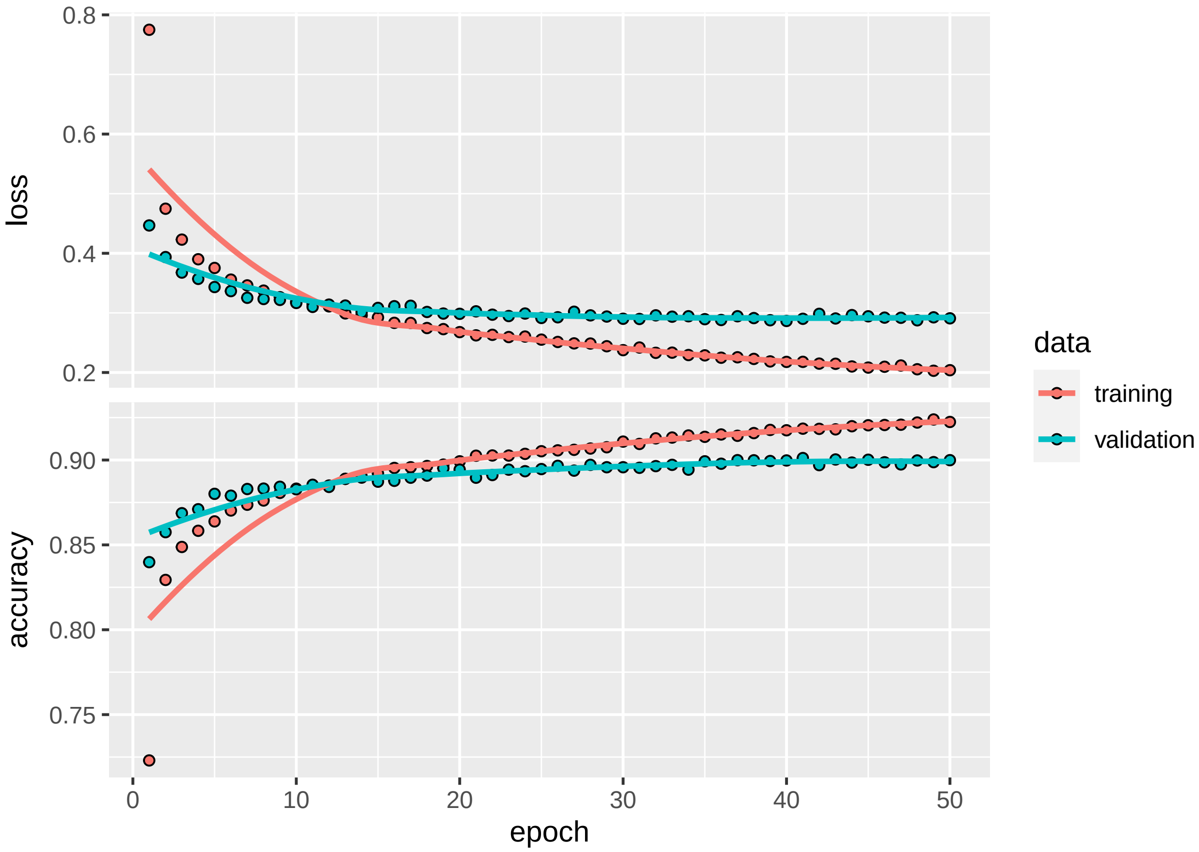
```
model_2 %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_adam(),
  metrics = 'accuracy')
```

```
model_2 %>% fit(train_images,
                train_labels,
                epochs = 50,
                validation_split = 0.2,
                batch_size = 96, ) -> model_2_dnn
```

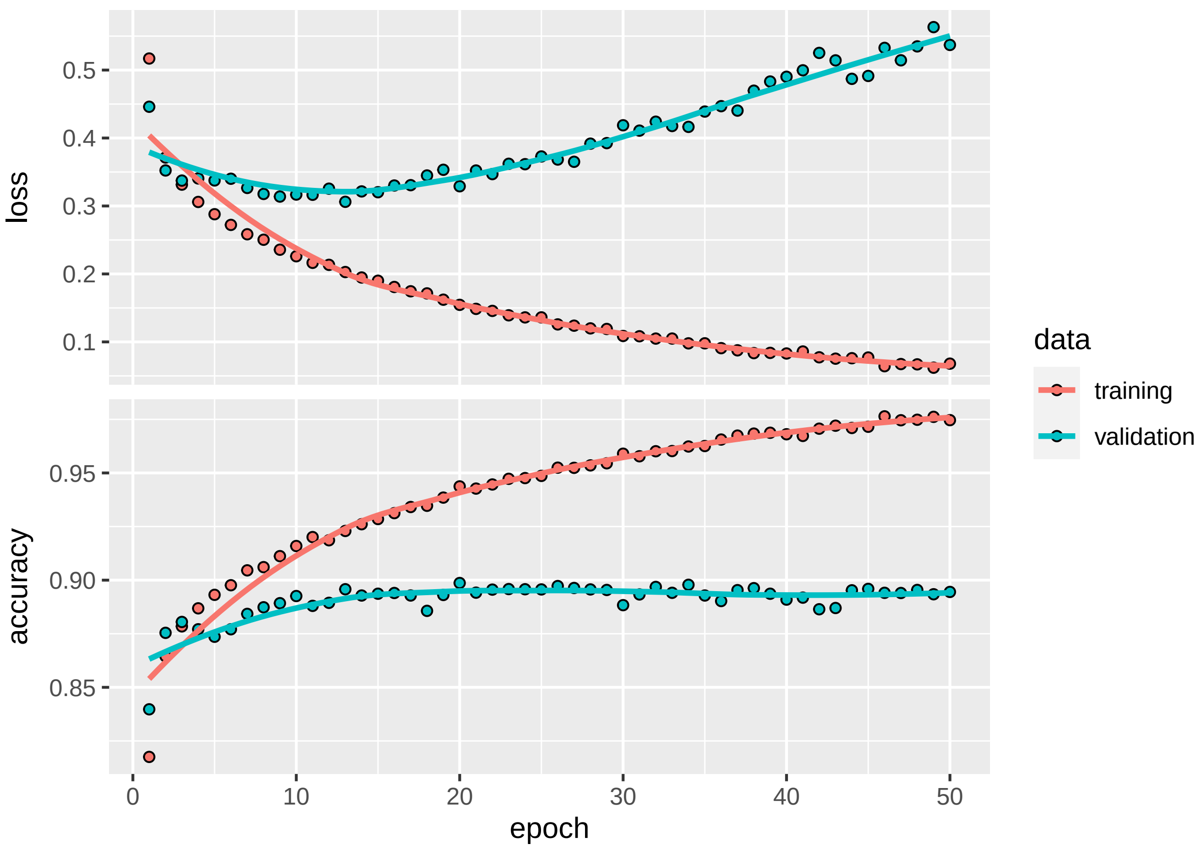
Models summary

Training and validation performance

```
plot(model_1_dnn)
```




```
plot(model_2_dnn)
```



Models evaluation

```
# Model 1 - train data set
model_1 %>% evaluate(train_images, train_labels)
```

```
##      loss  accuracy
## 0.1784874 0.9346167
```

```
# Model 1 - test data set
model_1 %>% evaluate(test_images, test_labels)
```

```
##      loss  accuracy
## 0.3120304 0.8940000
```

```
# Model 2 - train data set
model_2 %>% evaluate(train_images, train_labels)
```

```
##      loss  accuracy
## 0.1472071 0.9644000
```

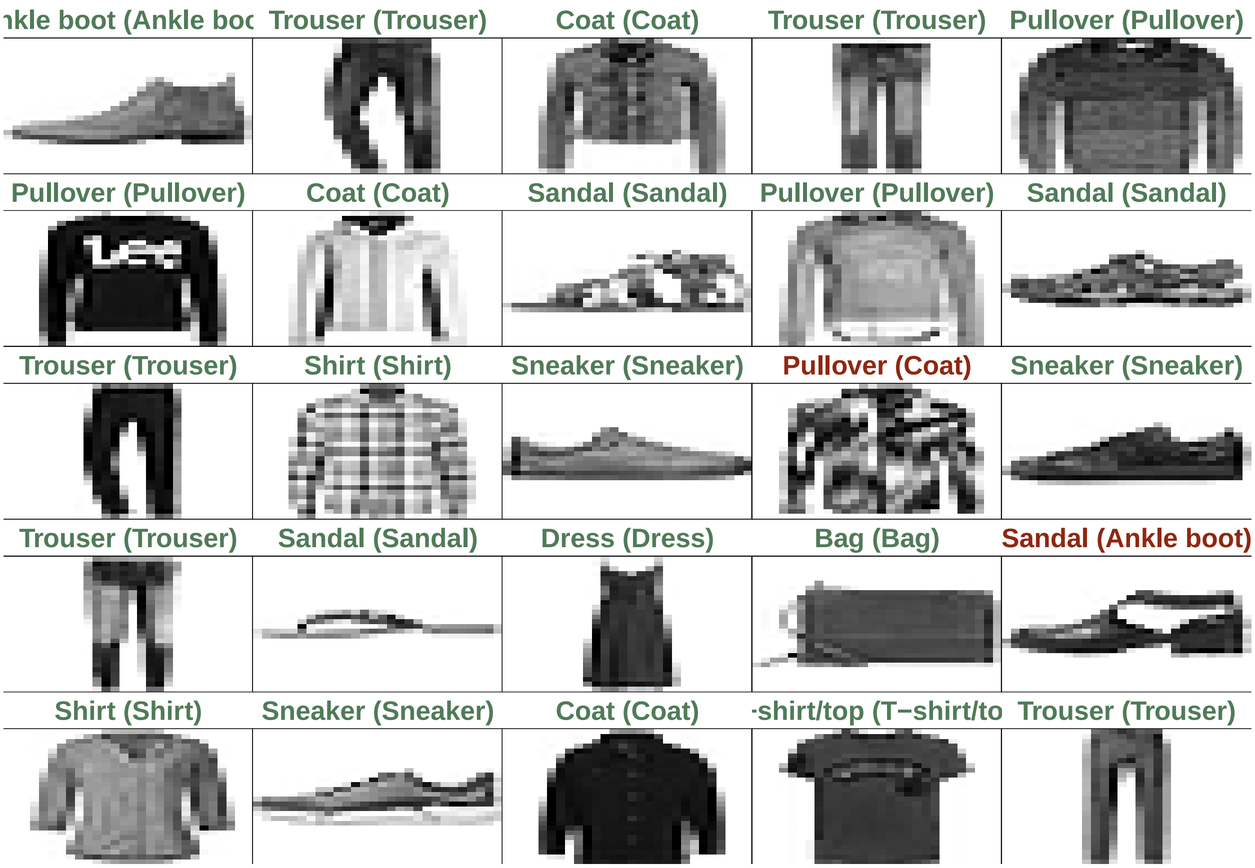
```
# Model 2 - test data set
model_2 %>% evaluate(test_images, test_labels)
```

```
##      loss  accuracy
## 0.5950837 0.8935000
```

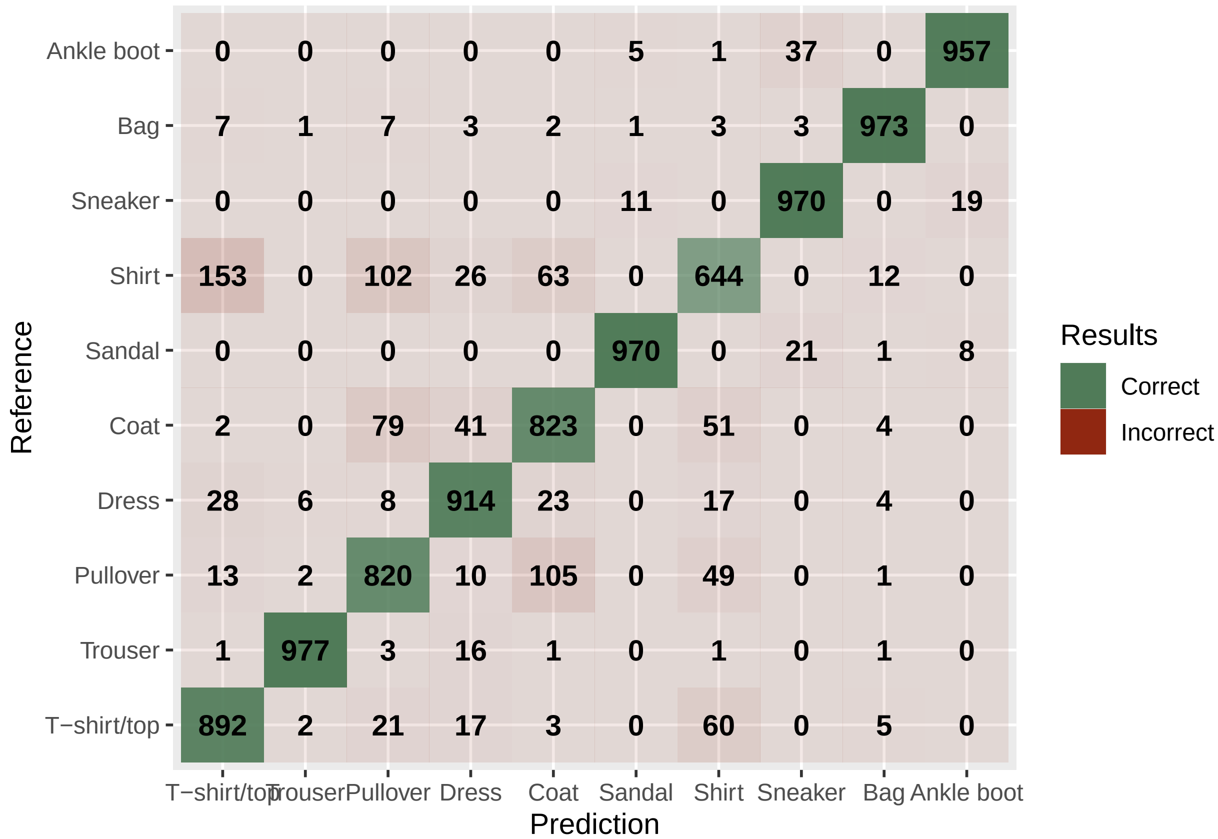
Accuracy and loss are better for **model_1**.

Prediction

```
# Predicted probabilities on test data
model_1 %>% predict(test_images) -> predictions

# Predicted classes on test data
model_1 %>% predict(test_images) %>%
  k_argmax() %>%
  as.numeric() -> predicted_clothes

# See prediction results: prediction (real)
par(mfcol = c(5, 5))
par(mar = c(0, 0, 1.5, 0), xaxs = 'i', yaxs = 'i')
for (i in 1:25) {
  img <- fashion_mnist$test$x[i, , ]
  img <- t(apply(img, 2, rev))
  if (predicted_clothes[i] == fashion_mnist$test$y[i]) {
    color <- '#507B58'
  } else {
    color <- '#902711'
  }
  image(1:28, 1:28, img, col = gray((255:0) / 255), xaxt = 'n', yaxt = 'n',
        # prediction
        main = paste0(class_names[predicted_clothes[i]+1], ' (',
        # (real)
        class_names[fashion_mnist$test$y[i]+1], ')'),
        col.main = color)
}
```




```r
# Confusion matrix
data.frame(table(predicted_clothes, fashion_mnist$test$y)) %>%
  setNames(c('Prediction', 'Reference', 'Freq')) %>%
  mutate(Results = ifelse(Prediction == Reference, 'Correct', 'Incorrect')) -> conf_table

conf_table %>%
  ggplot(aes(y = Reference, x = Prediction, fill = Results, alpha = Freq)) +
  geom_tile() +
  geom_text(aes(label = Freq), vjust = 0.5, fontface = 'bold', alpha = 1) +
  scale_fill_manual(values = c(Correct = '#507B58', Incorrect = '#902711')) +
  ylim(rev(levels(conf_table$Reference))) +
  guides(alpha = FALSE) +
  scale_x_discrete(labels=class_names) +
  scale_y_discrete(labels=class_names)
```



References

1. **Data set**: https://www.kaggle.com/datasets/zalando-research/fashionmnist