Lab 3

Thomas Haase

September 15, 2025

Table of contents

Part 1: Salary prediction

In the first part of this lab, we will work with a dataset (adults) containing information about a sample of US individuals' salaries as well as a handful sociodemographic variables. We will also consider an augmented version of this dataset (adults_aug), combining the original data with (simulated) highly nuanced lifecourse information. With this, the objective is to explore how accurately we can predict the salaries of individuals, comparing neural networks with other methods from previous weeks.

```
library(keras)
library(keras3)
library(tensorflow)

library(kableExtra)
library(tibble)

library(data.table)
```

```r
library(ggplot2)

# set wd dynamically through Rstudio API
# rstudioapi::getSourceEditorContext()$path |>
#   dirname() |>
#   setwd()

set.seed(42)
```

Task 1

1. Begin by importing adults.rds and partition the data into a training and test set. We will use the training set to fit our model, and the test set to assess accuracy and compare across models. The dataset contains cirka 50,000 individuals, and for each we have information about five traditional variables (age, education, hours worked per week, capital gain and capital loss). We are reasonably sure there are no complex interactions or non-linearities. With this as the background, do you believe a neural network model is likely to excel on this dataset? Why/why not?

```r
d_adults <- readRDS("adults.rds")


idx <- sample(seq_len(nrow(d_adults)),
              size = floor(0.8 * nrow(d_adults)))
d_adults_train <- d_adults[idx, ]
d_adults_test  <- d_adults[-idx, ]
```

I expect a neural network to be as good as logistic regression, because a neural network is particularly good at detecting complex interactions. But if a neural network is only fit to a simple dataset, it will not be able to make use of its core strength and basically regress to becoming a logistic regression or completely overfit.

Task 2

2. Begin with estimating a standard logistic regression model to the training set you just created. Hint: you may use the glm(. . . , family='binomial') function to estimate a standard logistic regression model. When estimation has finished, use the predict() function to predict the outcome on the test dataset, and calculate (report) the accuracy.

```r
model_glm <- glm(y ~ .,
           data = d_adults_train |> as.data.frame(),
           family = "binomial")

predictions_glm <- cbind(d_adults_test |> as.data.frame(),
                         predictions = ifelse(
                           predict(model_glm, as.data.frame(d_adults_test),
                                   type = "response") >= 0.5,
```

```
                            1, 0))

accuracy_glm <- (sum(predictions_glm$predictions == predictions_glm$y)/nrow(predictions_glm)) |>
```

The accuracy of the glm is 0.808.

Task 3

3. Estimate a neural network with 1 hidden layer and 5 hidden units. In the compile() function, set the optimizer to optimizer_adam(), the loss function equal to "binary_crossentropy" (as the outcome is binary), and metrics to "accuracy". Report the accuracy. Did the result match your expectations formulated in #1?

```
k_clear_session()

model_nn <- keras_model_sequential() |>
  # hidden layer
  layer_dense(units = 5, activation = "relu") |>
  # output layer for binary classification:
  layer_dense(units = 1, activation = "sigmoid")

model_nn |>
  compile(
    optimizer = optimizer_adam(1e-3),   # optimizer/learning rate
    loss = "binary_crossentropy",
    metrics = "accuracy"
  )

history_nn <- model_nn |>
  fit(
    x = subset(d_adults_train, select = -y),
    y = d_adults_train[,"y"],
    epochs = 25,
    batch_size = 32,
    validation_split = 0.2,
    verbose = 0
  )

predictions_nn <- as.numeric(model_nn |> predict(subset(d_adults_test, select = -y)))
```

```
306/306 - 0s - 1ms/step
```

```
accuracy_nn <- (sum(ifelse(predictions_nn >= 0.5,1,0) == d_adults_test[,"y"])/nrow(d_adults_test)
```

The accuracy of the neural network is 0.814. As expected the accuracy of the neural network is only as good as the accuracy of the logistic regression.

Task 4

4. Next, you shall estimate a considerably more complex neural network, containing 4 hidden layers. The first hidden layer should have 256 hidden units, the second hidden layer 128 hidden units, the third hidden layer 64 hidden units, and the fourth hidden layer 32 hidden units. Use the same settings for compile() as in #3. Report the total number of parameters and then estimate the model. Do you find that it outperforms #3 meaningfully? Why do you think this is (or is not) the case?

```
k_clear_session()

model_nn.2 <- keras_model_sequential() |>
  # hidden layers
  layer_dense(units = 256, activation = "relu") |>
  layer_dense(units = 128, activation = "relu") |>
  layer_dense(units = 64, activation = "relu") |>
  layer_dense(units = 32, activation = "relu") |>
  # output layer for binary classification:
  layer_dense(units = 1, activation = "sigmoid")

model_nn.2 |>
  compile(
    optimizer = optimizer_adam(1e-3),   # optimizer/learning rate
    loss = "binary_crossentropy",
    metrics = "accuracy"
  )

history_nn <- model_nn.2 |>
  fit(
    x = subset(d_adults_train, select = -y),
    y = d_adults_train[,"y"],
    epochs = 10,
    batch_size = 32,
    validation_split = 0.2,
    verbose = 0
  )

predictions_nn.2 <- as.numeric(model_nn.2 |> predict(subset(d_adults_test, select = -y)))
```

```
306/306 - 0s - 1ms/step
```

```
accuracy_nn.2 <- (sum(ifelse(predictions_nn.2 >= 0.5,1,0) == d_adults_test[,"y"])/nrow(d_adults_t

summary(model_nn.2)
```

```
Model: "sequential"
```

```
 Layer (type)                         Output Shape                      Param #

 dense (Dense)                        (None, 256)                         1,536

 dense_1 (Dense)                      (None, 128)                        32,896

 dense_2 (Dense)                      (None, 64)                          8,256

 dense_3 (Dense)                      (None, 32)                          2,080

 dense_4 (Dense)                      (None, 1)                              33

Total params: 134,405 (525.02 KB)
Trainable params: 44,801 (175.00 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 89,604 (350.02 KB)
```

The model was not more predictive (accuracy of 0.819) since there are no complex interactions in the data that the neurel net could pick up.

Task 5

5. Suppose now that we retrieve a dataset (adults_aug) that expands upon the original 'adults dataset. This dataset contains 6 extra variables; which capture complex aspects of the individuals life courses, with various interdependencies between them and possible non-linearities. Could this expanded data make a difference in terms of more clearly outperforming the standard logit? Investigate this by repeating steps 2-4 on the adults_aug dataset. Does your conclusion about the relevance of more complex network structure differ between the two datasets? Why?

```r
# load new data
d_adults_aug <- readRDS("adults_aug.rds")

idx <- sample(seq_len(nrow(d_adults_aug)),
              size = floor(0.8 * nrow(d_adults_aug)))
d_adults_aug_train <- d_adults_aug[idx, ]
d_adults_aug_test  <- d_adults_aug[-idx, ]


# Calculate new glm
model_aug_glm <- glm(y ~ .,
                     data = d_adults_aug_train |> as.data.frame(),
                     family = "binomial")

predictions_aug_glm <- cbind(d_adults_aug_test |> as.data.frame(),
                             predictions = ifelse(
                               predict(model_glm, as.data.frame(d_adults_aug_test),
```

```
                                        type = "response") >= 0.5,
                                1, 0))

accuracy_aug_glm <- (sum(predictions_aug_glm$predictions == predictions_aug_glm$y)/nrow(predictio


# Calculate new simple nn
k_clear_session()

model_aug_nn <- keras_model_sequential() |>
  layer_dense(units = 5, activation = "relu",
              input_shape = ncol(subset(d_adults_aug_train, select = -y))) |>
  layer_dense(units = 1, activation = "sigmoid")

model_aug_nn |>
  compile(
    optimizer = optimizer_adam(1e-3),   # optimizer/learning rate
    loss = "binary_crossentropy",
    metrics = "accuracy"
  )

history_aug_nn <- model_aug_nn |>
  fit(
    x = subset(d_adults_aug_train, select = -y),
    y = d_adults_aug_train[,"y"],
    epochs = 10,
    batch_size = 32,
    validation_split = 0.2,
    verbose = 0
  )

predictions_aug_nn <- as.numeric(model_aug_nn |> predict(subset(d_adults_aug_test, select = -y)))
```

```
306/306 - 0s - 1ms/step
```

```
accuracy_aug_nn <- (sum(ifelse(predictions_aug_nn >= 0.5,1,0) == d_adults_aug_test[,"y"])/nrow(d_


# Calculate new complex nn
k_clear_session()

model_aug_nn.2 <- keras_model_sequential() |>
  layer_dense(units = 256, activation = "relu") |>
  layer_dense(units = 128, activation = "relu") |>
  layer_dense(units = 64, activation = "relu") |>
  layer_dense(units = 32, activation = "relu") |>
  layer_dense(units = 1, activation = "sigmoid")
```

```
model_aug_nn.2 |>
  compile(
    optimizer = optimizer_adam(1e-3),   # optimizer/learning rate
    loss = "binary_crossentropy",
    metrics = "accuracy"
  )

history_aug_nn <- model_aug_nn.2 |>
  fit(
    x = subset(d_adults_aug_train, select = -y),
    y = d_adults_aug_train[,"y"],
    epochs = 10,
    batch_size = 32,
    validation_split = 0.2,
    verbose = 0
  )

predictions_aug_nn.2 <- as.numeric(model_aug_nn.2 |> predict(subset(d_adults_aug_test, select = -
```

```
306/306 - 0s - 1ms/step
```

```
accuracy_aug_nn.2 <- (sum(ifelse(predictions_aug_nn.2 >= 0.5,1,0) == d_adults_aug_test[,"y"])/nro

# make nice table
tribble(
  ~Model,       ~`Simple Data`, ~`Complex Data`,
  "glm",        accuracy_glm,   accuracy_aug_glm,
  "simple nn",  accuracy_nn,    accuracy_aug_nn,
  "complex nn", accuracy_nn.2,  accuracy_aug_nn.2
) |> kable()
```

Model	Simple Data	Complex Data
glm	0.808	0.817
simple nn	0.814	0.959
complex nn	0.819	0.997

The newly introduced data seems to introduce a lot of informations through their interactions/latent variables about the datagenerating processes. The neural networks are made for picking up on those. Especially the more complex neural network with all it's hidden layers is very good at picking those up.

Task 6

6. Given the results, you may find it unnecessary to to try further revisions. But for learning purposes, suppose we want to re-estimate the best-performing model on the adults_aug data using dropout

learning. Update the model to perform dropout for the first three hidden layers. Estimate one model where you inactivate 10% for each of the hidden layers, and a second model where you inactivate 90% per layer, and report the results. In what direction does the results change? Do you attribute this change to a change in bias or in variance and why?

```
# first nn
k_clear_session()

model_final_nn <- keras_model_sequential() |>
  layer_dropout(rate = 0.1) |>
  layer_dense(units = 256, activation = "relu",
              input_shape = ncol(subset(d_adults_aug_train, select = -y))) |>
  layer_dropout(rate = 0.1) |>
  layer_dense(units = 128, activation = "relu",
              input_shape = ncol(subset(d_adults_aug_train, select = -y))) |>
  layer_dropout(rate = 0.1) |>
  layer_dense(units = 64, activation = "relu",
              input_shape = ncol(subset(d_adults_aug_train, select = -y))) |>
  layer_dense(units = 32, activation = "relu",
              input_shape = ncol(subset(d_adults_aug_train, select = -y))) |>
  layer_dense(units = 1, activation = "sigmoid")

model_final_nn |>
  compile(
    optimizer = optimizer_adam(1e-3),   # optimizer/learning rate
    loss = "binary_crossentropy",
    metrics = "accuracy"
  )

history_final_nn <- model_final_nn |>
  fit(
    x = subset(d_adults_aug_train, select = -y),
    y = d_adults_aug_train[,"y"],
    epochs = 10,
    batch_size = 32,
    validation_split = 0.2,
    verbose = 0
  )

predictions_final_nn <- as.numeric(model_final_nn |> predict(subset(d_adults_aug_test, select = -
```

```
306/306 - 0s - 1ms/step
```

```
accuracy_aug_nn <- (sum(ifelse(predictions_final_nn >= 0.5,1,0) == d_adults_aug_test[,"y"])/
                      nrow(d_adults_aug_test)) |> round(3)

# second nn
```

```
k_clear_session()

model_final_nn.2 <- keras_model_sequential() |>
  layer_dropout(rate = 0.9) |>
  layer_dense(units = 256, activation = "relu") |>
  layer_dropout(rate = 0.9) |>
  layer_dense(units = 128, activation = "relu") |>
  layer_dropout(rate = 0.9) |>
  layer_dense(units = 64, activation = "relu") |>
  layer_dense(units = 32, activation = "relu") |>
  layer_dense(units = 1, activation = "sigmoid")

model_final_nn.2 |>
  compile(
    optimizer = optimizer_adam(1e-3),   # optimizer/learning rate
    loss = "binary_crossentropy",
    metrics = "accuracy"
  )

history_final_nn.2 <- model_final_nn.2 |>
  fit(
    x = subset(d_adults_aug_train, select = -y),
    y = d_adults_aug_train[,"y"],
    epochs = 10,
    batch_size = 32,
    validation_split = 0.2,
    verbose = 0
  )

predictions_final_nn.2 <- as.numeric(model_final_nn.2 |> predict(subset(d_adults_aug_test, select
```

```
306/306 - 0s - 1ms/step
```

```
accuracy_aug_nn.2 <- (sum(ifelse(predictions_final_nn.2 >= 0.5,1,0) == d_adults_aug_test[,"y"])/
                        nrow(d_adults_aug_test)) |> round(3)

# make nice table
tribble(
  ~Model,       ~`Accuracy`,
  "10% Dropout",  accuracy_aug_nn,
  "90% Dropout",  accuracy_aug_nn.2) |> kable()
```

Model	Accuracy
10% Dropout	0.993
90% Dropout	0.764

The 10% dropout rate reduced variance in the model, making it slightly less overfit while maintaining nearly the same accuracy. However, when the dropout rate is increased to 90%, the model lacks sufficient information to capture complex interactions. This leads to higher bias and a poorer representation of the data-yet it still manages to reach 75% accuracy, raising the question of whether the simplification might actually make the model perform better in some respects.

Part 2: image prediction

Task 1

1. Begin by importing the datafiles "fashion_2016_train.rds" and "fashion_2016_test.rds".

```
rm(list = ls())

d_fashion_train <- readRDS("fashion_2016_train.rds")
d_fashion_test <- readRDS("fashion_2016_test.rds")
```

Task 2

2. Next, estimate a simple convolutional neural network with $K = 1$ convolutional layers and $M = 1$ regular type of (fully connected) hidden layers. Specify the number of filters (in the convolutional layer) and the number of hidden units (in the fully connected / regular hidden slides) to be 8. Remember also that included after each added convolutional layer, pooling (max-pooling, $2 * 2$) should to be applied. Finally, include `layer_dense(units = 10, activation = "softmax")` at the end. When compiling the model using the compile() function, set loss = "sparse_categorical_crossentroy". Report the results, and reflect on whether you think improvement can be made by increasing either M, K, or the number of filters or hidden units in the regular hidden layer.

```
inspect_training_image <- function(ITEMNUMBER){

  label <- d_fashion_train$fashion_labels[d_fashion_train$labels[ITEMNUMBER] + 1]

  par(mar = c(0,0,0,0))
  img <- d_fashion_train$images[ITEMNUMBER,1:28,1:28,1]
  image(t(apply(img, 2, rev)), col = gray.colors(256), axes = FALSE)

  return(list(image = img, label = label))
}

inspect_test_image <- function(ITEMNUMBER){

  label <- d_fashion_test$fashion_labels[d_fashion_test$labels[ITEMNUMBER] + 1]

  par(mar = c(0,0,0,0))
```

```
  img <- d_fashion_test$images[ITEMNUMBER,1:28,1:28,1]
  image(t(apply(img, 2, rev)), col = gray.colors(256), axes = FALSE)

  return(list(image = img, label = label))
}
```

```
k_clear_session()

model_cnn <- keras_model_sequential() |>
  layer_conv_2d(
    filters = 8,          # number of feature maps learned
    kernel_size = c(3,3), # size of the filters
    activation = "relu",  # nonlinearity after convolution
    # dimensions of input images (H, W, C)
    input_shape = c(28,28,1)
    ) |>
  # downsample by taking local maxima over 2x2 windows
  layer_max_pooling_2d(pool_size = c(2,2)) |>
  layer_flatten() |> # convert 2D feature maps -> 1D vector for Dense layer
  # Dense hidden layer after convolution (feature combination step)
  layer_dense(units = 8, activation = "relu") |>
  # Output Layer: units: classes, softmax probabilities sum to 1
  layer_dense(units = 10, activation = "softmax")

model_cnn |>
  compile(
    optimizer = optimizer_adam(), # adaptive learning rate
    loss      = "sparse_categorical_crossentropy",
    metrics   = "accuracy"
  )

history_cnn <- model_cnn |>
  fit(
    x = d_fashion_train$images,
    y = d_fashion_train$labels,
    epochs = 5,
    batch_size = 128, # how many observations to process before updating params
    validation_split = 0.1, # hold out part of training for monitoring
    verbose = 0
  )

# Evaluate on test set
eval_model_cnn <-  model_cnn |>
  evaluate(d_fashion_test$images, d_fashion_test$labels, verbose = 0)

tribble(
  ~Parameter,  ~`Value`,
```

```
  "Accuracy",  eval_model_cnn$accuracy |> round(3),
  "Loss",      eval_model_cnn$loss |> round(3)) |> kable()
```

Parameter	Value
Accuracy	0.851
Loss	0.435

The M stands for the amount of convolutional layers in the model, while the K notates the number of regular hidden layers in the model. A higher number of convolutional layers would pick up on more complex features of the image, while a larger K results in a more complex classifier. Since the image is so small a second convolutional layer would probably pick up on more details, which would improve performance but a stronger classifier (more regular layers and higher M) would probably lead to a higher improvement in accuracy since only one hidden layer is unable to pick up on any interactions. The larger the count of filters, the more features of similar size the neuralnet looks for. The larger the filters the larger the features a filter picks up in an image. I expect that a neuralnet works best when the amount of units in the hidden layer are of similar size to the number of filters that are fed into the hidden layer.

Task 3

3. Next, you shall estimate a slightly more complex convolutional neural network, increasing the number of filters to 32 in the first layer, and 64 in the second convolutional layer. Similarly, increase the number of hidden units in the first regular/fully-connected hidden layer to 64, and the second to 32. Report the results. Does this slightly more complicated model improve/degrade upon the simple one in #2? Speculate why in terms of the bias/variance trade-off. Report and contrast also the number of parameters between the two.

```
k_clear_session()

model_cnn2 <- keras_model_sequential() |>
  layer_conv_2d(
    filters = 32,
    kernel_size = c(3,3),
    activation = "relu",
    input_shape = c(28,28,1)
    ) |>
  layer_max_pooling_2d(pool_size = c(2,2)) |>
  layer_conv_2d(
    filters = 64,
    kernel_size = c(3,3),
    activation = "relu",
    input_shape = c(28,28,1)
    ) |>
  layer_max_pooling_2d(pool_size = c(2,2)) |>
  layer_flatten() |>
```

```
  layer_dense(units = 64, activation = "relu") |>
  layer_dense(units = 32, activation = "relu") |>
  layer_dense(units = 10, activation = "softmax")

model_cnn2 |>
  compile(
    optimizer = optimizer_adam(), # adaptive learning rate
    loss      = "sparse_categorical_crossentropy",
    metrics   = "accuracy"
  )

history_cnn2 <- model_cnn2 |>
  fit(
    x = d_fashion_train$images,
    y = d_fashion_train$labels,
    epochs = 3,
    batch_size = 128, # how many observations to process before updating params
    validation_split = 0.1, # hold out part of training for monitoring
    verbose = 0
  )

# Evaluate on test set
eval_model_cnn2 <-  model_cnn2 |>
  evaluate(d_fashion_test$images, d_fashion_test$labels, verbose = 0)

tribble(
  ~Parameter,  ~`Value`,
  "Accuracy",  eval_model_cnn2$accuracy |> round(3),
  "Loss",      eval_model_cnn2$loss |> round(3)) |> kable()
```

Parameter	Value
Accuracy	0.870
Loss	0.367

```
summary(model_cnn)
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 8)	80
max_pooling2d (MaxPooling2D)	(None, 13, 13, 8)	0
flatten (Flatten)	(None, 1352)	0

```
 dense (Dense)                       (None, 8)                         10,824

 dense_1 (Dense)                     (None, 10)                           90

Total params: 32,984 (128.85 KB)
Trainable params: 10,994 (42.95 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 21,990 (85.90 KB)
```

```
summary(model_cnn2)
```

```
Model: "sequential"

 Layer (type)                        Output Shape                      Param #

 conv2d (Conv2D)                     (None, 26, 26, 32)                   320

 max_pooling2d (MaxPooling2D)        (None, 13, 13, 32)                     0

 conv2d_1 (Conv2D)                   (None, 11, 11, 64)                18,496

 max_pooling2d_1 (MaxPooling2D)      (None, 5, 5, 64)                       0

 flatten (Flatten)                   (None, 1600)                           0

 dense (Dense)                       (None, 64)                       102,464

 dense_1 (Dense)                     (None, 32)                         2,080

 dense_2 (Dense)                     (None, 10)                           330

Total params: 371,072 (1.42 MB)
Trainable params: 123,690 (483.16 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 247,382 (966.34 KB)
```

The simpler model has fewer parameters, which reduces variance but increases bias. The complex model reduces bias (it can capture more structure), but variance risk increases. Since you only added a moderate amount of complexity, test accuracy improved slightly, suggesting the extra capacity helped reduce bias without yet overfitting. The first CNN has 32,984 total parameters and the second CNN has 371,072 total parameters.

Task 4

4. In #2-3, we used max-pooling of 2*2. Why is it generally not a good idea to increase the dimension of the pooling to become large relative to the images? Would this increase bias or variance?

Pooling reduces the dimensionality of feature maps, and increases location invariance. This puts the emphasis on the presence of signal rather than the location of it. For example "maximum pooling" keeps largest value within $k \times k$ patches (here k = 2). The larger the dimension of the pooling window size, the more the spatial information is dilluted which makes the model more biased leading to decreased variance.

Task 5

5. Because we are interested in using the predictions from these models for downstream analysis (of trends in consumer polarization), it is extra important to validate the measure. This is what you shall do now, focusing on the CNN model you deemed the best thus far. In particular you should break down the accuracy per item category. Is there meaninful difference in predictability beween item classes? Does the ordering make substantive sense? Hint: to compute accuracy by group, you may use the following code:

```
accuracy_by_class_dt <- data.table(actual = d_fashion_train$labels,
                                   predicted = apply(predict(model_cnn2, d_fashion_train$images),
```

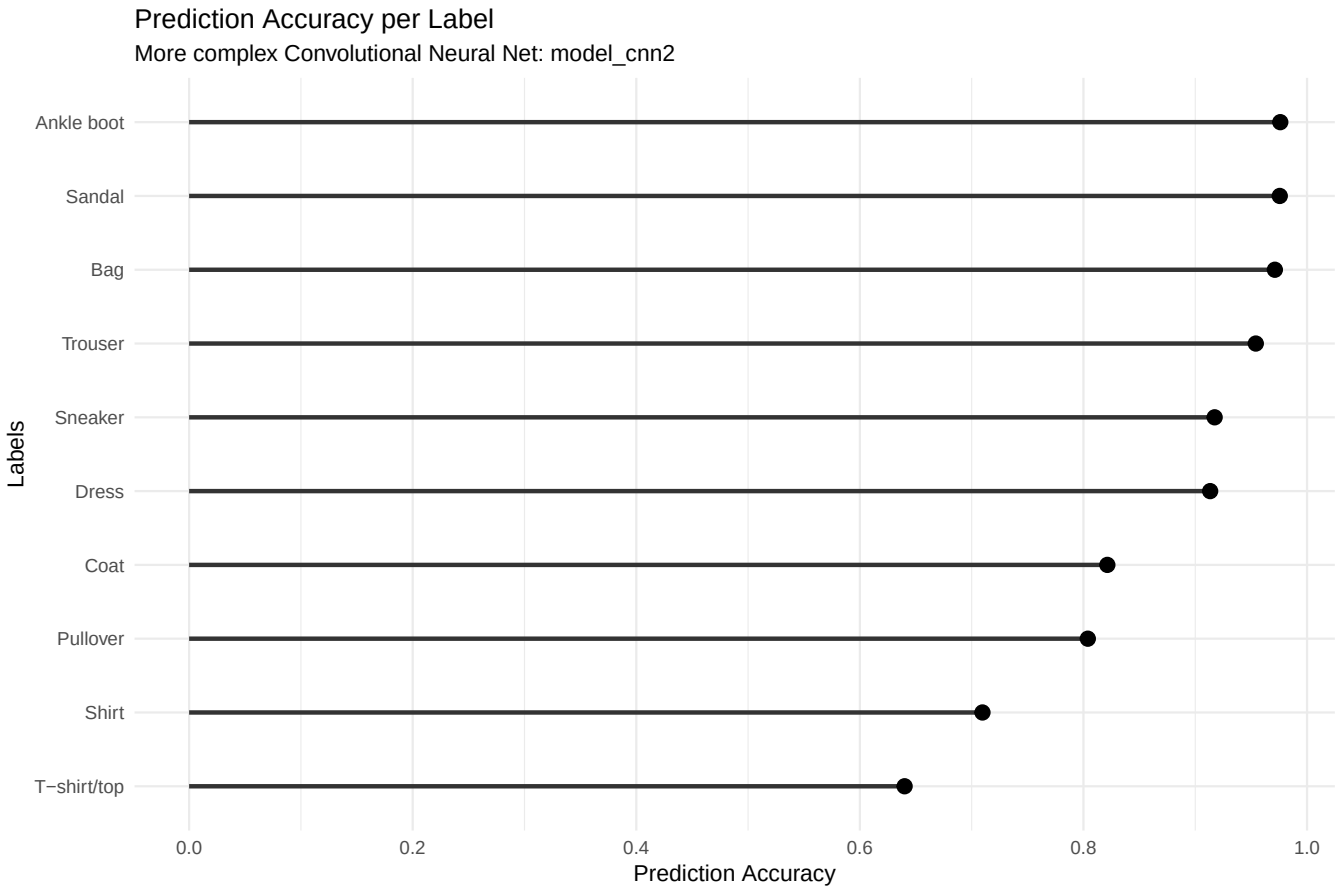
```
1155/1155 - 3s - 3ms/step
```

```
accuracy_by_class_dt <- merge(x = accuracy_by_class_dt,
                              y = data.table(label = d_fashion_train$fashion_labels,
                                             number = 0:9),
                              by.x='actual',by.y='number')

accuracy_by_class_dt[,.(accuracy=mean(correct)),by=label][order(accuracy,decreasing = T)] |>
  kable()
```

label	accuracy
Ankle boot	0.9760544
Sandal	0.9755899
Bag	0.9712071
Trouser	0.9541532
Sneaker	0.9173420
Dress	0.9133244
Coat	0.8213608
Pullover	0.8038902
Shirt	0.7096515
T-shirt/top	0.6400108

```
accuracy_by_class_dt[,.(accuracy=mean(correct)),by=label][order(accuracy,decreasing = T)] |>
  ggplot(aes(y=reorder(label, accuracy),x = accuracy)) +
  geom_segment(aes(x = 0, xend = accuracy,
                   y = label, yend = label),
```

```
              color = "#333333",
              linewidth = 1) +
  geom_point(size=3) +
  labs(title = "Prediction Accuracy per Label",
       subtitle = "More complex Convolutional Neural Net: model_cnn2",
       x = "Prediction Accuracy",
       y = "Labels") +
  scale_x_continuous(breaks = seq(0,1,0.2)) +
  theme_minimal()
```



Most productimages can be classified with a high accuracy by the convolutional neural network. The Shirt stands out as the product which can not be identified from an image with an accuracy of less than 50%.

```
# inspect random tshirt
inspect_test_image(sample(which(d_fashion_test$labels == 6),1))$image

# inspect random Ankle boot
inspect_test_image(sample(which(d_fashion_test$labels == 9),1))$image
```

After investigating a few images from the best and worst predicted label the differences in prediction accuracy most probably result from the variability of pictures within a certain label. Most ankle boots have a very similar shape and are often of light color, while the shirts have very different shapes and colors. Maybe adding higher differentiation of different shirts could improve the prediction accuracy (adding more labels).

Task 6

6. Next, we shall finally adress the question we set out to answer: how patterns in consumer behavior changed between 2016 to 2017. To answer this question, please follow these steps:

- Import the fashion2016_2017_unlabeled.rds file, which contains all images and the sociodemographic info of the individual associated with the image. Note: the data has already been preprocced (normalized pixel values, etc.).

```
rm(list = setdiff(ls(), c("eval_model_cnn2", "history_cnn2", "model_cnn2")))

d <- readRDS("fashion_2015_2016_unlabeled.rds")
```

- Use the predict() function to predict for this dataset based on the CNN model you thought were the best. Note that, when you use the predict() function for a model where you have multiple-category outcome variables, like here, each observation get a probability over the items. To assign the prediction to the maximum value, you may use this code: `apply(preds_2016_2017, 1, which.max) - 1`

```
preds <- predict(model_cnn2, d$images)
```

```
2188/2188 - 7s - 3ms/step
```

```
predicted_labels <- c(apply(preds, 1, which.max) - 1)
rm(preds)
```

- Then you can simply merge the predicted category with the demographic variable data.frame, and calculate various associations by classical statistical means.

```
df <- cbind(d$demographics, predicted_labels)

labels_map <- data.frame(
  label = c("T-shirt/top", "Trouser", "Pullover", "Dress", "Coat",
            "Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot"),
  number = 0:9
  )

df <- merge(df, labels_map,
            by.x = "predicted_labels",
            by.y = "number",
```

```
            all.x = TRUE)

rm(d,eval_model_cnn2,history_cnn2,labels_map,model_cnn2,predicted_labels)

saveRDS(df, file = "final_dataset.rds")
```

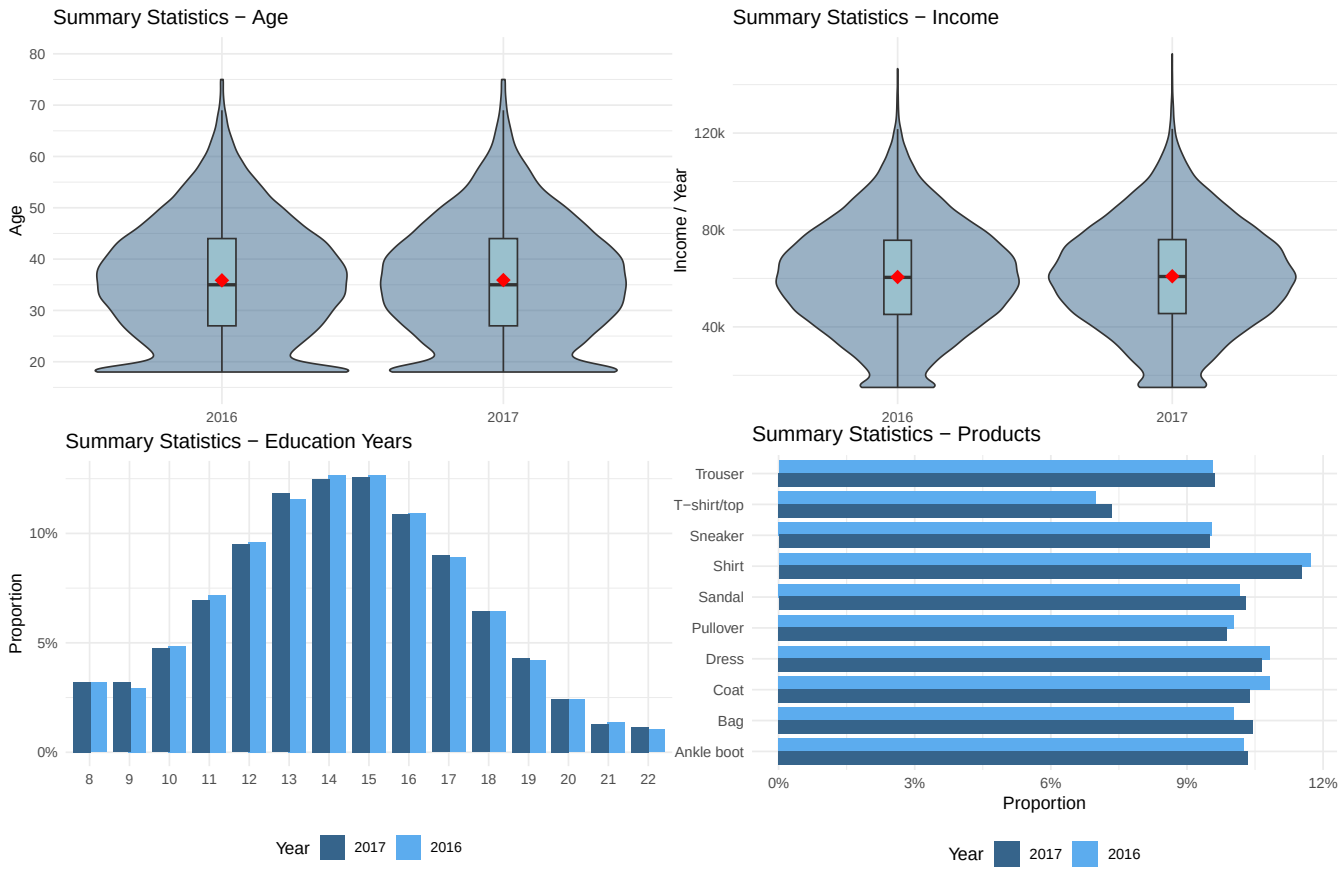
```
library(tidyverse)
library(kableExtra)

d <- readRDS("final_dataset.rds") |>
  subset(select = -c(predicted_labels))

d$year |> table() |> kable(col.names = c("Year", "Observations"))
```

Year	Observations
2016	46200
2017	23800

There are almost twice as many observations of customer behaviour for the year 2016 compared to 2017.

```
ggplot(d, aes(x = factor(year), y = age)) +
  geom_violin(fill = "steelblue4", alpha = 0.5) +
  geom_boxplot(width = 0.1, outlier.shape = NA, fill = "lightblue3") +
  stat_summary(fun = mean, geom = "point", shape = 18, color = "red", size = 4) +
  labs(title = "Summary Statistics - Age", x = "", y = "Age") +
  scale_y_continuous(breaks = seq(0,80,10), limits = c(15,80)) +
  theme_minimal()
ggplot(d, aes(x = factor(year), y = income)) +
  geom_violin(fill = "steelblue4", alpha = 0.5) +
  geom_boxplot(width = 0.1, outlier.shape = NA, fill = "lightblue3") +
  stat_summary(fun = mean, geom = "point", shape = 18, color = "red", size = 4) +
  labs(title = "Summary Statistics - Income", x = "", y = "Income / Year") +
  scale_y_continuous(labels = function(x) paste0(x/1000, "k")) +
  theme_minimal()
d |>
  count(year, education_years) |>
  group_by(year) |>
  mutate(prop = (n / sum(n)))  |>
  ungroup() |>
ggplot(aes(x = factor(education_years), y = prop,
           fill = factor(year, levels = c("2017","2016")))) +
  geom_bar(stat = "identity", position = position_dodge(width = 0.8)) +
  scale_y_continuous(labels = scales::percent) +
  labs(title = "Summary Statistics - Education Years",
       x = "", y = "Proportion", fill = "Year") +
```

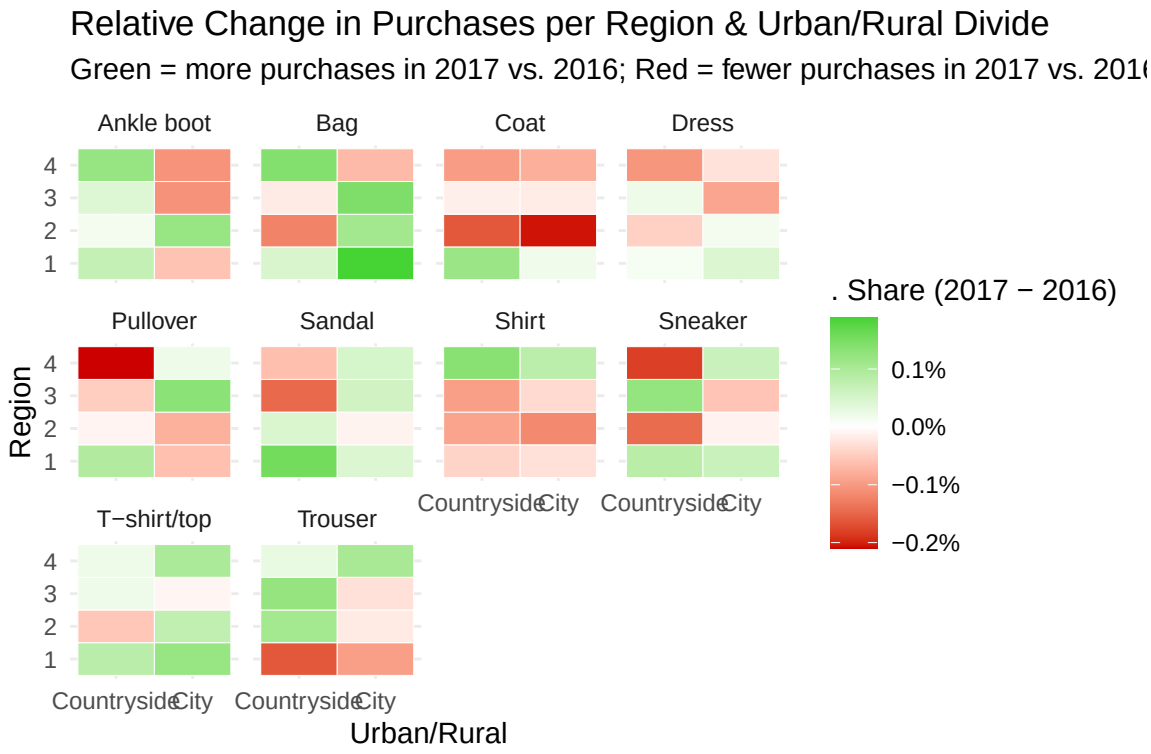
```
  scale_fill_manual(values = c("2016" = "steelblue2",
                               "2017" = "steelblue4")) +
  theme_minimal() +
  theme(legend.position = "bottom")
d %>%
  count(year, label) |>
  group_by(year) |>
  mutate(prop = n / sum(n)) |>
  ungroup() |>
ggplot(aes(x = factor(label), y = prop,
           fill = factor(year, levels = c("2017","2016")))) +
  geom_bar(stat = "identity", position = position_dodge(width = 0.8)) +
  scale_y_continuous(labels = scales::percent) +
  labs(title = "Summary Statistics - Products",
       x = "", y = "Proportion", fill = "Year") +
  coord_flip() +
  scale_fill_manual(values = c("2016" = "steelblue2",
                               "2017" = "steelblue4")) +
  theme_minimal() +
  theme(legend.position = "bottom")
```



Across all background variables - age, income, education, and product preferences - the distributions

in 2016 and 2017 are strikingly similar. While total purchases dropped substantially, the underlying customer profile and shopping patterns remained essentially unchanged, with only tiny relative shifts that carry little substantive meaning.

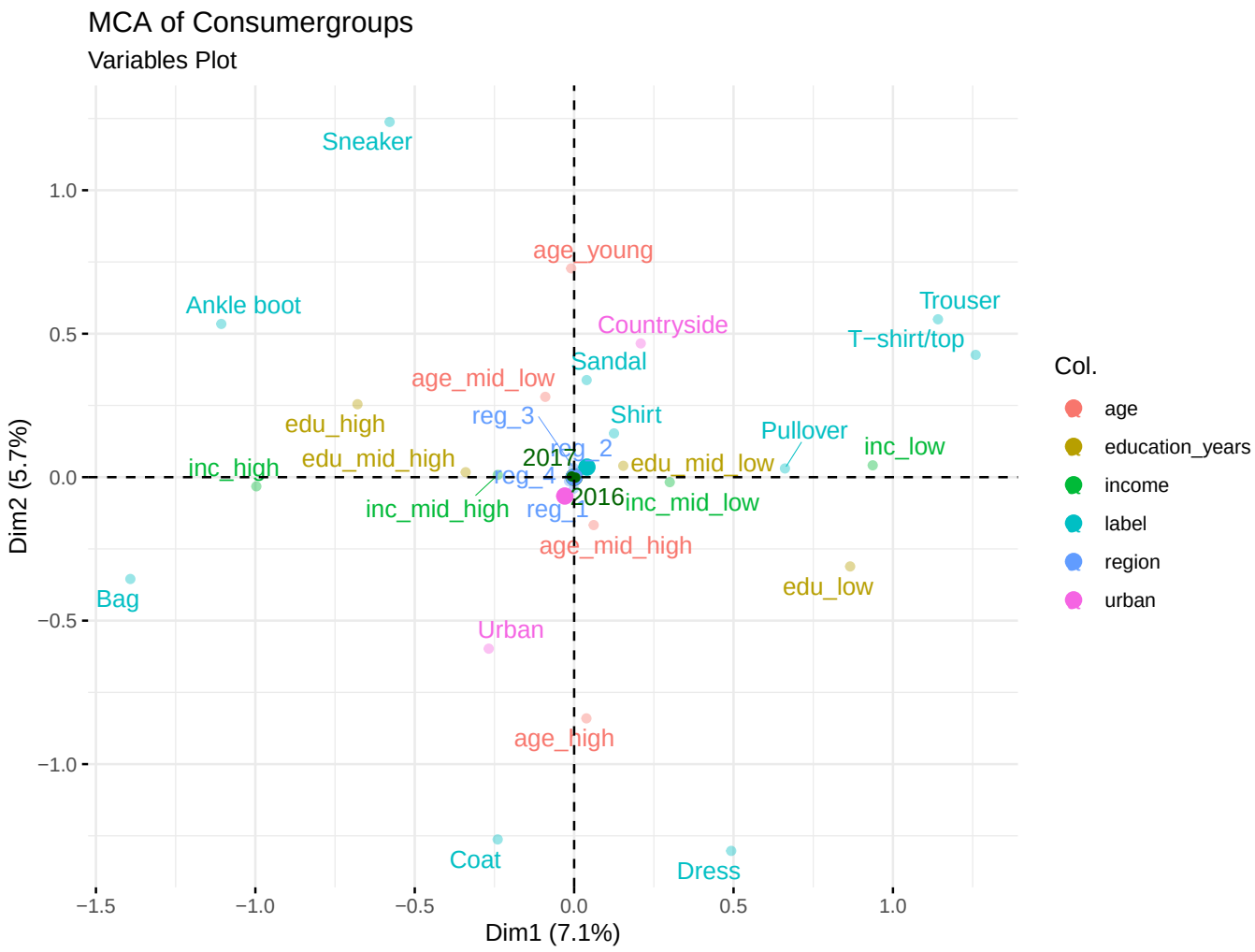
```
d |>
  count(year, label, region, urban) |>
  group_by(year) |>
  mutate(prop = n / sum(n)) |>
  ungroup() |>
  select(-n) |>
  pivot_wider(names_from = year, values_from = prop, values_fill = 0) |>
  mutate(diff = `2017` - `2016`) |>
  ggplot(aes(
    x = factor(urban, levels = c(0,1), labels = c("Countryside","City")),
    y = factor(region),
    fill = diff
  )) +
  geom_tile(color = "white") +
  scale_fill_gradient2(low = "red3", high = "green3",
                       labels = scales::percent) +
  facet_wrap(~ label) +
  labs(title = "Relative Change in Purchases per Region & Urban/Rural Divide",
       subtitle = "Green = more purchases in 2017 vs. 2016; Red = fewer purchases in 2017 vs. 201
       x = "Urban/Rural", y = "Region",
       fill = "Δ Share (2017 - 2016)") +
  theme_minimal()
```



Overall, the relative changes in purchase shares between 2016 and 2017 are very small, with most shifts staying well below a few percentage points. While some categories and regions show slight increases or decreases, the patterns suggest that the urban-rural divide and regional variation remained largely stable across the two years.

```
library(FactoMineR)
library(factoextra)
library(explor)

d_mca <- d |>
  select(-ses_score) |>
  mutate(
    age = case_when(ntile(age, 4) == 1 ~ "age_young",
                    ntile(age, 4) == 2 ~ "age_mid_low",
                    ntile(age, 4) == 3 ~ "age_mid_high",
                    ntile(age, 4) == 4 ~ "age_high"),

    income = case_when(ntile(income, 4) == 1 ~ "inc_low",
                       ntile(income, 4) == 2 ~ "inc_mid_low",
                       ntile(income, 4) == 3 ~ "inc_mid_high",
                       ntile(income, 4) == 4 ~ "inc_high"),

    education_years = case_when(ntile(education_years, 4) == 1 ~ "edu_low",
                                ntile(education_years, 4) == 2 ~ "edu_mid_low",
                                ntile(education_years, 4) == 3 ~ "edu_mid_high",
                                ntile(education_years, 4) == 4 ~ "edu_high"),

    urban = case_when(urban == 0 ~ "Countryside",
                      urban == 1 ~ "Urban"),

    region = case_when(region == 1 ~ "reg_1",
                       region == 2 ~ "reg_2",
                       region == 3 ~ "reg_3",
                       region == 4 ~ "reg_4"),

    label = label,
    year = as.character(year))

mca <- MCA(d_mca,
           quanti.sup = c(),
           quali.sup = c("year"),
           graph = FALSE)

fviz_mca_var(mca, title = "MCA of Consumergroups",
             repel = T, shape.var = 19, alpha.var = 0.4,
             subtitle = "Variables Plot",
             col.var = c(rep("age",4), rep("income",4),
                         rep("education_years",4),
```

```
                    rep("urban",2), rep("region",4),
                    rep("label",10)))
```



The MCA was calculated with the years as a supporting variable (left out of estimation but still plotted). Both axis explain very little variance in the data. The first axis is made up of the distincion between los and high income. The second axis is mainly driven by the age variable. perpendicular to both (like a starshape) it is possible to distinguish a countryside-urban axis and an education axis. The regions and years seem to have no impact on the consumer choices. We can see that young customers with low income buy mainly trousers and tshirts. Young customers with higher income start buying more shoes and at some point ankleboots instead of sneakers. Old customers with low income like to buy dresses while high age customers with high income buy coats. High income is generally connected to buying Bags while low income is generally connected to buying Pullovers.

The main takeaway is that the consumerbehaviour does not change through years but rather is driven by age and income.