Day12 exercise solutions

Ali Movasati, Isabelle Caroline Rose Cretton, Tristan Koning

Dec. 06th, 2024

```
# Set global code chunk options
knitr::opts_chunk$set(warning = FALSE)
```

```
# load required libraries
library("extremefit")
library("extRemes")
```

```
## Loading required package: Lmoments
```

```
## Loading required package: distillery
```

```
##
## Attaching package: 'extRemes'
```

```
## The following objects are masked from 'package:stats':
##
##     qqnorm, qqplot
```

```
library("ismev")
```

```
## Loading required package: mgcv
```

```
## Loading required package: nlme
```

```
## This is mgcv 1.9-1. For overview type 'help("mgcv-package")'.
```

```
library("skimr")
library("dplyr")
```

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

```
## The following object is masked from 'package:nlme':
##
##     collapse
```

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

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

```
library("tidyr")
library("magrittr")
```

```
##
## Attaching package: 'magrittr'
```

```
## The following object is masked from 'package:tidyr':
##
##     extract
```

```
library("ggplot2")
library("lubridate")
```

```
##
## Attaching package: 'lubridate'
```

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

```
library("fields")
```

```
## Loading required package: spam
```

```
## Spam version 2.10-0 (2023-10-23) is loaded.
## Type 'help( Spam)' or 'demo( spam)' for a short introduction
## and overview of this package.
## Help for individual functions is also obtained by adding the
## suffix '.spam' to the function name, e.g. 'help( chol.spam)'.
```

```
##
## Attaching package: 'spam'
```

```
## The following objects are masked from 'package:base':
##
##     backsolve, forwardsolve
```

```
## Loading required package: viridisLite
```

```
##
## Try help(fields) to get started.
```

```
library("keras3")
library("nnet")
```

```
##
## Attaching package: 'nnet'
```

```
## The following object is masked from 'package:mgcv':
##
##     multinom
```

```
# define functions
`%notin%` <- Negate(`%in%`)
```

Exercise 1

(a)

```
data("iris")
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
iris_subset <- subset(iris, Species %in% c("versicolor", "virginica"))
iris_subset$Species <- as.numeric(iris_subset$Species) - 2  # Convert to binary (0 and 1)
glm_model <- glm(Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width, data = iris_subset,
summary(glm_model)
```

```
##
## Call:
## glm(formula = Species ~ Sepal.Length + Sepal.Width + Petal.Length +
##     Petal.Width, family = binomial, data = iris_subset)
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -42.638     25.707  -1.659   0.0972 .
## Sepal.Length   -2.465      2.394  -1.030   0.3032
```

```
## Sepal.Width    -6.681      4.480  -1.491   0.1359
## Petal.Length    9.429      4.737   1.991   0.0465 *
## Petal.Width    18.286      9.743   1.877   0.0605 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 138.629  on 99  degrees of freedom
## Residual deviance:  11.899  on 95  degrees of freedom
## AIC: 21.899
##
## Number of Fisher Scoring iterations: 10
```

(b)

```
perceptron_model <- nnet(Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width,
                         data = iris_subset,
                         act.fct = "logistic",
                         size = 1,
                         linout = FALSE)
```

```
## # weights:  7
## initial  value 26.174550
## iter  10 value 18.052116
## iter  20 value 1.111160
## iter  30 value 1.001341
## iter  40 value 1.000245
## iter  50 value 1.000149
## iter  60 value 1.000136
## iter  70 value 1.000128
## final  value 1.000127
## converged
```

```
summary(perceptron_model)
```

```
## a 4-1-1 network with 7 weights
## options were -
##  b->h1 i1->h1 i2->h1 i3->h1 i4->h1
## -39.22 -11.73 -11.63  23.12  19.18
##   b->o  h1->o
## -23.63  39.69
```

```
perceptron_weights <- perceptron_model$wts
glm_weights <- coef(glm_model)
glm_weights
```

```
##  (Intercept) Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##   -42.637804    -2.465220    -6.680887     9.429385    18.286137
```

```
perceptron_weights
```

```
## [1] -39.22328 -11.73436 -11.62774  23.12427  19.17762 -23.63128  39.69184
```

```
# TODO: compare weights
```

(c)

```
reticulate::py_install("keras")
```

```
## Using virtual environment "C:/Users/Tristan/Documents/.virtualenvs/r-reticulate" ...
```

```
## + "C:/Users/Tristan/Documents/.virtualenvs/r-reticulate/Scripts/python.exe" -m pip install --upgrade
```

```
reticulate::py_install("tensorflow")
```

```
## Using virtual environment "C:/Users/Tristan/Documents/.virtualenvs/r-reticulate" ...
```

```
## + "C:/Users/Tristan/Documents/.virtualenvs/r-reticulate/Scripts/python.exe" -m pip install --upgrade
```

```
# Training and testing sets
set.seed(123)
train_indices <- sample(1:nrow(iris_subset), 0.7*nrow(iris_subset))
train_data <- iris_subset[train_indices,]
test_data <- iris_subset[-train_indices,]

# Model
dnn_model <- keras_model_sequential() %>%
  layer_dense(units = 64, activation = "relu", input_shape = 4) %>%
  layer_dense(units = 64, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")

dnn_model %>% compile(
    optimizer = 'adam',
    loss = 'binary_crossentropy',
    metrics = c('accuracy')
)

# Train
history <- dnn_model %>% fit(
    as.matrix(train_data[, -5]), train_data$Species,
    epochs = 100,
    batch_size = 10,
    validation_split = 0.2
)
```

```
## Epoch 1/100
## 6/6 - 1s - 142ms/step - accuracy: 0.5714 - loss: 0.7282 - val_accuracy: 0.5000 - val_loss: 0.6602
```

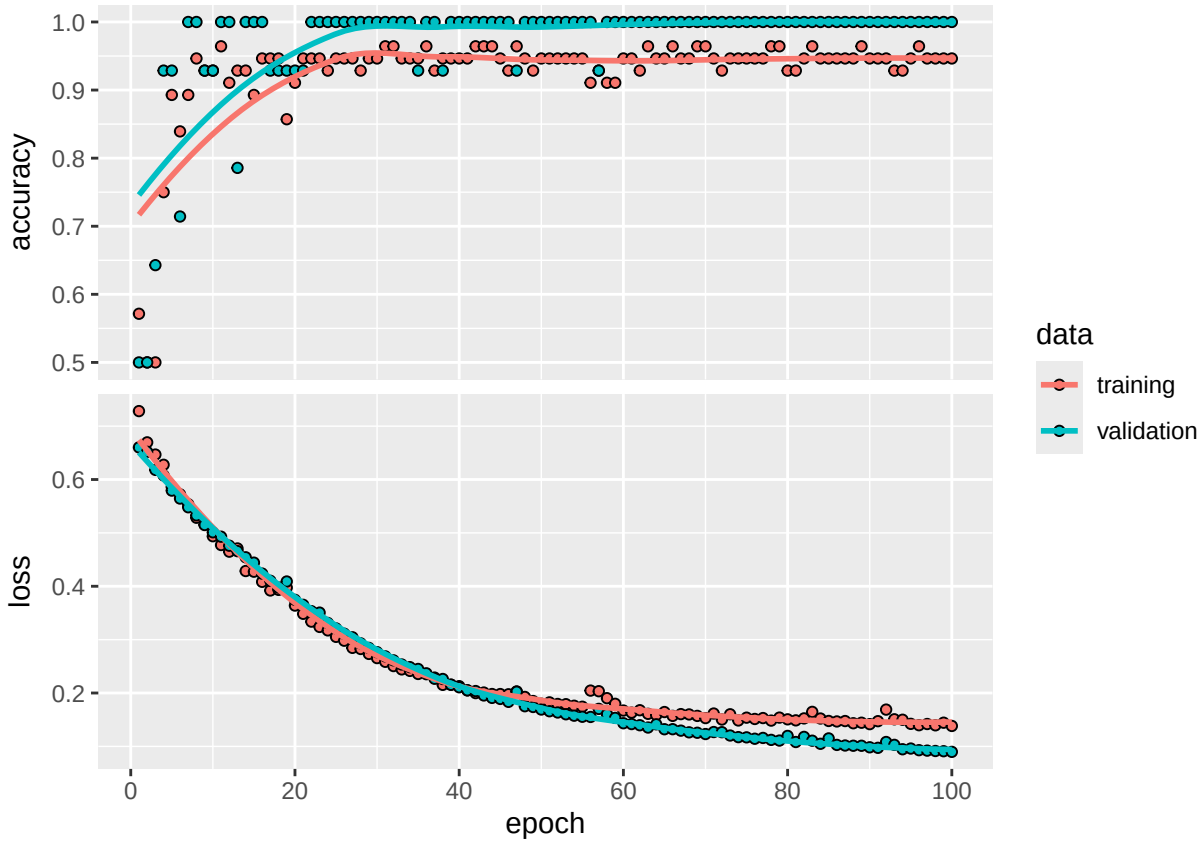
```
## Epoch 2/100
## 6/6 - 0s - 12ms/step - accuracy: 0.5000 - loss: 0.6696 - val_accuracy: 0.5000 - val_loss: 0.6519
## Epoch 3/100
## 6/6 - 0s - 11ms/step - accuracy: 0.5000 - loss: 0.6467 - val_accuracy: 0.6429 - val_loss: 0.6181
## Epoch 4/100
## 6/6 - 0s - 11ms/step - accuracy: 0.7500 - loss: 0.6276 - val_accuracy: 0.9286 - val_loss: 0.6077
## Epoch 5/100
## 6/6 - 0s - 11ms/step - accuracy: 0.8929 - loss: 0.5861 - val_accuracy: 0.9286 - val_loss: 0.5792
## Epoch 6/100
## 6/6 - 0s - 12ms/step - accuracy: 0.8393 - loss: 0.5722 - val_accuracy: 0.7143 - val_loss: 0.5644
## Epoch 7/100
## 6/6 - 0s - 12ms/step - accuracy: 0.8929 - loss: 0.5538 - val_accuracy: 1.0000 - val_loss: 0.5480
## Epoch 8/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.5284 - val_accuracy: 1.0000 - val_loss: 0.5328
## Epoch 9/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9286 - loss: 0.5157 - val_accuracy: 0.9286 - val_loss: 0.5148
## Epoch 10/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9286 - loss: 0.4937 - val_accuracy: 0.9286 - val_loss: 0.5011
## Epoch 11/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9643 - loss: 0.4773 - val_accuracy: 1.0000 - val_loss: 0.4928
## Epoch 12/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9107 - loss: 0.4648 - val_accuracy: 1.0000 - val_loss: 0.4760
## Epoch 13/100
## 6/6 - 0s - 14ms/step - accuracy: 0.9286 - loss: 0.4708 - val_accuracy: 0.7857 - val_loss: 0.4658
## Epoch 14/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9286 - loss: 0.4286 - val_accuracy: 1.0000 - val_loss: 0.4543
## Epoch 15/100
## 6/6 - 0s - 12ms/step - accuracy: 0.8929 - loss: 0.4273 - val_accuracy: 1.0000 - val_loss: 0.4443
## Epoch 16/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.4083 - val_accuracy: 1.0000 - val_loss: 0.4238
## Epoch 17/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.3921 - val_accuracy: 0.9286 - val_loss: 0.4103
## Epoch 18/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.3933 - val_accuracy: 0.9286 - val_loss: 0.3975
## Epoch 19/100
## 6/6 - 0s - 12ms/step - accuracy: 0.8571 - loss: 0.3975 - val_accuracy: 0.9286 - val_loss: 0.4089
## Epoch 20/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9107 - loss: 0.3639 - val_accuracy: 0.9286 - val_loss: 0.3746
## Epoch 21/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.3484 - val_accuracy: 0.9286 - val_loss: 0.3652
## Epoch 22/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.3338 - val_accuracy: 1.0000 - val_loss: 0.3533
## Epoch 23/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.3239 - val_accuracy: 1.0000 - val_loss: 0.3506
## Epoch 24/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9286 - loss: 0.3174 - val_accuracy: 1.0000 - val_loss: 0.3312
## Epoch 25/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.3052 - val_accuracy: 1.0000 - val_loss: 0.3210
## Epoch 26/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.2980 - val_accuracy: 1.0000 - val_loss: 0.3111
## Epoch 27/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.2846 - val_accuracy: 1.0000 - val_loss: 0.3044
## Epoch 28/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9286 - loss: 0.2826 - val_accuracy: 1.0000 - val_loss: 0.2931
```

```
## Epoch 29/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.2733 - val_accuracy: 1.0000 - val_loss: 0.2845
## Epoch 30/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.2654 - val_accuracy: 1.0000 - val_loss: 0.2765
## Epoch 31/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9643 - loss: 0.2585 - val_accuracy: 1.0000 - val_loss: 0.2686
## Epoch 32/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9643 - loss: 0.2505 - val_accuracy: 1.0000 - val_loss: 0.2612
## Epoch 33/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.2443 - val_accuracy: 1.0000 - val_loss: 0.2534
## Epoch 34/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.2415 - val_accuracy: 1.0000 - val_loss: 0.2485
## Epoch 35/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.2357 - val_accuracy: 0.9286 - val_loss: 0.2451
## Epoch 36/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9643 - loss: 0.2353 - val_accuracy: 1.0000 - val_loss: 0.2369
## Epoch 37/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9286 - loss: 0.2289 - val_accuracy: 1.0000 - val_loss: 0.2267
## Epoch 38/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.2153 - val_accuracy: 0.9286 - val_loss: 0.2263
## Epoch 39/100
## 6/6 - 0s - 10ms/step - accuracy: 0.9464 - loss: 0.2157 - val_accuracy: 1.0000 - val_loss: 0.2155
## Epoch 40/100
## 6/6 - 0s - 13ms/step - accuracy: 0.9464 - loss: 0.2128 - val_accuracy: 1.0000 - val_loss: 0.2108
## Epoch 41/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.2050 - val_accuracy: 1.0000 - val_loss: 0.2051
## Epoch 42/100
## 6/6 - 0s - 11ms/step - accuracy: 0.9643 - loss: 0.1998 - val_accuracy: 1.0000 - val_loss: 0.2037
## Epoch 43/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9643 - loss: 0.2009 - val_accuracy: 1.0000 - val_loss: 0.1954
## Epoch 44/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9643 - loss: 0.1981 - val_accuracy: 1.0000 - val_loss: 0.1909
## Epoch 45/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.1981 - val_accuracy: 1.0000 - val_loss: 0.1890
## Epoch 46/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9286 - loss: 0.1976 - val_accuracy: 1.0000 - val_loss: 0.1835
## Epoch 47/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9643 - loss: 0.2029 - val_accuracy: 0.9286 - val_loss: 0.2024
## Epoch 48/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.1927 - val_accuracy: 1.0000 - val_loss: 0.1753
## Epoch 49/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9286 - loss: 0.1850 - val_accuracy: 1.0000 - val_loss: 0.1739
## Epoch 50/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.1799 - val_accuracy: 1.0000 - val_loss: 0.1691
## Epoch 51/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.1825 - val_accuracy: 1.0000 - val_loss: 0.1659
## Epoch 52/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.1795 - val_accuracy: 1.0000 - val_loss: 0.1635
## Epoch 53/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.1791 - val_accuracy: 1.0000 - val_loss: 0.1602
## Epoch 54/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.1762 - val_accuracy: 1.0000 - val_loss: 0.1575
## Epoch 55/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.1743 - val_accuracy: 1.0000 - val_loss: 0.1555
```

```
## Epoch 56/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9107 - loss: 0.2045 - val_accuracy: 1.0000 - val_loss: 0.1552
## Epoch 57/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9286 - loss: 0.2034 - val_accuracy: 0.9286 - val_loss: 0.1703
## Epoch 58/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9107 - loss: 0.1904 - val_accuracy: 1.0000 - val_loss: 0.1603
## Epoch 59/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9107 - loss: 0.1797 - val_accuracy: 1.0000 - val_loss: 0.1534
## Epoch 60/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.1672 - val_accuracy: 1.0000 - val_loss: 0.1434
## Epoch 61/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.1632 - val_accuracy: 1.0000 - val_loss: 0.1418
## Epoch 62/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9286 - loss: 0.1678 - val_accuracy: 1.0000 - val_loss: 0.1397
## Epoch 63/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9643 - loss: 0.1610 - val_accuracy: 1.0000 - val_loss: 0.1354
## Epoch 64/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.1586 - val_accuracy: 1.0000 - val_loss: 0.1414
## Epoch 65/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.1644 - val_accuracy: 1.0000 - val_loss: 0.1321
## Epoch 66/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9643 - loss: 0.1572 - val_accuracy: 1.0000 - val_loss: 0.1323
## Epoch 67/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.1605 - val_accuracy: 1.0000 - val_loss: 0.1295
## Epoch 68/100
## 6/6 - 0s - 13ms/step - accuracy: 0.9464 - loss: 0.1599 - val_accuracy: 1.0000 - val_loss: 0.1263
## Epoch 69/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9643 - loss: 0.1572 - val_accuracy: 1.0000 - val_loss: 0.1255
## Epoch 70/100
## 6/6 - 0s - 13ms/step - accuracy: 0.9643 - loss: 0.1530 - val_accuracy: 1.0000 - val_loss: 0.1231
## Epoch 71/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.1616 - val_accuracy: 1.0000 - val_loss: 0.1269
## Epoch 72/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9286 - loss: 0.1504 - val_accuracy: 1.0000 - val_loss: 0.1266
## Epoch 73/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.1602 - val_accuracy: 1.0000 - val_loss: 0.1204
## Epoch 74/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.1486 - val_accuracy: 1.0000 - val_loss: 0.1173
## Epoch 75/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.1538 - val_accuracy: 1.0000 - val_loss: 0.1169
## Epoch 76/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.1514 - val_accuracy: 1.0000 - val_loss: 0.1144
## Epoch 77/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.1531 - val_accuracy: 1.0000 - val_loss: 0.1159
## Epoch 78/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9643 - loss: 0.1484 - val_accuracy: 1.0000 - val_loss: 0.1122
## Epoch 79/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9643 - loss: 0.1539 - val_accuracy: 1.0000 - val_loss: 0.1109
## Epoch 80/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9286 - loss: 0.1507 - val_accuracy: 1.0000 - val_loss: 0.1197
## Epoch 81/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9286 - loss: 0.1488 - val_accuracy: 1.0000 - val_loss: 0.1082
## Epoch 82/100
## 6/6 - 0s - 11ms/step - accuracy: 0.9464 - loss: 0.1524 - val_accuracy: 1.0000 - val_loss: 0.1178
```

```
## Epoch 83/100
## 6/6 - 0s - 13ms/step - accuracy: 0.9643 - loss: 0.1647 - val_accuracy: 1.0000 - val_loss: 0.1107
## Epoch 84/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.1522 - val_accuracy: 1.0000 - val_loss: 0.1049
## Epoch 85/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.1477 - val_accuracy: 1.0000 - val_loss: 0.1150
## Epoch 86/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.1467 - val_accuracy: 1.0000 - val_loss: 0.1028
## Epoch 87/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.1475 - val_accuracy: 1.0000 - val_loss: 0.1017
## Epoch 88/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.1434 - val_accuracy: 1.0000 - val_loss: 0.1012
## Epoch 89/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9643 - loss: 0.1444 - val_accuracy: 1.0000 - val_loss: 0.1012
## Epoch 90/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.1417 - val_accuracy: 1.0000 - val_loss: 0.0983
## Epoch 91/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.1473 - val_accuracy: 1.0000 - val_loss: 0.0973
## Epoch 92/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.1689 - val_accuracy: 1.0000 - val_loss: 0.1082
## Epoch 93/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9286 - loss: 0.1508 - val_accuracy: 1.0000 - val_loss: 0.1029
## Epoch 94/100
## 6/6 - 0s - 11ms/step - accuracy: 0.9286 - loss: 0.1500 - val_accuracy: 1.0000 - val_loss: 0.0946
## Epoch 95/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.1422 - val_accuracy: 1.0000 - val_loss: 0.0960
## Epoch 96/100
## 6/6 - 0s - 13ms/step - accuracy: 0.9643 - loss: 0.1397 - val_accuracy: 1.0000 - val_loss: 0.0929
## Epoch 97/100
## 6/6 - 0s - 12ms/step - accuracy: 0.9464 - loss: 0.1419 - val_accuracy: 1.0000 - val_loss: 0.0921
## Epoch 98/100
## 6/6 - 0s - 11ms/step - accuracy: 0.9464 - loss: 0.1395 - val_accuracy: 1.0000 - val_loss: 0.0914
## Epoch 99/100
## 6/6 - 0s - 11ms/step - accuracy: 0.9464 - loss: 0.1444 - val_accuracy: 1.0000 - val_loss: 0.0910
## Epoch 100/100
## 6/6 - 0s - 13ms/step - accuracy: 0.9464 - loss: 0.1383 - val_accuracy: 1.0000 - val_loss: 0.0898
```

```r
# Evaluate
plot(history)
```




```
score <- dnn_model %>% evaluate(as.matrix(test_data[, -5]), test_data$Species)
```

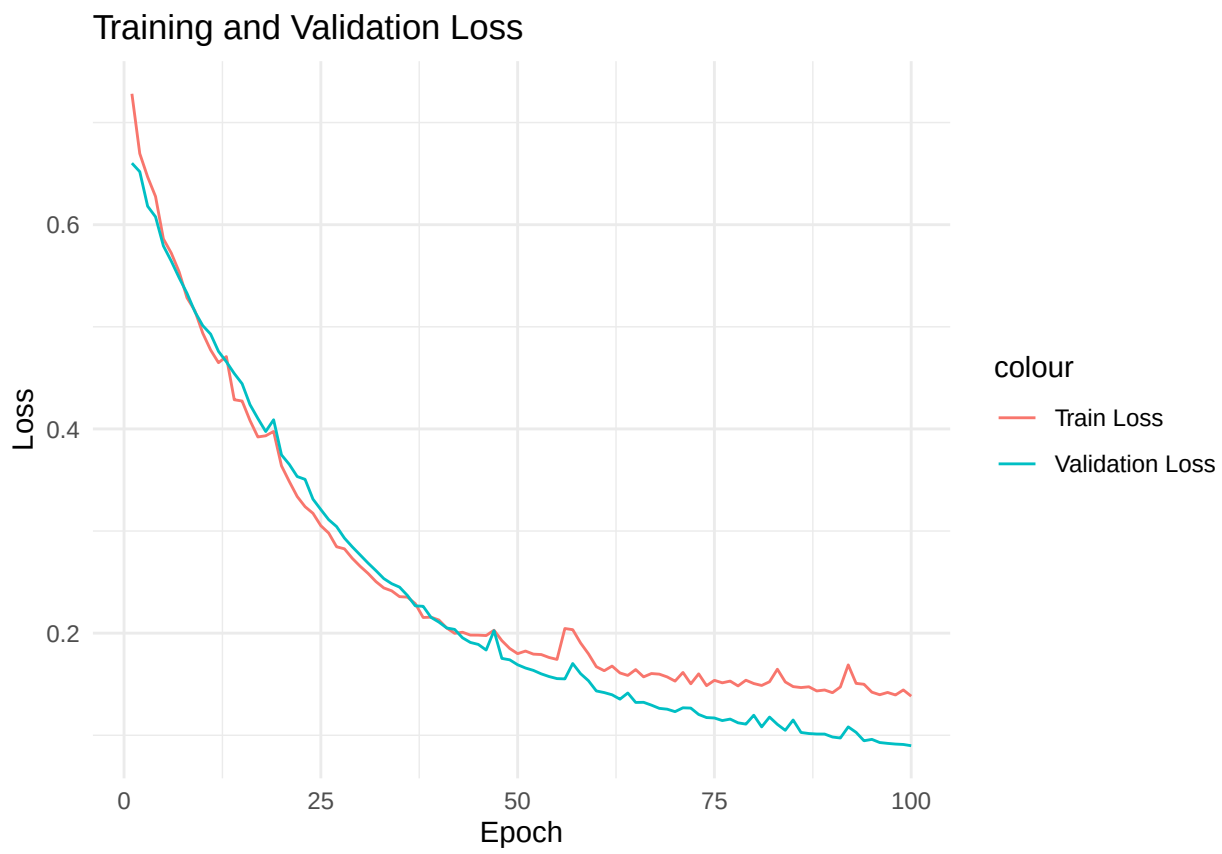
```
## 1/1 - 0s - 37ms/step - accuracy: 0.9667 - loss: 0.1061
```

```
cat("Test accuracy:", score$accuracy, "\n")
```

```
## Test accuracy: 0.9666666
```

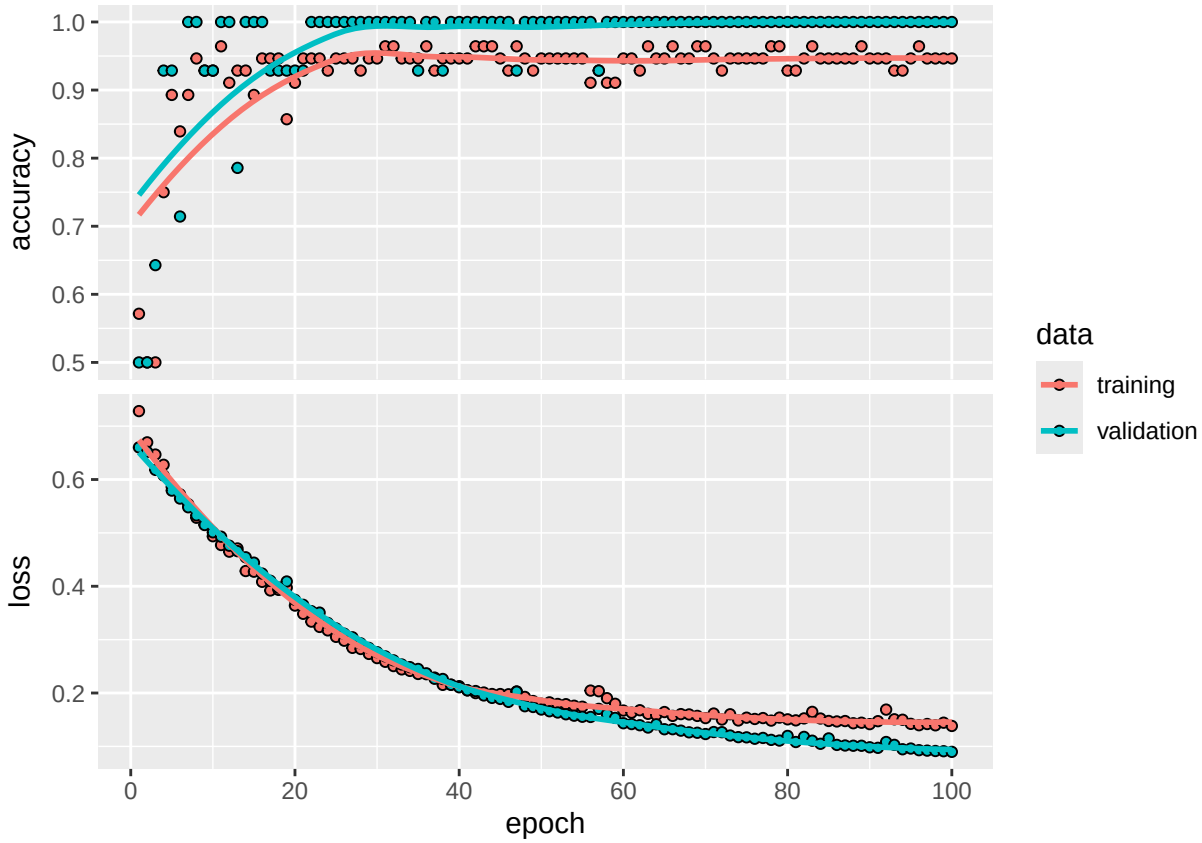
```
# Extract loss and accuracy from the history object
train_loss <- history$metrics$loss
val_loss <- history$metrics$val_loss
train_acc <- history$metrics$accuracy
val_acc <- history$metrics$val_accuracy

# Create a data frame for plotting
epochs <- 1:100
loss_data <- data.frame(
  Epoch = epochs,
  Train_Loss = train_loss,
  Val_Loss = val_loss,
  Train_Accuracy = train_acc,
  Val_Accuracy = val_acc
)
```

```r
# Plot training and validation loss
ggplot(loss_data, aes(x = Epoch)) +
  geom_line(aes(y = Train_Loss, color = "Train Loss")) +
  geom_line(aes(y = Val_Loss, color = "Validation Loss")) +
  labs(title = "Training and Validation Loss", y = "Loss") +
  theme_minimal()
```




```r
# Plot training and validation accuracy
ggplot(loss_data, aes(x = Epoch)) +
  geom_line(aes(y = Train_Accuracy, color = "Train Accuracy")) +
  geom_line(aes(y = Val_Accuracy, color = "Validation Accuracy")) +
  labs(title = "Training and Validation Accuracy", y = "Accuracy") +
  theme_minimal()
```


```
plot(history)
```



Exercise 2

(a)

Exercise 3

(a)

```
cifar10full <- dataset_cifar10() # See ?dataset_cifar10 for more info
sela <- (cifar10full$train$y < 6)
# select only first six categories
selb <- (cifar10full$test$y < 6)
cifar10 <- list(train=list(x=cifar10full$train$x[sela,,,],
                           y=cifar10full$train$y[sela]),
                test=list(x=cifar10full$test$x[selb,,,],
                          y=cifar10full$test$y[selb]))
# Scale RGB values in test and train inputs:
x_train <- cifar10$train$x / 255
x_test <- cifar10$test$x / 255
y_train <- to_categorical(cifar10$train$y, num_classes = 6)
y_test <- to_categorical(cifar10$test$y, num_classes = 6)
```

```r
# Transform (manually) to grayscale
x_train2 <- 0.3 * x_train[,,,1] + 0.59 * x_train[,,,2] + 0.11 * x_train[,,,3]
x_test2 <- 0.3 * x_test[,,,1] + 0.59 * x_test[,,,2] + 0.11 * x_test[,,,3]
x_train2 <- x_train2[,2:31,2:31] # omit a one pixel border
x_test2 <- x_test2[,2:31,2:31]
dim(x_train2) <- c(dim(x_train2), 1) # Fitting requires 4-dimensional array
dim(x_test2) <- c(dim(x_test2), 1)
rm(x_train, x_test, cifar10, cifar10full) # not needed anymore

# Define a function to create the model with a specified dropout rate
create_model <- function(dropout_rate) {
  model <- keras_model_sequential() %>%
    layer_conv_2d(filters = 32, kernel_size = c(3, 3), activation = "relu", input_shape = c(30, 30, 1)) %>%
    layer_max_pooling_2d(pool_size = c(2, 2)) %>%
    layer_dropout(rate = dropout_rate) %>%
    layer_conv_2d(filters = 64, kernel_size = c(3, 3), activation = "relu") %>%
    layer_max_pooling_2d(pool_size = c(2, 2)) %>%
    layer_dropout(rate = dropout_rate) %>%
    layer_flatten() %>%
    layer_dense(units = 128, activation = "relu") %>%
    layer_dropout(rate = dropout_rate) %>%
    layer_dense(units = 6) %>%
    layer_activation("softmax")

  model %>% compile(
    loss = "categorical_crossentropy",
    optimizer = optimizer_adam(),
    metrics = c("accuracy")
  )

  return(model)
}

# Train and evaluate the baseline model
baseline_model <- create_model(dropout_rate = 0.25)
baseline_history <- baseline_model %>% fit(
  x_train2, y_train,
  epochs = 2,
  batch_size = 64,
  validation_data = list(x_test2, y_test)
)
```

```
## Epoch 1/2
## 469/469 - 7s - 15ms/step - accuracy: 0.4143 - loss: 1.4308 - val_accuracy: 0.5397 - val_loss: 1.1885
## Epoch 2/2
## 469/469 - 6s - 13ms/step - accuracy: 0.5431 - loss: 1.1611 - val_accuracy: 0.5933 - val_loss: 1.0410
```

```r
baseline_score <- baseline_model %>% evaluate(x_test2, y_test)
```

```
## 188/188 - 0s - 2ms/step - accuracy: 0.5933 - loss: 1.0410
```

```
# Train and evaluate the model with increased dropout rate
increased_dropout_model <- create_model(dropout_rate = 0.5)
increased_dropout_history <- increased_dropout_model %>% fit(
  x_train2, y_train,
  epochs = 2,
  batch_size = 64,
  validation_data = list(x_test2, y_test)
)
```

```
## Epoch 1/2
## 469/469 - 7s - 15ms/step - accuracy: 0.3680 - loss: 1.5225 - val_accuracy: 0.5148 - val_loss: 1.2730
## Epoch 2/2
## 469/469 - 6s - 13ms/step - accuracy: 0.4967 - loss: 1.2668 - val_accuracy: 0.5595 - val_loss: 1.1379
```

```
increased_dropout_score <- increased_dropout_model %>% evaluate(x_test2, y_test)
```

```
## 188/188 - 0s - 3ms/step - accuracy: 0.5595 - loss: 1.1379
```

```
# Train and evaluate the model with decreased dropout rate
decreased_dropout_model <- create_model(dropout_rate = 0.1)
decreased_dropout_history <- decreased_dropout_model %>% fit(
  x_train2, y_train,
  epochs = 2,
  batch_size = 64,
  validation_data = list(x_test2, y_test)
)
```

```
## Epoch 1/2
## 469/469 - 7s - 15ms/step - accuracy: 0.4493 - loss: 1.3633 - val_accuracy: 0.5372 - val_loss: 1.1931
## Epoch 2/2
## 469/469 - 6s - 13ms/step - accuracy: 0.5641 - loss: 1.1126 - val_accuracy: 0.6097 - val_loss: 1.0152
```

```
decreased_dropout_score <- decreased_dropout_model %>% evaluate(x_test2, y_test)
```

```
## 188/188 - 0s - 2ms/step - accuracy: 0.6097 - loss: 1.0152
```

```
# Compare the results
results <- data.frame(
  Model = c("Baseline", "Increased Dropout", "Decreased Dropout"),
  Accuracy = c(baseline_score$accuracy, increased_dropout_score$accuracy, decreased_dropout_score$accura
)
print(results)
```

```
##               Model  Accuracy
## 1          Baseline 0.5933333
## 2 Increased Dropout 0.5595000
## 3 Decreased Dropout 0.6096666
```

With increased dropout rate, we do achieve a lower accuracy, while a decreased dropout rate does increase our accuracy.

(b)

```
cifar10full <- dataset_cifar10() # See ?dataset_cifar10 for more info
sela <- (cifar10full$train$y < 6)
# select only first six categories
selb <- (cifar10full$test$y < 6)
cifar10 <- list(train=list(x=cifar10full$train$x[sela,,,],
                           y=cifar10full$train$y[sela]),
                test=list(x=cifar10full$test$x[selb,,,],
                          y=cifar10full$test$y[selb]))
# Scale RGB values in test and train inputs:
x_train <- cifar10$train$x / 255
x_test <- cifar10$test$x / 255
y_train <- to_categorical(cifar10$train$y, num_classes = 6)
y_test <- to_categorical(cifar10$test$y, num_classes = 6)

# Transform (manually) to grayscale
x_train2 <- 0.3 * x_train[,,,1] + 0.59 * x_train[,,,2] + 0.11 * x_train[,,,3]
x_test2 <- 0.3 * x_test[,,,1] + 0.59 * x_test[,,,2] + 0.11 * x_test[,,,3]
x_train2 <- x_train2[,2:31,2:31] # omit a one pixel border
x_test2 <- x_test2[,2:31,2:31]
dim(x_train2) <- c(dim(x_train2), 1) # Fitting requires 4-dimensional array
dim(x_test2) <- c(dim(x_test2), 1)
rm(x_train, x_test, cifar10, cifar10full) # not needed anymore

# Define a function to create the model with a specified filter size
create_model <- function(filter_size) {
  model <- keras_model_sequential() %>%
    layer_conv_2d(filters = 32, kernel_size = filter_size, activation = "relu", input_shape = c(30, 30,
    layer_max_pooling_2d(pool_size = c(2, 2)) %>%
    layer_dropout(rate = 0.25) %>%
    layer_conv_2d(filters = 64, kernel_size = filter_size, activation = "relu") %>%
    layer_max_pooling_2d(pool_size = c(2, 2)) %>%
    layer_dropout(rate = 0.25) %>%
    layer_flatten() %>%
    layer_dense(units = 128, activation = "relu") %>%
    layer_dropout(rate = 0.25) %>%
    layer_dense(units = 6) %>%
    layer_activation("softmax")

  model %>% compile(
    loss = "categorical_crossentropy",
    optimizer = optimizer_adam(),
    metrics = c("accuracy")
  )

  return(model)
}

# Train and evaluate the baseline model
baseline_model <- create_model(filter_size = c(3, 3))
baseline_history <- baseline_model %>% fit(
  x_train2, y_train,
```

```
  epochs = 10,
  batch_size = 64,
  validation_split = 0.2
)
```

```
## Epoch 1/10
## 375/375 - 6s - 16ms/step - accuracy: 0.4127 - loss: 1.4342 - val_accuracy: 0.5443 - val_loss: 1.1891
## Epoch 2/10
## 375/375 - 5s - 13ms/step - accuracy: 0.5425 - loss: 1.1636 - val_accuracy: 0.5935 - val_loss: 1.0595
## Epoch 3/10
## 375/375 - 5s - 13ms/step - accuracy: 0.5870 - loss: 1.0622 - val_accuracy: 0.6113 - val_loss: 1.0030
## Epoch 4/10
## 375/375 - 5s - 13ms/step - accuracy: 0.6137 - loss: 0.9983 - val_accuracy: 0.6407 - val_loss: 0.9441
## Epoch 5/10
## 375/375 - 5s - 14ms/step - accuracy: 0.6344 - loss: 0.9514 - val_accuracy: 0.6462 - val_loss: 0.9282
## Epoch 6/10
## 375/375 - 5s - 13ms/step - accuracy: 0.6482 - loss: 0.9128 - val_accuracy: 0.6635 - val_loss: 0.8931
## Epoch 7/10
## 375/375 - 5s - 13ms/step - accuracy: 0.6641 - loss: 0.8749 - val_accuracy: 0.6673 - val_loss: 0.8749
## Epoch 8/10
## 375/375 - 5s - 13ms/step - accuracy: 0.6836 - loss: 0.8322 - val_accuracy: 0.6722 - val_loss: 0.8579
## Epoch 9/10
## 375/375 - 5s - 14ms/step - accuracy: 0.6917 - loss: 0.8054 - val_accuracy: 0.6855 - val_loss: 0.8409
## Epoch 10/10
## 375/375 - 5s - 13ms/step - accuracy: 0.7013 - loss: 0.7818 - val_accuracy: 0.6880 - val_loss: 0.8235
```

```
baseline_score <- baseline_model %>% evaluate(x_test2, y_test)
```

```
## 188/188 - 0s - 2ms/step - accuracy: 0.6823 - loss: 0.8373
```

```
# Train and evaluate the model with increased filter size
increased_filter_model <- create_model(filter_size = c(5, 5))
increased_filter_history <- increased_filter_model %>% fit(
  x_train2, y_train,
  epochs = 10,
  batch_size = 64,
  validation_split = 0.2
)
```

```
## Epoch 1/10
## 375/375 - 6s - 15ms/step - accuracy: 0.3820 - loss: 1.4760 - val_accuracy: 0.4920 - val_loss: 1.2715
## Epoch 2/10
## 375/375 - 5s - 13ms/step - accuracy: 0.4979 - loss: 1.2633 - val_accuracy: 0.5382 - val_loss: 1.1810
## Epoch 3/10
## 375/375 - 5s - 13ms/step - accuracy: 0.5401 - loss: 1.1696 - val_accuracy: 0.5802 - val_loss: 1.0923
## Epoch 4/10
## 375/375 - 5s - 13ms/step - accuracy: 0.5675 - loss: 1.1072 - val_accuracy: 0.5982 - val_loss: 1.0602
## Epoch 5/10
## 375/375 - 5s - 13ms/step - accuracy: 0.5834 - loss: 1.0664 - val_accuracy: 0.6063 - val_loss: 1.0307
## Epoch 6/10
## 375/375 - 5s - 13ms/step - accuracy: 0.6045 - loss: 1.0227 - val_accuracy: 0.6272 - val_loss: 0.9944
## Epoch 7/10
```

```
## 375/375 - 5s - 13ms/step - accuracy: 0.6182 - loss: 0.9880 - val_accuracy: 0.6162 - val_loss: 0.9852
## Epoch 8/10
## 375/375 - 5s - 13ms/step - accuracy: 0.6344 - loss: 0.9523 - val_accuracy: 0.6515 - val_loss: 0.9225
## Epoch 9/10
## 375/375 - 5s - 14ms/step - accuracy: 0.6442 - loss: 0.9297 - val_accuracy: 0.6440 - val_loss: 0.9160
## Epoch 10/10
## 375/375 - 5s - 13ms/step - accuracy: 0.6540 - loss: 0.9050 - val_accuracy: 0.6653 - val_loss: 0.8868
```

```
increased_filter_score <- increased_filter_model %>% evaluate(x_test2, y_test)
```

```
## 188/188 - 0s - 2ms/step - accuracy: 0.6555 - loss: 0.8988
```

```
# Train and evaluate the model with decreased filter size
decreased_filter_model <- create_model(filter_size = c(2, 2))
decreased_filter_history <- decreased_filter_model %>% fit(
  x_train2, y_train,
  epochs = 10,
  batch_size = 64,
  validation_split = 0.2
)
```

```
## Epoch 1/10
## 375/375 - 6s - 16ms/step - accuracy: 0.3887 - loss: 1.4873 - val_accuracy: 0.4828 - val_loss: 1.3045
## Epoch 2/10
## 375/375 - 5s - 13ms/step - accuracy: 0.5093 - loss: 1.2470 - val_accuracy: 0.5485 - val_loss: 1.1742
## Epoch 3/10
## 375/375 - 5s - 13ms/step - accuracy: 0.5532 - loss: 1.1462 - val_accuracy: 0.5952 - val_loss: 1.0671
## Epoch 4/10
## 375/375 - 5s - 13ms/step - accuracy: 0.5840 - loss: 1.0736 - val_accuracy: 0.6022 - val_loss: 1.0568
## Epoch 5/10
## 375/375 - 5s - 14ms/step - accuracy: 0.6053 - loss: 1.0231 - val_accuracy: 0.6195 - val_loss: 1.0226
## Epoch 6/10
## 375/375 - 5s - 13ms/step - accuracy: 0.6202 - loss: 0.9871 - val_accuracy: 0.6410 - val_loss: 0.9710
## Epoch 7/10
## 375/375 - 5s - 14ms/step - accuracy: 0.6347 - loss: 0.9547 - val_accuracy: 0.6500 - val_loss: 0.9541
## Epoch 8/10
## 375/375 - 5s - 14ms/step - accuracy: 0.6453 - loss: 0.9246 - val_accuracy: 0.6632 - val_loss: 0.9172
## Epoch 9/10
## 375/375 - 5s - 14ms/step - accuracy: 0.6563 - loss: 0.9022 - val_accuracy: 0.6590 - val_loss: 0.9148
## Epoch 10/10
## 375/375 - 5s - 14ms/step - accuracy: 0.6681 - loss: 0.8716 - val_accuracy: 0.6637 - val_loss: 0.8936
```

```
decreased_filter_score <- decreased_filter_model %>% evaluate(x_test2, y_test)
```

```
## 188/188 - 0s - 3ms/step - accuracy: 0.6610 - loss: 0.9004
```

```
# Compare the results
results <- data.frame(
  Model = c("Baseline", "Increased Filter Size", "Decreased Filter Size"),
  Accuracy = c(baseline_score$accuracy, increased_filter_score$accuracy, decreased_filter_score$accuracy
)
print(results)
```

```
##                    Model  Accuracy
## 1               Baseline 0.6823334
## 2 Increased Filter Size 0.6555000
## 3 Decreased Filter Size 0.6610000
```

From the results, we cannot observe a big difference in accuracy, though the Baseline is the most accurate.

(c)

```
cifar10full <- dataset_cifar10() # See ?dataset_cifar10 for more info
sela <- (cifar10full$train$y < 6)
# select only first six categories
selb <- (cifar10full$test$y < 6)
cifar10 <- list(train=list(x=cifar10full$train$x[sela,,,],
                           y=cifar10full$train$y[sela]),
                test=list(x=cifar10full$test$x[selb,,,],
                          y=cifar10full$test$y[selb]))
# Scale RGB values in test and train inputs:
x_train <- cifar10$train$x / 255
x_test <- cifar10$test$x / 255
y_train <- to_categorical(cifar10$train$y, num_classes = 6)
y_test <- to_categorical(cifar10$test$y, num_classes = 6)

# Transform (manually) to grayscale
x_train2 <- 0.3 * x_train[,,,1] + 0.59 * x_train[,,,2] + 0.11 * x_train[,,,3]
x_test2 <- 0.3 * x_test[,,,1] + 0.59 * x_test[,,,2] + 0.11 * x_test[,,,3]
x_train2 <- x_train2[,2:31,2:31] # omit a one pixel border
x_test2 <- x_test2[,2:31,2:31]
dim(x_train2) <- c(dim(x_train2), 1) # Fitting requires 4-dimensional array
dim(x_test2) <- c(dim(x_test2), 1)
rm(x_train, x_test, cifar10, cifar10full) # not needed anymore

# Define a function to create the model with a specified filter size for the second layer
create_model <- function(second_layer_filter_size) {
  model <- keras_model_sequential() %>%
    layer_conv_2d(filters = 32, kernel_size = c(3, 3), activation = "relu", input_shape = c(30, 30, 1)) %>%
    layer_max_pooling_2d(pool_size = c(2, 2)) %>%
    layer_dropout(rate = 0.25) %>%
    layer_conv_2d(filters = 64, kernel_size = second_layer_filter_size, activation = "relu") %>%
    layer_max_pooling_2d(pool_size = c(2, 2)) %>%
    layer_dropout(rate = 0.25) %>%
    layer_flatten() %>%
    layer_dense(units = 128, activation = "relu") %>%
    layer_dropout(rate = 0.25) %>%
    layer_dense(units = 6) %>%
    layer_activation("softmax")

  model %>% compile(
    loss = "categorical_crossentropy",
    optimizer = optimizer_adam(),
    metrics = c("accuracy")
  )
```

```
  return(model)
}

# Train and evaluate the baseline model
baseline_model <- create_model(second_layer_filter_size = c(3, 3))
baseline_history <- baseline_model %>% fit(
  x_train2, y_train,
  epochs = 10,
  batch_size = 64,
  validation_split = 0.2
)
```

```
## Epoch 1/10
## 375/375 - 7s - 20ms/step - accuracy: 0.4164 - loss: 1.4255 - val_accuracy: 0.5065 - val_loss: 1.2473
## Epoch 2/10
## 375/375 - 6s - 15ms/step - accuracy: 0.5396 - loss: 1.1747 - val_accuracy: 0.5865 - val_loss: 1.0872
## Epoch 3/10
## 375/375 - 6s - 15ms/step - accuracy: 0.5814 - loss: 1.0704 - val_accuracy: 0.6135 - val_loss: 1.0032
## Epoch 4/10
## 375/375 - 6s - 15ms/step - accuracy: 0.6106 - loss: 1.0115 - val_accuracy: 0.6360 - val_loss: 0.9446
## Epoch 5/10
## 375/375 - 6s - 15ms/step - accuracy: 0.6318 - loss: 0.9591 - val_accuracy: 0.6275 - val_loss: 0.9576
## Epoch 6/10
## 375/375 - 6s - 15ms/step - accuracy: 0.6469 - loss: 0.9163 - val_accuracy: 0.6598 - val_loss: 0.8925
## Epoch 7/10
## 375/375 - 6s - 15ms/step - accuracy: 0.6628 - loss: 0.8757 - val_accuracy: 0.6613 - val_loss: 0.8974
## Epoch 8/10
## 375/375 - 6s - 15ms/step - accuracy: 0.6777 - loss: 0.8485 - val_accuracy: 0.6767 - val_loss: 0.8562
## Epoch 9/10
## 375/375 - 6s - 15ms/step - accuracy: 0.6918 - loss: 0.8106 - val_accuracy: 0.6842 - val_loss: 0.8382
## Epoch 10/10
## 375/375 - 6s - 15ms/step - accuracy: 0.6967 - loss: 0.7887 - val_accuracy: 0.6815 - val_loss: 0.8327
```

```
baseline_score <- baseline_model %>% evaluate(x_test2, y_test)
```

```
## 188/188 - 1s - 3ms/step - accuracy: 0.6815 - loss: 0.8355
```

```
# Train and evaluate the model with increased filter size in the second layer
increased_filter_model <- create_model(second_layer_filter_size = c(5, 5))
increased_filter_history <- increased_filter_model %>% fit(
  x_train2, y_train,
  epochs = 10,
  batch_size = 64,
  validation_split = 0.2
)
```

```
## Epoch 1/10
## 375/375 - 8s - 20ms/step - accuracy: 0.3740 - loss: 1.4942 - val_accuracy: 0.5160 - val_loss: 1.2451
## Epoch 2/10
## 375/375 - 6s - 16ms/step - accuracy: 0.5045 - loss: 1.2370 - val_accuracy: 0.5542 - val_loss: 1.1585
## Epoch 3/10
```

```
## 375/375 - 6s - 16ms/step - accuracy: 0.5477 - loss: 1.1547 - val_accuracy: 0.5763 - val_loss: 1.0761
## Epoch 4/10
## 375/375 - 6s - 16ms/step - accuracy: 0.5745 - loss: 1.0933 - val_accuracy: 0.6070 - val_loss: 1.0496
## Epoch 5/10
## 375/375 - 6s - 16ms/step - accuracy: 0.5957 - loss: 1.0390 - val_accuracy: 0.6318 - val_loss: 0.9718
## Epoch 6/10
## 375/375 - 6s - 16ms/step - accuracy: 0.6149 - loss: 0.9933 - val_accuracy: 0.6257 - val_loss: 0.9634
## Epoch 7/10
## 375/375 - 6s - 15ms/step - accuracy: 0.6315 - loss: 0.9553 - val_accuracy: 0.6538 - val_loss: 0.9177
## Epoch 8/10
## 375/375 - 6s - 15ms/step - accuracy: 0.6504 - loss: 0.9170 - val_accuracy: 0.6522 - val_loss: 0.8928
## Epoch 9/10
## 375/375 - 6s - 15ms/step - accuracy: 0.6645 - loss: 0.8830 - val_accuracy: 0.6523 - val_loss: 0.9057
## Epoch 10/10
## 375/375 - 5s - 14ms/step - accuracy: 0.6719 - loss: 0.8569 - val_accuracy: 0.6797 - val_loss: 0.8542
```

```
increased_filter_score <- increased_filter_model %>% evaluate(x_test2, y_test)
```

```
## 188/188 - 0s - 3ms/step - accuracy: 0.6662 - loss: 0.8767
```

```
# Compare the results
results <- data.frame(
  Model = c("Baseline", "Increased Filter Size in Second Layer"),
  Accuracy = c(baseline_score$accuracy, increased_filter_score$accuracy)
)
print(results)
```

```
##                                   Model  Accuracy
## 1                              Baseline 0.6815000
## 2 Increased Filter Size in Second Layer 0.6661667
```

Indeed, it does not imporve performance, the performance stays the same.