hw8_shuangyu_zhao

Shuangyu Zhao

2023-04-23

```r
library(keras)
library(caret)
```

```
## Loading required package: ggplot2

## Loading required package: lattice
```

```r
library(sqldf)
```

```
## Loading required package: gsubfn

## Loading required package: proto

## Loading required package: RSQLite
```

```r
library(ggplot2)
library(neuralnet)
library(ISLR2)
library(xgboost)
```

1. For the star dataset,

```r
star <- read.csv("/Users/apple/Desktop/STT811_appl_stat_model/data/star_classification.csv")
star <- na.omit(star)
head(star)
```

```
##         obj_ID    alpha      delta        u        g        r        i        z
## 1 1.237661e+18 135.6891 32.4946318 23.87882 22.27530 20.39501 19.16573 18.79371
## 2 1.237665e+18 144.8261 31.2741849 24.77759 22.83188 22.58444 21.16812 21.61427
## 3 1.237661e+18 142.1888 35.5824442 25.26307 22.66389 20.60976 19.34857 18.94827
## 4 1.237663e+18 338.7410 -0.4028276 22.13682 23.77656 21.61162 20.50454 19.25010
## 5 1.237680e+18 345.2826 21.1838656 19.43718 17.58028 16.49747 15.97711 15.54461
## 6 1.237680e+18 340.9951 20.5894763 23.48827 23.33776 21.32195 20.25615 19.54544
##   run_ID rerun_ID cam_col field_ID  spec_obj_ID  class  redshift plate   MJD
## 1   3606      301       2       79 6.543777e+18 GALAXY 0.6347936  5812 56354
## 2   4518      301       5      119 1.176014e+19 GALAXY 0.7791360 10445 58158
## 3   3606      301       2      120 5.152200e+18 GALAXY 0.6441945  4576 55592
## 4   4192      301       3      214 1.030107e+19 GALAXY 0.9323456  9149 58039
```

```
## 5   8102      301       3      137 6.891865e+18 GALAXY 0.1161227  6121 56187
## 6   8102      301       3      110 5.658977e+18    QSO 1.4246590  5026 55855
##   fiber_ID
## 1      171
## 2      427
## 3      299
## 4      775
## 5      842
## 6      741
```

```
sqldf("SELECT DISTINCT class
      FROM star")
```

```
##    class
## 1 GALAXY
## 2    QSO
## 3   STAR
```

a. Create a 70/30 train test split.

```
star$y <- ifelse(star$class == "GALAXY", 1, ifelse(star$class == "QSO", 2, 3))
split_pct <- 0.7
n <- split_pct * length(star$obj_ID)
set.seed(123)
row_samp <- sample(1:length(star$obj_ID), n, replace = FALSE)
train_star <- star[row_samp, ]
test_star <- star[-row_samp, ]
```

b. Create a neural network model for type(class), using u, g, z, and Redshift. Use a single hidden layer with 3 nodes and tanh activation functions. Compute the confusion matrix for the train dataset.(CNN)

```
star_nn_1 <- neuralnet(y~ u+g+z+redshift, data = train_star,act.fct = 'tanh', hidden = c(3), linear.outp

y_pred_1b <- neuralnet::compute(star_nn_1, test_star)
pred_labels_1b <- apply(y_pred_1b$net.result, 1, which.max)


accuracy_1b <- sum(pred_labels_1b == test_star$y) / length(test_star$y)
print(paste0("Accuracy: ", round(accuracy_1b, 3)))
```

```
## [1] "Accuracy: 0.598"
```

c. Re-create the model predictions in (b) with algebraic operations.

```
X_1c <- cbind(test_star$u,test_star$g,test_star$z,test_star$redshift)

logi <- function(x) 1/(1 + exp(-1*x))

n1 <- logi(star_nn_1$result.matrix[4]
           + star_nn_1$result.matrix[5]*X_1c[,1]
           + star_nn_1$result.matrix[6]*X_1c[,2]
```

```r
           + star_nn_1$result.matrix[7]*X_1c[,3]
           + star_nn_1$result.matrix[8]*X_1c[,4])
n2 <- logi(star_nn_1$result.matrix[9]
           + star_nn_1$result.matrix[10]*X_1c[,1]
           + star_nn_1$result.matrix[11]*X_1c[,2]
           + star_nn_1$result.matrix[12]*X_1c[,3]
           + star_nn_1$result.matrix[13]*X_1c[,4])
n3 <- logi(star_nn_1$result.matrix[14]
           + star_nn_1$result.matrix[15]*X_1c[,1]
           + star_nn_1$result.matrix[16]*X_1c[,2]
           + star_nn_1$result.matrix[17]*X_1c[,3]
           + star_nn_1$result.matrix[18]*X_1c[,4])

test_1d <- cbind(n1, n2, n3)

pr <- logi(star_nn_1$result.matrix[19] + star_nn_1$result.matrix[20]*n1 + star_nn_1$result.matrix[21]*n

head(cbind(pr,predict(star_nn_1, test_star)[,1]))
```

```
##          pr
## 2  0.997071 0.9999999
## 3  0.997071 0.9999999
## 9  0.997071 0.9999999
## 11 0.997071 0.9999999
## 13 0.997071 0.9999999
## 14 0.997071 0.9999999
```

d. Create an xgboost model using the Texas 2-step. Use the outputs of the hidden layer from (c) as inputs to an xgboost model to create predictions for Class. Compare the results to what we get in (b).

```r
X_1d_train <- cbind(train_star$u, train_star$g, train_star$z, train_star$redshift)
n1 <- logi(star_nn_1$result.matrix[4]
           + star_nn_1$result.matrix[5]*X_1d_train[,1]
           + star_nn_1$result.matrix[6]*X_1d_train[,2]
           + star_nn_1$result.matrix[7]*X_1d_train[,3]
           + star_nn_1$result.matrix[8]*X_1d_train[,4])
n2 <- logi(star_nn_1$result.matrix[9]
           + star_nn_1$result.matrix[10]*X_1d_train[,1]
           + star_nn_1$result.matrix[11]*X_1d_train[,2]
           + star_nn_1$result.matrix[12]*X_1d_train[,3]
           + star_nn_1$result.matrix[13]*X_1d_train[,4])
n3 <- logi(star_nn_1$result.matrix[14]
           + star_nn_1$result.matrix[15]*X_1d_train[,1]
           + star_nn_1$result.matrix[16]*X_1d_train[,2]
           + star_nn_1$result.matrix[17]*X_1d_train[,3]
           + star_nn_1$result.matrix[18]*X_1d_train[,4])
train_1d <- cbind(n1,n2,n3)

train_star$y <- as.integer(factor(train_star$y)) - 1
test_star$y <- as.integer(factor(test_star$y)) - 1

star_xgb <- xgboost(data = data.matrix(train_1d), nrounds = 100, max_depth = 2, eta = 0.3,
                    label = train_star$y, objective = "multi:softmax", num_class = 3)
```

```
## [1]  train-mlogloss:1.006863
## [2]  train-mlogloss:0.951331
## [3]  train-mlogloss:0.915954
## [4]  train-mlogloss:0.892602
## [5]  train-mlogloss:0.876806
## [6]  train-mlogloss:0.865833
## [7]  train-mlogloss:0.857991
## [8]  train-mlogloss:0.852376
## [9]  train-mlogloss:0.847935
## [10] train-mlogloss:0.844581
## [11] train-mlogloss:0.842132
## [12] train-mlogloss:0.840100
## [13] train-mlogloss:0.838609
## [14] train-mlogloss:0.837337
## [15] train-mlogloss:0.836344
## [16] train-mlogloss:0.835576
## [17] train-mlogloss:0.834961
## [18] train-mlogloss:0.834468
## [19] train-mlogloss:0.834033
## [20] train-mlogloss:0.833687
## [21] train-mlogloss:0.833422
## [22] train-mlogloss:0.833202
## [23] train-mlogloss:0.833004
## [24] train-mlogloss:0.832852
## [25] train-mlogloss:0.832700
## [26] train-mlogloss:0.832563
## [27] train-mlogloss:0.832457
## [28] train-mlogloss:0.832365
## [29] train-mlogloss:0.832271
## [30] train-mlogloss:0.832187
## [31] train-mlogloss:0.832125
## [32] train-mlogloss:0.832064
## [33] train-mlogloss:0.831989
## [34] train-mlogloss:0.831907
## [35] train-mlogloss:0.831855
## [36] train-mlogloss:0.831777
## [37] train-mlogloss:0.831704
## [38] train-mlogloss:0.831609
## [39] train-mlogloss:0.831548
## [40] train-mlogloss:0.831506
## [41] train-mlogloss:0.831430
## [42] train-mlogloss:0.831374
## [43] train-mlogloss:0.831353
## [44] train-mlogloss:0.831300
## [45] train-mlogloss:0.831269
## [46] train-mlogloss:0.831232
## [47] train-mlogloss:0.831167
## [48] train-mlogloss:0.831133
## [49] train-mlogloss:0.831069
## [50] train-mlogloss:0.831033
## [51] train-mlogloss:0.830976
## [52] train-mlogloss:0.830926
## [53] train-mlogloss:0.830902
## [54] train-mlogloss:0.830844
```

```
## [55] train-mlogloss:0.830794
## [56] train-mlogloss:0.830753
## [57] train-mlogloss:0.830705
## [58] train-mlogloss:0.830657
## [59] train-mlogloss:0.830617
## [60] train-mlogloss:0.830551
## [61] train-mlogloss:0.830503
## [62] train-mlogloss:0.830482
## [63] train-mlogloss:0.830435
## [64] train-mlogloss:0.830385
## [65] train-mlogloss:0.830354
## [66] train-mlogloss:0.830301
## [67] train-mlogloss:0.830261
## [68] train-mlogloss:0.830186
## [69] train-mlogloss:0.830157
## [70] train-mlogloss:0.830132
## [71] train-mlogloss:0.830099
## [72] train-mlogloss:0.830063
## [73] train-mlogloss:0.830035
## [74] train-mlogloss:0.830000
## [75] train-mlogloss:0.829955
## [76] train-mlogloss:0.829895
## [77] train-mlogloss:0.829882
## [78] train-mlogloss:0.829829
## [79] train-mlogloss:0.829788
## [80] train-mlogloss:0.829738
## [81] train-mlogloss:0.829692
## [82] train-mlogloss:0.829657
## [83] train-mlogloss:0.829616
## [84] train-mlogloss:0.829561
## [85] train-mlogloss:0.829531
## [86] train-mlogloss:0.829506
## [87] train-mlogloss:0.829477
## [88] train-mlogloss:0.829458
## [89] train-mlogloss:0.829434
## [90] train-mlogloss:0.829377
## [91] train-mlogloss:0.829336
## [92] train-mlogloss:0.829273
## [93] train-mlogloss:0.829243
## [94] train-mlogloss:0.829228
## [95] train-mlogloss:0.829212
## [96] train-mlogloss:0.829199
## [97] train-mlogloss:0.829148
## [98] train-mlogloss:0.829066
## [99] train-mlogloss:0.828973
## [100]    train-mlogloss:0.828916
```

```r
pred_1d <- predict(star_xgb, as.matrix(test_1d[]))
pred_labels_1d <- max.col(pred_1d) - 1

# Evaluate the accuracy of the predictions
accuracy_1d <- sum(pred_labels_1d == test_star$y) / length(test_star$y)
print(paste0("Accuracy: ", round(accuracy_1d, 3)))
```

```
## [1] "Accuracy: 0.598"
```

2. Fit a neural network to the Default data. Use a single hidden layer with 10 units, and dropout regularization. Have a look at Labs 10.9.1-10.9.2 for guidance. Compare the results from using different dropout regularization rates.

```
default <- na.omit(Default)
head(default)
```

```
##   default student   balance    income
## 1      No      No  729.5265 44361.625
## 2      No     Yes  817.1804 12106.135
## 3      No      No 1073.5492 31767.139
## 4      No      No  529.2506 35704.494
## 5      No      No  785.6559 38463.496
## 6      No     Yes  919.5885  7491.559
```

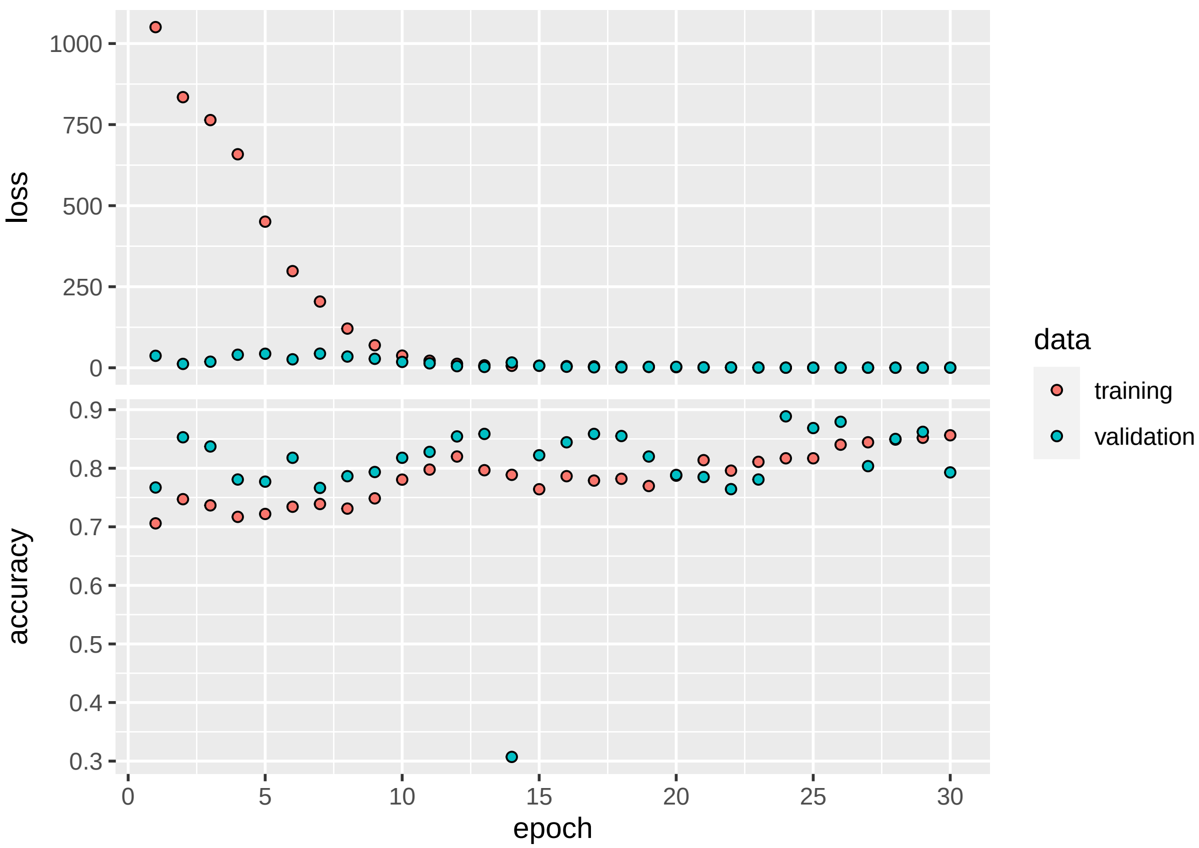
```
split_pct <- 0.7
n <- split_pct * length(default$default)
set.seed(123)
row_samp <- sample(1:length(default$default), n, replace = FALSE)
train_default <- default[row_samp, ]
test_default <- default[-row_samp, ]
```

```
default_train_feature <- as.matrix(train_default[, c(3, 4)])
default_test_feature <- as.matrix(test_default[, c(3, 4)])

default_train_target <- ifelse(train_default$student == "Yes", 1, 0)
default_test_target <- ifelse(test_default$student == "Yes", 1, 0)
default_train_target_onehot <- to_categorical(default_train_target, 2)
default_test_target_onehot <- to_categorical(default_test_target, 2)
```

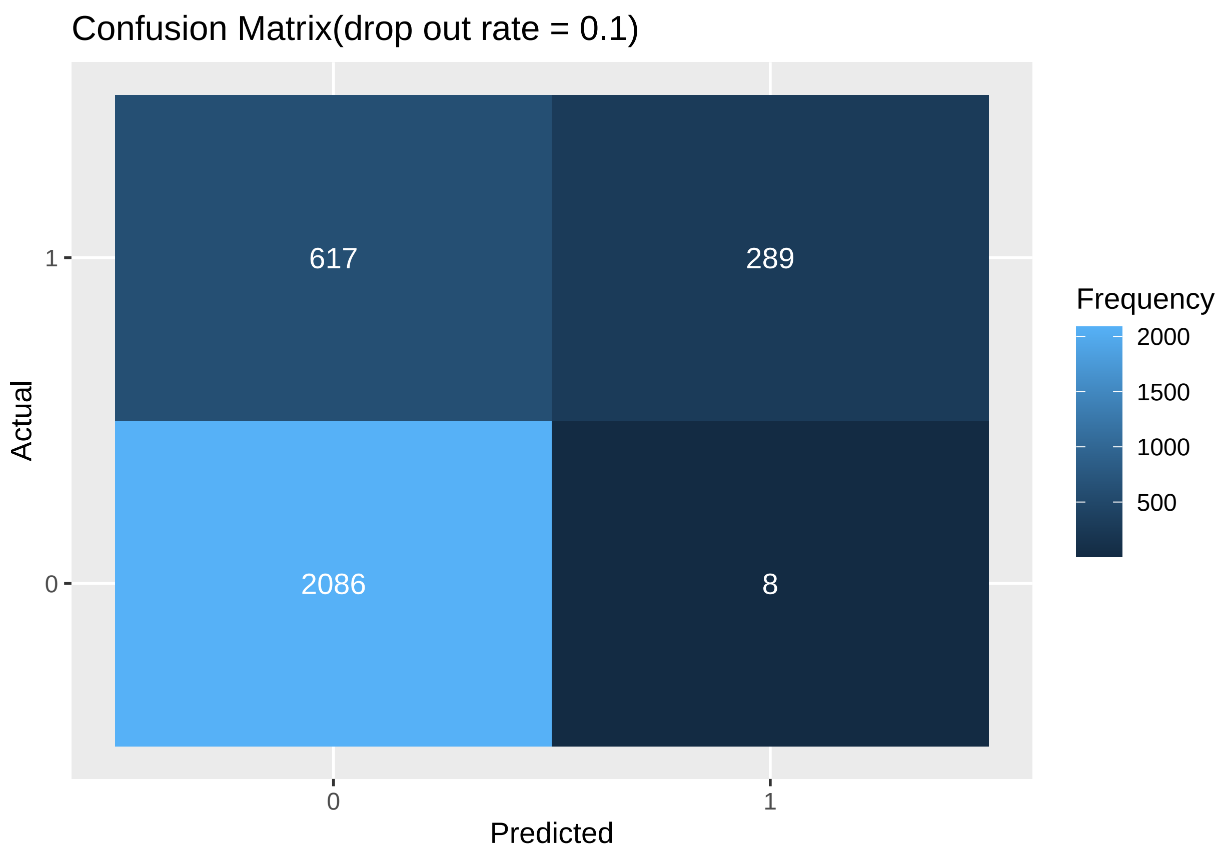
```
# drop out rate = 0.1
model_nn_2_01 <- keras_model_sequential() %>%
  layer_dense(units = 10, activation = "relu", input_shape =c(2)) %>%
  layer_dropout(rate = 0.1) %>%
  layer_dense(units = 2, activation = "softmax")

model_nn_2_01 %>% compile(loss = "categorical_crossentropy", optimizer = optimizer_rmsprop(), metrics =

system.time(
  history <- model_nn_2_01 %>%
    fit(default_train_feature, default_train_target_onehot, epochs = 30, batch_size = 128,
        validation_split = 0.2)
)
```

```
##    user  system elapsed
##   4.239   0.447   4.233
```

```
plot(history, smooth = FALSE)
```



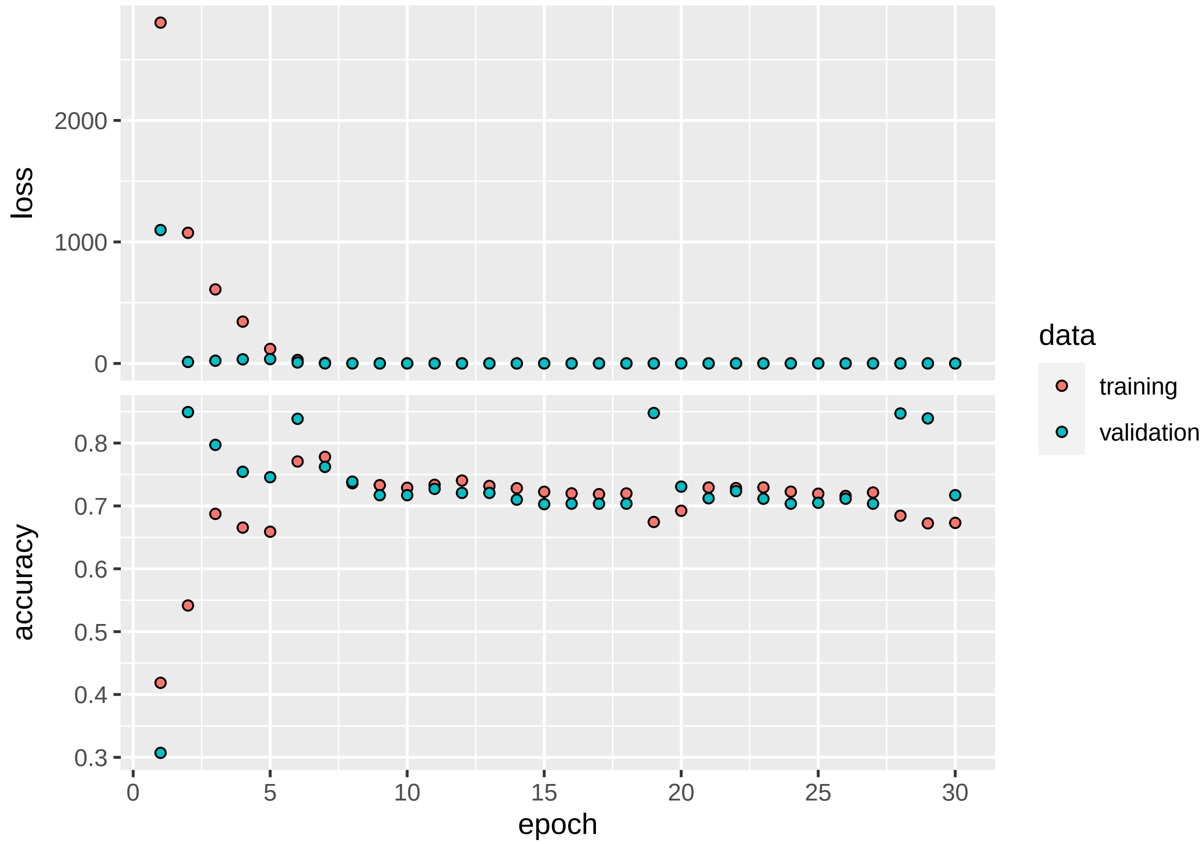

```
test_pred_2_01 <-  predict(model_nn_2_01, default_test_feature)
test_pred_2_01_label <- apply(test_pred_2_01, 1, which.max) - 1
train_cm <- confusionMatrix(data =as.factor(test_pred_2_01_label), reference = as.factor(default_test_ta
ggplot(data = data.frame(train_cm$table), aes(x = Prediction, y = Reference, fill = Freq)) +
  geom_tile() +
  geom_text(aes(label = Freq), color = "white") +
  labs(title = "Confusion Matrix(drop out rate = 0.1)",
       x = "Predicted",
       y = "Actual",
       fill = "Frequency")
```



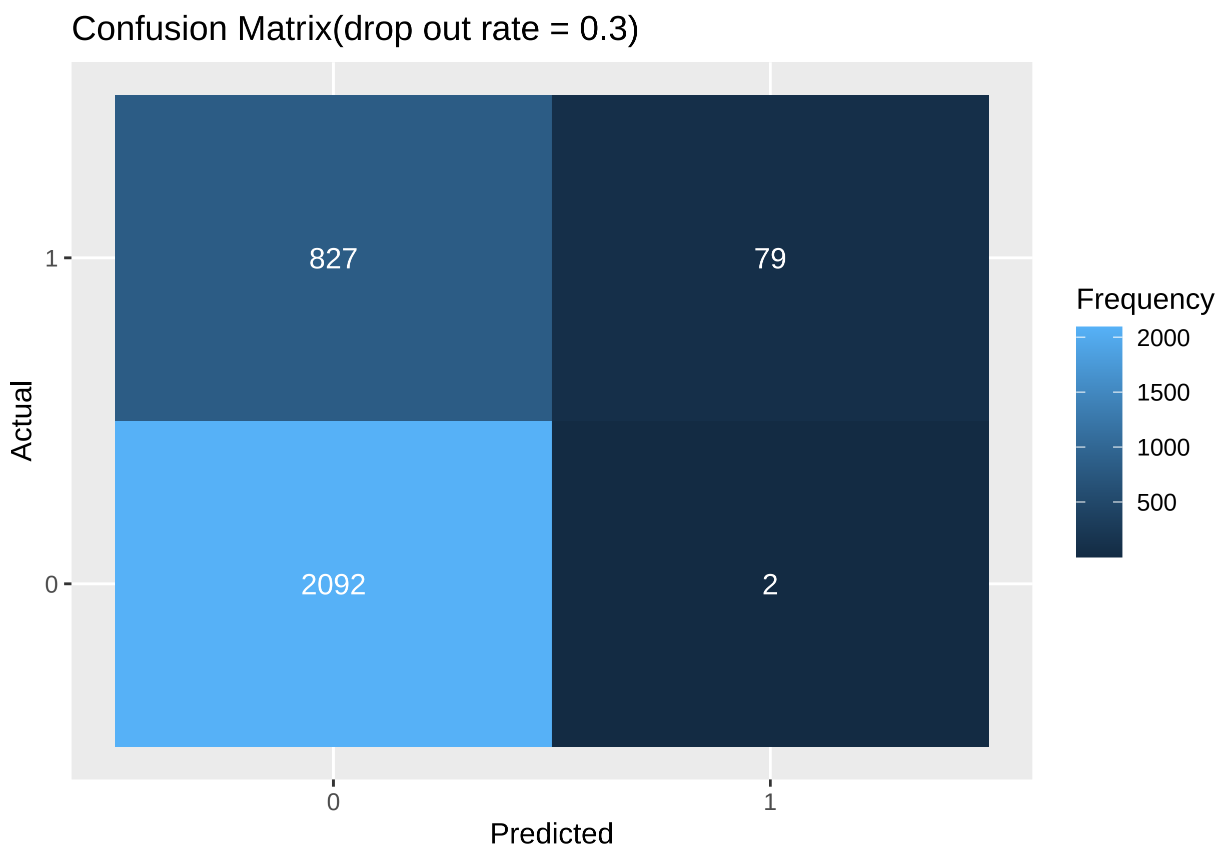

```r
# drop out rate = 0.3
model_nn_2_03 <- keras_model_sequential() %>%
  layer_dense(units = 10, activation = "relu", input_shape =c(2)) %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 2, activation = "softmax")

model_nn_2_03 %>% compile(loss = "categorical_crossentropy", optimizer = optimizer_rmsprop(), metrics =

system.time(
  history <- model_nn_2_03 %>%
    fit(default_train_feature, default_train_target_onehot, epochs = 30, batch_size = 128,
        validation_split = 0.2)
)
```

```
##    user  system elapsed
##   3.970   0.378   3.773
```

```r
plot(history, smooth = FALSE)
```



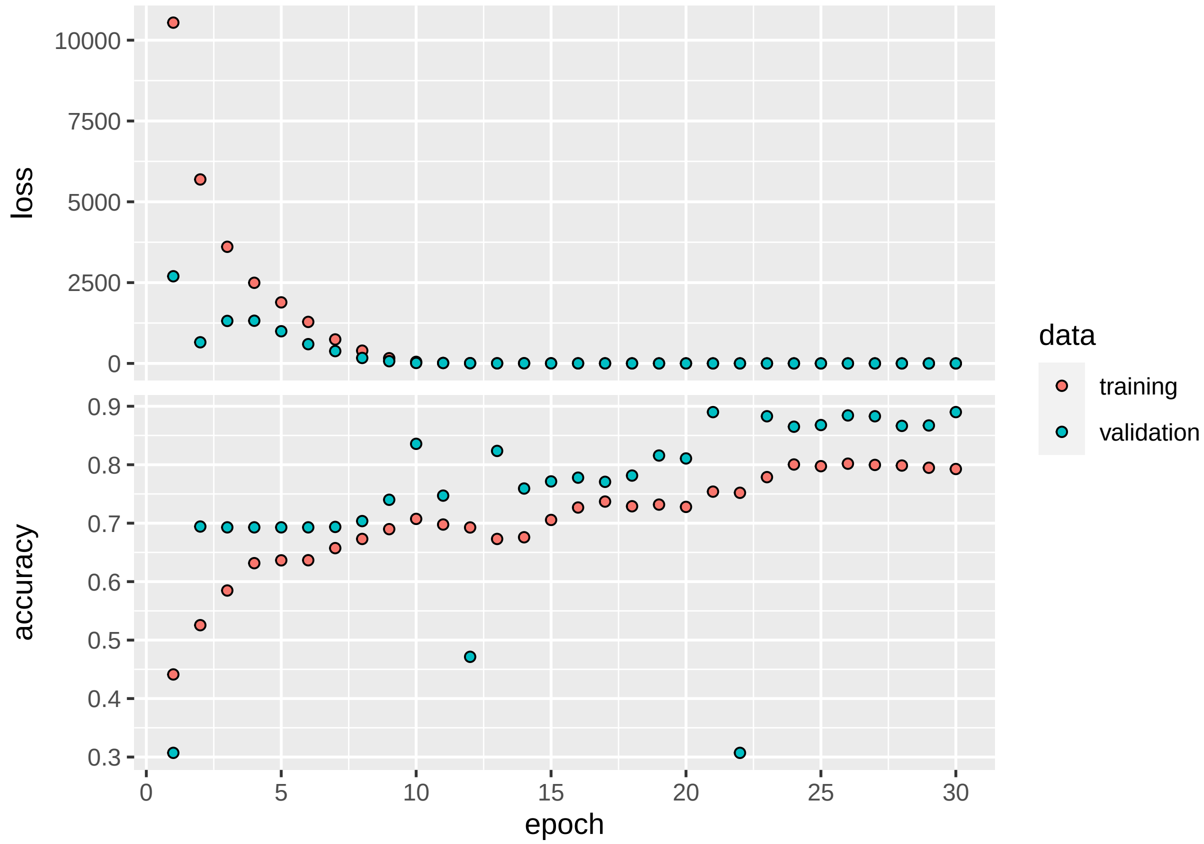

```
test_pred_2_03 <-  predict(model_nn_2_03, default_test_feature)
test_pred_2_03_label <- apply(test_pred_2_03, 1, which.max) - 1
train_cm <- confusionMatrix(data =as.factor(test_pred_2_03_label), reference = as.factor(default_test_ta
ggplot(data = data.frame(train_cm$table), aes(x = Prediction, y = Reference, fill = Freq)) +
  geom_tile() +
  geom_text(aes(label = Freq), color = "white") +
  labs(title = "Confusion Matrix(drop out rate = 0.3)",
       x = "Predicted",
       y = "Actual",
       fill = "Frequency")
```



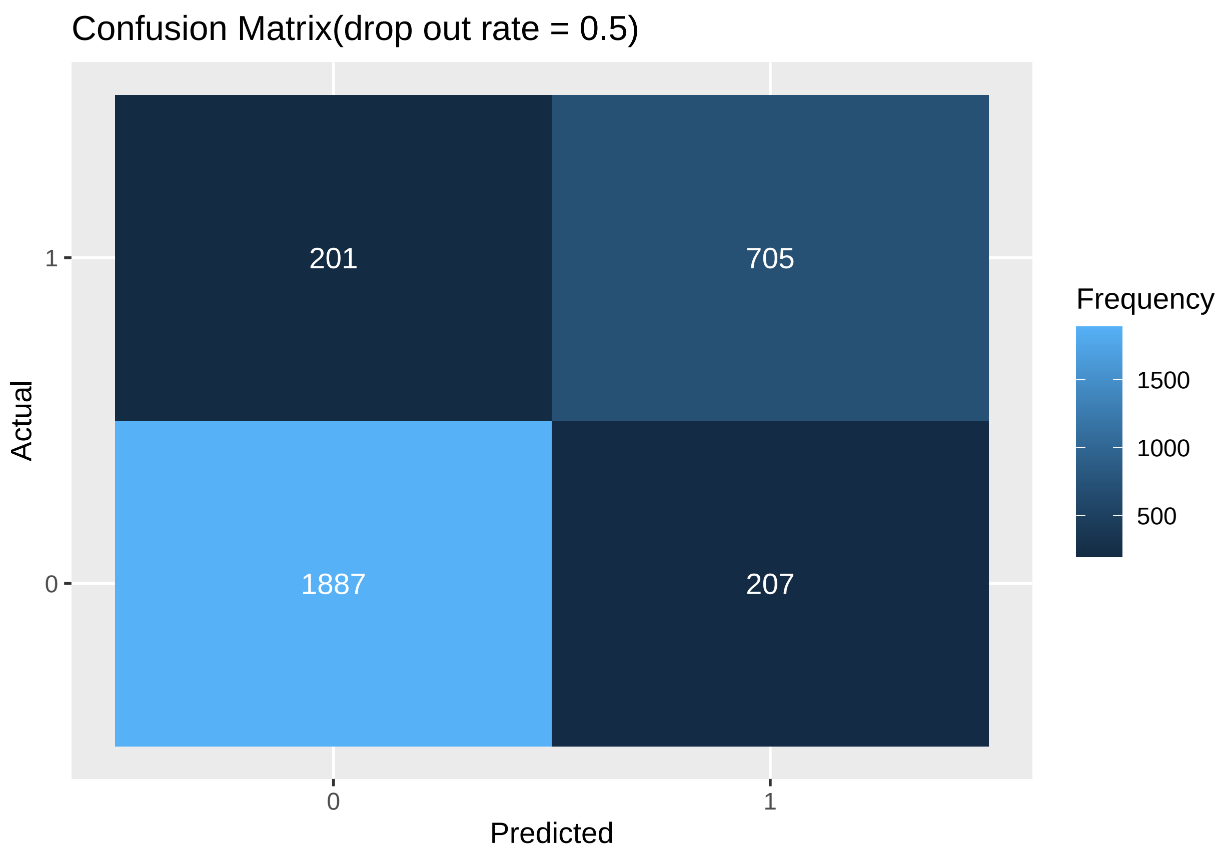

```r
# drop out rate = 0.5
model_nn_2_05 <- keras_model_sequential() %>%
  layer_dense(units = 10, activation = "relu", input_shape =c(2)) %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 2, activation = "softmax")

model_nn_2_05 %>% compile(loss = "categorical_crossentropy", optimizer = optimizer_rmsprop(), metrics =

system.time(
  history <- model_nn_2_05 %>%
    fit(default_train_feature, default_train_target_onehot, epochs = 30, batch_size = 128,
        validation_split = 0.2)
)
```

```
##    user  system elapsed
##   4.023   0.386   3.852
```

```r
plot(history, smooth = FALSE)
```




```
test_pred_2_05 <-  predict(model_nn_2_05, default_test_feature)
test_pred_2_05_label <- apply(test_pred_2_05, 1, which.max) - 1
train_cm <- confusionMatrix(data =as.factor(test_pred_2_05_label), reference = as.factor(default_test_ta
ggplot(data = data.frame(train_cm$table), aes(x = Prediction, y = Reference, fill = Freq)) +
  geom_tile() +
  geom_text(aes(label = Freq), color = "white") +
  labs(title = "Confusion Matrix(drop out rate = 0.5)",
       x = "Predicted",
       y = "Actual",
       fill = "Frequency")
```



3. Use the code in ISLR1 Section 10.9.3 to create a CNN for the CIFAR data. Compare the accuracy results to 4 or 5 modifications of your choice, such as:

```
cifar100 <- dataset_cifar100()
x_train <- cifar100$train$x
y_train <- cifar100$train$y
x_test <- cifar100$test$x
y_test <- cifar100$test$y
```

```
x_train_3 <- x_train/255
x_test_3 <- x_test/255
y_train_3 <- to_categorical(y_train, 100)
y_test_3 <- to_categorical(y_test, 100)
```

```
#model_3 <- keras_model_sequential() %>%
#  layer_conv_2d(filters = 32, kernel_size = c(3, 3), padding = "same", activation = "relu", input_shap
#  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
#  layer_conv_2d(filters = 64, kernel_size = c(3, 3), padding = "same", activation = "relu") %>%
#  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
#  layer_conv_2d(filters = 128, kernel_size = c(3, 3), padding = "same", activation = "relu") %>%
#  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
#  layer_conv_2d(filters = 256, kernel_size = c(3, 3), padding = "same", activation = "relu") %>%
#  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
#  layer_flatten() %>%
#  layer_dropout(rate = 0.5) %>%
```

```
#  layer_dense(units = 512, activation = "relu") %>%
#  layer_dense(units = 100, activation = "softmax")
```

a. Changing max pooling to average pooling

```
model_a <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3, 3), padding = "same", activation = "relu", input_shape
  layer_average_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3, 3), padding = "same", activation = "relu") %>%
  layer_average_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 128, kernel_size = c(3, 3), padding = "same", activation = "relu") %>%
  layer_average_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 256, kernel_size = c(3, 3), padding = "same", activation = "relu") %>%
  layer_average_pooling_2d(pool_size = c(2, 2)) %>%
  layer_flatten() %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 512, activation = "relu") %>%
  layer_dense(units = 100, activation = "softmax")

summary(model_a)
```

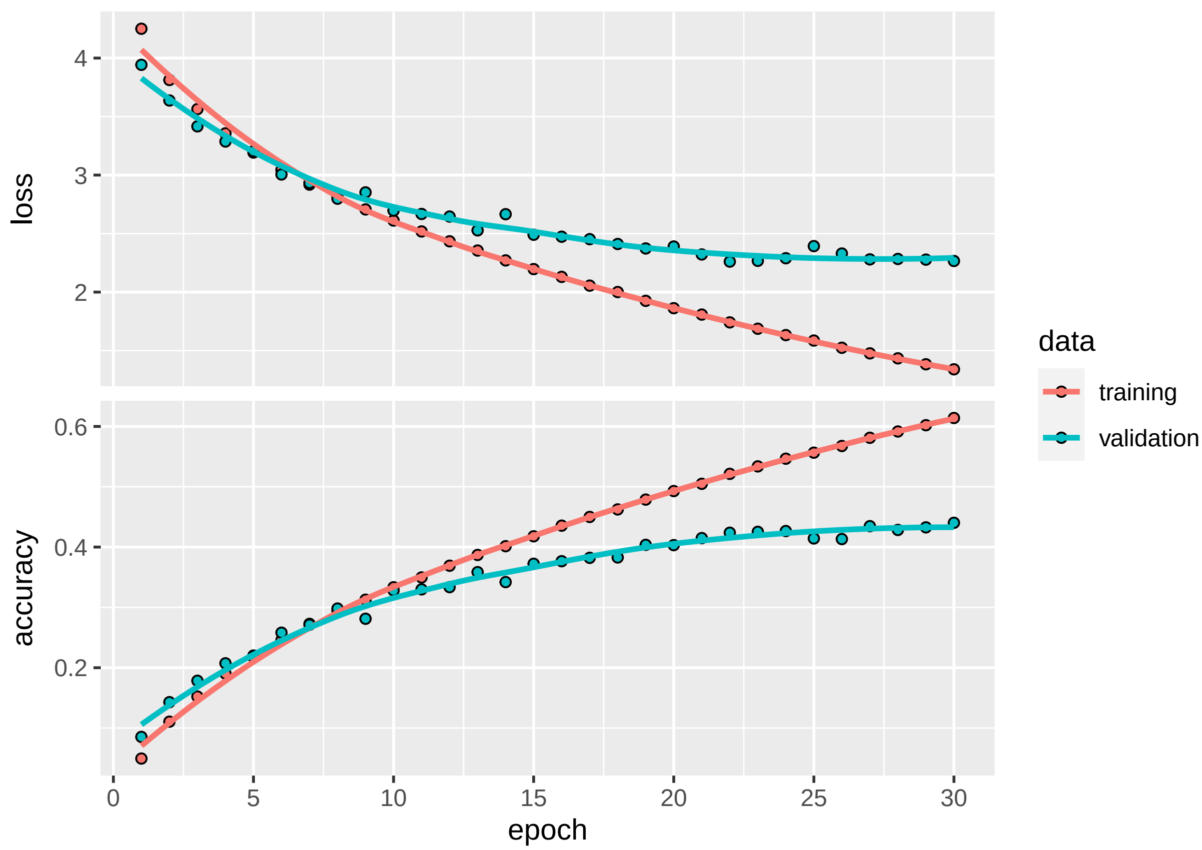
```
## Model: "sequential_3"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                      Param #
## ================================================================================
##  conv2d_3 (Conv2D)                  (None, 32, 32, 32)                896
##  average_pooling2d_3 (AveragePoolin  (None, 16, 16, 32)               0
##  g2D)
##  conv2d_2 (Conv2D)                  (None, 16, 16, 64)                18496
##  average_pooling2d_2 (AveragePoolin  (None, 8, 8, 64)                 0
##  g2D)
##  conv2d_1 (Conv2D)                  (None, 8, 8, 128)                 73856
##  average_pooling2d_1 (AveragePoolin  (None, 4, 4, 128)                0
##  g2D)
##  conv2d (Conv2D)                    (None, 4, 4, 256)                 295168
##  average_pooling2d (AveragePooling2  (None, 2, 2, 256)                0
##  D)
##  flatten (Flatten)                  (None, 1024)                      0
##  dropout_3 (Dropout)                (None, 1024)                      0
##  dense_7 (Dense)                    (None, 512)                       524800
##  dense_6 (Dense)                    (None, 100)                       51300
## ================================================================================
## Total params: 964,516
## Trainable params: 964,516
## Non-trainable params: 0
## ________________________________________________________________________________
```

```
model_a %>% compile(loss = "categorical_crossentropy", optimizer = optimizer_rmsprop(), metrics = c("ac
history <- model_a %>% fit(x_train_3, y_train_3, epochs = 30, batch_size = 128, validation_split = 0.2)
```

```
accuracy <- function(pred, truth)
  mean(drop(pred) == drop(truth))
y_pred_a <- predict(model_a, x_test_3)
accuracy(y_pred_a, y_test_3)
```

```
## [1] 2e-06
```

```
plot(history)
```



b. Change to pooling size

```
model_b <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3, 3), padding = "same", activation = "relu", input_shape
  layer_average_pooling_2d(pool_size = c(3, 3)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3, 3), padding = "same", activation = "relu") %>%
  layer_average_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 128, kernel_size = c(3, 3), padding = "same", activation = "relu") %>%
  layer_average_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 256, kernel_size = c(3, 3), padding = "same", activation = "relu") %>%
  layer_average_pooling_2d(pool_size = c(2, 2)) %>%
  layer_flatten() %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 512, activation = "relu") %>%
  layer_dense(units = 100, activation = "softmax")
```

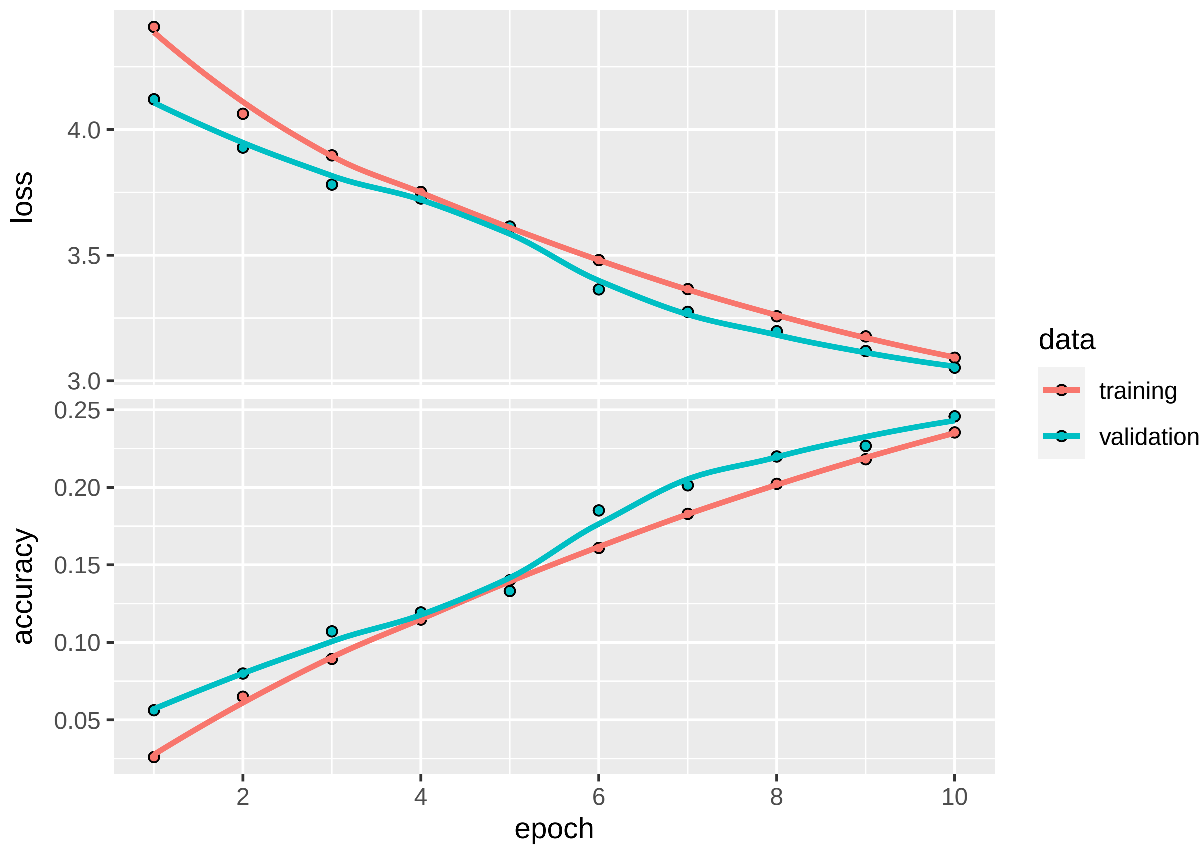
```r
summary(model_b)
```

```
## Model: "sequential_4"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## ================================================================================
##  conv2d_7 (Conv2D)                  (None, 32, 32, 32)              896
##  average_pooling2d_7 (AveragePoolin  (None, 10, 10, 32)             0
##  g2D)
##  conv2d_6 (Conv2D)                  (None, 10, 10, 64)              18496
##  average_pooling2d_6 (AveragePoolin  (None, 5, 5, 64)               0
##  g2D)
##  conv2d_5 (Conv2D)                  (None, 5, 5, 128)               73856
##  average_pooling2d_5 (AveragePoolin  (None, 2, 2, 128)              0
##  g2D)
##  conv2d_4 (Conv2D)                  (None, 2, 2, 256)               295168
##  average_pooling2d_4 (AveragePoolin  (None, 1, 1, 256)              0
##  g2D)
##  flatten_1 (Flatten)                (None, 256)                     0
##  dropout_4 (Dropout)                (None, 256)                     0
##  dense_9 (Dense)                    (None, 512)                     131584
##  dense_8 (Dense)                    (None, 100)                     51300
## ================================================================================
## Total params: 571,300
## Trainable params: 571,300
## Non-trainable params: 0
## ________________________________________________________________________________
```

```r
model_b %>% compile(loss = "categorical_crossentropy", optimizer = optimizer_rmsprop(), metrics = c("ac
history <- model_b %>% fit(x_train_3, y_train_3, epochs = 10, batch_size = 128, validation_split = 0.2)
y_pred_b <- predict(model_b, x_test_3)
accuracy(y_pred_b, y_test_3)
```

```
## [1] 0
```

```r
plot(history)
```



c. Varying the dropout rate

```
model_c <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3, 3), padding = "same", activation = "relu", input_shape
  layer_average_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3, 3), padding = "same", activation = "relu") %>%
  layer_average_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 128, kernel_size = c(3, 3), padding = "same", activation = "relu") %>%
  layer_average_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 256, kernel_size = c(3, 3), padding = "same", activation = "relu") %>%
  layer_average_pooling_2d(pool_size = c(2, 2)) %>%
  layer_flatten() %>%
  layer_dropout(rate = 0.1) %>%
  layer_dense(units = 512, activation = "relu") %>%
  layer_dense(units = 100, activation = "softmax")

summary(model_c)
```

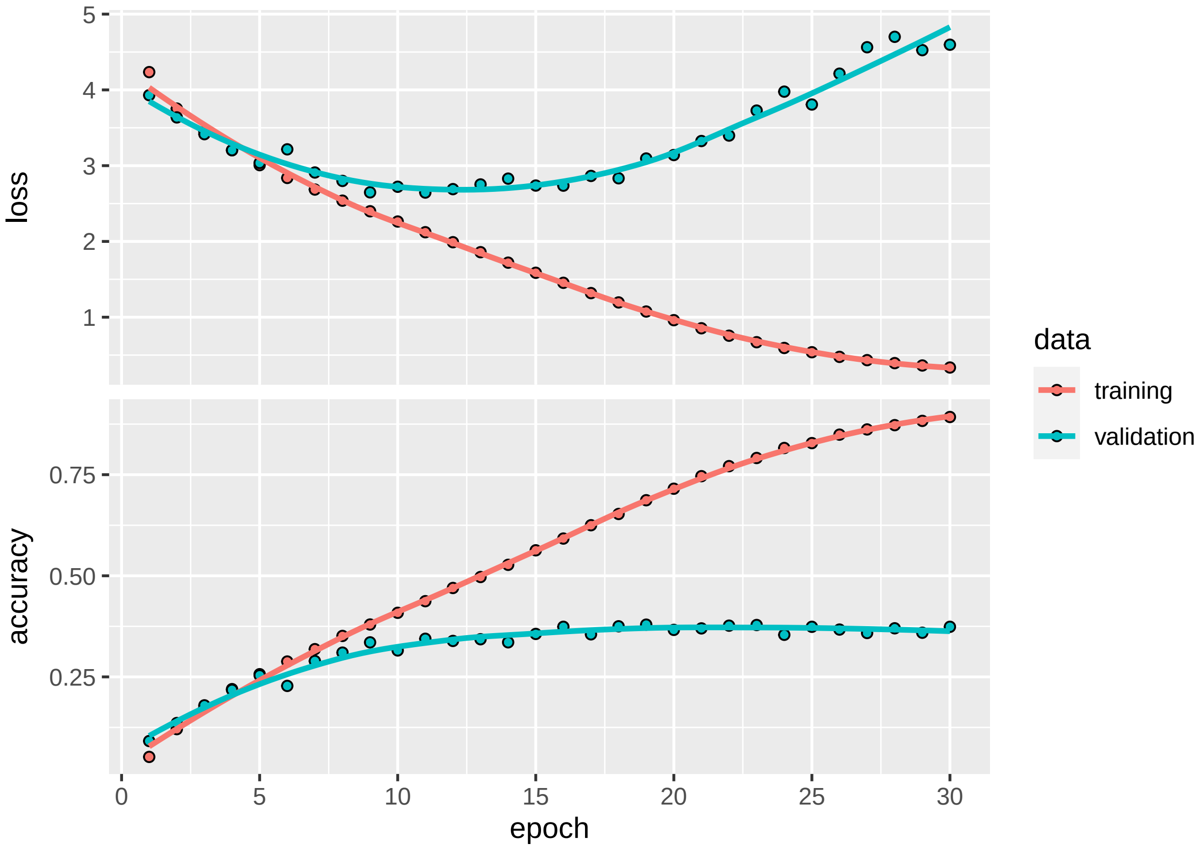
```
## Model: "sequential_5"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## ================================================================================
##  conv2d_11 (Conv2D)                 (None, 32, 32, 32)              896
##  average_pooling2d_11 (AveragePooli  (None, 16, 16, 32)             0
##  ng2D)
```

```
##  conv2d_10 (Conv2D)                 (None, 16, 16, 64)             18496
##  average_pooling2d_10 (AveragePooli  (None, 8, 8, 64)              0
##  ng2D)
##  conv2d_9 (Conv2D)                  (None, 8, 8, 128)              73856
##  average_pooling2d_9 (AveragePoolin  (None, 4, 4, 128)             0
##  g2D)
##  conv2d_8 (Conv2D)                  (None, 4, 4, 256)              295168
##  average_pooling2d_8 (AveragePoolin  (None, 2, 2, 256)             0
##  g2D)
##  flatten_2 (Flatten)                (None, 1024)                   0
##  dropout_5 (Dropout)                (None, 1024)                   0
##  dense_11 (Dense)                   (None, 512)                    524800
##  dense_10 (Dense)                   (None, 100)                    51300
## ================================================================================
## Total params: 964,516
## Trainable params: 964,516
## Non-trainable params: 0
## ________________________________________________________________________________
```

```
model_c %>% compile(loss = "categorical_crossentropy", optimizer = optimizer_rmsprop(), metrics = c("ac
history <- model_c %>% fit(x_train_3, y_train_3, epochs = 30, batch_size = 128, validation_split = 0.2)
y_pred_c <- predict(model_c, x_test_3)
accuracy(y_pred_c, y_test_3)
```

```
## [1] 7e-04
```

```
plot(history)
```



d. Changing the activation function to softmax

```
model_d <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3, 3), padding = "same", activation = "softmax", input_sha
  layer_average_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3, 3), padding = "same", activation = "softmax") %>%
  layer_average_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 128, kernel_size = c(3, 3), padding = "same", activation = "softmax") %>%
  layer_average_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 256, kernel_size = c(3, 3), padding = "same", activation = "softmax") %>%
  layer_average_pooling_2d(pool_size = c(2, 2)) %>%
  layer_flatten() %>%
  layer_dropout(rate = 0.1) %>%
  layer_dense(units = 512, activation = "softmax") %>%
  layer_dense(units = 100, activation = "softmax")

summary(model_d)
```

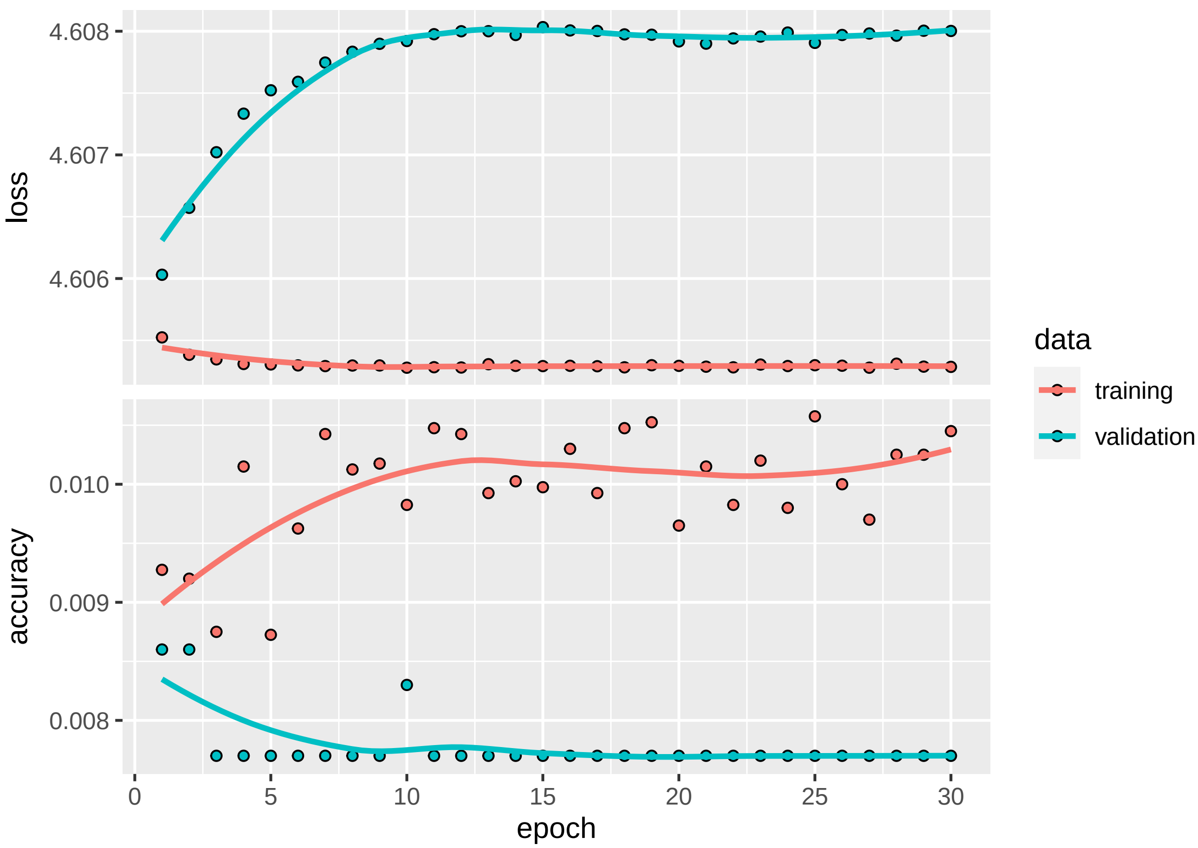
```
## Model: "sequential_6"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                  Param #
## ================================================================================
##  conv2d_15 (Conv2D)                 (None, 32, 32, 32)            896
##  average_pooling2d_15 (AveragePooli  (None, 16, 16, 32)           0
##  ng2D)
```

```
##  conv2d_14 (Conv2D)                 (None, 16, 16, 64)              18496
##  average_pooling2d_14 (AveragePooli  (None, 8, 8, 64)               0
##  ng2D)
##  conv2d_13 (Conv2D)                 (None, 8, 8, 128)               73856
##  average_pooling2d_13 (AveragePooli  (None, 4, 4, 128)              0
##  ng2D)
##  conv2d_12 (Conv2D)                 (None, 4, 4, 256)               295168
##  average_pooling2d_12 (AveragePooli  (None, 2, 2, 256)              0
##  ng2D)
##  flatten_3 (Flatten)                (None, 1024)                    0
##  dropout_6 (Dropout)                (None, 1024)                    0
##  dense_13 (Dense)                   (None, 512)                     524800
##  dense_12 (Dense)                   (None, 100)                     51300
## ================================================================================
## Total params: 964,516
## Trainable params: 964,516
## Non-trainable params: 0
## ________________________________________________________________________________
```

```
model_d %>% compile(loss = "categorical_crossentropy", optimizer = optimizer_rmsprop(), metrics = c("ac
history <- model_d %>% fit(x_train_3, y_train_3, epochs = 30, batch_size = 128, validation_split = 0.2)
y_pred_d <- predict(model_d, x_test_3)
accuracy(y_pred_d, y_test_3)
```

```
## [1] 0
```

```
plot(history)
```



loss
accuracy
epoch
4.608
4.607
4.606
0.010
0.009
0.008
0
5
10
15
20
25
30
data
training
validation