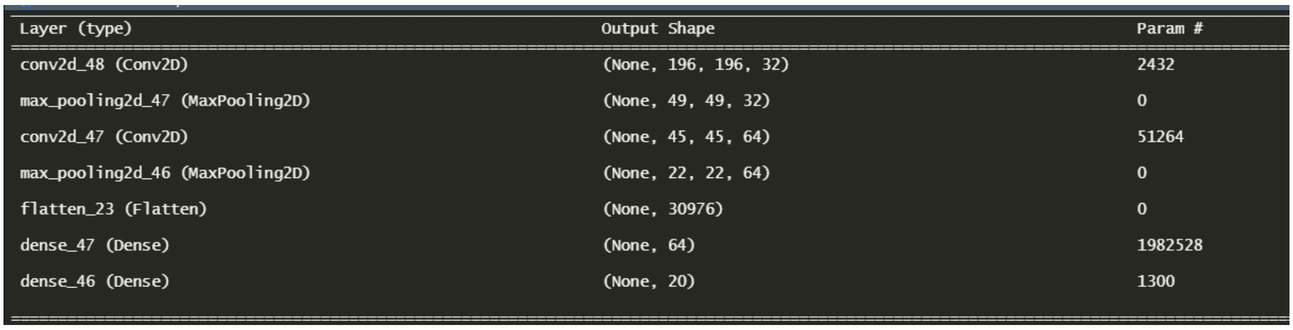
Part 2

Exercise 1

Code

```
library(keras)


keras_model_sequential()


model <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(5, 5), activation = "relu",
              input_shape = c(200, 200, 3)) %>%
  layer_max_pooling_2d(pool_size = c(4, 4)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(5, 5), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_flatten() %>%
  layer_dense(units = 64, activation = "relu") %>%
  layer_dense(units = 20, activation = "softmax") # in case of multiclass


model
```

Output



```
Layer (type)                          Output Shape                    Param #
==============================================================================
conv2d_48 (Conv2D)                    (None, 196, 196, 32)            2432
max_pooling2d_47 (MaxPooling2D)       (None, 49, 49, 32)              0
conv2d_47 (Conv2D)                    (None, 45, 45, 64)              51264
max_pooling2d_46 (MaxPooling2D)       (None, 22, 22, 64)              0
flatten_23 (Flatten)                  (None, 30976)                   0
dense_47 (Dense)                      (None, 64)                      1982528
dense_46 (Dense)                      (None, 20)                      1300
==============================================================================
```

Explanation

First of all, I must set input_shape as (200,200,3) because the height and width dimension of the images are 200 each. Also, because it's colored images, the last dimension must be 3, because this is the color channel. And for RGB images, we have 3 channels.

As this is a multi-class classification problem with 20 labels, I give the last dense layer 20 units, and set softmax as activation. Softmax will output 20 probabilities in respect to what class the neural network thinks the image belongs to.

Then, I have placed 32 in the filter argument for the first layer, and 64 in the second layer, because the filter argument is equal to the number of feature maps. I use a kernel size of 5,5 in both layers because I'm told to let the feature maps use a 5x5 filter.

Then, I add 2 max pooling layers between the 2d layers, and I have a problem making the output shape of the first layer into (48,48,32). I have tried every possible pool size value, and the closest I can come is the shape (49,49,32) by using a pool size of 4x4. But, the shape is (22,22,64) using 2x2 pool size after the second 2d layer.

Then I finish with placing a hidden DCN layer consisting of 64 units (neurons). The las DCN output layer has been explained.

I use ReLu activation function all along. We normally use that in image classification.

Exercise 2

Explanation

Now, the batch size depends on whether im on a CPU or GPU. In my case, I would choose 20 or something like that. The reason we flow in the images the way we do is because normal laptops can't cope huge loadings of images.

Now, I will only augment the training data as the valid data should be untouched. If we want a model that performs well on real data, and perform well on images in the real world, we shouldn't manipulate them. So for the validation generator I don't augment. I would only rescale by 1/255 to normalize the data.

The steps per epoch depends on training data set size and batch size. So train_size/batch_size, which means 500/20 = 25.

And the validation_steps is calculated by valid_size / batch_size = 250/20 = 10.

So batch size of 20 in training generator, steps_per_epoch = 25, and validation_steps = 10

Exercise 3

Code

```
###### Exercise 3 ######


library(keras)

complaints_train <- read.csv(file = "C:/Users/XXX/OneDrive - Aarhus universitet/Skrivebord/BI/2. semester/ML2/Exam 2022/complaints_train.csv")

complaints_test <- read.csv(file = "C:/Users/XXX/OneDrive - Aarhus universitet/Skrivebord/BI/2. semester/ML2/Exam 2022/complaints_test.csv")



set.seed(202)

options(scipen = 999)
```

```
# Split the dataset

train_proportion <- round(nrow(complaints_train)*0.05)

train <- sample(nrow(complaints_train), train_proportion)
complaints_train <- complaints_train[train, ]



# x
x_train <- c(complaints_train[,2]) # take sequences from df and turn into a list/vector
x_test <- c(complaints_test[,2]) # take sequences from df and turn into a list/vector

# y
y_train <- c(complaints_train$product)
y_train <- as.factor(y_train)
y_train <- as.integer(y_train)-1 # levels must start from 0

y_test <- c(complaints_test$product)
y_test <- as.factor(y_test)
y_test <- as.integer(y_test)-1 # levels must start from 0

# one-hot encode labels
y_train <- to_categorical(y_train, num_classes = 5)
y_test <- to_categorical(y_test, num_classes = 5)



library(stringr)

# Mean
```

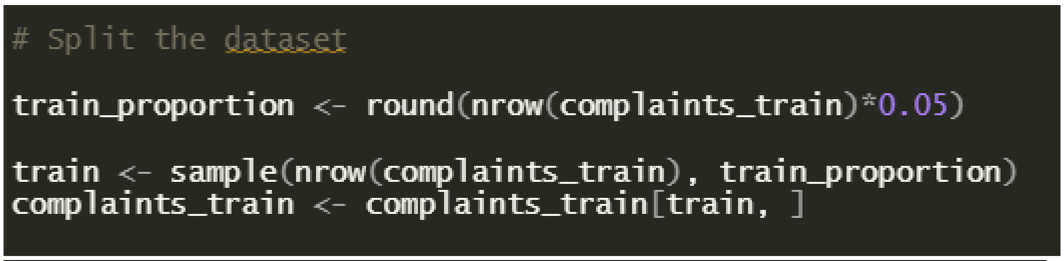
```
lengths <- lengths(gregexpr("[A-z]\\W+", x_train)) + 1L

mean <- round(mean(lengths)) # mean = 81


# Extracting words

sub <- word(x_train, start = 1L, end = mean)
library(tidyverse)
sub <- replace_na(sub, replace = "hi")
head(sub)


for (i in 1:length(x_train)) {
  if (sub[i] == "hi") {
    sub[i] = x_train[i]
  }
}

x_train <- sub
head(x_train)


sub <- word(x_test, start = 1L, end = mean)
library(tidyverse)
sub <- replace_na(sub, replace = "hi")
head(sub)


for (i in 1:length(x_test)) {
  if (sub[i] == "hi") {
    sub[i] = x_test[i]
  }
}


x_test <- sub
head(x_test)



# Define number of words to consider


num_words <- 1000  # number of words to consider as vocabulary




# turn sequences into word index


tokenizer <- text_tokenizer(num_words = num_words,
                            filters = "!\"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n",
                            lower = TRUE,
                            split = " ",
                            char_level = FALSE,
                            oov_token = NULL) %>% fit_text_tokenizer(x_train) # turn sequences into word index



sequences.train <- texts_to_sequences(tokenizer, x_train) # turn index into list of integer indices
sequences.test <- texts_to_sequences(tokenizer, x_test) # turn index into list of integer indices
head(sequences.train)

# List padding (turning the sequences into a tensor)
```

x_train <- pad_sequences(sequences.train, maxlen = mean) # Turns the lists of integers into a 2D integer tensor of shape (samples, maxlen)

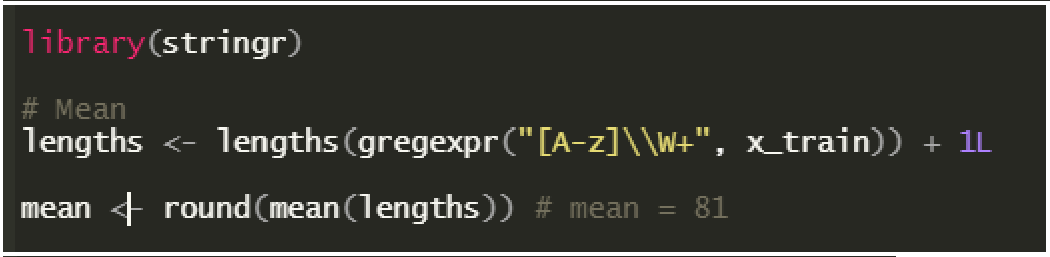
x_test <- pad_sequences(sequences.test, maxlen = mean) # Turns the lists of integers into a 2D integer tensor of shape (samples, maxlen)

head(x_train) # not at all as sparse as in one-hot encoding. We see for the dirst complaint, the first 9 embedding dimensions aren't activated.

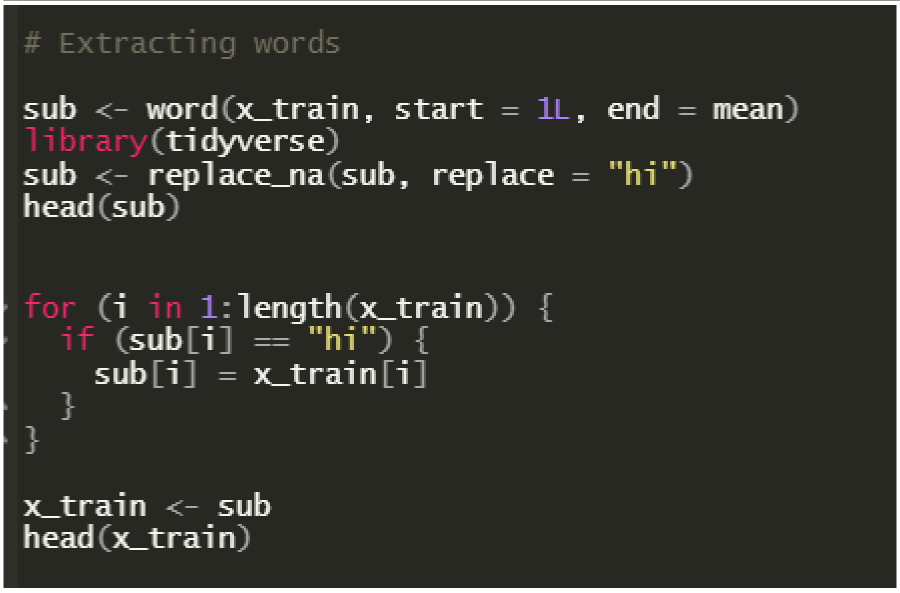
Output



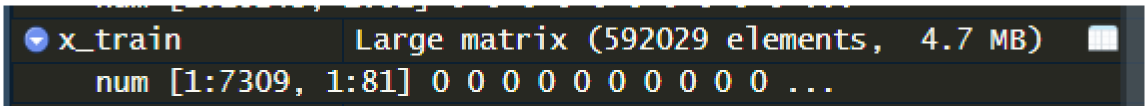
```
# Split the dataset

train_proportion <- round(nrow(complaints_train)*0.05)

train <- sample(nrow(complaints_train), train_proportion)
complaints_train <- complaints_train[train, ]
```



```
library(stringr)

# Mean
lengths <- lengths(gregexpr("[A-z]\\W+", x_train)) + 1L

mean <- round(mean(lengths)) # mean = 81
```



```
# Extracting words

sub <- word(x_train, start = 1L, end = mean)
library(tidyverse)
sub <- replace_na(sub, replace = "hi")
head(sub)


for (i in 1:length(x_train)) {
  if (sub[i] == "hi") {
    sub[i] = x_train[i]
  }
}

x_train <- sub
head(x_train)
```



```
x_train          Large matrix (592029 elements,  4.7 MB)
   num [1:7309, 1:81] 0 0 0 0 0 0 0 0 0 0 ...
```

Explanation

I have inserted the complete code for this task, as well as some of the most important results I find. So, I start by splitting the dataset by taking 5% of the train data.

Then I find the mean using the string library. Then I build a loop that cuts the sentences to the mean (81 words). Then I create the vocabulary using the tokenizer and lastly do list padding ending up with a matrix that has 81 dimensions.

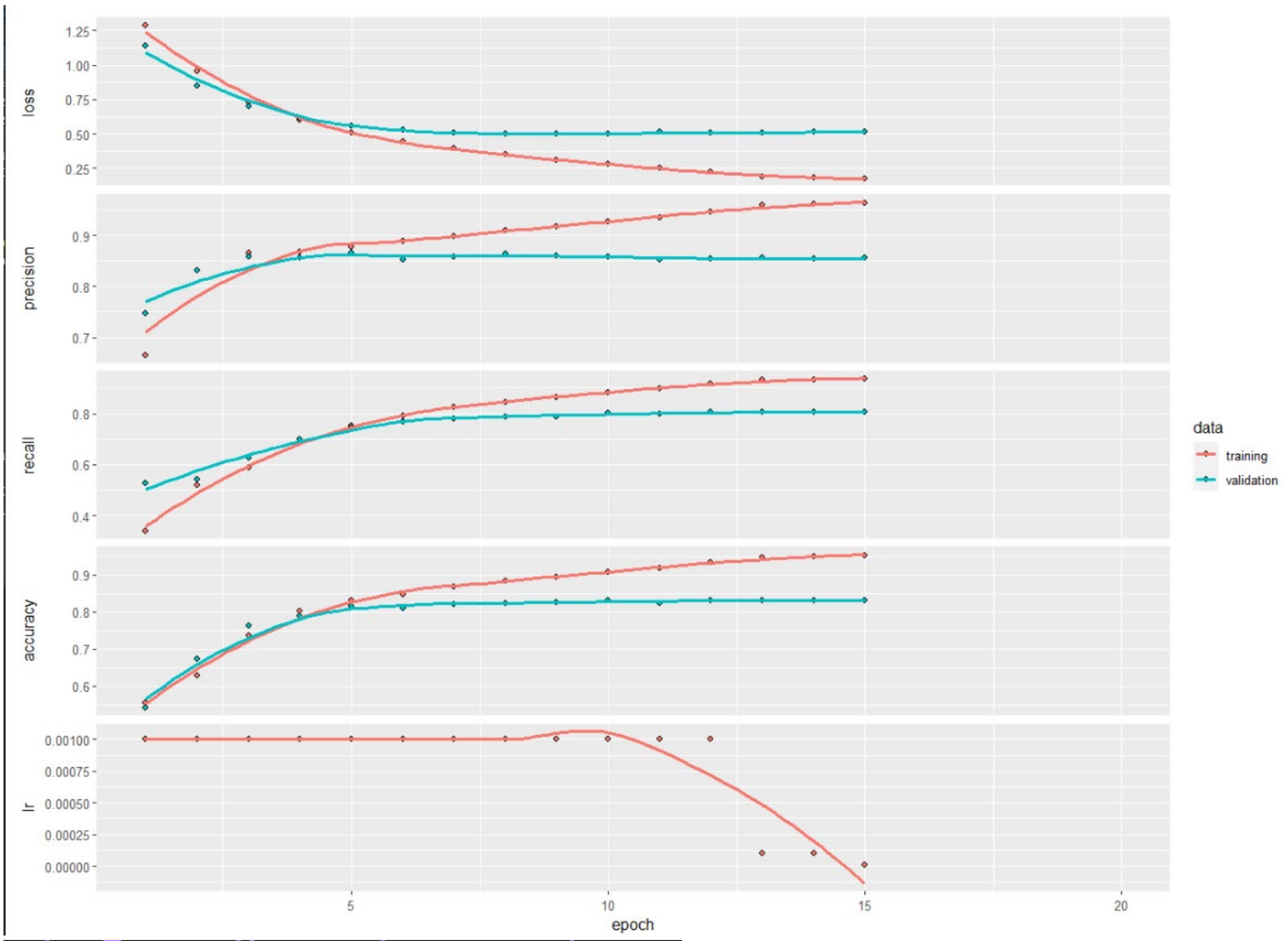
Exercise 4

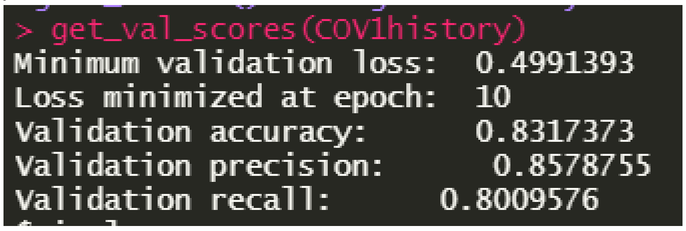
Code

```
###### Exercise 4 ######



get_val_scores <- function(history){


  min_loss <-  min(history$metrics$val_loss)

  min_epoch <- which.min(history$metrics$val_loss)

  min_acc <- history$metrics$val_accuracy[which.min(history$metrics$val_loss)]

  min_precision <- history$metrics$val_precision[which.min(history$metrics$val_loss)]

  min_recall <- history$metrics$val_recall[which.min(history$metrics$val_loss)]


  cat('Minimum validation loss:  ')

  cat(min_loss)

  cat('\n')


  cat('Loss minimized at epoch:  ')

  cat(min_epoch)

  cat('\n')


  cat('Validation accuracy:       ')

  cat(min_acc)

  cat('\n')


  cat('Validation precision:      ')

  cat(min_precision)

  cat('\n')
  cat('Validation recall:      ')
  cat(min_recall)
  cat('\n')


  return(list(min_loss = min_loss,
            min_epoch = min_epoch,
            min_acc = min_acc,
            min_precision = min_precision,
            min_recall = min_recall))

}


# 1st model

embedding_dim <- 50
embedding_dim2 <- round(num_words ^ (1/4))
embedding_dim3 <- 64
embedding_dim4 <- 100



k_clear_session()
set_random_seed(123)

COV1 <- keras_model_sequential() %>%
  layer_embedding(input_dim = num_words,
              input_length = mean,
              output_dim = embedding_dim4) %>%
  layer_conv_1d(filters = 32, kernel_size = 7, activation = "relu") %>%
  layer_global_max_pooling_1d() %>% # either layer_flatten or globalmaxpooling so that we turn spatial
feature maps into vectors
  layer_dense(units = 16, activation = "relu") %>%
  layer_dense(units = 5, activation = "softmax")

COV1 %>% compile(
  optimizer = "rmsprop",
  loss = "categorical_crossentropy",
  metrics = list(metric_precision(), metric_recall(),
                'accuracy')
)

COV1history <- COV1 %>% fit(
  x_train, y_train,
  epochs = 20,
  batch_size = 128,
  validation_split = 0.2,
  verbose = 1,
  callbacks = list(
    callback_reduce_lr_on_plateau(patience = 2, factor = 0.1),
    callback_early_stopping(patience = 5,
                            restore_best_weights = TRUE,
                            min_delta = 0.0001))
)

plot(COV1history)

get_val_scores(COV1history)
```

Output

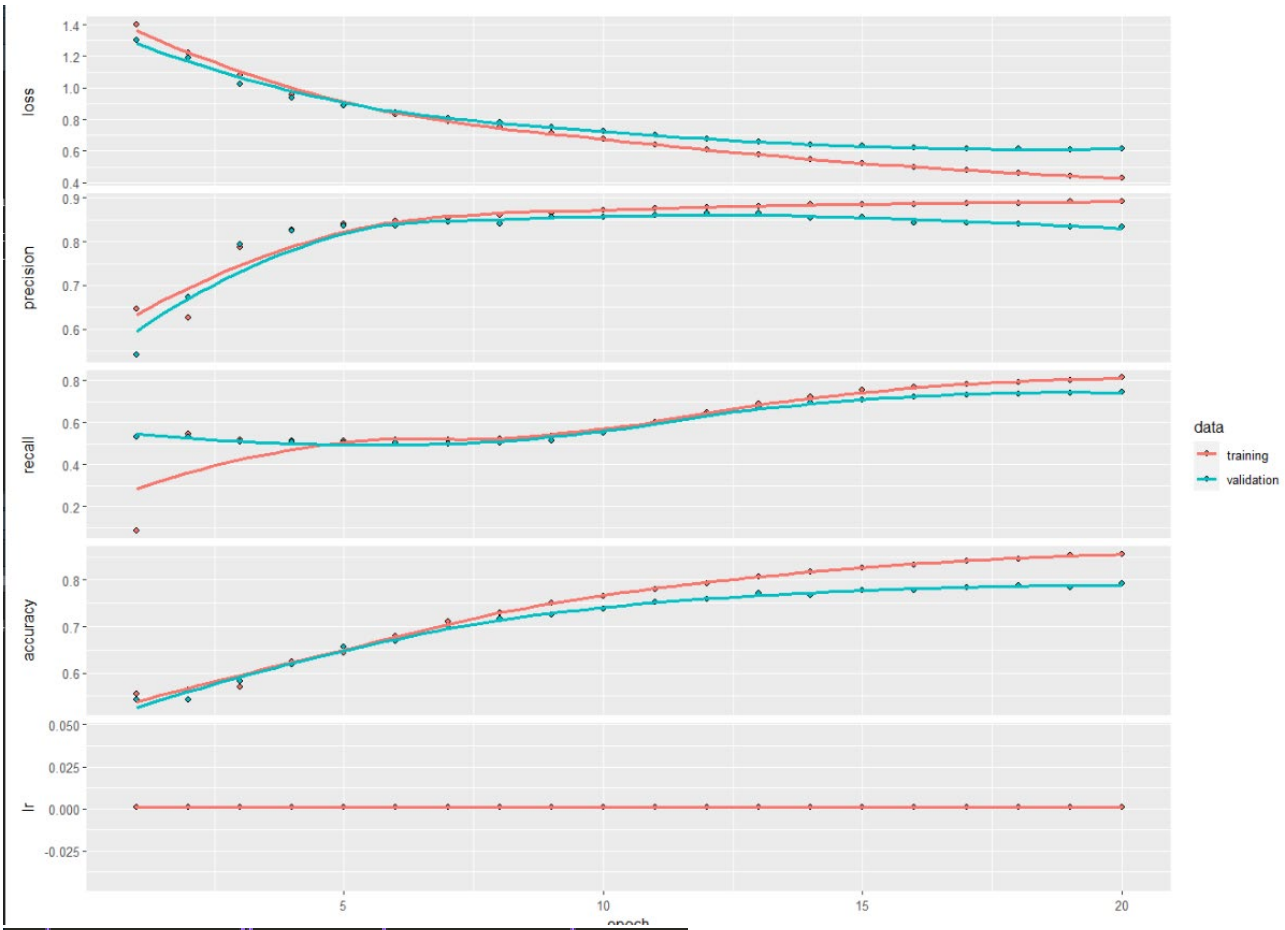
1st model (50 embedding dimensions)

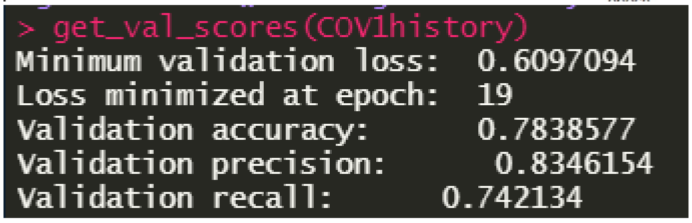




```
> get_val_scores(COV1history)
Minimum validation loss:   0.4991393
Loss minimized at epoch:   10
Validation accuracy:       0.8317373
Validation precision:       0.8578755
Validation recall:       0.8009576
```

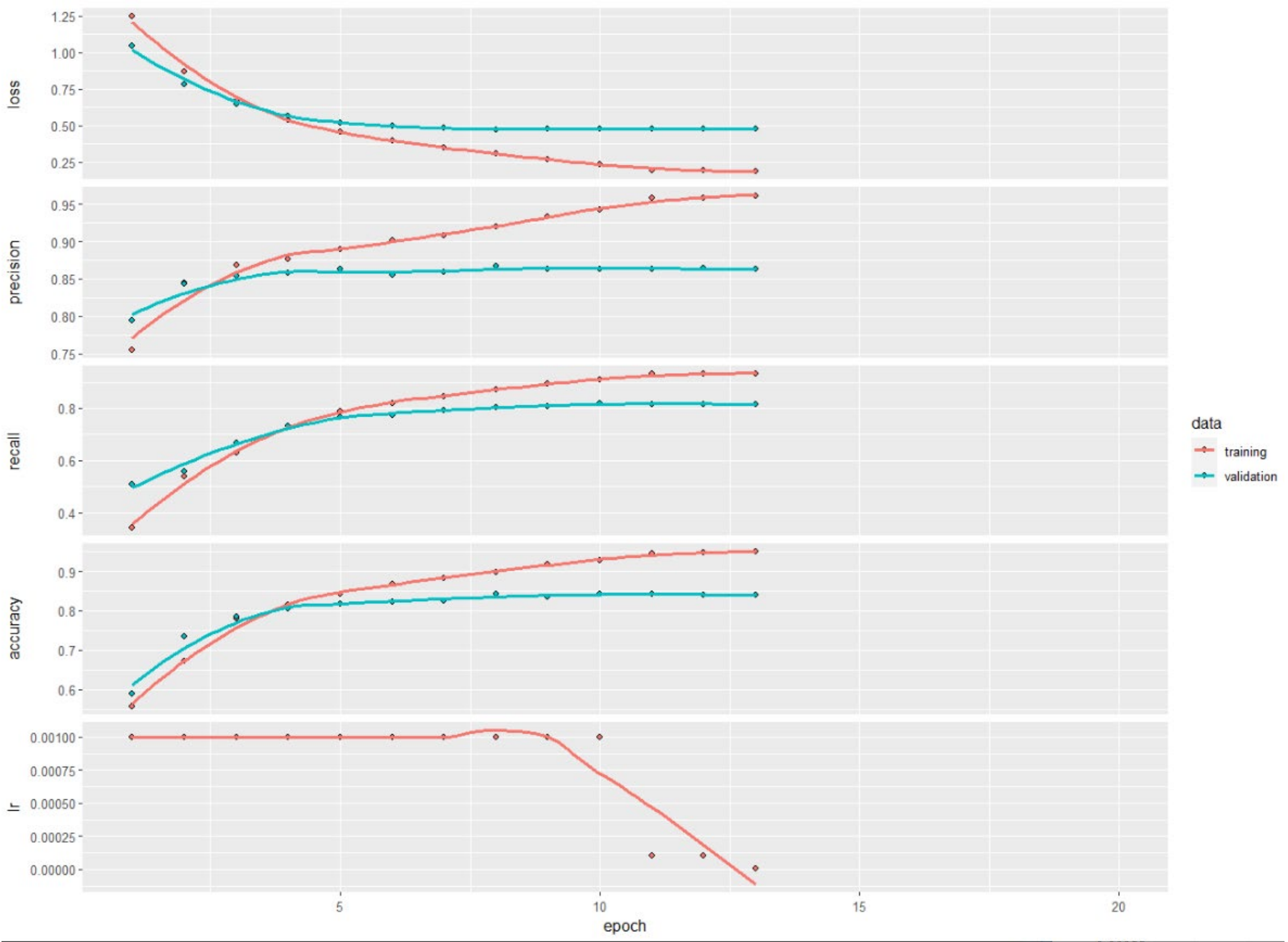
2nd model (6 embedding dimensions):

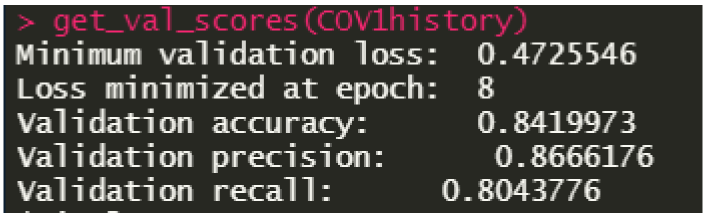




```
> get_val_scores(COV1history)
Minimum validation loss:  0.6097094
Loss minimized at epoch:  19
Validation accuracy:      0.7838577
Validation precision:      0.8346154
Validation recall:     0.742134
```

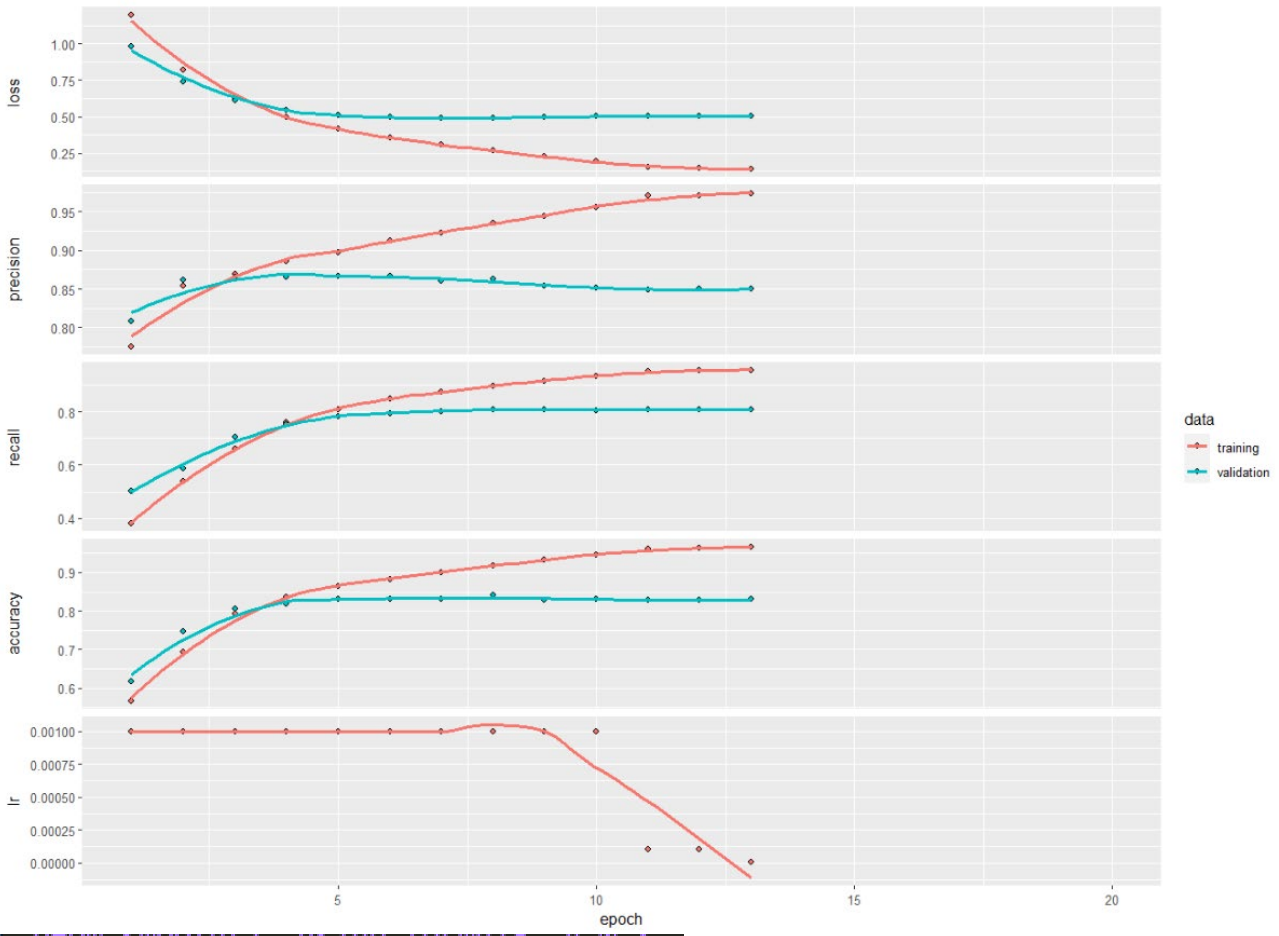
3rd model (64 dimensions):

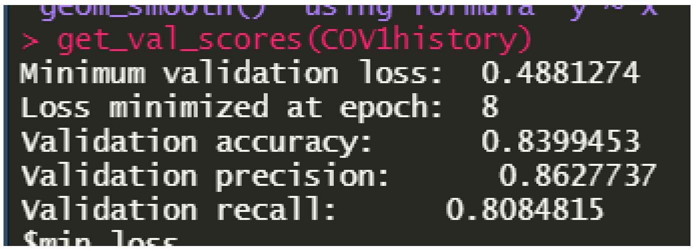




```
> get_val_scores(COV1history)
Minimum validation loss:  0.4725546
Loss minimized at epoch:  8
Validation accuracy:      0.8419973
Validation precision:      0.8666176
Validation recall:      0.8043776
```

4th model (128 embedding dimensions):




```
> get_val_scores(COV1history)
Minimum validation loss:  0.4881274
Loss minimized at epoch:  8
Validation accuracy:      0.8399453
Validation precision:      0.8627737
Validation recall:      0.8084815
```

Explanation

I have build a convenience function to track the minimum validation loss. I have created 4 different models using respectively 50, 6, 64, 100.

The reason I did 6 is because Google's research guideline for the embedding dimensions are round(num_words ^ (1/4)).

However, the 50-model had 0.4991 in loss, the 6-model had 0.6097, the 64-model had 0.4725, and the 100-model had 0.4881.

I prefer the model with 64 embedding dimensions, as this one has the lowest loss. Also, the model with 6 dimensions seems to be underfitting a lot, because looking at the plot, it gets the minimum loss at epoch 19 (I train for 20). So I would never use that one.

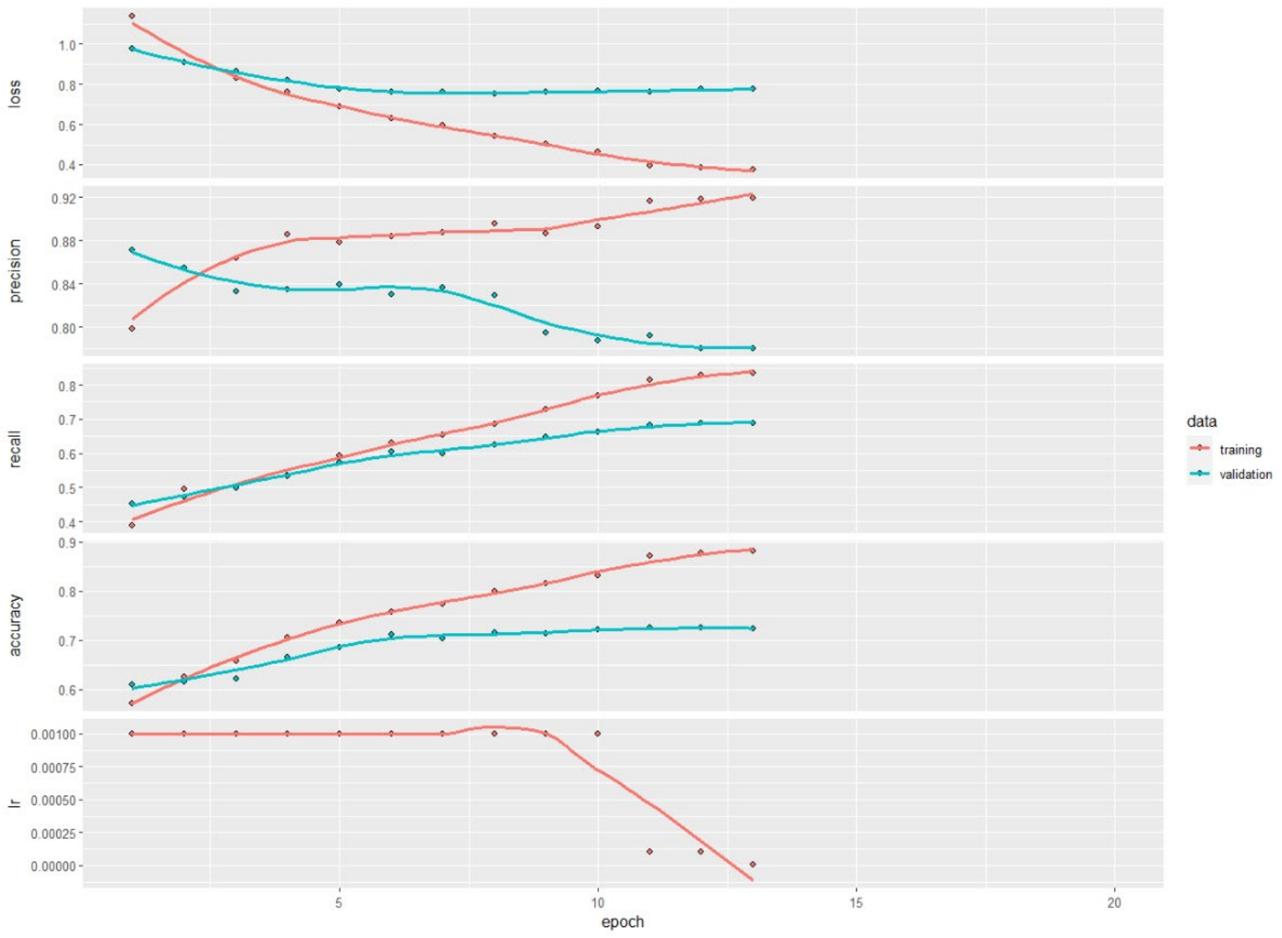
The 64-embedding model seems to be reasonable, in that it starts overfitting the same time as the model with 100 dimensions. And also, it takes shorter time to run the 64-model. I think it hasn't completely reached borderline between overfitting and underfitting, but due to time constraints I cant provide another model. Ideally I would like a model that overfitted a bit more, so that I could regularize it a bit.

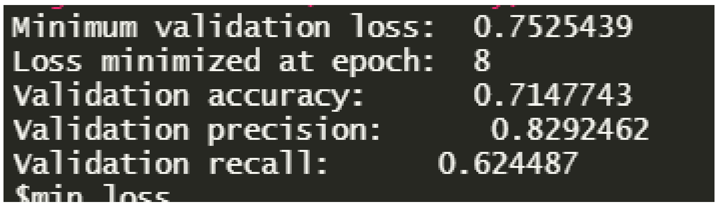
Exercise 5

Code

```
###### Exercise 5 ######





k_clear_session()

set_random_seed(123)


COV1 <- keras_model_sequential() %>%

  layer_embedding(input_dim = num_words,

                input_length = mean,

                output_dim = embedding_dim3) %>%

  layer_conv_1d(filters = 32, kernel_size = 7, activation = "relu") %>%

  layer_gru(units = embedding_dim) %>%

  layer_dense(units = 16, activation = "relu") %>%

  layer_dense(units = 5, activation = "softmax")


COV1 %>% compile(

  optimizer = "rmsprop",

  loss = "categorical_crossentropy",

  metrics = list(metric_precision(), metric_recall(),

               'accuracy')

)


COV1history <- COV1 %>% fit(

  x_train, y_train,

  epochs = 20,

  batch_size = 128,
  validation_split = 0.2,

  verbose = 1,

  callbacks = list(

    callback_reduce_lr_on_plateau(patience = 2, factor = 0.1),

    callback_early_stopping(patience = 5,

                            restore_best_weights = TRUE,

                            min_delta = 0.0001))

)


plot(COV1history)


get_val_scores(COV1history)
```

Output




```
Minimum validation loss:  0.7525439
Loss minimized at epoch:  8
Validation accuracy:      0.7147743
Validation precision:      0.8292462
Validation recall:      0.624487
```

Explanation

So, if it is important that more recent data points should be interpreted differently from older data points, then a 1D convnet will fail (eg. we saw that in temperature forecasting example from the lecture). However, for many text problems, this will not be a problem, because patterns of keywords associated with a output class are informative independently of where they are found in the input sentences.

However, to fix and check the order sensitivity, I will use 1D convnet (a 1D conv layer) as a preprocessing step before an RNN (a gru layer).

It works by letting the 1D convnet turn long input sequence into much shorter sequences of higher level features. This sequence of extracted features then becomes the input to the RNN part of the network.

I see that my new 1D + RNN model is performing worse than the pure 1D network. This new model has a loss of 0.75, compared to the earlier 0.4725. Thus, the sentence order inside the window doesn't seem to be important for this dataset.

Exercise 6

Code

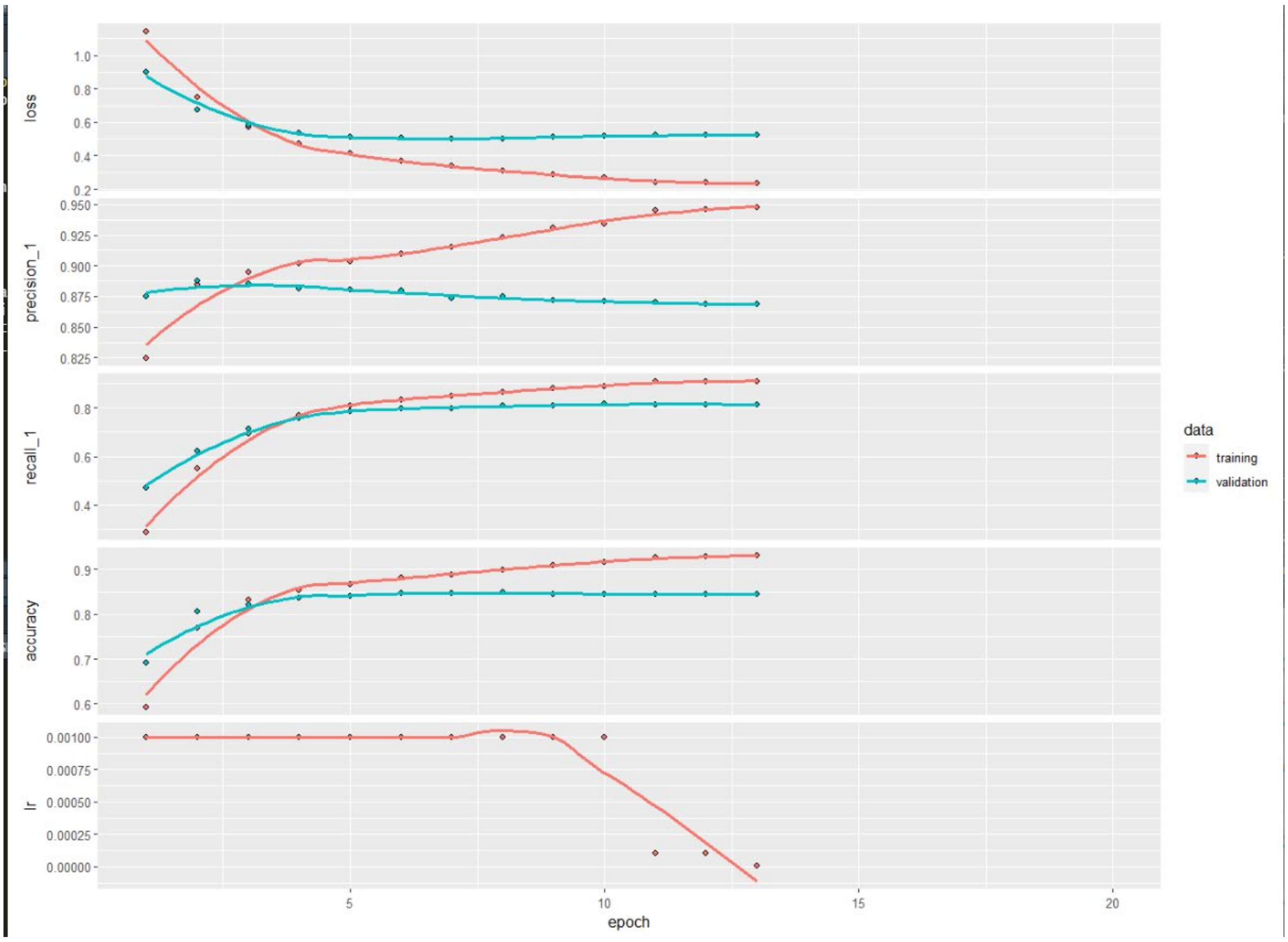
```
###### Exercise 6 ######




library(keras)

complaints_train <- read.csv(file = "C:/Users/XXX/OneDrive - Aarhus universitet/Skrivebord/BI/2. semester/ML2/Exam 2022/complaints_train.csv")

complaints_test <- read.csv(file = "C:/Users/XXX/OneDrive - Aarhus universitet/Skrivebord/BI/2. semester/ML2/Exam 2022/complaints_test.csv")



set.seed(202)

options(scipen = 999)


# Split the dataset


train_proportion <- round(nrow(complaints_train)*0.05)
```

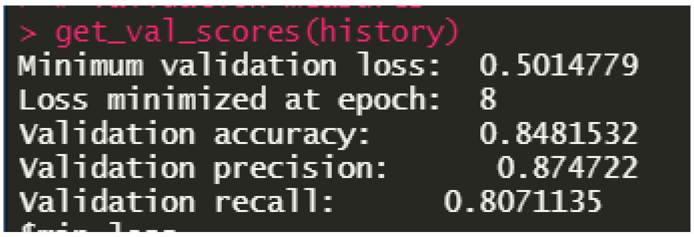
```
train <- sample(nrow(complaints_train), train_proportion)

complaints_train <- complaints_train[train, ]



# x

x_train <- c(complaints_train[,2]) # take sequences from df and turn into a list/vector

x_test <- c(complaints_test[,2]) # take sequences from df and turn into a list/vector


# y

y_train <- c(complaints_train$product)

y_train <- as.factor(y_train)

y_train <- as.integer(y_train)-1 # levels must start from 0


y_test <- c(complaints_test$product)

y_test <- as.factor(y_test)

y_test <- as.integer(y_test)-1 # levels must start from 0


# one-hot encode labels

y_train <- to_categorical(y_train, num_classes = 5)

y_test <- to_categorical(y_test, num_classes = 5)




library(stringr)


# Mean

lengths <- lengths(gregexpr("[A-z]\\W+", x_train)) + 1L


mean <- round(mean(lengths)) # mean = 81
```

```
# Extracting words

sub <- word(x_train, start = 1L, end = mean)
library(tidyverse)
sub <- replace_na(sub, replace = "hi")
head(sub)


for (i in 1:length(x_train)) {
  if (sub[i] == "hi") {
    sub[i] = x_train[i]
  }
}

x_train <- sub
head(x_train)


sub <- word(x_test, start = 1L, end = mean)
library(tidyverse)
sub <- replace_na(sub, replace = "hi")
head(sub)


for (i in 1:length(x_test)) {
  if (sub[i] == "hi") {
    sub[i] = x_test[i]
  }
}


x_test <- sub
head(x_test)




tokenizer <- text_tokenizer(num_words = num_words,
                            filters = "!\"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n",
                            lower = TRUE,
                            split = " ",
                            char_level = FALSE,
                            oov_token = NULL) %>% fit_text_tokenizer(x_train) # turn sequences into a word index

# what are the most common words?



tokenizer$word_index %>%
  head()
# most common word is credit



# Not useful here, but it gives word index:
word_index <- tokenizer$word_index # word index
cat("Found", length(word_index), "unique tokens.\n") # how many unique tokens we have
# 43811 unique tokens!
```

```
one_hot_results_train <- texts_to_matrix(tokenizer, x_train, mode = "binary") # This one hot encodes the data without need of embedding and padding

one_hot_results_test <- texts_to_matrix(tokenizer, x_test, mode = "binary") # This one hot encodes the data without need of embedding and padding


# ^^ each column represents one token/word (one of the 1000 most popular words), and 1 row corresponds to 1 tweet. So if 2 of the most common words are used in a tweet, then 2 cells has a "1" value

dim(one_hot_results_train)



model <- keras_model_sequential() %>%
  layer_dense(units = 32, activation = "relu",
            input_shape = c(num_words)) %>% # we don't even need to set input shape argument
  layer_dense(units = 5, activation = "softmax")


model %>% compile(
  optimizer = "rmsprop",
  loss = "categorical_crossentropy",
  metrics = list(metric_precision(), metric_recall(),
              'accuracy')
)


history <- model %>% fit(
  one_hot_results_train, y_train,
  epochs = 20,
  batch_size = 128,
  validation_split = 0.2,
  verbose = 0,
  callbacks = list(
    callback_reduce_lr_on_plateau(patience = 2, factor = 0.1),
    callback_early_stopping(patience = 5,
                    restore_best_weights = TRUE,
                    min_delta = 0.0001)
  )
)

# Plot
plot(history)
# Validation measures
get_val_scores(history)
```

Output





```
> get_val_scores(history)
Minimum validation loss:  0.5014779
Loss minimized at epoch:  8
Validation accuracy:      0.8481532
Validation precision:      0.874722
Validation recall:      0.8071135
```

Explanation

It would be a one-hot encoded model, that it actually easier and simpler than the embedding model. It's an obvious contender because this classification task is relatively simple, and we don't seem to get much better results by using more complex models such as 1D, 1D+GRU and I can also imagine BRNN or clean GRU/LSTM would not be extremely more better.

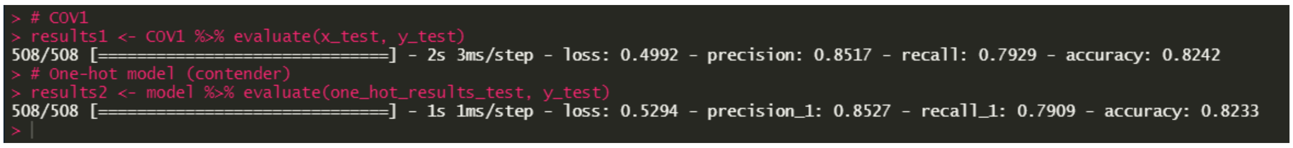
The new one-hot encoded DCN model is getting a loss of 0.5, and thus it is slightly worse than the 0.4725 we got from a 64-embedding model. However, the one-hot encoded model is much faster to run. This shows that word occurrences are important for this dataset.

Exercise 7

Code

```
###### Exercise 7 ######


# COV1

results1 <- COV1 %>% evaluate(x_test, y_test)


# One-hot model (contender)

results2 <- model %>% evaluate(one_hot_results_test, y_test)
```

Output



```
> # COV1
> results1 <- COV1 %>% evaluate(x_test, y_test)
508/508 [==============================] - 2s 3ms/step - loss: 0.4992 - precision: 0.8517 - recall: 0.7929 - accuracy: 0.8242
> # One-hot model (contender)
> results2 <- model %>% evaluate(one_hot_results_test, y_test)
508/508 [==============================] - 1s 1ms/step - loss: 0.5294 - precision_1: 0.8527 - recall_1: 0.7909 - accuracy: 0.8233
> |
```

Explanation

The COV1 now has a loss of 0.4992 while the one-hot-model has a loss of 0.5294.

The same tendency goes for accuracy and the other measures. The accuracy of both models are around 82%

The majority class is a good baseline. This is in the 50s (cant remember specifically. It should however be better than the majority class if we want to use it in real life.

Also, the f1 measure is good for multiclass problems, as this looks at the overall misclassification.