Module6

Andrew Estes

4/22/2022

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.6     v dplyr   1.0.7
## v tidyr   1.1.4     v stringr 1.4.0
## v readr   2.0.0     v forcats 0.5.1

## Warning: package 'tibble' was built under R version 4.1.2

## Warning: package 'tidyr' was built under R version 4.1.2

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(keras)
```

```
## Warning: package 'keras' was built under R version 4.1.3
```

```
library(knitr)
```

```
k_clear_session()
```

```
## Loaded Tensorflow version 2.8.0
```

1

Downloads the CIFAR-10 image set and saves the dog and frog photos in the appropriate directories. (This can be lifted verbatim from the code I used. Once you've used it once, you can comment it out or otherwise set it not to run so that you're no re-creating the directories more than once.)

```
cifar <- dataset_cifar10()
cifar$train$x <- cifar$train$x[cifar$train$y %in% c(5, 6), , , ]/255
cifar$train$y <- cifar$train$y[cifar$train$y %in% c(5, 6)]
cifar$train$y <- factor(if_else(cifar$train$y == 5, "Dog", "Frog"))
cifar$test$x <- cifar$test$x[cifar$test$y %in% c(5, 6), , , ]/255
cifar$test$y <- cifar$test$y[cifar$test$y %in% c(5, 6)]
cifar$test$y <- factor(if_else(cifar$test$y == 5, "Dog", "Frog"))
```

```
par(mfrow=c(1,2))
plot.new(); plot.window(xlim=c(0,1), ylim=c(0,1), asp=1)
rasterImage(cifar$train$x[1, , , ], 0, 0, 1, 1)
text(x=0.5, y=0.1, label=cifar$train$y[1], cex=2, col="white")
plot.new(); plot.window(xlim=c(0,1), ylim=c(0,1), asp=1)
rasterImage(cifar$train$x[7, , , ], 0, 0, 1, 1)
text(x=0.5, y=0.1, label=cifar$train$y[7], cex=2, col="white")
```

```
#dir.create("Train")
#dir.create("Test")
#for(lab in levels(cifar$train$y)){
#  dir.create(paste0("Train/", lab))
#  dir.create(paste0("Test/", lab))
#}
#for(i in 1:dim(cifar$train$x)[1]){
#  png::writePNG(cifar$train$x[i, , , ],
#                paste0("Train/", cifar$train$y[i], "/img", i, ".png"))
#}
#for(i in 1:dim(cifar$test$x)[1]){
#  png::writePNG(cifar$test$x[i, , , ],
#                paste0("Test/", cifar$test$y[i], "/img", i, ".png"))
#}
#rm(cifar)
```

```
training_image_gen <- image_data_generator(
  rescale = 1/255,
  width_shift_range = 0.2,
  height_shift_range = 0.2,
  rotation_range = 20,
  horizontal_flip = TRUE,
  validation_split=0.2
)

training_image_flow <- flow_images_from_directory(
  directory = "Train/",
  generator = training_image_gen,
  subset = "training",
  class_mode = "binary",
  batch_size = 20,
  target_size = c(32, 32),
  # Randomizing inputs is important so that the NN doesn't get stuck in a rut.
  # It's the default, but I've put it in explicitly.
  shuffle = TRUE
)
# A separate directory and generator could be used with a fixed validation set.
validation_image_flow <- flow_images_from_directory(
  directory = "Test/",
  generator = training_image_gen,
  subset = "validation",
  class_mode = "binary",
  batch_size = 20,
  target_size = c(32, 32),
  shuffle = FALSE
```

```
)
```

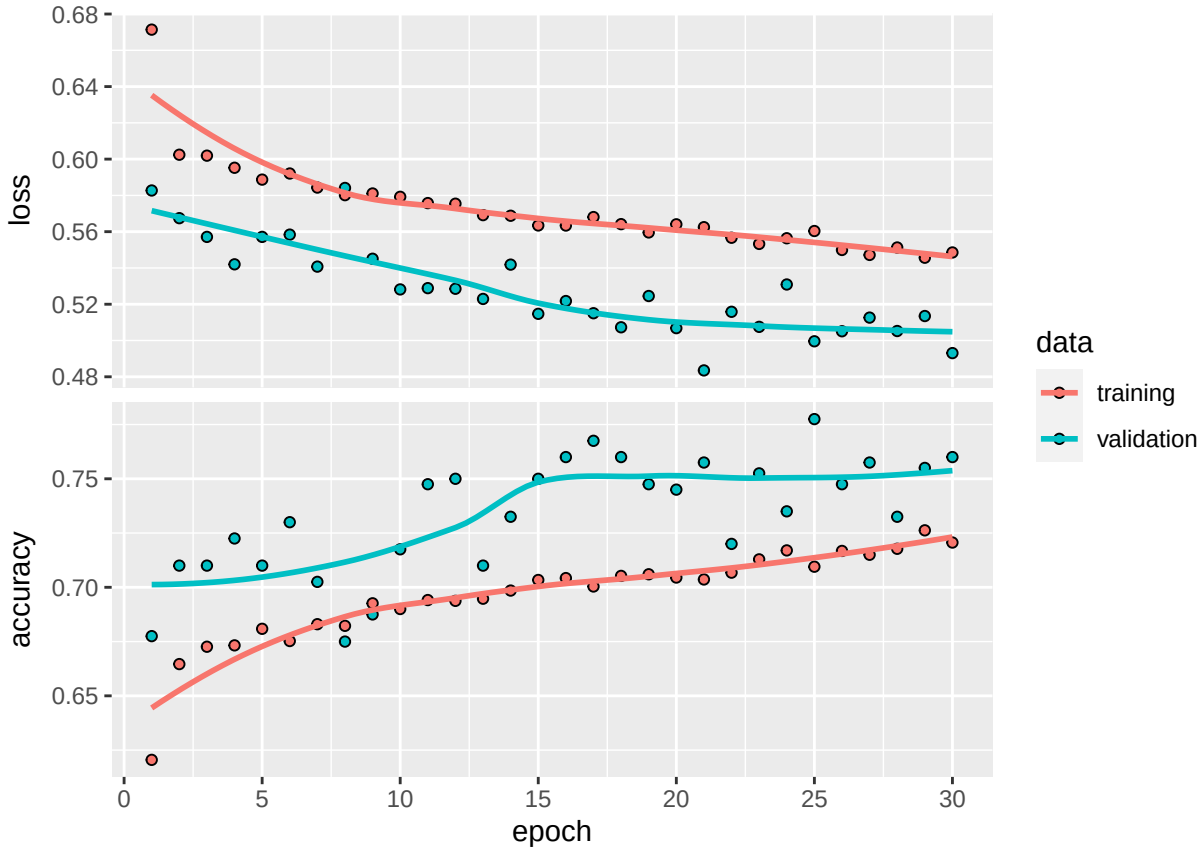
2

Train a regular (non-convolutional) network on the image data for at least 30 epochs (you can go farther if you want to–how high does validation accuracy go if you've got a lot of time?)

#3 Plot the validation accuracy, and use the evaluate command to evaluate the accuracy of the non-convolutional model on the test set. Note that you can use an "image flow" input to this normal neural network as long as the first layer is layer_flatten(input_shape=c(...)) where the dots are replaced by the geometry of the images.

```
k_clear_session()


dfnn <- keras_model_sequential() %>%
  # Note that layer_dense expects a "flat" vector, not an image, so we add
  # layer_flatten first.
  layer_flatten() %>%
  layer_dense(units=256, activation="relu", input_shape=c(32, 32, 3)) %>%
  layer_dropout(rate=0.2) %>%
  layer_dense(units=1, activation="sigmoid") %>%
  compile(
    loss = "binary_crossentropy",
    optimizer = "adam",
    metrics = "accuracy")


history <- dfnn %>% fit(
  x=training_image_flow,
  validation_data=validation_image_flow,
  epochs=30
)

plot(history)
```

```
## `geom_smooth()` using formula 'y ~ x'
```




```
evaluate(dfnn, training_image_flow, validation_image_flow)
```

```
##      loss  accuracy
## 0.5262996 0.7358750
```

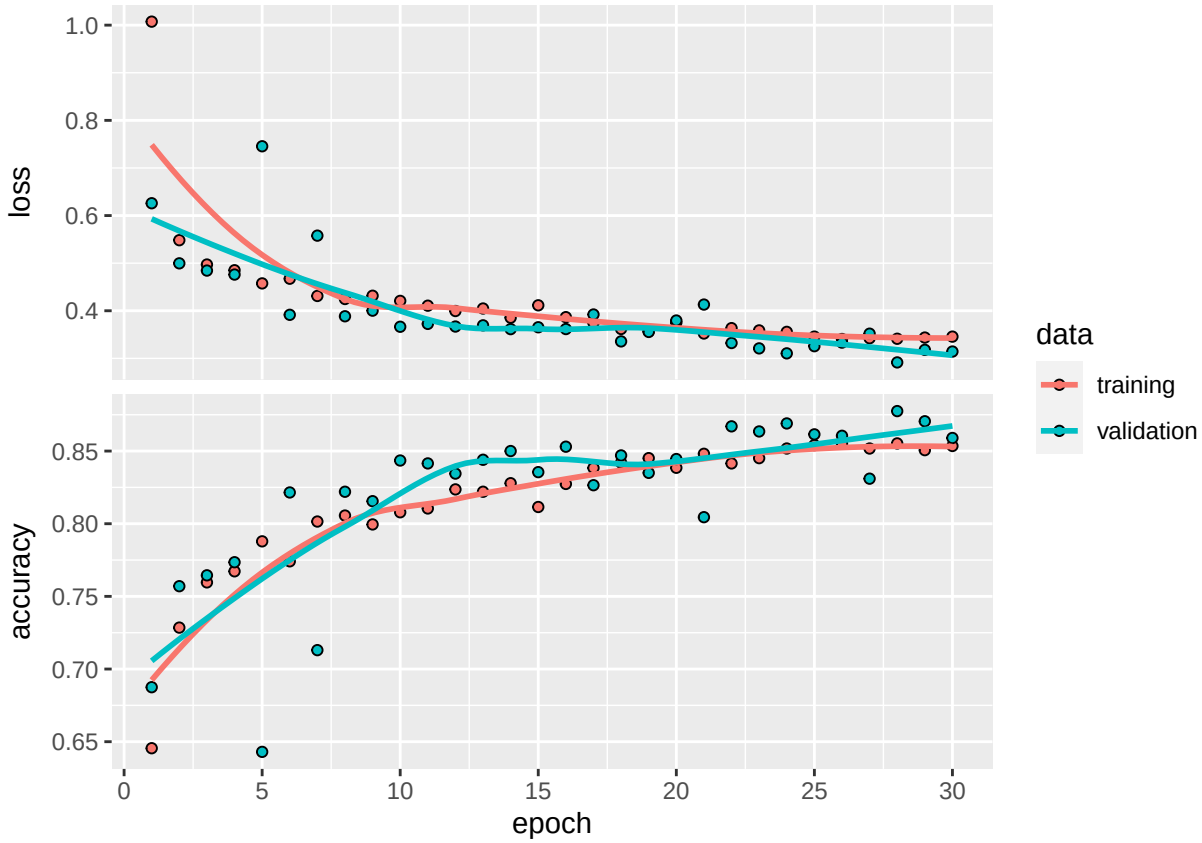
4 and 5

Train a convolutional neural network on the same data set for at least 30 epochs (again, more if you want). Plot validation accuracy and evaluate accuracy on the test set for this model.

```
k_clear_session()

training_image_gen <- image_data_generator(
  width_shift_range = 0.2,
  height_shift_range = 0.2,
  # I really want to try to train on the serif/sans details, so the next two
  # ensure that letters are in a variety of orientations.
  rotation_range = 45,
  horizontal_flip = TRUE,
  validation_split=0.2
)

training_image_flow <- flow_images_from_directory(
  directory = "Train/",
  generator = training_image_gen,
  subset = "training",
  class_mode = "binary",
  batch_size = 20,
  target_size = c(32, 32),
  # Randomizing inputs is important so that the NN doesn't get stuck in a rut.
  # It's the default, but I've put it in explicitly.
  shuffle = TRUE
)
# A separate directory and generator could be used with a fixed validation set.
validation_image_flow <- flow_images_from_directory(
  directory = "Train/",
  generator = training_image_gen,
  subset = "validation",
  class_mode = "binary",
  batch_size = 20,
  target_size = c(32, 32),
  shuffle = FALSE
)



fit.conv <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3,3), activation = 'relu',
                input_shape = c(32,32,3)) %>%
  #layer_conv_2d(filters = 32, kernel_size = c(3,3), activation = 'relu') %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_dropout(rate = 0.25) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = 'relu') %>%
  #layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = 'relu') %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 128, kernel_size = c(3,3), activation = 'relu') %>%
  #layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = 'relu') %>%
  layer_dropout(rate = 0.25) %>%
```

```
  layer_flatten() %>%
  layer_dense(units = 128, activation = 'relu') %>%
  #layer_dropout(rate = 0.25) %>%
  # Because we have a binary predictor, we only need one output unit and need
  # the `sigmoid` activation function. That's important!
  layer_dense(units = 1, activation = 'sigmoid') %>%
  compile(
    loss = "binary_crossentropy",
    # Here's a run-down of optimizers:
    # https://towardsdatascience.com/7-tips-to-choose-the-best-optimizer-47bb9c1219e
    optimizer = "adam",
    metrics = "accuracy"
  )


history2 <- fit.conv %>% fit(
  x = training_image_flow,
  epochs = 30,
  steps_per_epoch = training_image_flow$n/training_image_flow$batch_size,
  validation_data = validation_image_flow,
  validation_steps = validation_image_flow$n/validation_image_flow$batch_size
)


plot(history2)
```

```
## `geom_smooth()` using formula 'y ~ x'
```




```
evaluate(fit.conv, training_image_flow, validation_image_flow)
```

```
##      loss  accuracy
## 0.3144907 0.8605000
```

6

Write a paragraph, along with a table of results, that compares how well each of these models did in accurately predicting dog or frog. The table should have columns for accuracy, number of epochs and time it took to run on your computer. (Time can be can just be added up by hand from the on-screen output of the fit command.)

```
Title <- c("Accuracy", "Epochs", "Seconds", "Sec/Epoch")
Non_Con <- c(74.20, 30, 257, round(257/30))
Convolutional <- c(86.12, 30, 363, round(363/30))


results <- data.frame(Title, Non_Con, Convolutional)
kable(results)
```

Title	Non_Con	Convolutional
Accuracy	74.2	86.12
Epochs	30.0	30.00
Seconds	257.0	363.00
Sec/Epoch	9.0	12.00

The non-convolutional network was quicker to process (by 1.5 minutes per 30 epochs) but had a 12% less accuracy metric compared to the convolutional network.

For an extra 90-seconds I would certainly select the convolutional network as it had a 12% higher accuracy result.