STAT 471: Homework 5

Nico Melton

Due: December 8, 2021 at 11:59pm

Contents

Instructions **1**

Setup . . . 1

Collaboration . . . 2

Writeup . . . 2

Programming . . . 2

Grading . . . 2

Submission . . . 2

Fashion MNIST Data **3**

1 Data exploration **4**

2 Model training **6**

2.1 Multi-class logistic regression . . . 6

2.2 Fully connected neural network . . . 7

2.3 Convolutional neural network . . . 10

3 Evaluation **13**

Instructions

Setup

Pull the latest version of this assignment from Github and set your working directory to `stat-471-fall-2021/homework/homework-5`. Consult the getting started guide if you need to brush up on `R` or `Git`.

Collaboration

The collaboration policy is as stated on the Syllabus:

> "Students are permitted to work together on homework assignments, but solutions must be written up and submitted individually. Students must disclose any sources of assistance they received; furthermore, they are prohibited from verbatim copying from any source and from consulting solutions to problems that may be available online and/or from past iterations of the course."

In accordance with this policy,

Please list anyone you discussed this homework with:

Please list what external references you consulted (e.g. articles, books, or websites):

Writeup

Use this document as a starting point for your writeup, adding your solutions after "**Solution**". Add your R code using code chunks and add your text answers using **bold text**. Consult the preparing reports guide for guidance on compilation, creation of figures and tables, and presentation quality.

Programming

The `tidyverse` paradigm for data wrangling, manipulation, and visualization is strongly encouraged, but points will not be deducted for using base `R`.

We'll need to use the following R packages:

```
library(keras)          # to train neural networks
library(kableExtra)     # to print tables
library(cowplot)        # to print side-by-side plots
library(tidyverse)      # tidyverse
```

We'll also need the deep learning helper functions written for STAT 471:

```
source("../../functions/deep_learning_helpers.R")
```

Grading

The point value for each problem sub-part is indicated. Additionally, the presentation quality of the solution for each problem (as exemplified by the guidelines in Section 3 of the preparing reports guide will be evaluated on a per-problem basis (e.g. in this homework, there are three problems).

Submission

Compile your writeup to PDF and submit to Gradescope.

Fashion MNIST Data

In this homework, we will analyze the Fashion MNIST data, which is like MNIST but with clothing items rather than handwritten digits. There are ten classes, as listed in Table 1.

Table 1: The ten classes in the Fashion MNIST data.

Index	Name
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

The code provided below loads the data, and prepares it for modeling with `keras`.

```
# load the data
fashion_mnist <- dataset_fashion_mnist()
```

```
## Loaded Tensorflow version 2.7.0
```

```
# extract information about the images
num_classes = nrow(class_names)                    # number of image classes
num_train_images = dim(fashion_mnist$train$x)[1]   # number of training images
num_test_images = dim(fashion_mnist$test$x)[1]     # number of test images
img_rows <- dim(fashion_mnist$train$x)[2]          # rows per image
img_cols <- dim(fashion_mnist$train$x)[3]          # columns per image
num_pixels = img_rows*img_cols                     # pixels per image
max_intensity = 255                                # max pixel intensity

# normalize and reshape the images
x_train <- array_reshape(fashion_mnist$train$x/max_intensity,
                         c(num_train_images, img_rows, img_cols, 1))
x_test <- array_reshape(fashion_mnist$test$x/max_intensity,
                        c(num_test_images, img_rows, img_cols, 1))

# extract the responses from the training and test data
g_train <- fashion_mnist$train$y
g_test <- fashion_mnist$test$y

# recode response labels using "one-hot" representation
y_train <- to_categorical(g_train, num_classes)
y_test <- to_categorical(g_test, num_classes)
```

1 Data exploration

i. How many observations in each class are there in the training data? (Kable output optional.)

Solution.

```
# number of obs of each class in tibble
class_count = tribble(
  ~class, ~number,
  "T-shirt/top", sum(g_train == 0),
  "Trouser", sum(g_train == 1),
  "Pullover", sum(g_train == 2),
  "Dress", sum(g_train == 3),
  "Coat", sum(g_train == 4),
  "Sandal", sum(g_train == 5),
  "Shirt", sum(g_train == 6),
  "Sneaker", sum(g_train == 7),
  "Bag", sum(g_train == 8),
  "Ankle boot", sum(g_train == 9),
)
# display with kable
class_count %>% kable(format = "latex", row.names = NA,
                      booktabs = TRUE,
                      digits = 2,
                      col.names = c("Class Name",
                                    "# of Observations"),
                      caption = "The number of observations in each class in the Fashion MNIST data.")
  kable_styling(position = "center") %>%
  kable_styling(latex_options = "HOLD_position")
```

Table 2: The number of observations in each class in the Fashion MNIST data.

Class Name	# of Observations
T-shirt/top	6000
Trouser	6000
Pullover	6000
Dress	6000
Coat	6000
Sandal	6000
Shirt	6000
Sneaker	6000
Bag	6000
Ankle boot	6000

See Table 2. There are 6000 observations in each class in teh training data.

ii. Plot the first six training images in a 2×3 grid, each image titled with its class name from the second column of Table 1.

Solution

```
# plot six images
p1 = plot_grayscale(x_train[1,,,], class_names %>% filter(class == g_train[1]) %>% pull(name))
p2 = plot_grayscale(x_train[2,,,], class_names %>% filter(class == g_train[2]) %>% pull(name))
p3 = plot_grayscale(x_train[3,,,], class_names %>% filter(class == g_train[3]) %>% pull(name))
p4 = plot_grayscale(x_train[4,,,], class_names %>% filter(class == g_train[4]) %>% pull(name))
p5 = plot_grayscale(x_train[5,,,], class_names %>% filter(class == g_train[5]) %>% pull(name))
p6 = plot_grayscale(x_train[6,,,], class_names %>% filter(class == g_train[6]) %>% pull(name))
plot_grid(p1, p2, p3, p4, p5, p6, nrow = 2)
```

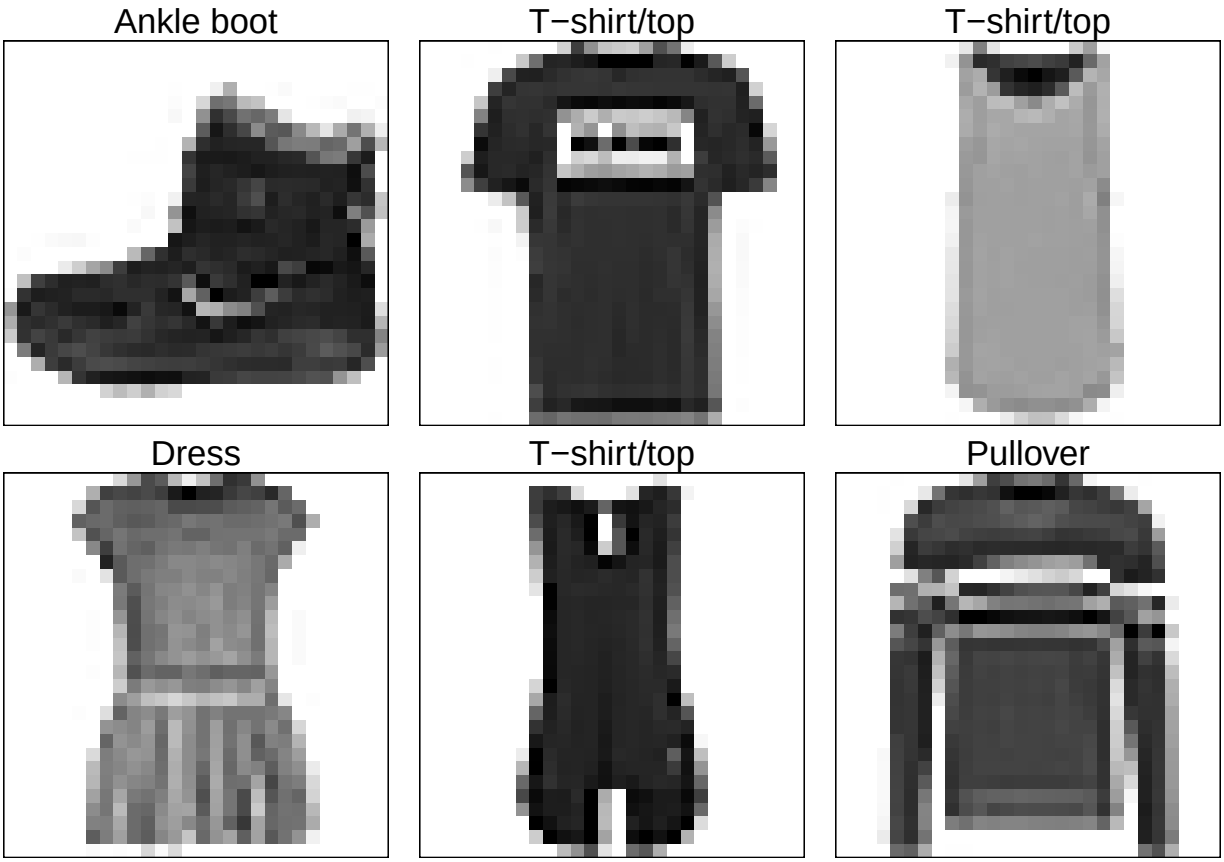

Figure 1: First six training images with class name.

See Figure 1.

iii. Comment on the extent to which you (a human) would have been able to successfully classify the observations plotted in part ii. Would you have had any trouble? If so, with which observations?

Solution

I would probably not be able to successfully classify the third (top right) and fifth (bottom middle) images because of how blurry they are and they look like dresses. I can imagine that humans would misclassify many of these images.

iv. What is the human performance on this classification task? You can find it at the Fashion MNIST webpage linked above by searching for "human performance."

Solution.

The human performance on this classification is 0.835.

2 Model training

2.1 Multi-class logistic regression

i. Define a `keras_model_sequential` object called `model_lr` for multi-class logistic regression, and compile it using the `categorical_crossentropy` loss, the `adam` optimizer, and the `accuracy` metric.

Solution.

```
# define `keras_model_sequential` object
model_lr = keras_model_sequential() %>%
  layer_flatten(input_shape =
                  c(img_rows, img_cols, 1)) %>%
  layer_dense(units = num_classes,
              activation = "softmax")
# compile model
model_lr %>%
  compile(loss = "categorical_crossentropy",
          optimizer = optimizer_adam(),
          metrics = c("accuracy"))
```

ii. Print the summary of the model. How many total parameters are there? How does this number follow from the architecture of this simple neural network?

Solution.

```
# print summary of model
summary(model_lr)
```

```
## Model: "sequential"
## _________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## =========================================================================
##  flatten (Flatten)                  (None, 784)                     0
##
##  dense (Dense)                      (None, 10)                      7850
##
## =========================================================================
## Total params: 7,850
## Trainable params: 7,850
## Non-trainable params: 0
## _________________________________________________________________________
```

There are 7,850 total parameters. This comes from the 784 inputs in the flattened data (28 rows times 28 columns), plus 1 for the constant, times 10 for the number of classes in the data: $(784 + 1) \times 10 = 7850$. **The 784 inputs are the pixels in each 28 by 28 image:** $28 \times 28 = 784$

iii. Train the model for 10 epochs, using a batch size of 128, and a validation split of 20%. Save the model to `model_lr.h5` and its history to `model_lr_hist.RDS`, and then set this code chunk to `eval = FALSE` to avoid recomputation. How many total stochastic gradient steps were taken while training this model, and how did you arrive at this number? Based on the output printed during training, roughly how many milliseconds did each stochastic gradient step take?

Solution

```
# train model
model_lr %>%
  fit(x_train,
      y_train,
      epochs = 10,
      batch_size = 128,
      validation_split = 0.2)
# save model
# save_model_hdf5(model_lr, "model_lr.h5")
# save model history
# saveRDS(model_lr$history$history, "model_lr_hist.RDS")
```

There were 3750 stochastic gradient steps taken while training this model. I arrived at this number by first finding how many observations are used to train by taking the total 60,000 observations and only taking those that aren't assigned to the validation set (80% are not for validation): $60,000 \times 0.8 = 48,000$**. Then I divided this number by the batch size of 128 to get the number of stochastic gradient steps per epoch:** $48,000 \div 128 = 375$**. Finally, since there are 10 epochs, the total number of stochastic gradient steps is:** $375 \times 10 = 3750$**. Each stochastic gradient step took about 2ms.**

iv. Load the model and its history from the files saved in part iii. Create a plot of the training history. Based on the shape of the validation loss curve, has any overfitting occurred during the first 10 epochs?

Solution.

```
# load model
model_lr = load_model_hdf5("model_lr.h5")
# load model history
model_lr_hist = readRDS("model_lr_hist.RDS")
# plot training history
plot_model_history(model_lr_hist)
```

See Figure 2. Based on this plot, overfitting has not occurred during the first 10 epochs because the loss curve is always decreasing and the accuracy curve is always increasing.

2.2 Fully connected neural network

i. Define a `keras_model_sequential` object called `model_nn` for a fully connected neural network with three hidden layers with 256, 128, and 64 units, `relu` activations, and dropout proportions 0.4, 0.3, and 0.2, respectively. Compile it using the `categorical_crossentropy` loss, the `rmsprop` optimizer, and the `accuracy` metric.

Solution.

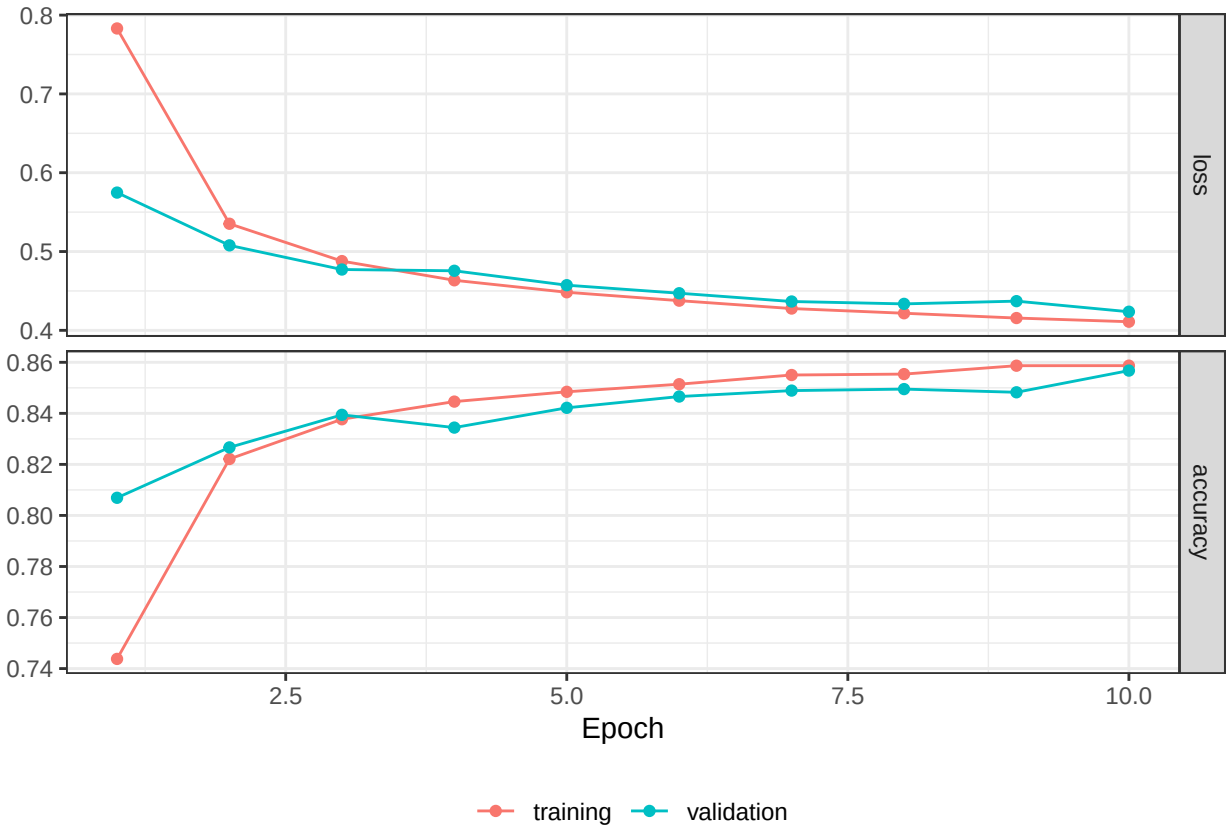

Figure 2: Training history of multi-class logistic model.

```
# define `keras_model_sequential` object
model_nn = keras_model_sequential() %>%
  layer_flatten(input_shape = c(img_rows, img_cols, 1)) %>%
  layer_dense(units = 256, activation = "relu") %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 128, activation = "relu") %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 64, activation = "relu") %>%
  layer_dropout(rate = 0.2) %>%
  layer_dense(units = num_classes,
              activation = "softmax")

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

ii. Print the summary of the model. How many total parameters are there? How many parameters correspond to the connections between the second and third hidden layers? How does this number follow from the architecture of the neural network?

Solution.

```
# print summary of model
summary(model_nn)
```

```
## Model: "sequential_1"
## _________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## =========================================================================
##  flatten_1 (Flatten)                (None, 784)                     0
##
##  dense_4 (Dense)                    (None, 256)                     200960
##
##  dropout_2 (Dropout)                (None, 256)                     0
##
##  dense_3 (Dense)                    (None, 128)                     32896
##
##  dropout_1 (Dropout)                (None, 128)                     0
##
##  dense_2 (Dense)                    (None, 64)                      8256
##
##  dropout (Dropout)                  (None, 64)                      0
##
##  dense_1 (Dense)                    (None, 10)                      650
##
## =========================================================================
## Total params: 242,762
## Trainable params: 242,762
## Non-trainable params: 0
## _________________________________________________________________________
```

There are 242,762 total parameters. 8,256 parameters correspond to the connections between the second and third hidden layers. This comes from the 128 units in the second hidden layer, plus 1 for the constant, times the 64 units in the third hidden layer: $(128 + 1) \times 64 = 8,256$

iii. Train the model using 15 epochs, a batch size of 128, and a validation split of 0.2. Save the model to `model_nn.h5` and its history to `model_nn_hist.RDS`, and then set this code chunk to `eval = FALSE` to avoid recomputation. Based on the output printed during training, roughly how many milliseconds did each stochastic gradient step take?

Solution.

```
# train model
model_nn %>%
  fit(x_train,
      y_train,
      epochs = 15,
      batch_size = 128,
      validation_split = 0.2)
# save model
# save_model_hdf5(model_nn, "model_nn.h5")
# save model history
# saveRDS(model_nn$history$history, "model_nn_hist.RDS")
```

Each stochastic gradient step took about 4ms.

iv. Load the model and its history from the files saved in part iii. Create a plot of the training history.

Solution.

```
# load model
model_nn = load_model_hdf5("model_nn.h5")
# load model history
model_nn_hist = readRDS("model_nn_hist.RDS")
# plot training history
plot_model_history(model_nn_hist)
```

See Figure 3.

2.3 Convolutional neural network

i. Define a `keras_model_sequential` object called `model_cnn` for a convolutional neural network with a convolutional layer with 32 3×3 filters, followed by a convolutional layer with 64 3×3 filters, followed by a max-pooling step with 2×2 pool size with 25% dropout, followed by a fully-connected layer with 128 units and 50% dropout, followed by a softmax output layer. All layers except the output layer should have `relu` activations. Compile the model using the `categorical_crossentropy` loss, the `adadelta` optimizer, and the `accuracy` metric.

Solution.

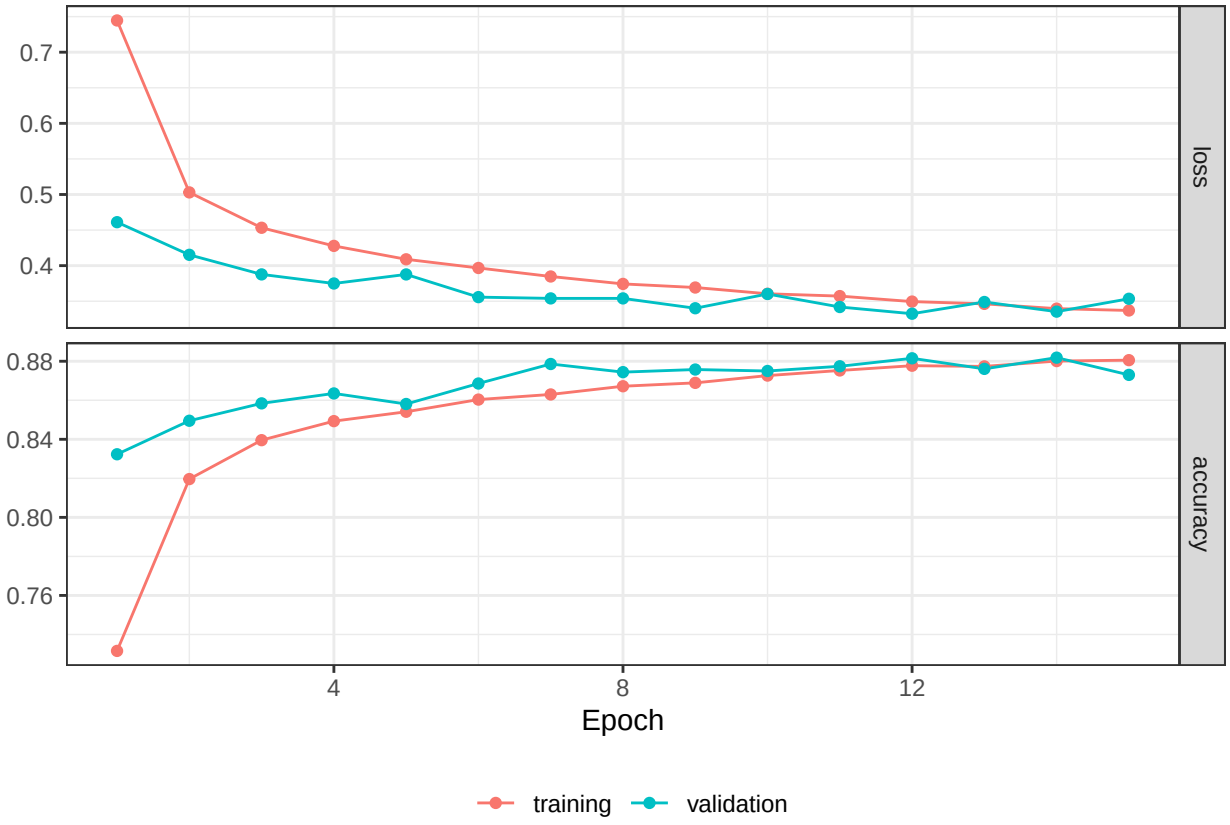

Figure 3: Training history of fully connected neural network.

```r
# define `keras_model_sequential` object
model_cnn = keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3,3),
                activation = "relu",
                input_shape = c(img_rows, img_cols, 1)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3,3),
                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 = 0.5) %>%
  layer_dense(units = num_classes, activation = "softmax")

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

ii. Print the summary of the model. How many total parameters are there? How many parameters correspond to the connections between the first and second convolutional layers? How does this number follow from the architecture of the neural network?

Solution.

```r
# print summary of model
summary(model_cnn)
```

```
## Model: "sequential_2"
## _________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## =========================================================================
##  conv2d_1 (Conv2D)                  (None, 26, 26, 32)              320
##
##  conv2d (Conv2D)                    (None, 24, 24, 64)              18496
##
##  max_pooling2d (MaxPooling2D)       (None, 12, 12, 64)              0
##
##  dropout_4 (Dropout)                (None, 12, 12, 64)              0
##
##  flatten_2 (Flatten)                (None, 9216)                    0
##
##  dense_6 (Dense)                    (None, 128)                     1179776
##
##  dropout_3 (Dropout)                (None, 128)                     0
##
##  dense_5 (Dense)                    (None, 10)                      1290
##
## =========================================================================
## Total params: 1,199,882
## Trainable params: 1,199,882
```

```
## Non-trainable params: 0
## _________________________________________________________________________
```

There are 1,199,882 total parameters. 18,496 parameters correspond to the connections between the first and second convolutional layers. This number comes from multiplying the lengths of the sides of the filter (3x3) by the number of layers in the previous step (32) and then adding 1 for bias and multiplying that by the depth of the filter in the current layer (64): $((3 \times 3 \times 32) + 1) \times 64 = 18,496$

iii. Train the model using 8 epochs, a batch size of 128, and a validation split of 0.2. Save the model to `model_cnn.h5` and its history to `model_cnn_hist.RDS`, and then set this code chunk to `eval = FALSE` to avoid recomputation. Based on the output printed during training, roughly how many milliseconds did each stochastic gradient step take?

Solution.

```
# train model
model_cnn %>%
  fit(x_train,
      y_train,
      epochs = 8,
      batch_size = 128,
      validation_split = 0.2)
# save model
# save_model_hdf5(model_cnn, "model_cnn.h5")
# save model history
# saveRDS(model_cnn$history$history, "model_cnn_hist.RDS")
```

Each stochastic gradient step took about 80ms.

iv. Load the model and its history from the files saved in part iii. Create a plot of the training history.

Solution.

```
# load model
model_cnn = load_model_hdf5("model_cnn.h5")
# load model history
model_cnn_hist = readRDS("model_cnn_hist.RDS")
# plot training history
plot_model_history(model_cnn_hist)
```

See Figure 4.

3 Evaluation

i. Evaluate the test accuracy for each of the three trained neural network models. Output this information in a table, along with the number of layers, number of parameters, and milliseconds per stochastic gradient descent step. Also include a row in the table for human performance. Compare and contrast the three neural networks and human performance based on this table.

Solution.

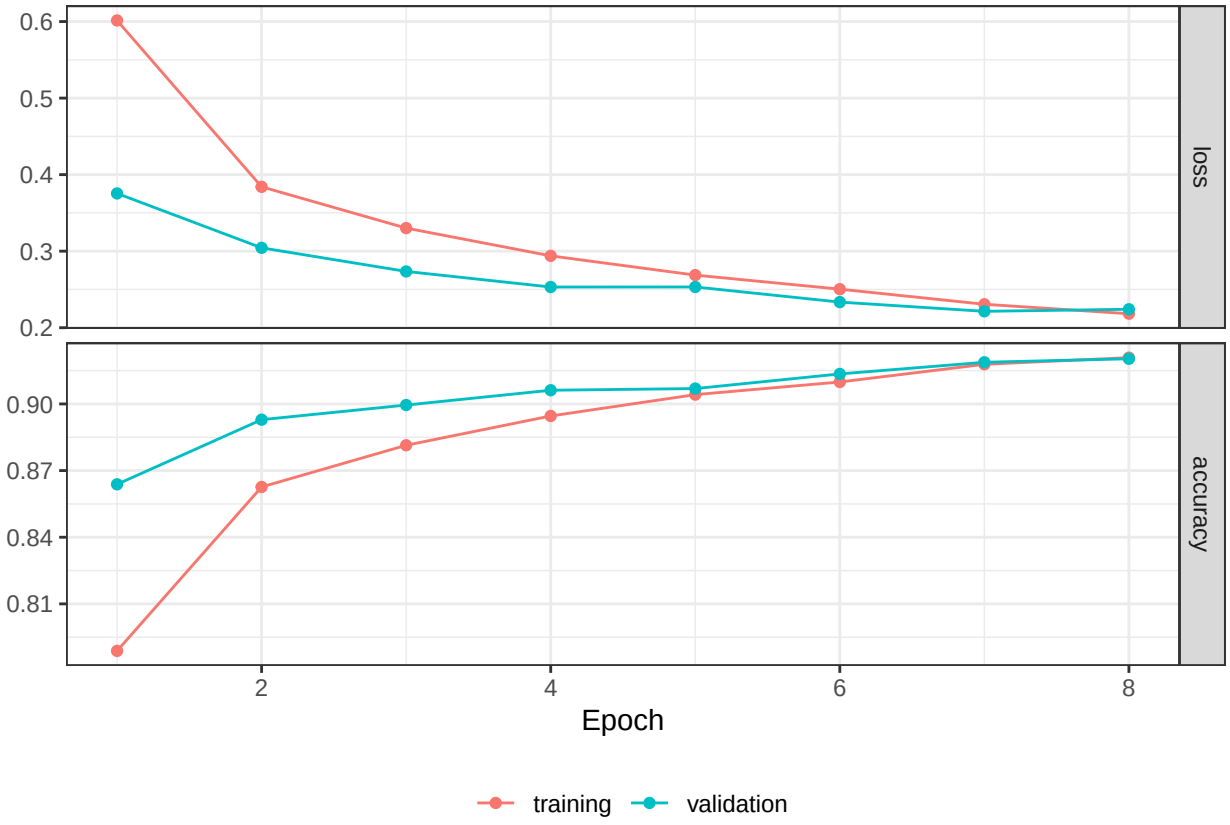

Figure 4: Training history of convolutional neural network.

ii. Plot confusion matrices for each of the three methods. For each method, what class gets misclassified most frequently? What is most frequent wrong label for this class?

Solution.

iii. Consider CNN's most frequently misclassified class. What are the three most common incorrect classifications for this class? Extract one image representing each of these three type of misclassifications, and plot these side by side (titled with their predicted labels). Would you have gotten these right?

Solution.