STAT 471: Homework 5

Name(s)

Due: December 8, 2021 at 11:59pm

Contents

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`.

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

```
## Loaded Tensorflow version 2.5.0
```

```r
# 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.)

```r
g_train_tibble = g_train%>% as_tibble()
g_train_tibble%>%count(value)%>%rename(class=value, count=n)%>%
  kable(format = "latex", row.names = NA,
                        booktabs = TRUE,
                        digits = 2,
                        col.names = c("Class",
                                      "Count"),
                        caption = "The number of observations in each class in the training data.") %>%
  kable_styling(position = "center") %>%
  kable_styling(latex_options = "HOLD_position")
```

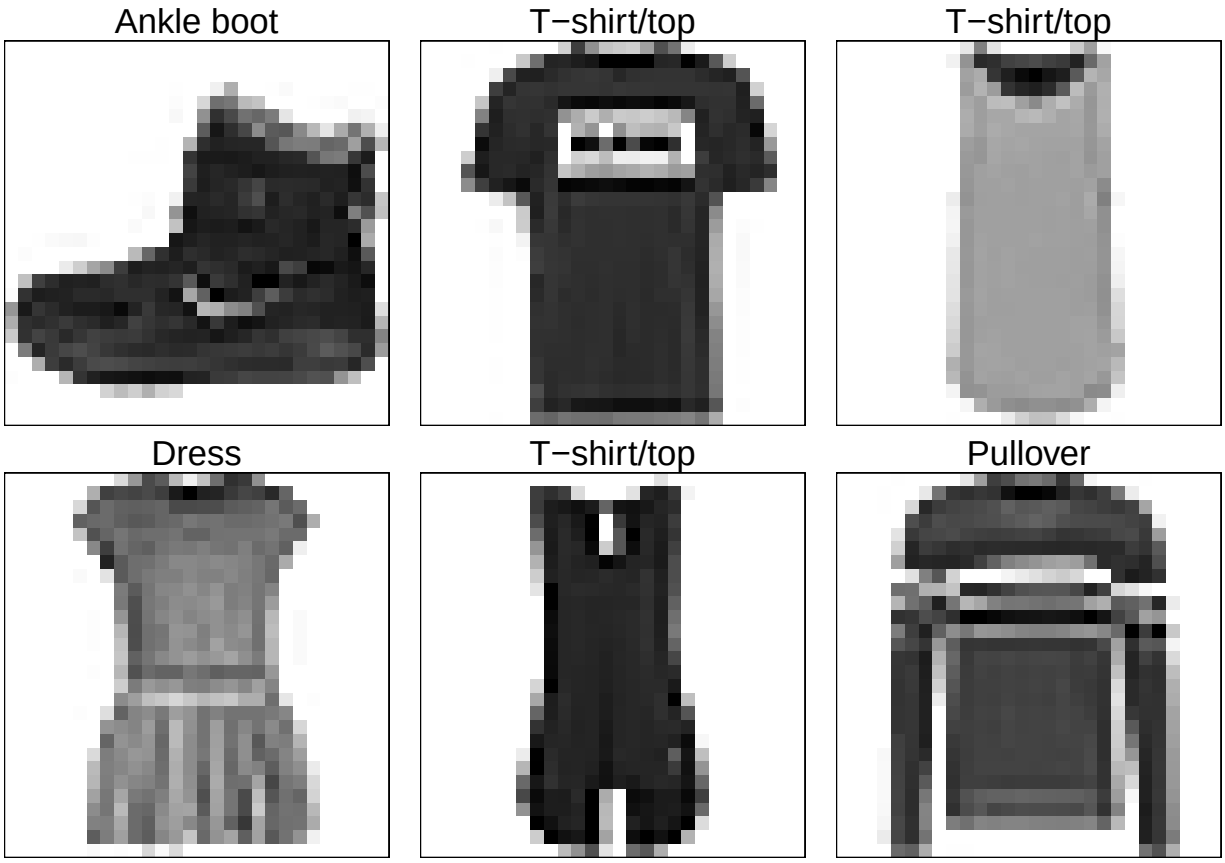
Table 2: The number of observations in each class in the training data.

Class	Count
0	6000
1	6000
2	6000
3	6000
4	6000
5	6000
6	6000
7	6000
8	6000
9	6000

There are 6000 observations in each class in the 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.

```
p1 = plot_grayscale(x_train[1,,,], g_train[1], class_names)
p2 = plot_grayscale(x_train[2,,,], g_train[2], class_names)
p3 = plot_grayscale(x_train[3,,,], g_train[3], class_names)
p4 = plot_grayscale(x_train[4,,,], g_train[4], class_names)
p5 = plot_grayscale(x_train[5,,,], g_train[5], class_names)
p6 = plot_grayscale(x_train[6,,,], g_train[6], class_names)
plot_grid(p1, p2, p3, p4, p5, p6, nrow = 2)
```



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?

As a human, I had trouble with classifying the third and fifth plots. My success rate is about 66%.

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."

The human performance (i.e., test accuracy rate) on this task 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.

```
model_lr = keras_model_sequential() %>%
  layer_flatten(input_shape =             # flatten during model-building
                  c(img_rows, img_cols, 1)) %>%
  layer_dense(units = num_classes,        # number of outputs
              activation = "softmax")     # type of activation function
```

```
model_lr %>%                                  # note: modifying model_lr in place
  compile(loss = "categorical_crossentropy",  # which loss to use
          optimizer = optimizer_adam(),     # how to optimize the loss
          metrics = c("accuracy"))            # how to evaluate the fit
```

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?

```
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 7850 parameters in this model. In a fully connected neural network, each of the 784 pixels of each image is connected with each of the 10 neurons in the output layer, which results in 7840 connections. There is also a bias term in the input layer that is connected with the 10 neurons in the output layer, which results in 10 additional connections. Thus, there are 7850 connections in total. Since each connection has a weight, there are 7850 parameters/weights.

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?

```
model_lr %>%
  fit(x_train,                  # supply training features
      y_train,                  # supply training responses
      epochs = 10,               # an epoch is a gradient step
      batch_size = 128,         # we will learn about batches in Lecture 2
      validation_split = 0.2)   # use 20% of the training data for validation

# save model
save_model_hdf5(model_lr, "model_lr.h5")

# save history
saveRDS(model_lr$history$history, "model_lr_hist.RDS")
```

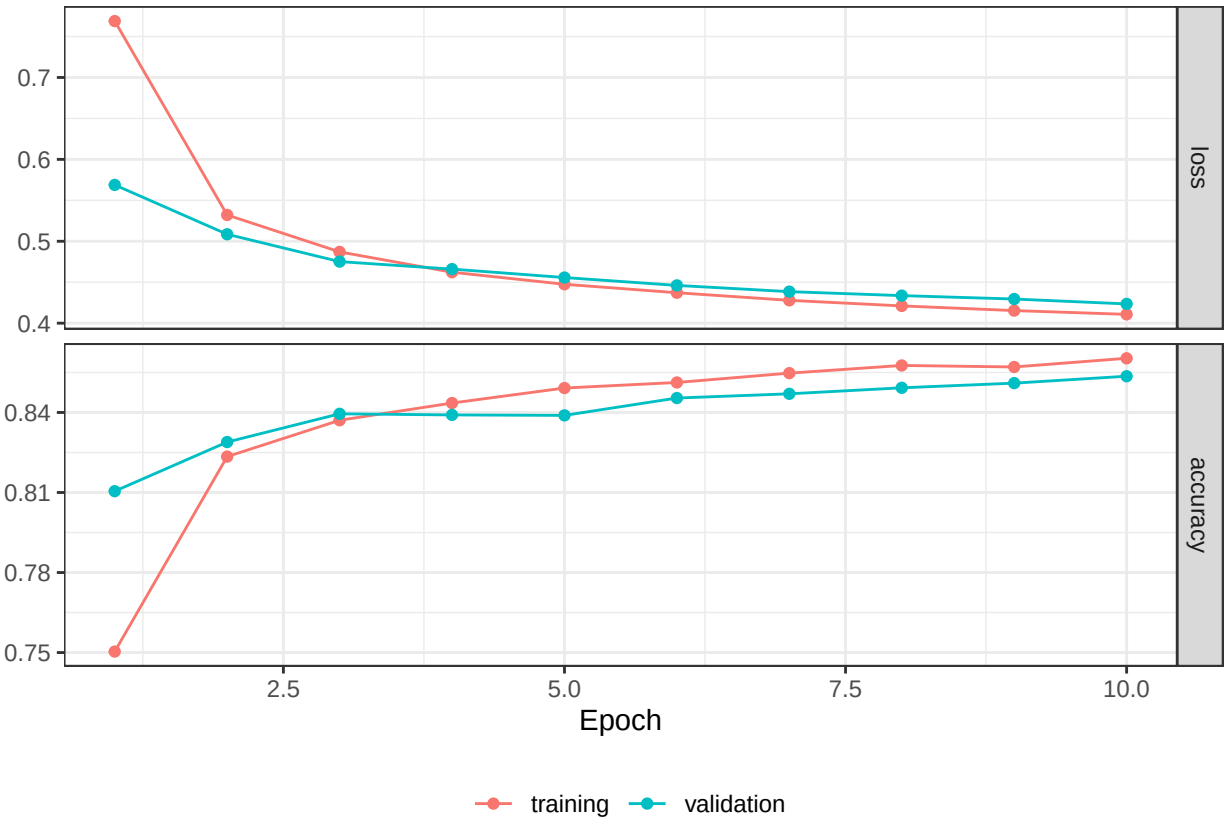
3750 total stochastic gradient steps were taken while training this model. There are 60000*0.8 = 48000 images used for training. Since the mini-batch size is 128, each gradient step is calculated based on 128 observations. Thus, 48000/128 = 375 steps can be made during each epoch. Since there are 10 epochs, 375*10=3750 total steps are taken.

Based on the output printed during training, each stochastic gradient step took roughly 5 milliseconds.

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?

```
# load model
model_lr = load_model_hdf5("model_lr.h5")

# load history
```

```
model_lr_hist = readRDS("model_lr_hist.RDS")

#plot the training history
plot_model_history(model_lr_hist)
```



Based on the shape of the validation loss curve, there is no overfitting occurred during the first 10 epochs. The validation loss does not start to increase by the tenth epoch. Though training loss is better than the validation loss by the tenth epoch, the former is not significantly better than the latter.

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.

```
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 = 10, activation = "softmax")
```

```
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?

```
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 8256 parameters that correspond to the connections between the second and third hidden layers. There are 128 neurons in the second hidden layer, and 64 neurons in the third hidden layer. Since the two layers are fully connected, counting the one bias term to the second layer, there are (128+1)*64=8256 connections. Since each connection has its own weight, there are 8256 weights/parameters.

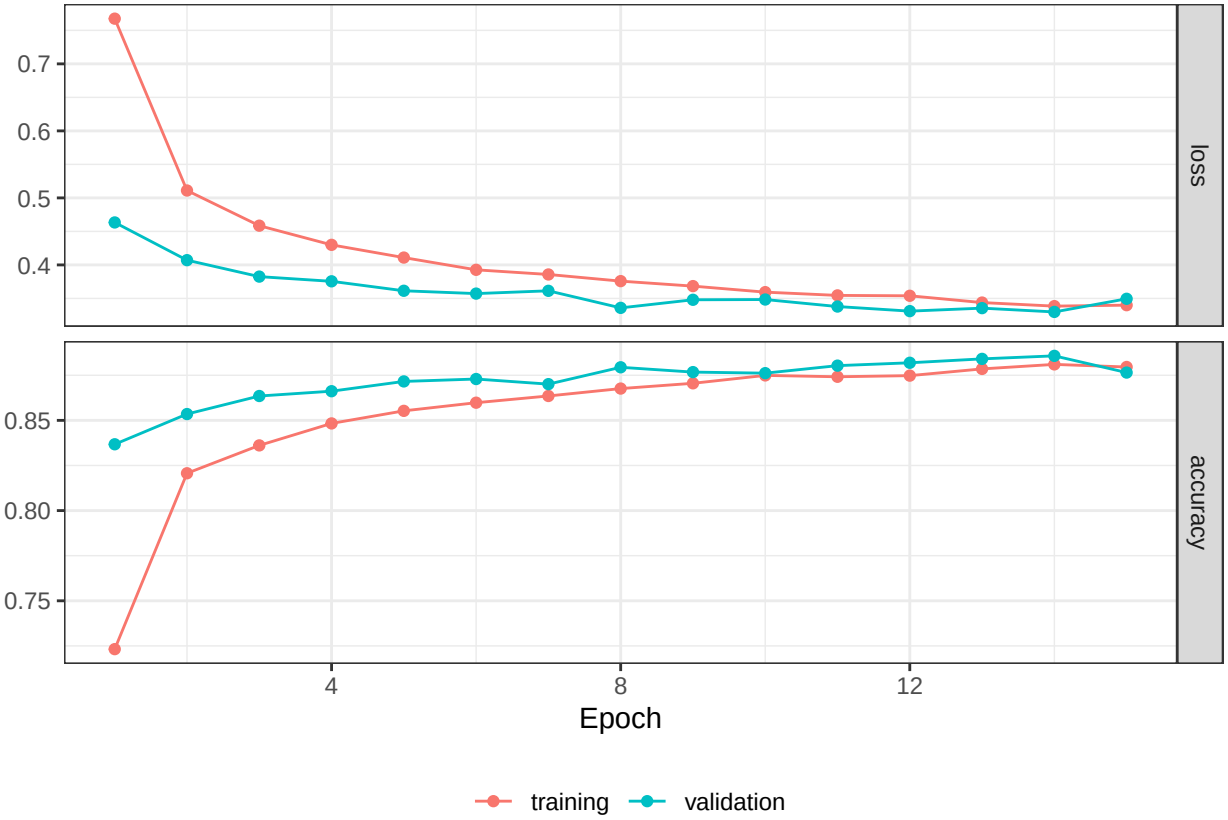
iii. Train the model using 15 epochs, a batch size of 128, and a validation split of 0.2. Save the model to `model_nn.h5` and its history to `model_nn_hist.RDS`, and then set this code chunk to `eval = FALSE` to avoid recomputation. Based on the output printed during training, roughly how many milliseconds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

```
history = model_nn %>%
  fit(x_train,                  # supply training features
      y_train,                  # supply training responses
      epochs = 15,              # an epoch is a gradient step
      batch_size = 128,         # we will learn about batches in Lecture 2
      validation_split = 0.2)   # use 20% of the training data for validation

# save model
save_model_hdf5(model_nn, "model_nn.h5")

# save history
saveRDS(model_nn$history$history, "model_nn_hist.RDS")
```

Based on the output printed during training, each stochastic gradient step took roughly 12 milliseconds.

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

```
# load model
model_nn = load_model_hdf5("model_nn.h5")

# load history
model_nn_hist = readRDS("model_nn_hist.RDS")

#plot the training history
plot_model_history(model_nn_hist)
```



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.

```
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')
```

```
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?

```
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. There are 18496 parameters that correspond to the connections between the first and second convolutional layers. The input to the second convolutional layer has 32 channels. The filter size of the second convolutional layer is 3 by 3. Thus, counting the one bias term, there are 3*3*32+1=289 parameters for each filter. Since there are 64 filters in second convolutional layer, there are 64*289=18496 parameters in total.

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?

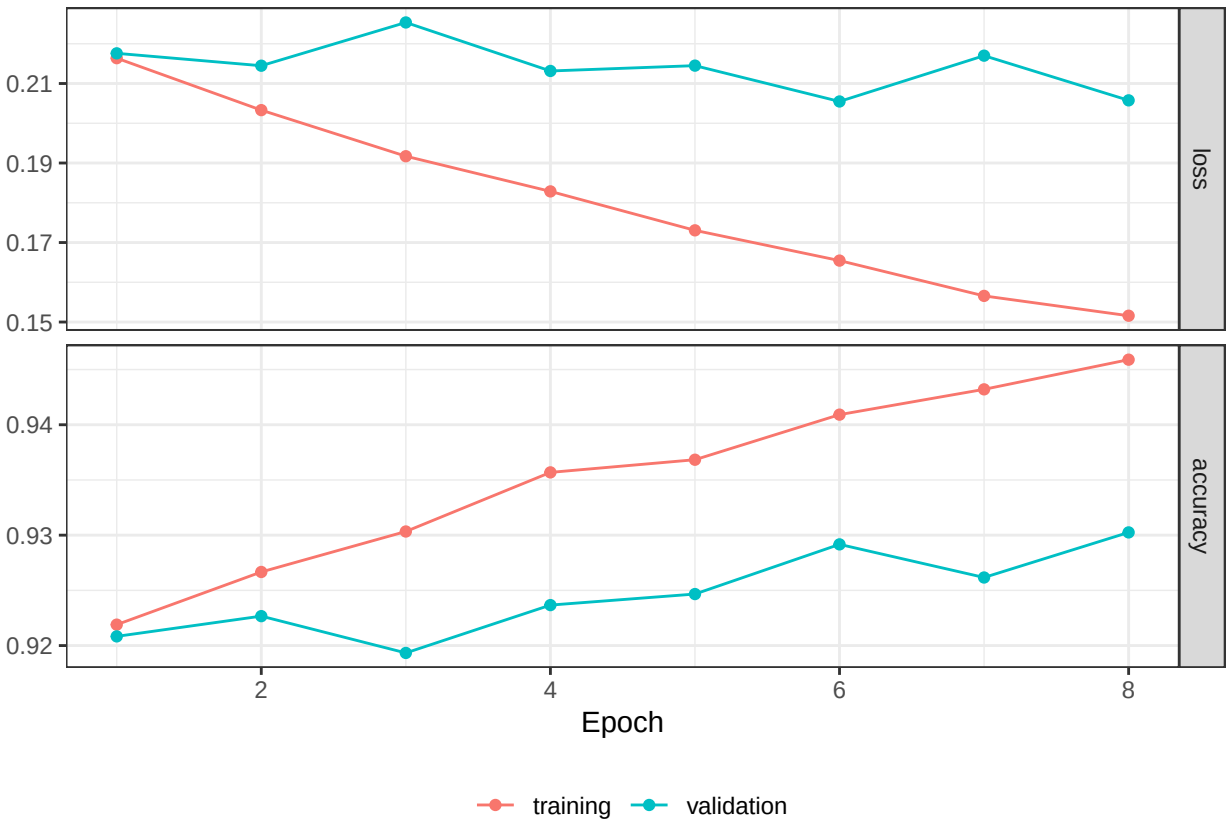
```
model_cnn %>%
  fit(x_train,                  # supply training features
      y_train,                  # supply training responses
      epochs = 8,               # an epoch cycles through all mini-batches
```

```
      batch_size = 128,          # mini-batch size
      validation_split = 0.2)    # use 20% of the training data for validation

# save model
save_model_hdf5(model_cnn, "model_cnn.h5")

# save history
saveRDS(model_cnn$history$history, "model_cnn_hist.RDS")
```

Based on the output printed during training, each stochastic gradient step took roughly 207 milliseconds.

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

```
# load model
model_cnn = load_model_hdf5("model_cnn.h5")

# load history
model_cnn_hist = readRDS("model_cnn_hist.RDS")

#plot the training history
plot_model_history(model_cnn_hist)
```



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.

```
acc_lr = evaluate(model_lr, x_test, y_test, verbose = FALSE)[2]
acc_nn = evaluate(model_nn, x_test, y_test, verbose = FALSE)[2]
acc_cnn = evaluate(model_cnn, x_test, y_test, verbose = FALSE)[2]
```

```
summary_tibble = tibble(model=c("Multi-class logistic regression",
                                "Fully connected neural network",
                                "Convolutional neural network",
                                "Human performance"),
                        acc=c(acc_lr, acc_nn, acc_cnn, 0.835),
                        num_layers=c(1,4,5,NA),
                        num_param=c(7850, 242762, 1199882, NA),
                        step_time=c(5,12,207,NA))
summary_tibble%>%
  kable(format = "latex", row.names = NA,
                        booktabs = TRUE,
                        digits = 3,
                        col.names = c("Model","Accuracy rate",
                                      "Number of layers",
                                      "Number of parameters",
                                      "Milliseconds per step"),
                        caption = "The summary table for comparing performance") %>%
  kable_styling(position = "center") %>%
  kable_styling(latex_options = "HOLD_position")
```

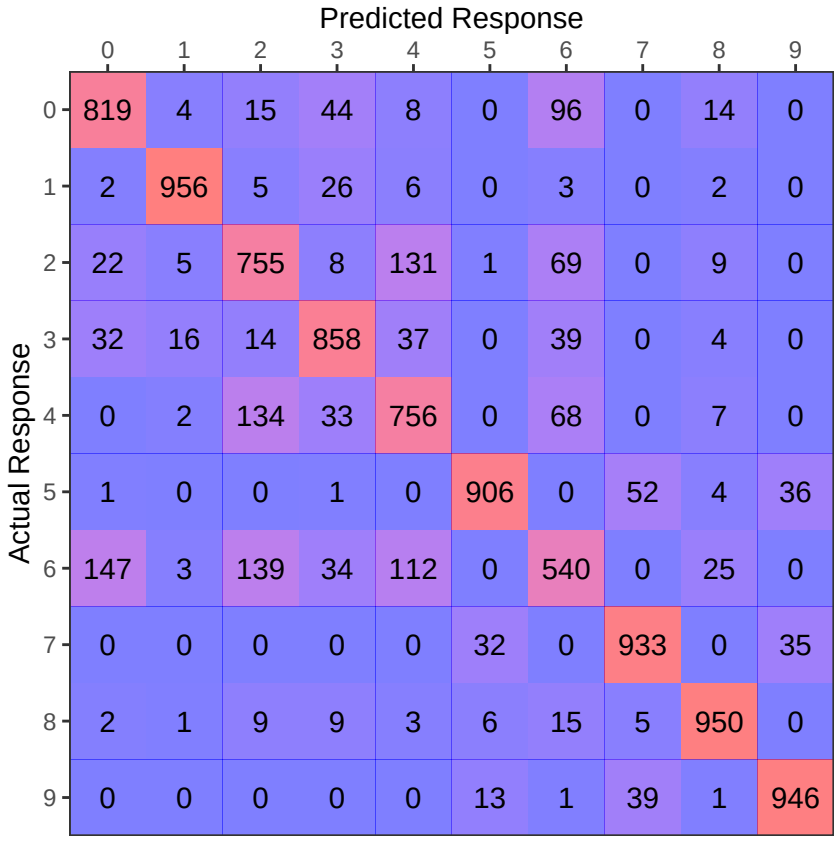
Table 3: The summary table for comparing performance

Model	Accuracy rate	Number of layers	Number of parameters	Milliseconds per step
Multi-class logistic regression	0.842	1	7850	5
Fully connected neural network	0.869	4	242762	12
Convolutional neural network	0.924	5	1199882	207
Human performance	0.835	NA	NA	NA

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?

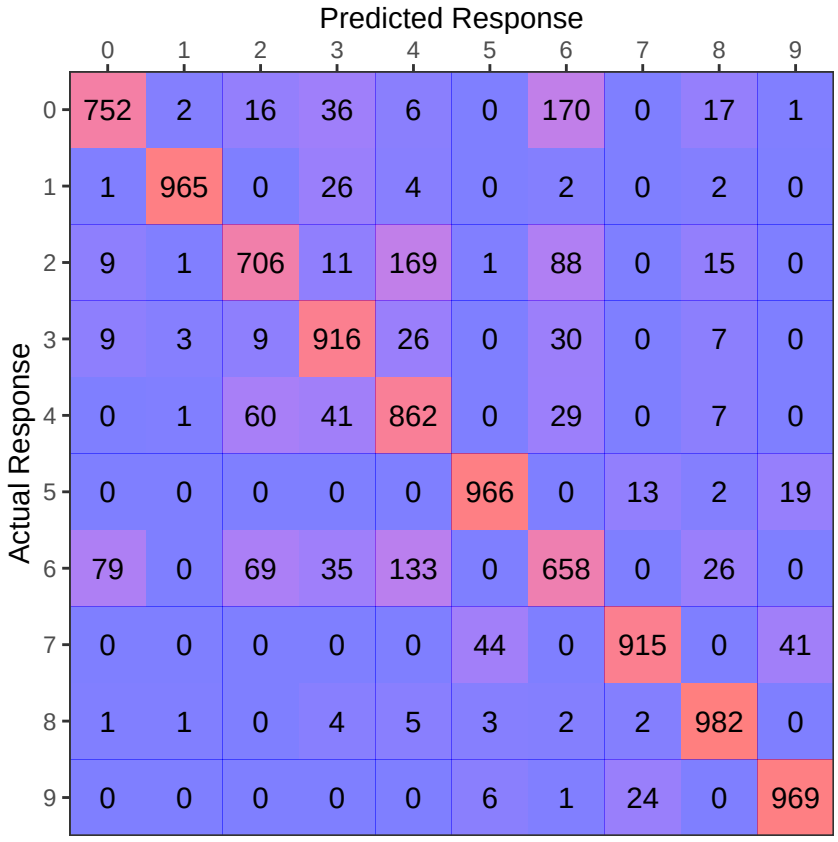
```
#LR
predicted_classes_lr = model_lr %>% predict(x_test) %>% k_argmax() %>% as.integer()
mx_lr = plot_confusion_matrix(predicted_responses = predicted_classes_lr,
                        actual_response = g_test)

#NN
predicted_classes_nn = model_nn %>% predict(x_test) %>% k_argmax() %>% as.integer()
mx_nn = plot_confusion_matrix(predicted_responses = predicted_classes_nn,
                        actual_response = g_test)

#CNN
predicted_classes_cnn = model_cnn %>% predict(x_test) %>% k_argmax() %>% as.integer()
mx_cnn = plot_confusion_matrix(predicted_responses = predicted_classes_cnn,
                        actual_response = g_test)
```

```
mx_lr
```



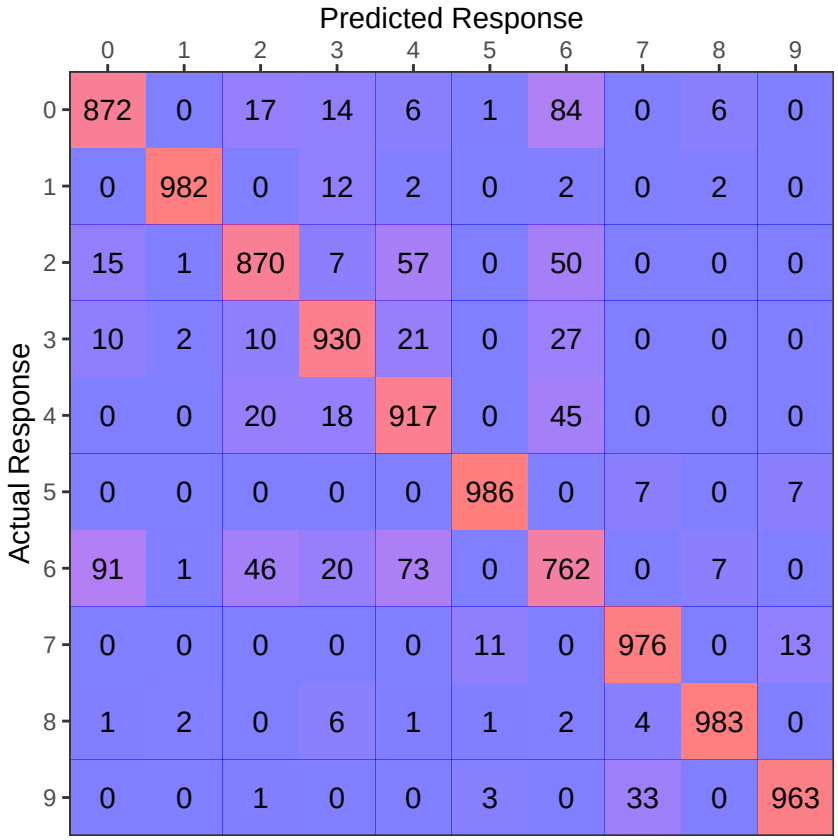
In the multi-class logistic regression model, the most frequently misclassified class is Shirt. The most frequent wrong label for this class is label 0 (T-shirt/top).

```
mx_nn
```



In the fully connected neural network, the most frequently misclassified class is Shirt. The most frequent wrong label for this class is label 4 (Coat).

```
mx_cnn
```

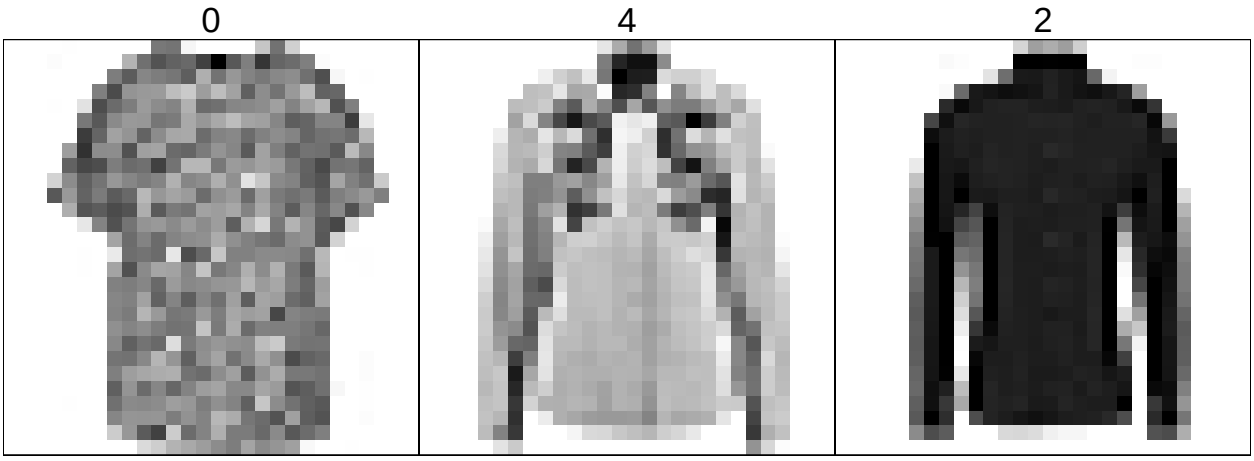


In the convolutional neural network, the most frequently misclassified class is Shirt. The most frequent wrong label for this class is label 0 (T-shirt/top).

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?

For CNN, the three most common incorrect classifications for class 'Shirt' is 'T-shirt/top'(label 0), 'Coat'(label 4), 'Pullover'(label 2). I would not have gotten these right.

```
misclassifications_0 = which(predicted_classes_cnn == 0 & g_test == 6)
idx_0 = misclassifications_0[1]
p1 = plot_grayscale(x_test[idx_0,,,], predicted_classes_cnn[idx_0])

misclassifications_4 = which(predicted_classes_cnn == 4 & g_test == 6)
idx_4 = misclassifications_4[1]
p2 = plot_grayscale(x_test[idx_4,,,], predicted_classes_cnn[idx_4])

misclassifications_2 = which(predicted_classes_cnn == 2 & g_test == 6)
idx_2 = misclassifications_2[3]
p3 = plot_grayscale(x_test[idx_2,,,], predicted_classes_cnn[idx_2])

plot_grid(p1, p2, p3, nrow = 1)
```


0
4
2