Question 7

```
# load the default data
library(ISLR2)
data(Default)
```

```
Default <- na.omit(Default)
n <- nrow(Default)
set.seed(123)
ntest <- trunc(n / 3)
testid <- sample(1:n, ntest)
```

```
# fit linear logistic regression, and calculate the accuracy
lfit <- glm(default ~ .,
            data = Default[-testid, ],
            family = binomial)

prob <- predict(lfit, Default[testid, ], type = "response")

pred <- ifelse(prob > 0.5, "Yes", "No")

mean(pred == Default$default[testid])
```

```
## [1] 0.9717972
```

```
# fit neural network
# first, scale model matrix
x <- scale(model.matrix(default ~ . -1, data = Default))
y <- ifelse(Default$default=="Yes", 1, 0)
```

```
# second, fit a neural network
library(keras3)
```

```
## Warning: package 'keras3' was built under R version 4.4.3
```

```
modnn <- keras_model_sequential () %>%
 layer_dense(units = 10, activation = "relu",input_shape = ncol(x)) %>%
 layer_dropout(rate = 0.4) %>%
 layer_dense(units = 1, activation = "sigmoid")
```

```
## Warning: Some Python package requirements declared via `py_require()` are not installed in the selec
##   pydot scipy pandas ipython
```

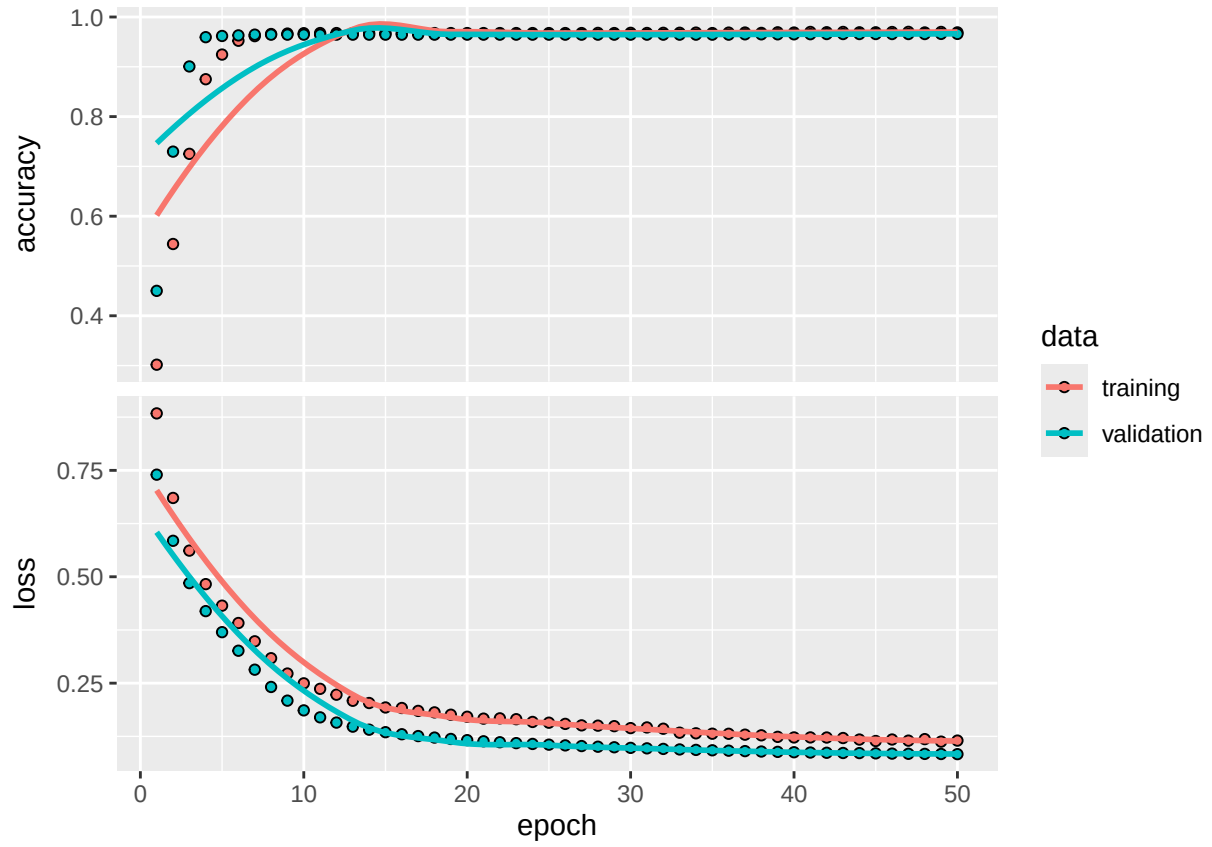
```
# third, compile
modnn %>% compile(
  optimizer = "adam",
  loss = "binary_crossentropy",
  metrics = "accuracy"
)
```

```r
# fourth, generate history
history <- modnn %>% fit(
  x[-testid, ], y[-testid],
  epochs = 50,
  batch_size = 128,
  validation_data = list(x[testid, ], y[testid]))
```

```
## Epoch 1/50
## 53/53 - 1s - 10ms/step - accuracy: 0.3018 - loss: 0.8838 - val_accuracy: 0.4500 - val_loss: 0.7398
## Epoch 2/50
## 53/53 - 0s - 1ms/step - accuracy: 0.5442 - loss: 0.6851 - val_accuracy: 0.7297 - val_loss: 0.5845
## Epoch 3/50
## 53/53 - 0s - 1ms/step - accuracy: 0.7252 - loss: 0.5614 - val_accuracy: 0.9007 - val_loss: 0.4851
## Epoch 4/50
## 53/53 - 0s - 1ms/step - accuracy: 0.8751 - loss: 0.4827 - val_accuracy: 0.9595 - val_loss: 0.4193
## Epoch 5/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9246 - loss: 0.4321 - val_accuracy: 0.9619 - val_loss: 0.3698
## Epoch 6/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9523 - loss: 0.3913 - val_accuracy: 0.9631 - val_loss: 0.3262
## Epoch 7/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9616 - loss: 0.3484 - val_accuracy: 0.9646 - val_loss: 0.2817
## Epoch 8/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9658 - loss: 0.3086 - val_accuracy: 0.9646 - val_loss: 0.2411
## Epoch 9/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9667 - loss: 0.2726 - val_accuracy: 0.9646 - val_loss: 0.2089
## Epoch 10/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9675 - loss: 0.2496 - val_accuracy: 0.9646 - val_loss: 0.1861
## Epoch 11/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9678 - loss: 0.2367 - val_accuracy: 0.9646 - val_loss: 0.1697
## Epoch 12/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9678 - loss: 0.2229 - val_accuracy: 0.9646 - val_loss: 0.1574
## Epoch 13/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9678 - loss: 0.2086 - val_accuracy: 0.9646 - val_loss: 0.1478
## Epoch 14/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9678 - loss: 0.2034 - val_accuracy: 0.9646 - val_loss: 0.1407
## Epoch 15/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9678 - loss: 0.1929 - val_accuracy: 0.9646 - val_loss: 0.1348
## Epoch 16/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9678 - loss: 0.1916 - val_accuracy: 0.9646 - val_loss: 0.1300
## Epoch 17/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9678 - loss: 0.1848 - val_accuracy: 0.9646 - val_loss: 0.1257
## Epoch 18/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9678 - loss: 0.1814 - val_accuracy: 0.9646 - val_loss: 0.1222
## Epoch 19/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9678 - loss: 0.1759 - val_accuracy: 0.9646 - val_loss: 0.1189
## Epoch 20/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9678 - loss: 0.1711 - val_accuracy: 0.9646 - val_loss: 0.1161
## Epoch 21/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9678 - loss: 0.1666 - val_accuracy: 0.9646 - val_loss: 0.1134
## Epoch 22/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9678 - loss: 0.1673 - val_accuracy: 0.9646 - val_loss: 0.1112
## Epoch 23/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9678 - loss: 0.1653 - val_accuracy: 0.9646 - val_loss: 0.1091
```

```
## Epoch 24/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9678 - loss: 0.1589 - val_accuracy: 0.9646 - val_loss: 0.1072
## Epoch 25/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9678 - loss: 0.1571 - val_accuracy: 0.9646 - val_loss: 0.1055
## Epoch 26/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9678 - loss: 0.1543 - val_accuracy: 0.9646 - val_loss: 0.1038
## Epoch 27/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9678 - loss: 0.1508 - val_accuracy: 0.9646 - val_loss: 0.1022
## Epoch 28/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9681 - loss: 0.1502 - val_accuracy: 0.9646 - val_loss: 0.1007
## Epoch 29/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9678 - loss: 0.1490 - val_accuracy: 0.9646 - val_loss: 0.0994
## Epoch 30/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9682 - loss: 0.1444 - val_accuracy: 0.9646 - val_loss: 0.0980
## Epoch 31/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9679 - loss: 0.1460 - val_accuracy: 0.9646 - val_loss: 0.0968
## Epoch 32/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9682 - loss: 0.1430 - val_accuracy: 0.9646 - val_loss: 0.0956
## Epoch 33/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9684 - loss: 0.1337 - val_accuracy: 0.9646 - val_loss: 0.0943
## Epoch 34/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9687 - loss: 0.1322 - val_accuracy: 0.9646 - val_loss: 0.0932
## Epoch 35/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9678 - loss: 0.1313 - val_accuracy: 0.9646 - val_loss: 0.0923
## Epoch 36/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9687 - loss: 0.1311 - val_accuracy: 0.9646 - val_loss: 0.0914
## Epoch 37/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9688 - loss: 0.1290 - val_accuracy: 0.9652 - val_loss: 0.0903
## Epoch 38/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9681 - loss: 0.1275 - val_accuracy: 0.9652 - val_loss: 0.0894
## Epoch 39/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9693 - loss: 0.1238 - val_accuracy: 0.9652 - val_loss: 0.0886
## Epoch 40/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9688 - loss: 0.1225 - val_accuracy: 0.9652 - val_loss: 0.0877
## Epoch 41/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9700 - loss: 0.1225 - val_accuracy: 0.9655 - val_loss: 0.0870
## Epoch 42/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9694 - loss: 0.1223 - val_accuracy: 0.9658 - val_loss: 0.0864
## Epoch 43/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9700 - loss: 0.1211 - val_accuracy: 0.9658 - val_loss: 0.0860
## Epoch 44/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9693 - loss: 0.1175 - val_accuracy: 0.9658 - val_loss: 0.0857
## Epoch 45/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9693 - loss: 0.1141 - val_accuracy: 0.9658 - val_loss: 0.0851
## Epoch 46/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9699 - loss: 0.1176 - val_accuracy: 0.9658 - val_loss: 0.0845
## Epoch 47/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9702 - loss: 0.1150 - val_accuracy: 0.9658 - val_loss: 0.0841
## Epoch 48/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9690 - loss: 0.1185 - val_accuracy: 0.9658 - val_loss: 0.0838
## Epoch 49/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9700 - loss: 0.1126 - val_accuracy: 0.9661 - val_loss: 0.0836
## Epoch 50/50
## 53/53 - 0s - 1ms/step - accuracy: 0.9691 - loss: 0.1151 - val_accuracy: 0.9661 - val_loss: 0.0832
```

```
plot(history)
```




```
# fifth, calculate accuracy
prob_nn <- modnn %>% predict(x[testid, ])
```

```
## 105/105 - 0s - 563us/step
```

```
pred_nn <- ifelse(prob_nn > 0.5, 1, 0)

mean(pred_nn == y[testid])
```

```
## [1] 0.9660966
```

Conclusions: by accuracy, the logistic regression (0.972) is slightly better than the neural network (0.966).

Question 8

```
# first, load the pretrained model
model <- application_resnet50(weights = "imagenet")
```

```
# second, load my animals
img_dir <- "/Users/luqiwang/Documents/GitHub/ECL298_2026/Deep_learning/my_animals"
image_names <- list.files(img_dir)
num_images <- length(image_names)
```

```r
# create input matrix
x <- array(dim = c(num_images , 224, 224, 3))

for (i in 1:num_images) {
  img_path <- paste(img_dir, image_names[i], sep = "/")
  img <- image_load(img_path, target_size = c(224, 224))
  x[i,,, ] <- image_to_array(img)
}
x <- imagenet_preprocess_input(x)
```

```r
# predict
pred10 <- model %>% predict(x) %>% imagenet_decode_predictions (top = 5)
```

```
## 1/1 - 1s - 968ms/step
```

```r
# report the classification and probability
for (i in seq_along(pred10)) {
  cat("\n--------------------------------\n")
  cat("Image:", image_names[i], "\n")
  print(pred10[[i]][, c("class_description", "score")])
}
```

```
##
## --------------------------------
## Image: cat.jpg
##   class_description      score
## 1      Egyptian_cat 0.64067602
## 2       Siamese_cat 0.13591579
## 3            bucket 0.05045168
## 4             tabby 0.03335772
## 5              lynx 0.01803129
##
## --------------------------------
## Image: cow.jpg
##   class_description       score
## 1                ox 0.927057147
## 2            oxcart 0.056994326
## 3              plow 0.003551440
## 4     water_buffalo 0.002845773
## 5               ram 0.001848965
##
## --------------------------------
## Image: dog.jpg
##    class_description       score
## 1   golden_retriever 0.959681749
## 2 Labrador_retriever 0.007350623
## 3     cocker_spaniel 0.004809695
## 4         toy_poodle 0.003892529
## 5          Chihuahua 0.003456752
##
## --------------------------------
## Image: elephant.jpg
```

```
##   class_description       score
## 1            tusker 0.614995539
## 2  African_elephant 0.341503590
## 3   Indian_elephant 0.028044462
## 4    frilled_lizard 0.003446487
## 5   standard_poodle 0.002623818
##
## --------------------------------
## Image: goat.jpg
##   class_description       score
## 1               ram 0.872128367
## 2           bighorn 0.115225233
## 3             llama 0.006183626
## 4              ibex 0.003843497
## 5                ox 0.000484370
##
## --------------------------------
## Image: horse.jpg
##             class_description      score
## 1             standard_poodle 0.82078093
## 2          Bedlington_terrier 0.04164288
## 3                       llama 0.03242446
## 4 soft-coated_wheaten_terrier 0.02212278
## 5                      kuvasz 0.01883873
##
## --------------------------------
## Image: lion.jpg
##   class_description        score
## 1              lion 9.992295e-01
## 2              chow 2.032054e-04
## 3           gorilla 1.304272e-04
## 4         orangutan 8.805774e-05
## 5           macaque 5.263506e-05
##
## --------------------------------
## Image: monkey.jpg
##   class_description      score
## 1           macaque 0.59775478
## 2              titi 0.12443015
## 3          mongoose 0.08525364
## 4           vulture 0.04471009
## 5     spider_monkey 0.03934811
##
## --------------------------------
## Image: sheep.jpg
##   class_description       score
## 1               ram 0.960072756
## 2           bighorn 0.019622369
## 3             llama 0.007530934
## 4          komondor 0.002147973
## 5   standard_poodle 0.001941283
##
## --------------------------------
## Image: tiger.jpg
```

```
##   class_description       score
## 1             tiger 0.853972614
## 2         tiger_cat 0.129047453
## 3              lynx 0.013698417
## 4           leopard 0.001281171
## 5            jaguar 0.001045595
```