ML-Tools-Assignment-2

Gyongyver Kamenar (2103380)

3/27/2022

```
library(tidyverse)
library(kableExtra)
library(keras)
library(ggplot2)
library(imager)

# Set my theme
devtools::source_url('https://raw.githubusercontent.com/gyongyver-droid/ceu-data-analysis/master/Assignmen
#theme_set(theme_gyongyver())
```

Problem 1

A)

What would be an appropriate metric to evaluate your models? Why?

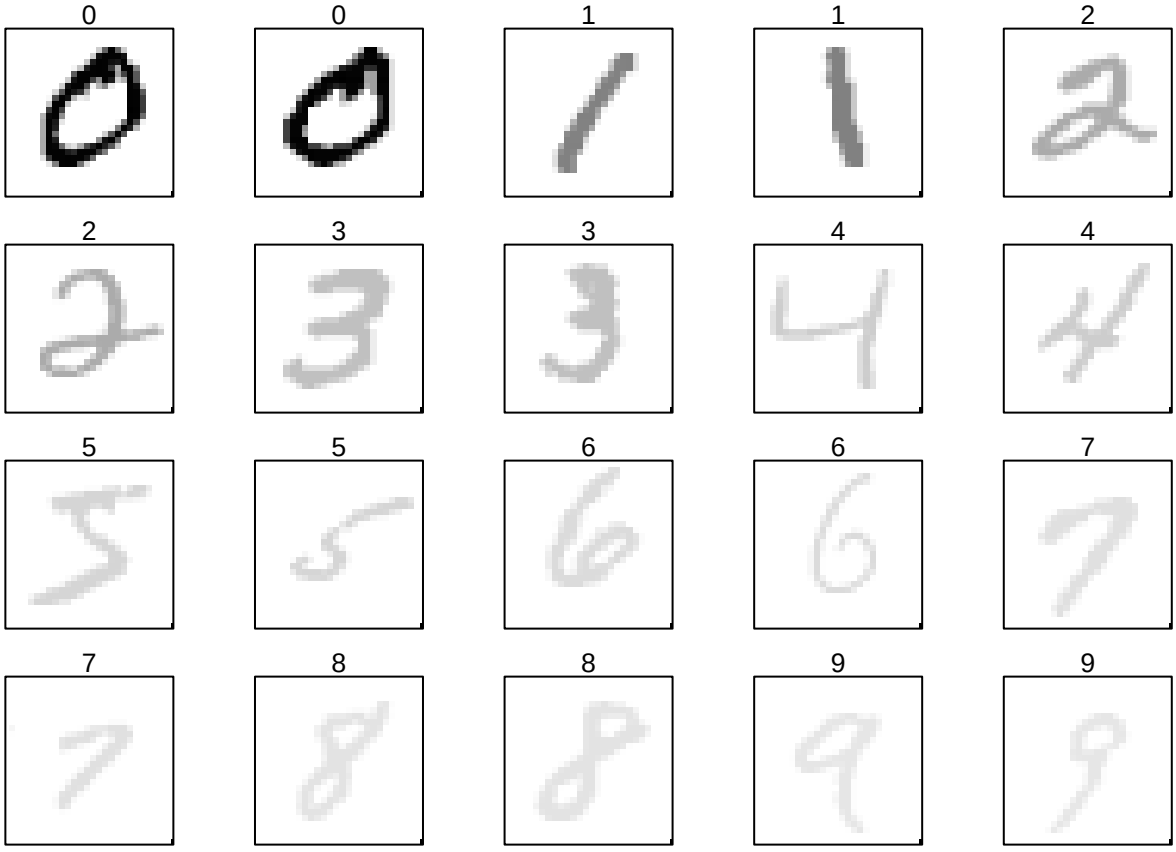
Accuracy would be an appropriate metric

B)

Get the data and show some example images from the data.

```
myseed<-20220403
set.seed(myseed)
data <-dataset_mnist()

train_x<-data$train$x
train_y<-data$train$y

test_x<-data$test$x
test_y<-data$test$y

# Merge features with labels for the plot
train <-cbind(as.tibble(data$train$x),y=data$train$y)

# Factorizing and scaling for plotting
train <- train %>% mutate(
  y = as.factor(y),
  across(-y, ~./255)
)



# Plot the images
rotate <- function(x) t(apply(x, 2, rev))
```

```
par(mfrow=c(4, 5), pty='s', mai=c(0.1, 0, 0.15, 0.1))
for (lab in 0:9) {
  samp <- train %>%
    filter(y == lab)
  for (i in 1:2) {
    img <- matrix(as.numeric(samp[i, -1]), 28, 28, byrow = TRUE)
    image(t(rotate(img))[,28:1], axes = FALSE, col = grey(seq(1, 0, length = 256)),xlim = c(1,0))
    box(lty = 'solid')
    title(main = lab, font.main=22)
  }
}
```



C)

Train a simple fully connected network with a single hidden layer to predict the digits. Do not forget to normalize the data similarly to what we saw with FMNIST in class.

```
train_x<-array_reshape(train_x/255,c(nrow(train_x),784))
test_x<-array_reshape(test_x/255,c(nrow(test_x),784))

train_y <- to_categorical(train_y, num_classes = 10)
test_y <- to_categorical(test_y, num_classes = 10)

simple_keras <- keras_model_sequential()
simple_keras |>
    layer_dense(units = 128, activation = 'relu', input_shape = c(784)) |>
    layer_dropout(rate = 0.2) |>
    layer_dense(units = 10, activation = 'softmax')
```

```
summary(simple_keras)
```

```
## Model: "sequential"
## _________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## =========================================================================
##  dense_1 (Dense)                    (None, 128)                     100480
##
##  dropout (Dropout)                  (None, 128)                     0
##
##  dense (Dense)                      (None, 10)                      1290
##
## =========================================================================
## Total params: 101,770
## Trainable params: 101,770
## Non-trainable params: 0
## _________________________________________________________________________
```

```
compile(
    simple_keras,
    loss = 'categorical_crossentropy',
    optimizer = optimizer_adam(),
    metrics = c('accuracy')
)

fit(
    simple_keras, train_x, train_y,
    epochs = 20, batch_size = 20,
    validation_data = list(test_x, test_y)
)
```

```
evaluate(simple_keras, test_x, test_y)
```

```
##       loss   accuracy
## 0.09759966 0.98049998
```

D)

Experiment with different network architectures and settings (number of hidden layers, number of nodes, type and extent of regularization, etc.). Train at least 5 models. Explain what you have tried, what worked and what did not. Make sure that you use enough epochs so that the validation error starts flattening out - provide a plot about the training history.

E)

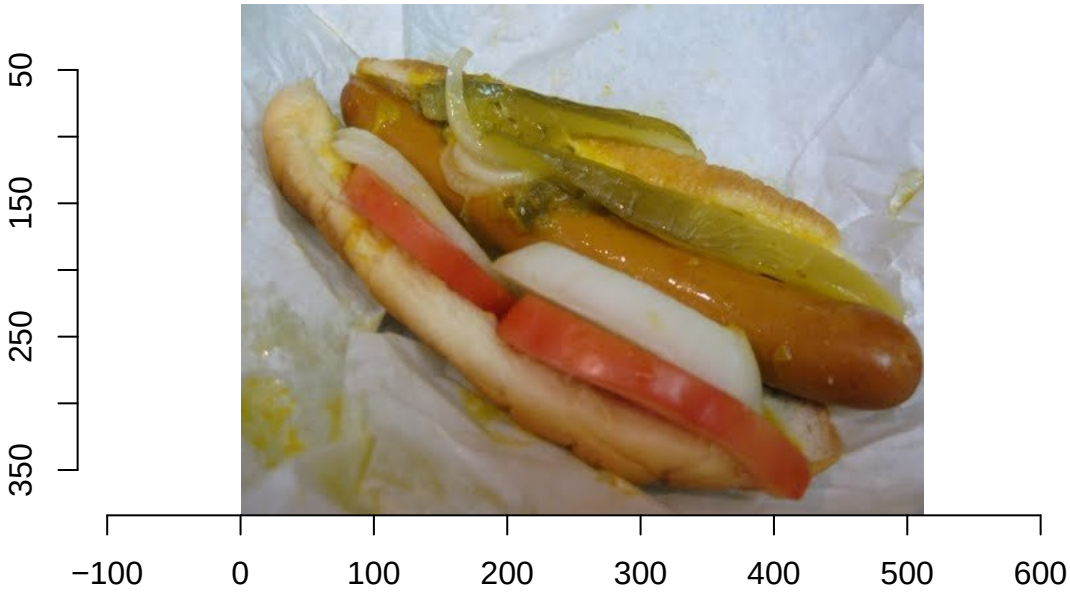
Choose a final model and evaluate it on the test set. How does test error compare to validation error?

Problem 2

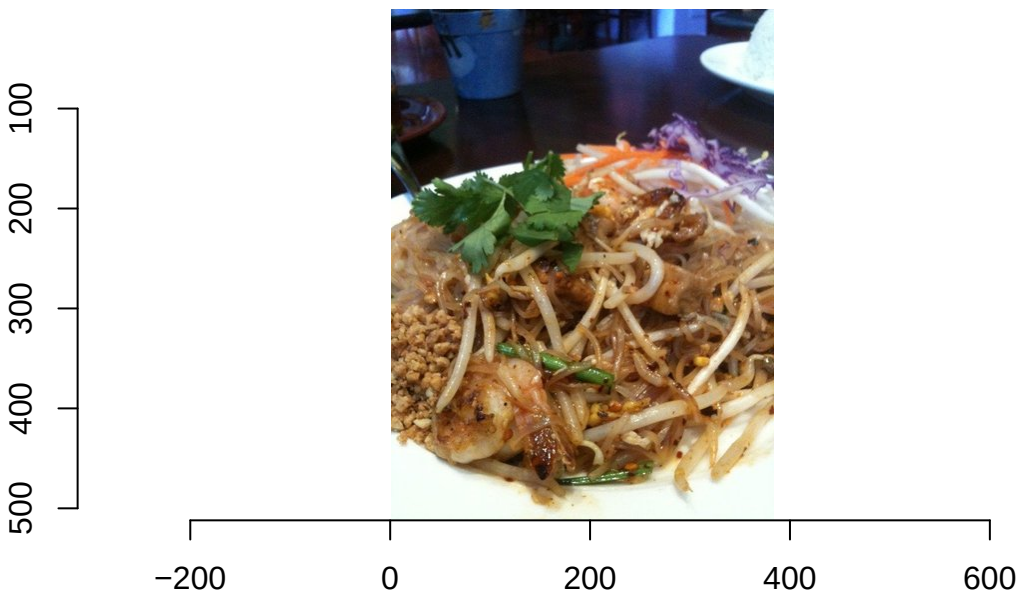
A)

Show two images: one hot dog and one not hot dog. (Hint: You may use knitr::include_graphics() or install the imager package to easily accomplish this.)

```
#Show hot dog
path<-"C:/CEU/DS1/ceu-ds1/data/train/"
hot_dog<-load.image(paste0(path,"hot_dog/7896.jpg"))
plot(hot_dog)
```




```
# Show not hot dog
not_hot_dog<-load.image(paste0(path,"not_hot_dog/4770.jpg"))
plot(not_hot_dog)
```



B)

What would be a good metric to evaluate such a prediction?

C)

To be able to train a neural network for prediction, let's first build data batch generator functions as we did in class for data augmentation (train_generator and valid_generator).

```
batch_size <- 16

train_datagen <- image_data_generator(rescale = 1/255)

valid_datagen <- image_data_generator(rescale = 1/255)

train_generator <- flow_images_from_directory(
    directory="C:/CEU/DS1/ceu-ds1/data/train/",
    class_mode = "binary",
    generator = train_datagen,
    batch_size = batch_size,
    target_size = c(128, 128)
)

valid_generator <- flow_images_from_directory(
    directory="C:/CEU/DS1/ceu-ds1/data/test/",
    class_mode = "binary",
    generator = valid_datagen,
    batch_size = batch_size,
    target_size = c(128, 128)
```

```
)
```

D)

Build a simple convolutional neural network (CNN) to predict if an image is a hot dog or not. Evaluate your model on the test set. (Hint: Account for the fact that you work with colored images in the input_shape parameter: set the third dimension to 3 instead of 1.)

```
cnn_model_w_augmentation <- keras_model_sequential()
cnn_model_w_augmentation |>
    layer_conv_2d(
        filters = 32,
        kernel_size = c(3, 3),
        activation = 'relu',
        input_shape = c(128, 128, 3)
    ) |>
    layer_max_pooling_2d(pool_size = c(2, 2)) |>
    layer_dropout(rate = 0.2) |>
    layer_flatten() |>
    layer_dense(units = 32, activation = 'relu') |>
    layer_dense(units = 1, activation = 'softmax')

compile(
    cnn_model_w_augmentation,
    loss = 'binary_crossentropy',
    optimizer = optimizer_adam(),
    metrics = c('accuracy')
)

fit(
    cnn_model_w_augmentation,
    train_generator,
    epochs = 20,
    steps_per_epoch = 10, #nrow(data_train_x) / batch_size,  # this does not make a difference here -- bat
    validation_data = valid_generator,
    validation_steps = 10#nrow(data_valid_x) / batch_size
)
```

The train and test accuracy are both about 50% which means, that this binary classification model is not better than random :(.

E)

Could data augmentation techniques help with achieving higher predictive accuracy? Try some augmentations that you think make sense and compare to your previous model.

optional

Try to rely on some pre-built neural networks to aid prediction. Can you achieve a better performance using transfer learning for this problem?