MNIST_CNN.r

esteban

2021-09-07

```
# Training an image recognizer on MNIST data
# CNN architecture

# Install & libraries
#devtools::install_github("rstudio/keras")
library(keras)
#install_keras()

#install.packages("caret")
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
# Data Preparation -----------------------------------------------------

batch_size <- 128
num_classes <- 10
epochs <- 15

# Input image dimensions
img_rows <- 28
img_cols <- 28


# input layer: use MNIST images
mnist <- dataset_mnist()
x_train <- mnist$train$x; y_train <- mnist$train$y
x_test <- mnist$test$x; y_test <- mnist$test$y

# Redefine  dimension of train/test inputs
x_train <- array_reshape(x_train, c(nrow(x_train), img_rows, img_cols, 1))
x_test <- array_reshape(x_test, c(nrow(x_test), img_rows, img_cols, 1))
input_shape <- c(img_rows, img_cols, 1)

# Transform RGB values into [0,1] range
x_train <- x_train / 255
x_test <- x_test / 255

cat('x_train_shape:', dim(x_train), '\n')
```

```
## x_train_shape: 60000 28 28 1
```

```
cat(nrow(x_train), 'train samples\n')
```

```
## 60000 train samples
```

```
cat(nrow(x_test), 'test samples\n')
```

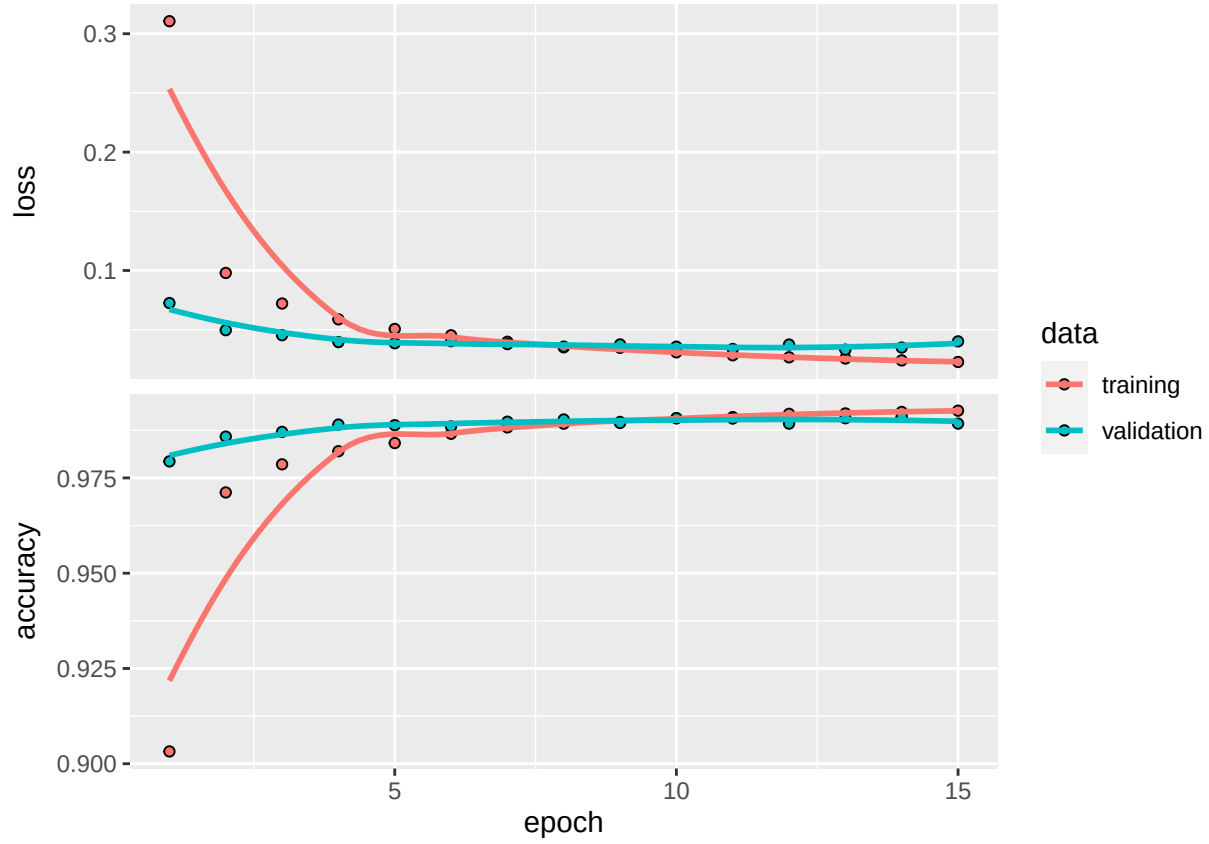
```
## 10000 test samples
```

```
# Convert class vectors to binary class matrices
y_train <- to_categorical(y_train, num_classes)
y_test <- to_categorical(y_test, num_classes)

################## Define Model ###################################

# defining the model and layers
model <- keras_model_sequential()
model %>%
  layer_conv_2d(filters = 32, kernel_size = c(3,3),
                activation = 'relu', input_shape = input_shape) %>%
  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')

summary(model)
```

```
## Model: "sequential"
## ________________________________________________________________________________
## 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_1 (Dropout)                 (None, 12, 12, 64)              0
## ________________________________________________________________________________
## flatten (Flatten)                   (None, 9216)                    0
## ________________________________________________________________________________
## dense_1 (Dense)                     (None, 128)                     1179776
## ________________________________________________________________________________
## dropout (Dropout)                   (None, 128)                     0
## ________________________________________________________________________________
## dense (Dense)                       (None, 10)                      1290
## ================================================================================
## Total params: 1,199,882
## Trainable params: 1,199,882
## Non-trainable params: 0
## ________________________________________________________________________________
```

```
# compile (define loss and optimizer)
model %>% compile(
  loss = loss_categorical_crossentropy,
  optimizer = optimizer_adadelta(),
  metrics = c('accuracy')
)
# train (fit)
history <- model %>% fit(
  x_train, y_train,
  batch_size = batch_size,
  epochs = epochs,
  validation_split = 0.2
)

plot(history)
```

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




```
# evaluate
scores <- model %>% evaluate(x_test, y_test, verbose=0)

# Output metrics
cat('Test loss:', scores[[1]], '\n')
```

```
## Test loss: 0.02708892
```

```
cat('Test accuracy:', scores[[2]], '\n')
```

```
## Test accuracy: 0.9918
```

```
#predict
# keras/tensorflow version < 2.6
#y_pred <- model %>% predict_classes(x_test)

# keras/tensorflow version >= 2.6
# se obtiene un objeto tf.tensor
y_pred <- model %>% predict(x_test) %>% k_argmax()
# se pasa a vector
# https://tensorflow.rstudio.com/guide/tensorflow/tensors/
y_pred <- y_pred %>% shape() %>% unlist()

y_pred[1:100]
```

```
##   [1] 7 2 1 0 4 1 4 9 5 9 0 6 9 0 1 5 9 7 3 4 9 6 6 5 4 0 7 4 0 1 3 1 3 4 7 2 7
##  [38] 1 2 1 1 7 4 2 3 5 1 2 4 4 6 3 5 5 6 0 4 1 9 5 7 8 9 3 7 4 6 4 3 0 7 0 2 9
##  [75] 1 7 3 2 9 7 7 6 2 7 8 4 7 3 6 1 3 6 4 3 1 4 1 7 6 9
```

```
#confusion Matrix

confusionMatrix(as.factor(mnist$test$y), as.factor(y_pred))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1    2    3    4    5    6    7    8    9
##          0  976    0    0    0    1    0    2    1    0    0
##          1    0 1133    2    0    0    0    0    0    0    0
##          2    2    1 1023    0    1    0    0    4    1    0
##          3    0    0    3 1004    0    2    0    0    1    0
##          4    0    0    0    0  980    0    1    0    0    1
##          5    2    0    1    6    0  878    3    0    1    1
##          6    5    2    0    0    2    2  946    0    1    0
##          7    0    3    3    1    0    0    0 1018    1    2
##          8    1    2    1    1    0    0    1    2  964    2
##          9    0    1    0    0    6    2    0    3    1  996
##
## Overall Statistics
##
##                Accuracy : 0.9918
##                  95% CI : (0.9898, 0.9935)
##     No Information Rate : 0.1142
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9909
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9899   0.9921   0.9903   0.9921   0.9899   0.9932
```

```
## Specificity            0.9996   0.9998   0.9990   0.9993   0.9998   0.9985
## Pos Pred Value         0.9959   0.9982   0.9913   0.9941   0.9980   0.9843
## Neg Pred Value         0.9989   0.9990   0.9989   0.9991   0.9989   0.9993
## Prevalence             0.0986   0.1142   0.1033   0.1012   0.0990   0.0884
## Detection Rate         0.0976   0.1133   0.1023   0.1004   0.0980   0.0878
## Detection Prevalence   0.0980   0.1135   0.1032   0.1010   0.0982   0.0892
## Balanced Accuracy      0.9947   0.9959   0.9947   0.9957   0.9948   0.9958
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity            0.9927   0.9903   0.9938   0.9940
## Specificity            0.9987   0.9989   0.9989   0.9986
## Pos Pred Value         0.9875   0.9903   0.9897   0.9871
## Neg Pred Value         0.9992   0.9989   0.9993   0.9993
## Prevalence             0.0953   0.1028   0.0970   0.1002
## Detection Rate         0.0946   0.1018   0.0964   0.0996
## Detection Prevalence   0.0958   0.1028   0.0974   0.1009
## Balanced Accuracy      0.9957   0.9946   0.9964   0.9963
```