Deep neural networks – convolutional neural network

Tomasz Górecki

Last update: 18.05.2022

Libraries

```
library(keras)
library(dplyr, quietly = TRUE, warn.conflicts = FALSE)
library(ggplot2, quietly = TRUE, warn.conflicts = FALSE)
```

MNIST data set example

Description

The MNIST dataset is included with Keras and can be accessed using the dataset_mnist() function. The MNIST database of handwritten digits, available from this page, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image. There have been a number of scientific papers on attempts to achieve the lowest error rate; one paper, using a hierarchical system of convolutional neural networks, manages to get an error rate on the MNIST database of 0.17%.

Type	Classifier	Error rate (%)
Linear classifier	Pairwise linear classifier	7.60
Non-linear classifier	40 PCA + quadratic classifier	3.30
Random Forest (RF)	Fast Unified Random Forests (RF-SRC)	2.80
Deep neural network (DNN)	2-layer 784-800-10	1.60
Boosted Stumps	Product of stumps on Haar features	0.87
Deep neural network (DNN)	2-layer 784-800-10	0.70
Support-vector machine (SVM)	Virtual SVM, deg-9 poly	0.56
K-Nearest Neighbors (KNN)	K-NN with non-linear deformation	0.52
Deep neural network (DNN)	6-layer 784-2500-2000-1500-1000-500-10	0.35
Convolutional neural network (CNN)	6-layer 784-40-80-500-1000-2000-10	0.31
Convolutional neural network (CNN)	6-layer 784-50-100-500-1000-10-10	0.27
Convolutional neural network (CNN)	Committee of 35 CNNs, 1-20-P-40-P-150-10	0.23
Convolutional neural network (CNN)	Committee of 5 CNNs, 6-layer 784-50-100-500-1000-10-10	0.21
Random Multimodel Deep Learning (RMDL)	10 NN-10 RNN - 10 CNN	0.18
Convolutional neural network (CNN)	Committee of 20 CNNS with Squeeze-and-Excitation Networks	0.17

Loading data set

```
mnist <- dataset_mnist()
c(x_train, y_train) %<-% mnist$train # Train set features, train set labels
c(x_test, y_test) %<-% mnist$test # Test set features, test set labels

dim(x_train) # The dimensions of the training feature set (the images)
```

```
## [1] 60000    28    28
```

```
dim(x_test)
```

```
## [1] 10000    28    28
```

The x data is a 3-d array (images, width, height) of grayscale values. The y data is an integer vector with values ranging from 0 to 9.

Data prepare

These images are not in the the correct shape as tensors, as the number of channels is missing.

```
# Reshape
img_rows <- 28
img_cols <- 28
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)
```

The data must be normalized. Since the pixel values represent brigness on a scale from 0 (black) to 255 (white), they can all be rescaled by dividing each by the maximum value of 255.

```
# Rescale
x_train <- x_train / 255
x_test <- x_test / 255
```

To prepare this data for training we one-hot encode the vectors into binary class matrices using the Keras to_categorical() function. One hot encoding is a vector representation where all elements of the vector are 0 except one, which has 1 as its value (assigning 1 to working feature and 0's to other idle features).

```
num_classes <- 10
y_train <- to_categorical(y_train, num_classes)
y_test <- to_categorical(y_test, num_classes)
```

9 first encoded digits from trainig set are below (the first column corresponds to zero, second to one, etc.):

```
head(y_train, 9)
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
##  [1,]    0    0    0    0    0    1    0    0    0     0
##  [2,]    1    0    0    0    0    0    0    0    0     0
##  [3,]    0    0    0    0    1    0    0    0    0     0
##  [4,]    0    1    0    0    0    0    0    0    0     0
##  [5,]    0    0    0    0    0    0    0    0    0     1
##  [6,]    0    0    1    0    0    0    0    0    0     0
##  [7,]    0    1    0    0    0    0    0    0    0     0
##  [8,]    0    0    0    1    0    0    0    0    0     0
##  [9,]    0    1    0    0    0    0    0    0    0     0
```

Defining the model

```
model <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32,
                kernel_size = c(3, 3),
                activation = 'relu',
                input_shape = input_shape) %>% # (3 * 3 * 1 + 1) * 32 = 320
  # (filter_rows * filter_cols * filters_previous_layer + 1) * filters
  layer_batch_normalization() %>% # 2 * 32 + 2 * 32 = 128 (2 learnable & 2 non-learnable)
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 64,
                kernel_size = c(3, 3),
                activation = 'relu') %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>% # (3 * 3 * 32 + 1) * 64 = 18 496
  layer_dropout(rate = 0.25) %>%
  layer_flatten() %>% # 5 * 5 * 64 neurons
  layer_dense(units = 64,
              activation = 'relu') %>% # 1600 * 64 + 64 = 102 464
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = num_classes,
              activation = 'softmax') # 64 * 10 + 10 = 650
```

Summary the model

A summary of the model shows 122058 learnable parameters.

```
summary(model)
```

```
## Model: "sequential"
## ________________________________________________________________________________
## Layer (type)                     Output Shape                     Param #
## ================================================================================
## conv2d_1 (Conv2D)                (None, 26, 26, 32)               320
## ________________________________________________________________________________
## batch_normalization (BatchNormaliza (None, 26, 26, 32)            128
## ________________________________________________________________________________
## max_pooling2d_1 (MaxPooling2D)   (None, 13, 13, 32)               0
## ________________________________________________________________________________
## conv2d (Conv2D)                  (None, 11, 11, 64)               18496
## ________________________________________________________________________________
## max_pooling2d (MaxPooling2D)     (None, 5, 5, 64)                 0
## ________________________________________________________________________________
## dropout_1 (Dropout)              (None, 5, 5, 64)                 0
## ________________________________________________________________________________
## flatten (Flatten)                (None, 1600)                     0
## ________________________________________________________________________________
## dense_1 (Dense)                    (None, 64)                      102464
## ________________________________________________________________________________
## dropout (Dropout)                  (None, 64)                      0
## ________________________________________________________________________________
## dense (Dense)                      (None, 10)                      650
## ================================================================================
## Total params: 122,058
## Trainable params: 121,994
## Non-trainable params: 64
## ________________________________________________________________________________
```

Plot the model

```
# devtools::install_github('andrie/deepviz')
# deepviz::plot_model(model)
```

Compile the model

- Loss function – This measures how accurate the model is during training. We want to minimize this function to 'steer' the model in the right direction.
- Optimizer – This is how the model is updated based on the data it sees and its loss function.
- Metrics – Used to monitor the training and testing steps. The following example uses accuracy, the fraction of the digits that are correctly classified.

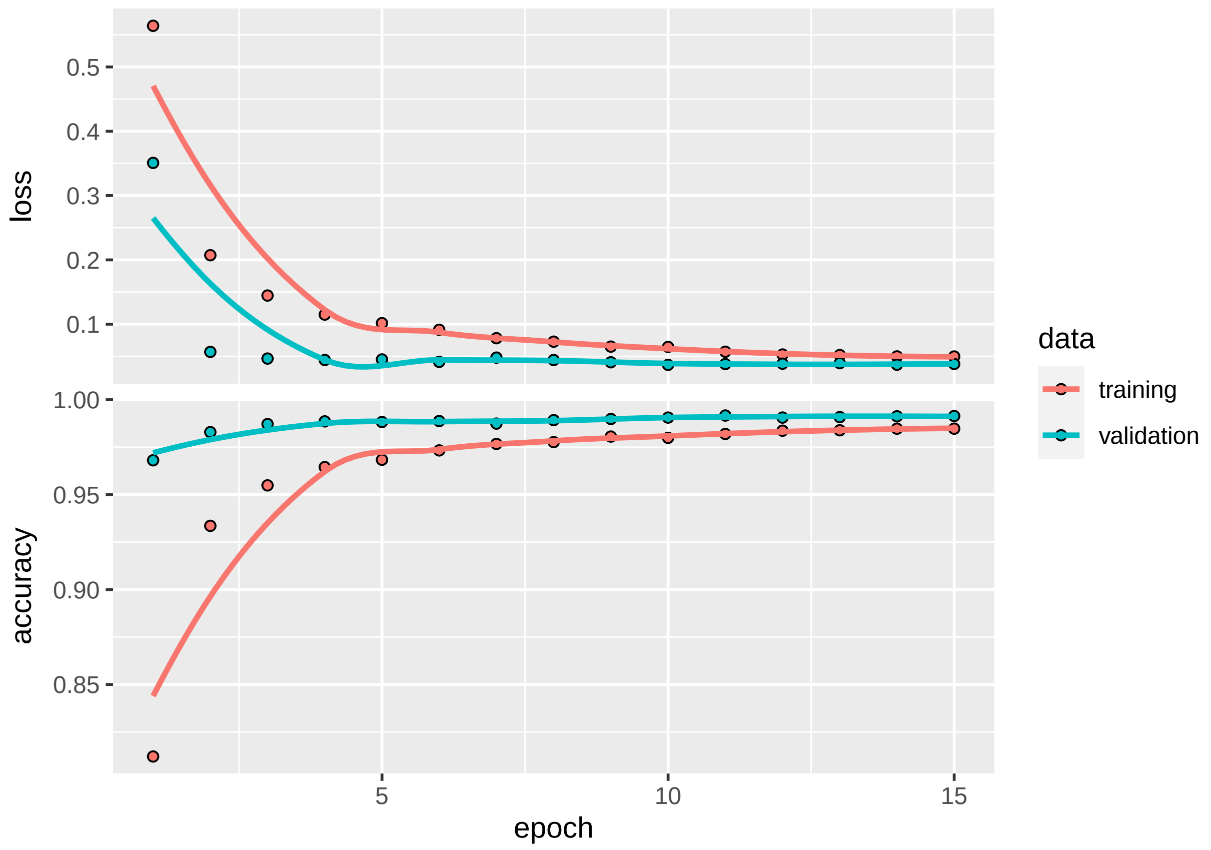
```
model %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_adam(),
  metrics = 'accuracy')
```

Training the model

A mini-batch size of 128 will allow the tensors to fit into the memory most of NVidia graphics processing unit. The model will run over 15 epochs, with a validation split set at 0.2.

```
# tensorboard('logs/run_a')
model %>% fit(x_train,
              y_train,
              epochs = 15, # Number of epochs
              batch_size = 128, # Size of batch in single step
              validation_split = 0.2 # Percent of data in validation sets
              # callbacks = callback_tensorboard('logs/run_a')
) -> model_cnn
plot(model_cnn)
```

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



Evaluate the model

```
model %>% evaluate(x_train, y_train, verbose = 0) # Evaluate the model's performance on the train data
```

```
## $loss
## [1] 0.01478
##
## $accuracy
## [1] 0.9958
```

```
model %>% evaluate(x_test, y_test, verbose = 0) # Evaluate the model's performance on the test data
```

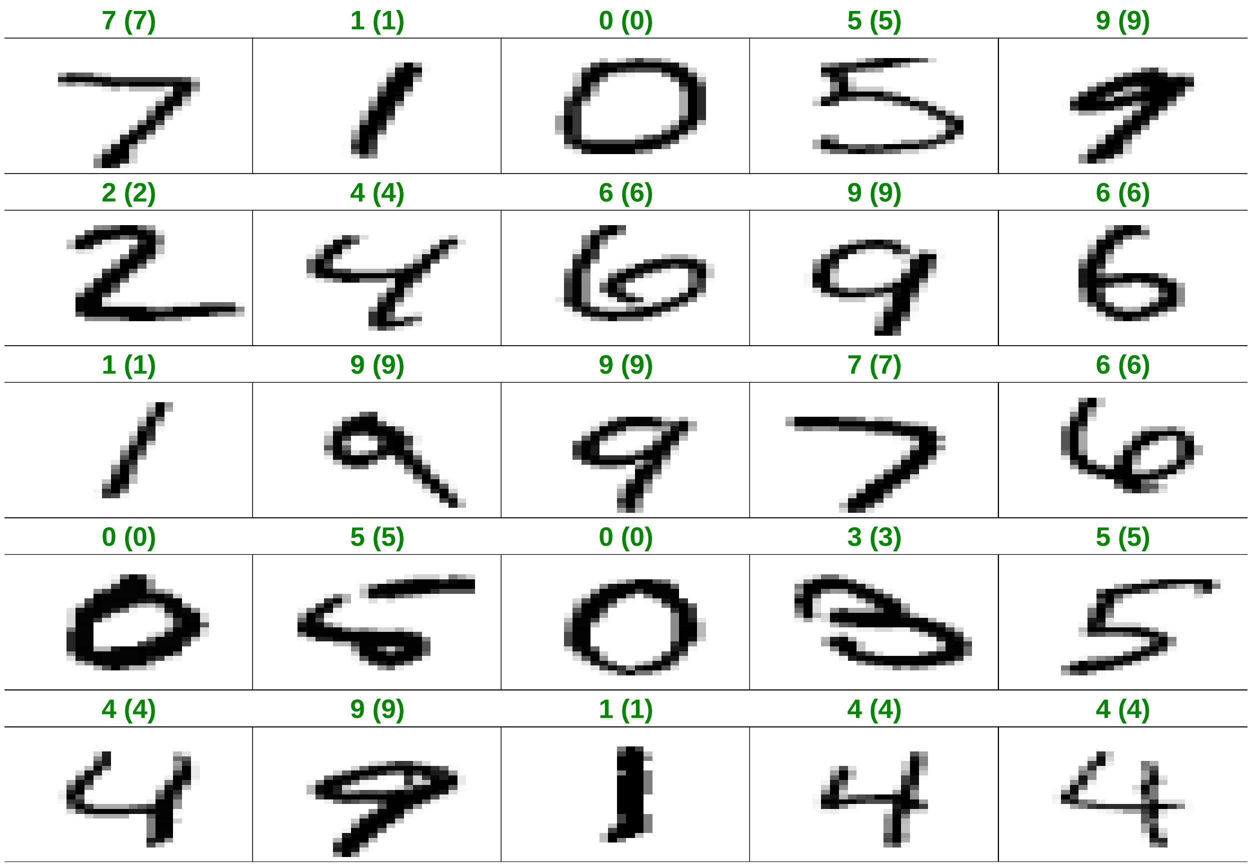
```
## $loss
## [1] 0.03012
##
## $accuracy
## [1] 0.9905
```

Predictions

```
model %>% predict(x_test) -> predictions # Predicted probabilities on test data
model %>%
  predict(x_test) %>%
  k_argmax() %>%
  as.numeric() -> predicted_digits # Predicted digits on test data
```

A prediction is an array of 10 numbers. These describe the 'confidence' of the model that the image corresponds to each of the 10 different digits. Let's plot several images with their predictions. Correct prediction labels are green and incorrect prediction labels are red.

```r
par(mfcol = c(5, 5))
par(mar = c(0, 0, 1.5, 0), xaxs = 'i', yaxs = 'i')
for (i in 1:25) {
  img <- mnist$test$x[i, , ]
  img <- t(apply(img, 2, rev))
  if (predicted_digits[i] == mnist$test$y[i]) {
    color <- '#008800'
  } else {
    color <- '#bb0000'
  }
  image(1:28, 1:28, img, col = gray((255:0) / 255), xaxt = 'n', yaxt = 'n',
        main = paste0(predicted_digits[i], ' (',
                      mnist$test$y[i], ')'),
        col.main = color)
}
```

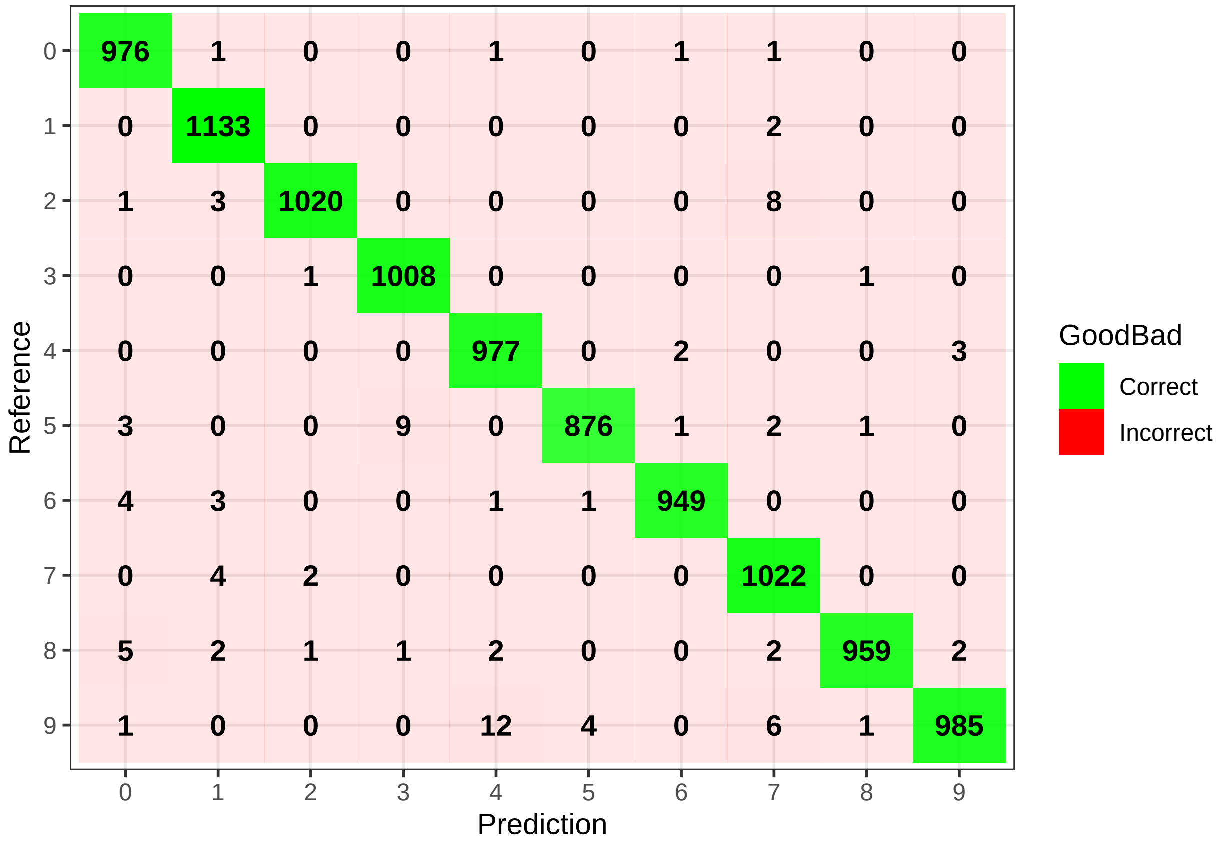


Confusion matrix

```r
data.frame(table(predicted_digits, mnist$test$y)) %>%
  setNames(c('Prediction', 'Reference', 'Freq')) %>%
  mutate(GoodBad = ifelse(Prediction == Reference, 'Correct', 'Incorrect')) -> conf_table
```

```
conf_table %>%
  ggplot(aes(y = Reference, x = Prediction, fill = GoodBad, alpha = Freq)) +
  geom_tile() +
  geom_text(aes(label = Freq), vjust = 0.5, fontface  = 'bold', alpha = 1) +
  scale_fill_manual(values = c(Correct = 'green', Incorrect = 'red')) +
  guides(alpha = FALSE) +
  theme_bw() +
  ylim(rev(levels(conf_table$Reference)))
```

```
## Warning: `guides(<scale> = FALSE)` is deprecated. Please use `guides(<scale> =
## "none")` instead.
```



Reference \ Prediction	0	1	2	3	4	5	6	7	8	9
0	976	1	0	0	1	0	1	1	0	0
1	0	1133	0	0	0	0	0	2	0	0
2	1	3	1020	0	0	0	0	8	0	0
3	0	0	1	1008	0	0	0	0	1	0
4	0	0	0	0	977	0	2	0	0	3
5	3	0	0	9	0	876	1	2	1	0
6	4	3	0	0	1	1	949	0	0	0
7	0	4	2	0	0	0	0	1022	0	0
8	5	2	1	1	2	0	0	2	959	2
9	1	0	0	0	12	4	0	6	1	985