# Deep Learning for Computer Vision

MNIST: Multiclass Image Classification of Handwritten Digits

*Alex Philip Berger*

*8/28/2019*

## 1. Introduction

Solving MNIST is known as the *'Hello World'* of Deep Learning.

In this notebook, I start with a *densely connected network*. Then, I set up a *convolutional network*, and compare performance of the two.

## 2. Data & The Problem

I load libraries, and get data from the Keras package.

```
library(tidyverse)
library(keras)
```

```
mnist <- dataset_mnist()
train_images <- mnist$train$x
train_labels <- mnist$train$y
test_images <- mnist$test$x
test_labels <- mnist$test$y
```

The task is to classify grayscale images of handwritten digits - presented in 28x28 pixes. From the above data import, I have 60000 pictures to train a model on, and 10000 to test on.

Here is a subset of how *one* picture (sample) looks, vectorized:

```
train_images[1, 1:8, 1:13]
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
## [1,]    0    0    0    0    0    0    0    0    0     0     0     0     0
## [2,]    0    0    0    0    0    0    0    0    0     0     0     0     0
## [3,]    0    0    0    0    0    0    0    0    0     0     0     0     0
## [4,]    0    0    0    0    0    0    0    0    0     0     0     0     0
## [5,]    0    0    0    0    0    0    0    0    0     0     0     0     0
## [6,]    0    0    0    0    0    0    0    0    0     0     0     0     3
## [7,]    0    0    0    0    0    0    0    0   30    36    94   154   170
## [8,]    0    0    0    0    0    0    0   49  238   253   253   253   253
```

The associated number (label):

```
train_labels[1]
```

```
## [1] 5
```

Similarly, all pictures represent a number between 0-9.

## 3. Model 1: Densely Connected Network

### Network Architechture

I choose two *densely connected* neural layers, where the last one has a *softmax* activation, meaning it will output 10 probability scores: A probabiliy of the given picture belonging to each of the 10 digit classes.

```
network_1 <- keras_model_sequential() %>%
  layer_dense(units = 512, activation = "relu", input_shape = c(28 * 28)) %>%
  layer_dense(units = 10, activation = "softmax")
```

I compile the network, which requires me to set choose optimizer, loss and metrics. For multiclass, single-label classification, *categorical_crossentropy* is a natural choice for *loss function*. The *loss function* is the feedback signal for learning the weight tensors, and the reduction happens via mini-batch stochastic gradient descent, and the exact rules are determined by the optimizer (rmsprop).

```
network_1 %>% compile(
  optimizer = "rmsprop",
  loss = "categorical_crossentropy",
  metrics = c("accuracy")
)
```

### Prepare Data

The chosen network expects a 2D tensor, so I need to reshape data. Further, I need to trainsform data from *int* to *double*, so it takes values on the [0, 1] interval and not on [0, 255]. This is basic preparation for any Deep Learning model.

```
train_images <- keras::array_reshape(train_images, c(60000, 28 * 28))
train_images <- train_images / 255
test_images <- keras::array_reshape(test_images, c(10000, 28 * 28))
test_images <- test_images / 255
```

I also transform labels to categorical variables (also a basic step in Deep Learning):

```
train_labels <- to_categorical(train_labels)
test_labels <- to_categorical(test_labels)
```

### Fit Model & Assess Performance

Now I can train the network:

```
network_1 %>% fit(train_images, train_labels, epochs = 5, batch_size = 128)
```

Finally, I can evaluate performance on test data:

```
(metrics_1 <- network_1 %>% evaluate(test_images, test_labels, verbose = 0))
```

```
## $loss
## [1] 0.07482542
##
## $acc
## [1] 0.9767
```

## 4. Model 2: Convolutional Network

*Convolutional Networks* are standard practice in image-classification problems.

Their fundamental difference to *dense layers* are that they learn *local patterns* rather than *global patterns*.

**Network Architechture**

My convolutional network will consist of (1) A Basic Convnet and (2) A Densely Connected Classifier Network. (1) works with 3D tensors, and (2) works with 1D, so in between I insert a layer to flatten from 3D to 1D.

```
network_2 <- keras_model_sequential() %>%
  # Basic Convnet:
  layer_conv_2d(filters = 32, kernel_size = c(3, 3), activation = "relu",
                input_shape = c(28, 28, 1)) %>%
  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)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3, 3), activation = "relu") %>%
  # Flatten (from 3D tensors to 1D)
  layer_flatten() %>%
  # Densely Connected Classifier Network:
  layer_dense(units = 64, activation = "relu") %>%
  layer_dense(units = 10, activation = "softmax")
```

The Basic Convnet is simply a stack of *layer_conv_2d* and *layer_max_pooling_2d*. For kernel size, 3*3 is a common choice: We look for patterns (in the picture) of that size, and, in the first layer we work with 32 such filters.

In the final layer, I once again apply a softmax activation with 10 units, to give out 10 probabilities.

The whole network looks like this:

```
network_2
```

```
## Model
## _________________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =========================================================================
## conv2d (Conv2D)              (None, 26, 26, 32)        320
## _________________________________________________________________________
## max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
## _________________________________________________________________________
## conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
## _________________________________________________________________________
## max_pooling2d_1 (MaxPooling2D) (None, 5, 5, 64)        0
## _________________________________________________________________________
## conv2d_2 (Conv2D)            (None, 3, 3, 64)          36928
## _________________________________________________________________________
## flatten (Flatten)            (None, 576)               0
## _________________________________________________________________________
## dense_2 (Dense)              (None, 64)                36928
## _________________________________________________________________________
## dense_3 (Dense)              (None, 10)                650
## =========================================================================
```

```
## Total params: 93,322
## Trainable params: 93,322
## Non-trainable params: 0
## _________________________________________________________________________
```

The compilation step is as before:

```
network_2 %>% compile(
  optimizer = "rmsprop",
  loss = "categorical_crossentropy",
  metrics = c("accuracy")
)
```

### Prepare Data

As we work with a Connvolutional Network, I once again need to reshape data (each sample should be a 3D tensor - meaning a 4D tensor in overall):

```
train_images <- keras::array_reshape(train_images, c(60000, 28, 28, 1))
test_images <- keras::array_reshape(test_images, c(10000, 28, 28, 1))
```

In *train_data*, we now have 60.000 images, of height = 28, width = 28, and depth = 1.

```
dim(train_images)
```

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

### Fit Model & Assess Performance

Fitting the network is similar to before (only batch_size is decreased):

```
network_2 %>% fit(train_images, train_labels, epochs = 5, batch_size = 64)
```

Now, applying the trained network to test data, I obtain the following results:

```
(metrics_2 <- network_2 %>% evaluate(test_images, test_labels))
```

```
## $loss
## [1] 0.03482176
##
## $acc
## [1] 0.9908
```

## 5. Conclusion

I have built a *densely connected network* and *convolutional network*, with accuracy 0.9767 and 0.9908, respectively.

Thus, I have decreased the error rate of the network by 61 %.