Klasyfikacja odręcznie pisanych cyfr za pomocą sieci neuronowych

Piotr Zawal

31 lipca 2019

#https://www.r-bloggers.com/exploring-handwritten-digit-classification-a-tidy-analysis-of-the-mnist-dataset/ Deep neural networks with ~99.8% accuracy #http://repository.supsi.ch/5145/1/IDSIA-04-12.pdf # https://www.kaggle.com/kobakhit/digital-recognizer-in-r !!!! #https://rrighart.github.io/Digits/ !!!! #https://www.kaggle.com/srlmayor/easy-neural-network-in-r-for-0-994 # http://varianceexplained.org/r/digit-eda/

Wstęp

Wczytanie potrzebnych bibliotek:

```
library(ggplot2) # rysowanie wykresów
library(readr) # czytanie plików CSV
library(dplyr) # porządkowanie danych
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(tidyr) # porządkowanie danych
library(purrr) # ???

# ustal ziarno dla powtarzalności wyników
set.seed(123)

# ustaw styl wykresów
theme_set(theme_light())
```

Dane są podzielone na zbiór treningowy i testowy i zapisane w plikach *.csv.

```
# wczytaj dane
train <-  read.csv("./train.csv")
test <- read.csv("./test.csv")
```

Zbiór treningowy zawiera 42 000 obserwacji, a testowy 28 000.

```
# rozmiar zbioru treningowego i testowego
dim(train)
```

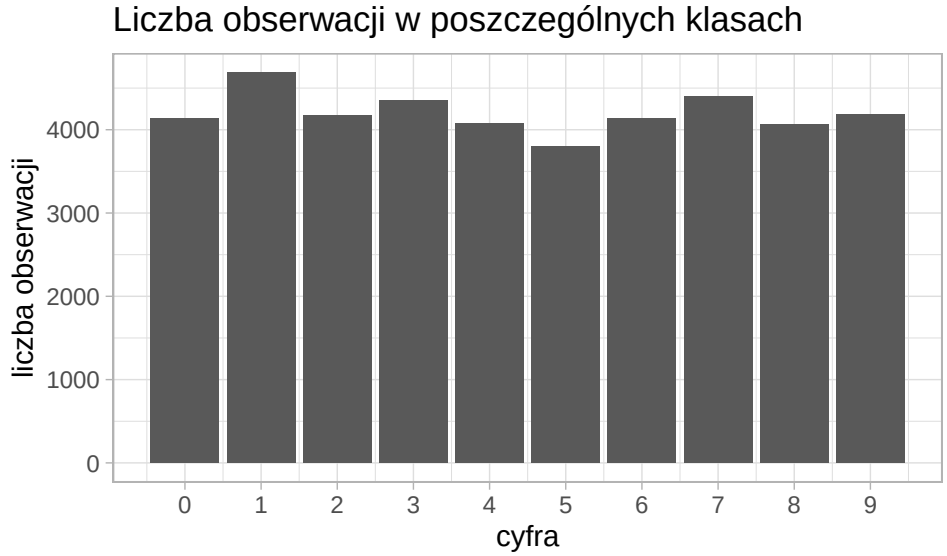
```
## [1] 42000   785
```

```
dim(test)
```

```
## [1] 28000   784
```

W zbiorze treningowym pojawia się kolumna *label*, zawierająca klasę każdej cyfry. Kolumna ta nie pojawia się w zbiorze testowym, który ma 784 kolumny. W kolumnach zapisano wartośc liczbową w zakresie [0, 255], odpowiadającą kolorowi piksela w skali szarości. Taka struktura danych jest jednowymiarową reprezentacją macierzy 28x28, której wykreślenie w dalszej części pracy pozwoli na wizualizację cyfr. Powodem takiej formy danych jest to, że jest ona już przygotowana do implementacji w algorytmach uczenia maszynowego.

```
# rozkład liczby obserwacji (cyfr) w zbiorze treningowym i testowym
ggplot(train, aes(x = label)) +
    geom_bar() +
    scale_x_continuous(breaks = 0:10) +
    ggtitle("Liczba obserwacji w poszczególnych klasach") +
    xlab("cyfra") +
    ylab("liczba obserwacji")
```



Analiza eksploracyjna

Wymiary macierzy: $\sqrt{784} = 28$, zatem wymiary obrazka to 28×28 pikseli. Tak wygląda losowo wybrana cyfra po transformacji z wektora do macierzy 28x28.

```
matrix(train[15, -1], nrow = 28, ncol = 28, byrow = TRUE) %>%
    write.table(row.names = FALSE, col.names = FALSE, sep = "  ")
```

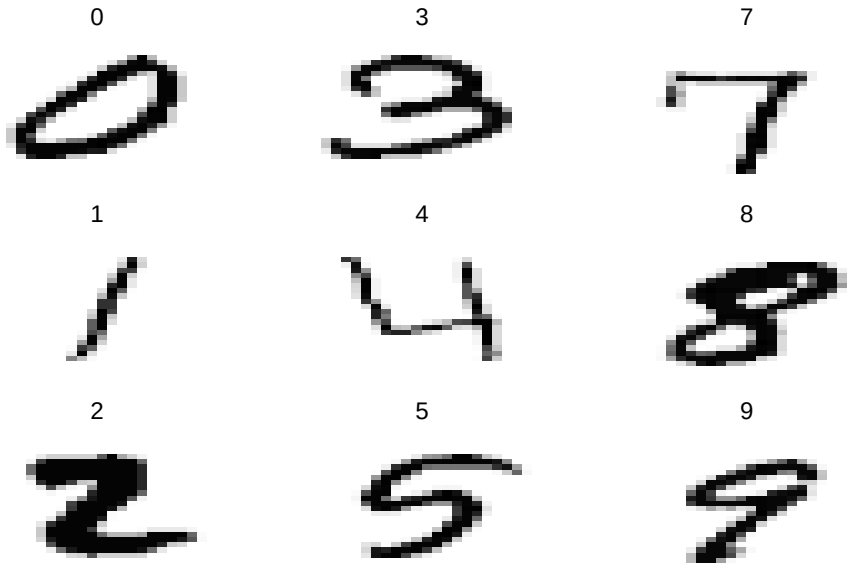
```
## 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
## 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
## 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
## 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
## 0  0  0  0  0  0  0  0  0  0  101  222  253  253  192  113  88  0  0  0  0  0  0  0  0  0  0  0
## 0  0  0  0  0  0  0  10  85  226  249  252  252  252  253  252  246  209  38  0  0  0  0  0  0  0  0  0
## 0  0  0  0  0  0  13  156  252  253  233  195  195  195  196  214  252  252  221  32  0  0  0  0  0
## 0  0  0  0  0  0  57  252  252  162  56  0  0  0  0  28  121  252  252  216  18  0  0  0  0  0  0  0  0
## 0  0  0  0  0  0  57  252  173  0  0  0  0  0  0  0  25  205  252  253  27  0  0  0  0  0  0  0
## 0  0  0  0  0  0  57  253  253  0  0  0  0  0  0  0  0  92  253  255  27  0  0  0  0  0  0  0
```

```
## 0  0  0  0  0  0  38  224  252  126  0  0  0  0  0  0  51  243  252  253  27  0  0  0  0  0  0  0
## 0  0  0  0  0  0  0  53  195  110  0  0  0  0  0  51  101  252  252  190  12  0  0  0  0  0  0  0
## 0  0  0  0  0  0  0  0  0  0  0  7  29  29  92  243  252  252  252  253  177  53  0  0  0  0  0  0
## 0  0  0  0  0  0  0  0  0  0  126  165  252  252  253  252  252  252  252  253  252  195  0  0  0  0
## 0  0  0  0  0  0  0  0  0  0  226  253  253  253  255  215  140  140  140  192  253  253  146  0  0
## 0  0  0  0  0  0  0  0  0  0  178  252  242  167  106  18  0  0  0  12  228  252  223  0  0  0  0  0
## 0  0  0  0  0  0  0  0  0  0  19  55  49  0  0  0  0  0  0  0  225  252  223  0  0  0  0  0
## 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  16  92  243  252  129  0  0  0  0  0
## 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  16  203  253  252  220  37  0  0  0  0  0
## 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  26  207  253  255  215  31  0  0  0  0  0  0
## 0  0  0  0  101  225  175  0  0  0  0  0  10  85  147  225  231  252  252  168  33  0  0  0  0  0  0
## 0  0  0  0  113  252  208  57  57  57  57  166  203  252  253  252  239  195  118  0  0  0  0  0  0
## 0  0  0  0  38  234  252  252  252  253  252  252  252  252  225  176  65  0  0  0  0  0  0  0  0  0
## 0  0  0  0  0  100  221  252  252  253  127  112  112  112  0  0  0  0  0  0  0  0  0  0  0  0  0  0
## 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
## 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
## 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
## 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

W celu wykonania lepszej jakości obrazów przekształcimy dane do postaci dwuwymiarowej.

```
train_2dim <- train %>%
                head(1000) %>%
                # dodaj dodatkowy rząd z numerem wiersza
                mutate(instance = row_number()) %>%
                # zmień format tabeli (wide data -> long data)
                gather(pixel, value, -label, -instance) %>%
                # ekstrakcja numeru piksela z kolumny "pixel" za pomocą wyrażeń regularnych
                tidyr::extract(pixel, "pixel", "(\\d+)", convert = TRUE) %>%
                # dodaj kolumnę z koordynatami piksela
                mutate(x = pixel %% 28,
                       y = 28 - pixel %/% 28)
```

```
train_2dim %>%
    filter(instance <= 18) %>%
    ggplot(aes(x, y, fill = value)) +
    geom_tile(show.legend = FALSE) +
    facet_wrap(facets = ~ label,
               dir = "v") +
    scale_fill_gradient2(low = "white",
                         high = "black",
                         mid = "gray",
                         midpoint = 127.5) +
    xlab("") +
    ylab("") +
    theme_void()
```



Obrazowanie danych w postaci dwuwymiarowej będzie wykorzystywane kilkukrotnie podczas analizy zbioru, dlatego zdefiniowano funkcję `{r}printDigits()`. Argumentami funkcji jest cyfra, która będzie wykreślona (digit) oraz liczba paneli (domyślna wartość argumentu numOfDigits to 16).

```
printDigits <- function(digit, numOfDigits = 16) {
    if (!require("gridExtra")) install.packages("gridExtra")

    plots <- list()

    for (i in 1:numOfDigits) {

        digits_data <- train[train$label == digit, ] %>% sample_n(1)

        plots [[i]] <- digits_data %>%
            gather() %>%
            filter( key != "label") %>%
            mutate(row = row_number() - 1) %>%
            mutate(col = row %% 28, row = row %/% 28) %>%
            ggplot() +
            geom_tile(aes(col, 28 - row, fill = value), show.legend = FALSE) +
            scale_fill_gradient(low = "white", high = "black") +
            coord_equal() +
            theme_void() +
            theme(plot.background = element_rect(fill = "gray80"))
    }

do.call("grid.arrange", c(plots, ncol = 4, nrow = ceiling(numOfDigits / 4)))
}
```

```
printDigits <- function(dataset, digit, numOfDigits = 16, plotRandom = TRUE) {
    if (!require("gridExtra")) install.packages("gridExtra")

    plots <- list()

    for (i in 1:numOfDigits) {
```

```
        if (plotRandom) {
            digits_data <- dataset[dataset$label == digit, ] %>% sample_n(1)
        } else {

            digits_data <- dataset[dataset$label == digit, ][i, ]
        }


        plots [[i]] <- digits_data %>%
            gather() %>%
            filter( key != "label") %>%
            mutate(row = row_number() - 1) %>%
            mutate(col = row %% 28, row = row %/% 28) %>%
            ggplot() +
            geom_tile(aes(col, 28 - row, fill = value), show.legend = FALSE) +
            scale_fill_gradient(low = "white", high = "black") +
            coord_equal() +
            theme_void() +
            theme(plot.background = element_rect(fill = "gray80"))
    }

do.call("grid.arrange", c(plots, ncol = 4, nrow = ceiling(numOfDigits / 4)))
}
```

Z pomocą tej funkcji łatwo będzie można wykreślić losowo wybrane cyfry, np:

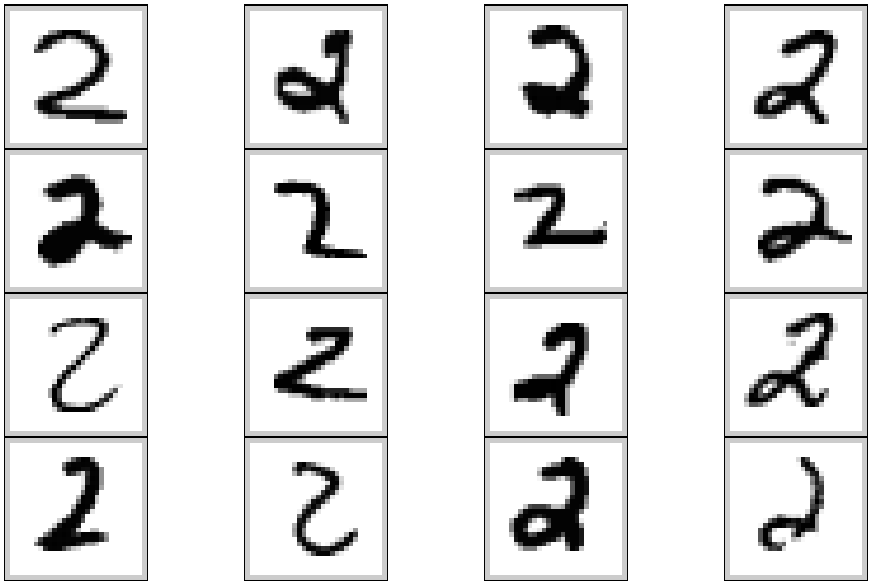
```
printDigits(train, 2)
```

```
## Loading required package: gridExtra
```

```
## Warning: package 'gridExtra' was built under R version 3.6.2
```

```
##
## Attaching package: 'gridExtra'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

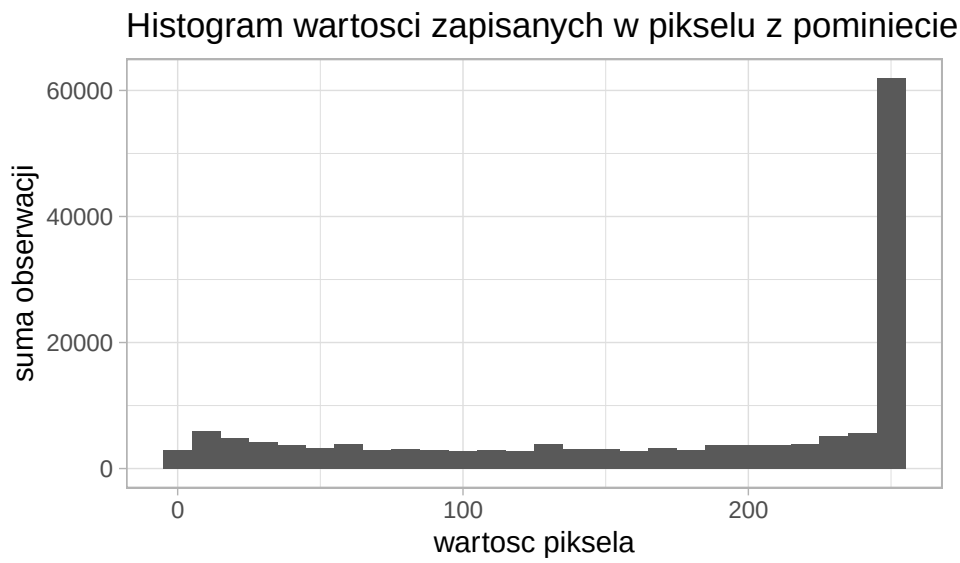


Data exploration

```
ggplot(train_2dim, aes(value)) +
    geom_histogram(binwidth = 10) +
    xlab("wartość piksela") +
    ylab("suma obserwacji") +
    ggtitle("Histogram wartości zapisanych w pikselu")
```

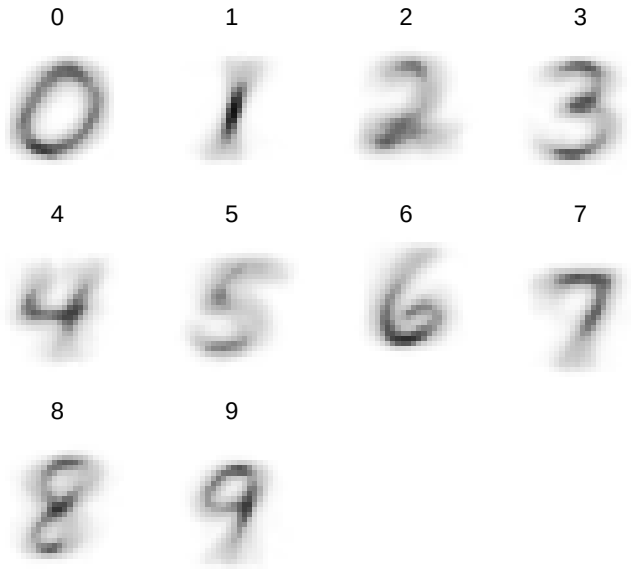
Histogram z wyłączeniem - wynika z niego, że dominującą wartością jest 255 (kolor czarny), cyfry powinny mieć więc dobrze zdefiniowane kontury.

```
train_2dim %>%
    filter(value != 0) %>%
    ggplot(aes(value)) +
    geom_histogram(binwidth = 10) +
    xlab("wartość piksela") +
    ylab("suma obserwacji") +
    ggtitle("Histogram wartości zapisanych w pikselu z pominięciem wartości 0")
```



Liczymy jak wygląda "średnia" cyfra:

```
# oblicz jak wygląda uśredniona cyfra
average_digit <- train_2dim %>%
    group_by(x, y, label) %>%
    summarize(mean_pixel_value = mean(value)) %>%
    ungroup()

# average_digit

average_digit %>%
    ggplot(aes(x, y, fill = mean_pixel_value)) +
        geom_tile(show.legend = FALSE) +
        facet_wrap(facets = ~ label) +
        scale_fill_gradient2(low = "white",
                             high = "black",
                             mid = "gray",
                             midpoint = 127.5) +
        xlab("") +
        ylab("") +
        coord_equal() +
        theme_void()
```



Wszystkie uśrednione cyfry są czytelne i mogą bez trudu być rozróżnione przez człowieka. W zbiorze na pewno znajdują się jednak cyfry, których kształt odbiega od uśrednionego kształtu. Aby ocenić skalę zjawiska sprawdzono, w której klasie jest największa wariancja. Jako miarę różnicy pomiędzy średnią i wybraną obserwacją, wybrano średnią odległość euklidesową dla każdej cyfry:

$$d_{Euklides} = \sqrt{\frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{n}}$$

gdzie: y_i to wartość i-tego piksela, $\hat{y}_i$ to wartość piksela uśredniona po wszystkich obserwacjach w danej klasie a n to liczba wszystkich obserwacji.

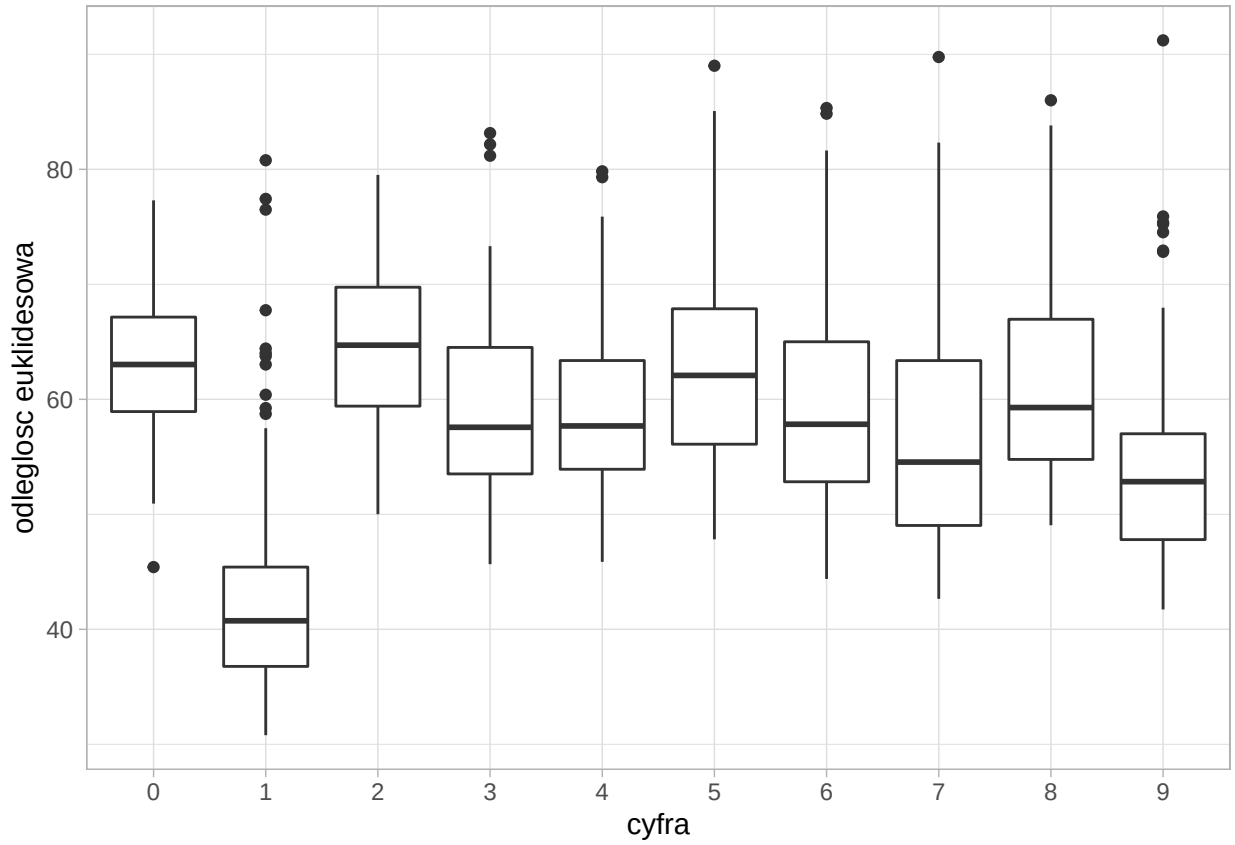
```
digits_joined <- inner_join(train_2dim, average_digit, by = c("label", "x", "y"))

digit_distances <- digits_joined %>%
    group_by(label, instance) %>%
    summarize(euclidean_distance = sqrt(mean((value - mean_pixel_value) ^ 2)))

digit_distances %>%
    arrange(desc(euclidean_distance)) %>%
    head(20)
```

```
## # A tibble: 20 x 3
## # Groups:   label [7]
##    label instance euclidean_distance
##    <int>    <int>              <dbl>
##  1     9      500               91.2
##  2     7      680               89.8
##  3     5      444               89.0
##  4     8      682               86.0
##  5     6      180               85.3
##  6     5       20               85.1
##  7     6      923               84.8
##  8     8       68               83.8
##  9     3      241               83.2
## 10     7      133               82.3
## 11     8      177               82.3
## 12     3      904               82.2
```

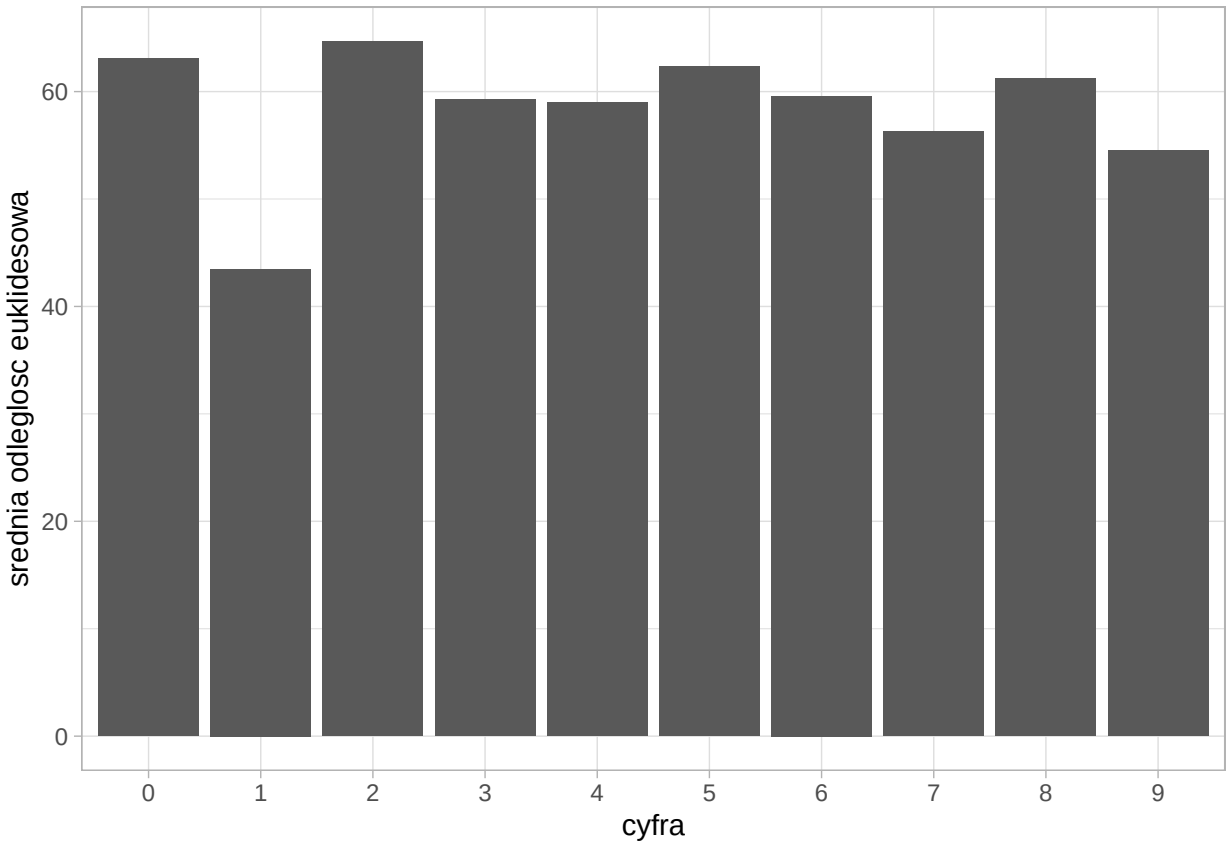
```
## 13      6       757             81.6
## 14      8       863             81.4
## 15      3       716             81.2
## 16      6       991             81.1
## 17      1       898             80.8
## 18      8       198             80.6
## 19      6       501             80.3
## 20      5       916             80.2
```

W dwudziestu pierwszych obserwacjach odległość euklidesowa ma wartość wyższą niż 80.

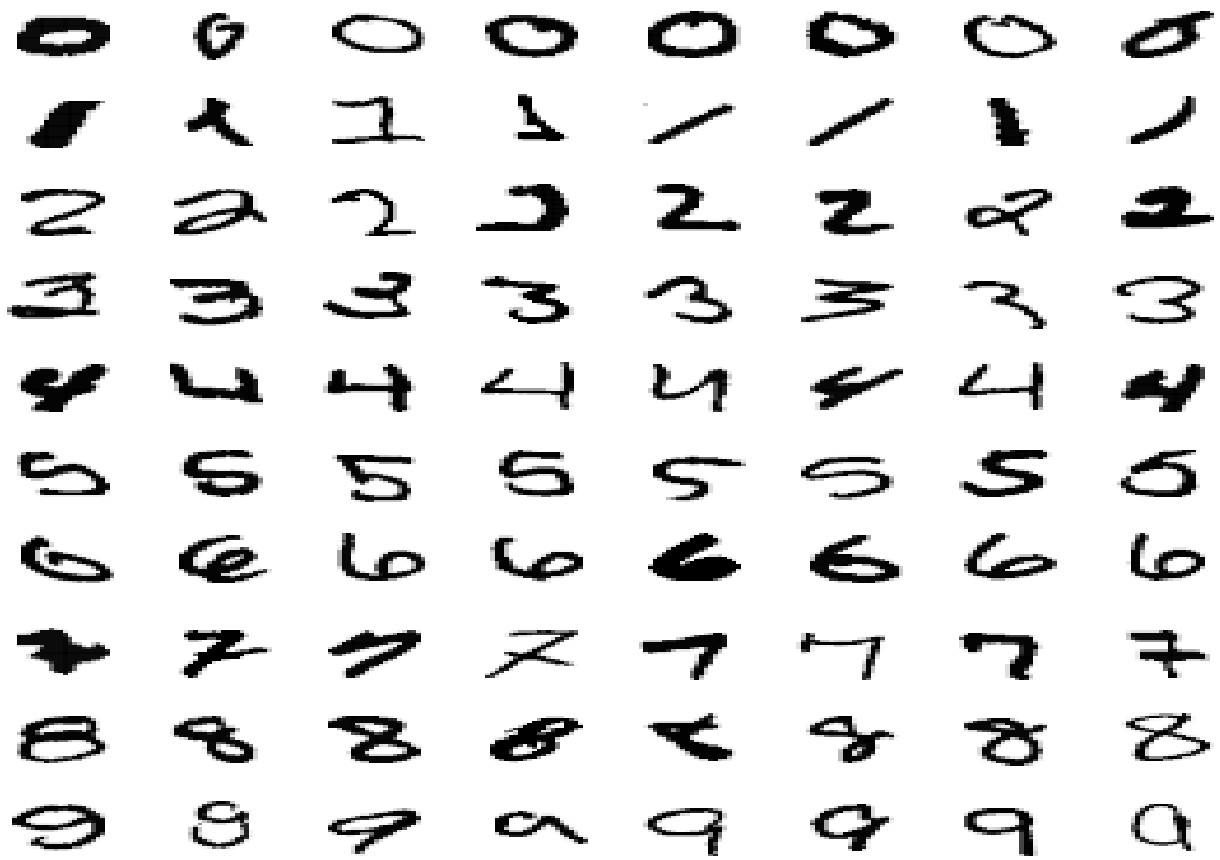
```
ggplot(digit_distances, aes(x = factor(label), y = euclidean_distance)) +
    geom_boxplot() +
    xlab("cyfra") +
    ylab("odległość euklidesowa")
```




```
digit_distances %>%
    group_by(label) %>%
    summarise(mean_distance = mean(euclidean_distance)) %>%
    ggplot(aes(x = as.factor(label), y = mean_distance)) +
    geom_bar(stat = "identity") +
    ylab("średnia odległość euklidesowa") +
    xlab("cyfra")
```




```
highest_distances <- as.tbl(digit_distances) %>%
    # dla każdej klasy wybierz 8 obserwacji z największymi odległościami euklidesowymi
    top_n(euclidean_distance, n = 8) %>%
    # dodaj kolumnę szeregującą obserwacje wg. odległości euklidesowej
    mutate(number = rank(-euclidean_distance))


inner_join(train_2dim, highest_distances, by = c("label", "instance")) %>%
    ggplot(aes(x, y, fill = value)) +
    geom_tile(show.legend = FALSE) +
    scale_fill_gradient2(low = "white",
                         high = "black",
                         mid = "gray",
                         midpoint = 127.5) +
    facet_grid(label ~ number) +
    theme_void() +
    theme(strip.text = element_blank())
```



Przygotowanie danych do modelowania

https://www.youtube.com/watch?v=5bso_5X7Zu4

```
train_y <- train$label
train_y <- as.factor(train_y)
train_x <- train[, -1]

dim(test)
```

```
## [1] 28000   784
```

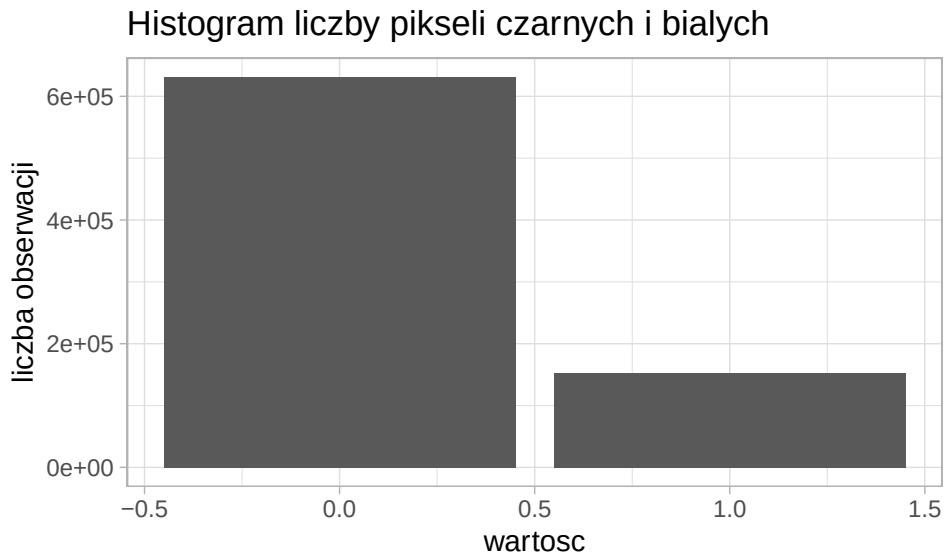
```
dim(train_x)
```

```
## [1] 42000   784
```

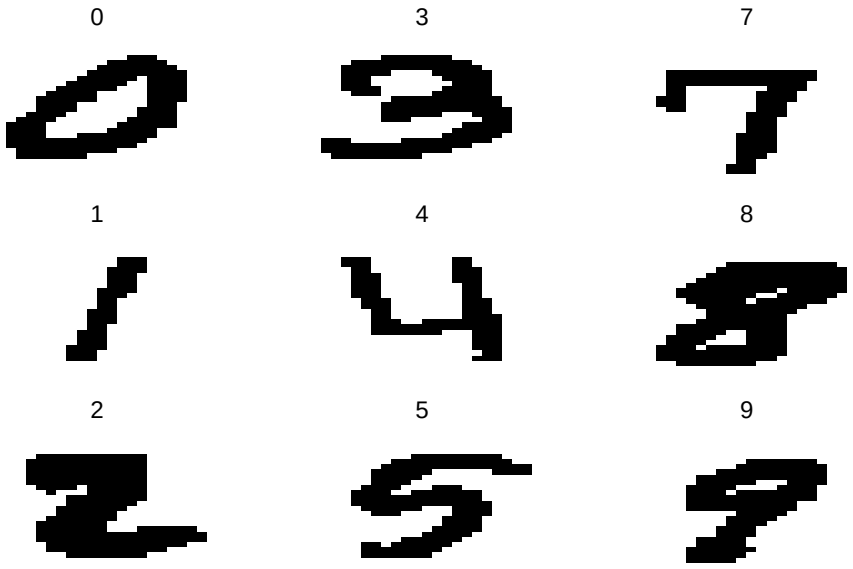
```
# standaryzowanie danych
train_x <- ceiling(train_x / 255)
test_x <- ceiling(test / 255)
```

W celu łatwiejszej wizualizacji ustandaryzowanych danych posłużono się zmienną `{r} train_2dim`, wobec której zastosowano taką samą technikę standaryzacji wartości pikseli.

```
ggplot(train_2dim, aes(ceiling(value / 255))) +
    geom_bar() +
    xlab("wartość") +
    ylab("liczba obserwacji") +
    ggtitle("Histogram liczby pikseli czarnych i białych")
```




```
train_2dim %>%
    filter(instance <= 18) %>%
    ggplot(aes(x, y, fill = ceiling(value / 255))) +
    geom_tile(show.legend = FALSE) +
    facet_wrap(facets = ~ label,
               dir = "v") +
    scale_fill_gradient(low = "white", high = "black") +
    xlab("") +
    ylab("") +
    theme_void()
```




```
# wczytaj bibliotekę do głębokich sieci neuronowych
library(keras)

# przygotowanie zbioru treningowego i testowego
train_images <- train[, -1]
train_labels <- train$label
```

```
test_images <- test

train_images <- train_images / 255

test_images <- test_images / 255
```

Keras, jako danych wejściowych, wymaga macierzy. Obecny typ danych to lista:

```
typeof(train_images)
```

```
## [1] "list"
```

```
typeof(test_images)
```

```
## [1] "list"
```

Zbiory przekształcono w typ macierzy:

```
# keras wymaga macierzy jako typu danych wejściowych
train_images <- as.matrix(train_images)
test_image <- as.matrix(test_images)
```

Dodatkowo należy zmienić typ danych zawierających klasy (cyfry). W przeciwnym wypadku byłyby traktowane jako wartości liczbowe, co prowadziłoby do nieprawidłowego wytrenowania modelu.

```
train_labels <- to_categorical(train_labels)
str(train_labels)
```

```
##  num [1:42000, 1:10] 0 1 0 0 1 1 0 0 0 0 ...
```

```
head(train_labels, 10)
```

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

Definicja funkcji, która obliczone predykcje wpisuje do pliku, który następnie zostaje wysłany do Kaggle celem sprawdzenia poprawności predykcji.

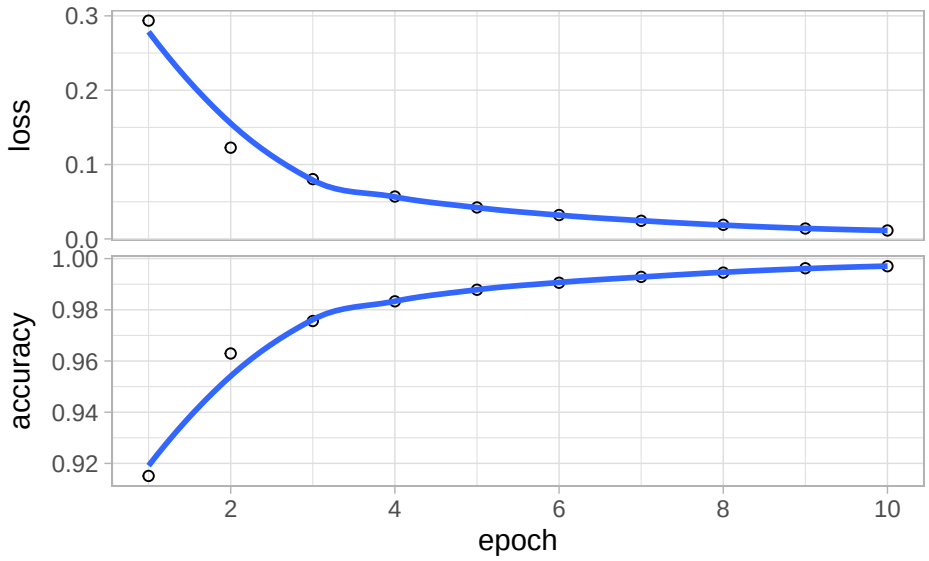
```
submitKaggle <- function(network) {
    predictions <- network %>%
                    predict_classes(as.matrix(test_images))
    submit_predictions <-  read.csv("./sample_submission.csv")
    submit_predictions$Label <- predictions
    write.csv(submit_predictions, "./sample_submission_keras.csv", row.names = FALSE)
}
```

Definicja modelu:

```
# keras - model 1
network <- keras_model_sequential() %>%
```

```
  layer_dense(units = 512, activation = "relu", input_shape = c(784)) %>%
  layer_dense(units = 10, activation = "softmax")

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

model_1 <- network %>% fit(train_images,
                           train_labels,
                           epochs = 10,
                           batch_size = 128)

plot(model_1)
```

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



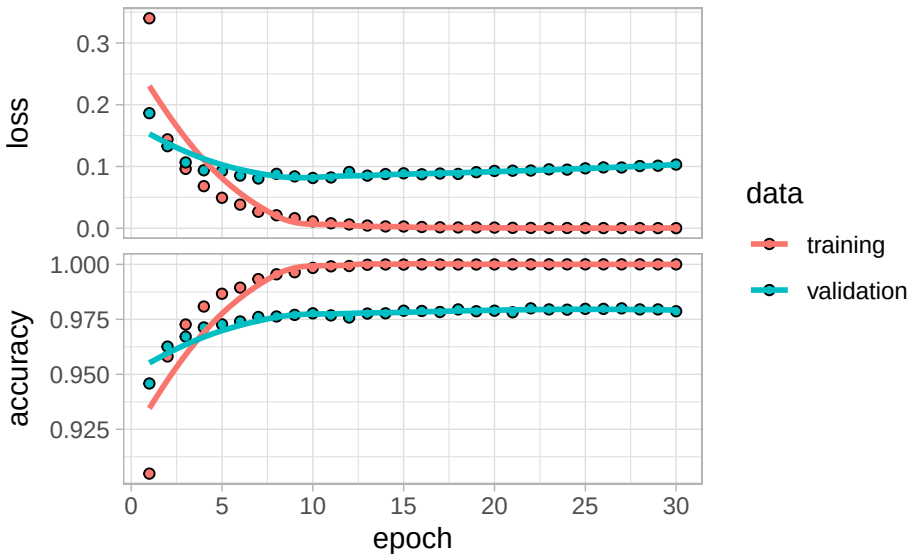

```
submitKaggle(network)


# Kaggle 30 epochs 0.97885
# Kaggle 10 epochs: 0.97857
```

Otrzymaliśmy bardzo zadowalający wynik rzędu 97.89%. Pomimo dobrej dokładności, dalsza optymalizacja tego modelu jest utrudniona ze względu na to, że brak tu jakiejkolwiek kontroli na zbiorze treningowym. Przekłada się to na brak monitorowania i kontroli nad zjawiskiem overfittingu, a jedyną możliwością kontroli rzeczywistej dokładności modelu jest wysłanie wyników do Kaggle (limit 5 dziennie). Aby zoptymalizować model, podczas procedury treningu wydzielono losowo wybrane obserwacje stanowiące 20% zbioru treningowego. Z drugiej strony, spowoduje to najprawdopodobniej spadek dokładności modelu: zamiast 42 000 obserwacji do trenowania modelu wykorzystanych zostanie 33 600. Po zoptymalizowaniu architektury sieci, trenowanie modelu będzie przeprowadzone na pełnym zbiorze treningowym.

```
# keras - model 2
network2 <- keras_model_sequential() %>%
  layer_dense(units = 512, activation = "relu", input_shape = c(784)) %>%
  layer_dense(units = 10, activation = "softmax")
```

```
network2 %>% compile(
  optimizer = "adam",
  loss = "categorical_crossentropy",
  metrics = c("accuracy"))

model_2 <- network2 %>% fit(train_images,
                            train_labels,
                            epochs = 30,
                            batch_size = 128,
                            validation_split = 0.2)

plot(model_2)
```

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




```
# Optimizer: rmsprop
# units = 256 loss: 1.7884e-04 - accuracy: 1.0000 - val_loss: 0.1498 - val_accuracy: 0.9763
# units = 512 loss: 4.0712e-04 - accuracy: 0.9999 - val_loss: 0.1437 - val_accuracy: 0.9792
# units = 784 loss: 1.0470e-04 - accuracy: 1.0000 - val_loss: 0.1496 - val_accuracy: 0.9808

# Optimizer: adam
# units = 256 loss: 4.7305e-04 - accuracy: 1.0000 - val_loss: 0.1009 - val_accuracy: 0.9770
# units = 512 loss: 2.8389e-04 - accuracy: 1.0000 - val_loss: 0.0938 - val_accuracy: 0.9802
# units = 784 loss: 1.1348e-04 - accuracy: 1.0000 - val_loss: 0.0984 - val_accuracy: 0.9814

# Batch sizes (for 512 units):
# batch_size = 512: loss: 0.0033 - accuracy: 1.0000 - val_loss: 0.0866 - val_accuracy: 0.9760
# batch_size = 256: loss: 8.8377e-04 - accuracy: 1.0000 - val_loss: 0.0871 - val_accuracy: 0.9790
# batch_size = 128: loss: 2.8389e-04 - accuracy: 1.0000 - val_loss: 0.0938 - val_accuracy: 0.9802
# batch_size = 32:  loss: 0.0041 - accuracy: 0.9986 - val_loss: 0.1683 - val_accuracy: 0.9754
```

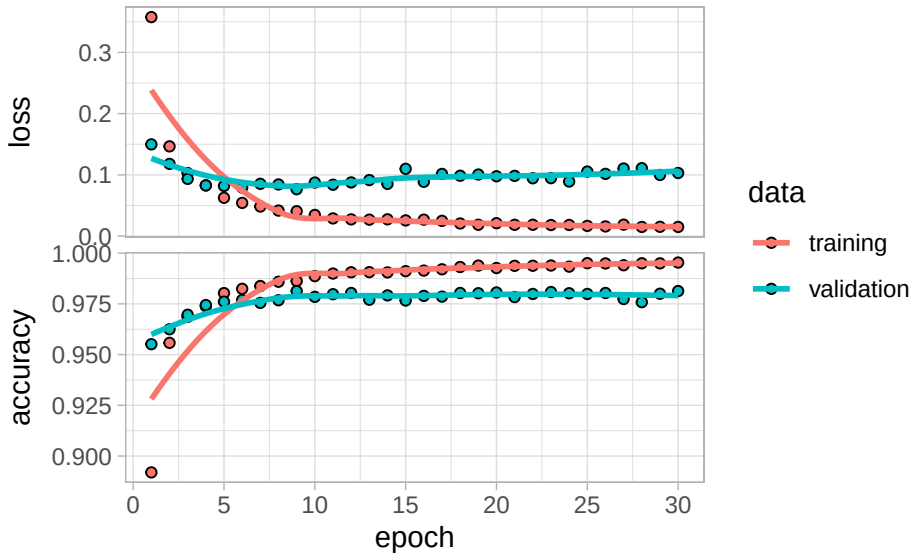
Widać tutaj, że parametr val_loss wzrasta wraz z kolejnymi iteracjami. Jest to skutek overfittingu, tj. model "zapamiętuje" dane i optymalizuje się pod kątem zbioru treningowego zamiast coraz lepiej klasyfikować obserwacje w zbiorze testowym.

Zjawisko to jest jeszcze lepiej widoczne po dodaniu kolejnej warstwy:

```
knitr::opts_chunk$set(cache = TRUE)
# model 3 - spróbujemy dodać więcej warstw
network3 <- keras_model_sequential() %>%
  layer_dense(units = 512, activation = "relu", input_shape = c(784)) %>%
  layer_dense(units = 512, activation = "relu", input_shape = c(784)) %>%
  layer_dense(units = 10, activation = "softmax")

network3 %>% compile(
  optimizer = "adam",
  loss = "categorical_crossentropy",
  metrics = c("accuracy"))

model_3 <- network3 %>% fit(train_images,
                            train_labels,
                            epochs = 30,
                            batch_size = 128,
                            validation_split = 0.2)

# loss: 0.0082 - accuracy: 0.9978 - val_loss: 0.1524 - val_accuracy: 0.9768
```

```
knitr::opts_chunk$set(cache = TRUE)
# model 4 - dodanie dropoutów
network4 <- keras_model_sequential() %>%
    layer_dense(units = 512, activation = "relu", input_shape = c(784)) %>%
    layer_dropout(rate = 0.3) %>%
    layer_dense(units = 512, activation = "relu", input_shape = c(784)) %>%
    layer_dropout(rate = 0.3) %>%
    layer_dense(units = 10, activation = "softmax")

network4 %>% compile(
  optimizer = "adam",
  loss = "categorical_crossentropy",
  metrics = c("accuracy"))


#load_model_weights_hdf5(network4, "./network4.h5")
model_4 <- network4 %>% fit(train_images,
                            train_labels,
                            epochs = 30,
                            batch_size = 128,
                            validation_split = 0.2)
plot(model_4)
```

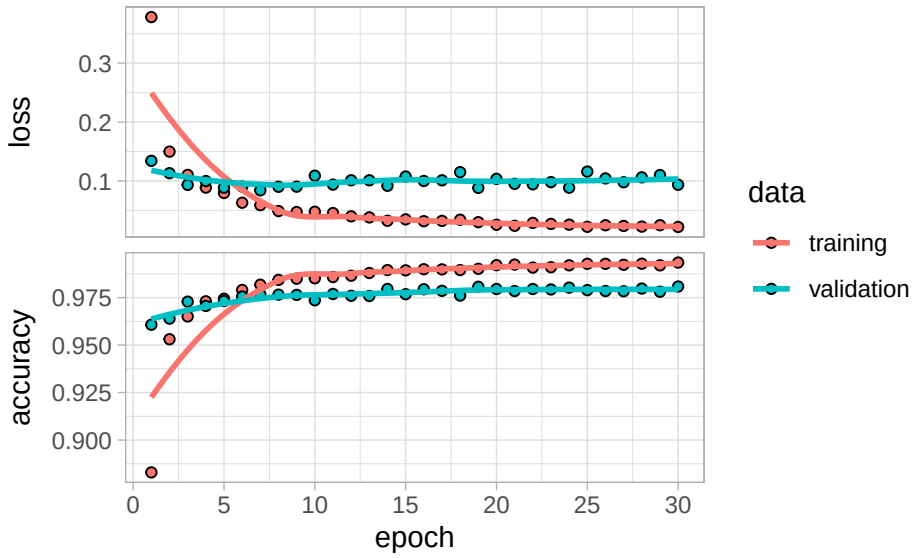
```
## `geom_smooth()` using formula 'y ~ x'
```




```
# dropout rate = 0.7, 0.5   loss: 0.0801 - accuracy: 0.9750 - val_loss: 0.0764 - val_accuracy: 0.9774
# dropout rate = 0.5, 0.5   loss: 0.0742 - accuracy: 0.9753 - val_loss: 0.0791 - val_accuracy: 0.9775
# dropout rate = 0.45, 0.45 loss: 0.0294 - accuracy: 0.9902 - val_loss: 0.0976 - val_accuracy: 0.9808
# dropout rate = 0.3, 0.3   loss: 0.0156 - accuracy: 0.9949 - val_loss: 0.0948 - val_accuracy: 0.9820 0
# dropout rate = 0.2, 0.2   loss: 0.0126 - accuracy: 0.9961 - val_loss: 0.1122 - val_accuracy: 0.9802
# dropout rate = 0.3, 0.2   loss: 0.0133 - accuracy: 0.9953 - val_loss: 0.0963 - val_accuracy: 0.9811
```

```
# model 4 - dodanie jeszcze jednej warstwy
network5 <- keras_model_sequential() %>%
    layer_dense(units = 512, activation = "relu", input_shape = c(784)) %>%
    layer_dropout(rate = 0.3) %>%
    layer_dense(units = 512, activation = "relu", input_shape = c(784)) %>%
    layer_dropout(rate = 0.3) %>%
    layer_dense(units = 512, activation = "relu", input_shape = c(784)) %>%
    layer_dropout(rate = 0.3) %>%
    layer_dense(units = 10, activation = "softmax")

network5 %>% compile(
  optimizer = "adam",
  loss = "categorical_crossentropy",
  metrics = c("accuracy"))


#load_model_weights_hdf5(network4, "./network4.h5")
model_5 <- network5 %>% fit(train_images,
                            train_labels,
                            epochs = 30,
                            batch_size = 128,
                            validation_split = 0.2)
plot(model_5)
```

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




```
# hidden layers = 2   loss: 0.0156 - accuracy: 0.9949 - val_loss: 0.0948 - val_accuracy: 0.9820 OVRF
# hidden layers = 3   loss: 0.0231 - accuracy: 0.9929 - val_loss: 0.1167 - val_accuracy: 0.9768
```

Zabawa z learning rate:

```
learning_rates = c(0.01, 0.005, 0.001, 0.0005, 0.0001)
accuracy_vector = c()
loss_vector = c()
val_accuracy_vector = c()
val_loss_vector = c()

for (rate in learning_rates) {

    network_lr <- keras_model_sequential() %>%
    layer_dense(units = 512, activation = "relu", input_shape = c(784)) %>%
    layer_dropout(rate = 0.3) %>%
    layer_dense(units = 512, activation = "relu", input_shape = c(784)) %>%
    layer_dropout(rate = 0.3) %>%
    layer_dense(units = 10, activation = "softmax")

    network_lr %>% compile(optimizer = "adam",
                           loss = "categorical_crossentropy",
                           metrics = c("accuracy"))

    model_lr <- network_lr %>% fit(train_images,
                            train_labels,
                            epochs = 30,
                            batch_size = 128,
                            validation_split = 0.2,
                            learning_rates = rate)
    plot(model_lr)

    accuracy_vector <- append(accuracy_vector,
                              tail(model_lr$metrics$accuracy, n = 1))
    loss_vector <- append(loss_vector,
                          tail(model_lr$metrics$loss, n = 1))
```

```
    val_accuracy_vector <- append(val_accuracy_vector,
                                  tail(model_lr$metrics$val_accuracy, n = 1))
    val_loss_vector <- append(val_loss_vector,
                              tail(model_lr$metrics$val_loss, n = 1))

}

# hidden layers = 2   loss: 0.0156 - accuracy: 0.9949 - val_loss: 0.0948 - val_accuracy: 0.9820 OVRF
# hidden layers = 3   loss: 0.0231 - accuracy: 0.9929 - val_loss: 0.1167 - val_accuracy: 0.9768
```

```
learning_rates_df <- rbind(accuracy_vector, loss_vector, val_accuracy_vector, val_loss_vector) %>%
                       data.frame()
colnames(learning_rates_df) <- learning_rates

print(learning_rates_df)
```

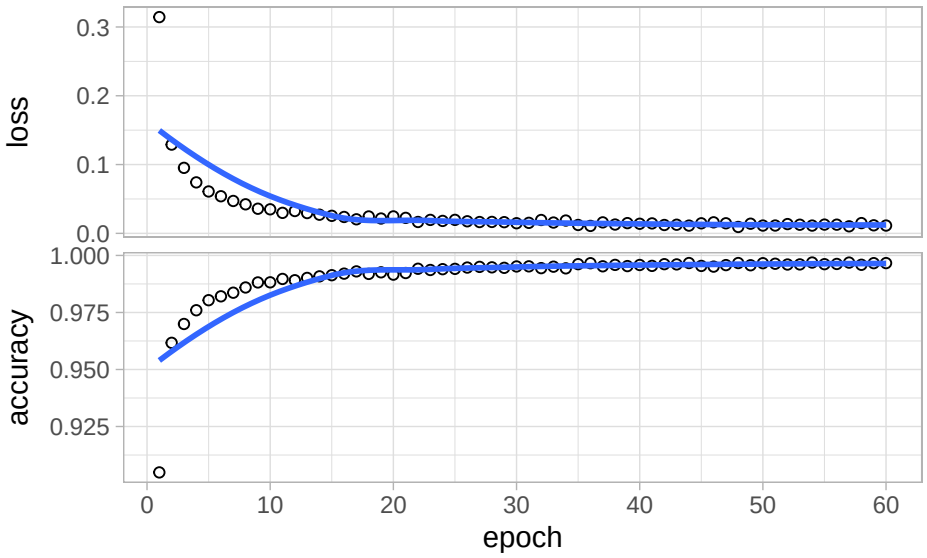
```
##                          0.01      0.005      0.001      5e-04      1e-04
## accuracy_vector     0.9949405 0.99517858 0.99502975 0.99407738 0.99458331
## loss_vector         0.0161085 0.01450321 0.01602013 0.01793178 0.01663357
## val_accuracy_vector 0.9777381 0.97904760 0.97988093 0.97964287 0.97654760
## val_loss_vector     0.1095799 0.11064669 0.11003859 0.11632969 0.12197081
```

Na podstawie otrzymanych wyników widać, że najlepsze metryki otrzymano dla domyślnej wartości parametru learning_rate wynoszącej 0.001.

Trening zoptymalizowanej sieci na całym (42 00 obserwacji) zbiorze treningowym:

```
mlp_network_final <- keras_model_sequential() %>%
    layer_dense(units = 512, activation = "relu", input_shape = c(784)) %>%
    layer_dropout(rate = 0.3) %>%
    layer_dense(units = 512, activation = "relu", input_shape = c(784)) %>%
    layer_dropout(rate = 0.3) %>%
    layer_dense(units = 10, activation = "softmax")

mlp_network_final %>% compile(
  optimizer = "adam",
  loss = "categorical_crossentropy",
  metrics = c("accuracy"))


#load_model_weights_hdf5(network4, "./network4.h5")
model_final <- mlp_network_final %>% fit(train_images,
                                         train_labels,
                                         epochs = 60,
                                         batch_size = 128,
                                         learning_rate = 0.001)
plot(model_final)
```

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




```
submitKaggle(mlp_network_final)

# Kaggle submission results:
# epochs = 30  0.98028
# epochs = 60  0.98128
# epochs = 100 0.98057

mlp_network_final_accuracy <- tail(model_final$metrics$accuracy, n = 1)
mlp_network_final_loss <- tail(model_final$metrics$loss, n = 1)
```

```
## accuracy:  0.9966428
```

```
## loss:  0.01133421
```

```
# # data_frame z prawdziwą i przewidzianą klasą
# model_1
# model_2
# pred_class <- predict_classes(network,
#                               train,
#                               batch_size = 32,
#                               verbose = 1)
#
# train_df <- tibble(train_images = apply(train_images, 1, which.max) - 1, pred_class)
# table(train_df)
```

https://www.kaggle.com/timokerremans/cnn-for-minst-dataset

```
#train_df %>%
#     count(train_images, pred_class) %>%
#     ungroup() %>%
#     filter(train_images != pred_class) %>%
#     ggplot() +
#         geom_tile(aes(as.factor(train_images), as.factor(pred_class), fill = n))
```