HarvardX - PH125.9x - DataScience: Capstone - ImageProcessing

Jan Brettschneider

15th April 2020

1. Introduction

This document summarizes the chosen project "Image Processing" for the Capstone Exam in course of the HardvardX Data Science Certificate.

The dataset used is the **Fashion-MNIST** dataset published by the company Zalando. It contains 28x28 pixel grayscale images of the articles which are associated with 10 classes. It is commonly used for benchmarking machine learning algorithms. Due to the fact that we already used the classical MNIST dataset with numbers, I'm really interested in the processing of pictures and this dataset is already tidy and proven, I decided to give it a go.

This project again will have the classical approach of machine learning projects, although I will put the focus on the machine learning part of things. The steps performed are as follows:

1. Download the raw data and import the files into R Studio
2. Process and explore the data, to find out more how to proceed during model creation
3. Build and optimize different models
4. Use the best model on the test set and evaluate the results
5. Conclution, limitations and comments

The ambition is to get a better understanding of which parameters and methods can be used for image processing problems.

The dataset

The original dataset can be found in different sources. I downloaded the data from **Kaggle** and reuploaded it into my dropbox so it can be automatically downloaded without having to register on the website.

In addition to the packages used during the MovieLens project I utilized the **downloader package**. The code for downloading and extracting the model looks as follows:

```
################################
# Create train set, test set
################################

# Load required packages if required
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(downloader)) install.packages("downloader", repos = "http://cran.us.r-project.org")
if(!require(matrixStats)) install.packages("matrixStats", repos = "http://cran.us.r-project.org")

# Download the dataset from my Dropbox and unzip it
url <- "https://www.dropbox.com/s/7yzdmv2im26hauh/fashionmnist.zip?raw=1"
download(url, dest="fashionmnist.zip", mode="wb")
unzip ("fashionmnist.zip", exdir = ".")
```

Afterwards we are loading the .csv files into R and storing them into our variables:

```
# Create the data set for model building and verification
verification = read.csv("fashion-mnist_test.csv", header = TRUE)
modelling = read.csv("fashion-mnist_train.csv", header = TRUE)
```

Now two datasets have been created one for modelling with 60,000 samples and one for the testing/verification of our final model with 10,000 samples. Let's take a look at the structure of the data:

```
modelling[1:5,1:10]
```

```
##   label pixel1 pixel2 pixel3 pixel4 pixel5 pixel6 pixel7 pixel8 pixel9
## 1     2      0      0      0      0      0      0      0      0      0
## 2     9      0      0      0      0      0      0      0      0      0
## 3     6      0      0      0      0      0      0      0      5      0
## 4     0      0      0      0      1      2      0      0      0      0
## 5     3      0      0      0      0      0      0      0      0      0
```

Each row contains the label in the first column and greyvalue for the 28 by 28 pixels of the image stored in the columns 2 to 785. Formulated differently, we have one picture plus label stored in each row of the matrix.

For easier processing we are also seperating the labels from the images:

```
# Seperate labels and images
x <- modelling[,2:ncol(modelling)]
y <- factor(modelling[,1])
x_verification <- verification[,2:ncol(verification)]
y_verification <- factor(verification[,1])
```

The images are currently stored in a matrix as you can see here:

```
dim(x)
```

```
## [1] 60000   784
```

```
dim(x_verification)
```

```
## [1] 10000   784
```

For some kind of analysis it is easier to have arrays, so we create an additional dataset with reshaped data:

```
x_images <- array(as.numeric(unlist(x)), dim=c(60000, 28, 28))
x_verification_images <- array(as.numeric(unlist(x_verification)),
                               dim=c(10000, 28, 28))
```

Now we have the data in the following format:

```
dim(x_images)
```

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

```
dim(x_verification_images)
```

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

To allow for faster computation we use a subset out of our training set, which we create as follows:

```
x_sub <- x_images[1:1000,,]
y_sub <- y[1:1000]
x_test_sub <- x_images[1001:1200,,]
y_test_sub <- y[1001:1200]
```

And as a last step in this section we will create the fashion categories according to the numbers which indicate the categories:

```
fashion_cat = c('T-shirt/top',
                'Trouser',
                'Pullover',
                'Dress',
                'Coat',
                'Sandal',
                'Shirt',
                'Sneaker',
                'Bag',
                'Ankle boot')
```

2. Analysing the data

In this section we will explore the data more in depth. How do the images acutally look like, do we see patterns between the categories or do we have some promising points for leveraging the most out of our method toolbox?

Let's take a look at the first 32 pictures of the dataset:

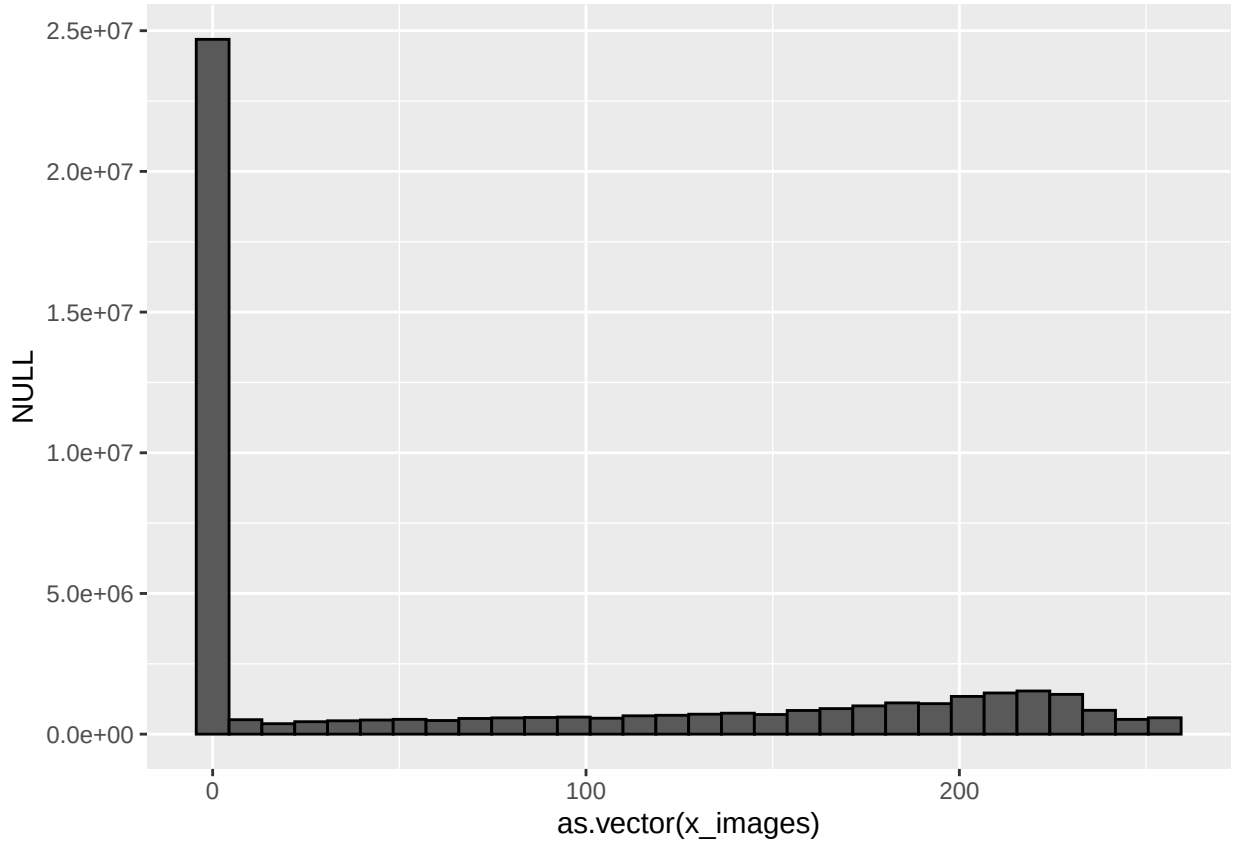
```
# Plot the first 16 images with labels to get an impression how they look
# Define a four by four grid for the plot
par(mfrow=c(4,4))
# Setting the margins around the pictures with "mar"
par(mar=c(0, 0, 1.5, 0))

# Plot every image (remember to reverse) on a gray scale with label
# Reverse the color scheme to get a white background
for (i in 1:16) {
  image(1:28, 1:28, as.matrix(x_images[i,,])[,28:1], col = gray((255:0)/255), xaxt = 'n',
        yaxt = 'n', xlab = "", ylab = "", asp = 1, main = paste(fashion_cat[y[i]]))
}
```


```
# Plot the next 16 images with the same logic
for (i in 17:32) {
  image(1:28, 1:28, as.matrix(x_images[i,,])[,28:1], col = gray((255:0)/255), xaxt = 'n',
        yaxt = 'n', xlab = "", ylab = "", asp = 1, main = paste(fashion_cat[y[i]]))
}
```

As we can see, we have some "low quality" pictures of several types of fashion, which can be relatively easily identified manually. But how can we do that with help of machine learning?

Let us start simple. First let us check whether we can set a threshold for making the pictures even simpler:

```
qplot(as.vector(x_images), bins = 30, color = I("black"))
```

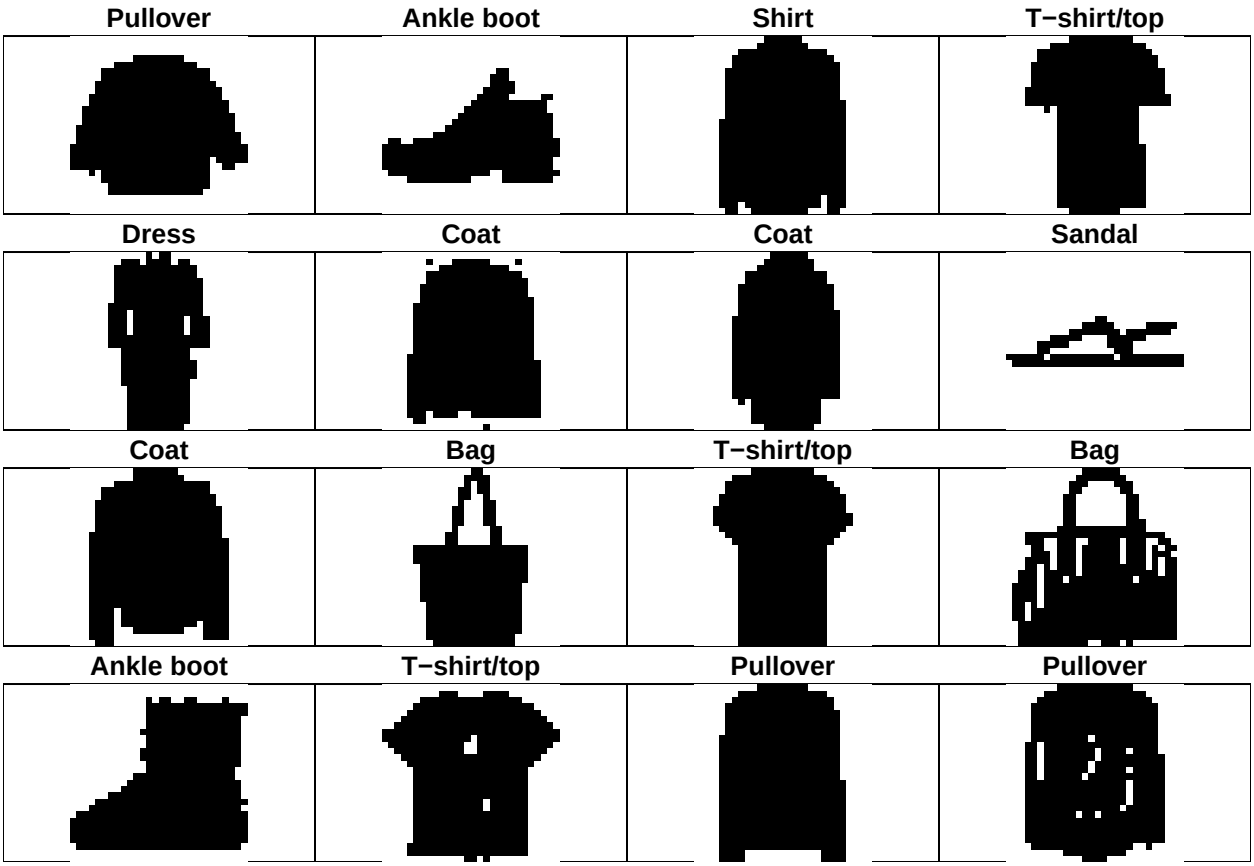


As we can see there is no clear cutoff, but we can try to set a low cutoff threshold, so we only have black and white left without anything in between. Here is the code we use on the subset (faster computation) for a cutoff below 10:

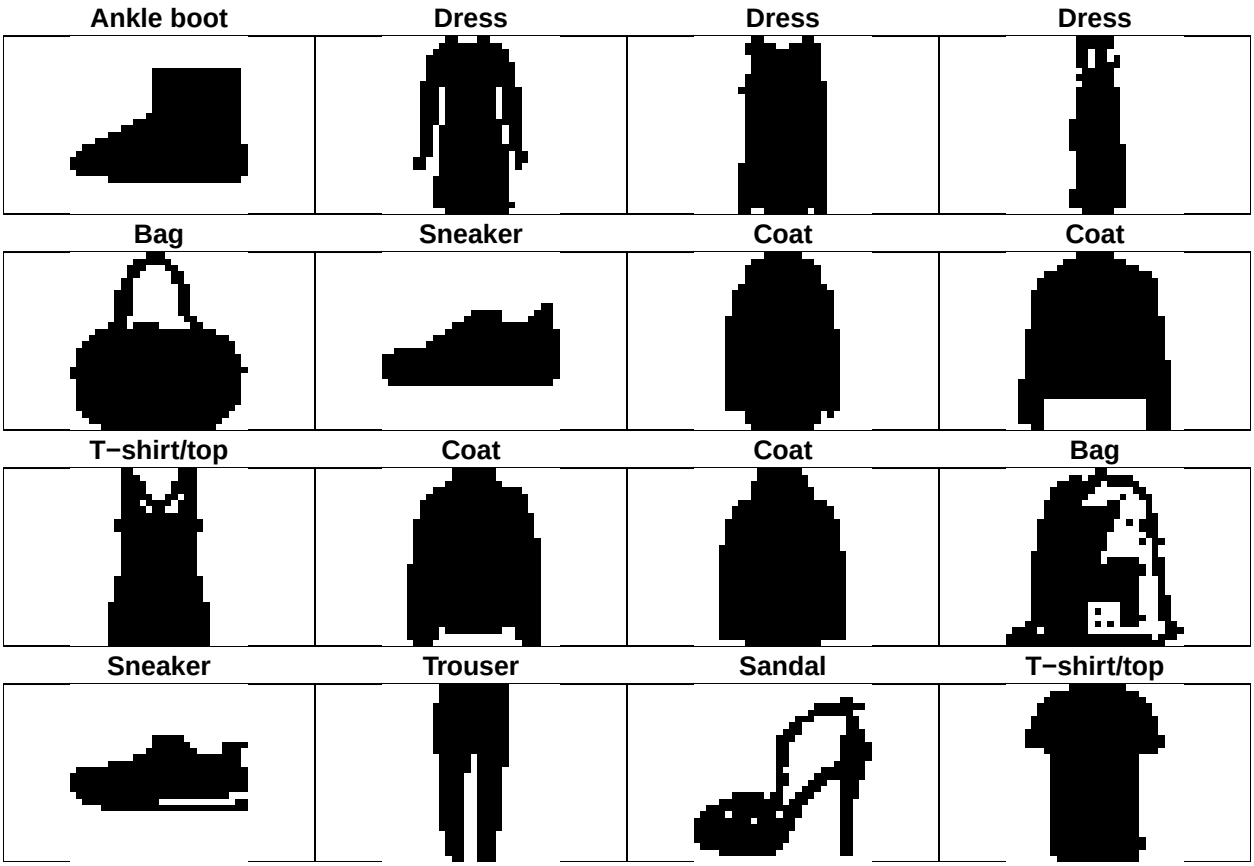
```
x_sub_bw <- ifelse(x_sub < 10, 0 , 1)
```

Here are the same 32 pictures again

```
# Plot the first 16 images with labels to get an impression how they look
# Define a four by four grid for the plot
par(mfrow=c(4,4))
# Setting the margins around the pictures with "mar"
par(mar=c(0, 0, 1.5, 0))

# Plot every image (remember to reverse) on a gray scale with label
# Reverse the color scheme to get a white background
for (i in 1:16) {
  image(1:28, 1:28, as.matrix(x_sub_bw[i,,])[,28:1], col = gray(1:0), xaxt = 'n',
        yaxt = 'n', xlab = "", ylab = "", asp = 1, main = paste(fashion_cat[y[i]]))
}
```




```
# Plot the next 16 images with the same logic
for (i in 17:32) {
  image(1:28, 1:28, as.matrix(x_sub_bw[i,,])[,28:1], col = gray(1:0), xaxt = 'n',
        yaxt = 'n', xlab = "", ylab = "", asp = 1, main = paste(fashion_cat[y[i]]))
}
```



As we can see, this does not help us. for example picture 3, 6 and 15 (shirt, coat and pullover) now look even more similar. So we will not pursue this approach any further.

Next, we can take a look at the intensity of the pictures, so the mean graycolor value for each picture. We exclude the first column, which contains the label, group by the label and add the fashion category vector so we have a better understanding:

```
arrange(cbind(modelling %>% mutate(row_mean = rowMeans(modelling[,-1])) %>%
        group_by(label) %>% summarize(avg_pic_int = mean(row_mean)),fashion_cat),
        desc(avg_pic_int))
```

```
##    label avg_pic_int fashion_cat
## 1      4    98.15566        Coat
## 2      2    95.71933    Pullover
## 3      8    90.03576         Bag
## 4      6    85.11510       Shirt
## 5      0    82.87926 T-shirt/top
## 6      9    77.00715  Ankle boot
## 7      3    66.20149       Dress
## 8      1    56.74837     Trouser
## 9      7    42.83567     Sneaker
## 10     5    34.87052      Sandal
```

Here is the standard deviation for the intensity by category:

```
arrange(cbind(modelling %>% mutate(row_sds = rowSds(as.matrix(modelling[,-1]))) %>%
        group_by(label) %>% summarize(avg_sds = mean(row_sds)),fashion_cat),
        desc(avg_sds))
```

```
##    label  avg_sds fashion_cat
## 1      9 91.94726  Ankle boot
## 2      4 90.36731        Coat
## 3      8 87.32423         Bag
## 4      3 85.63856       Dress
## 5      1 85.02875     Trouser
## 6      2 82.15846    Pullover
## 7      0 81.42681 T-shirt/top
## 8      6 76.85188       Shirt
## 9      7 72.85919     Sneaker
## 10     5 63.28603      Sandal
```

Furthermore we look how many pixels are unequal to zero, and therefore used for the actual image excluding the background:

```
arrange(cbind(modelling %>% mutate(pixels = rowSums(modelling[,-1]>0)) %>%
        group_by(label) %>% summarize(avg_pixels_used = mean(pixels)),fashion_cat),
        desc(avg_pixels_used))
```

```
##    label avg_pixels_used fashion_cat
## 1      2        508.4588    Pullover
## 2      6        493.3368       Shirt
## 3      4        471.9568        Coat
## 4      0        466.3948 T-shirt/top
## 5      8        458.3810         Bag
## 6      9        380.6383  Ankle boot
## 7      3        336.6065       Dress
## 8      1        273.8315     Trouser
## 9      7        264.6172     Sneaker
## 10     5        251.6750      Sandal
```

We can also calculate the average pixel intensity by dividing the sum of the grayscale values divided by the number of pixels unequal to zero:

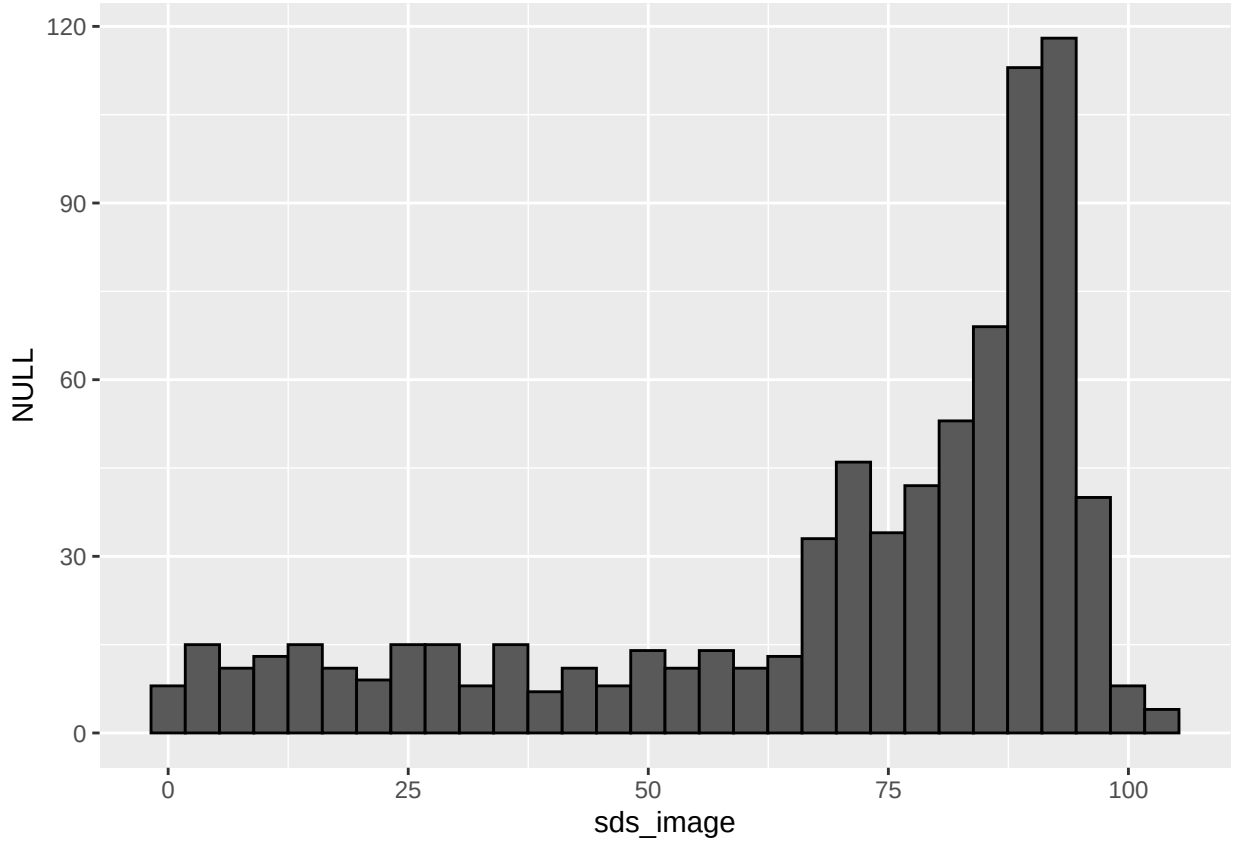
```
arrange(cbind(modelling %>% mutate(pixel_int = rowSums(modelling[,-1])/
        rowSums(modelling[,-1]>0)) %>% group_by(label) %>%
        summarize(avg_pixel_int = mean(pixel_int)),fashion_cat),
        desc(avg_pixel_int))
```

```
##    label avg_pixel_int fashion_cat
## 1      1      163.6052     Trouser
## 2      4      163.0429        Coat
## 3      9      158.2658  Ankle boot
## 4      3      155.0386       Dress
## 5      8      152.7075         Bag
## 6      2      147.7495    Pullover
## 7      0      139.5437 T-shirt/top
```

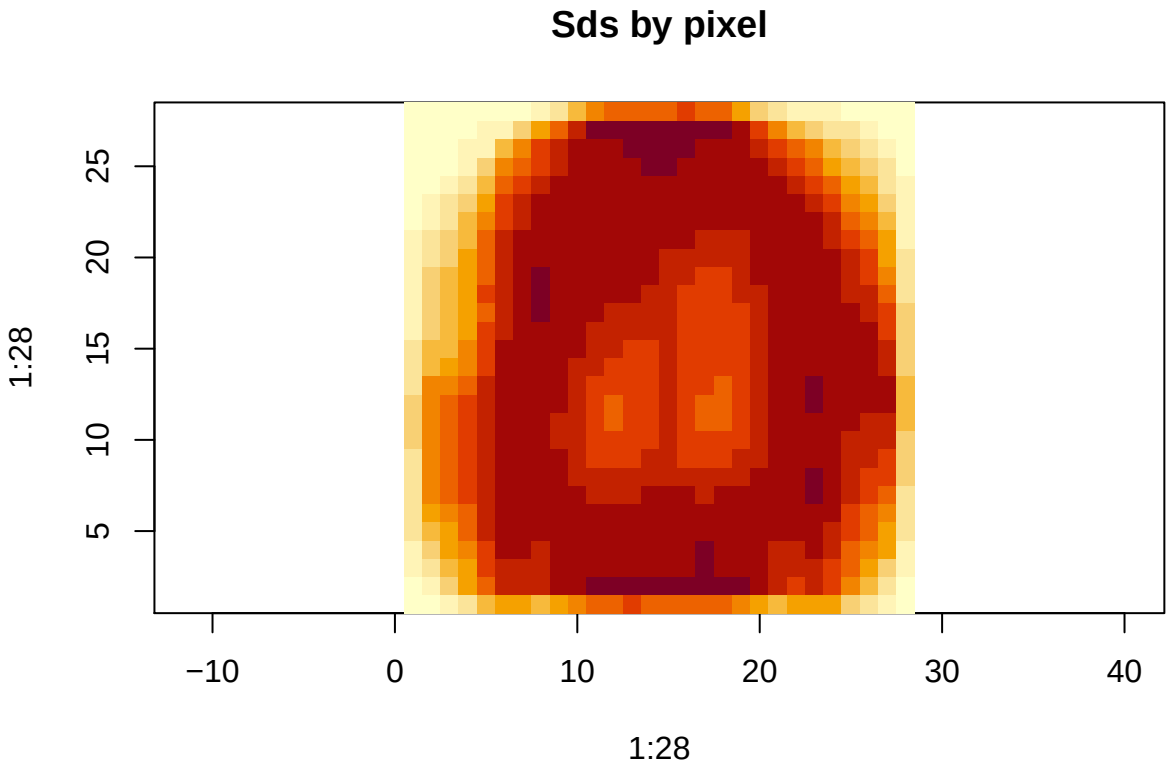
```
## 8       6       135.2090       Shirt
## 9       7       126.4115     Sneaker
## 10      5       107.2801      Sandal
```

Finally we take a look at the variability by pixel on overall level. Herefore we calculate and plot the standard deviation by pixel and make it even more understandable by visualizing the results. The code looks as follows:

```
sds_image <- colSds(as.matrix(x))
qplot(sds_image, bins = "30", color = I("black"))
```




```
image(1:28, 1:28, matrix(sds_image, 28, 28)[,28:1], asp = 1, main = "Sds by pixel")
```



So what can we conclude from the analysis? We can see that we can't set a threshold for the grayscale value to enhance the possibility for recognizing images, because the shapes in some categories are to similar.