Keras-Fashion_MNIST

Akshata Bhandiwad(axb200026)

Fashion Images Classification

```
if(!require("pacman")) install.packages("pacman")
pacman::p_load(dplyr, ggplot2, tfruns, tfestimators, keras, tidyr,
               data.table,dslabs, tensorflow)

library(keras)
library(tensorflow)
#install_keras()
#install_tensorflow()

knitr::opts_chunk$set(echo = TRUE)
search()
theme_set(theme_classic())
set.seed(42)
options(scipen = 100, digits = 4)
```

Loading required packages

```
fashion <- dataset_fashion_mnist()
```

Loading the data

1. Verifying the images in the training data set

```
set.seed(42)

# training and Test data sets
training_images <- fashion$train$x
training_labels <- fashion$train$y


test_images <- fashion$test$x
test_labels <- fashion$test$y

# number of images in the training data set
dim(training_images)
```

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

```
dim(training_labels)
```

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

```
# verifying label range
sort(unique(training_labels))
```

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

```
# Label names
Labels <- data.frame("Category" = sort(unique(training_labels)),
                     "Names" = c('T-shirt/top', 'Trouser', 'Pullover',
                                 'Dress','Coat', 'Sandal', 'Shirt',
                                 'Sneaker', 'Bag', 'Ankle boot'))

Labels
```

```
##    Category       Names
## 1         0 T-shirt/top
## 2         1     Trouser
## 3         2    Pullover
## 4         3       Dress
## 5         4        Coat
## 6         5      Sandal
## 7         6       Shirt
## 8         7     Sneaker
## 9         8         Bag
## 10        9  Ankle boot
```

The above results verify that The training image data is a 3-d array (images, width, height) of grayscale values. There are 60,000 images in the training data and each image is represented as 28x28 pixels.

The labels contain 10 unique categories and its details can be seen from the table above.

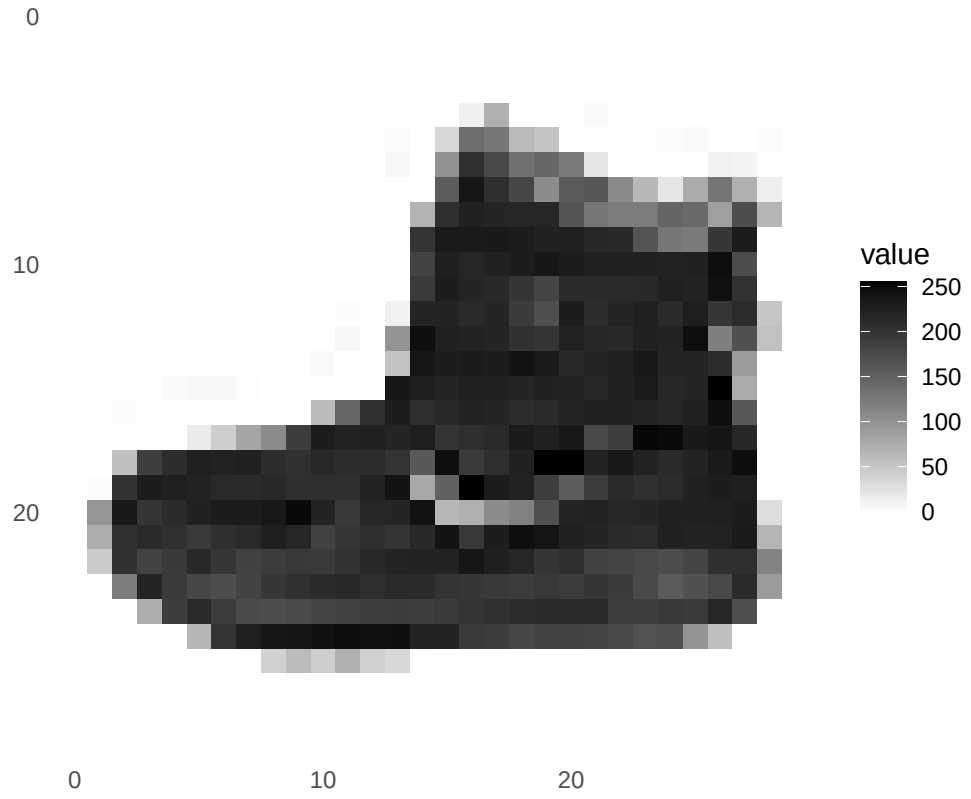
2. Standardizing the pixel values of the images

```
set.seed(42)

# Printing the first image of training
image_1 <- as.data.frame(training_images[1, , ])
colnames(image_1) <- seq_len(ncol(image_1))

image_1$y <- seq_len(nrow(image_1))
image_1 <- gather(image_1, "x", "value", -y)
image_1$x <- as.integer(image_1$x)

# Plotting the image
ggplot(image_1, aes(x = x, y = y, fill = value)) +
                      geom_tile() +
                      scale_fill_gradient(low = "white",
                                          high = "black",
                                          na.value = NA) +
                      scale_y_reverse() +
                      theme_minimal() +
                      theme(panel.grid = element_blank())   +
                      theme(aspect.ratio = 1) +
                      xlab("") +
                      ylab("")
```




```
label_1 <- Labels$Names[ Labels$Category %in% training_labels[1]]
label_1
```

```
## [1] "Ankle boot"
```

```
# Rename columns and standardize feature values
# Train
colnames(training_images) <- paste0("V", 1:ncol(training_images))
training_images <- training_images / 255

# Test
colnames(test_images) <- paste0("V", 1:ncol(test_images))
test_images <- test_images / 255
```

As seen from the image plotted, the pixel value for each image ranges between 0 and 255. The training set and the testing set are preprocessed to scale the values to a range of 0 to 1.

3. Setting up the network architecture

```
set.seed(42)

# Initiate the network with two hidden layers
model <- keras_model_sequential()

# Network architecture
model %>%
  layer_flatten(input_shape = c(28, 28)) %>%
  layer_dense(units = 128, activation = "relu") %>%
  layer_dense(units = 64, activation = "relu") %>%
  layer_dense(units = 10, activation = "softmax")


summary(model)
```

```
## Model: "sequential"
## _________________________________________________________________________
## Layer (type)                         Output Shape                     Param #
## =========================================================================
## flatten (Flatten)                    (None, 784)                      0
## _________________________________________________________________________
## dense_2 (Dense)                      (None, 128)                      100480
## _________________________________________________________________________
## dense_1 (Dense)                      (None, 64)                       8256
## _________________________________________________________________________
## dense (Dense)                        (None, 10)                       650
## =========================================================================
## Total params: 109,386
## Trainable params: 109,386
## Non-trainable params: 0
## _________________________________________________________________________
```

Since this is a multiclass classification problem and the output layer should be with 10 nodes(one node per class), activation function of softmax is used. This fuction outputs a value between 0 and 1, like the perprocessed image values. The neurons is output layer returns a score (logits) at each node that indicates the probability that the current image belongs to one of the 10 digit classes.

As seen from the results, there are around 109K total parameters.

4. Setting up back propogation

```
set.seed(42)

model %>%
              compile(optimizer = optimizer_rmsprop(),
              loss = 'sparse_categorical_crossentropy',
              metrics = c('accuracy'))
```

5. Training the neural network

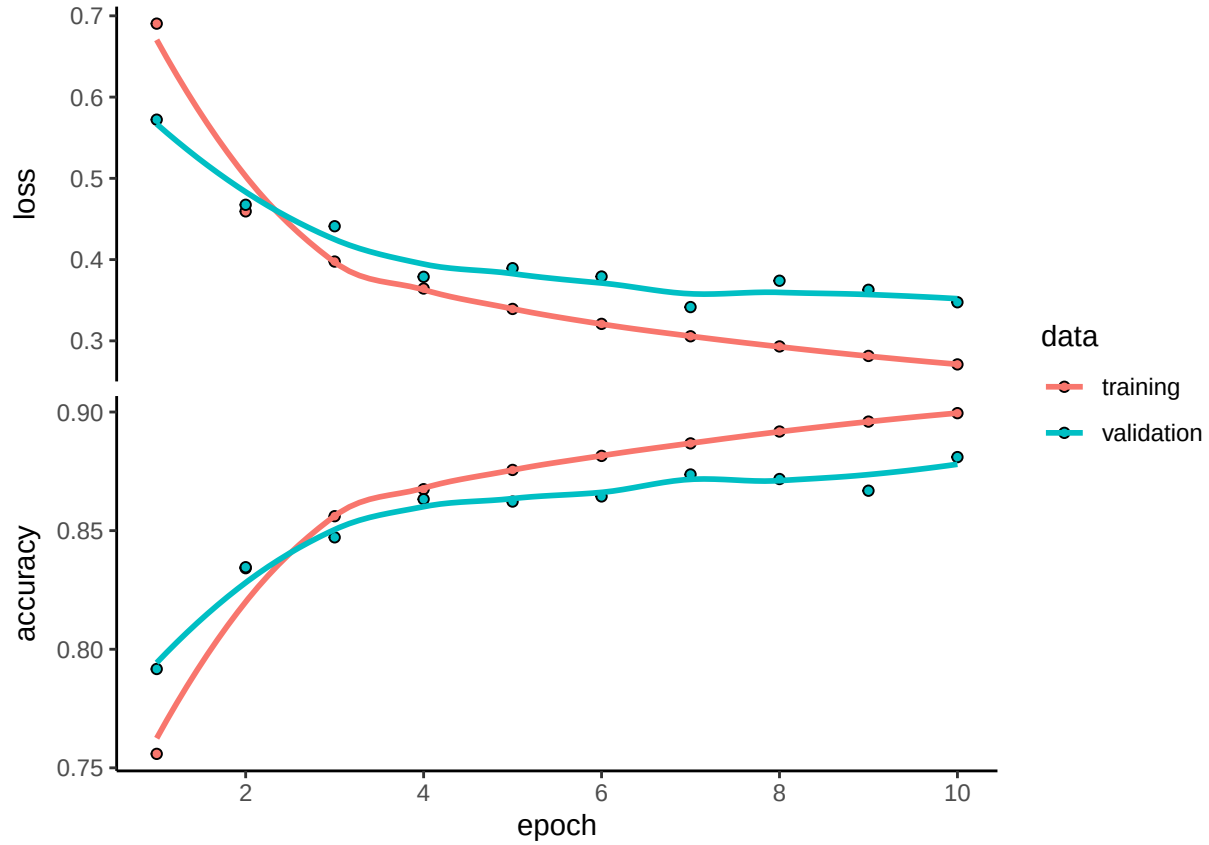
```
set.seed(42)

# Training the model for 10 epochs
model_fit <- model %>% fit(x = training_images,
                           y = training_labels,
                           epochs = 10,
                           batch_size = 256,
                           validation_split=0.2)

# Display output
model_fit
```

```
##
## Final epoch (plot to see history):
##         loss: 0.271
##     accuracy: 0.8995
##     val_loss: 0.3475
## val_accuracy: 0.881
```

```
# Plot loss and accuracy matrix
plot(model_fit)
```



The training data set has been again split into 80% training data and 20% validation dataset. The accuracy of training data is 0.89 and the accuracy of validation dataset is 0.88. This means that the neural network has figured out a pattern match between the image and the labels that worked 89% of the time in training data. There not a huge difference between the training and validation accuracy.

The above plot shows the variation in loss and accuracy for training data (in red) and validation data (in green). After 8 epochs, the training dataset has been stabilized and there is no significant decrease in loss and increase in accuracy. The loss function in validation also stabilizes after 8 epochs. To avoid overfitting, we can consider using 8 epochs.

6. Evaluating accuracy of test dataset

```
set.seed(42)

# Test
score <- model %>%
            evaluate(x = test_images, y = test_labels, batch_size = 256)


score <- as.data.frame(score)
row.names(score) <- 1 : 2
names(score)<-make.names(names(score))



# Test Loss
cat("Test loss:", score[1,1], "\n")
```

```
## Test loss: 0.3722
```

```
# Test Accuracy
cat("Test Accuracy:", score[2,1], "\n")
```

```
## Test Accuracy: 0.8709
```

The accuracy of test data set is approximately 87% with loss function of 0.37

7. Prediction of first five observations in Test data

```
set.seed(42)

predictions <- model %>%
                predict_classes(test_images, batch_size = 256)

Predicted <-  predictions[1:5]
Actual_class <- test_labels[1:5]

Predicted
```

```
## [1] 9 2 1 1 6
```

```
Actual_class
```

```
## [1] 9 2 1 1 6
```

Among the first five predictions, all images are predicted correctly. The first observation is predicted as class 9(Ankle boot) while the actual class is also 9(Ankle boot).