Fashion MNIST Deep Learning Project

TAO JIN

June 27, 2018

Fashion MNIST Deep Learning Project

In this project, we will use two Deep learning models to Classify the image from Fashion-MNIST dataset

- Basic Model (fully connected layer model)
- Convolutional neural networks (ConvNets)

All model will use the Keras framework with R implementation

Fashion-MNIST Dataset

- *https://www.kaggle.com/zalando-research/fashionmnist/data*
- 60000 images for training and 10000 images for testing
- Each example is a 28x28 gray-scale image, associated with a label from 10 classes

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

- The dataset is CSV format. The detailed format is **label, pixel1, pixel2, pixel3, ... pixel784**. Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total

Define the Problem and Assembling Data Set

Problem: Train the 60000 images and test 10000 images to classify the image's label

- Problem Type: Multi-Class and single-label classification
- Model Configuration: the ***softmax*** as the last-layer's activation and the loss function will use the ***categorical_crossentropy***.
- Hyper-parameters setting : use the ***dropout*** and ***L2 regularization*** to reduce over-fitting effects.

Assembling Data Set

- Using the dataset_fashion_mnist() function from Keras to download the dataset (simple model)
- Downloaded from the Kaggle.com, then use the read_csv() to manipulate the data (ConvNets Model)

Simple Neural Network Model

Data Preparation

- dataset_fashion_mnist() function from Keras to get the data.
- prepare the data for training we convert the 3-d arrays into matrices by reshaping width and height into a single dimension (28x28 images are flattened into length 784 vectors).
- Convert the gray-scaled normalization

```
FashionMNIST <- dataset_fashion_mnist()
Train_X <- FashionMNIST$train$x
Train_Y <- FashionMNIST$train$y
Test_X <- FashionMNIST$test$x
Test_Y <- FashionMNIST$test$y
# Reshape
Train_X <- array_reshape(Train_X, c(nrow(Train_X), 784))
Test_X <- array_reshape(Test_X, c(nrow(Test_X), 784))
# Value Normalization
Train_X <- Train_X / 255
Test_X <- Test_X / 255
```

Model

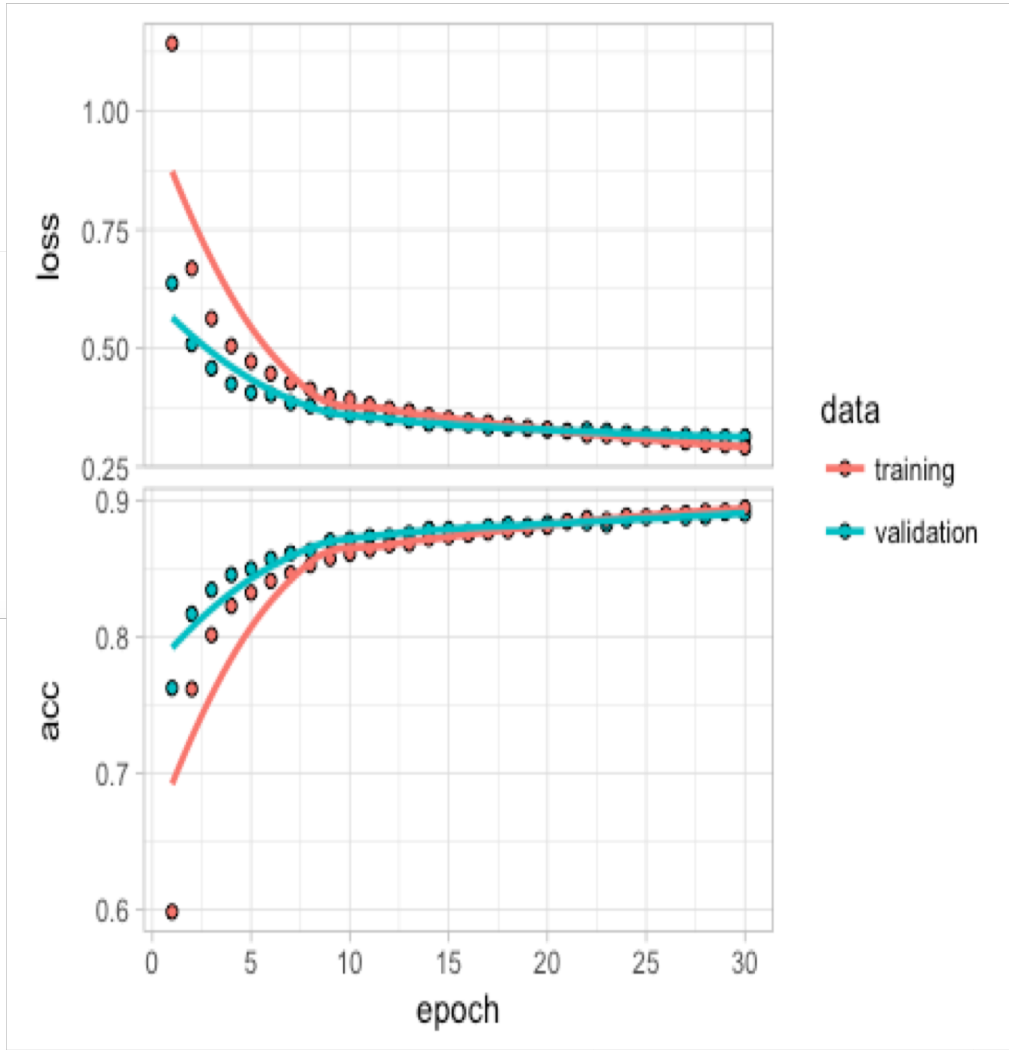
- use a simple stack of fully connected (“dense”) layers with “relu” activations.
- introduce the dropout method to avoid the over-fitting.

```
model <- keras_model_sequential()
model %>%
  layer_dense(units = 256, activation = 'relu', input_shape = c(784)) %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 256, activation = 'relu') %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 256, activation = 'relu') %>%
  layer_dropout(rate = 0.2) %>%
  layer_dense(units = 10, activation = 'softmax')
```

Simple Neural Network Model

- **Simple deep learning model achieves an accuracy of 88.11% and loss of 34.15%.**

```
> summary(model)
_____________________________________________________________________________
Layer (type)                 Output Shape              Param #
=============================================================================
dense_1 (Dense)              (None, 256)               200960
_____________________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_____________________________________________________________________________
dense_2 (Dense)              (None, 256)               65792
_____________________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_____________________________________________________________________________
dense_3 (Dense)              (None, 256)               65792
_____________________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_____________________________________________________________________________
dense_4 (Dense)              (None, 10)                2570
=============================================================================
Total params: 335,114
Trainable params: 335,114
Non-trainable params: 0
_____________________________________________________________________________
>
```



Convolutional Neural Networks Model

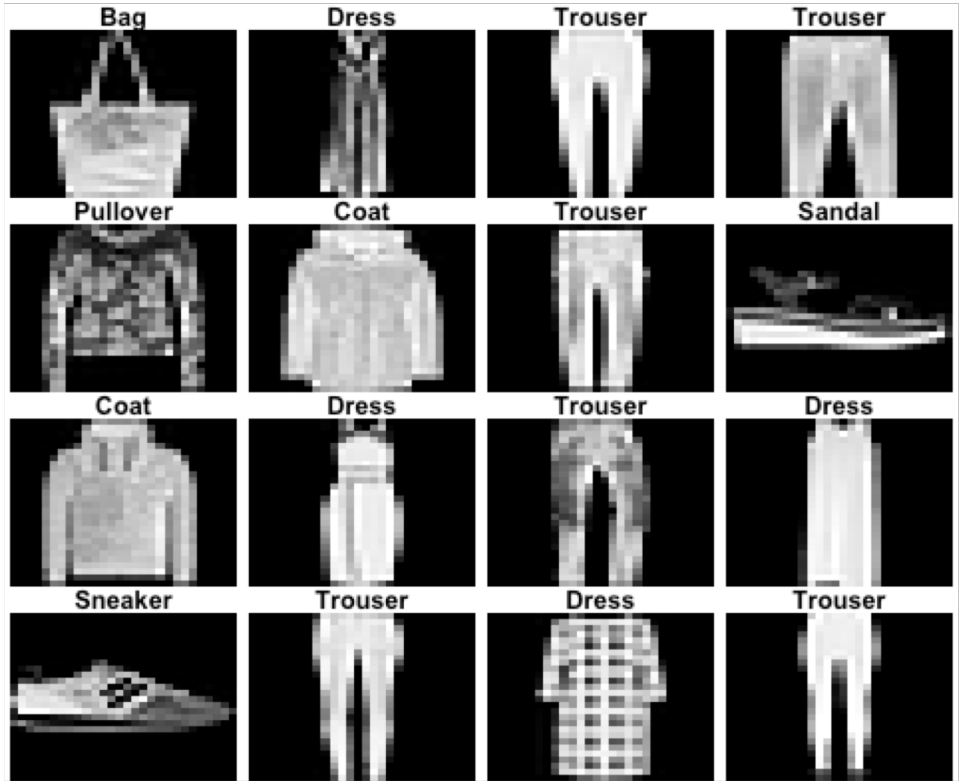
Data Preparation

- **In the ConvNets model, we will use the original CSV data and prepare the 4D Tensors image data format.**

```
# Data downloaded from https://www.kaggle.com/zalando-research/fashionmnist
TrainSetData <- read_csv("fashion-mnist_train.csv", col_types = cols(.default = "i"))
TestSetData <- read_csv("fashion-mnist_test.csv", col_types = cols(.default = "i"))
```

- **Unflattening, Reshaping and Normalization the data**
- **Plot the images as examples**

```
# Function to plot image from a matrix x
plot_image <- function(x, title = "", title.color = "black") {
  dim(x) <- c(ImgRows, ImgCols)
  image(rotate(rotate(x)), axes = FALSE,
        col = grey(seq(0, 1, length = 256)),
        main = list(title, col = title.color))
}
```



Convolutional Neural Networks Model

ConvNets Model

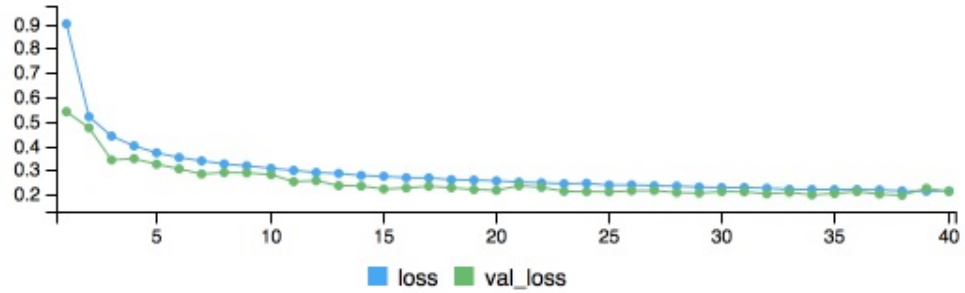
- the stack of ***layer_conv_2d*** and ***layer_max_pooling_2d*** layers.
- A convent takes as input tensors of (***image_height, image_width, image_channel***). In this case, the input for the first layer's size is ***input_shape*** = (28,28,1)
- Batch size is 256, epochs is 40, kernel size first is (5,5) then (3,3), dropout rate is 0.25/0.25/0.4/0.3

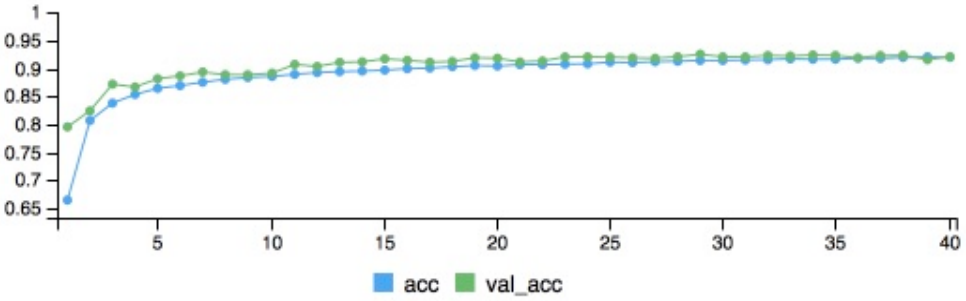
```
model <- keras_model_sequential()
model %>%
  layer_conv_2d(filters = 32, kernel_size = c(5,5), activation = 'relu',
                input_shape = input_shape) %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_dropout(rate = 0.25) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = 'relu') %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_dropout(rate = 0.25) %>%
  layer_conv_2d(filters = 128, kernel_size = c(3,3), activation = 'relu') %>%
  layer_dropout(rate = 0.4) %>%
  layer_flatten() %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = num_classes, activation = 'softmax')
```

Convolutional Neural Networks Model

- **ConvNets model achieves an accuracy of 91.85% and loss is 20%, up from the previous model's accuracy of 88.11% and loss of 34.15%.**

```
> summary(model)
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 24, 24, 32)        832
_________________________________________________________________
max_pooling2d_1 (MaxPooling2D) (None, 12, 12, 32)      0
_________________________________________________________________
dropout_1 (Dropout)          (None, 12, 12, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 10, 10, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2D) (None, 5, 5, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 3, 3, 128)         73856
_________________________________________________________________
dropout_3 (Dropout)          (None, 3, 3, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1152)              0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               147584
_________________________________________________________________
dropout_4 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 242,058
Trainable params: 242,058
Non-trainable params: 0
_________________________________________________________________
>
```

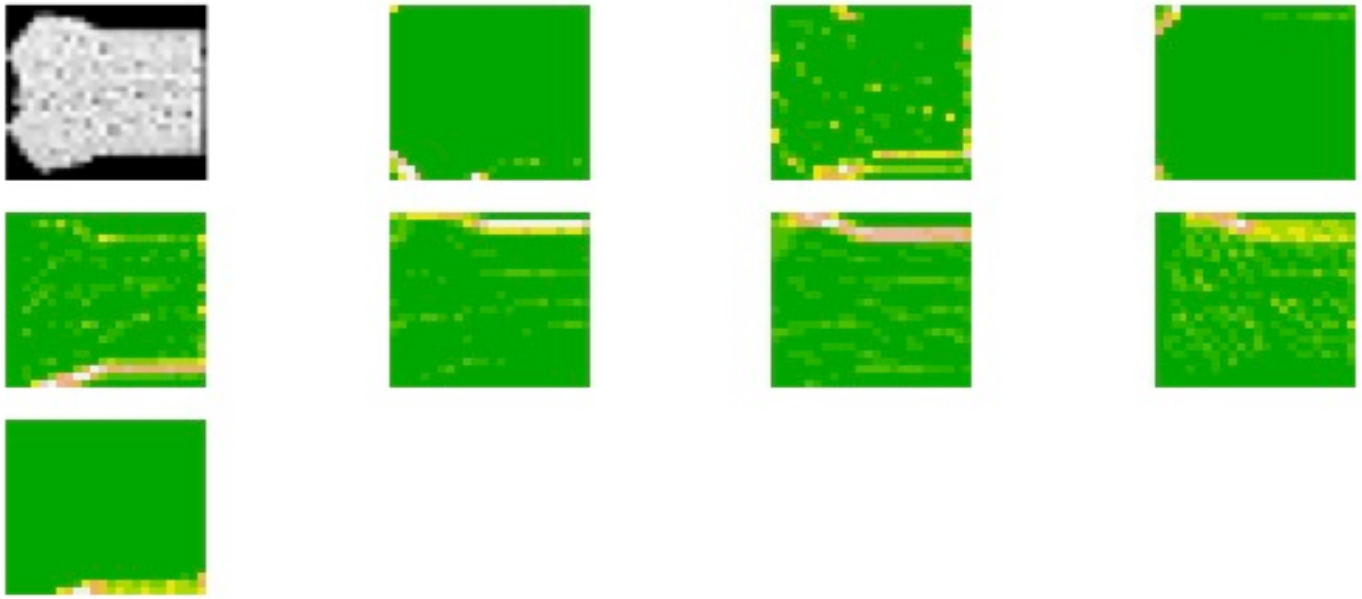




Convolutional Neural Networks Model

Visualizing the Model Predictions

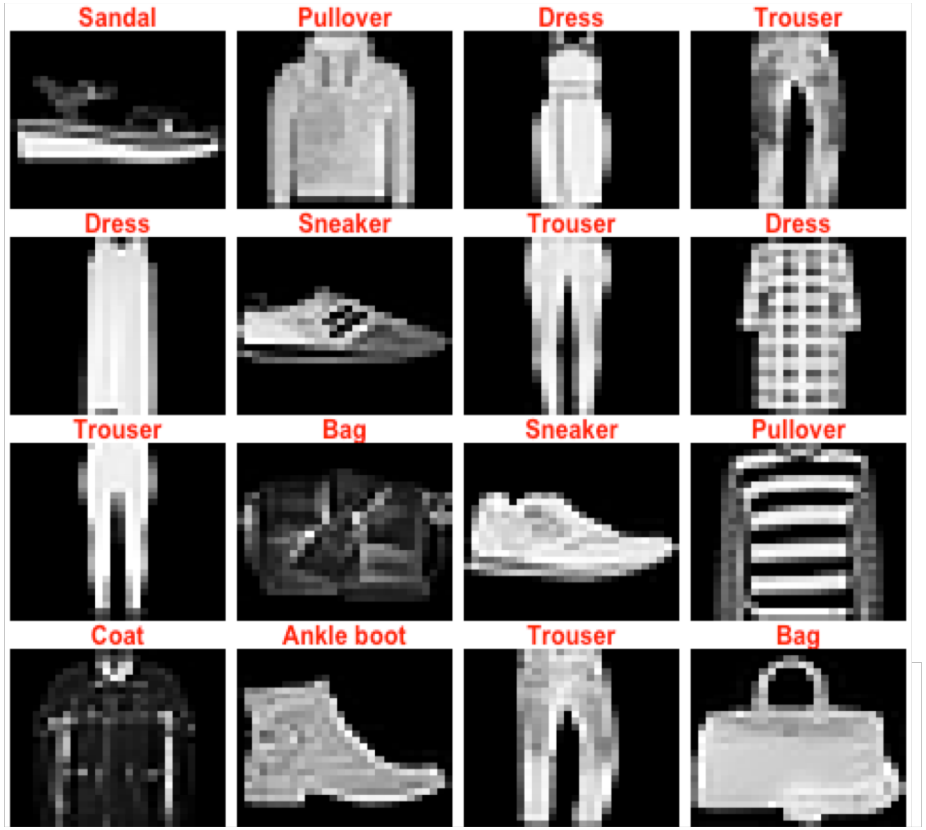
- Lower layer acts as a collection of various edge detectors
- Higher layer carry increasingly less information about the visual contents of the image, and increasingly more information related to the class of the image



Convolutional Neural Networks Model

- **Visualizing the Model Predictions**

```
for (i in 1:32) {
  n_row <- i * 10
  T_Tensor <- Train_X[n_row, , , 1]
  dim(T_Tensor) <- c(1, ImgRows, ImgCols, 1)
  pred <- model %>% predict(T_Tensor)
  plot_image(Train_X[n_row, , , 1],
             Fashion_Labels[which.max(pred)],
             "red")
}
```



Convolutional Neural Networks Model

Summary

- Our ConvNets model achieved an accuracy of 91.85%. It turns out our classifier does better than the Kaggle's best baseline reported here, which is an SVM classifier with mean accuracy of 89.7%.
- Comparing the simple model, ConvNets is the best model for the attacking the image classification problems.
- Tuning the model and hyper-parameters is very important to improve the accuracy.