**Fashion MNIST Deep Learning Project**

Contents

1. Project Overview 2

2. Define the Problem and Assembling Data Set 2

2.1 Define the Problem 2

2.2 Data Preparation 2

3. Simple Neural Network Model 3

3.1 Data Preparation 3

3.2 Model 3

3.3 Model Evaluation 4

4. Convolutional Neural Networks (CNN) Model 5

4.1 Data Preparation 5

4.2 Convolutional Neural Network Model 6

4.3 Visualizing the ConvNets Output 7

4.4 Model Evaluation 10

4.5 Visualizing the Model Predictions 10

5. Summary 11

# Project Overview

In this project, we will use the basic neural network model and the convolutional neural networks (ConvNets) model to classify the images from the Fashion-MNIST dataset under R and Keras framework. We will evaluate two models’ performances and plot theirs results.

Fashion-MNIST is the MNIST-like dataset, which is consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 gray-scale image, associated with a label from 10 classes, which includes:

0 T-shirt/top

1 Trouser

2 Pullover

3 Dress

4 Coat

5 Sandal

6 Shirt

7 Sneaker

8 Bag

9 Ankle boot

Each training and test example is assigned to one of the labels.

# Define the Problem and Assembling Data Set

In this section, we will define the image classification problem, so we can choose the right model architecture, loss function and so on. Also we will explore and assemble the data set and be ready for learning.

## Define the Problem

Based on the Fashion-MNIST dataset, we will train the 60000 images and test 10000 images to classify the image’s label. This is Multi-Class and single-label classification problem. So, for the deep learning model’s architecture, we will use the ***softmax*** as the last-layer’s activation and the loss function will use the ***categorical\_crossentropy***.

For tuning the model’s hyper-parameters, we will use the ***dropout*** and ***L2 regularization*** to reduce over-fitting effects.

## Data Preparation

The Fashion-MNIST data set is downloaded from

*https://www.kaggle.com/zalando-research/fashionmnist/data*

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. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255. The training and test data sets have 785 columns. The first column consists of the class labels (see above), and represents the article of clothing. The rest of the columns contain the pixel-values of the associated image.

In Keras framework, image data is defined in 4D tensors: (samples, height, width, channel). For this project, the gray-scaled image, the channel is 1.The height and width is 28.

There are two ways to assess the dataset:

1. Using the dataset\_fashion\_mnist() function from Keras to download the dataset
2. Downloaded from the Kaggle.com, then use the read\_csv() to manipulate the data

In the simple neural network model, we will use the method 1 to get the data and in the ConvNets model we will use read\_csv function to assess the dataset.

# Simple Neural Network Model

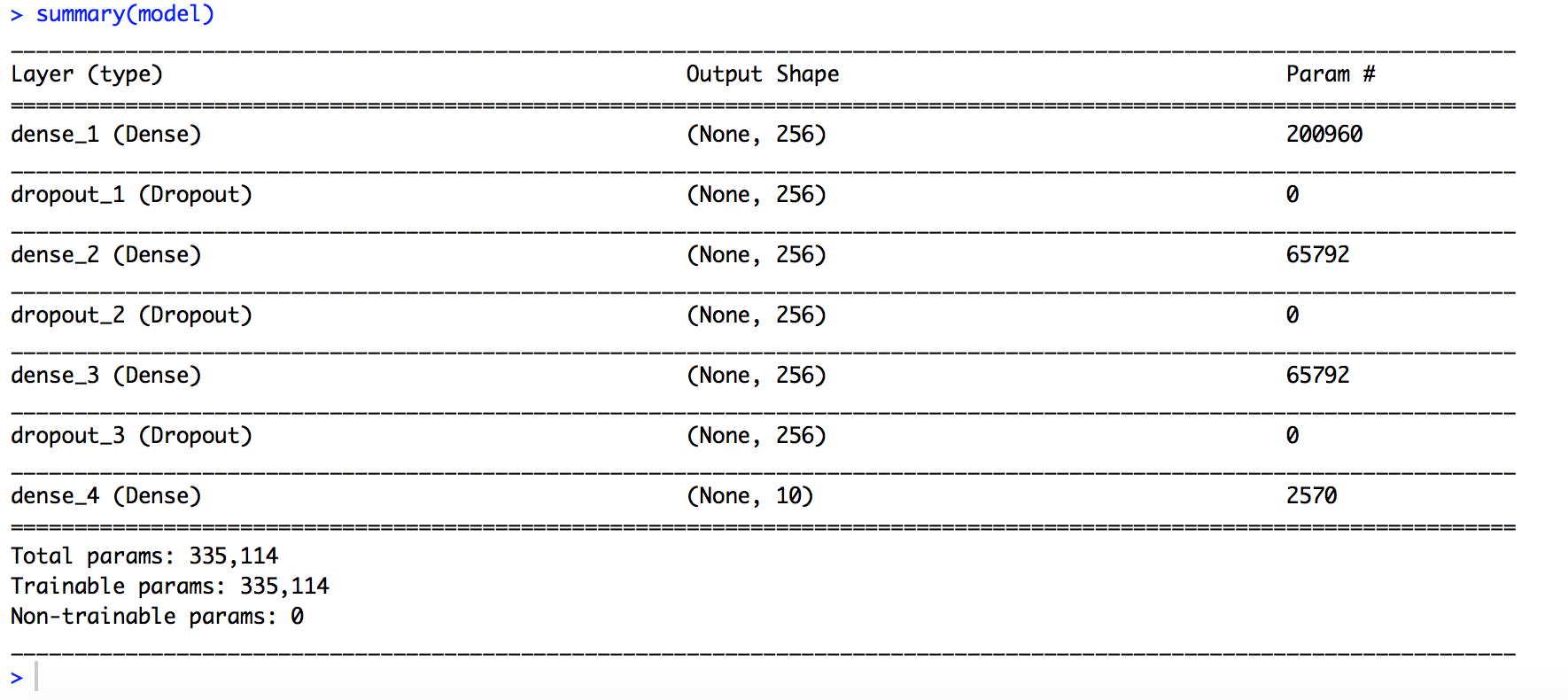
## Data Preparation

**# Data Preparation  
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  
  
Train\_Y <-** to\_categorical**(Train\_Y, 10)  
Test\_Y <-** to\_categorical**(Test\_Y, 10)**We will dataset\_fashion\_mnist() function from Keras to get the data. To 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). Then, we convert the gray-scaled values from integers ranging between 0 to 255 into floating point values ranging between 0 and 1.

## Model

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

# The simple deep learning model  
  
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')  
  
summary(model)



The total training parameters are 335,114. Without GPU, the whole training time will be around 15 minutes.

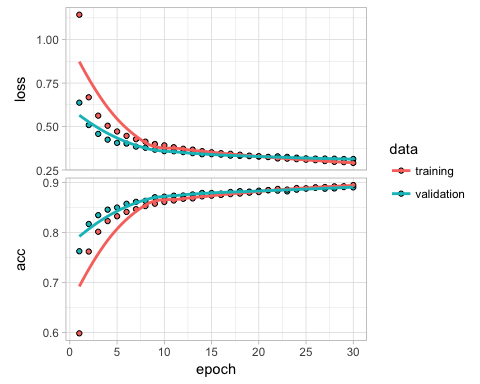
# Setting the loss function, optimizer and regularization   
  
model %>% compile(  
 loss = 'categorical\_crossentropy',  
 optimizer = optimizer\_rmsprop(lr = 1e-4),  
 metrics = c('accuracy')  
)  
We will use the categorical\_crossentropy loss function, the optimizer is rmsprop with the L2 regularization.

Training the model: the validation set ratio is 20%. The batch size is 128 and epochs times is 30.

history <- model %>% fit(  
 Train\_X, Train\_Y,   
 epochs = 30, batch\_size = 128,   
 validation\_split = 0.2  
);

## Model Evaluation

plot(history)



# Evaluate the model’s performance on the test data:  
model %>% evaluate(Test\_X, Test\_Y)

## $loss  
## [1] 0.3415341  
##   
## $acc  
## [1] 0.8811

This simple deep learning model achieves an accuracy of 88.11% and loss of 34.15%.

# Convolutional Neural Networks (CNN) Model

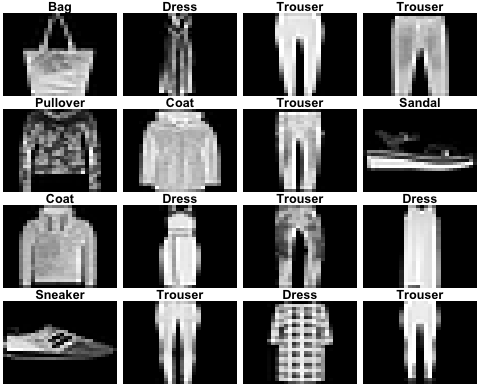
## Data Preparation

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"))  
  
# Fashion MNIST Image Data is 28\*28 pixels  
ImgRows <- 28  
ImgCols <- 28  
  
# Data Preparation  
  
Train\_X <- as.matrix(TrainSetData[, 2:dim(TrainSetData)[2]])  
Train\_Y <- as.matrix(TrainSetData[, 1])  
  
# Unflattening the data.  
dim(Train\_X) <- c(nrow(Train\_X), ImgRows, ImgCols, 1)   
  
Test\_X <- as.matrix(TestSetData[, 2:dim(TrainSetData)[2]])  
Test\_Y <- as.matrix(TestSetData[, 1])  
dim(Test\_X) <- c(nrow(Test\_X), ImgRows, ImgCols, 1)

We will set the Labels set and plot the images with labels as examples.

Fashion\_Labels<-c( "T-shirt/top", "Trouser", "Pullover", "Dress", "Coat",  
 "Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot")  
  
# Function to rotate matrices  
rotate <- function(x) t(apply(x, 2, rev))  
  
# 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))  
}



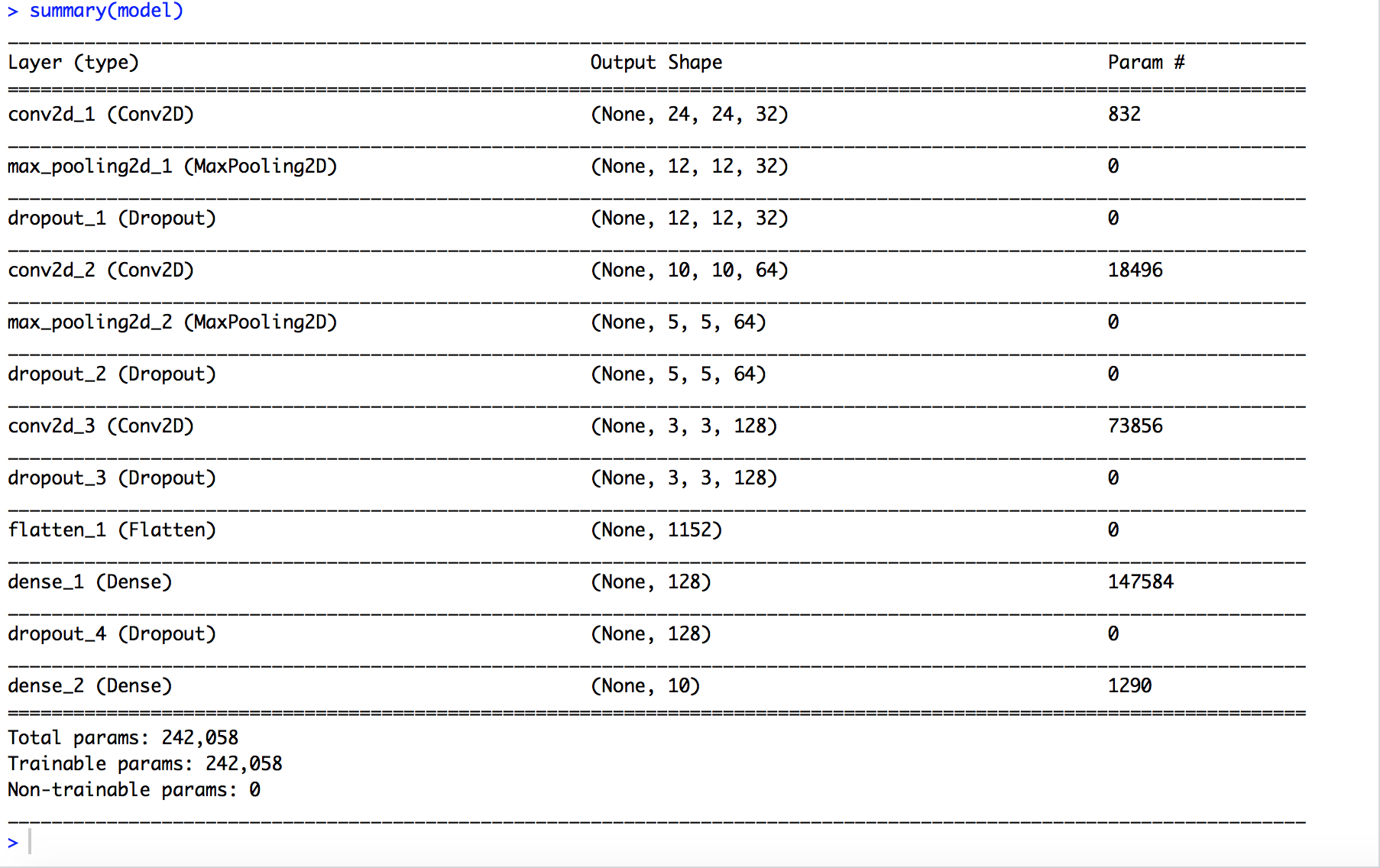
# Hyperparameters Setting  
batch\_size <- 256  
num\_classes <- 10  
epochs <- 40  
  
input\_shape <- c(ImgRows, ImgCols, 1)  
Train\_X <- Train\_X / 255  
Test\_X <- Test\_X / 255  
  
Train\_Y <- to\_categorical(Train\_Y, num\_classes)  
Test\_Y <- to\_categorical(Test\_Y, num\_classes)

## Convolutional Neural Network Model

We will build the ConvNets model by using 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). For the pooling layer, the role is to aggressively down-sample feature maps, much like strided convolutions. the reason to use down-sampling is to reduce the number of feature-map coefficients to process, as well as to induce spatial-filter hierarchies by making successive convolution layers look at increasingly large Windows.

# Convolutional Nerual Network Model   
  
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')  
  
summary(model)

After tuning the model several times, we decide use the above layer architecture and kernel size.



The total training parameters will be 242,058. The training time on laptop will be around 15 minutes.

# compile model  
model %>% compile(  
 loss = loss\_categorical\_crossentropy,  
 optimizer = optimizer\_adadelta(),  
 metrics = c('accuracy')  
)  
  
# train and evaluate  
model %>% fit(  
 Train\_X, Train\_Y,  
 batch\_size = batch\_size,  
 epochs = epochs,  
 verbose = 1,  
 validation\_data = list(Test\_X, Test\_Y)  
)

## Visualizing the ConvNets Output

Visualizing intermediate ConvNets outputs is also called "intermediate activations". This is useful to understand how successive ConvNets layers transform their input, and to get a first idea of the meaning of individual ConvNets filters.

Intermediate activations visualization consists of displaying the feature maps that are output by various convolution and pooling layers in a network, given a certain input (the output of a layer is often called its activation, the output of the activation function). This gives a view into how an input is decomposed unto the different filters learned by the network.

* + 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

# Extracts the outputs of the top 8 layers:  
layer\_outputs <- lapply(model$layers[1:8], function(layer) layer$output)  
# Creates a model that will return these outputs, given the model input:  
activation\_model <- keras\_model(inputs = model$input, outputs = layer\_outputs)

We will randomly choose one image to show the ConvNets output. Here we select image Train\_X[1002,,,1].

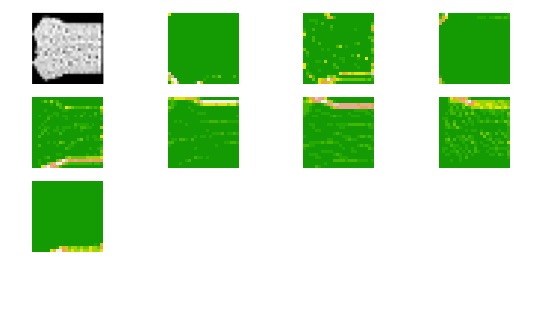
# Visualization the activation of Convolutional layers   
plot(as.raster(Train\_X[1002,,,1]))  
img\_tensor <- Train\_X[1002,,,1]  
img\_tensor <- array\_reshape(img\_tensor, c(1, 28, 28, 1))  
  
dim(img\_tensor)

## [1] 1 28 28 1

activations <- activation\_model %>% predict(img\_tensor)  
  
first\_layer\_activation <- activations[[1]]  
dim(first\_layer\_activation)

## [1] 1 24 24 32

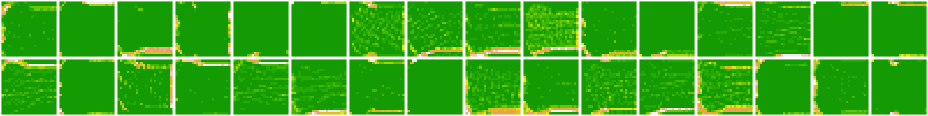
plot\_channel <- function(channel) {  
 rotate <- function(x) t(apply(x, 2, rev))  
 image(rotate(channel), axes = FALSE, asp = 1,   
 col = terrain.colors(12))  
}  
  
for (i in 1:8) {  
 plot\_channel(first\_layer\_activation[1,,,i])  
}



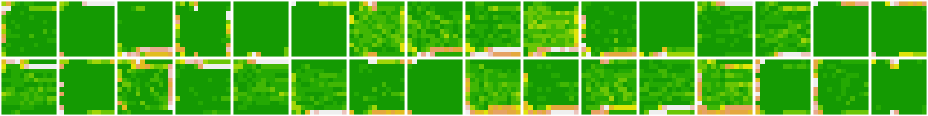
We will plot a complete visualization of all activations in the network and plot every channel in each eight activation maps.

dir.create("Fashion\_activations")

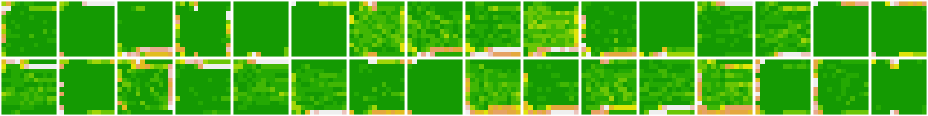
image\_size <- 58  
images\_per\_row <- 16  
  
for (i in 1:8) {   
 layer\_activation <- activations[[i]]  
 layer\_name <- model$layers[[i]]$name  
   
 n\_features <- dim(layer\_activation)[[4]]  
 n\_cols <- n\_features %/% images\_per\_row  
   
 png(paste0("Fashion\_activations/", i, "\_", layer\_name, ".png"),   
 width = image\_size \* images\_per\_row,   
 height = image\_size \* n\_cols)  
 op <- par(mfrow = c(n\_cols, images\_per\_row), mai = rep\_len(0.02, 4))  
   
 for (col in 0:(n\_cols-1)) {  
 for (row in 0:(images\_per\_row-1)) {  
 channel\_image <- layer\_activation[1,,,(col\*images\_per\_row) + row + 1]  
 plot\_channel(channel\_image)  
 }  
 }  
  
 par(op)  
 dev.off()  
}



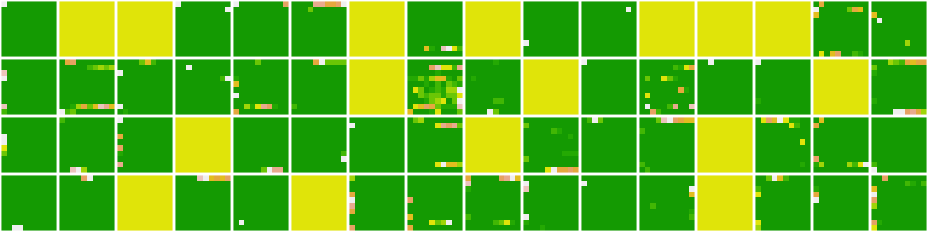
1\_conv2d\_1



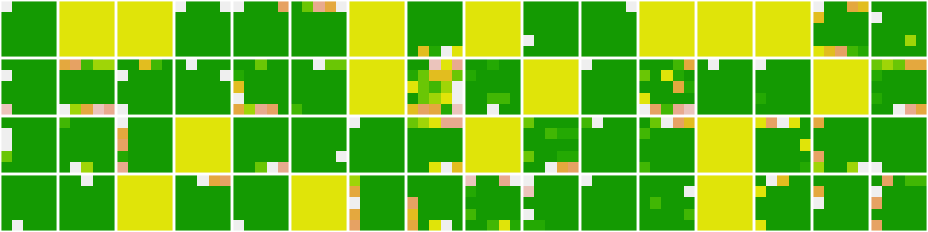
2\_max\_polling2d\_1



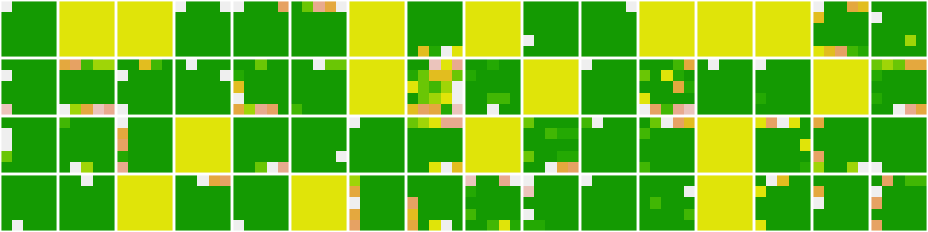
3\_dropout\_1



4\_conv2d\_2



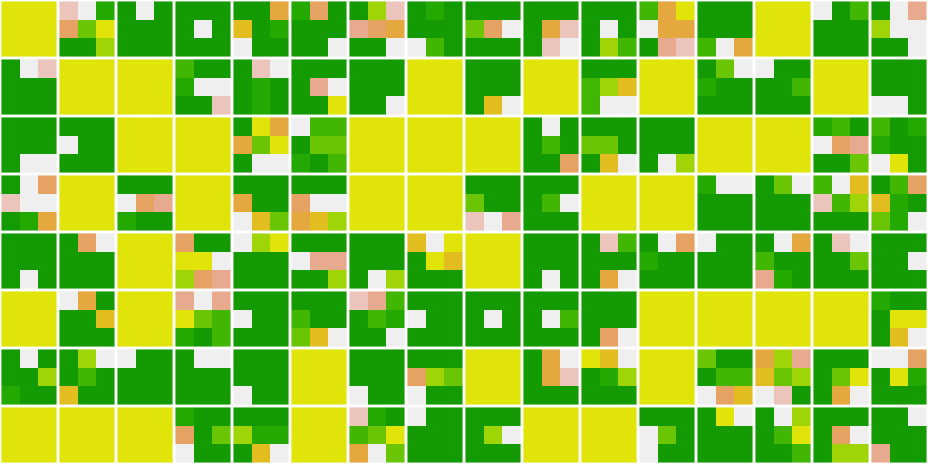
5\_max\_pooling2d\_2



6\_dropout\_2



7\_conv2d\_3



8\_dropout\_3

## Model Evaluation

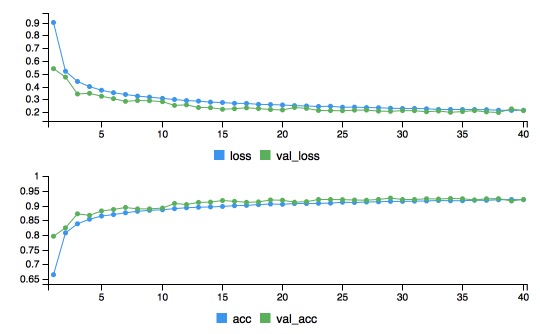
Model\_Performance <- model %>% evaluate(  
 Test\_X, Test\_Y, verbose = 0  
)  
cat('Test loss:', Model\_Performance[[1]], '\n')

## Test loss: 0.2090449

cat('Test accuracy:', Model\_Performance[[2]], '\n')

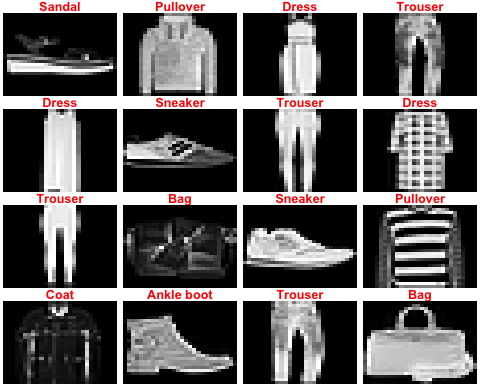
## Test accuracy: 0.9185

Using the ConvNets model, we have achieved an accuracy of 91.85% and loss is 20%, up from the previous model’s accuracy of 88.11% and loss of 34.15%.



## Visualizing the Model Predictions

# Visulization 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")  
}



# Summary

By using the ConvNets model, we have achieved an accuracy of 91.85%. It turns out our classifier does better than the Kaggle’s best baseline reported [here](http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/), 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. By using the pooling, dropout method and choosing the right loss function and optimizer are the key points to the success. Tuning the model and hyper-parameters is very important to improve the accuracy. Next phase, we may use the pre-trained the model to do feature extraction and fine-tuning.