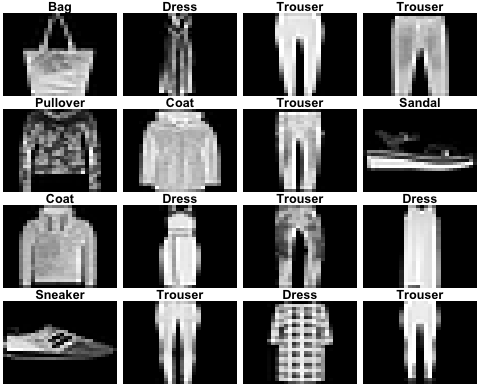
ConvNets\_FashionMNIST\_Visual.R

my\_macbook

Tue Jun 26 23:42:04 2018

library(knitr)  
opts\_chunk$set(message = FALSE, warning = FALSE, cache = TRUE, cache.lazy = FALSE)  
options(width = 120, dplyr.width = 120)  
library(ggplot2)  
theme\_set(theme\_light())  
  
  
library(readr)  
library(keras)  
  
  
# 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)   
  
  
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))  
}  
  
# Plot images from the training set  
par(mfrow=c(4, 4), mar=c(0, 0.2, 1, 0.2))  
for (i in 1:16) {  
 n\_row <- i \* 10  
 plot\_image(Train\_X[n\_row, , , 1],  
 Fashion\_Labels[as.numeric(TrainSetData[n\_row, 1] + 1)])  
}



# 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 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)

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
## 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  
## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# 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)  
)  
  
# Saving the model for visulization  
  
model %>% save\_model\_hdf5("CNN\_Fashion.h5")  
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  
## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# 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)  
  
# 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])  
}  
  
  
dir.create("Fashion\_activations")

## Warning in dir.create("Fashion\_activations"): 'Fashion\_activations' already exists

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

## Test loss: 0.2159092

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

## Test accuracy: 0.9208

# 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")  
}

