Trash image recognition experiments

Emil Westin

## Available trash datasets

* [Trashnet](https://github.com/garythung/trashnet)
* [Trash Annotations in Context](http://tacodataset.org/) : currently 1500 images with 4784 annotations

## Data

I have downloaded from [Trashnet](https://github.com/garythung/trashnet) containing 2527 images of six classes: glass, paper, cardboard, plastic, metal, and trash.

* 501 glass
* 594 paper
* 403 cardboard
* 482 plastic
* 410 metal
* 137 trash

### Libraries used

#tensorflow::install\_tensorflow(extra\_packages='pillow')  
library(keras)  
library(tensorflow)  
library(tidyverse)

### Importing data:

getwd()

## [1] "C:/Users/emiwes/Desktop"

setwd(dir = "C:/Users/emiwes/Desktop")  
base\_dir <- "C:/Users/emiwes/Downloads/dataset-resized/dataset-resized"  
fnames <- list.files(base\_dir, full.names = TRUE,recursive = T,include.dirs = F)  
n <- fnames %>% length()

### Image preprocessing in keras

The images are stored as tensors of shape (samples, height, width, color\_depth). In this case (n, 150, 150, 3), i.e. 3 colors.

Consider the following example:

# load the first file name (cardboard1.jpg) with size 150x150 pixels  
tt <- image\_load(fnames[1], target\_size = c(150, 150) )  
tt

## <PIL.Image.Image image mode=RGB size=150x150 at 0x280EA0B8>

We can convert it to array as follows:

tt2 <- image\_to\_array(tt)  
# 150 x 150 x 3   
dim(tt2)

## [1] 150 150 3

# shows 3 color values for pixel (1,1):  
tt2[1,1,]

## [1] 242 202 167

Neural networks prefer to deal with small input numbers. As such, it is typical to rescale the pixel values to a [0,1] interval by dividing the pixels by 255. The array\_reshape function does this and says to have the same dimension as before (150, 150, 3):

tt3 <- array\_reshape(tt2/255, c(150, 150, 3))  
dim(tt3)

## [1] 150 150 3

tt3[1,1,]

## [1] 0.9490196 0.7921569 0.6549020

In general, we can do this procedure for a large number of pictures.

data <- array(0, c(n,150,150,3))  
for(i in 1:n){  
 img\_path <- fnames[[i]]  
 # Convert it to an array with shape (150, 150, 3)  
 img <- image\_load(img\_path, target\_size = c(150, 150))  
 img\_array <- image\_to\_array(img)  
 data[i,,,] <- array\_reshape(img\_array/255, c(150, 150, 3))  
}  
# it is a good idea to save this since it can take some time to run:  
# save(data, file="img\_arr.RData")

After running the code above and saving, the data can be re-loaded as:

# LOADS THE DATA FILE WITH ThE IMAGE ARRAYS  
load(file = "C:/Users/emiwes/Desktop/img\_arr.RData")

We can plot any image from the tensor like this:

plot(as.raster(data[1,,,]))



## Split data into train/test/validation

#labs <- gsub("(.\*)/([a-z0-9]+.jpg)", "\\2", fnames) # only filename  
labs <- gsub(".\*/([a-z]+/[a-z0-9]+.jpg)", "\\1", fnames) # including directory  
labs <- as\_tibble(labs)  
labs %>% head() # this contains the names of the files

## # A tibble: 6 x 1  
## value   
## <chr>   
## 1 cardboard/cardboard1.jpg   
## 2 cardboard/cardboard10.jpg   
## 3 cardboard/cardboard100.jpg  
## 4 cardboard/cardboard101.jpg  
## 5 cardboard/cardboard102.jpg  
## 6 cardboard/cardboard103.jpg

class\_name <- gsub("(.\*)/([a-z]+)([0-9]+).jpg", "\\2", fnames)   
  
labs$class\_name <- class\_name  
labs$class\_name <- factor(labs$class\_name)  
  
binarized <- model.matrix( value ~ 0 + ., data = labs, contrasts.arg = list(contrasts=F) )

## Warning in model.matrix.default(value ~ 0 + ., data = labs, contrasts.arg =  
## list(contrasts = F)): variable 'contrasts' is absent, its contrast will be  
## ignored

labs <- cbind(id=labs$value, as\_tibble(binarized))   
  
colnames(labs) <- gsub("class\_name","", colnames(labs) )  
labs %>% head(5)

## id cardboard glass metal paper plastic trash  
## 1 cardboard/cardboard1.jpg 1 0 0 0 0 0  
## 2 cardboard/cardboard10.jpg 1 0 0 0 0 0  
## 3 cardboard/cardboard100.jpg 1 0 0 0 0 0  
## 4 cardboard/cardboard101.jpg 1 0 0 0 0 0  
## 5 cardboard/cardboard102.jpg 1 0 0 0 0 0

Lets say we split the data into 60% training and 40% testing.

set.seed(40)  
smp\_size <- floor(0.6 \* n )  
train\_ind <- sample(seq\_len(n), size = smp\_size)  
  
train\_images <- data[train\_ind,,,]  
test\_images <- data[-train\_ind,,,]  
  
train\_labels <- labs[train\_ind,]  
test\_labels <- labs[-train\_ind,]  
  
dim(train\_labels)[1] + dim(test\_labels)[1] == n

## [1] TRUE

dim(train\_images)

## [1] 1516 150 150 3

## Import images with flow\_images

train\_labels <- labs[train\_ind,]  
  
# split half of remaining indices into test and validation  
tmp <- labs[-train\_ind,]  
smp\_size <- floor(0.5 \* nrow(tmp) )  
set.seed(40)  
ind <- sample(seq\_len(nrow(tmp)), size = smp\_size)  
test\_labels <- tmp[ind,]  
val\_labels <- tmp[-ind,]  
  
nrow(val\_labels) + nrow(test\_labels) + nrow(train\_labels) == n

## [1] TRUE

#paste0(base\_dir, "/", gsub("([a-z+])/[a-z0-9]+.jpg", "\\1" , train\_labels$id )) %>% view()  
#?flow\_images\_from\_dataframe  
# https://www.kaggle.com/product-feedback/141059  
train\_datagen <- image\_data\_generator(rescale = 1/255)   
validation\_datagen <- image\_data\_generator(rescale = 1/255)  
test\_datagen <- image\_data\_generator(rescale = 1/255)  
  
  
train\_generator <- flow\_images\_from\_dataframe(  
 directory = base\_dir,  
 dataframe = train\_labels,  
 generator = train\_datagen,  
 x\_col = "id",  
 y\_col = list("cardboard","glass","metal","paper","plastic","trash") ,  
 batch\_size = 6,  
 seed = 40,  
 shuffle = TRUE,  
 class\_mode = "other"  
)  
  
test\_generator <- flow\_images\_from\_dataframe(  
 directory = base\_dir,  
 dataframe = test\_labels,  
 generator = test\_datagen,  
 x\_col = "id",  
 y\_col = list("cardboard","glass","metal","paper","plastic","trash") ,  
 batch\_size = 6,  
 seed = 40,  
 shuffle = TRUE,  
 class\_mode = "other"  
)  
  
validation\_generator <- flow\_images\_from\_dataframe(  
 directory = base\_dir,  
 dataframe = val\_labels,  
 generator = validation\_datagen,  
 x\_col = "id",  
 y\_col = list("cardboard","glass","metal","paper","plastic","trash") ,  
 batch\_size = 6,  
 seed = 40,  
 shuffle = TRUE,  
 class\_mode = "other"  
)

k <- c(3,3)  
model <- keras\_model\_sequential() %>%  
 layer\_conv\_2d(filters = 32, kernel\_size = k, activation = "relu", input\_shape = c(150,150,3) ) %>%  
 layer\_max\_pooling\_2d(pool\_size = c(2,2)) %>%  
 layer\_conv\_2d(filters = 64, kernel\_size = k, activation = "relu") %>%  
 layer\_max\_pooling\_2d(pool\_size = c(2,2)) %>%  
 layer\_conv\_2d(filters = 128, kernel\_size = k, activation = "relu") %>%  
 layer\_max\_pooling\_2d(pool\_size = c(2,2)) %>%  
 layer\_conv\_2d(filters = 6, kernel\_size = k, activation = "relu") %>%  
 layer\_max\_pooling\_2d(pool\_size = c(2,2)) %>%  
 layer\_flatten() %>%   
 layer\_dense(units = 512, activation = "relu") %>%  
 layer\_dense(units = 6, activation = "softmax")

summary(model)

## Model: "sequential"  
## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
## Layer (type) Output Shape Param #   
## ================================================================================  
## conv2d\_3 (Conv2D) (None, 148, 148, 32) 896   
## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
## max\_pooling2d\_3 (MaxPooling2D) (None, 74, 74, 32) 0   
## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
## conv2d\_2 (Conv2D) (None, 72, 72, 64) 18496   
## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
## max\_pooling2d\_2 (MaxPooling2D) (None, 36, 36, 64) 0   
## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
## conv2d\_1 (Conv2D) (None, 34, 34, 128) 73856   
## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
## max\_pooling2d\_1 (MaxPooling2D) (None, 17, 17, 128) 0   
## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
## conv2d (Conv2D) (None, 15, 15, 6) 6918   
## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
## max\_pooling2d (MaxPooling2D) (None, 7, 7, 6) 0   
## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
## flatten (Flatten) (None, 294) 0   
## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
## dense\_1 (Dense) (None, 512) 151040   
## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
## dense (Dense) (None, 6) 3078   
## ================================================================================  
## Total params: 254,284  
## Trainable params: 254,284  
## Non-trainable params: 0  
## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Compilation step:

#?compile  
model %>%  
 compile(  
 loss = 'categorical\_crossentropy',  
 optimizer = optimizer\_rmsprop(lr = 1e-4),  
 metrics = c('accuracy')  
 )

Fitting the model:

# https://www.javaer101.com/en/article/24494572.html  
#   
# history <- model %>%  
# fit\_generator( train\_generator,   
# steps\_per\_epoch = 10,  
# epochs = 10,   
#   
# validation\_data = validation\_generator,   
# validation\_steps = 10 )