Plane or Car? An Application of Neural Networks

Benedict Toh

2024-06-13

Introduction

This project aims to build a neural network model that classifies images into 2 categories: plane or car. To begin, I engineered my own dataset by using images downloaded from Google. Thereafter, I processed the data collected for training. I built a neural network using the Keras package and evaluated the model with the Accuracy as my metric.

Libraries

```
library(EBImage)
library(keras3)
library(tensorflow)

to_categorical <- tensorflow::tf$keras$utils$to_categorical
```

Setting up directories

```
path_root <-  "."
path_planes <- file.path(path_root, "planes")
path_cars <- file.path(path_root, "cars")
```

Load Image Data

We do so by writing a user defined function that goes through the specified path and returns a list of images.

```
read_images <- function(path, pattern = "\\.jpg$") {
  files <- list.files(path, pattern = pattern, full.names = TRUE)
  pics <- list()
  for (i in 1:length(files)) {
    tryCatch({
      pics[[i]] <- readImage(files[i])
    }, error = function(e) {
      message("Error reading file ", files[i], ": ", e)
    })
  }
  return(pics)
}
```

Here, we load our images and combine both plane and car images into a single list. We then remove images that are unable to be read (null).

```
planes_pic <- read_images(path_planes)
```

```
## Error reading file ./planes/p320.jpg: Error in readJPEG(x, ...): JPEG decompression error: Not a JPE
```

```
cars_pic <- read_images(path_cars)

mypic <- c(planes_pic, cars_pic)

mypic <- Filter(Negate(is.null), mypic)
```

Data Preparation

Moving on, we convert the images into grayscale.

Why?

1. To reduce computational power required.
2. Given the small dataset of about 300-400 images, reducing to grayscale reduces noise which would reduce overfitting.

We then convert all the images to the same dimensions.

```
for (i in 1:length(mypic)) {
  mypic[[i]] <- channel(mypic[[i]], "gray")
}

for (i in 1:length(mypic)) {
  mypic[[i]] <- resize(mypic[[i]], 64, 64)
}
```

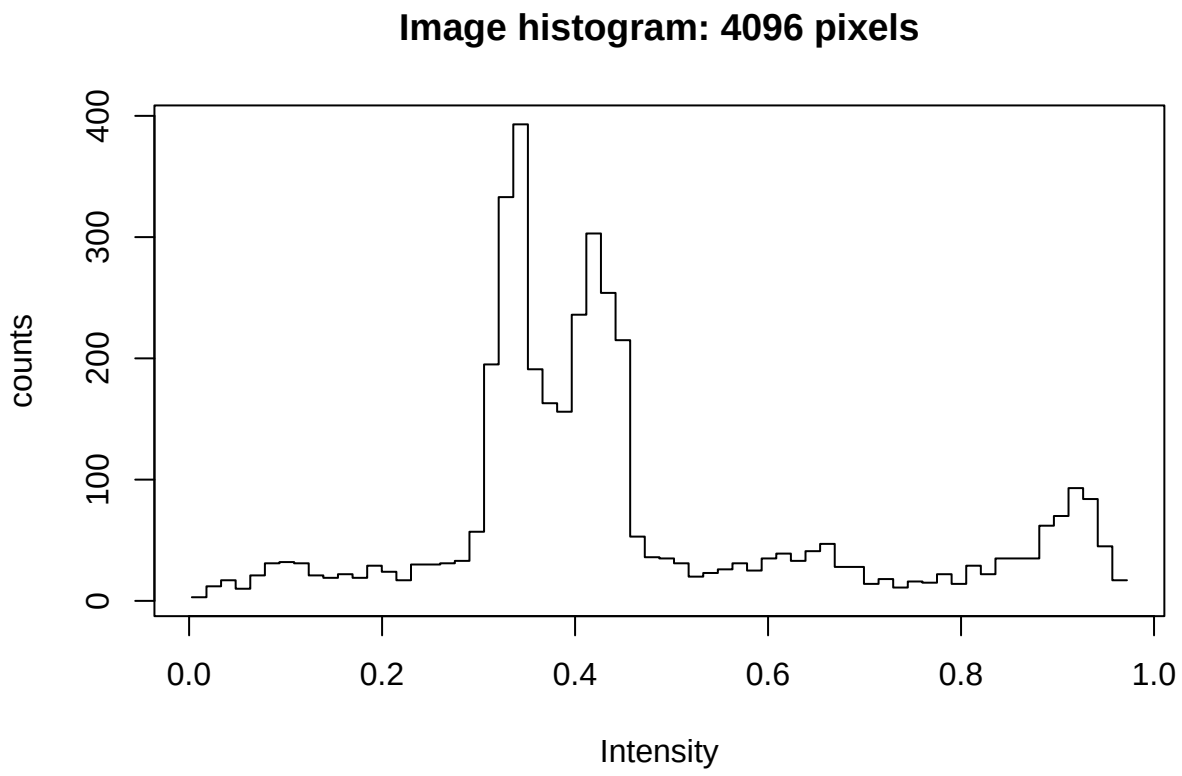
Exploring the data

```
print(mypic[[1]])
```

```
## Image
##   colorMode    : Grayscale
##   storage.mode : double
##   dim          : 64 64
##   frames.total : 1
##   frames.render: 1
##
## imageData(object)[1:5,1:6]
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
## [1,] 0.7247284 0.6915499 0.6589148 0.6723212 0.7270868 0.7145623
## [2,] 0.7075061 0.6780842 0.6808517 0.6647209 0.7207922 0.7509156
## [3,] 0.7154817 0.6712619 0.6781457 0.6565238 0.7179943 0.7882318
## [4,] 0.7160858 0.6867050 0.6638831 0.6608465 0.7108967 0.7621879
## [5,] 0.7089904 0.6794826 0.6677026 0.6763318 0.7393252 0.7979687
```

```
display(mypic[[300]])
```

```
hist(mypic[[11]])
```




```
str(mypic[[15]])
```

```
## Formal class 'Image' [package "EBImage"] with 2 slots
##   ..@ .Data    : num [1:64, 1:64] 1 1 1 1 1 1 1 1 1 1 ...
##   ..@ colormode: int 0
##   ..$ dim: int [1:2] 64 64
```

After exploration, we convert the images into matrices by reshaping - obtain a vector for each image.

```
for (i in 1:length(mypic)) {
  mypic[[i]] <- array_reshape(mypic[[i]], c(64, 64, 1))
}
```

Train-Test Split

We designate the last 10 images for both planes and cars to be the test set. An equal number of both classes was chosen to maintain class balance.

```
trainx <- NULL

# planes - 1 to 315/10, 316 to 325
for (i in 1:315) {
  trainx <- rbind(trainx, mypic[[i]])
}
```

```r
# cars - 326 to 731/10, 732 to 741
for (i in 326:731) {
  trainx <- rbind(trainx, mypic[[i]])
}

testx <- rbind(do.call(rbind, mypic[316:325]), do.call(rbind, mypic[732:741]))

# Create labels: 0 - plane, 1 - car
trainy <- as.integer(c(rep(0, times = 315), rep(1, times = 406)))
testy <- as.integer(c(rep(0, times = 10), rep(1, times = 10)))
num_classes <- 2  # Assuming binary classification

# One-hot encoding
trainLabels <- to_categorical(as.integer(trainy), num_classes = as.integer(num_classes))
testLabels <- to_categorical(as.integer(testy), num_classes = as.integer(num_classes))
```

Model Training

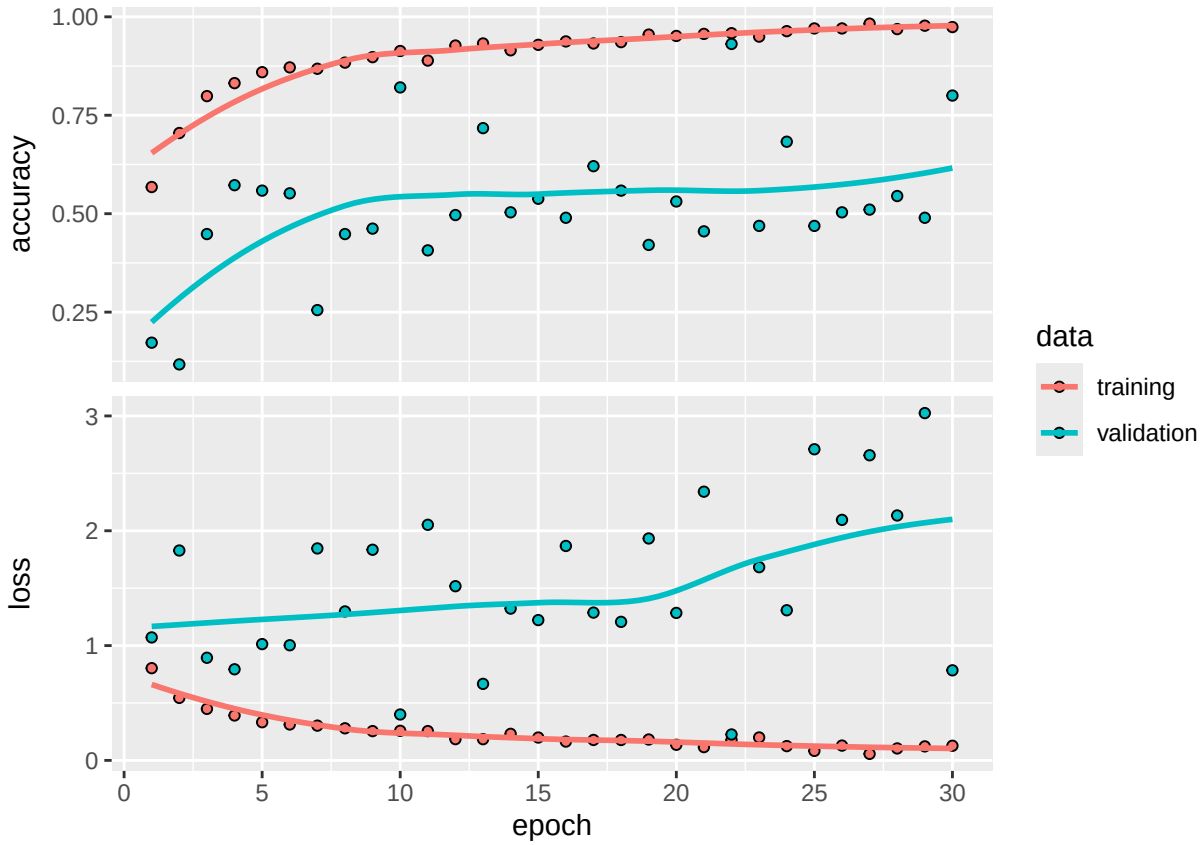
```r
set.seed(42)

model <- keras_model_sequential()
model %>%
  layer_dense(units = 256, activation = 'relu', input_shape = c(4096)) %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dense(units = 64, activation = 'relu') %>%
  layer_dense(units = 2, activation = 'softmax')

# Compile the model using the correct function
model %>% compile(
  loss = 'binary_crossentropy',
  optimizer = optimizer_rmsprop(),
  metrics = c('accuracy')
)

# Fit the model
history <- model %>%
  fit(trainx,
      trainLabels,
      epochs = 30,
      batch_size = 8,
      validation_split = 0.2)

plot(history)
```




```
# Extract final accuracy and validation accuracy
final_train_acc <- tail(history$metrics$accuracy, 1)
final_val_acc <- tail(history$metrics$val_accuracy, 1)
```

```
cat("Final Training Accuracy:", final_train_acc, "\n")
```

```
## Final Training Accuracy: 0.9739583
```

```
cat("Final Validation Accuracy:", final_val_acc, "\n")
```

```
## Final Validation Accuracy: 0.8
```

Evaluating the Model

```
# evaluation on train data
tr_eval <- model %>% evaluate(trainx, trainLabels)
pred <- model %>% predict(trainx)
predicted_classes <- apply(pred, 1, which.max) - 1
```

```
cat("Training Accuracy:", round(tr_eval$accuracy,3), "\n")
```

```
## Training Accuracy: 0.86
```

Confusion Matrix (Training)

```
table(predicted = predicted_classes, actual = trainy)
```

```
##           actual
## predicted   0   1
##         0 243  29
##         1  72 377
```

```
# evaluation on test data
te_eval <- model %>% evaluate(testx, testLabels)
pred <- model %>% predict(testx)
predicted_classes <- apply(pred, 1, which.max) -1
```

```
cat("Test Accuracy:", round(te_eval$accuracy,3), "\n")
```

```
## Test Accuracy: 0.9
```

Confusion Matrix (Test)

```
table(predicted = predicted_classes, actual = testy)
```

```
##           actual
## predicted  0  1
##         0  8  0
##         1  2 10
```