Identify Lesions within CT Scans Using Convolutional Neural Networks

Graham Chickering

November 12, 2020

Abstract

Convolutional Networks are a specific type of Neural Networks that have shown to be particularly effective at being able to identify distinct objects within images. This technique can be used to identify different types of anomalies, such as lesions, within medical images as well as detect other sorts of tumors or cancers. In theory this type of network is designed based on how the human brain works and the idea that multiple levels of neurons are connected together in order to detect and identify images. In practice though, running and training these types of neural networks can be very computationally expensive and require large amounts of memory and processing capabilities if working with a very large dataset. Especially when working in R which has limited memory capabilities, trying to run and train this type of model can be very slow and ineffective. Thanks to developments in cloud computing and distributed data processing engines such as Google Cloud Storage and Apache Spark, being able to run and train these types of models within R can become much faster and more efficient when trying to analyze large amounts of data. While there is more work to be done, this project shows how to create an infrastructure that efficiently stores data and trains these models when working with the R Studio Environment.

Keywords: Convolutional Neural Networks, Apache Spark, Google Cloud Storage, Lesions, Healthcare

1 Introduction

It has been estimated that roughly 80% of healthcare data is unstructured data, which can come in the form of videos, sensor data, images, or text. Although hospitals and researchers used to have a hard time extracting insights from this type of data, with the recent advances that have been made in data science and handling big data, this has created new application areas within the healthcare industry in sectors such as genomics, drug trials, predicting patient health, and medical imagery. Medical imaging research in particular has made significant progress recently with researchers being able to use different machine learning algorithms to detect different types of lesions and cancers from CT and other types of scans. In particular the advancements of convolutional neural networks to identify different types of lesions and cancers is extremely promising to the medical community and can be used to potentially help doctors identify tumors or lesions that they might have missed, and reduce the amount of hours and amount of expertise required to view CT scans.

When working with medical imagery data sets though one of the first problems someone may run into is how to process and handle these large data sets. When trying to perform analysis on small and medium sized data sets within R, one would rarely run into complications that would be attributed to how R is loading and dealing with the data itself. But what happens when one moves from the world of medium sized data to the world of Big Data and large data sets? While many of us have probably been able to read in and load our data for projects into the R Studio environment without issues and without having to worry about whether the entirety of our data can even be loaded in, one may begin to run into complications the larger the data set becomes. By default, R loads all data into memory and while memory size depends slightly on your computer's configuration settings under any circumstances one cant have more than 2,147,483,647 rows or columns, which is roughly equivalent to 2 GB of memory that R is using. (see `http://www.columbia.edu/~sjm2186/EPIC_R/EPIC_R_BigData.pdf`). If you do end up crossing into the threshold where R can no longer store all the data in an effective way, there are multiple potential solutions in the forms of choosing random subsets of the data, buying a computer with larger memory, or use parallelization and using multiple clusters to

perform the analysis. It is this solution of utilizing distributed computing, via the sparklyr package, that will allow us to perform computationally expensive analysis on large data sets.

On top of the issue of trying to run analysis on large data sets in R itself, is the issue of how to best store and load the original information and data. Often data sets are small enough that they can be stored on your local computer in a folder that is then uploaded into the R Studio Environment itself, but what should one do as the size of the data set substantially increases and one no longer wants to store large data sets directly on their machine. One solution to this problem is to take advantage of a cloud computing service and store the data directly in the cloud, freeing up space and memory on your personal computer. By storing the data on a cloud computing service, this can become especially useful when a project begins to get scaled up whether that is through adding new members to work on the project or when more and more data gets added to the project. In this project I will take advantage of Google Cloud Storage to store my data, which will look and store my data on the cloud without having to take up any space on my local machine.

By combining Google Cloud Storage with Apache Spark this will allow me to create a large data set consisting of medical images. This data set will then be used to train and create a convolutional neural network that will try to identify the lesions themselves within the images.

This project will allow me to answer the questions of what is the best way to store large datasets and perform computationally expensive analysis on those data sets? How does one handle and process images so that analysis can be performed on them, and how effective are convolutional neural networks at identifying anomalies within images?

1.1 Work I have completed so far

```
library(tensorflow)
library(tfdatasets)
library(keras)
library(cloudml)
```

```
library(readr)
library(tidyverse)
library(googleCloudStorageR)
library(googleAuthR)
library(sparklyr)
library(magick)
library(ggplot2)
library(cowplot)
library(reprex)
library(imager)
```

This is where we set up a connection between Google Cloud Storage and R

```
gcs_auth(json_file="account_credentials.json")


gcs_get_bucket("medical_images")
```

```
## ==Google Cloud Storage Bucket==
## Bucket:          medical_images
## Project Number:  1048020973776
## Location:        US
## Class:           STANDARD
## Created:         2020-11-07 18:56:59
## Updated:         2020-11-07 18:56:59
## Meta-generation: 1
## eTag:            CAE=
```

```
gcs_global_bucket("medical_images")
```

```
## Set default bucket name to 'medical_images'
```

```r
gcs_get_global_bucket()
```

```
## [1] "medical_images"
```

```r
objects <- gcs_list_objects()

all_images<-as.data.frame(objects[-c(1,2),-3])

image<-gcs_get_object(objects$name[[9]], saveToDisk="patient0.png", overwrite = TRUE)
```

```
## 2020-11-12 00:50:23 -- Saved archive/minideeplesion/000001_01_01/109.png to patient0
```

```r
medical_data<-gcs_get_object(objects$name[[2]])

#data_dir <- gs_data_dir("gs://medical_images/archive")
#medical_data<-read_csv(file.path("gs://medical_images/archiveDL_info.csv"))
#medical_data <- read_csv(file.path(data_dir, "DL_info.csv"))

path<-"gs://medical_images/archive/minideeplesion/"
path2<-"archive/minideeplesion/"

medical_data<-medical_data %>%
   janitor::clean_names() %>%
  mutate(first_part=substr(file_name,1,12), second_part=substr(file_name,14,20),
         file_path=paste0(path,first_part,paste("/", second_part, sep="")),
         object_path= paste0(path2,first_part,paste("/", second_part, sep="")),
         radius=substr(lesion_diameters_pixel,1,6),
         lesion_type=case_when(
           coarse_lesion_type == -1 ~"Unknown",
           coarse_lesion_type == 1 ~ "Bone",
           coarse_lesion_type == 2 ~ "Abdomen",
```

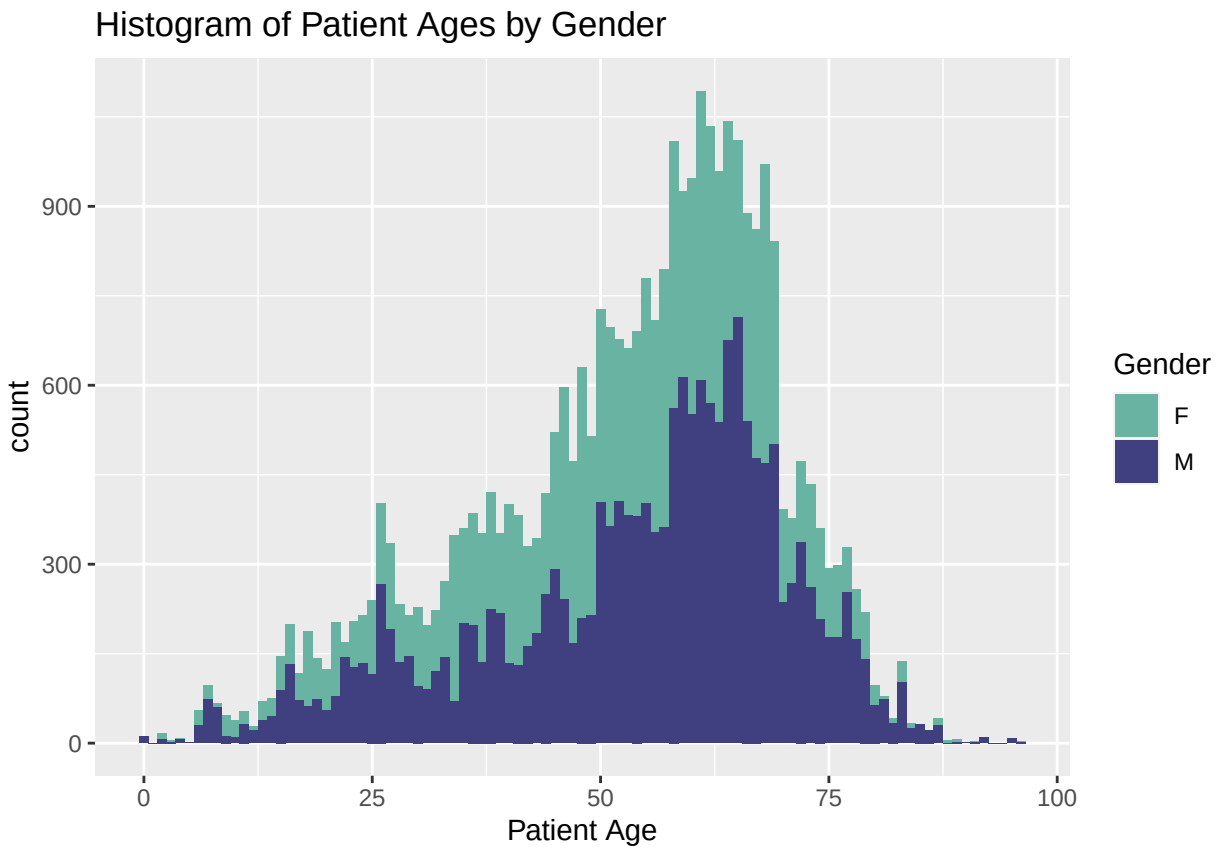
```
            coarse_lesion_type == 3 ~ "Mediastinum",
            coarse_lesion_type == 4 ~ "Liver",
            coarse_lesion_type == 5 ~"Lung",
            coarse_lesion_type == 6 ~ "Kidney",
            coarse_lesion_type == 7 ~ "Soft tissue",
            coarse_lesion_type == 8 ~ "Pelvis"
          ) ) %>%
  select(-first_part,-second_part) %>% arrange(file_name)
```

1.2 Exploratory Data Analysis

-These graphs will show the characteristics of the patients and types of legions of the patients in the study

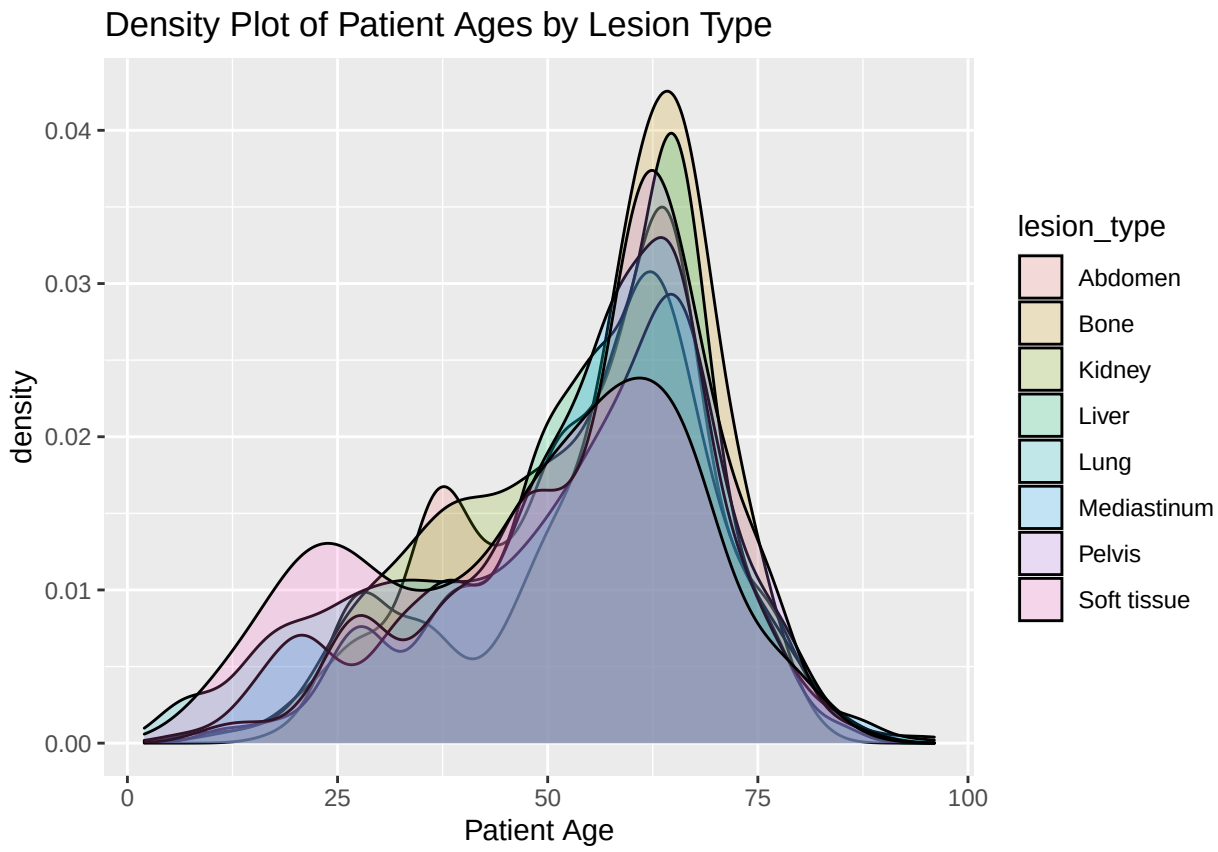
```
patients <- medical_data %>%
  filter(patient_age < 120) %>%
  mutate(radius=round(as.numeric(radius),0)) %>%
  rename(Gender = patient_gender)

ggplot(data = patients, aes(x = patient_age, fill = Gender)) +
  geom_histogram(binwidth = 1) +
  scale_fill_manual(values = c("#69b3a2", "#404080")) +
  labs(x = "Patient Age", title = "Histogram of Patient Ages by Gender")
```

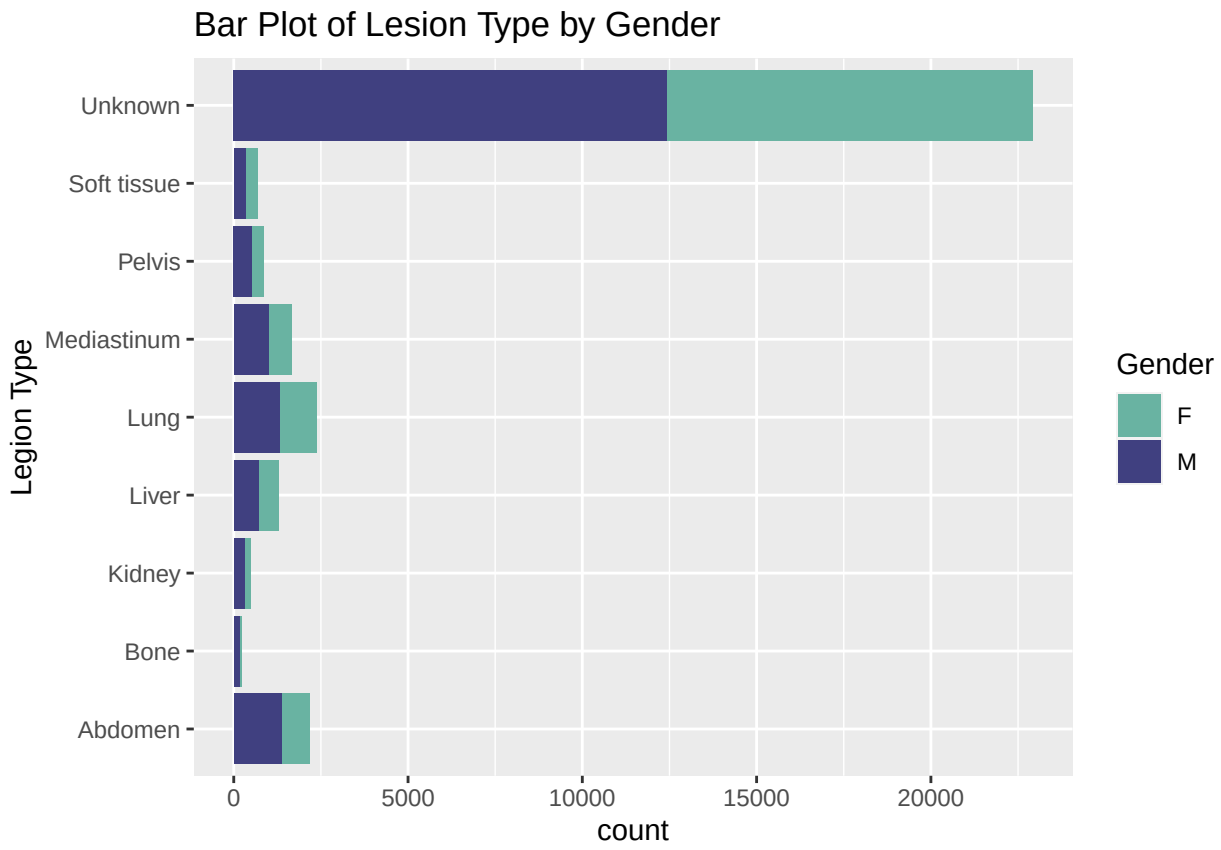



```
patients2<-patients %>% filter(lesion_type != "Unknown")


ggplot(patients2, aes(x=patient_age, fill = lesion_type)) +
  geom_density(alpha = 0.2) +
  labs(x = "Patient Age", title = "Density Plot of Patient Ages by Lesion Type")
```

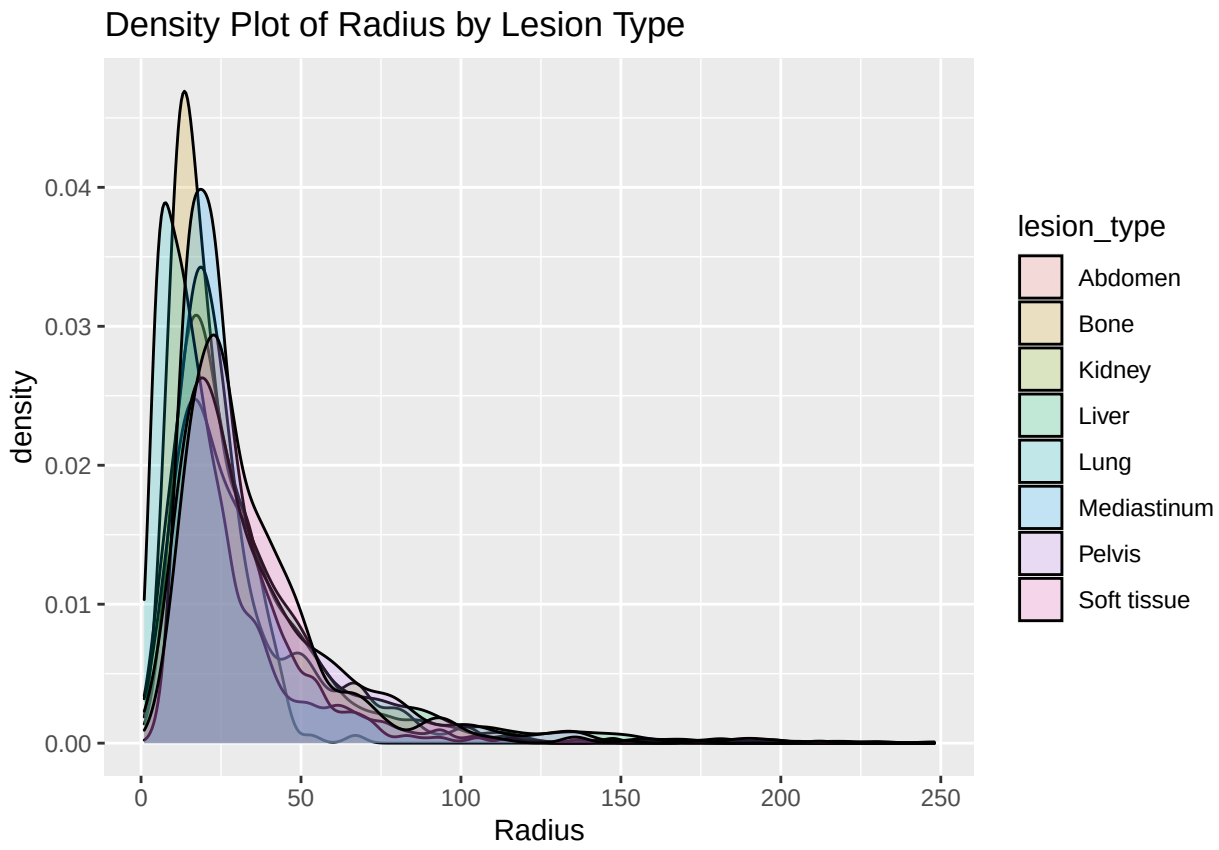



```r
ggplot(data = patients, aes(x = lesion_type, fill=Gender)) +
  geom_bar() +
  coord_flip()+
  scale_fill_manual(values = c("#69b3a2", "#404080")) +
  labs(x = "Legion Type", title = "Bar Plot of Lesion Type by Gender")
```




```
patients2<-patients %>% filter (radius<250 & lesion_type != "Unknown")


ggplot(data = patients2, aes(x = radius,fill=lesion_type)) +
  geom_density(alpha=0.2) +
  labs(x = "Radius", title = "Density Plot of Radius by Lesion Type")
```



1.3 This to to print out some of the images of the actual lesions

-This needs to be changed, meant to have a box around where the legion is but currently not working -Also planning to add multiple of these types of images

```
image<-gcs_get_object(objects$name[[9]], saveToDisk="patient0.jpeg", overwrite = TRUE)
```

```
## 2020-11-12 00:50:28 -- Saved archive/minideeplesion/000001_01_01/109.png to patient0
```

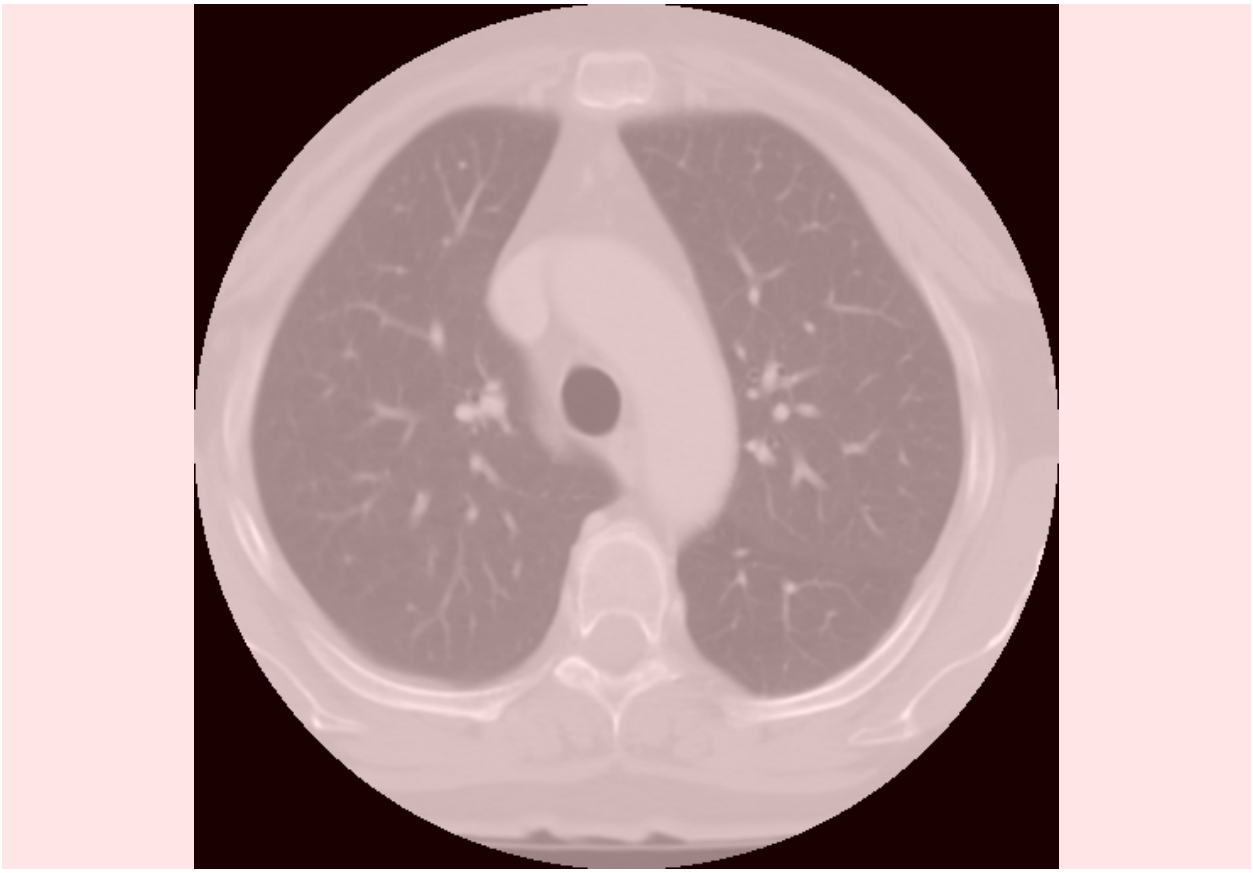
```
img<-image_read("patient0.jpeg")
img<-image_normalize(img)


image_info(img)
```

```
## # A tibble: 1 x 7
```

```
##   format width height colorspace matte filesize density
##   <chr>  <int>  <int> <chr>      <lgl>    <int> <chr>
## 1 PNG      512    512 Gray       FALSE        0 72x72
```

```
image<-ggdraw() +
  draw_image(
    img, scale = 1
  ) + geom_rect(aes(xmin = 0, xmax = 20, ymin = 0, ymax = 20), alpha = 1/10,fill = "red"
image
```



1.4 This is where I will begin my work with Spark now that all the data is available from Google Cloud Storage

-This is not complete

```r
#spark_install()

sc <- spark_connect(master = "local", version = "2.3")



medical <- copy_to(sc, medical_data)
images<-copy_to(sc,objects )
medical
```

```
## # Source: spark<medical_data> [?? x 22]
##    file_name patient_index study_index series_id key_slice_index
##    <chr>             <dbl>       <dbl>     <dbl>           <dbl>
##  1 000001_0~             1           1         1             109
##  2 000001_0~             1           2         1              14
##  3 000001_0~             1           2         1              17
##  4 000001_0~             1           3         1              88
##  5 000001_0~             1           4         1              17
##  6 000002_0~             2           1         1             162
##  7 000002_0~             2           1         1             176
##  8 000002_0~             2           2         1              50
##  9 000002_0~             2           2         1              52
## 10 000002_0~             2           2         1              65
## # ... with more rows, and 17 more variables: measurement_coordinates <chr>,
## #   bounding_boxes <chr>, lesion_diameters_pixel <chr>,
## #   normalized_lesion_location <chr>, coarse_lesion_type <dbl>,
## #   possibly_noisy <dbl>, slice_range <chr>, spacing_mm_px <chr>,
## #   image_size <chr>, dicom_windows <chr>, patient_gender <chr>,
## #   patient_age <dbl>, train_val_test <dbl>, file_path <chr>,
## #   object_path <chr>, radius <chr>, lesion_type <chr>
```

```
spark_web(sc)
```

```
spark_disconnect(sc)
```

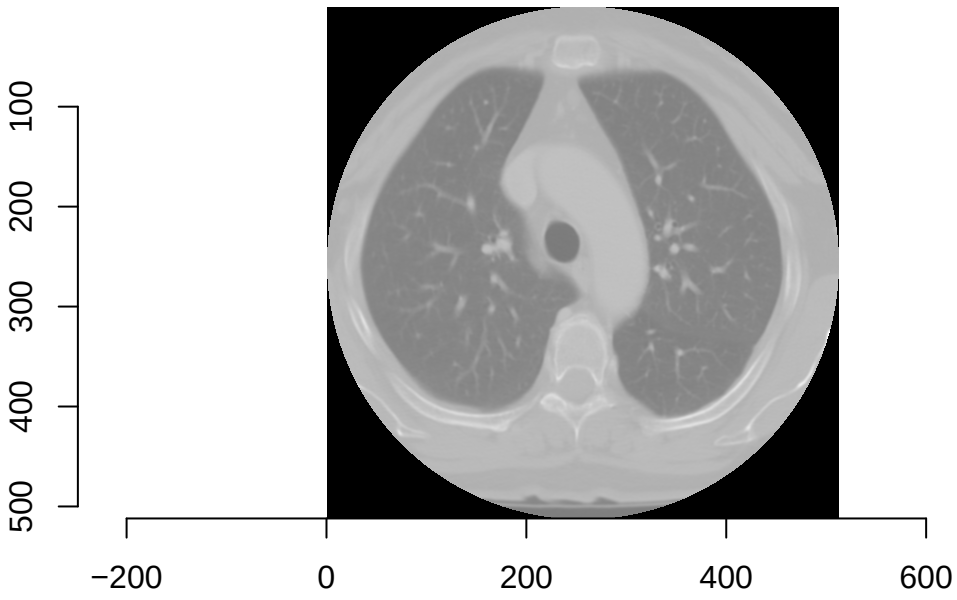
1.5 Data Preprocessing To Set Up Testing and Training Sets of the medical images

-This is not complete

```
im <- load.image("patient0.jpeg") %>% grayscale()
```

```
## Warning in grayscale(.): Image appears to already be in grayscale mode
```

```
plot(im)
```



```r
dim(im)
```

```
## [1] 512 512   1   1
```

```r
x<-as.matrix(im)
```

```r
set.seed(123)
random <- sample(1:nrow(all_images), 0.8 * nrow(all_images)) # 80%: training data, 20%:
train <- all_images[random, ]
train2<- train %>% left_join(medical_data, by= c("name"="object_path"))

test <- all_images[-random, ]
test2<- test %>% left_join(medical_data, by= c("name"="object_path"))
#image2<-gcs_get_object(train£name[[1]], saveToDisk="patient1.jpeg", overwrite = TRUE)
```

1.6 This is where I will do more of the machine learning and convolution neural netowrks within Spark

```r
# data_dir <- gs_data_dir("gs://medical_images/archive/minideeplesion")
# images <- list.files(data_dir, pattern = ".png", recursive = TRUE)
# length(images)
#
# classes <- list.dirs(data_dir, full.names = FALSE, recursive = FALSE)
# classes
#
# list_ds <- file_list_dataset(file_pattern = paste0(data_dir, "/*/*"))
# list_ds %>% reticulate::as_iterator() %>% reticulate::iter_next()
# list_ds
#
# get_label <- function(file_path) {
```

```
#   parts <- tf$strings$split(file_path, "/")
#   parts[-2] %>%
#     tf$equal(classes) %>%
#     tf$cast(dtype = tf$float32)
# }
#
# decode_img <- function(file_path, height = 224, width = 224) {
#
#   size <- as.integer(c(height, width))
#
#   file_path %>%
#     tf$io$read_file() %>%
#     tf$image$decode_jpeg(channels = 3) %>%
#     tf$image$convert_image_dtype(dtype = tf$float32) %>%
#     tf$image$resize(size = size)
# }
#
# preprocess_path <- function(file_path) {
#   list(
#     decode_img(file_path),
#     get_label(file_path)
#   )
# }
#
# labeled_ds <- list_ds %>%
#   dataset_map(preprocess_path, num_parallel_calls = tf$data$experimental$AUTOTUNE)
#
# labeled_ds %>%
#   reticulate::as_iterator() %>%
#   reticulate::iter_next()
```

```
# prepare <- function(ds, batch_size, shuffle_buffer_size) {
#
#   if (shuffle_buffer_size > 0)
#     ds <- ds %>% dataset_shuffle(shuffle_buffer_size)
#
#   ds %>%
#     dataset_batch(batch_size) %>%
#     # `prefetch` lets the dataset fetch batches in the background while the model
#     # is training.
#     dataset_prefetch(buffer_size = tf$data$experimental$AUTOTUNE)
# }
#
# model <- keras_model_sequential() %>%
#   layer_flatten() %>%
#   layer_dense(units = 128, activation = "relu") %>%
#   layer_dense(units = 128, activation = "relu") %>%
#   layer_dense(units = 5, activation = "softmax")
#
# model %>%
#   compile(
#     loss = "categorical_crossentropy",
#     optimizer = "adam",
#     metrics = "accuracy"
#   )
#
# model %>%
#   fit(
#     prepare(labeled_ds, batch_size = 32, shuffle_buffer_size = 1000),
#     epochs = 5,
#     verbose = 2
```

```
#   )

# trial<-flow_images_from_directory(directory=data_dir,
#                             generator = image_data_generator(rescale=1/255),
#                             target_size=c(256,256), color_mode = "grayscale")
# trial
# data_dir <- gs_data_dir("gs://medical_images/archive/minideeplesion/000002_02_01/044
# trial<-image_load(path="gs://medical_images/archive/minideeplesion/000002_02_01", gr
```