DA 6813 Autonomous Chest X-Ray Diagnosis
The Betas

Sherry Lin & Kalea Sebesta

February 28, 2018

Abstract. In today's world health is a concern to every individual. The health care in many areas is limited to those who have access to it and/or the financial funds to support it. Technologies exists today that give doctor's insight into the human condition at hand and yet there is still subjectiveness that occurs. Creating an autonomous way to use chest X-ray imaging to diagnosis aliments without the need of a radiologist to read them, would benefit a vast amount of the population. Having algorithms that could identify subtle changes between images could help discover disease sooner and thus aid in the prognosis of the patient.

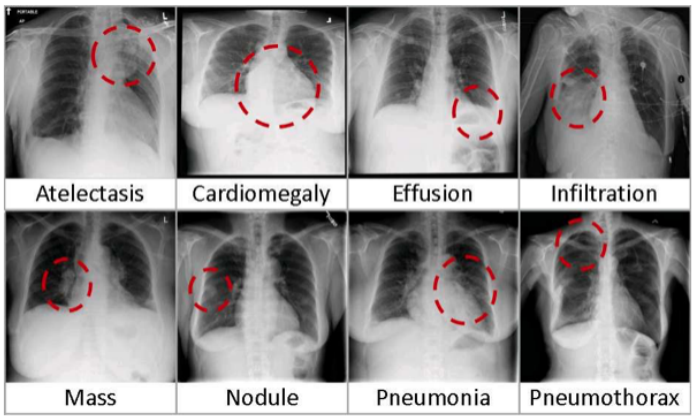

Fig. 1: Eight examples of the chest X-ray labeled diagnosis.

Contents

1 Introduction

Machine and deep learning are being leveraged in the medical arena specifically with image classification studies. Currently, there are many models which have accuracy levels around 95%. Such studies have focused on medical image analysis in CT scans, retinal imaging and even skin cancer image classification. There are many companies who's sole purpose is developing artificial intelligence (AI) in the arena of health care with the goal to help a wider population in discovering diagnosis quicker and at a decreased cost. This is a revolutionary field that can save lives by detecting critical ailments in their infancy and deploying the proper case before the ailment turns fatal.

1.1 Problem Statement

Developing a method for autonomously diagnosing aliments at a 90% accuracy rate would benefit all people and potentially reduce cost and time.

1.2 Questions

1. What is the most prominent aliment in the data set?
2. Which factors explain the diagnosis findings in the training data set images?
3. Can we predict disease diagnosis at the 90% accuracy rate?

2 Data

Source

The data source for this project was accessed through Kaggle (https://www.kaggle.com/nih-chest-xrays/data/data). The original images and CSV files have a primary resource from the NIH (National Institute of Health).

2.1 Quality/Volume

Within this data set there are 11 variables which include features about the image's original width, height, and pixel spacing. It also includes patient age, gender, and the view position that the X-ray was taken at. There are 15 classes of diseases and in some cases, multiple diseases per patient. There are two main CSV files that this project will be working with: File 1 is a CSV file that contains over 5,000 images that are classified for over 5,000 patients. This set will be treated as the training set. File 2 is a CSV file that contains over 112,000 images for over 30,000 patients. The 15 ailments in our data set are as follows:

Ailment Label	Definition
Cardiomegaly	Enlarged Heart
Effusion	Fluid in a Pleural Cavity
Nodule	Solid Overgrowth of Tissue
Atelactasis	Collapse or Incomplete Expansion of the Lung
Mass	Area of Consolidation
Consolidation	Fluid or Tissue fills the Lung
Fibrosis	"Scarring of the Lungs"; Respiratory Disease
Emphysema	Permanent enlargement of the airspaces
Hernia	Organ or fatty tissue squeezes through the fascia
Pneumothorax	Abnormal collation of air between the chest wall and lung
Pleural_Thickening	Lung Disease; Scarring that thickens the lungs
Edema	Excess fluid in lungs
Infiltration	Abnormal substance in lungs
Pneumonia	Lung Inflamation

2.2 Cleaning & Preprocessing

Some features in the labeled training data set will need to be processed prior to any analysis being conducted. Age for example is currently a categorical variable and will need to be converted to a numerical variable. Once the age was converted to a numerical value 2 observations became missing. Through investigation of these two observations imputation was conducted based on the original category. One entry was set as 13M which was recoded to 1 year after inspection and there was one observation that was labeled as 1D indicating that D may represent days and thus it was recoded as 0 year.

The 'finding_labels' contain all the diagnosis for the specific patient. This meant if there were multiple diagnosis for one patient the 'finding_labels' would contain a list of diagnosis separated by a pipe symbol. Therefore, NLP was utilized to split the finings into one of the 15 categories (specifically, the 1st finding).

Outliers were seen in the 'Patient.Age', and 'Follow.up..'. Any age that was over 100 was imputed with the median. The follow up appointments were heavily skewed, we chose to impute any values over 100 with the median. This approach however may not be the best since one could argue that with the ailments of diagnosis it is reasonable that a patient would have an increase number of follow up appointments. Without the domain expertise in this particular case we choose to impute with the median yet include the caveat that other ways may be better.

3 Analysis

Goals & Objectives

A long term objective would be to help identify slow changes occurring over the course of multiple chest X-rays to help doctors understand and treat patients. Ideally this project wants to achieve an accuracy rate above 90% with a CNN and also

explain how the factors in the label data set influence a diagnosis. The goal of this particular project was to accomplish two tasks, 1. inference with respect to the demographic factors of influences from the CSV file, 2. prediction and classification of new images.

3.1 Tools & Techniques

For this project R & Python were used to implement various modeling techniques. For preliminary analysis GBM, RandomForest, and SVM were used. Advanced methods such as CNN with TensorFlow and Keras were used for classification and prediction of new images, and XGBoost which is an extension of the gradient boosting tree algorithm that is designed to handle multilevel class responses.

3.2 Exploratory Analysis

In this section, we would like to explore the number of each disease in our data set, since our goal is to find the most common aliment in our data set and to predict the lung disease base on the X-Ray images. Through preliminary exploration of the data we found the distribution breakdown of age by gender, ailments by age and gender, simple and multiple ailment diagnosis, and frequency count of ailments as well as frequency count of ailments by age. It was discovered that our dataset was highly imbalanced. This makes sense considering many of the ailments in our dataset are decently rare. The below figure is a combination of the exploratory descriptive statistical images related to this data set.

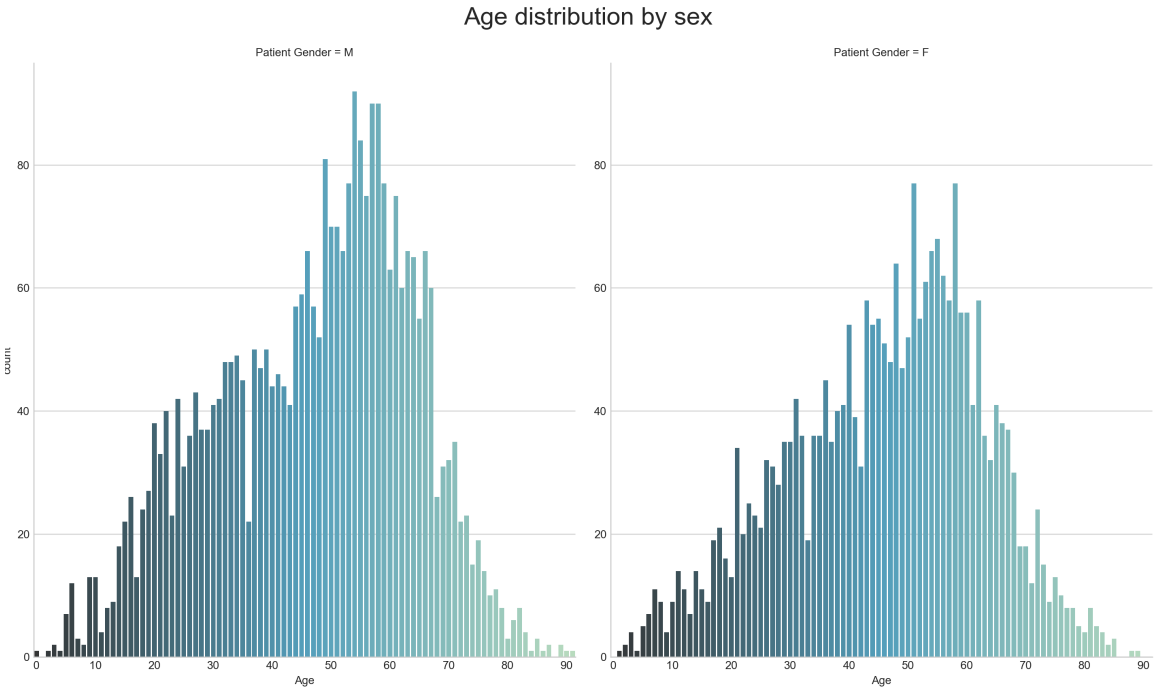

Fig. 2: This figure plots the age for males on the left and the age of females on the right. The range is 0 to 94 with a decently normal distribution and there are slightly more males than females in this dataset.

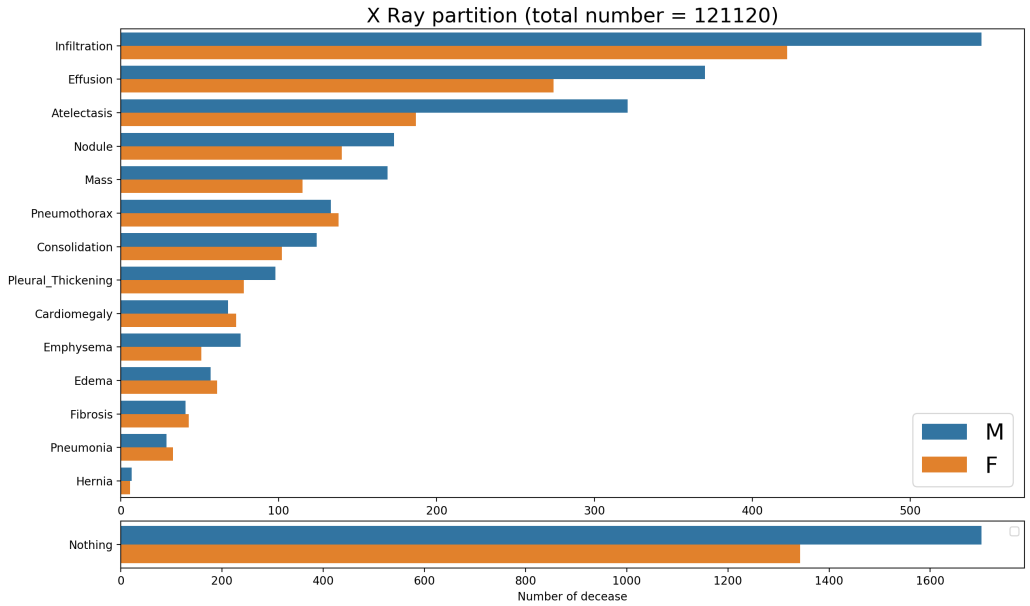

Fig. 3: This figure plots the frequency of the ailment broken down by gender. Orange represents female and blue represents male. This plot included all of our 121,120 observations, it shows the number of X-Ray. It is seen from this plot that 'nothing' is the most frequent diagnosis seen in the dataset. The second predominate ailment is 'Infiltration'.

Finding Labels	Cardiomegaly	Effusion	Nodule	Pneumothorax	Atelectasis	Mass	Consolidation	Infiltration
Effusion|Infiltration	0	1603	0	0	0	0	0	1603
Atelectasis|Infiltration	0	0	0	0	1350	0	0	1350
Atelectasis|Effusion	0	1165	0	0	1165	0	0	0
Infiltration|Nodule	0	0	829	0	0	0	0	829
Atelectasis|Effusion|Infiltration	0	737	0	0	737	0	0	737
Cardiomegaly|Effusion	484	484	0	0	0	0	0	0
Consolidation|Infiltration	0	0	0	0	0	0	441	441
Infiltration|Mass	0	0	0	0	0	420	0	420
Effusion|Pneumothorax	0	403	0	403	0	0	0	0
Effusion|Mass	0	402	0	0	0	402	0	0

Fig. 4: Top 10 Multiple Disease

As we can see in the above table, the top 7 of the ailments in the data set are infiltration, effusion, atelectasis, nodule, mass, consolidation, and pneumothorax. There is a strong unbalance between level of diseases, so we need to balance classes before we do our analysis.

Looking at Figure 3 it is seen that there are more than half of the X-Ray images that have more than one diseases at the same time, this could be explained by the complexity of our human lungs when certain section of the lung is infected with the diseases that could potentially spreading to other area. The rare classes in our data set are mostly related to other diseases, so we need to address this in our analysis to prevent the highly correlation between classes. When we look at the top 10 multiple diseases shown in figure 3. The effusion, infiltration and atelectasis are the three most occur aliments followed by all other lung diseases in our data set.

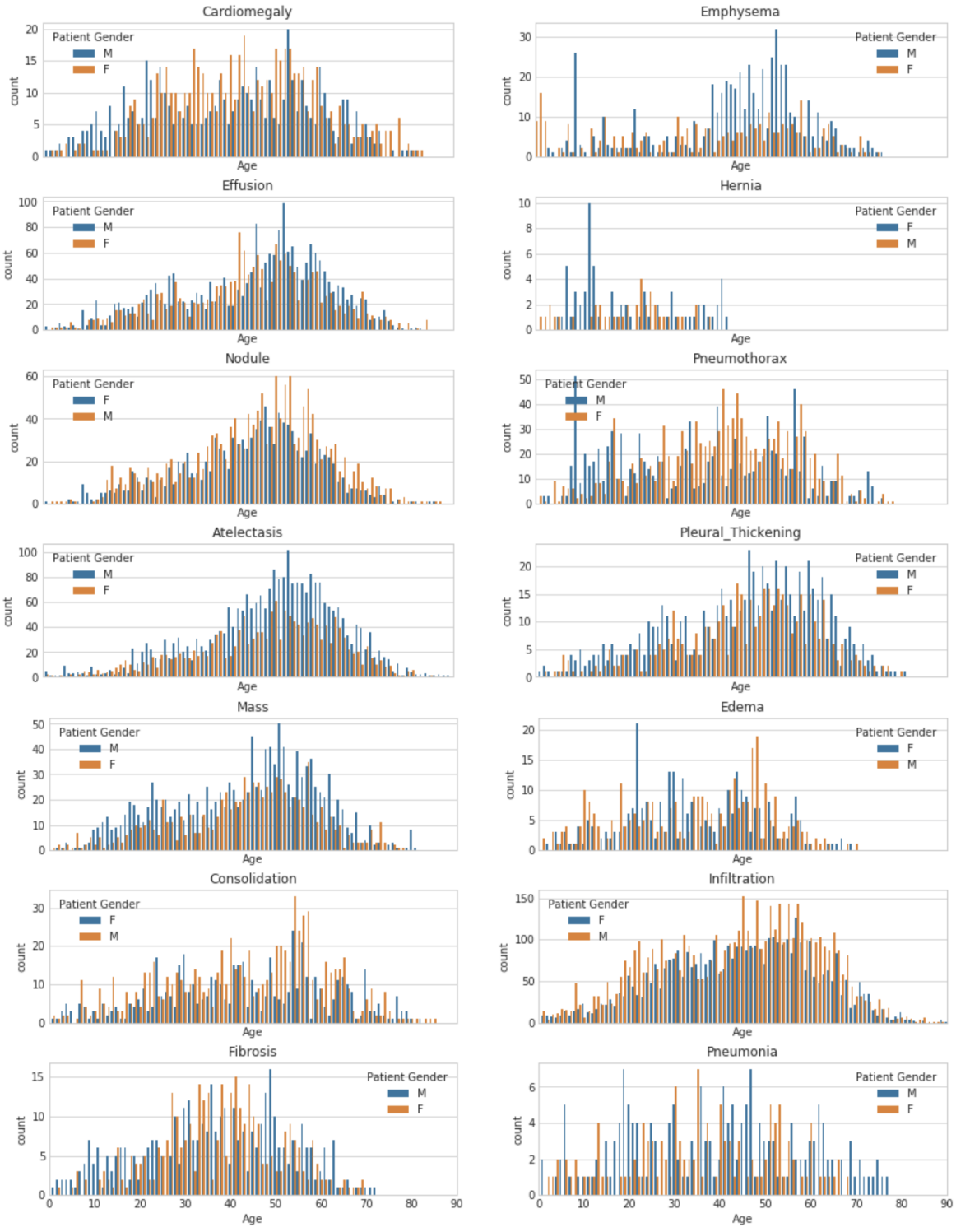

Fig. 5: Ailment frequency breakdown by sex and gender

In the figure above (Figure 5), we can identify the division between age and gender for each aliment. Effusion, nodule, atelectasis, mass, fibrosis, infiltration, and pleural thickening distributed normally. Effusion seen to have younger male patients and mid-age female patients. The patient with consolidation are more likely be male, and they disturbed equally across all ages. There are more male got emphysema in their early and mid-age. Hernia has very small amount of observations in our data set

because this disease could be difficult to detect.

3.3 Results

3.3.1 XGBoost

XGBoost is a package in R that employs an algorithm designed to handle multilevel class responses. It is an extension of the gradient boosting tree algorithm. In order to use XGBoost we had to prepare the data properly for the algorithm to use it. This preparation included converting the findings label to numerical values starting from 0. The first run through with XGBoost, we removed gender, the original finding.labels, and view positions. This left us with only factors that were numerical. In later analysis we included gender and view position and recoded them to be binary.

For this first model, once the labels were converted we balanced the data set by separating out each of the diagnosis into their own data set. From here we randomly sampled 100 cases from each data set and then joined them together back into one data set. Some diagnosis were under the 100 therefore when sampling we used the with replacement method.

After the dataset was balanced we created a training set with 75% of the data and a test set of 25% of the data. Then XGBoost was run on the training set and an accuracy of 66.7% was obtained with a sensitivity of 0.7143 and a specificity of 0.5714. After running the model on the training data we trained the full model and assessed the test error. The prediction accuracy was 81% with a sensitivity of 0.8333 and a specificity of 0.8. Plotting the variables of importance we arrived at the following plot.

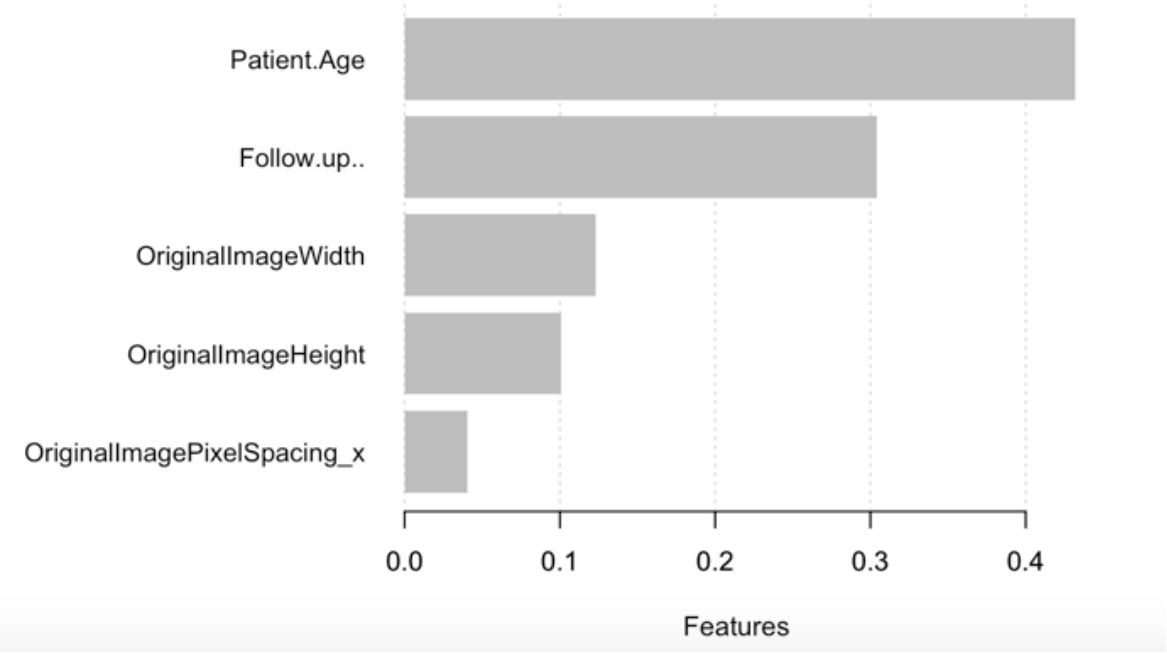

Fig. 6: From this plot we see that Patient Age, and Follow up are the features with the highest importance in our model.

We also ran two additional models, one with gender included and the other with only the image dimensions included. The results of these models did not perform as well as the original model and thus their findings are not included in this report.

3.3.2 CNN

A convolution neural network (CNN) was chosen for this predictive task because it is an image classifier that can predict each ailment based on the X-Ray images. The CNN used the Tensorflow library in Python that included numerical computation by using the data flow graphs. Since we used our personal computer for this project, we started from 5,000 samples then increasing our sample size to 10,000 and 15,000 for a higher accuracy. In the previous medical X-Ray images studies, researchers could achieve an accuracy above 90 percent. Our goal was to have a high accuracy for the high cost medical use, so we could not accept an accuracy rate below 60%.

Before we did the preprocessing, we attached the labels to the images. Then we describe a numpy array to include our labels for each disease. While we did the data inspection using the histogram for each class of the lung diseases, we found our data set has strong biased to the 'No finding' class which was the problem that could lead our CNN model to the wrong result. Rare class of lung diseases also become a problem in our data set, we kept the highest six diseases and combined all other rare disease into 'Other Rare Classes'. The final levels of class show as the following:

0: 'No Finding'	4:'Effusion'
1: 'Consolidation'	5:'Nodule Mass'
2: 'Infiltration'	6:'Atelectasis'
3:'Pneumothorax'	7: 'Other Rare Classes'

As we seen in the below figure (Figure 7), our levels of class were very biased to the 'No Finding', and it appears more than half of our observations. We chose upsampling and downsampling techniques to balance the number of samples of each class. The upsampling method for 5,000, 10,000 and 15,000 did not work well with our dataset and results biased to the 'No Finding' after we balance each class. The downsampling method appear to have lower accuracy rate but it has better results while we increase our sample sizes, so we chose the downsampling technique in this study.

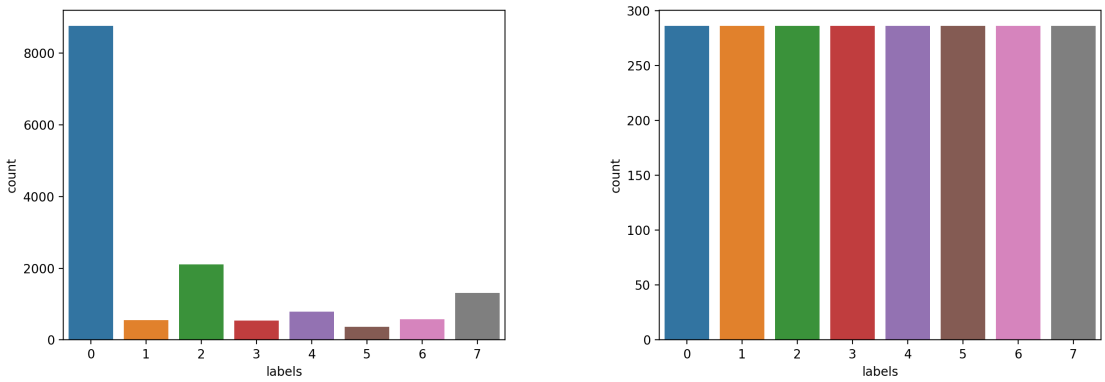

Fig. 7: The left plot shows the frequency for each ailment in the dataset before balancing methods were conducted. The right plot shows the frequency for each ailment in the data set after downsampling was conducted.

The right hand plot in Figure 7 shows the results after we ran the downsampling for 15,000 samples. Our final balance data was nearly 300 in each class and was ready to run the CNN model. We split our sample into 20 percent of testing set and 80 percent of training set, then we use compute_class_weight function in sklearn.utils library to balance the weight of each classes. Finally, we have balanced classes size and the weight of each class for the CNN model.

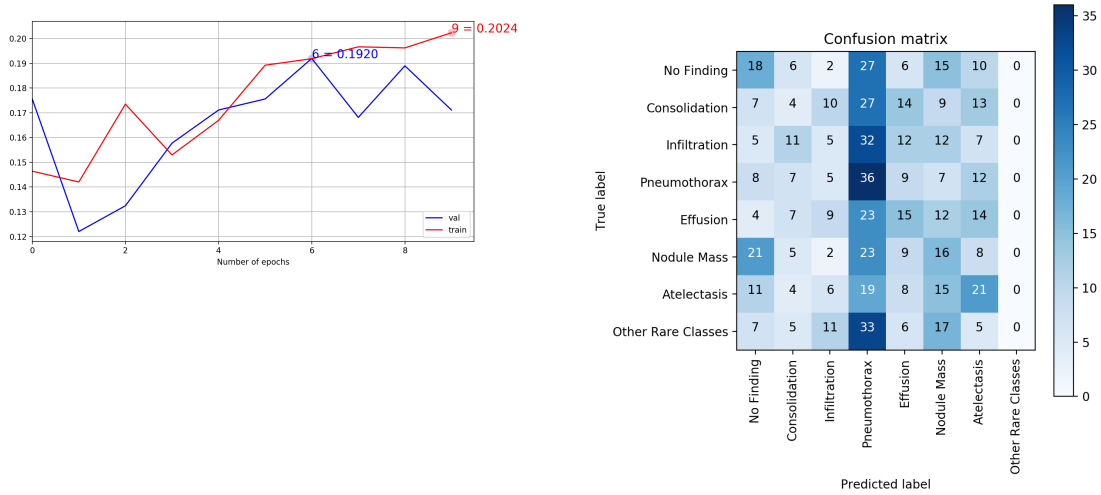

Fig. 8: The left graph represents the learning curves of the CNN and the right graph is the resulting confusion matrix.

The first CNN model cannot predict disease diagnosis at the 90% accuracy rate because of the limit machine we can used in for this study. The learning curves show in Figure 8, the best number of epoch is 9 for the training data (accuracy = 0.2024) and 6 for the testing data (accuracy = 0.1920). The confusion matrix for this model shows the relation between the true labels and the predict labels from this CNN model. This model did predict most of the X-Ray images into the Pneumothorax but did better in other classes. It cannot predict the rare classes we combined for the model.

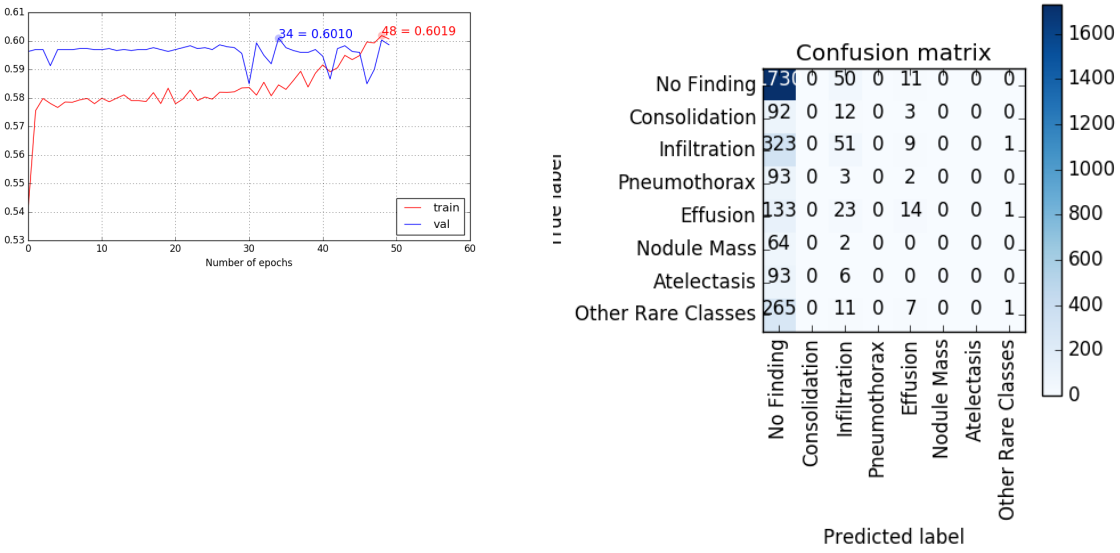

Fig. 9: The left graph represents the learning curves of the CNN with 15,000 images and 50 epochs and the right graph is the resulting confusion matrix.

With the second CNN model which ran 50 epochs on the 15,000 images we accomplished a test accuracy rate of 60.19% and a total run time of 250 seconds. Although the accuracy is higher, the confusion matrix suggests that a better model exists since it predicts an over representation of 'No Finding'.

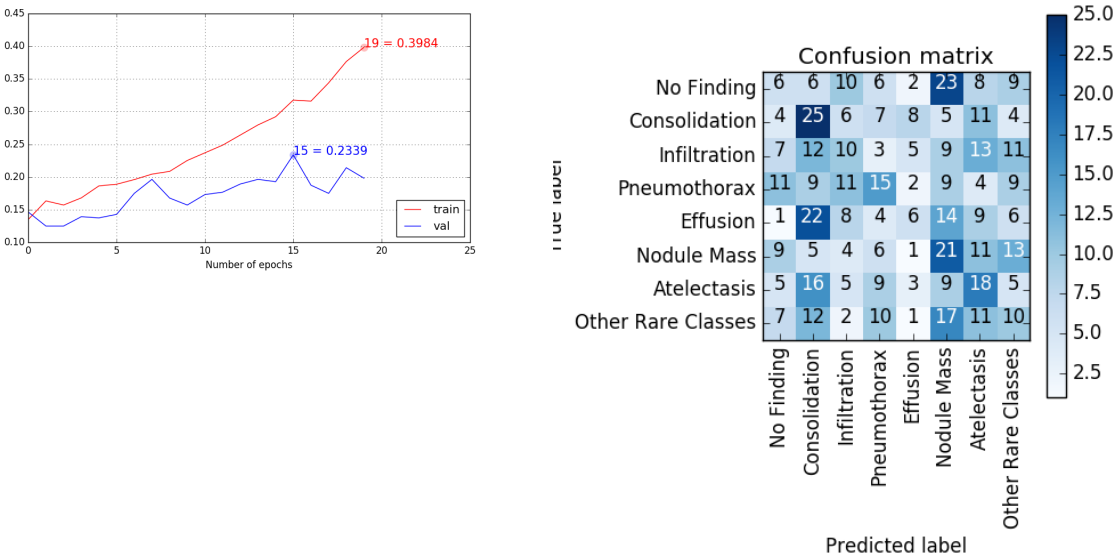

Fig. 10: The left graph represents the learning curves of the CNN with 15,000 SMOTE images and 20 epochs and the right graph is the resulting confusion matrix.

The last model we ran, SMOTE was used to get a better representation of the rarer labeled images. Although the test accuracy rate is lower at 23.39%, the confusion matrix shows that spread of the predicted labels are predicted more correct diagnosis than the other models and for the purpose of this report we conclude that the SMOTE model with 20 epochs is our final model. Although our final model did the job to handle the all the images we included, we look forward to having the chance to

run this model on the super computer again with all the images and an increase in the number of epochs to better our results.

4 Computer Specs

The super computer that was used to run the CNN was a Lambda Quad with the following specs:

GPU	4x1080Ti
Processor	Intel i7
Storage	1TB SSD
Ram	64GB DDR4
Price	$8,693

References

[1] https://www.kaggle.com/paultimothymooney/predicting-pathologies-in-x-ray-images/notebook

[2] http://openaccess.thecvf.com/content_cvpr_2017/papers/Wang_ChestX-ray8_Hospital-Scale_Chest_CVPR_2017_paper.pdf

Appendices

Preprocessing Code

````
---
title: "Project 2 Investigation"
author: "Kalea Sebesta"
date: "1/23/2018"
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

```{r echo=TRUE}
#install packages
cran <- getOption("repos")
cran["dmlc"] <- "https://s3-us-west-2.amazonaws.com/apache-mxnet/R/
   CRAN/"
options(repos = cran)
install.packages("mxnet",dependencies = T)
source("https://bioconductor.org/biocLite.R")
biocLite("EBImage")
```

```{r echo=TRUE}
#load libraries
library(mxnet)
library(EBImage)
library(randomForest)
library(Hmisc)
library(caret)
library(rpart)
library(rpart.plot)
library(mlr)
library(forcats)
library(tidyverse)
#library(rJava)
library(NLP)
#library(openNLP)
#library(RWeka)
library(stringr)
library(tokenizers)
library(Xmisc)
library(ggplot2)
library(lattice)
library(gam)
````
````

```
library(gbm)
```

Set Directory for Image File Location
```{r echo=TRUE}
#Set the directory where the training data lives
image_dir="/Users/kaleasebesta/Documents/DA 6813 Project 2 Sample
   Data/images"
```

Read in CVS File and Perform Descriptive Statistics
```{r echo=TRUE}
#read in the labeled csv file
sample_labels_csv=read.csv('/Users/kaleasebesta/Documents/DA 6813
   Project 2 Sample Data/sample_labels.csv')
#investigate descriptive statistics
#patient age is in a cateogorical variable
summary(sample_labels_csv)
```

Drop Variables that are Patient ID Related
```{r echo=TRUE}
#drop variables that are related to patient identifiers
sample_new=within(sample_labels_csv, rm(Patient.ID, Image.Index))
```

Look for Missing Values, Correlations, and Check Assupmtions
```{r echo=TRUE}
#view missing data (no missing data in our data set)
sapply(sample_labels_csv, function(x) sum(is.na(x)))
sapply(sample_new, function(x) sum(is.na(x)))
```

Calculate Frequencies for the Findings Labels (Diagnosis)
```{r echo=TRUE}
#frequencies
#finding labels have 244 different categories (might want to group
   some of these
# together?)
sapply(sample_new, function(x) length(unique(x)))
#calculating the frequency of each of the findings within the data
   set and the max entries
tt=table(sample_new$Finding.Labels)
max(table(sample_new$Finding.Labels))
```
```{r echo=TRUE}
#convert age category to numerical value
sample_new$Patient.Age=str_replace_all(sample_new$Patient.Age, "Y",
```

```
    "")
sample_new$Patient.Age=lstrip(sample_new$Patient.Age, char="0")

#why would changing to numeric introduce NAs?
#the 2 NAs that were introduced account for 0.03567%
sample_new$Patient.Age <- as.numeric(as.character(
   sample_new$Patient.Age))
sapply(sample_new, function(x) sum(is.na(x)))
which(is.na(sample_new$Patient.Age))

#the row for 3866 and 4750 have null, go to original file find
   those values and impute
#3866 is 13M meaning 13 months old
#4750 is 1D meaning 1 day old (this seems like an outlier
   potentially)
#for the time being I will impute the 13 months with 1 year and the
    1 day with 0
sample_new$Patient.Age[3866]=1
sample_new$Patient.Age[4750]=0
sapply(sample_new, function(x) sum(is.na(x)))
```
```

# Since we have multiple levels of cateogrical variable we will
   utlize the forcats package to see the frequency grouping the
   levels based on ones with more entries.
```{r echo=TRUE}
#convert categorical labels for findings into fewer categories
# run tests with 5, 10, 20, 30 categories
sample_labels_csv %>%
  mutate(Finding.Labels = fct_lump(Finding.Labels, n = 6)) %>%
  count(Finding.Labels)
#Top 10 groups are:
#Atelectasis, Consolidation, Effusion, Effusion|Infiltration,
   Infiltration,
# Mass, No Finding, Nodule, Pleural_Thickening, Pneumothorax
```

Using NLP libraries, a new target variable is created taking the
   first word of the original findings and placing it in a new
   column. This will decrease the amount of levels from 244 to 11
   but still keeps the original variable in case we want to drill
   down deeper into the diagnosis later on in modeling.
```{r echo=TRUE}
#will need to use NLP to recode the diagnosis labels
#get first word of each findings and put into new column
sample_new$Finding2 <- gsub("([A-Za-z]+).*", "\\1",
   sample_new$Finding.Labels)
sample_new$Finding2=str_replace_all(sample_new$Finding2, "No", "
```
```

```
   None")

#write out new csv file
write.csv(sample_new, file="/Users/kaleasebesta/Dropbox/DA 6813
   Data Applications Project 2/Data/cleanedLabels.csv")

#find frequency count of new variable
#This gives us our top 11 groups
sample_new %>%
  mutate(Finding2 = fct_lump(Finding2, n = 10)) %>%
  count(Finding2)

ggplot(data=sample_new, aes(x=Finding2))+
  geom_bar()+
    ggtitle("Frequency Count by Diagnosis")+
      labs(x="Diagnosis Finding", y="Frequency")+
      theme(axis.text.x = element_text(size=16, angle = 45, hjust =
         1))
```
```

#Image Classification within Labeled Data Set

```{r echo=TRUE}
#train the model
set.seed(1)
train=createDataPartition(y=sample_new$Finding2, p=0.7, list =FALSE
   )
train_set = sample_new[train,]
test_set=sample_new[-train,]
dim(train_set)
```
```{r echo=TRUE}
#exploratory plots
ggplot(train_set, aes(Finding2, OriginalImageWidth))+
  geom_boxplot()+ggtitle("Original Image Width")+
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

ggplot(train_set, aes(Finding2, OriginalImageHeight))+
  geom_boxplot()+ggtitle("Original Image Height")+
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

ggplot(train_set, aes(Finding2, OriginalImagePixelSpacing_x))+
  geom_boxplot()+ggtitle("Original Image Pixel X")+
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

ggplot(train_set, aes(Finding2, OriginalImagePixelSpacing_y))+
  geom_boxplot()+ggtitle("Original Image Pixel Y")+
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```
```

```
```

```{r echo=TRUE}
#decision tree model
modFit=caret::train(Finding2~., method="rpart", data=train_set)
rpart.plot(modFit$finalModel)

#decision tree test accuracy and prediction
predictions = predict(modFit, newdata = test_set)
confusionMatrix(predictions, test_set$Finding2)
```

```{r echo=TRUE}
#make training and test set with the sample_new2
#train the model
#drop finding.labels from the sample_new
sample_new2=within(sample_new, rm(Finding.Labels))

#convert findings in new set to factor so that random forest can
   handle it
sample_new2$Finding2 = factor(sample_new2$Finding2)
set.seed(1)
nvar <- dim(train_set)[2]-1
dim(train_set)

#using random forrest
#2000 trees has the optimal amount because lowest oob error rate
rf500=randomForest(Finding2~., data=sample_new2, mtry = sqrt(nvar),
    ntree = 500, proximity=TRUE,oob.prox=FALSE,importance=TRUE)
rf500

rf1000=randomForest(Finding2~., data=sample_new2, mtry = sqrt(nvar)
   , ntree = 1000, proximity=TRUE,oob.prox=FALSE,importance=TRUE)
rf1000

rf2000=randomForest(Finding2~., data=sample_new2, mtry = sqrt(nvar)
   , ntree = 2000, proximity=TRUE,oob.prox=FALSE,importance=TRUE)
rf2000
```

```{r echo=TRUE}
# check variable importance from random forest and plot as visual
   representation
varImpPlot(rf500)
varImpPlot(rf1000)
varImpPlot(rf2000)
```

```{r echo=TRUE}
#GAM
train <- sample(1:nrow(sample_new2), nrow(sample_new2)*.7, rep=
   FALSE)
test <- -train

training_data = sample_new2[train,]
testing_data = sample_new[test,]

gam = gam(Finding2 ~ Patient.Age+ Follow.up..+ OriginalImageWidth+
   OriginalImageHeight+Patient.Gender, family=binomial, data=
   training_data)
plot(gam,se=TRUE, col = "blue")

summary(gam)
```

```{r echo=TRUE}
#gbm (can handle high levels of categorical)
set.seed(456)
#CHANGE THE DATA TO THE TRAINING DATA
gbmModel = gbm(formula = Finding2 ~ .,
              distribution = "multinomial",
              data = training_data,
              n.trees = 2500,
              shrinkage = .01,
              #n.minobsinnode = 20,
              bag.fraction = 0.5,
              train.fraction = 0.5
              #cv.folds = 2
              )

gbmTrainPredictions=predict(object=gbmModel,
                          newdata = training_data,
                          n.trees= 1500,
                          type="response")

head(data.frame("Actual"=training_data$Finding2,
               "PredictedProbability"=gbmTrainPredictions))

#Fit a GBM and test with a holdout subset
# create a subset of trainin data
dataSubsetProportion = .2
randomRows = sample(1:nrow(training_data), floor(nrow(training_data
   ) * dataSubsetProportion))
trainingHoldoutSet = training_data[randomRows, ]
trainingNonHoldoutSet = training_data[!(1:nrow(training_data) %in%
   randomRows), ]
```

```
#Fit GBM on predictors indicated with image
gbmForTesting = gbm(formula = Finding2 ~ Patient.Age+ Follow.up..+
   Patient.Gender+OriginalImageWidth + OriginalImageHeight +
   OriginalImagePixelSpacing_x + OriginalImagePixelSpacing_y,
                  distribution = "multinomial",
                  data = trainingNonHoldoutSet,
                  n.trees = 1500,
                  shrinkage = .01,
                  n.minobsinnode = 50)

#summary
summary(gbmForTesting, plot=FALSE)

#make predictions using the model on the holdout and non holdout
   sets

gbmHoldoutPredictions = predict(object = gbmForTesting,
                              newdata = trainingHoldoutSet,
                              n.trees = 500,
                              type = "response")

gbmNonHoldoutPredictions = predict(object = gbmForTesting,
                                 newdata = trainingNonHoldoutSet,
                                 n.trees = 500,
                                 type = "response")

#CV
gbmWithCrossValidation = gbm(formula = Finding2 ~ .,
                            distribution = "multinomial",
                            data = trainingNonHoldoutSet,
                            n.trees = 2000,
                            shrinkage = .1,
                            n.minobsinnode = 200,
                            cv.folds = 2,
                            n.cores = 1)

#31 trees is the best number
bestTreeForPrediction = gbm.perf(gbmWithCrossValidation)

#make predictions of hold out and nonhold out set with best tree
   size
gbmHoldoutPredictions = predict(object = gbmWithCrossValidation,
                              newdata = trainingHoldoutSet,
                              n.trees = bestTreeForPrediction,
                              type = "response")

gbmNonHoldoutPredictions = predict(object = gbmWithCrossValidation,
                                 newdata = trainingNonHoldoutSet,
```

```
                                n.trees = bestTreeForPrediction,
                                type = "response")
```

```{r echo=TRUE}
#second try with GBM
gbm.train <- sample(1:nrow(sample_new2), nrow(sample_new2)*.7, rep=
   FALSE)
gbm.test <- -train

training_data2 = sample_new2[train,]
testing_data2 = sample_new[test,]

BST = gbm(Finding2~ Patient.Age+ Follow.up..+ Patient.Gender+
   OriginalImageWidth + OriginalImageHeight +
   OriginalImagePixelSpacing_x + OriginalImagePixelSpacing_y,
       data=training_data2,
        distribution='multinomial',
        n.trees=200,
        interaction.depth=4,
        #cv.folds=5,
        shrinkage=0.005)

predBST = predict(BST,n.trees=200, newdata=testing_data2,type='
   response')
p.predBST <- apply(predBST, 1, which.max)
p.predBST
```

```{r echo=TRUE}
#logistic regression
#compare no finding to finding therefore ditchomize the finding.
   labels variable
```

```{r echo=TRUE}
#drop variables that don't help with image recognition
# ex. view position, gender, follow up
```

```{r}
#write new data set to file
write_csv(sample_new, "sample_new.csv")
```

XGBoost Code

```
---
title: "XGBoost"
author: "Kalea Sebesta"
```

```
date: "2/10/2018"
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
library("xgboost") # the main algorithm
library("caret") # for the confusionmatrix() function (also needs
   e1071 package)
library("dplyr") # for some data preperation
library("car")
library("Ckmeans.1d.dp")
library("unbalanced")
```

Load the cleaned data set to begin XGBoost on.
```{r echo=TRUE}
#read in data file
data_original=read.csv('/Volumes/UTSA QRT2/DA 6813 Data Analytics
   Applications/sample_new.csv')
```

Prepare the data by converting the Finding labels in 'Finding2' to
   numerical values starting from 0. Also, I removed the gender,
   finding.labels, and view positions to leave just numerical
   values. It could be argued to convert the patient gender to a 0,
    1 scale as well as the position view. For this first model I
   choose to drop them from the model to see what the result would
   look like.
```{r echo=TRUE}
#data preparation
#XGBoost needs the target variable to be numeric starting at zero
   thus Finding2 will start from 0-14
#0=None, Ac
data_original$Finding2 <- recode(data_original$Finding2,
    "'Atelectasis'=1; 'Cardiomegaly'=2; 'Consolidation'=3; 'Edema
       '=4; 'Effusion'=5; 'Emphysema'=6;
      'Fibrosis'=7; 'Hernia'=8; 'Infiltration'=9; 'Mass'=10; '
         Nonedule'=11; 'Pleural'=12;
      'Pneumonia'=13; 'Pneumothorax'=14; 'None'=0")
data_original$Finding2=as.numeric(levels(data_original$Finding2)[
   data_original$Finding2])
#removing finding.labels Patient Gender and View Position
data_original=within(data_original, rm(Finding.Labels, Patient.
   Gender, View.Position))
summary(data_original)

#clean the age, replace outliers with median outlier being
```
```

```
stem(data_original$Patient.Age)
data_original$Patient.Age[data_original$Patient.Age>=100]=median(
   data_original$Patient.Age)

#examine Follow up replace with median (examine this further)
stem(data_original$Follow.up..)
data_original$Follow.up..[data_original$Follow.up..>=100]=median(
   data_original$Follow.up..)
```
```
Balance the data set by separating out each of the diagnosis into
   their own data set. From here I randomly sampled 100 cases from
   each data set and then joined them together back into one data
   set. Some diagnosis were under the 100 therefore when sampleing
   I used replacement method.
```{r echo=TRUE}
set.seed(747495)
#make separate data set for each diagnosis
data0=data_original[which(data_original$Finding2==0),] #3044 cases
data1=data_original[which(data_original$Finding2==1),] #508 cases
data2=data_original[which(data_original$Finding2==2),] #126 cases
data3=data_original[which(data_original$Finding2==3),] #161 cases
data4=data_original[which(data_original$Finding2==4),] #94 cases
data5=data_original[which(data_original$Finding2==5),] #393 cases
data6=data_original[which(data_original$Finding2==6),] #85 cases
data7=data_original[which(data_original$Finding2==7),] #57 caes
data8=data_original[which(data_original$Finding2==8),] #7 cases
data9=data_original[which(data_original$Finding2==9),] #624 cases
data10=data_original[which(data_original$Finding2==10),] #148 cases
data11=data_original[which(data_original$Finding2==11),] #159 cases
data12=data_original[which(data_original$Finding2==12),] #72 cases
data13=data_original[which(data_original$Finding2==13),] #14 cases
data14=data_original[which(data_original$Finding2==14),] #114 cases

#randomly subset each data set to 100 of the sets that had over 100
    cases
data0 <- data0[sample(1:nrow(data0), 100,
       replace=FALSE),]
data1 <- data1[sample(1:nrow(data1), 100,
       replace=FALSE),]
data2 <- data2[sample(1:nrow(data2), 100,
       replace=FALSE),]
data3 <- data3[sample(1:nrow(data3), 100,
       replace=FALSE),]
data5 <- data5[sample(1:nrow(data5), 100,
       replace=FALSE),]
data9 <- data9[sample(1:nrow(data9), 100,
       replace=FALSE),]
data10 <- data10[sample(1:nrow(data10), 100,
```
```

```
       replace=FALSE),]
data11 <- data11[sample(1:nrow(data11), 100,
       replace=FALSE),]
data14 <- data14[sample(1:nrow(data14), 100,
       replace=FALSE),]

#with replacement for classes that are under represented
data13 <- data13[sample(1:nrow(data13), 100,
       replace=TRUE),]
data8 <- data8[sample(1:nrow(data8), 100,
       replace=TRUE),]
data7 <- data7[sample(1:nrow(data7), 100,
       replace=TRUE),]

#combine all datasets into one
new_data=rbind(data13, data8, data7, data4, data6, data12, data14,
             data11, data10, data9, data5, data3, data2, data1,
                data0)
dim(new_data) #1451 rows 7 variables
```

Split the data into training and test set. For XGBoost, we want to make sure the dataframe is in a matrix form where the finding2 category is only in the label vector and not the variable matrix .

```{r echo=TRUE}
#train and split data
#change dataframe into matrix so that xgboost can work with it
data_original=new_data
data_original=data.matrix(data_original)
idx=sample(1:nrow(data_original), nrow(data_original)*0.75)
data_vars=as.matrix(data_original[,-7])
data_label=data_original[,"Finding2"]
data_matrix <- xgb.DMatrix(data = as.matrix(data_original), label =
    data_label)

# split train data and make xgb.DMatrix
train_data <- data_vars[idx,]
train_label <- data_label[idx]
train_matrix <- xgb.DMatrix(data = train_data, label = train_label)

# split test data and make xgb.DMatrix
test_data <- data_vars[-idx,]
test_label <- data_label[-idx]
test_matrix <- xgb.DMatrix(data = test_data, label = test_label)
```

Perform k-fold cross validation and calculate the estimated error

```{r echo=TRUE}
#K-folds cross validation and estimated error
numberOfClasses <- 15
xgb_params <- list("objective" = "multi:softprob",
                   "eval_metric" = "mlogloss",
                   "num_class" = numberOfClasses)
nround <- 50 # number of XGBoost rounds
cv.nfold <- 5
# Fit cv.nfold * cv.nround XGB models and save OOF predictions
cv_model <- xgb.cv(params = xgb_params,
                   data = train_matrix,
                   nrounds = nround,
                   nfold = cv.nfold,
                   verbose = FALSE,
                  prediction = TRUE)
```
Assess out of fold prediction error and the confusion matrix. In the OOB_prediction labels two values where not seen thus to make the confusion matrix work we needed to force the range of the original prediction labels (all labels 1-15). The accuracy is 66.6% with a sensativity of .71 and a specificity of .57.

```{r echo=TRUE}
#assess out of fold prediction error
set.seed(12345)
OOF_prediction <- data.frame(cv_model$pred) %>%
  mutate(max_prob = max.col(., ties.method = "last"),
         label = train_label +1)
head(OOF_prediction)

# confusion matrix (accuracy of model is 96.25%)
confusionMatrix(factor(OOF_prediction$label, levels=range(
   OOF_prediction$label)),
               factor(OOF_prediction$max_prob, levels=range(
                  OOF_prediction$max_prob)), mode = "everything")
```

Train full model and assess test set error.
The prediction accuracy is 81% with a sensativity of .8333 and a specificity of .8.

```{r echo=TRUE}
#train full model and assess test set error
set.seed(12345)
bst_model <- xgb.train(params = xgb_params,
                       data = train_matrix,
                       nrounds = nround)
# Predict hold-out test set
test_pred <- predict(bst_model, newdata = test_matrix)
test_prediction <- matrix(test_pred, nrow = numberOfClasses,
```

```
                        ncol=length(test_pred)/numberOfClasses) %>%
  t() %>%
  data.frame() %>%
  mutate(label = test_label + 1,
         max_prob = max.col(., "last"))
# confusion matrix of test set (model accuracy 96.2%)
confusionMatrix(factor(test_prediction$label, levels=range(
   test_prediction$label)),
             factor(test_prediction$max_prob, levels=range(
                test_prediction$max_prob)), mode = "everything")
```

```
```{r echo=TRUE}
#variable importance
# get the feature real names
names <- colnames(data_original[,-7])
# compute feature importance matrix
importance_matrix = xgb.importance(feature_names = names, model =
   bst_model)
head(importance_matrix)
```

```
```{r echo=TRUE}
# plot the importance variables
gp = xgb.plot.importance(importance_matrix, xlab="Features",
   left_margin=12)
print(gp)

gp2 = xgb.ggplot.importance(importance_matrix)
print(gp2)

gp2+theme( text = element_text(size = 16),
  axis.text.x = element_text(size = 16, angle = 45, hjust = 1))+
     ggtitle("Feature Importance")
```
```

XGBoost Imagine Dimensions Only Code

```
---
title: "XGBoost_ImgDimOnly"
author: "Kalea Sebesta"
date: "2/10/2018"
output: html_document

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
library("xgboost") # the main algorithm
library("caret") # for the confusionmatrix() function (also needs
```

```
   e1071 package)
library("dplyr") # for some data preperation
library("car")
library("Ckmeans.1d.dp")
library("unbalanced")
```

Load the cleaned data set to begin XGBoost on.

```{r echo=TRUE}
#read in data file
data_original=read.csv('/Volumes/UTSA QRT2/DA 6813 Data Analytics
   Applications/sample_new.csv')
```

Prepare data and look only at image dimension variables with
   regards to the finding diagnosis.

```{r echo=TRUE}
#data preparation
#XGBoost needs the target variable to be numeric starting at zero
   thus Finding2 will start from 0-14
#0=None, Ac
data_original$Finding2 <- recode(data_original$Finding2,
    "'Atelectasis'=1; 'Cardiomegaly'=2; 'Consolidation'=3; 'Edema
       '=4; 'Effusion'=5; 'Emphysema'=6;
      'Fibrosis'=7; 'Hernia'=8; 'Infiltration'=9; 'Mass'=10; '
         Nonedule'=11; 'Pleural'=12;
      'Pneumonia'=13; 'Pneumothorax'=14; 'None'=0")
data_original$Finding2=as.numeric(levels(data_original$Finding2)[
   data_original$Finding2])

#removing finding.labels Patient Gender and View Position
data_original=within(data_original, rm(Finding.Labels, Patient.
   Gender, Follow.up.., Patient.Age, View.Position))
summary(data_original)
```

Balance the data set by separating out each of the diagnosis into
   their own data set. From here I randomly sampled 100 cases from
   each data set and then joined them together back into one data
   set. Some diagnosis were under the 100 therefore when sampleing
   I used replacement method.

```{r echo=TRUE}
set.seed(747495)
#make separate data set for each diagnosis
data0=data_original[which(data_original$Finding2==0),] #3044 cases
data1=data_original[which(data_original$Finding2==1),] #508 cases
data2=data_original[which(data_original$Finding2==2),] #126 cases
data3=data_original[which(data_original$Finding2==3),] #161 cases
data4=data_original[which(data_original$Finding2==4),] #94 cases
```

```
data5=data_original[which(data_original$Finding2==5),] #393 cases
data6=data_original[which(data_original$Finding2==6),] #85 cases
data7=data_original[which(data_original$Finding2==7),] #57 caes
data8=data_original[which(data_original$Finding2==8),] #7 cases
data9=data_original[which(data_original$Finding2==9),] #624 cases
data10=data_original[which(data_original$Finding2==10),] #148 cases
data11=data_original[which(data_original$Finding2==11),] #159 cases
data12=data_original[which(data_original$Finding2==12),] #72 cases
data13=data_original[which(data_original$Finding2==13),] #14 cases
data14=data_original[which(data_original$Finding2==14),] #114 cases

#randomly subset each data set to 100 of the sets that had over 100
    cases
data0 <- data0[sample(1:nrow(data0), 100,
       replace=FALSE),]
data1 <- data1[sample(1:nrow(data1), 100,
       replace=FALSE),]
data2 <- data2[sample(1:nrow(data2), 100,
       replace=FALSE),]
data3 <- data3[sample(1:nrow(data3), 100,
       replace=FALSE),]
data5 <- data5[sample(1:nrow(data5), 100,
       replace=FALSE),]
data9 <- data9[sample(1:nrow(data9), 100,
       replace=FALSE),]
data10 <- data10[sample(1:nrow(data10), 100,
       replace=FALSE),]
data11 <- data11[sample(1:nrow(data11), 100,
       replace=FALSE),]
data14 <- data14[sample(1:nrow(data14), 100,
       replace=FALSE),]

#with replacement for classes that are under represented
data13 <- data13[sample(1:nrow(data13), 100,
       replace=TRUE),]
data8 <- data8[sample(1:nrow(data8), 100,
       replace=TRUE),]
data7 <- data7[sample(1:nrow(data7), 100,
       replace=TRUE),]

#combine all datasets into one
new_data=rbind(data13, data8, data7, data4, data6, data12, data14,
             data11, data10, data9, data5, data3, data2, data1,
                data0)
dim(new_data) #1451 rows 7 variables
```

Split the data into training and test set. For XGBoost, we want to

```
   make sure the dataframe is in a matrix form where the finding2
   category is only in the label vector and not the variable matrix
   .
```{r echo=TRUE}
#train and split data
#change dataframe into matrix so that xgboost can work with it
data_original=new_data
data_original=data.matrix(data_original)
idx=sample(1:nrow(data_original), nrow(data_original)*0.75)
data_vars=as.matrix(data_original[,-5])
data_label=data_original[,"Finding2"]
data_matrix <- xgb.DMatrix(data = as.matrix(data_original), label =
    data_label)

# split train data and make xgb.DMatrix
train_data <- data_vars[idx,]
train_label <- data_label[idx]
train_matrix <- xgb.DMatrix(data = train_data, label = train_label)

# split test data and make xgb.DMatrix
test_data <- data_vars[-idx,]
test_label <- data_label[-idx]
test_matrix <- xgb.DMatrix(data = test_data, label = test_label)
```

Perform k-fold cross validation and calculate the estimated error
```{r echo=TRUE}
#K-folds cross validation and estimated error
numberOfClasses <- 15
xgb_params <- list("objective" = "multi:softprob",
                   "eval_metric" = "mlogloss",
                   "num_class" = numberOfClasses)
nround <- 50 # number of XGBoost rounds
cv.nfold <- 5
# Fit cv.nfold * cv.nround XGB models and save OOF predictions
cv_model <- xgb.cv(params = xgb_params,
                   data = train_matrix,
                   nrounds = nround,
                   nfold = cv.nfold,
                   verbose = FALSE,
                  prediction = TRUE)
```
Assess out of fold prediction error and the confusion matrix. In
   the OOB_prediction labels two values where not seen thus to make
    the confusion matrix work we needed to force the range of the
   original prediction labels (all labels 1-15). The accuracy is
   50% with a sensativity of .33 and a specificity of .54.
```{r echo=TRUE}
```
```

```
#assess out of fold prediction error
set.seed(12345)
OOF_prediction <- data.frame(cv_model$pred) %>%
  mutate(max_prob = max.col(., ties.method = "last"),
         label = train_label +1)
head(OOF_prediction)

# confusion matrix (accuracy of model is 96.25%)
confusionMatrix(factor(OOF_prediction$label, levels=range(
   OOF_prediction$label)),
              factor(OOF_prediction$max_prob, levels=range(
                 OOF_prediction$max_prob)), mode = "everything")
```

Train full model and assess test set error.
The prediction accuracy is 50% with a sensativity of NA and a specificity of .5.

```{r echo=TRUE}
#train full model and assess test set error
set.seed(12345)
bst_model <- xgb.train(params = xgb_params,
                       data = train_matrix,
                       nrounds = nround)
# Predict hold-out test set
test_pred <- predict(bst_model, newdata = test_matrix)
test_prediction <- matrix(test_pred, nrow = numberOfClasses,
                          ncol=length(test_pred)/numberOfClasses) %>%
  t() %>%
  data.frame() %>%
  mutate(label = test_label + 1,
         max_prob = max.col(., "last"))
# confusion matrix of test set (model accuracy 96.2%)
confusionMatrix(factor(test_prediction$label, levels=range(
   test_prediction$label)),
             factor(test_prediction$max_prob, levels=range(
                test_prediction$max_prob)), mode = "everything")
```

```{r echo=TRUE}
#variable importance
# get the feature real names
names <- colnames(data_original[,-7])
# compute feature importance matrix
importance_matrix = xgb.importance(feature_names = names, model =
   bst_model)
head(importance_matrix)
```

```
```{r echo=TRUE}
# plot the importance variables
gp = xgb.plot.importance(importance_matrix, xlab="Features",
   left_margin=12)
print(gp)

gp2 = xgb.ggplot.importance(importance_matrix)
print(gp2)

gp2+theme( text = element_text(size = 16),
  axis.text.x = element_text(size = 16, angle = 45, hjust = 1))+
     ggtitle("Feature Importance")
```
```

XGBoost Gender Code

```
---
title: "XGBoost with Gender"
author: "Kalea Sebesta"
date: "2/10/2018"
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
library("xgboost") # the main algorithm
library("caret") # for the confusionmatrix() function (also needs
   e1071 package)
library("dplyr") # for some data preperation
library("car")
library("Ckmeans.1d.dp")
library("unbalanced")
```

Load the cleaned data set to begin XGBoost on.
```{r echo=TRUE}
#read in data file
data_original=read.csv('/Volumes/UTSA QRT2/DA 6813 Data Analytics
   Applications/sample_new.csv')
```

Prepare the data by converting the Finding labels in 'Finding2' to
   numerical values starting from 0. Also, I removed the gender,
   finding.labels, and view positions to leave just numerical
   values. It could be argued to convert the patient gender to a 0,
    1 scale as well as the position view. For this first model I
   choose to drop them from the model to see what the result would
   look like.
```{r echo=TRUE}
```
```

```
#data preparation
#XGBoost needs the target variable to be numeric starting at zero
   thus Finding2 will start from 0-14
#0=None, Ac
data_original$Finding2 <- recode(data_original$Finding2,
    "'Atelectasis'=1; 'Cardiomegaly'=2; 'Consolidation'=3; 'Edema
       '=4; 'Effusion'=5; 'Emphysema'=6;
      'Fibrosis'=7; 'Hernia'=8; 'Infiltration'=9; 'Mass'=10; '
         Nonedule'=11; 'Pleural'=12;
      'Pneumonia'=13; 'Pneumothorax'=14; 'None'=0")
data_original$Finding2=as.numeric(levels(data_original$Finding2)[
   data_original$Finding2])

#removing finding.labels Patient Gender and View Position
data_original=within(data_original, rm(Finding.Labels, View.
   Position))
summary(data_original)

#clean the age, replace outliers with median outlier being
stem(data_original$Patient.Age)
data_original$Patient.Age[data_original$Patient.Age>=100]=median(
   data_original$Patient.Age)

#examine Follow up replace with median (examine this further)
stem(data_original$Follow.up..)
data_original$Follow.up..[data_original$Follow.up..>=100]=median(
   data_original$Follow.up..)
```
```
Balance the data set by separating out each of the diagnosis into
   their own data set. From here I randomly sampled 100 cases from
   each data set and then joined them together back into one data
   set. Some diagnosis were under the 100 therefore when sampleing
   I used replacement method.
```{r echo=TRUE}
set.seed(747495)
#make separate data set for each diagnosis
data0=data_original[which(data_original$Finding2==0),] #3044 cases
data1=data_original[which(data_original$Finding2==1),] #508 cases
data2=data_original[which(data_original$Finding2==2),] #126 cases
data3=data_original[which(data_original$Finding2==3),] #161 cases
data4=data_original[which(data_original$Finding2==4),] #94 cases
data5=data_original[which(data_original$Finding2==5),] #393 cases
data6=data_original[which(data_original$Finding2==6),] #85 cases
data7=data_original[which(data_original$Finding2==7),] #57 caes
data8=data_original[which(data_original$Finding2==8),] #7 cases
data9=data_original[which(data_original$Finding2==9),] #624 cases
data10=data_original[which(data_original$Finding2==10),] #148 cases
data11=data_original[which(data_original$Finding2==11),] #159 cases
```
```

```
data12=data_original[which(data_original$Finding2==12),] #72 cases
data13=data_original[which(data_original$Finding2==13),] #14 cases
data14=data_original[which(data_original$Finding2==14),] #114 cases

#randomly subset each data set to 100 of the sets that had over 100
    cases
data0 <- data0[sample(1:nrow(data0), 100,
       replace=FALSE),]
data1 <- data1[sample(1:nrow(data1), 100,
       replace=FALSE),]
data2 <- data2[sample(1:nrow(data2), 100,
       replace=FALSE),]
data3 <- data3[sample(1:nrow(data3), 100,
       replace=FALSE),]
data5 <- data5[sample(1:nrow(data5), 100,
       replace=FALSE),]
data9 <- data9[sample(1:nrow(data9), 100,
       replace=FALSE),]
data10 <- data10[sample(1:nrow(data10), 100,
       replace=FALSE),]
data11 <- data11[sample(1:nrow(data11), 100,
       replace=FALSE),]
data14 <- data14[sample(1:nrow(data14), 100,
       replace=FALSE),]

#with replacement for classes that are under represented
data13 <- data13[sample(1:nrow(data13), 100,
       replace=TRUE),]
data8 <- data8[sample(1:nrow(data8), 100,
       replace=TRUE),]
data7 <- data7[sample(1:nrow(data7), 100,
       replace=TRUE),]

#combine all datasets into one
new_data=rbind(data13, data8, data7, data4, data6, data12, data14,
             data11, data10, data9, data5, data3, data2, data1,
                data0)
dim(new_data) #1451 rows 7 variables
```

Split the data into training and test set. For XGBoost, we want to make sure the dataframe is in a matrix form where the finding2 category is only in the label vector and not the variable matrix .

```
```{r echo=TRUE}
#train and split data
#change dataframe into matrix so that xgboost can work with it
data_original=new_data
```
```

```
data_original=data.matrix(data_original)
idx=sample(1:nrow(data_original), nrow(data_original)*0.75)
data_vars=as.matrix(data_original[,-8])
data_label=data_original[,"Finding2"]
data_matrix <- xgb.DMatrix(data = as.matrix(data_original), label =
    data_label)

# split train data and make xgb.DMatrix
train_data <- data_vars[idx,]
train_label <- data_label[idx]
train_matrix <- xgb.DMatrix(data = train_data, label = train_label)

# split test data and make xgb.DMatrix
test_data <- data_vars[-idx,]
test_label <- data_label[-idx]
test_matrix <- xgb.DMatrix(data = test_data, label = test_label)
```

Perform k-fold cross validation and calculate the estimated error

```{r echo=TRUE}
#K-folds cross validation and estimated error
numberOfClasses <- 15
xgb_params <- list("objective" = "multi:softprob",
                   "eval_metric" = "mlogloss",
                   "num_class" = numberOfClasses)
nround <- 50 # number of XGBoost rounds
cv.nfold <- 5
# Fit cv.nfold * cv.nround XGB models and save OOF predictions
cv_model <- xgb.cv(params = xgb_params,
                   data = train_matrix,
                   nrounds = nround,
                   nfold = cv.nfold,
                   verbose = FALSE,
                  prediction = TRUE)
```

Assess out of fold prediction error and the confusion matrix. In the OOB_prediction labels two values where not seen thus to make the confusion matrix work we needed to force the range of the original prediction labels (all labels 1-15). The accuracy is 66.6% with a sensativity of .71 and a specificity of .57.

```{r echo=TRUE}
#assess out of fold prediction error
set.seed(12345)
OOF_prediction <- data.frame(cv_model$pred) %>%
  mutate(max_prob = max.col(., ties.method = "last"),
         label = train_label +1)
head(OOF_prediction)
```

```
# confusion matrix (accuracy of model is 96.25%)
confusionMatrix(factor(OOF_prediction$label, levels=range(
   OOF_prediction$label)),
              factor(OOF_prediction$max_prob, levels=range(
                 OOF_prediction$max_prob)), mode = "everything")
```

Train full model and assess test set error.
The prediction accuracy is 81% with a sensativity of .8333 and a specificity of .8.

```{r echo=TRUE}
#train full model and assess test set error
set.seed(12345)
bst_model <- xgb.train(params = xgb_params,
                    data = train_matrix,
                    nrounds = nround)
# Predict hold-out test set
test_pred <- predict(bst_model, newdata = test_matrix)
test_prediction <- matrix(test_pred, nrow = numberOfClasses,
                       ncol=length(test_pred)/numberOfClasses) %>%
  t() %>%
  data.frame() %>%
  mutate(label = test_label + 1,
         max_prob = max.col(., "last"))
# confusion matrix of test set (model accuracy 96.2%)
confusionMatrix(factor(test_prediction$label, levels=range(
   test_prediction$label)),
            factor(test_prediction$max_prob, levels=range(
               test_prediction$max_prob)), mode = "everything")
```

```{r echo=TRUE}
#variable importance
# get the feature real names
names <- colnames(data_original[,-7])
# compute feature importance matrix
importance_matrix = xgb.importance(feature_names = names, model =
   bst_model)
head(importance_matrix)
```

```{r echo=TRUE}
# plot the importance variables
gp = xgb.plot.importance(importance_matrix, xlab="Features",
   left_margin=12)
print(gp)
```

```
gp2 = xgb.ggplot.importance(importance_matrix)
print(gp2)

gp2+theme( text = element_text(size = 16),
  axis.text.x = element_text(size = 16, angle = 45, hjust = 1))+
     ggtitle("Feature Importance")
```

CNN Code

```
import pandas as pd
import numpy as np
import os
import itertools
import sklearn
import matplotlib.pylab as plt
import cv2
import seaborn as sns
from glob import glob
from sklearn.model_selection import train_test_split
from keras.utils.np_utils import to_categorical
from keras.callbacks import Callback
from keras.layers import Dense, Dropout, Flatten
from keras.optimizers import RMSprop
from keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import confusion_matrix
from keras.models import Sequential
from keras.layers import Conv2D, MaxPool2D, BatchNormalization
from imblearn.under_sampling import RandomUnderSampler

PATH = os.path.abspath(os.path.join('..', 'Project 2'))
labels = pd.read_csv('..//Project 2/sample/Data_Entry_2017.csv')

SOURCE_IMAGES = os.path.join(PATH, "sample", "images")
images = glob(os.path.join(SOURCE_IMAGES, "*.png"))

labels = labels[['Image Index','Finding Labels','Follow-up #','
   Patient ID','Patient Age','Patient Gender']]

#Preprocess Data
def proc_images():
    """
    Returns two arrays:
        x is an array of resized images
        y is an array of labels
    """
    NoFinding = "No Finding" #0
    Consolidation="Consolidation" #1
    Infiltration="Infiltration" #2
```

```
Pneumothorax="Pneumothorax" #3
Edema="Edema" # 7
Emphysema="Emphysema" #7
Fibrosis="Fibrosis" #7
Effusion="Effusion" #4
Pneumonia="Pneumonia" #7
Pleural_Thickening="Pleural_Thickening" #7
Cardiomegaly="Cardiomegaly" #7
NoduleMass="Nodule" #5
Hernia="Hernia" #7
Atelectasis="Atelectasis" #6
RareClass = ["Edema", "Emphysema", "Fibrosis", "Pneumonia", "
   Pleural_Thickening", "Cardiomegaly","Hernia"]
x = [] # images as arrays
y = [] # labels Infiltration or Not_infiltration
WIDTH = 64
HEIGHT = 64
for img in images:
    base = os.path.basename(img)
    finding = labels["Finding Labels"][labels["Image Index"] ==
       base].values[0]
    # Read and resize image
    full_size_image = cv2.imread(img)
    x.append(cv2.resize(full_size_image, (WIDTH,HEIGHT),
       interpolation=cv2.INTER_CUBIC))
    # Labels
    if NoFinding in finding:
        finding = 0
        y.append(finding)
    elif Consolidation in finding:
        finding = 1
        y.append(finding)
    elif Infiltration in finding:
        finding = 2
        y.append(finding)
    elif Pneumothorax in finding:
        finding = 3
        y.append(finding)
    elif Edema in finding:
        finding = 7
        y.append(finding)
    elif Emphysema in finding:
        finding = 7
        y.append(finding)
    elif Fibrosis in finding:
        finding = 7
        y.append(finding)
    elif Effusion in finding:
```

```
            finding = 4
            y.append(finding)
        elif Pneumonia in finding:
            finding = 7
            y.append(finding)
        elif Pleural_Thickening in finding:
            finding = 7
            y.append(finding)
        elif Cardiomegaly in finding:
            finding = 7
            y.append(finding)
        elif NoduleMass in finding:
            finding = 5
            y.append(finding)
        elif Hernia in finding:
            finding = 7
            y.append(finding)
        elif Atelectasis in finding:
            finding = 6
            y.append(finding)
        else:
            finding = 7
            y.append(finding)
    return x,y

#part 2
X,y = proc_images()
df = pd.DataFrame()
df["images"]=X
df["labels"]=y

#Describe new numpy arrays
dict_characters = {0: 'No Finding', 1: 'Consolidation', 2: '
    Infiltration',
        3: 'Pneumothorax', 4:'Effusion', 5: 'Nodule Mass', 6: '
            Atelectasis', 7: "Other Rare Classes"}

print(df.head(10))
print("")
print(dict_characters)

#show unbalance
lab = df['labels']
dist = lab.value_counts()
sns.countplot(lab)
print(dict_characters)

X=np.array(X)
```

```
X=X/255.0

#test&train

X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size
   =0.2)
print("Training Data Shape:", X_train.shape)
print("Testing Data Shape:", X_test.shape)
print("Training Data Shape:", len(X_train), X_train[0].shape)
print("Testing Data Shape:", len(X_test), X_test[0].shape)

# try to use a CNN to predict each ailment based off of the X-Ray
   image.
# Encode labels to hot vectors (ex : 2 -> [0,0,1,0,0,0,0,0,0,0])
Y_trainHot = to_categorical(Y_train, num_classes = 8)
Y_testHot = to_categorical(Y_test, num_classes = 8)

# In order to avoid having a biased model because of skewed class
   sizes,
# We need to modify the class_weights parameter in order to give
   more weight to the rare classes.
# In this case the class_weights parameter will eventually be
   passed to the model.fit function.

from sklearn.utils import class_weight
class_weight = class_weight.compute_class_weight('balanced', np.
   unique(y), y)
print(class_weight)

# Helper Functions Learning Curves and Confusion Matrix


class MetricsCheckpoint(Callback):
    """Callback that saves metrics after each epoch"""
    def __init__(self, savepath):
        super(MetricsCheckpoint, self).__init__()
        self.savepath = savepath
        self.history = {}
    def on_epoch_end(self, epoch, logs=None):
        for k, v in logs.items():
            self.history.setdefault(k, []).append(v)
        np.save(self.savepath, self.history)

def plotKerasLearningCurve():
    plt.figure(figsize=(10,5))
    metrics = np.load('logs.npy')[()]
    filt = ['acc'] # try to add 'loss' to see the loss learning
       curve
```

```
    for k in filter(lambda x : np.any([kk in x for kk in filt]),
       metrics.keys()):
        l = np.array(metrics[k])
        plt.plot(l, c= 'r' if 'val' not in k else 'b', label='val'
           if 'val' in k else 'train')
        x = np.argmin(l) if 'loss' in k else np.argmax(l)
        y = l[x]
        plt.scatter(x,y, lw=0, alpha=0.25, s=100, c='r' if 'val' not
            in k else 'b')
        plt.text(x, y, '{} = {:.4f}'.format(x,y), size='15', color=
           'r' if 'val' not in k else 'b')
    plt.legend(loc=4)
    plt.axis([0, None, None, None]);
    plt.grid()
    plt.xlabel('Number of epochs')

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.figure(figsize = (5,5))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=90)
    plt.yticks(tick_marks, classes)
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.
       shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

#CNN
def runCNNconfusion(a,b,c,d):
    # In -> [[Conv2D->relu]*2 -> MaxPool2D -> Dropout]*2 -> Flatten
        -> Dense -> Dropout -> Out
```

```
batch_size = 128
num_classes = 8
epochs = 20
img_rows, img_cols = X_train.shape[1],X_train.shape[2]
input_shape = (img_rows, img_cols, 3)
model = Sequential()
model.add(Conv2D(filters = 32, kernel_size = (3,3),padding = '
    Same',
                 activation ='relu', input_shape = input_shape))
model.add(Conv2D(filters = 32, kernel_size = (3,3),padding = '
    Same',
                 activation ='relu'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Dropout(0.25))
model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = '
    Same',
                 activation ='relu'))
model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = '
    Same',
                 activation ='relu'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Dropout(0.25))
model.add(Conv2D(filters = 86, kernel_size = (3,3),padding = '
    Same',
                 activation ='relu'))
model.add(Conv2D(filters = 86, kernel_size = (3,3),padding = '
    Same',
                 activation ='relu'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Dropout(0.25))
model.add(Flatten())
#model.add(Dense(1024, activation = "relu"))
#model.add(Dropout(0.5))
model.add(Dense(512, activation = "relu"))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation = "softmax"))
# Define the optimizer
optimizer = RMSprop(lr=0.001, decay=1e-6)
model.compile(optimizer = optimizer , loss = "
    categorical_crossentropy", metrics=["accuracy"])
datagen = ImageDataGenerator(
    featurewise_center=False, # set input mean to 0 over the
       dataset
    samplewise_center=False, # set each sample mean to 0
    featurewise_std_normalization=False, # divide inputs by std
```

```
          of the dataset
       samplewise_std_normalization=False, # divide each input by
          its std
       zca_whitening=False, # apply ZCA whitening
# rotation_range=10, # randomly rotate images in the range (degrees
   , 0 to 180)
# width_shift_range=0.1, # randomly shift images horizontally (
   fraction of total width)
# height_shift_range=0.1, # randomly shift images vertically (
   fraction of total height)
# horizontal_flip=True, # randomly flip images
       vertical_flip=False) # randomly flip images
    datagen.fit(a)
    model.fit_generator(datagen.flow(a,b, batch_size=32),
                        steps_per_epoch=len(a) / 32, epochs=epochs,
                           class_weight = class_weight,
                           validation_data = [c, d],callbacks = [
                           MetricsCheckpoint('logs')])
    score = model.evaluate(c,d, verbose=0)
    print('\nKeras CNN #2B - accuracy:', score[1],'\n')
    Y_pred = model.predict(c)
    print('\n', sklearn.metrics.classification_report(np.where(d >
       0)[1], np.argmax(Y_pred, axis=1), target_names=list(
       dict_characters.values())), sep='')
    Y_pred_classes = np.argmax(Y_pred,axis = 1)
    Y_true = np.argmax(d,axis = 1)
    confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)
    plot_confusion_matrix(confusion_mtx, classes = list(
       dict_characters.values()))

#Make Data 1D for compatability upsampling methods
X_trainShape = X_train.shape[1]*X_train.shape[2]*X_train.shape[3]
X_testShape = X_test.shape[1]*X_test.shape[2]*X_test.shape[3]
X_trainFlat = X_train.reshape(X_train.shape[0], X_trainShape)
X_testFlat = X_test.reshape(X_test.shape[0], X_testShape)
print("X_train Shape: ",X_train.shape)
print("X_test Shape: ",X_test.shape)
print("X_trainFlat Shape: ",X_trainFlat.shape)
print("X_testFlat Shape: ",X_testFlat.shape)

#downsize
ros = RandomUnderSampler(ratio='auto')
X_trainRos, Y_trainRos = ros.fit_sample(X_trainFlat, Y_train)
X_testRos, Y_testRos = ros.fit_sample(X_testFlat, Y_test)

# Encode labels to hot vectors (ex : 2 -> [0,0,1,0,0,0,0,0,0,0])
Y_trainRosHot = to_categorical(Y_trainRos, num_classes = 8)
Y_testRosHot = to_categorical(Y_testRos, num_classes = 8)
```

```
print("X_train: ", X_train.shape)
print("X_trainFlat: ", X_trainFlat.shape)
print("X_trainRos Shape: ",X_trainRos.shape)
print("X_testRos Shape: ",X_testRos.shape)

for i in range(len(X_trainRos)):
    height, width, channels = 64,64,3
    X_trainRosReshaped = X_trainRos.reshape(len(X_trainRos),height,
       width,channels)
print("X_trainRos Shape: ",X_trainRos.shape)
print("X_trainRosReshaped Shape: ",X_trainRosReshaped.shape)

for i in range(len(X_testRos)):
    height, width, channels = 64,64,3
    X_testRosReshaped = X_testRos.reshape(len(X_testRos),height,
       width,channels)
print("X_testRos Shape: ",X_testRos.shape)
print("X_testRosReshaped Shape: ",X_testRosReshaped.shape)

dfRos = pd.DataFrame()
dfRos["labels"]=Y_trainRos
labRos = dfRos['labels']
distRos = lab.value_counts()
sns.countplot(labRos)
print(dict_characters)

from sklearn.utils import class_weight
class_weight = class_weight.compute_class_weight('balanced', np.
   unique(y), y)
print("Old Class Weights: ",class_weight)
from sklearn.utils import class_weight
class_weight = class_weight.compute_class_weight('balanced', np.
   unique(Y_trainRos), Y_trainRos)
print("New Class Weights: ",class_weight)

#runCNNconfusion(X_trainRosReshaped[:10000], Y_trainRosHot[:10000],
    X_testRosReshaped[:3000], Y_testRosHot[:3000])
runCNNconfusion(X_trainRosReshaped, Y_trainRosHot,
   X_testRosReshaped, Y_testRosHot)
plotKerasLearningCurve()

#------------------------------------------
import pandas as pd
import numpy as np
import os
import itertools
import sklearn
import matplotlib.pylab as plt
```

```
import cv2
import seaborn as sns
from glob import glob
from sklearn.model_selection import train_test_split
from keras.utils.np_utils import to_categorical
from keras.callbacks import Callback
from keras.layers import Dense, Dropout, Flatten
from keras.optimizers import RMSprop
from keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import confusion_matrix
from keras.models import Sequential
from keras.layers import Conv2D, MaxPool2D, BatchNormalization
from imblearn.under_sampling import RandomUnderSampler

PATH = os.path.abspath(os.path.join('..', 'Project 2')) #need to
    change
labels = pd.read_csv('..//Project 2/sample/Data_Entry_2017.csv') #
    need to change

SOURCE_IMAGES = os.path.join(PATH, "sample", "images") #need to
    change
images = glob(os.path.join(SOURCE_IMAGES, "*.png"))

labels = labels[['Image Index','Finding Labels','Follow-up #','
    Patient ID','Patient Age','Patient Gender']]

#Preprocess Data
def proc_images():
    """
    Returns two arrays:
        x is an array of resized images
        y is an array of labels
    """
    NoFinding = "No Finding" #0
    Consolidation="Consolidation" #1
    Infiltration="Infiltration" #2
    Pneumothorax="Pneumothorax" #3
    Edema="Edema" # 7
    Emphysema="Emphysema" #7
    Fibrosis="Fibrosis" #7
    Effusion="Effusion" #4
    Pneumonia="Pneumonia" #7
    Pleural_Thickening="Pleural_Thickening" #7
    Cardiomegaly="Cardiomegaly" #7
    NoduleMass="Nodule" #5
    Hernia="Hernia" #7
    Atelectasis="Atelectasis" #6
    RareClass = ["Edema", "Emphysema", "Fibrosis", "Pneumonia", "
```

```
    Pleural_Thickening", "Cardiomegaly","Hernia"]
x = [] # images as arrays
y = [] # labels Infiltration or Not_infiltration
WIDTH = 64
HEIGHT = 64
for img in images:
    base = os.path.basename(img)
    finding = labels["Finding Labels"][labels["Image Index"] ==
        base].values[0]
    # Read and resize image
    full_size_image = cv2.imread(img)
    x.append(cv2.resize(full_size_image, (WIDTH,HEIGHT),
        interpolation=cv2.INTER_CUBIC))
    # Labels
    if NoFinding in finding:
        finding = 0
        y.append(finding)
    elif Consolidation in finding:
        finding = 1
        y.append(finding)
    elif Infiltration in finding:
        finding = 2
        y.append(finding)
    elif Pneumothorax in finding:
        finding = 3
        y.append(finding)
    elif Edema in finding:
        finding = 7
        y.append(finding)
    elif Emphysema in finding:
        finding = 7
        y.append(finding)
    elif Fibrosis in finding:
        finding = 7
        y.append(finding)
    elif Effusion in finding:
        finding = 4
        y.append(finding)
    elif Pneumonia in finding:
        finding = 7
        y.append(finding)
    elif Pleural_Thickening in finding:
        finding = 7
        y.append(finding)
    elif Cardiomegaly in finding:
        finding = 7
        y.append(finding)
    elif NoduleMass in finding:
```

```
            finding = 5
            y.append(finding)
        elif Hernia in finding:
            finding = 7
            y.append(finding)
        elif Atelectasis in finding:
            finding = 6
            y.append(finding)
        else:
            finding = 7
            y.append(finding)
    return x,y

#part 2
X,y = proc_images()
df = pd.DataFrame()
df["images"]=X
df["labels"]=y

#Describe new numpy arrays
dict_characters = {0: 'No Finding', 1: 'Consolidation', 2: '
    Infiltration',
        3: 'Pneumothorax', 4:'Effusion', 5: 'Nodule Mass', 6: '
            Atelectasis', 7: "Other Rare Classes"}

print(df.head(10))
print("")
print(dict_characters)

#show unbalance
lab = df['labels']
dist = lab.value_counts()
sns.countplot(lab)
print(dict_characters)

X=np.array(X)
X=X/255.0

#test&train

X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size
    =0.2)
print("Training Data Shape:", X_train.shape)
print("Testing Data Shape:", X_test.shape)
print("Training Data Shape:", len(X_train), X_train[0].shape)
print("Testing Data Shape:", len(X_test), X_test[0].shape)

# try to use a CNN to predict each ailment based off of the X-Ray
```

```
    image.
# Encode labels to hot vectors (ex : 2 -> [0,0,1,0,0,0,0,0,0,0])
Y_trainHot = to_categorical(Y_train, num_classes = 8)
Y_testHot = to_categorical(Y_test, num_classes = 8)

# In order to avoid having a biased model because of skewed class
    sizes,
# We need to modify the class_weights parameter in order to give
    more weight to the rare classes.
# In this case the class_weights parameter will eventually be
    passed to the model.fit function.

from sklearn.utils import class_weight
class_weight = class_weight.compute_class_weight('balanced', np.
    unique(y), y)
print(class_weight)

# Helper Functions Learning Curves and Confusion Matrix


class MetricsCheckpoint(Callback):
    """Callback that saves metrics after each epoch"""
    def __init__(self, savepath):
        super(MetricsCheckpoint, self).__init__()
        self.savepath = savepath
        self.history = {}
    def on_epoch_end(self, epoch, logs=None):
        for k, v in logs.items():
            self.history.setdefault(k, []).append(v)
        np.save(self.savepath, self.history)

def plotKerasLearningCurve():
    plt.figure(figsize=(10,5))
    metrics = np.load('logs.npy')[()]
    filt = ['acc'] # try to add 'loss' to see the loss learning
        curve
    for k in filter(lambda x : np.any([kk in x for kk in filt]),
        metrics.keys()):
        l = np.array(metrics[k])
        plt.plot(l, c= 'r' if 'val' not in k else 'b', label='val'
            if 'val' in k else 'train')
        x = np.argmin(l) if 'loss' in k else np.argmax(l)
        y = l[x]
        plt.scatter(x,y, lw=0, alpha=0.25, s=100, c='r' if 'val' not
             in k else 'b')
        plt.text(x, y, '{} = {:.4f}'.format(x,y), size='15', color=
            'r' if 'val' not in k else 'b')
    plt.legend(loc=4)
```

```
    plt.axis([0, None, None, None]);
    plt.grid()
    plt.xlabel('Number of epochs')

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.figure(figsize = (5,5))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=90)
    plt.yticks(tick_marks, classes)
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.
       shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

#CNN
def runCNNconfusion(a,b,c,d):
    # In -> [[Conv2D->relu]*2 -> MaxPool2D -> Dropout]*2 -> Flatten
        -> Dense -> Dropout -> Out
    batch_size = 128
    num_classes = 8
    epochs = 10
    img_rows, img_cols = X_train.shape[1],X_train.shape[2]
    input_shape = (img_rows, img_cols, 3)
    model = Sequential()
    model.add(Conv2D(filters = 32, kernel_size = (3,3),padding = '
       Same',
                     activation ='relu', input_shape = input_shape))
    model.add(Conv2D(filters = 32, kernel_size = (3,3),padding = '
       Same',
                     activation ='relu'))
```

```
    model.add(MaxPool2D(pool_size=(2,2)))
    model.add(BatchNormalization())
    model.add(Dropout(0.25))
    model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = '
        Same',
                     activation ='relu'))
    model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = '
        Same',
                     activation ='relu'))
    model.add(MaxPool2D(pool_size=(2,2)))
    model.add(BatchNormalization())
    model.add(Dropout(0.25))
    model.add(Conv2D(filters = 86, kernel_size = (3,3),padding = '
        Same',
                     activation ='relu'))
    model.add(Conv2D(filters = 86, kernel_size = (3,3),padding = '
        Same',
                     activation ='relu'))
    model.add(MaxPool2D(pool_size=(2,2)))
    model.add(BatchNormalization())
    model.add(Dropout(0.25))
    model.add(Flatten())
    #model.add(Dense(1024, activation = "relu"))
    #model.add(Dropout(0.5))
    model.add(Dense(512, activation = "relu"))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation = "softmax"))
    # Define the optimizer
    optimizer = RMSprop(lr=0.001, decay=1e-6)
    model.compile(optimizer = optimizer , loss = "
        categorical_crossentropy", metrics=["accuracy"])
    datagen = ImageDataGenerator(
        featurewise_center=False, # set input mean to 0 over the
            dataset
        samplewise_center=False, # set each sample mean to 0
        featurewise_std_normalization=False, # divide inputs by std
            of the dataset
        samplewise_std_normalization=False, # divide each input by
            its std
        zca_whitening=False, # apply ZCA whitening
# rotation_range=10, # randomly rotate images in the range (degrees
    , 0 to 180)
# width_shift_range=0.1, # randomly shift images horizontally (
    fraction of total width)
# height_shift_range=0.1, # randomly shift images vertically (
    fraction of total height)
# horizontal_flip=True, # randomly flip images
        vertical_flip=False) # randomly flip images
```

```
    datagen.fit(a)
    model.fit_generator(datagen.flow(a,b, batch_size=32),
                        steps_per_epoch=len(a) / 32, epochs=epochs,
                            class_weight = class_weight,
                            validation_data = [c, d],callbacks = [
                            MetricsCheckpoint('logs')])
    score = model.evaluate(c,d, verbose=0)
    print('\nKeras CNN #2B - accuracy:', score[1],'\n')
    Y_pred = model.predict(c)
    print('\n', sklearn.metrics.classification_report(np.where(d >
        0)[1], np.argmax(Y_pred, axis=1), target_names=list(
        dict_characters.values())), sep='')
    Y_pred_classes = np.argmax(Y_pred,axis = 1)
    Y_true = np.argmax(d,axis = 1)
    confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)
    plot_confusion_matrix(confusion_mtx, classes = list(
        dict_characters.values()))

#Make Data 1D for compatability upsampling methods
X_trainShape = X_train.shape[1]*X_train.shape[2]*X_train.shape[3]
X_testShape = X_test.shape[1]*X_test.shape[2]*X_test.shape[3]
X_trainFlat = X_train.reshape(X_train.shape[0], X_trainShape)
X_testFlat = X_test.reshape(X_test.shape[0], X_testShape)
print("X_train Shape: ",X_train.shape)
print("X_test Shape: ",X_test.shape)
print("X_trainFlat Shape: ",X_trainFlat.shape)
print("X_testFlat Shape: ",X_testFlat.shape)

#downsize
ros = RandomUnderSampler(ratio='auto')
X_trainRos, Y_trainRos = ros.fit_sample(X_trainFlat, Y_train)
X_testRos, Y_testRos = ros.fit_sample(X_testFlat, Y_test)

# Encode labels to hot vectors (ex : 2 -> [0,0,1,0,0,0,0,0,0,0])
Y_trainRosHot = to_categorical(Y_trainRos, num_classes = 8)
Y_testRosHot = to_categorical(Y_testRos, num_classes = 8)
print("X_train: ", X_train.shape)
print("X_trainFlat: ", X_trainFlat.shape)
print("X_trainRos Shape: ",X_trainRos.shape)
print("X_testRos Shape: ",X_testRos.shape)

for i in range(len(X_trainRos)):
    height, width, channels = 64,64,3
    X_trainRosReshaped = X_trainRos.reshape(len(X_trainRos),height,
        width,channels)
print("X_trainRos Shape: ",X_trainRos.shape)
print("X_trainRosReshaped Shape: ",X_trainRosReshaped.shape)
```

```
for i in range(len(X_testRos)):
    height, width, channels = 64,64,3
    X_testRosReshaped = X_testRos.reshape(len(X_testRos),height,
       width,channels)
print("X_testRos Shape: ",X_testRos.shape)
print("X_testRosReshaped Shape: ",X_testRosReshaped.shape)

dfRos = pd.DataFrame()
dfRos["labels"]=Y_trainRos
labRos = dfRos['labels']
distRos = lab.value_counts()
sns.countplot(labRos)
print(dict_characters)

from sklearn.utils import class_weight
class_weight = class_weight.compute_class_weight('balanced', np.
   unique(y), y)
print("Old Class Weights: ",class_weight)
from sklearn.utils import class_weight
class_weight = class_weight.compute_class_weight('balanced', np.
   unique(Y_trainRos), Y_trainRos)
print("New Class Weights: ",class_weight)

runCNNconfusion(X_trainRosReshaped, Y_trainRosHot,
   X_testRosReshaped, Y_testRosHot)
plotKerasLearningCurve()

#------------------------------------------------------------
import pandas as pd
import numpy as np
import os
import itertools
import sklearn
import matplotlib.pylab as plt
import cv2
import seaborn as sns
from glob import glob
from sklearn.model_selection import train_test_split
from keras.utils.np_utils import to_categorical
from keras.callbacks import Callback
from keras.layers import Dense, Dropout, Flatten
from keras.optimizers import RMSprop
from keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import confusion_matrix
from keras.models import Sequential
from keras.layers import Conv2D, MaxPool2D, BatchNormalization
from imblearn.over_sampling import RandomOverSampler
```

```
PATH = os.path.abspath(os.path.join('..', 'Project 2'))
labels = pd.read_csv('..//Project 2/sample/Data_Entry_2017.csv')

SOURCE_IMAGES = os.path.join(PATH, "sample", "images")
images = glob(os.path.join(SOURCE_IMAGES, "*.png"))

labels = labels[['Image Index','Finding Labels','Follow-up #','
   Patient ID','Patient Age','Patient Gender']]

#Preprocess Data
def proc_images():
    """
    Returns two arrays:
        x is an array of resized images
        y is an array of labels
    """
    NoFinding = "No Finding" #0
    Consolidation="Consolidation" #1
    Infiltration="Infiltration" #2
    Pneumothorax="Pneumothorax" #3
    Edema="Edema" # 7
    Emphysema="Emphysema" #7
    Fibrosis="Fibrosis" #7
    Effusion="Effusion" #4
    Pneumonia="Pneumonia" #7
    Pleural_Thickening="Pleural_Thickening" #7
    Cardiomegaly="Cardiomegaly" #7
    NoduleMass="Nodule" #5
    Hernia="Hernia" #7
    Atelectasis="Atelectasis" #6
    RareClass = ["Edema", "Emphysema", "Fibrosis", "Pneumonia", "
       Pleural_Thickening", "Cardiomegaly","Hernia"]
    x = [] # images as arrays
    y = [] # labels Infiltration or Not_infiltration
    WIDTH = 64
    HEIGHT = 64
    for img in images:
        base = os.path.basename(img)
        finding = labels["Finding Labels"][labels["Image Index"] ==
           base].values[0]
        # Read and resize image
        full_size_image = cv2.imread(img)
        x.append(cv2.resize(full_size_image, (WIDTH,HEIGHT),
           interpolation=cv2.INTER_CUBIC))
        # Labels
        if NoFinding in finding:
            finding = 0
            y.append(finding)
```

```
        elif Consolidation in finding:
            finding = 1
            y.append(finding)
        elif Infiltration in finding:
            finding = 2
            y.append(finding)
        elif Pneumothorax in finding:
            finding = 3
            y.append(finding)
        elif Edema in finding:
            finding = 7
            y.append(finding)
        elif Emphysema in finding:
            finding = 7
            y.append(finding)
        elif Fibrosis in finding:
            finding = 7
            y.append(finding)
        elif Effusion in finding:
            finding = 4
            y.append(finding)
        elif Pneumonia in finding:
            finding = 7
            y.append(finding)
        elif Pleural_Thickening in finding:
            finding = 7
            y.append(finding)
        elif Cardiomegaly in finding:
            finding = 7
            y.append(finding)
        elif NoduleMass in finding:
            finding = 5
            y.append(finding)
        elif Hernia in finding:
            finding = 7
            y.append(finding)
        elif Atelectasis in finding:
            finding = 6
            y.append(finding)
        else:
            finding = 7
            y.append(finding)
    return x,y

#part 2
X,y = proc_images()
df = pd.DataFrame()
df["images"]=X
```

```
df["labels"]=y

#Describe new numpy arrays
dict_characters = {0: 'No Finding', 1: 'Consolidation', 2: '
   Infiltration',
       3: 'Pneumothorax', 4:'Effusion', 5: 'Nodule Mass', 6: '
          Atelectasis', 7: "Other Rare Classes"}

print(df.head(10))
print("")
print(dict_characters)

#show unbalance
lab = df['labels']
dist = lab.value_counts()
sns.countplot(lab)
print(dict_characters)

X=np.array(X)
X=X/255.0

#test&train

X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size
   =0.2)
print("Training Data Shape:", X_train.shape)
print("Testing Data Shape:", X_test.shape)
print("Training Data Shape:", len(X_train), X_train[0].shape)
print("Testing Data Shape:", len(X_test), X_test[0].shape)

# try to use a CNN to predict each ailment based off of the X-Ray
   image.
# Encode labels to hot vectors (ex : 2 -> [0,0,1,0,0,0,0,0,0,0])
Y_trainHot = to_categorical(Y_train, num_classes = 8)
Y_testHot = to_categorical(Y_test, num_classes = 8)

# In order to avoid having a biased model because of skewed class
   sizes,
# We need to modify the class_weights parameter in order to give
   more weight to the rare classes.
# In this case the class_weights parameter will eventually be
   passed to the model.fit function.

from sklearn.utils import class_weight
class_weight = class_weight.compute_class_weight('balanced', np.
   unique(y), y)
print(class_weight)
```

```
# Helper Functions Learning Curves and Confusion Matrix


class MetricsCheckpoint(Callback):
    """Callback that saves metrics after each epoch"""
    def __init__(self, savepath):
        super(MetricsCheckpoint, self).__init__()
        self.savepath = savepath
        self.history = {}
    def on_epoch_end(self, epoch, logs=None):
        for k, v in logs.items():
            self.history.setdefault(k, []).append(v)
        np.save(self.savepath, self.history)

def plotKerasLearningCurve():
    plt.figure(figsize=(10,5))
    metrics = np.load('logs.npy')[()]
    filt = ['acc'] # try to add 'loss' to see the loss learning
       curve
    for k in filter(lambda x : np.any([kk in x for kk in filt]),
       metrics.keys()):
        l = np.array(metrics[k])
        plt.plot(l, c= 'r' if 'val' not in k else 'b', label='val'
           if 'val' in k else 'train')
        x = np.argmin(l) if 'loss' in k else np.argmax(l)
        y = l[x]
        plt.scatter(x,y, lw=0, alpha=0.25, s=100, c='r' if 'val' not
            in k else 'b')
        plt.text(x, y, '{} = {:.4f}'.format(x,y), size='15', color=
           'r' if 'val' not in k else 'b')
    plt.legend(loc=4)
    plt.axis([0, None, None, None]);
    plt.grid()
    plt.xlabel('Number of epochs')

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.figure(figsize = (5,5))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
```

```
    plt.xticks(tick_marks, classes, rotation=90)
    plt.yticks(tick_marks, classes)
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.
       shape[1])):
        plt.text(j, i, cm[i, j],
                horizontalalignment="center",
                color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

#CNN
def runCNNconfusion(a,b,c,d):
    # In -> [[Conv2D->relu]*2 -> MaxPool2D -> Dropout]*2 -> Flatten
        -> Dense -> Dropout -> Out
    batch_size = 128
    num_classes = 8
    epochs = 10
    img_rows, img_cols = X_train.shape[1],X_train.shape[2]
    input_shape = (img_rows, img_cols, 3)
    model = Sequential()
    model.add(Conv2D(filters = 32, kernel_size = (3,3),padding = '
       Same',
                     activation ='relu', input_shape = input_shape))
    model.add(Conv2D(filters = 32, kernel_size = (3,3),padding = '
       Same',
                     activation ='relu'))
    model.add(MaxPool2D(pool_size=(2,2)))
    model.add(BatchNormalization())
    model.add(Dropout(0.25))
    model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = '
       Same',
                     activation ='relu'))
    model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = '
       Same',
                     activation ='relu'))
    model.add(MaxPool2D(pool_size=(2,2)))
    model.add(BatchNormalization())
    model.add(Dropout(0.25))
    model.add(Conv2D(filters = 86, kernel_size = (3,3),padding = '
       Same',
                     activation ='relu'))
    model.add(Conv2D(filters = 86, kernel_size = (3,3),padding = '
       Same',
```

```
                    activation ='relu'))
    model.add(MaxPool2D(pool_size=(2,2)))
    model.add(BatchNormalization())
    model.add(Dropout(0.25))
    model.add(Flatten())
    #model.add(Dense(1024, activation = "relu"))
    #model.add(Dropout(0.5))
    model.add(Dense(512, activation = "relu"))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation = "softmax"))
    # Define the optimizer
    optimizer = RMSprop(lr=0.001, decay=1e-6)
    model.compile(optimizer = optimizer , loss = "
       categorical_crossentropy", metrics=["accuracy"])
    datagen = ImageDataGenerator(
        featurewise_center=False, # set input mean to 0 over the
           dataset
        samplewise_center=False, # set each sample mean to 0
        featurewise_std_normalization=False, # divide inputs by std
           of the dataset
        samplewise_std_normalization=False, # divide each input by
           its std
        zca_whitening=False, # apply ZCA whitening
# rotation_range=10, # randomly rotate images in the range (degrees
    , 0 to 180)
# width_shift_range=0.1, # randomly shift images horizontally (
    fraction of total width)
# height_shift_range=0.1, # randomly shift images vertically (
    fraction of total height)
# horizontal_flip=True, # randomly flip images
        vertical_flip=False) # randomly flip images
    datagen.fit(a)
    model.fit_generator(datagen.flow(a,b, batch_size=32),
                        steps_per_epoch=len(a) / 32, epochs=epochs,
                           class_weight = class_weight,
                           validation_data = [c, d],callbacks = [
                           MetricsCheckpoint('logs')])
    score = model.evaluate(c,d, verbose=0)
    print('\nKeras CNN #2B - accuracy:', score[1],'\n')
    Y_pred = model.predict(c)
    print('\n', sklearn.metrics.classification_report(np.where(d >
       0)[1], np.argmax(Y_pred, axis=1), target_names=list(
       dict_characters.values())), sep='')
    Y_pred_classes = np.argmax(Y_pred,axis = 1)
    Y_true = np.argmax(d,axis = 1)
    confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)
    plot_confusion_matrix(confusion_mtx, classes = list(
       dict_characters.values()))
```

```
#Make Data 1D for compatability upsampling methods
X_trainShape = X_train.shape[1]*X_train.shape[2]*X_train.shape[3]
X_testShape = X_test.shape[1]*X_test.shape[2]*X_test.shape[3]
X_trainFlat = X_train.reshape(X_train.shape[0], X_trainShape)
X_testFlat = X_test.reshape(X_test.shape[0], X_testShape)
print("X_train Shape: ",X_train.shape)
print("X_test Shape: ",X_test.shape)
print("X_trainFlat Shape: ",X_trainFlat.shape)
print("X_testFlat Shape: ",X_testFlat.shape)

#UPsizing
ros = RandomOverSampler(ratio='auto')
X_trainRos, Y_trainRos = ros.fit_sample(X_trainFlat, Y_train)
X_testRos, Y_testRos = ros.fit_sample(X_testFlat, Y_test)

# Encode labels to hot vectors (ex : 2 -> [0,0,1,0,0,0,0,0,0,0])
Y_trainRosHot = to_categorical(Y_trainRos, num_classes = 8)
Y_testRosHot = to_categorical(Y_testRos, num_classes = 8)
print("X_train: ", X_train.shape)
print("X_trainFlat: ", X_trainFlat.shape)
print("X_trainRos Shape: ",X_trainRos.shape)
print("X_testRos Shape: ",X_testRos.shape)

for i in range(len(X_trainRos)):
    height, width, channels = 64,64,3
    X_trainRosReshaped = X_trainRos.reshape(len(X_trainRos),height,
       width,channels)
print("X_trainRos Shape: ",X_trainRos.shape)
print("X_trainRosReshaped Shape: ",X_trainRosReshaped.shape)

for i in range(len(X_testRos)):
    height, width, channels = 64,64,3
    X_testRosReshaped = X_testRos.reshape(len(X_testRos),height,
       width,channels)
print("X_testRos Shape: ",X_testRos.shape)
print("X_testRosReshaped Shape: ",X_testRosReshaped.shape)

dfRos = pd.DataFrame()
dfRos["labels"]=Y_trainRos
labRos = dfRos['labels']
distRos = lab.value_counts()
sns.countplot(labRos)
print(dict_characters)

from sklearn.utils import class_weight
class_weight = class_weight.compute_class_weight('balanced', np.
   unique(y), y)
```

```
print("Old Class Weights: ",class_weight)
from sklearn.utils import class_weight
class_weight = class_weight.compute_class_weight('balanced', np.
   unique(Y_trainRos), Y_trainRos)
print("New Class Weights: ",class_weight)

runCNNconfusion(X_trainRosReshaped, Y_trainRosHot,
   X_testRosReshaped, Y_testRosHot)
plotKerasLearningCurve()
#------------------------------------------------
import pandas as pd
import numpy as np
import os
from glob import glob
import random
import matplotlib.pylab as plt
import cv2
import matplotlib.gridspec as gridspec
import seaborn as sns
import zlib

PATH = os.path.abspath(os.path.join('..', 'Project 2'))
SOURCE_IMAGES = os.path.join(PATH, "sample", "images")
images = glob(os.path.join(SOURCE_IMAGES, "*.png"))

images[0:10] # image paths

labels = pd.read_csv('../Project 2/sample/sample_labels.csv')
labels.head(10) #first 10 rows in csv file

# Plot a representative image (install OpenCV in python 3.6)
multipleImages = glob('/Users/sherrylin/Desktop/DA6813 Application/
   Project 2/sample/images/**')
i_ = 0
plt.rcParams['figure.figsize'] = (10.0, 10.0)
plt.subplots_adjust(wspace=0, hspace=0)
for l in multipleImages[:25]:
    im = cv2.imread(l)
    im = cv2.resize(im, (64, 64))
    plt.subplot(3, 3, i_+1) #set_title(l) & set the table size to
       open image
    plt.imshow(cv2.cvtColor(im, cv2.COLOR_BGR2RGB)); plt.axis('off
       ')
    i_ += 1

# What types of ailments are identified in these annotated X-ray
   images? (install seaborn for python3.6)
```

```
#drop unused columns
labels = labels[['Image Index','Finding Labels','Follow-up #','
   Patient ID','Patient Age','Patient Gender']]
#create new columns for each decease
pathology_list = ['Cardiomegaly','Emphysema','Effusion','Hernia','
   Nodule','Pneumothorax','Atelectasis','Pleural_Thickening','Mass
   ','Edema','Consolidation','Infiltration','Fibrosis','Pneumonia']
for pathology in pathology_list :
    labels[pathology] = labels['Finding Labels'].apply(lambda x: 1
       if pathology in x else 0)
#remove Y after age
labels['Age']=labels['Patient Age'].apply(lambda x: x[:-1]).astype(
   int)

plt.figure(figsize=(15,10))
gs = gridspec.GridSpec(8,1)
ax1 = plt.subplot(gs[:7, :])
ax2 = plt.subplot(gs[7, :])
data1 = pd.melt(labels,
            id_vars=['Patient Gender'],
            value_vars = list(pathology_list),
            var_name = 'Category',
            value_name = 'Count')
data1 = data1.loc[data1.Count>0]
g=sns.countplot(y='Category',hue='Patient Gender',data=data1, ax=
   ax1, order = data1['Category'].value_counts().index)
ax1.set( ylabel="",xlabel="")
ax1.legend(fontsize=20)
ax1.set_title('X Ray partition (total number = 121120)',fontsize
   =18);

labels['Nothing']=labels['Finding Labels'].apply(lambda x: 1 if 'No
    Finding' in x else 0)

data2 = pd.melt(labels,
            id_vars=['Patient Gender'],
            value_vars = list(['Nothing']),
            var_name = 'Category',
            value_name = 'Count')
data2 = data2.loc[data2.Count>0]
g=sns.countplot(y='Category',hue='Patient Gender',data=data2,ax=ax2
   )
ax2.set( ylabel="",xlabel="Number of decease")
ax2.legend('')
plt.subplots_adjust(hspace=.5)


# Boxplot (Ailments vs Age)
```

```
f, axarr = plt.subplots(7, 2, sharex=True,figsize=(10, 15))

i=0
j=0
x=np.arange(0,100,10)
for pathology in pathology_list :
    g=sns.boxplot(x='Age', hue="Patient Gender",data=labels[labels
       ['Finding Labels']==pathology], ax=axarr[i, j])
    axarr[i, j].set_title(pathology)
    g.set_xlim(0,90)
    g.set_xticks(x)
    g.set_xticklabels(x)
    j=(j+1)%2
    if j==0:
        i=(i+1)%7
f.subplots_adjust(hspace=0.3)

#Age vs Count by Gender
f, axarr = plt.subplots(7, 2, sharex=True,figsize=(15, 20))

i=0
j=0
x=np.arange(0,100,10)
for pathology in pathology_list :
    g=sns.countplot(x='Age', hue="Patient Gender",data=labels[
       labels['Finding Labels']==pathology], ax=axarr[i, j])
    axarr[i, j].set_title(pathology)
    g.set_xlim(0,90)
    g.set_xticks(x)
    g.set_xticklabels(x)
    j=(j+1)%2
    if j==0:
        i=(i+1)%7
f.subplots_adjust(hspace=0.3)

# Convert annotated .png images into labeled numpy arrays
def proc_images():
    """
    Returns two arrays:
        x is an array of resized images
        y is an array of labels
    """
    NoFinding = "No Finding" #0
    Consolidation="Consolidation" #1
    Infiltration="Infiltration" #2
    Pneumothorax="Pneumothorax" #3
    Edema="Edema" # 7
    Emphysema="Emphysema" #7
```

```
Fibrosis="Fibrosis" #7
Effusion="Effusion" #4
Pneumonia="Pneumonia" #7
Pleural_Thickening="Pleural_Thickening" #7
Cardiomegaly="Cardiomegaly" #7
NoduleMass="Nodule" #5
Hernia="Hernia" #7
Atelectasis="Atelectasis" #6
RareClass = ["Edema", "Emphysema", "Fibrosis", "Pneumonia", "
   Pleural_Thickening", "Cardiomegaly","Hernia"]
x = [] # images as arrays
y = [] # labels Infiltration or Not_infiltration
WIDTH = 64
HEIGHT = 64
for img in images:
    base = os.path.basename(img)
    finding = labels["Finding Labels"][labels["Image Index"] ==
       base].values[0]
    # Read and resize image
    full_size_image = cv2.imread(img)
    x.append(cv2.resize(full_size_image, (WIDTH,HEIGHT),
       interpolation=cv2.INTER_CUBIC))
    # Labels
    if NoFinding in finding:
        finding = 0
        y.append(finding)
    elif Consolidation in finding:
        finding = 1
        y.append(finding)
    elif Infiltration in finding:
        finding = 2
        y.append(finding)
    elif Pneumothorax in finding:
        finding = 3
        y.append(finding)
    elif Edema in finding:
        finding = 7
        y.append(finding)
    elif Emphysema in finding:
        finding = 7
        y.append(finding)
    elif Fibrosis in finding:
        finding = 7
        y.append(finding)
    elif Effusion in finding:
        finding = 4
        y.append(finding)
    elif Pneumonia in finding:
```

```
            finding = 7
            y.append(finding)
        elif Pleural_Thickening in finding:
            finding = 7
            y.append(finding)
        elif Cardiomegaly in finding:
            finding = 7
            y.append(finding)
        elif NoduleMass in finding:
            finding = 5
            y.append(finding)
        elif Hernia in finding:
            finding = 7
            y.append(finding)
        elif Atelectasis in finding:
            finding = 6
            y.append(finding)
        else:
            finding = 7
            y.append(finding)
    return x,y

# use a for loop to shorten this function
# put variables into a dictionary(take few seconds)

X,y = proc_images()
df = pd.DataFrame()
df["images"]=X
df["labels"]=y
#print(len(df), df.images[0].shape)
#print(type(X))

#Describe new numpy arrays
dict_characters = {0: 'No Finding', 1: 'Consolidation', 2: '
    Infiltration',
        3: 'Pneumothorax', 4:'Effusion', 5: 'Nodule Mass', 6: '
            Atelectasis', 7: "Other Rare Classes"}

print(df.head(10))
print("")
print(dict_characters)

# Describe the distribution of pixel intensities within a
    representative image
def plotHistogram(a):
    """
    Plot histogram of RGB Pixel Intensities
    """
```

```
    plt.figure(figsize=(10,5))
    plt.subplot(1,2,1)
    plt.title('Representative Image')
    b = cv2.resize(a, (512,512))
    plt.imshow(b)
    plt.axis('off')
    histo = plt.subplot(1,2,2)
    histo.set_ylabel('Count')
    histo.set_xlabel('Pixel Intensity')
    n_bins = 30
    plt.hist(a[:,:,0].flatten(), bins= n_bins, lw = 0, color='r',
       alpha=0.5);
    plt.hist(a[:,:,1].flatten(), bins= n_bins, lw = 0, color='g',
       alpha=0.5);
    plt.hist(a[:,:,2].flatten(), bins= n_bins, lw = 0, color='b',
       alpha=0.5);
plotHistogram(X[1])
# Normalize the pixel intensities between zero and one.
X=np.array(X)
X=X/255.0
plotHistogram(X[1])

# Describe distribution of class labels (graph to show unbalance
   between all class)
# {0: 'No Finding', 1: 'Consolidation', 2: 'Infiltration', 3: '
   Pneumothorax', 4: 'Effusion', 5: 'Nodule Mass', 6: 'Atelectasis
   ', 7: 'Other Rare Classes'}
lab = df['labels']
dist = lab.value_counts()
sns.countplot(lab)
print(dict_characters)


#We have imbalanced sample sizes. This is a problem that needs to
   be addressed.
#But for now we can proceed with a preliminary analysis.
#install: keras

from sklearn.model_selection import train_test_split
from keras.utils.np_utils import to_categorical # convert to one-
   hot-encoding

X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size
   =0.2)
# Reduce Sample Size for DeBugging
X_train = X_train[0:5000]
Y_train = Y_train[0:5000]
X_test = X_test[0:2000]
```

```
Y_test = Y_test[0:2000]

print("Training Data Shape:", X_train.shape)
print("Testing Data Shape:", X_test.shape)
print("Training Data Shape:", len(X_train), X_train[0].shape)
print("Testing Data Shape:", len(X_test), X_test[0].shape)

# Now, try to use a CNN to predict each ailment based off of the X-
   Ray image.

# Encode labels to hot vectors (ex : 2 -> [0,0,1,0,0,0,0,0,0,0])
Y_trainHot = to_categorical(Y_train, num_classes = 8)
Y_testHot = to_categorical(Y_test, num_classes = 8)

# In order to avoid having a biased model because of skewed class
   sizes,
# We need to modify the class_weights parameter in order to give
   more weight to the rare classes.
# In this case the class_weights parameter will eventually be
   passed to the model.fit function.

from sklearn.utils import class_weight
class_weight = class_weight.compute_class_weight('balanced', np.
   unique(y), y)
print(class_weight)

# Helper Functions Learning Curves and Confusion Matrix

from keras.callbacks import Callback, EarlyStopping,
   ReduceLROnPlateau, ModelCheckpoint

class MetricsCheckpoint(Callback):
    """Callback that saves metrics after each epoch"""
    def __init__(self, savepath):
        super(MetricsCheckpoint, self).__init__()
        self.savepath = savepath
        self.history = {}
    def on_epoch_end(self, epoch, logs=None):
        for k, v in logs.items():
            self.history.setdefault(k, []).append(v)
        np.save(self.savepath, self.history)

def plotKerasLearningCurve():
    plt.figure(figsize=(10,5))
    metrics = np.load('logs.npy')[()]
    filt = ['acc'] # try to add 'loss' to see the loss learning
       curve
    for k in filter(lambda x : np.any([kk in x for kk in filt]),
```

```
        metrics.keys()):
        l = np.array(metrics[k])
        plt.plot(l, c= 'r' if 'val' not in k else 'b', label='val'
            if 'val' in k else 'train')
        x = np.argmin(l) if 'loss' in k else np.argmax(l)
        y = l[x]
        plt.scatter(x,y, lw=0, alpha=0.25, s=100, c='r' if 'val' not
             in k else 'b')
        plt.text(x, y, '{} = {:.4f}'.format(x,y), size='15', color=
            'r' if 'val' not in k else 'b')
    plt.legend(loc=4)
    plt.axis([0, None, None, None]);
    plt.grid()
    plt.xlabel('Number of epochs')

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.figure(figsize = (5,5))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=90)
    plt.yticks(tick_marks, classes)
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.
        shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

#CNN
def runCNNconfusion(a,b,c,d):
    # In -> [[Conv2D->relu]*2 -> MaxPool2D -> Dropout]*2 -> Flatten
         -> Dense -> Dropout -> Out
    batch_size = 128
```

```
num_classes = 8
epochs = 10
img_rows, img_cols = X_train.shape[1],X_train.shape[2]
input_shape = (img_rows, img_cols, 3)
model = Sequential()
model.add(Conv2D(filters = 32, kernel_size = (3,3),padding = '
    Same',
                 activation ='relu', input_shape = input_shape))
model.add(Conv2D(filters = 32, kernel_size = (3,3),padding = '
    Same',
                 activation ='relu'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Dropout(0.25))
model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = '
    Same',
                 activation ='relu'))
model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = '
    Same',
                 activation ='relu'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Dropout(0.25))
model.add(Conv2D(filters = 86, kernel_size = (3,3),padding = '
    Same',
                 activation ='relu'))
model.add(Conv2D(filters = 86, kernel_size = (3,3),padding = '
    Same',
                 activation ='relu'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Dropout(0.25))
model.add(Flatten())
#model.add(Dense(1024, activation = "relu"))
#model.add(Dropout(0.5))
model.add(Dense(512, activation = "relu"))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation = "softmax"))
# Define the optimizer
optimizer = RMSprop(lr=0.001, decay=1e-6)
model.compile(optimizer = optimizer , loss = "
    categorical_crossentropy", metrics=["accuracy"])
datagen = ImageDataGenerator(
    featurewise_center=False, # set input mean to 0 over the
       dataset
    samplewise_center=False, # set each sample mean to 0
    featurewise_std_normalization=False, # divide inputs by std
       of the dataset
```

```
        samplewise_std_normalization=False, # divide each input by
            its std
        zca_whitening=False, # apply ZCA whitening
# rotation_range=10, # randomly rotate images in the range (degrees
    , 0 to 180)
# width_shift_range=0.1, # randomly shift images horizontally (
    fraction of total width)
# height_shift_range=0.1, # randomly shift images vertically (
    fraction of total height)
# horizontal_flip=True, # randomly flip images
        vertical_flip=False) # randomly flip images
    datagen.fit(a)
    model.fit_generator(datagen.flow(a,b, batch_size=32),
                        steps_per_epoch=len(a) / 32, epochs=epochs,
                            class_weight = class_weight,
                            validation_data = [c, d],callbacks = [
                            MetricsCheckpoint('logs')])
    score = model.evaluate(c,d, verbose=0)
    print('\nKeras CNN #2B - accuracy:', score[1],'\n')
    Y_pred = model.predict(c)
    print('\n', sklearn.metrics.classification_report(np.where(d >
        0)[1], np.argmax(Y_pred, axis=1), target_names=list(
        dict_characters.values())), sep='')
    Y_pred_classes = np.argmax(Y_pred,axis = 1)
    Y_true = np.argmax(d,axis = 1)
    confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)
    plot_confusion_matrix(confusion_mtx, classes = list(
        dict_characters.values()))

# Warning!!!! take more than 10 mins
#runCNNconfusion(X_train, Y_trainHot, X_test, Y_testHot)
# the result is: 847ms/step loss: 1.5077 - acc: 0.5406 - val_loss:
    1.5105 - val_acc: 0.5508
# Keras CNN #2B - accuracy: 0.550802139037

#plotKerasLearningCurve()

# The imbalance in our dataset has resulted in a biased model.
# Tried to prevent this by modifying the class_weights parameter
    and using in the model.fit function but apparently that was not
    enough.
# Now, we try to compensate for the imbalanced sample size by
    oversampling or upsampling the minority classes.

lab = df['labels']
dist = lab.value_counts()
sns.countplot(lab)
print(dict_characters)
```

```
# It is very import to do upsampling AFTER the train_test_split
   function
# otherwise you can end up with values in the testing dataset that
   are related to the values within the training dataset.
X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size
   =0.2)
# Reduce Sample Size for DeBugging
X_train = X_train[0:5000]
Y_train = Y_train[0:5000]
X_test = X_test[0:2000]
Y_test = Y_test[0:2000]

import numpy as np
# Make Data 1D for compatability upsampling methods
X_trainShape = X_train.shape[1]*X_train.shape[2]*X_train.shape[3]
X_testShape = X_test.shape[1]*X_test.shape[2]*X_test.shape[3]
X_trainFlat = X_train.reshape(X_train.shape[0], X_trainShape)
X_testFlat = X_test.reshape(X_test.shape[0], X_testShape)
print("X_train Shape: ",X_train.shape) #(4484, 64, 64, 3)
print("X_test Shape: ",X_test.shape) #(1122, 64, 64, 3)
print("X_trainFlat Shape: ",X_trainFlat.shape) #(4484, 12288)
print("X_testFlat Shape: ",X_testFlat.shape) #(1122, 12288)

from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(ratio='auto')
X_trainRos, Y_trainRos = ros.fit_sample(X_trainFlat, Y_train)
X_testRos, Y_testRos = ros.fit_sample(X_testFlat, Y_test)

# Encode labels to hot vectors (ex : 2 -> [0,0,1,0,0,0,0,0,0,0])
Y_trainRosHot = to_categorical(Y_trainRos, num_classes = 8)
Y_testRosHot = to_categorical(Y_testRos, num_classes = 8)
print("X_train: ", X_train.shape) # X_train: (4484, 64, 64, 3)
print("X_trainFlat: ", X_trainFlat.shape) # X_trainFlat: (4484,
   12288)
print("X_trainRos Shape: ",X_trainRos.shape) # X_trainRos Shape:
   (19488, 12288)
print("X_testRos Shape: ",X_testRos.shape)# X_testRos Shape: (4864,
    12288)

#
for i in range(len(X_trainRos)):
    height, width, channels = 64,64,3
    X_trainRosReshaped = X_trainRos.reshape(len(X_trainRos),height,
       width,channels)
print("X_trainRos Shape: ",X_trainRos.shape) #(19488, 12288)
print("X_trainRosReshaped Shape: ",X_trainRosReshaped.shape)
   #(19488, 64, 64, 3)
```

```
for i in range(len(X_testRos)):
    height, width, channels = 64,64,3
    X_testRosReshaped = X_testRos.reshape(len(X_testRos),height,
       width,channels)
print("X_testRos Shape: ",X_testRos.shape) #(4864, 12288)
print("X_testRosReshaped Shape: ",X_testRosReshaped.shape) #(4864,
   64, 64, 3)

#final balance class
dfRos = pd.DataFrame()
dfRos["labels"]=Y_trainRos
labRos = dfRos['labels']
distRos = lab.value_counts()
sns.countplot(labRos)
print(dict_characters)

#Now we have a much more even distriution of sample sizes for each
   of our 13 ailments (plus a 14th category for other/typos).
# This should help make our model less biased in favor of the
   majority class (0=No Finding).
from sklearn.utils import class_weight
class_weight = class_weight.compute_class_weight('balanced', np.
   unique(y), y)
print("Old Class Weights: ",class_weight)
from sklearn.utils import class_weight
class_weight = class_weight.compute_class_weight('balanced', np.
   unique(Y_trainRos), Y_trainRos)
print("New Class Weights: ",class_weight)

#------------------------------------------------------------
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.
   read_csv)
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.gridspec as gridspec
import matplotlib.ticker as ticker
sns.set_style('whitegrid')

#from IPython import get_ipython

df = pd.read_csv('../Project 2/sample/sample_labels.csv')
df.head()
df.describe()

# A. data cleaning
# drop unused columns
```

```
df = df[['Image Index','Finding Labels','Follow-up #','Patient ID
    ','Patient Age','Patient Gender']]

#create new columns for each decease
pathology_list = ['Cardiomegaly','Emphysema','Effusion','Hernia','
    Nodule','Pneumothorax','Atelectasis','Pleural_Thickening','Mass
    ','Edema','Consolidation','Infiltration','Fibrosis','Pneumonia']

for pathology in pathology_list :
    df[pathology] = df['Finding Labels'].apply(lambda x: 1 if
        pathology in x else 0)

# B. Patients age analysis :
df['Age Type']=df['Patient Age'].apply(lambda x: x[-1:])
df['Age Type'].unique() # => Y, M and D
# we mainly have ages expressed in Years, but also a few expressed
    in Months or in Days
print(df[df['Age Type']=='Y']['Patient ID'].count()) # 5604
print(df[df['Age Type']=='M']['Patient ID'].count()) # 1
print(df[df['Age Type']=='D']['Patient ID'].count()) # 1

# we are going to remove remove character after patients age, and
    transform D and M in years
df['Age']=df['Patient Age'].apply(lambda x: x[:-1]).astype(int)
df.loc[df['Age Type']=='M',['Age']] = df[df['Age Type']=='M']['Age
    '].apply(lambda x: round(x/12.)).astype(int)
df.loc[df['Age Type']=='D',['Age']] = df[df['Age Type']=='D']['Age
    '].apply(lambda x: round(x/365.)).astype(int)
print(df[df['Age Type']=='D']['Age']) # 4749 0
df[df['Age Type']=='M']['Age']

# We have some strange values for patient ages ( 16 are upper to
    145 years old !) :
df['Age'].sort_values(ascending=False).head(20)

# We also have strange values for 'Follow-up #' comparing to '
    Patient Age'. So we cant consider that 'Follow-up #' are on
    chronological orderer :
df.loc[df['Patient ID']==5567,['Patient Age','Finding Labels','
    Follow-up #']].sort_values('Follow-up #',ascending=True)
df.loc[df['Patient ID']==5567,['Patient Age','Finding Labels','
    Follow-up #']].sort_values('Patient Age',ascending=False)



# C. Display number of each diseases by patient gender
plt.figure(figsize=(15,10))
gs = gridspec.GridSpec(8,1)
ax1 = plt.subplot(gs[:7, :])
```

```
ax2 = plt.subplot(gs[7, :])
data1 = pd.melt(df,
            id_vars=['Patient Gender'],
            value_vars = list(pathology_list),
            var_name = 'Category',
            value_name = 'Count')
data1 = data1.loc[data1.Count>0]
g=sns.countplot(y='Category',hue='Patient Gender',data=data1, ax=
   ax1, order = data1['Category'].value_counts().index)
ax1.set( ylabel="",xlabel="")
ax1.legend(fontsize=20)
ax1.set_title('X Ray partition',fontsize=18);

df['Nothing']=df['Finding Labels'].apply(lambda x: 1 if 'No Finding
   ' in x else 0)

data2 = pd.melt(df,
            id_vars=['Patient Gender'],
            value_vars = list(['Nothing']),
            var_name = 'Category',
            value_name = 'Count')
data2 = data2.loc[data2.Count>0]
g=sns.countplot(y='Category',hue='Patient Gender',data=data2,ax=ax2
   )
ax2.set( ylabel="",xlabel="Number of disease")
ax2.legend('')
plt.subplots_adjust(hspace=.5)

# age distribution by gender
g = sns.factorplot(x="Age", col="Patient Gender",data=df, kind="
   count",size=10, aspect=0.8,palette="GnBu_d");
g.set_xticklabels(np.arange(0,100));
g.set_xticklabels(step=10);
g.fig.suptitle('Age distribution by gender',fontsize=22);
g.fig.subplots_adjust(top=.9)

# E. Display diseases distribution by age and gender.
f, axarr = plt.subplots(7, 2, sharex=True,figsize=(15, 20))
i=0
j=0
x=np.arange(0,100,10)
for pathology in pathology_list :
    g=sns.countplot(x='Age', hue="Patient Gender",data=df[df['
       Finding Labels']==pathology], ax=axarr[i, j])
    axarr[i, j].set_title(pathology)
    g.set_xlim(0,90)
    g.set_xticks(x)
    g.set_xticklabels(x)
```

```
    j=(j+1)%2
    if j==0:
        i=(i+1)%7
f.subplots_adjust(hspace=0.3)

# F. Display patient number by Follow-up in details:
f, (ax1,ax2) = plt.subplots( 2, figsize=(15, 10))

data = df[df['Follow-up #']<15]
g = sns.countplot(x='Follow-up #',data=data,palette="GnBu_d",ax=ax1
    );

ax1.set_title('Follow-up distribution');
data = df[df['Follow-up #']>14]
g = sns.countplot(x='Follow-up #',data=data,palette="GnBu_d",ax=ax2
    );
x=np.arange(15,100,10)
g.set_ylim(15,450)
g.set_xlim(15,100)
g.set_xticks(x)
g.set_xticklabels(x)
f.subplots_adjust(top=1)

# G. Try to find links between pathologies.
# First display Top 10 multiple diseases
data=df.groupby('Finding Labels').count().sort_values('Patient ID',
    ascending=False).head(23)
data=data[['|' in index for index in data.index.values]]
data

# Now we need to compare ratio between simple and multiple diseases
data=df.groupby('Finding Labels').count().sort_values('Patient ID',
    ascending=False)
df1=data[['|' in index for index in data.index]].copy()
df2=data[['|' not in index for index in data.index]]
df2=df2[['No Finding' not in index for index in df2.index]]
df2['Finding Labels']=df2.index.values
df1['Finding Labels']=df1.index.values

f, ax = plt.subplots(sharex=True,figsize=(15, 10))
sns.set_color_codes("pastel")
g=sns.countplot(y='Category',data=data1, ax=ax, order = data1['
    Category'].value_counts().index,color='b',label="Multiple
    diseases")
sns.set_color_codes("muted")
g=sns.barplot(x='Patient ID',y='Finding Labels',data=df2, ax=ax,
    color="b",label="Single disease")
ax.legend(ncol=2, loc="center right", frameon=True,fontsize=20)
```

```
ax.set( ylabel="",xlabel="Number of diseases")
ax.set_title("Comparison between single or multiple diseases",
   fontsize=20)
sns.despine(left=True)

# Plot most important pathologies groups for each disease
# we just keep groups of pathologies which appear more than 30
   times
df3=df1.loc[df1['Patient ID']>30,['Patient ID','Finding Labels']]

for pathology in pathology_list:
    df3[pathology]=df3.apply(lambda x: x['Patient ID'] if pathology
        in x['Finding Labels'] else 0, axis=1)

df3.head(20)

# 'Hernia' has not enough values to figure here
df4=df3[df3['Hernia']>0] # df4.size == 0
# remove 'Hernia' from list
pat_list=[elem for elem in pathology_list if 'Hernia' not in elem]

f, axarr = plt.subplots(13, sharex=True,figsize=(10, 140))
i=0
for pathology in pat_list :
    df4=df3[df3[pathology]>0]
    if df4.size>0: # 'Hernia' has not enough values to figure here
        axarr[i].pie(df4[pathology],labels=df4['Finding Labels'],
           autopct='%1.1f%%')
        axarr[i].set_title('main desease : '+pathology,fontsize=14)
        i +=1
```