Pneumonia

Graham Chickering

11/13/2020

```
#install_tensorflow(version = "2.0.0")
library(tensorflow)
library(reticulate)
library(tfdatasets)
library(keras)
library(tidyverse)
library(ggplot2)
library(tfruns)
```

Setting size image size and channels

```
chest_list <- c("NORMAL","PNEUMONIA")
output_n<-length(chest_list)

img_width <- 64
img_height <- 64
target_size <- c(img_width, img_height)

#this is for grayscale images
channels <- 1

batch_size<-32

# path to image folders
##comeback and change this
train_image_files_path <- file.path("medical_images/chest_xray/train")
```

Here I set the target size of the image that I want to work with (64x64), as well as the number of channels for the image I will be working with which is 1 once I'm working with grayscale images. We then set the file path to where we can locate our training images folder.

```
train_data_gen = image_data_generator(
  rescale = 1/255,
  validation_split=0.2
)
```

This is going to be used by our next function in order to load in the data from the files without actually storing the files in R's memory. This also works to rescale the images down to a much smaller size and then also splits up the training data into both a training and validation set.

```
train_image_array_gen <- flow_images_from_directory(train_image_files_path,
                                          train_data_gen,
                                          subset = 'training',
                                          target_size = target_size,
                                          color_mode="grayscale",
                                          class_mode = "binary",
                                          classes = chest_list,
                                          shuffle=TRUE,
                                          batch_size=batch_size,
                                          seed = 27)

val_image_array_gen <- flow_images_from_directory(train_image_files_path,
                                          train_data_gen,
                                          subset = 'validation',
                                          color_mode="grayscale",
                                          target_size = target_size,
                                          class_mode = "binary",
                                          classes = chest_list,
                                          shuffle=TRUE,
                                          batch_size=batch_size,
                                          seed = 27)
```

This is where we actually convert the images in the training set and the images in the validation set into a form that can be used to perform analysis on them. This function converts the images into tensors that are a representation of the pixels and their intensity. By converting the images into tensors this will allow us to go on and create the models I want to create. We can also see that there are 4173 images in the training set and 1043 images available in the testing set.

```
cat("\nClass label vs index mapping:\n")
```

```
##
## Class label vs index mapping:
```

```
train_image_array_gen$class_indices
```

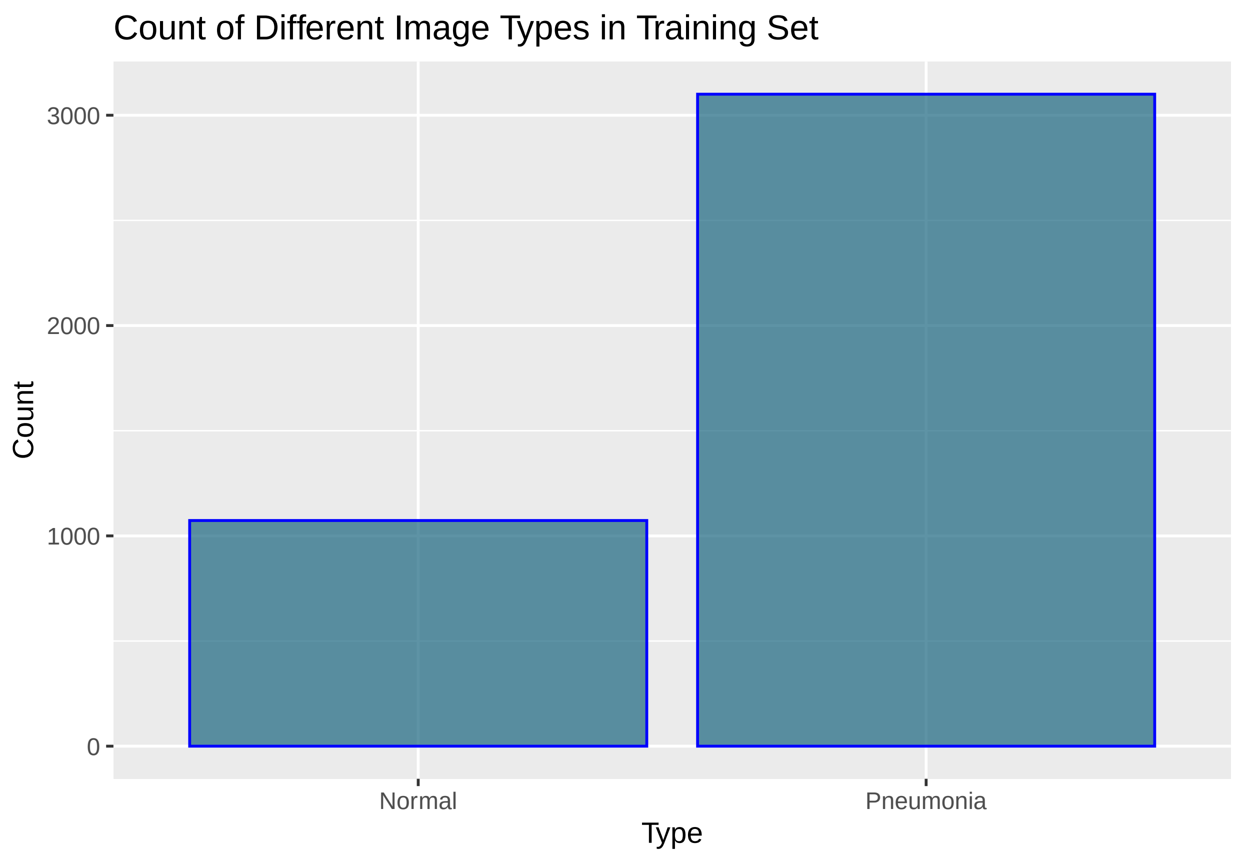
```
## $NORMAL
## [1] 0
##
## $PNEUMONIA
## [1] 1
```

```
table(factor(train_image_array_gen$classes))
```

```
##
##    0    1
## 1073 3100
```

Here we can see that the training set includes 3100 images that are labeled pneumonia and 1073 images that are labeled normal. As we can see this is not an even split. Therefore when we train future models we want them to achieve better than 74% accuracy for them to be an improvement over the known baseline.

```r
tb<-tribble(
  ~Type, ~Count,
  "Normal",1073,
  "Pneumonia",3100
)

count<-ggplot(data=tb, aes(x=Type, y=Count)) +
  geom_bar(stat="identity", color="blue", fill=rgb(0.1,0.4,0.5,0.7) ) + ggtitle("Count of Different Imag
count
```



```r
chest_classes_indices <- train_image_array_gen$class_indices
save(chest_classes_indices, file ="chest_indices.Rdata")
```

```r
# number of training samples
train_samples <- train_image_array_gen$n
# number of validation samples
valid_samples <- val_image_array_gen$n

# define batch size and number of epochs
batch_size <- 32
epochs <- 10
```

This is where we set the number of training and validation samples that are available for training, as well as the number of epochs and the batch size, which is how many images will be used to train the model during each epoch.

```r
model1<- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3,3), activation = "relu",
                input_shape = c(64,64,1)) %>%
  layer_max_pooling_2d(pool_size = c(3,3)) %>%
  layer_flatten() %>%
  layer_dropout(rate=0.5) %>%
  layer_dense(1, activation="softmax")
```

```r
model1 %>% compile(
  optimizer = "adam",
  loss = "binary_crossentropy",
  metrics = "accuracy"
)
summary(model1)
```

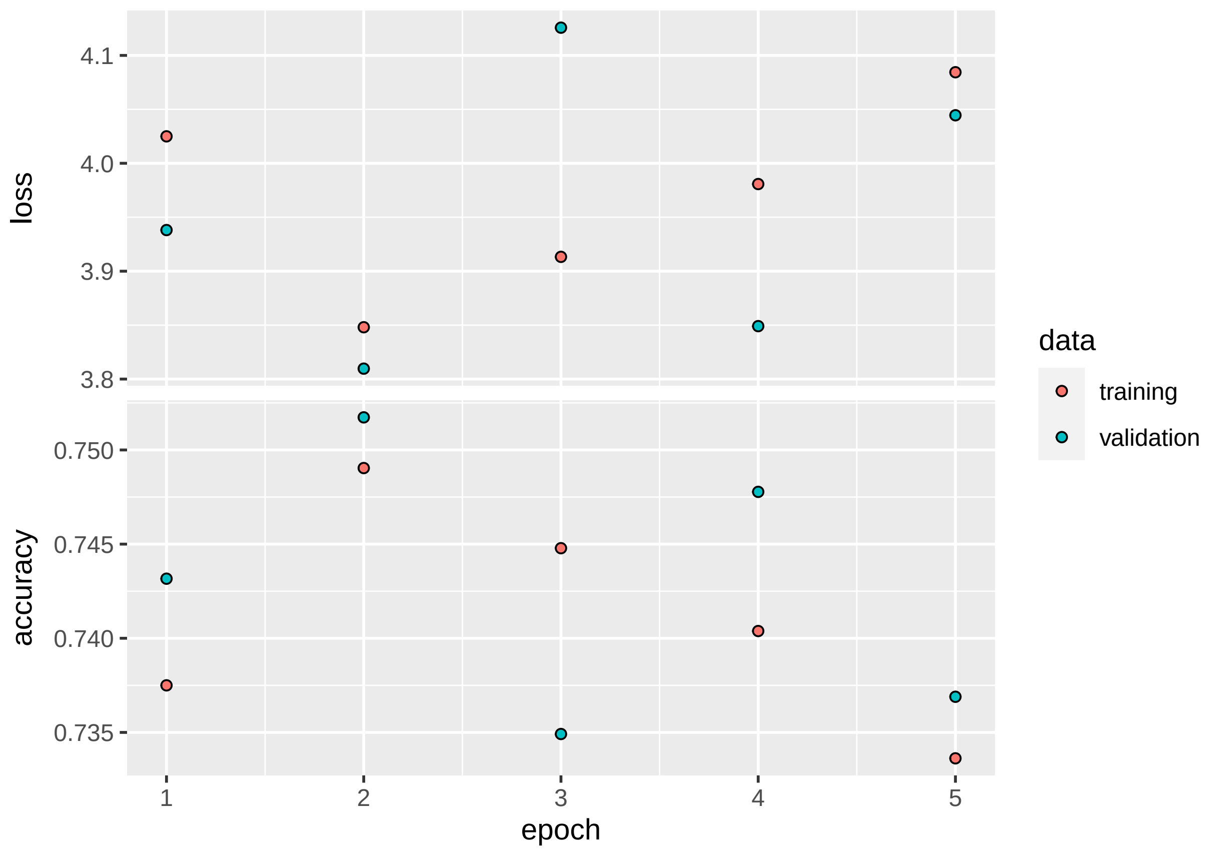
```
## Model: "sequential"
## ________________________________________________________________________________
## Layer (type)                        Output Shape                    Param #
## ================================================================================
## conv2d (Conv2D)                     (None, 62, 62, 32)              320
## ________________________________________________________________________________
## max_pooling2d (MaxPooling2D)        (None, 20, 20, 32)              0
## ________________________________________________________________________________
## flatten (Flatten)                   (None, 12800)                   0
## ________________________________________________________________________________
## dropout (Dropout)                   (None, 12800)                   0
## ________________________________________________________________________________
## dense (Dense)                       (None, 1)                       12801
## ================================================================================
## Total params: 13,121
## Trainable params: 13,121
## Non-trainable params: 0
## ________________________________________________________________________________
```

For my first model I decided to build a model that had one convolution layer, one pooling layer, one flattening layer, a dropout layer that removed half of the available units available in the previous layer during each new training iteration, and then finally one dense layer. The flattening layer works to replace all dimensions of the previous tensors down to one dimension, which is the dimension size we want our output layer to be. The dropout layer is used to make the model more generalizable and requires the model to fit the units with only half the units of the previous layer available during any iteration. The final dense layer is used to create a layer of units, in this case one unit, where every unit in this new layer is connected to every unit in the previous layer, making it densely connected. From looking at the output one can see that there are 13,121 trainable parameters. These are all the weights between the different layers that the model will try to optimize during each training run.

After creating the model itself we have to select the parameters that will be used to actually used to optimize and compare the models. For my optimizer, I used the Adam optimizer which is a robust, gradient based optimization method that has been shown to work extremely well for machine learning problems. For my loss metric, I will be using binary cross entropy to compute the loss. Binary cross entropy is a loss function that measures the performance of a binary classification model where the probability output is between 0 and 1. It calculates the loss by comparing the predicted class value to the true class value and tries to minimize this difference. Therefore the model will use this to minimize the loss and get the loss value as close to zero as it can. For comparing the models themselves I will be comparing the model's accuracy scores which just tells us what percentage of pictures the model will correctly classify. We will use these metrics to compare all the models I create.

```r
set.seed(27)
batch_size<-32
#tensorboard("logs/run_a")
hist <- model1 %>% fit_generator(
  # training data
  train_image_array_gen,

  # epochs
  steps_per_epoch = as.integer(train_samples / batch_size)/2,
  epochs = 5,

  # validation data
  validation_data = val_image_array_gen,
  validation_steps = as.integer(valid_samples / batch_size),

  #callbacks = callback_tensorboard("logs/run_a"),
)
```

```r
plot(hist)
```



```r
model1 %>% save_model_hdf5("my_model1.h5")
```

After training this model we can see that although it starts at around being 75% accurate on both the training and validation sets, that even after 5 epochs that the model does not do any better at being able to discern between whether or not an image is normal or pneumonia than just random guessing. This suggests that either the model does not have enough layers to it and is not able to extract distinguishable features from the images or that there were not enough epochs to train the model. It is also worth noting that this model took 2 and a half minutes to complete its training, with roughly 30 seconds per epoch. This long run time is due to the large number of trainable parameters in the model.

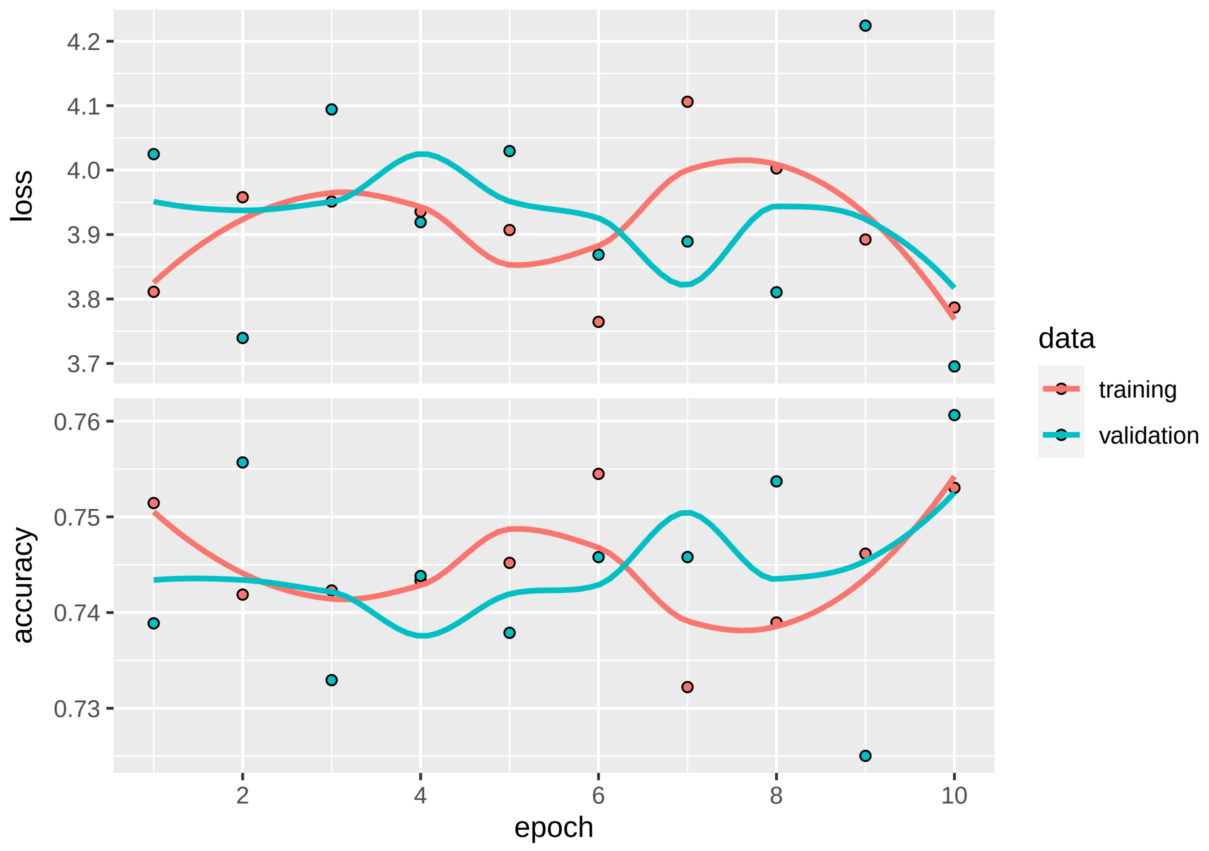
```
model2<- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(5,5), activation = "relu",
                input_shape = c(64,64,1)) %>%
  layer_max_pooling_2d(pool_size = c(3,3)) %>%
  layer_conv_2d(filters = 32, kernel_size = c(5,5), activation = "relu",
                input_shape = c(64,64,1)) %>%
  layer_flatten() %>%
  layer_dropout(rate=0.5) %>%
  layer_dense(1, activation="relu") %>%
  layer_dense(1, activation="softmax")

model2 %>% compile(
  optimizer = "adam",
  loss = "binary_crossentropy",
  metrics = "accuracy"
)
summary(model2)
```

```
## Model: "sequential_1"
## ________________________________________________________________________________
## Layer (type)                        Output Shape                    Param #
## ================================================================================
## conv2d_2 (Conv2D)                   (None, 60, 60, 32)              832
## ________________________________________________________________________________
## max_pooling2d_1 (MaxPooling2D)      (None, 20, 20, 32)              0
## ________________________________________________________________________________
## conv2d_1 (Conv2D)                   (None, 16, 16, 32)              25632
## ________________________________________________________________________________
## flatten_1 (Flatten)                 (None, 8192)                    0
## ________________________________________________________________________________
## dropout_1 (Dropout)                 (None, 8192)                    0
## ________________________________________________________________________________
## dense_2 (Dense)                     (None, 1)                       8193
## ________________________________________________________________________________
## dense_1 (Dense)                     (None, 1)                       2
## ================================================================================
## Total params: 34,659
## Trainable params: 34,659
## Non-trainable params: 0
## ________________________________________________________________________________
```

```
set.seed(27)
#tensorboard("logs/run_b")
hist2 <- model2 %>% fit_generator(
  # training data
  train_image_array_gen,

  # epochs
  steps_per_epoch = as.integer(train_samples / batch_size)/2,
  epochs = epochs,

  # validation data
  validation_data = val_image_array_gen,
  validation_steps = as.integer(valid_samples / batch_size),

  #callbacks = callback_tensorboard("logs/run_b")
)
```

```
plot(hist2)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



```
model2 %>% save_model_hdf5("my_model2.h5")
```

```
model3 <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3,3), activation = "relu",
                input_shape = c(64,64,1)) %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
```

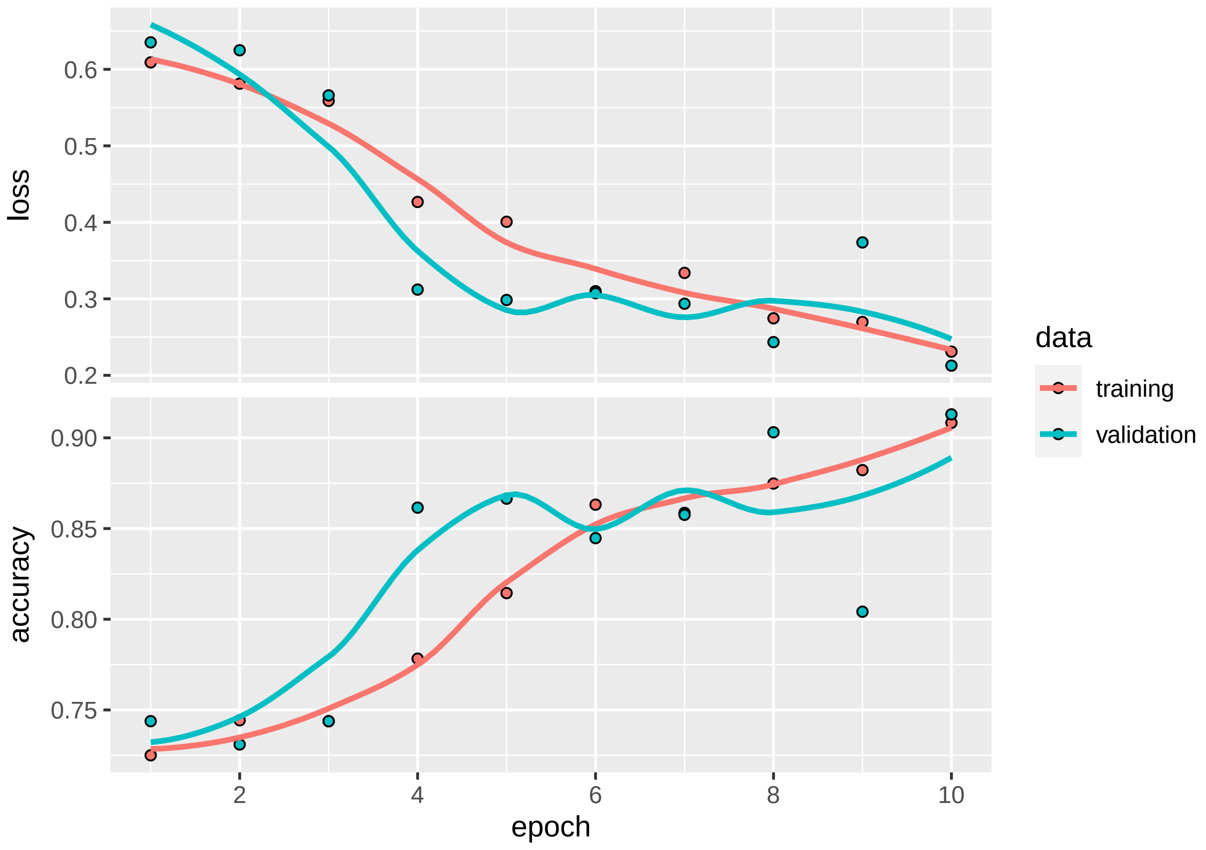
```r
  layer_conv_2d(filters = 32, kernel_size = c(3,3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_dropout(rate=0.5) %>%

  layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_dropout(rate=0.5) %>%

  layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_dropout(rate=0.5) %>%

  layer_flatten() %>%
  layer_dropout(rate=0.2) %>%
  layer_dense(128, activation="relu") %>%
  layer_dense(units = 1, activation = "sigmoid")

model3 %>% compile(
  optimizer = "adam",
  loss = "binary_crossentropy",
  metrics = "accuracy"
)

summary(model3)
```

```
## Model: "sequential_2"
## ________________________________________________________________________________
## Layer (type)                       Output Shape                      Param #
## ================================================================================
## conv2d_6 (Conv2D)                  (None, 62, 62, 32)                320
## ________________________________________________________________________________
## max_pooling2d_5 (MaxPooling2D)     (None, 31, 31, 32)                0
## ________________________________________________________________________________
## conv2d_5 (Conv2D)                  (None, 29, 29, 32)                9248
## ________________________________________________________________________________
## max_pooling2d_4 (MaxPooling2D)     (None, 14, 14, 32)                0
## ________________________________________________________________________________
## dropout_5 (Dropout)                (None, 14, 14, 32)                0
## ________________________________________________________________________________
## conv2d_4 (Conv2D)                  (None, 12, 12, 64)                18496
## ________________________________________________________________________________
## max_pooling2d_3 (MaxPooling2D)     (None, 6, 6, 64)                  0
## ________________________________________________________________________________
## dropout_4 (Dropout)                (None, 6, 6, 64)                  0
## ________________________________________________________________________________
## conv2d_3 (Conv2D)                  (None, 4, 4, 64)                  36928
## ________________________________________________________________________________
## max_pooling2d_2 (MaxPooling2D)     (None, 2, 2, 64)                  0
## ________________________________________________________________________________
## dropout_3 (Dropout)                (None, 2, 2, 64)                  0
## ________________________________________________________________________________
## flatten_2 (Flatten)                (None, 256)                       0
## ________________________________________________________________________________
## dropout_2 (Dropout)                 (None, 256)                     0
## ________________________________________________________________________________
## dense_4 (Dense)                     (None, 128)                     32896
## ________________________________________________________________________________
## dense_3 (Dense)                     (None, 1)                       129
## ================================================================================
## Total params: 98,017
## Trainable params: 98,017
## Non-trainable params: 0
## ________________________________________________________________________________
```

```r
set.seed(27)
#tensorboard("logs/run_c")
hist3 <- model3 %>% fit_generator(
  # training data
  train_image_array_gen,

  # epochs
  steps_per_epoch = as.integer(train_samples / batch_size)/2,
  epochs = epochs,

  # validation data
  validation_data = val_image_array_gen,
  validation_steps = as.integer(valid_samples / batch_size),

  #callbacks = callback_tensorboard("logs/run_c")
)
```

```r
plot(hist3)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



```
#model3 %>% save_model_hdf5("my_model3.h5")
model3 <- load_model_hdf5("my_model3.h5")
```

##Testing the model

```
 test_image_files_path<-file.path("medical_images/chest_xray/test")
```

```
test_datagen <- image_data_generator(rescale = 1/255)

test_generator <- flow_images_from_directory(
        test_image_files_path,
        test_datagen,
        color_mode="grayscale",
        target_size = target_size,
        class_mode = "binary",
        classes = chest_list,
        batch_size = 1,
        shuffle = FALSE,
        seed = 42)
```

```
set.seed(2)
test_results<-model3 %>%
  evaluate_generator(test_generator,
                     steps = as.integer(test_generator$n))
```

```r
#test_results<-test_results %>% as_tibble()
test_results
```

```
## $loss
## [1] 0.3810876
##
## $accuracy
## [1] 0.8461539
```

After running model 3 on the testing set, the model was able to achieve an accuracy score of 86% and a loss score of 0.381 (which would ideally be at 0 if there was perfect accuracy). The testing set contained 234 normal images and 390 pneumonia images so if one were to guess pneumonia every time they would be correct 62.5% of the time. While ideally the model would be able to achieve a perfect 100% accuracy and 0 loss, being able to train a model that achieves an accuracy of 87% on a task as challenging as being able to identify whether or not someone has pneumonia is a very promising result.

```r
classes <- test_generator$classes %>%
  factor() %>%
  table() %>%
  as_tibble()
colnames(classes)[1] <- "value"

indices <- test_generator$class_indices %>%
  as.data.frame() %>%
  gather() %>%
  mutate(value = as.character(value)) %>%
  left_join(classes, by = "value")
indices2<- indices %>% rename(count=n)
indices2
```

```
##         key value count
## 1    NORMAL     0   234
## 2 PNEUMONIA     1   390
```

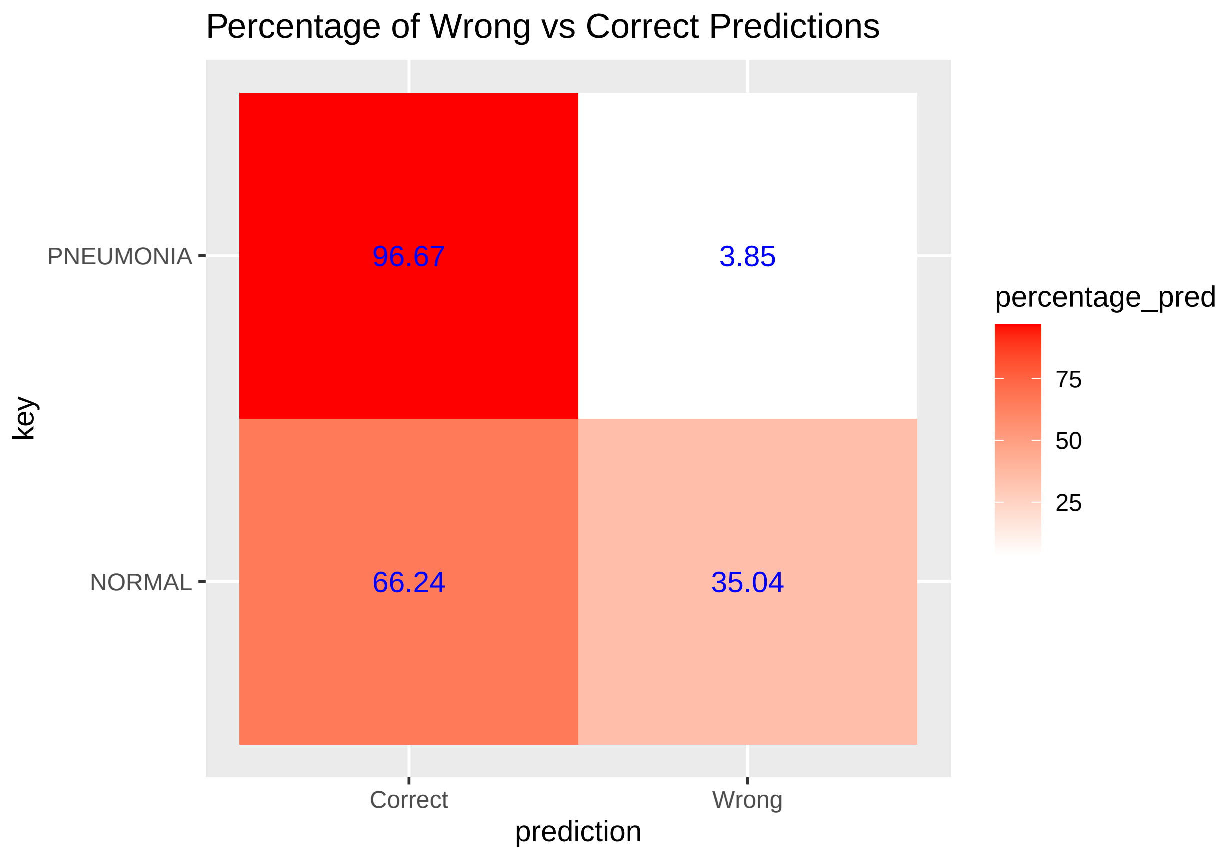
```r
test_generator$reset()
set.seed(1)
predictions <- model3 %>%
  predict_generator(
    generator = test_generator,
    steps = as.integer(test_generator$n)
    ) %>%
  round(digits = 2) %>%
  as_tibble() %>% mutate(V2=1-V1)
```

```
## Warning: The `x` argument of `as_tibble.matrix()` must have unique column names if `.name_repair` is
## Using compatibility `.name_repair`.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```r
trial<-indices$key
colnames(predictions) <- indices$key
```

```r
#predictions
predictions <- predictions %>%
   mutate(truth_idx = as.character(test_generator$classes)) %>%
  left_join(indices, by = c("truth_idx" = "value"))

pred_analysis <- predictions %>%
  mutate(img_id = seq(1:test_generator$n))  %>%
 gather(pred_lbl, y, NORMAL:PNEUMONIA) %>%
 group_by(img_id) %>%
 filter(y == max(y)) %>%
 arrange(img_id) %>%
 group_by(key, n, pred_lbl) %>%
 count()
```

```r
matrix<- pred_analysis %>%
  mutate(prediction = case_when(
    key == pred_lbl ~ "Wrong",
    TRUE ~ "Correct"
  )) %>%
  group_by(key, prediction, n) %>%
  summarise(sum = sum(nn)) %>%
  mutate(percentage_pred = sum / n * 100) %>%
  ggplot(aes(x = key, y = prediction,
             fill = percentage_pred,
             label = round(percentage_pred, 2))) +
    geom_tile() +
    scale_fill_continuous() +
    geom_text(color = "blue") +
    coord_flip() +
    scale_fill_gradient(low = "white", high = "red") + labs(title = "Percentage of Wrong vs Correct Pre
matrix
```



When we break down the accuracy score further and look into how the model did at identifying the two different classes one can begin to look into areas where the model does very well and where the model struggles. By looking at the confusion matrix in Figure 11, we can see that the model does an extremely good job of identifying when the image is of someone who has pneumonia, being able to predict is correctly 96.67% of the time. We can also see that the model is only able to correctly predict whether someone is normal at a 66.4% of the time. What this output tells us is that the model is more likely to identify a patient as having pneumonia when they have do not have it, rather than the opposite scenario of telling a patient they do not have pneumonia when they in fact have it which is probably the more dangerous scenario. When thinking about why the model does such a better job at identifying patients who have pneumonia, it comes back to the type of data we used to train the model. Since the training and validation data contained 3x as many pictures of pneumonia, it makes sense that the model does a much better job of being able to recognize when someone has pneumonia. So while being able to achieve 85% accuracy with a convolutional neural network is very promising, these results show that there is future work that can be done that would be able to improve the model's accuracy even more.