As we found for natural language processing, `R` is a little bit limited when it comes to deep neural networks. Unfortunately, these are currently the basis of gold-standard methods for medical image analysis. The ML/DL `R` packages that do exist provide interfaces (APIs) to libraries built in more faster languages like C/C++ and Java. This means there are a few "non-reproducible" installation and data gathering steps to run first. It also means there are a few "compromises" to get a practical that will A) be runnable B) actually complete within the allotted time and C) even vaguely works.
Doing this on a full real dataset would likely follow a similar role (admittedly with more intensive training and more expressive architectures)

Installing Keras/Tensorflow

We want to install deep learning libraries for R.

Firstly, if you are using windows please install `anaconda` on your systems using the installer and instructions here

Then, we are going to install `tensorflow` in a `reticulate` environment:

- `install.packages("tensorflow")`
- `tensorflow::install_tensorflow()`

Then check the installation was successful by seeing if the follow outputs `tf.Tensor(b'Hellow world', shape=(), dtype=string)` (don't worry if there are some GPU related errors).

```
library("tensorflow")
tensorflow::tf$constant("Hello world")
```

```
## Loaded Tensorflow version 2.9.2
```

```
## tf.Tensor(b'Hello world', shape=(), dtype=string)
```

Then we are going to install the `keras` library:

- `install.packages('keras')`
- `library('keras')`

```
# some timing information for knitr
knitr::knit_hooks$set(time_it = local({
  now <- NULL
  function(before, options) {
    if (before) {
      # record the current time before each chunk
      now <<- Sys.time()
    } else {
      # calculate the time difference after a chunk
      res <- difftime(Sys.time(), now)
      # return a character string to show the time
      paste("Time for this code chunk to run:", res)
    }
  }
}))
knitr::opts_chunk$set(time_it = TRUE, echo=TRUE)


library("keras")
library("tensorflow")
# library("imager")
library("pROC")
```

```
## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
library("caret")
```

```
## Loading required package: ggplot2

## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:tensorflow':
##
##     train
```

Diagnosing Pneumonia from Chest X-Rays

Parsing the data

Today, we are going to look at a series of chest X-rays from children (from this paper) and see if we can accurately diagnose pneumonia from them.

Chest X-ray images (anterior-posterior) were selected from retrospective cohorts of pediatric patients of one to five years old from Guangzhou Women and Children's Medical Center, Guangzhou. All chest X-ray imaging was performed as part of patients' routine clinical care. For the analysis of chest x-ray images, all chest radiographs were initially screened for quality control by removing all low quality or unreadable scans. The diagnoses for the images were then graded by two expert physicians before being cleared for inclusion. In order to account for any grading errors, the evaluation set was also checked by a third expert.

We can download, unzip, then inspect the format of this dataset as follows:

- `https://drive.google.com/file/d/14H_FilWf12ONOJ_G4vvzNDDGvY7Ccqtm/view?usp=sharing`
- `unzip('lab3_chest_xray.zip')`

```
data_folder <- "lab3_chest_xray"

files <- list.files(data_folder, full.names=TRUE, recursive=TRUE)
sort(sample(files, 20))
```

```
##  [1] "lab3_chest_xray/test/NORMAL/IM-0081-0001.jpeg"
##  [2] "lab3_chest_xray/test/NORMAL/NORMAL2-IM-0033-0001.jpeg"
##  [3] "lab3_chest_xray/test/NORMAL/NORMAL2-IM-0081-0001.jpeg"
##  [4] "lab3_chest_xray/train/NORMAL/IM-0131-0001.jpeg"
##  [5] "lab3_chest_xray/train/NORMAL/IM-0145-0001.jpeg"
##  [6] "lab3_chest_xray/train/NORMAL/IM-0286-0001.jpeg"
##  [7] "lab3_chest_xray/train/NORMAL/IM-0387-0001.jpeg"
##  [8] "lab3_chest_xray/train/NORMAL/IM-0461-0001.jpeg"
##  [9] "lab3_chest_xray/train/NORMAL/IM-0549-0001-0001.jpeg"
## [10] "lab3_chest_xray/train/NORMAL/IM-0553-0001-0001.jpeg"
## [11] "lab3_chest_xray/train/NORMAL/IM-0602-0001.jpeg"
## [12] "lab3_chest_xray/train/NORMAL/IM-0618-0001-0001.jpeg"
## [13] "lab3_chest_xray/train/PNEUMONIA/person1037_bacteria_2971.jpeg"
## [14] "lab3_chest_xray/train/PNEUMONIA/person1067_virus_1770.jpeg"
```

```
## [15] "lab3_chest_xray/train/PNEUMONIA/person1125_bacteria_3066.jpeg"
## [16] "lab3_chest_xray/train/PNEUMONIA/person1158_virus_1942.jpeg"
## [17] "lab3_chest_xray/train/PNEUMONIA/person1216_virus_2062.jpeg"
## [18] "lab3_chest_xray/train/PNEUMONIA/person1228_virus_2079.jpeg"
## [19] "lab3_chest_xray/train/PNEUMONIA/person1248_virus_2117.jpeg"
## [20] "lab3_chest_xray/train/PNEUMONIA/person1251_bacteria_3208.jpeg"
```

Time for this code chunk to run: 0.0188970565795898 As with every data analysis, the first thing we need to do is learn about our data. If we are diagnosing pneumonia, we should use the internet to better understand what pneumonia actually is and what types of pneumonia exist.

0 What is pneumonia and what is the point/benefit of being able to identify it from X-rays automatically?

From Mayoclinic: pneumonia is an infection that inflames the air sacs in one or both lungs. It could be useful to be able to identify it automatically from x-rays to reduce the amount of time a patient goes between scan and diagnosis, and also to free up those doing the interpretations.

In the output above, you can see the filenames for all the chest X-rays. Look carefully at the filenames for the X-rays showing pneumonia (i.e., those in `{train,test}/PNEUMONIA`).

1 Do these filenames tell you anything about pneumonia? Why might this make predicting pneumonia more challenging?

From looking at the filenames, it seems to me like the numbers after "person" and "bacteria/virus" indicate that pneumonia can present itself in many different ways in different people, which could make it harder to predict or detect.

Let's inspect a couple of these files as it is always worth looking directly at data.

```
# plot(imager::load.image(paste(data_folder, "train/NORMAL/IM-0140-0001.jpeg", sep='/')))
```

Time for this code chunk to run: 0.000698089599609375

```
# plot(imager::load.image(paste(data_folder, "train/PNEUMONIA/person1066_virus_1769.jpeg", sep='/')))
```

Time for this code chunk to run: 0.000668048858642578

```
# plot(imager::load.image(paste(data_folder, "train/PNEUMONIA/person1101_bacteria_3042.jpeg", sep='/')))
```

Time for this code chunk to run: 0.00069117546081543

2 From looking at only 3 images do you see any attribute of the images that we may have to normalise before training a model?

The dimensions of the three images are all different. This will have to be normalized before training a model.

Image data is relatively large so generally we don't want to read all of them into memory at once. Instead `keras` (and most DL libraries) use generators that read the images (`kerass::image_data_generator`) in batches for training/validation/testing (`keras::flow_images_from_directory`), perform normalizing (and potentially augmenting) transformations, then pass them to the specified model.

```
seed <- 42
set.seed(seed)

train_dir = paste(data_folder, "train", sep='/')
validation_dir = paste(data_folder, "validate", sep='/')
test_dir = paste(data_folder, "test", sep='/')

batch_size <- 32

train_datagen <- keras::image_data_generator(rescale = 1/255, validation_split=0.1)
```

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

train_generator <- keras::flow_images_from_directory(
  train_dir,                            # Target directory
  train_datagen,                        # Data generator
  classes = c('NORMAL', 'PNEUMONIA'),
  target_size = c(224, 224),            # Resizes all images
  batch_size = batch_size,
  class_mode = "categorical",
  shuffle = T,
  seed = seed,
  subset='training'
)

validation_generator <- keras::flow_images_from_directory(
  train_dir,                            # Target directory
  train_datagen,                        # Data generator
  classes = c('NORMAL', 'PNEUMONIA'),
  target_size = c(224, 224),            # Resizes all images
  batch_size = batch_size,
  class_mode = "categorical",
  seed = seed,
  subset='validation'
)


test_generator <- keras::flow_images_from_directory(
  test_dir,
  classes = c('NORMAL', 'PNEUMONIA'),
  test_datagen,
  target_size = c(224, 224),
  batch_size = 1,
  class_mode = "categorical",
  shuffle = FALSE
)
```

Time for this code chunk to run: 0.0499019622802734

3 How big is our training data set? How could we more efficiently use our data while still getting information about validation performance?

Our training data set contains 1200 images. We could use k-fold cross validation to more efficiently use the data.

Specifying an initial model

We are going to start by using the `MobileNet` architecture from keras' set of available networks.

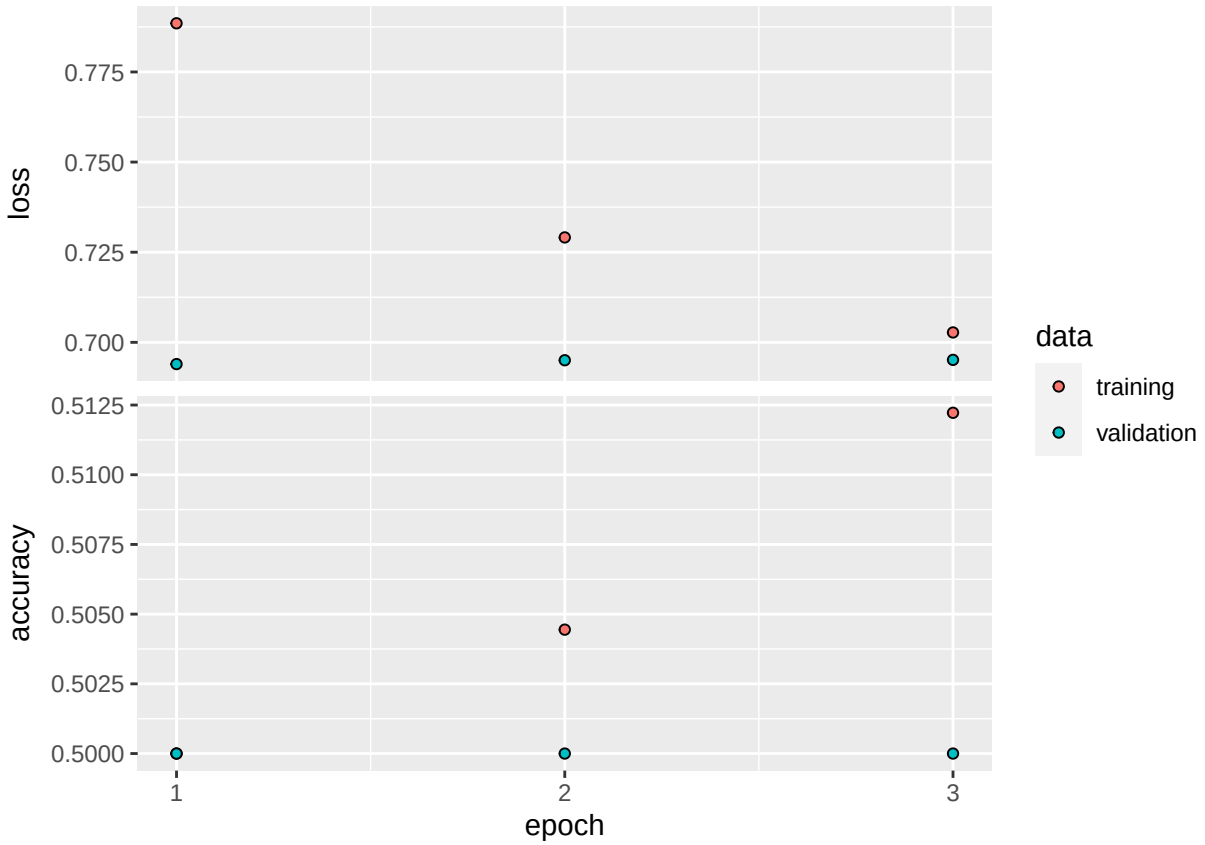
4 Why do you think we are using this architecture in this practical assignment? Hint: MobileNet publication

According to the publication, the MobileNet architecture is useful for creating lightweight computer vision deep neural nets, which would be suitable for a practical setting.

5 How many parameters does this network have? How does this compare to better performing networks available in `keras`?

MobileNet has 4.2 million parameters, while something like ResNet has 60 million.

```r
conv_base <- keras::application_mobilenet(
  weights = NULL,
  include_top = FALSE,
  input_shape = c(224, 224, 3)
)


model_basic <- keras::keras_model_sequential() %>%
  conv_base %>%
  layer_global_average_pooling_2d(trainable = T) %>%
  layer_dropout(rate = 0.2, trainable = T) %>%
  layer_dense(units = 224, activation = "relu", trainable = T) %>%
  layer_dense(units = 2, activation = "softmax", trainable = T)


model_basic %>% keras::compile(
  loss = "categorical_crossentropy",
  optimizer = keras::optimizer_rmsprop(learning_rate = 1e-6),
  metrics = c("accuracy")
)


history_basic <- model_basic %>% keras::fit(
  train_generator,
  steps_per_epoch = ceiling(900 / batch_size),
  epochs = 3,
  validation_data = validation_generator,
  validation_step_size = ceiling(100 / batch_size)
)

plot(history_basic)
```



Time for this code chunk to run: 37.1605429649353

6 Based on the training and validation accuracy is this model actually managing to classify X-rays into pneumonia vs normal? What do you think contributes to this?

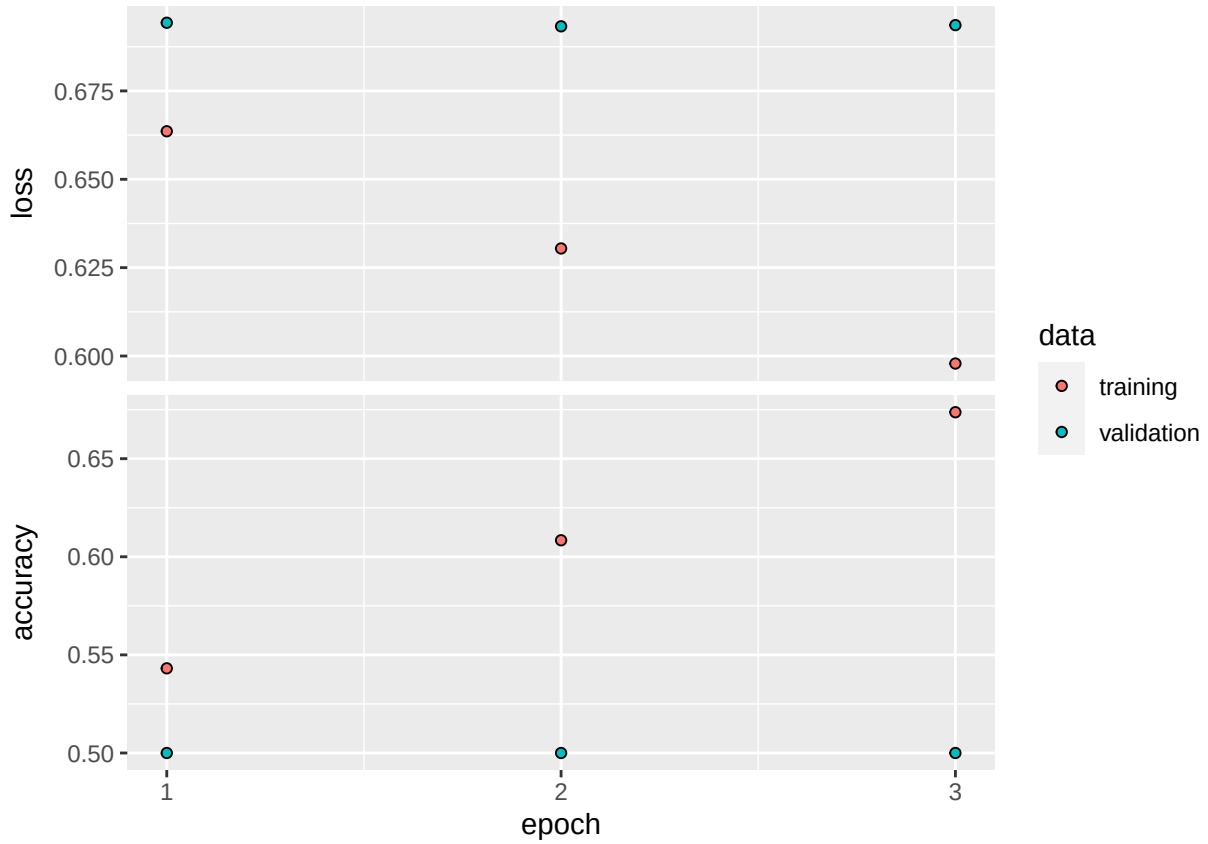
Looking at the validation accuracy being 0.5 for all three epochs, it looks like the model is not managing to classify the x-rays. It's basically a coin flip at this point.

Data augmentation

Let's see if we can expand the training data by adding transformations and noise to the `keras::image_data_generator`. These largely take the form of affine transformations but can also be probabilistic.

```
train_datagen <- keras::image_data_generator(
  rescale = 1/255,
  rotation_range = 5,
  horizontal_flip = TRUE,
  vertical_flip = FALSE,
  fill_mode = "reflect")

train_generator <- keras::flow_images_from_directory(
  train_dir,                          # Target directory
  train_datagen,                      # Data generator
  classes = c('NORMAL', 'PNEUMONIA'),
  target_size = c(224, 224),          # Resizes all images
  batch_size = batch_size,
  class_mode = "categorical",
  shuffle = T,
  seed = seed
```

```
)

history_aug <- model_basic %>% keras::fit(
  train_generator,
  steps_per_epoch = ceiling(900 / batch_size),
  epochs = 3,
  validation_data = validation_generator,
  validation_step_size = ceiling(100 / batch_size)
)
plot(history_aug)
```



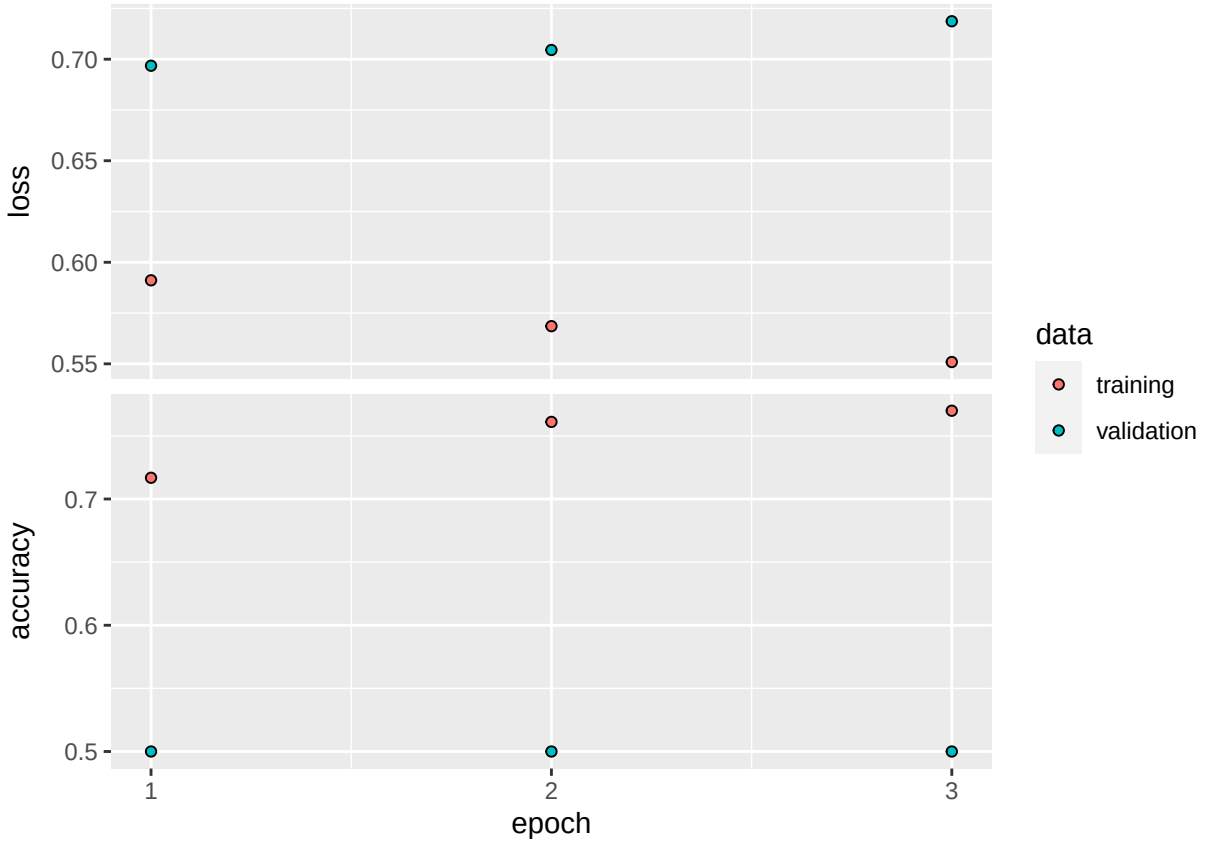
Time for this code chunk to run: 37.5957839488983

7 Does this perform better than the unagumented model? By looking at the documentation for `?keras::image_data_generator` re-run the above with additional augmenting and normalising transformations (note: make sure to keep `rescale=1/255` and `validation_split=0.1` or the images won't feed into the network).

The accuracy of the augmented model is the same as the unaugmented one.

```
train_datagen <- keras::image_data_generator(
  rescale = 1/255,
  rotation_range = 5,
  horizontal_flip = TRUE,
  vertical_flip = FALSE,
  zoom_range = 0.2,
  fill_mode = "reflect")

train_generator <- keras::flow_images_from_directory(
```

```
  train_dir,                              # Target directory
  train_datagen,                          # Data generator
  classes = c('NORMAL', 'PNEUMONIA'),
  target_size = c(224, 224),              # Resizes all images
  batch_size = batch_size,
  class_mode = "categorical",
  shuffle = T,
  seed = seed
)

history_aug <- model_basic %>% keras::fit(
  train_generator,
  steps_per_epoch = ceiling(900 / batch_size),
  epochs = 3,
  validation_data = validation_generator,
  validation_step_size = ceiling(100 / batch_size)
)
plot(history_aug)
```



Time for this code chunk to run: 38.0731410980225

Transfer Learning

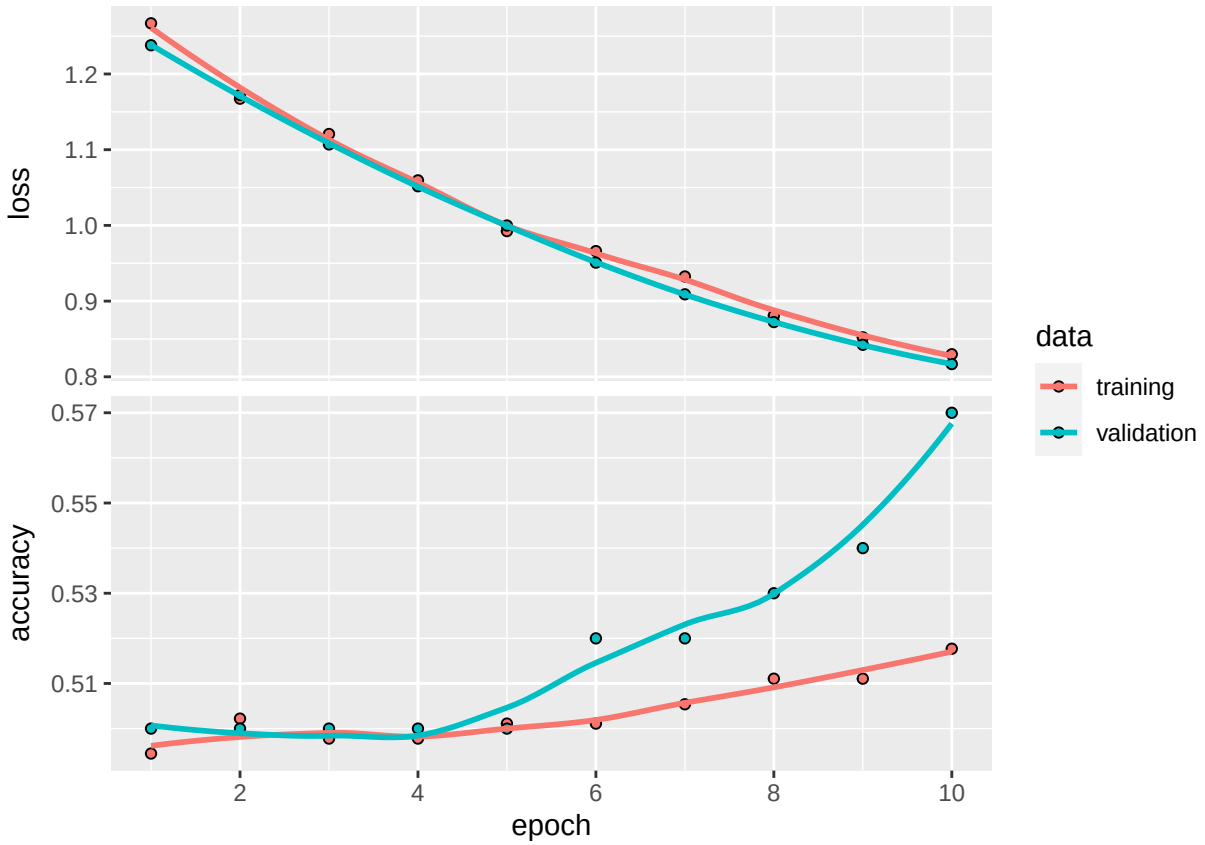
So far we have been initialising the `mobilenet` model with random weights and training all the parameters form our (augmented) dataset. However, keras provides versions of all the integrated architectures that have already been trained on the `ImageNet` dataset.

8 What is ImageNet aka "ImageNet Large Scale Visual Recognition Challenge 2012"? How

many images and classes does it involve? Why might this help us?

ImageNet contains 10 million images from over 10 thousand categories of images. ImageNet may help us with this problem as knowledge gained from ImageNet can be used to identify pneumonia in this dataset.

```
conv_base <- keras::application_mobilenet(
  weights = "imagenet",
  include_top = FALSE,
  input_shape = c(224, 224, 3)
)


model_tf <- keras::keras_model_sequential() %>%
  conv_base %>%
  layer_global_average_pooling_2d(trainable = T) %>%
  layer_dropout(rate = 0.2, trainable = T) %>%
  layer_dense(units = 224, activation = "relu", trainable = T) %>%
  layer_dense(units = 2, activation = "softmax", trainable = T)

keras::freeze_weights(conv_base)

model_tf %>% keras::compile(
  loss = "categorical_crossentropy",
  optimizer = keras::optimizer_rmsprop(learning_rate = 1e-6),
  metrics = c("accuracy")
)


history_pre <- model_tf %>% keras::fit(
  train_generator,
  steps_per_epoch = ceiling(900 / batch_size),
  epochs = 10,
  validation_data = validation_generator,
  validation_step_size = ceiling(100 / batch_size)
)

plot(history_pre)
```



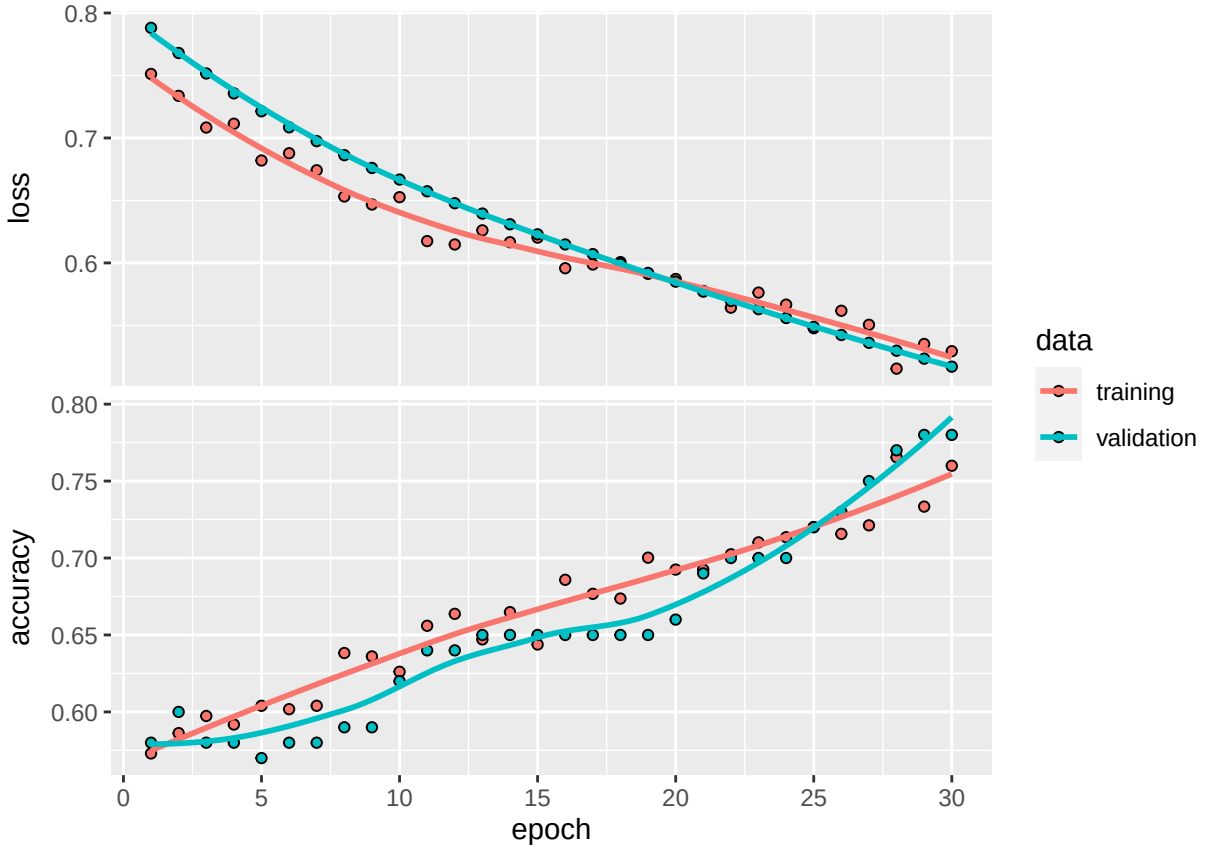
Time for this code chunk to run: 2.00241231520971

10 Using the `tflearning` code chunk above, train a different network architecture? Does this perform better? Recommend: increasing the number of epochs from 3-10 to 30 to `keras::fit` and leaving it for a while!

After running mobilenet_v2 for 30 epochs, we do see significant improvement. Validation accuracy capped out at around 0.74.

```
conv_base <- keras::application_mobilenet_v2(
  weights = "imagenet",
  include_top = FALSE,
  input_shape = c(224, 224, 3)
)


model_tf_v2 <- keras::keras_model_sequential() %>%
  conv_base %>%
  layer_global_average_pooling_2d(trainable = T) %>%
  layer_dropout(rate = 0.2, trainable = T) %>%
  layer_dense(units = 224, activation = "relu", trainable = T) %>%
  layer_dense(units = 2, activation = "softmax", trainable = T)

keras::freeze_weights(conv_base)

model_tf_v2 %>% keras::compile(
  loss = "categorical_crossentropy",
  optimizer = keras::optimizer_rmsprop(learning_rate = 1e-6),
  metrics = c("accuracy")
```

```
)


history_pre <- model_tf_v2 %>% keras::fit(
  train_generator,
  steps_per_epoch = ceiling(900 / batch_size),
  epochs = 30,
  validation_data = validation_generator,
  validation_step_size = ceiling(100 / batch_size)
)

plot(history_pre)
```

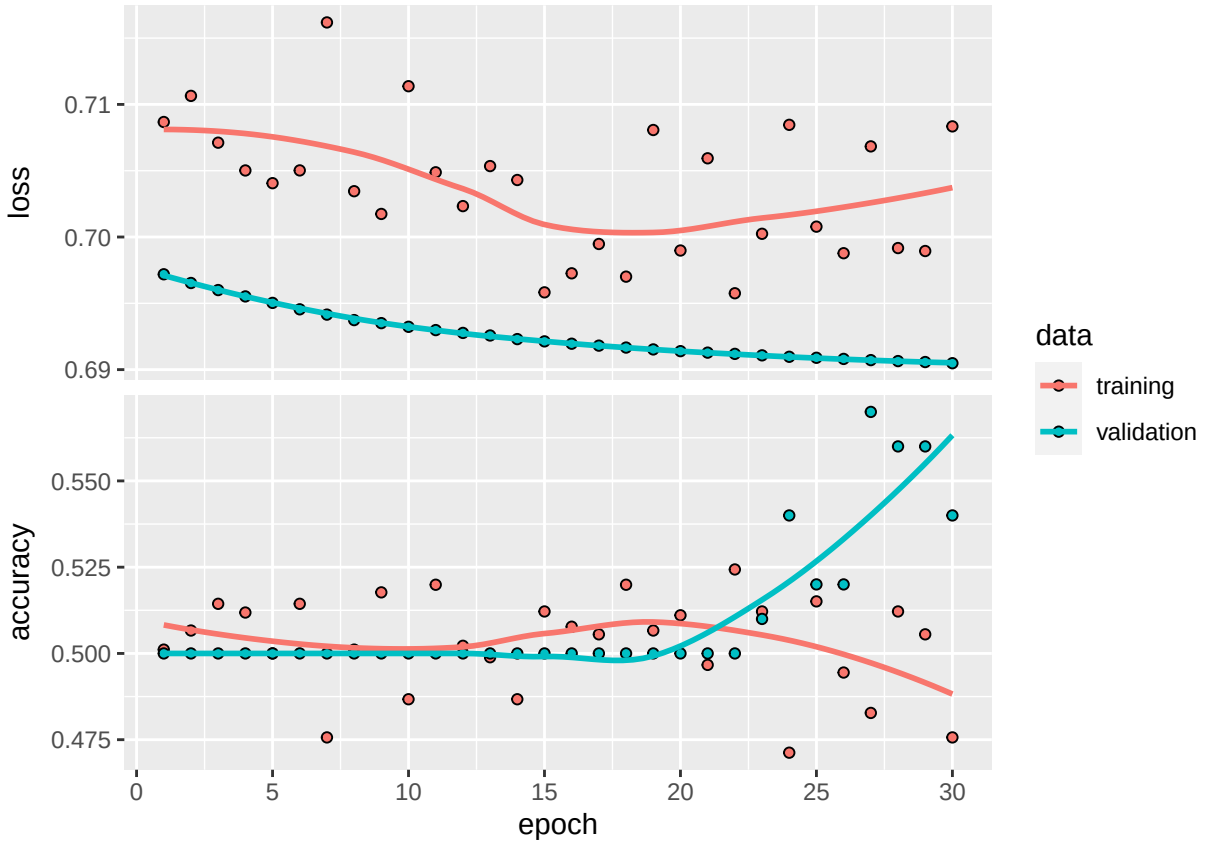


Time for this code chunk to run: 6.05999766588211

Moving to mobilenet_v3_large for 30 epochs, we see max validation much lower, similar to v1 at 0.52:

```
conv_base <- keras::application_mobilenet_v3_large(
  weights = "imagenet",
  include_top = FALSE,
  input_shape = c(224, 224, 3)
)


model_tf_v3 <- keras::keras_model_sequential() %>%
  conv_base %>%
  layer_global_average_pooling_2d(trainable = T) %>%
  layer_dropout(rate = 0.2, trainable = T) %>%
```

```
  layer_dense(units = 224, activation = "relu", trainable = T) %>%
  layer_dense(units = 2, activation = "softmax", trainable = T)

keras::freeze_weights(conv_base)

model_tf_v3 %>% keras::compile(
  loss = "categorical_crossentropy",
  optimizer = keras::optimizer_rmsprop(learning_rate = 1e-6),
  metrics = c("accuracy")
)


history_pre <- model_tf_v3 %>% keras::fit(
  train_generator,
  steps_per_epoch = ceiling(900 / batch_size),
  epochs = 30,
  validation_data = validation_generator,
  validation_step_size = ceiling(100 / batch_size)
)

plot(history_pre)
```



Time for this code chunk to run: 6.0416935523351

Test performance

Once we have identified our best performing network on the validation dataset replace `model_tf` below with the model variable corresponding to that model.

Then we can use the test set to evaluate final performance.

```
preds = keras::predict_generator(model_tf_v2,
                        test_generator,
                        steps = length(list.files(test_dir, recursive = T)))
```

```
## Warning in keras::predict_generator(model_tf_v2, test_generator, steps =
## length(list.files(test_dir, : `predict_generator` is deprecated. Use `predict`
## instead, it now accept generators.
```

```
predictions = data.frame(test_generator$filenames)
predictions$prob_pneumonia = preds[,2]
colnames(predictions) = c('Filename', 'Prob_Pneumonia')

predictions$Class_predicted = 'Normal'
predictions$Class_predicted[predictions$Prob_Pneumonia >= 0.5] = 'Pneumonia'
predictions$Class_actual = 'Normal'
predictions$Class_actual[grep("PNEUMONIA", predictions$Filename)] = 'Pneumonia'

predictions$Class_actual = as.factor(predictions$Class_actual)
predictions$Class_predicted = as.factor(predictions$Class_predicted)

roc = pROC::roc(response = predictions$Class_actual,
          predictor = as.vector(predictions$Prob_Pneumonia),
          ci=T,
          levels = c('Normal', 'Pneumonia'))
```

```
## Setting direction: controls < cases
```

```
threshold = pROC::coords(roc, x = 'best', best.method='youden')
threshold
```

```
##   threshold specificity sensitivity
## 1 0.6886417        0.82        0.45
## 2 0.6923998        0.83        0.44
## 3 0.6983925        0.84        0.43
```

Time for this code chunk to run: 3.71604299545288

11 Overall how useful do you good is your trained model at predicting pneumonia? Do you think this would be useful? How could you better solve this problem?

Overall, I'd say that the model performs quite poorly at predicting pneumonia. While The specificity is reasonably high, the sensitivity is really low, meaning that it misses most true cases.