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 2.9.1
```

```
## 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")
```

```
## Loading required package: magrittr
##
## Attaching package: 'imager'
## The following object is masked from 'package:magrittr':
##
##     add
## The following objects are masked from 'package:stats':
##
##     convolve, spectrum
## The following object is masked from 'package:graphics':
##
##     frame
## The following object is masked from 'package:base':
##
##     save.image
```

```
library("pROC")
```

```
## Type 'citation("pROC")' for a citation.
##
## Attaching package: 'pROC'
## The following object is masked from 'package:imager':
##
##     ci
## 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-0101-0001.jpeg"
##  [2] "lab3_chest_xray/test/NORMAL/NORMAL2-IM-0060-0001.jpeg"
##  [3] "lab3_chest_xray/train/NORMAL/IM-0162-0001.jpeg"
##  [4] "lab3_chest_xray/train/NORMAL/IM-0255-0001.jpeg"
##  [5] "lab3_chest_xray/train/NORMAL/IM-0285-0001.jpeg"
##  [6] "lab3_chest_xray/train/NORMAL/IM-0291-0001.jpeg"
##  [7] "lab3_chest_xray/train/NORMAL/IM-0348-0001.jpeg"
##  [8] "lab3_chest_xray/train/NORMAL/IM-0497-0001-0001.jpeg"
##  [9] "lab3_chest_xray/train/NORMAL/IM-0531-0001-0001.jpeg"
## [10] "lab3_chest_xray/train/NORMAL/IM-0540-0001.jpeg"
## [11] "lab3_chest_xray/train/NORMAL/IM-0631-0001.jpeg"
## [12] "lab3_chest_xray/train/NORMAL/IM-0632-0001.jpeg"
## [13] "lab3_chest_xray/train/NORMAL/IM-0642-0001.jpeg"
## [14] "lab3_chest_xray/train/NORMAL/IM-0647-0001.jpeg"
## [15] "lab3_chest_xray/train/PNEUMONIA/person1040_virus_1735.jpeg"
## [16] "lab3_chest_xray/train/PNEUMONIA/person1085_virus_1797.jpeg"
## [17] "lab3_chest_xray/train/PNEUMONIA/person110_virus_205.jpeg"
## [18] "lab3_chest_xray/train/PNEUMONIA/person1159_virus_1944.jpeg"
## [19] "lab3_chest_xray/train/PNEUMONIA/person1165_virus_1959.jpeg"
## [20] "lab3_chest_xray/train/PNEUMONIA/person1179_virus_2006.jpeg"
```

Time for this code chunk to run: 0.0209953784942627 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?

A: Pneumonia is a condition in which a fluid fills alveoli within the lungs, which causes various issues with lung function. Pneumonia has multiple causes, including infections by bacteria and viruses. One of the primary uses of automatic detection would be to prevent early childhood deaths from pneumonia, which are especially common in developing countries. This is important because in said countries, and particularly in low-income areas there may be radiological imaging data, but not the systems in place for its timely analysis. With an automated system, the X-ray data could be analysed by the automated system, reducing workload for clinicians and allowing for accurate diagnoses to be given, helping prevent pneumonia deaths.

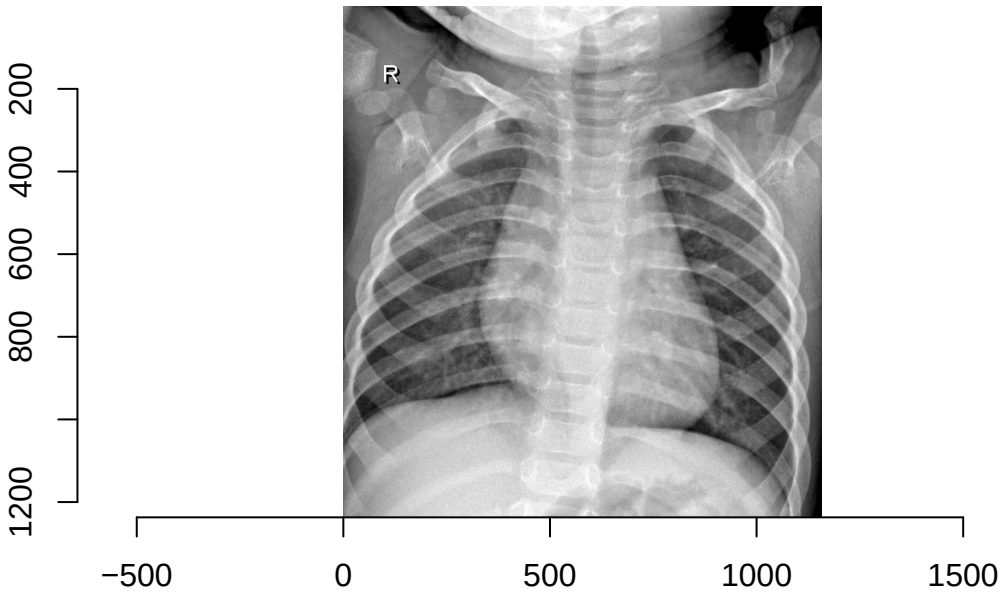
In the output above, you can see the filenames for all the chest X-rays. Look carefully at the filenames for the X-rays showing pneumonia (i.e., those in `{train,test}/PNEUMONIA`).

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

A: The filenames for positive cases indicate both the individual from which the scans were taken (an index number), and the type of pneumonia (virus or bacteria). Finally, the there is a number indexing the sequence of bacteria and virus, showing a count of how many are viruses, and how many are bacteria cases. This could potentially cause issues for training with regard to the ordering of data, keeping related cases close together, but this can be circumvented by shuffling the data when making a test-train-validation split.

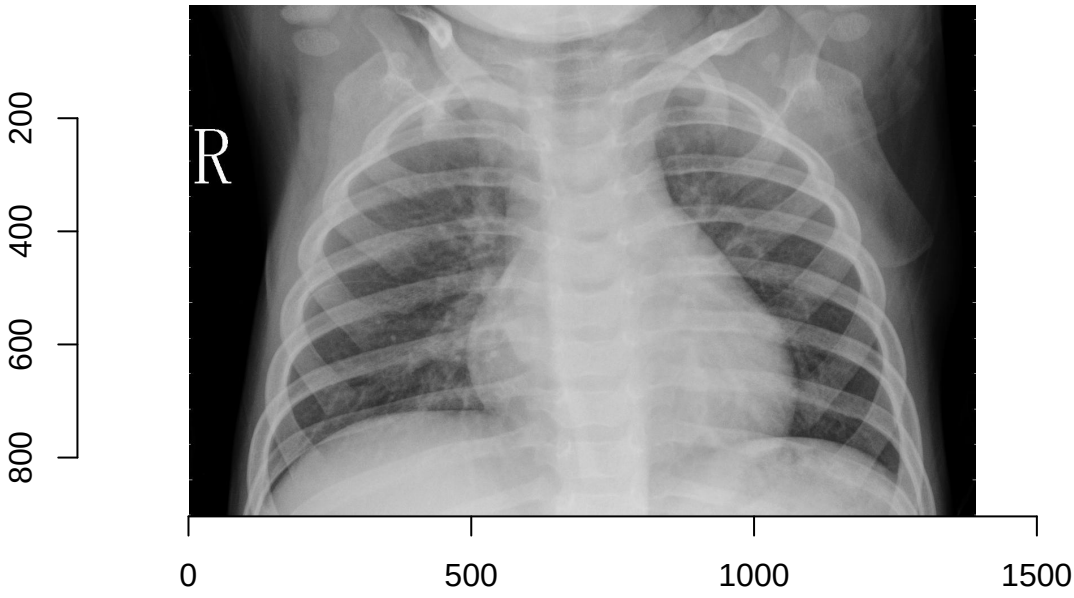
Let's inspect a couple of these files as it is always worth looking directly at data.

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



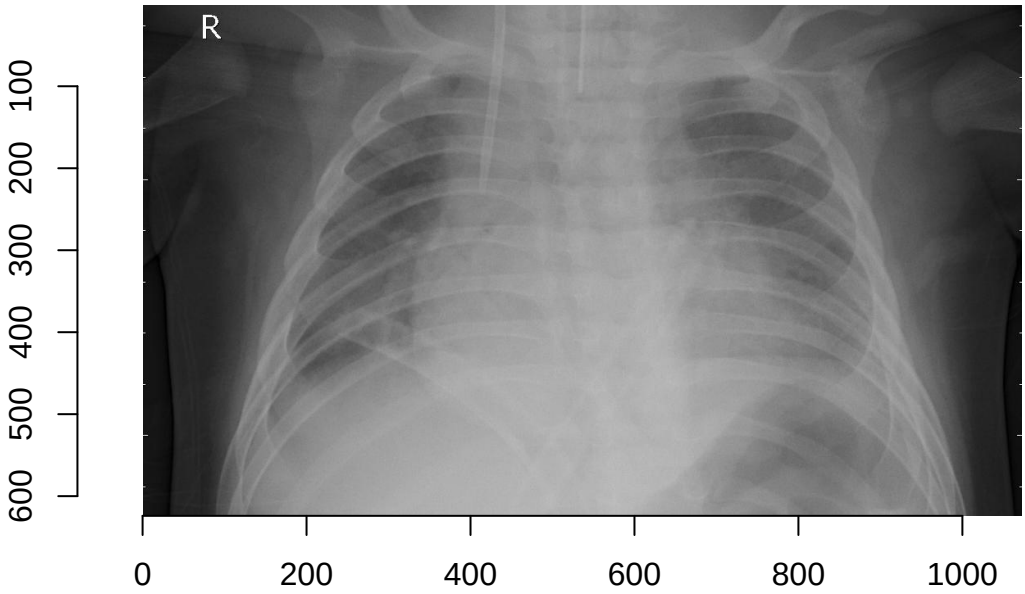
Time for this code chunk to run: 1.33248710632324

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



Time for this code chunk to run: 0.530639886856079

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



Time for this code chunk to run: 0.457083702087402 **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?

A: Image dimensions should probably be normalised for the sake of making the machine learning model more simple, this would probably entail scaling the images to a shared height/width ratio and then scaling them up or down to a specified resolution.

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: 0.0831353664398193 **3** How big is our training data set? How could we more efficiently use our data while still getting information about validation performance?

A: The training dataset is 900 images. We could more efficiently use our data with k-fold cross validation, so that in each batch a different subset of the training dataset is left out and treated as the validation set.

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

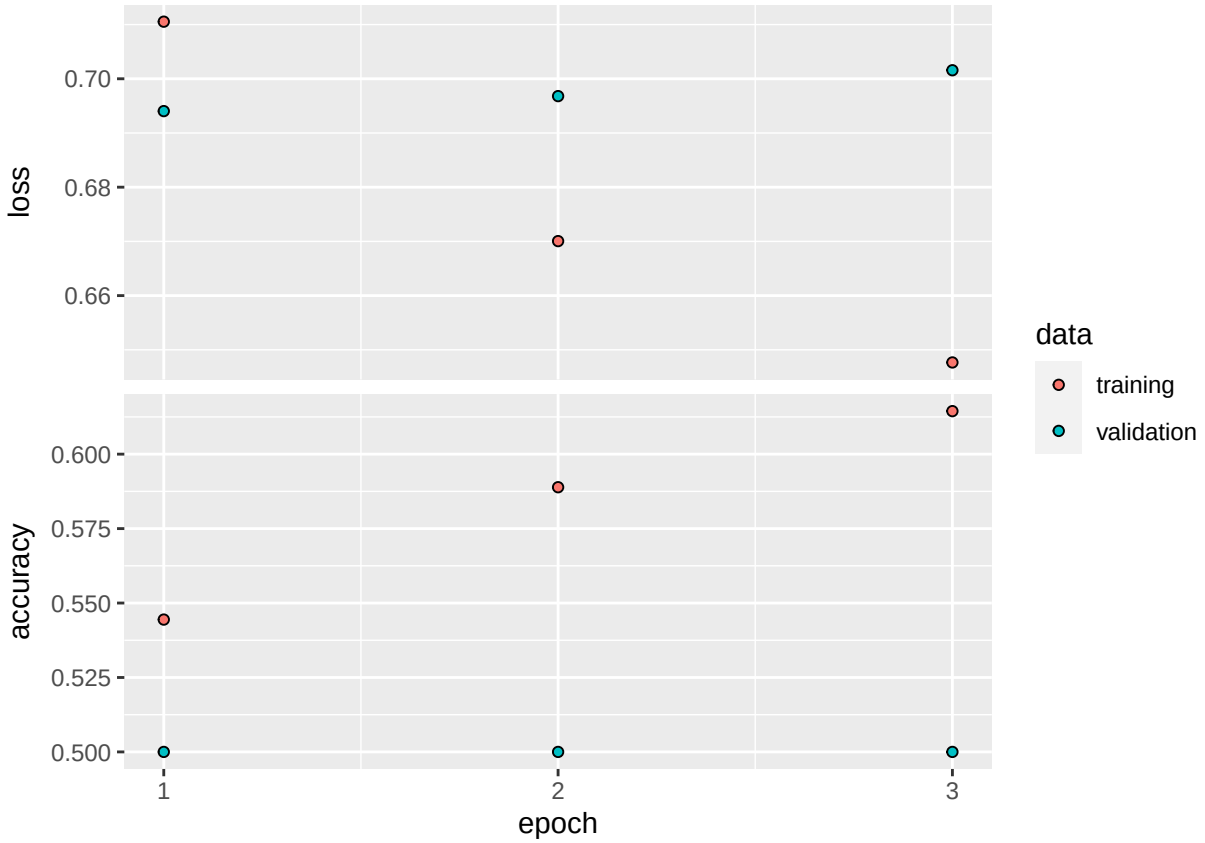
A: We're using this architecture because it is specialised for light-weight purposes, reducing training time which would otherwise be problematic on lower-end systems.

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

A: The MobileNet class has 4.3 million parameters, which is relatively few compared to for instance the ResNet50 class, which has 25.6 million parameters and achieves generally higher accuracy on image classification tasks. On the much higher end of the spectrum would be EfficientNetV2L, with 119 million parameters and substantially higher accuracy. One primary benefit of MobileNet is its acceptable performance given its relatively smaller count of parameters.

```
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: 6.76797787745794

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?

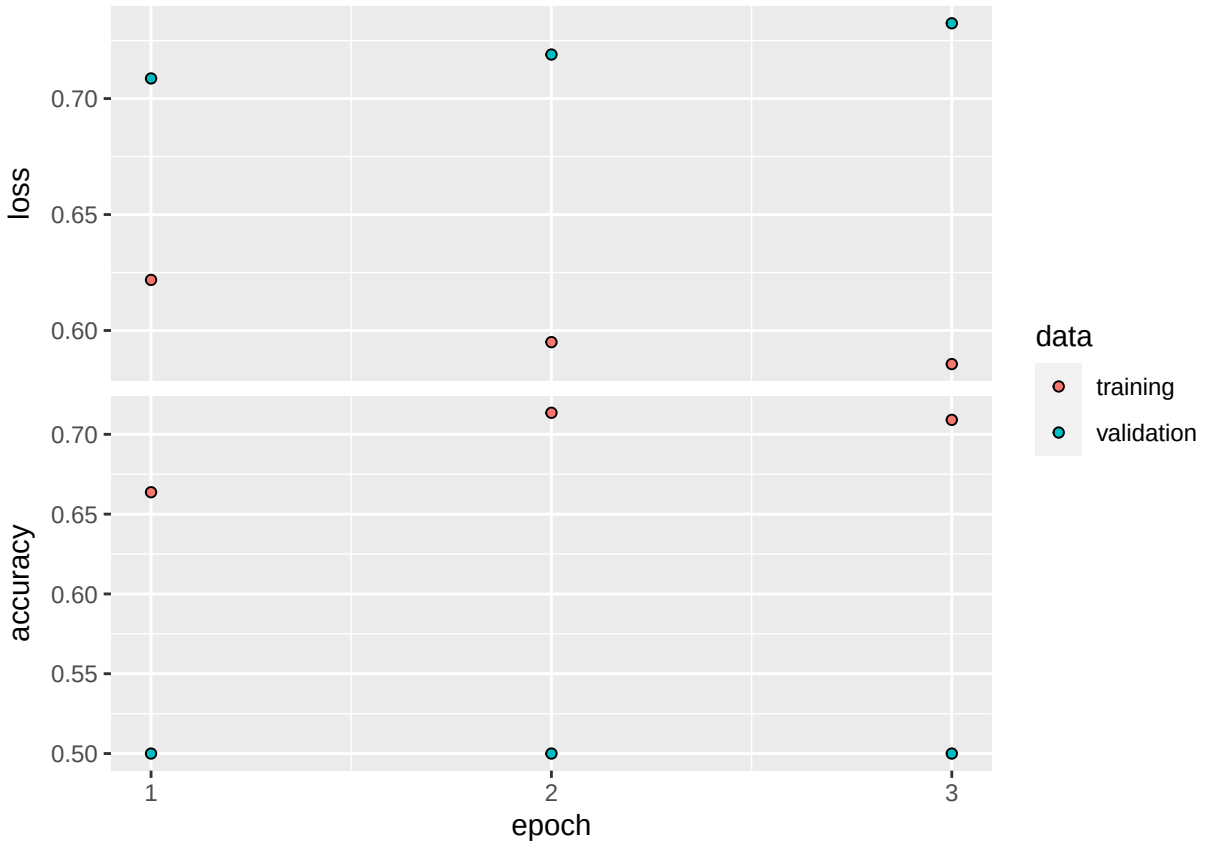
A: No, it would seem not to be succeeding at all, especially when looking at the accuracy, which barely straws from 50%, i.e. the expected value accuracy of random guessing. The fact that validation loss and accuracy does not change at all also shows that the changes that are being made over the 3 epochs have little

to no effect on its predictions outside of the training data.

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: 6.79604326089223

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).

A: This model performs *differently* to the other, and achieves more scores that are higher than not when compared to the previous model, but it's a struggle to say that it performs "better". Given that validation accuracy remains at 0.5 the whole time, and that the validation loss ends up at the same value of 0.7 which was constant in the previous model, we can conclude that it has the same performance as the previous model where non-training-set data is concerned. The higher accuracy and lower loss reported here in regard to training are almost certainly the result of overfitting.

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.

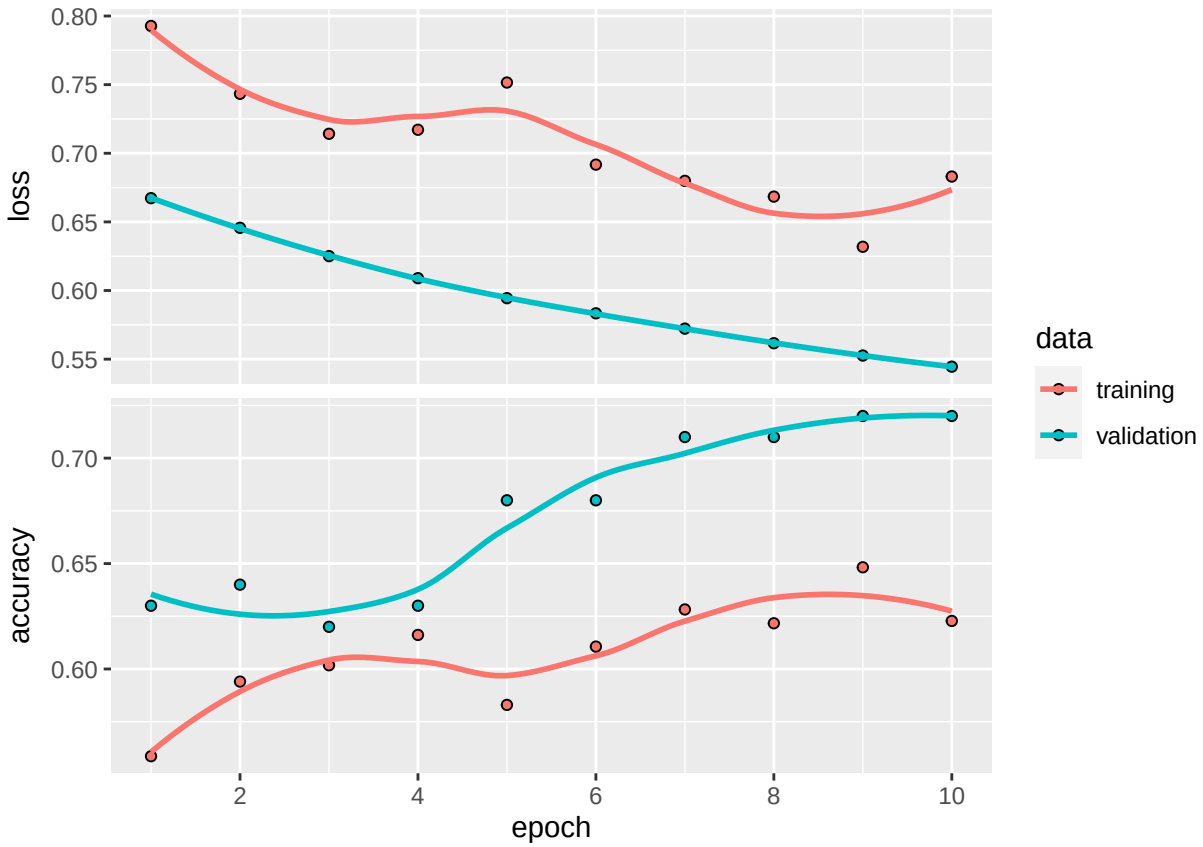
8 What is ImageNet aka "ImageNet Large Scale Visual Recognition Challenge 2012"? How many images and classes does it involve? Why might this help us?

A: ImageNet is a large set of images arranged to cover a number of individual topics very thoroughly. Each topic or class, has hundreds of thousands of images in the ImageNet database.

For a classification task, fitting to the ImageNet dataset will make a network become able to classify a wide range of subjects, possibly including ones that we are interested in for our own analysis. Additionally, aside from the actual classes that are the focus of ImageNet classification, a network trained on ImageNet will compose a number of features which may prove generally useful in image classification tasks, and these

general features may be what our current network is missing.

```
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)
```



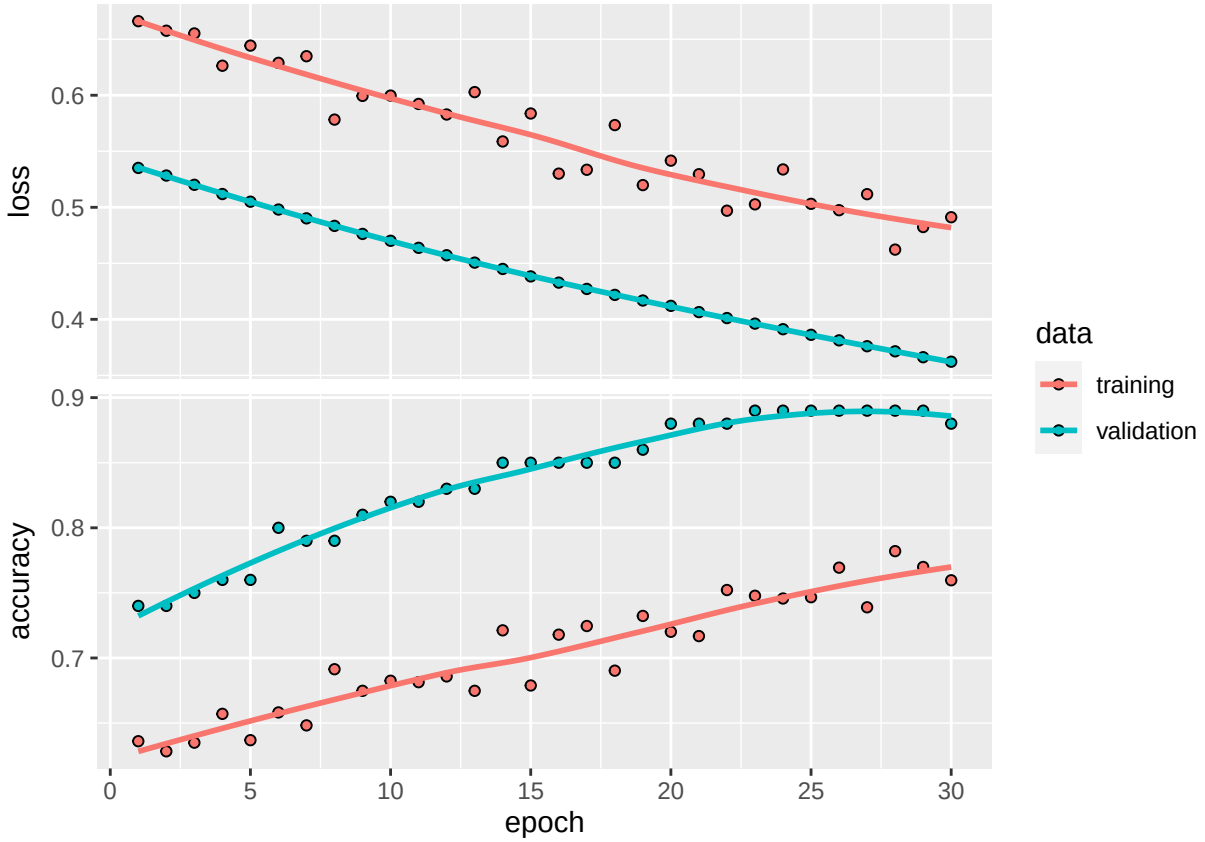
Time for this code chunk to run: 6.11436165173849

9 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!

A: For this question, mobilenet_v2 was used, see below. (I would try with more architectures in addition to this one, but this took a long time to run.)

```
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 %>% 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: 18.3833498080572

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  0.751694        0.64        0.63
```

Time for this code chunk to run: 8.0290539264679

10 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?

A: Overall, with a high specificity and a low sensitivity, although the model is much better at predicting pneumonia than before, it is not necessarily good at predicting pneumonia where it counts. With these performance metrics, despite accuracy which is high enough to be useful in positive cases, the model would not be clinically useful as a screening method. With low sensitivity it will be prone to false negatives, which means that there's a fair chance that a patient actually does have pneumonia despite the network judging otherwise. If it were to be used in a clinical context though, then the positives it detects are quite likely to be positive, which is useful information, but ultimately a clinician would still need to manually check over the negative X-rays, and the positive ones would have to be checked over manually just by the nature of needing to analyse the X-rays to determine treatment steps to take, even ignoring the need to double check the results.

On the whole, a tool with metrics like these (or even metrics better than these with the same general trend of high specificity low sensitivity) would not be clinically useful for detecting pneumonia, as it doesn't automate away anything which could be reasonably automated. On the other hand a tool with the inverse dynamic would be useful, as it could be used to be reasonably sure that pneumonia can be ruled out and thus doesn't need to be checked. With this type of screening tool, false positives are preferable to false negatives.